1 Introduction to *Live Coding: A User's Manual*

So what do we mean by calling this book "a user's manual"? This is no arbitrary choice of terminology. A manual is a handbook, traditionally small enough to hold in the hand (*manus* being the Latin word for "hand"), a support for action. The manual can be repeatedly referred to—it is a source of information that can be applied or performed. We hope that this book can serve as a source and inspiration for live coders and for wider users of code. When it comes to computing (not least), the term *user* seems at first to be rather limited and functionary, even when describing well-meaning notions such as user-centered or user-friendly, which are both part of a more general shift toward seemingly inclusive and participatory forms. However, as much as it is clear that the user is open to subtle forms of exploitation,[1] we argue that the user might also be a force for reinvention.

If you are expecting a conventional user's manual, then put this book down. As the title suggests, live coding is there to be used, not least to make visible the materiality of the computer and the varied experience of its usage. It is a user's manual of sorts, but not in any conventional technical sense of being a prescribed set of instructions or a how-to recipe book on how to do live coding. Indeed, there can be no conventional user's manual for live coding because there can be no universal way to understand or practice live coding.[2] There is no universal language to code in, to start with. It is this pluriversal capacity of live coding to resist or trouble any easy classification, categorization, or explanation that we take as our provocation for (the impossibility, or at least challenge of) writing this book.

Certainly, there is something perverse in writing a book about live coding. On the one hand, books are not particularly well suited for registering the dynamic processes of how writing and coding operate as distinctive cognitive practices, unfolding in time. Even static code is poorly expressed once printed on the pages of a book—although it is common enough for programming manuals and textbooks to do just this, encouraging

the reader to read and doggedly type out the program in order to execute it. On the other hand, the very constraints of the printed form allow for other critical reflections to take place, while the textual qualities of writing code can be understood in the broader context of linguistic form, both on the syntactic and semantic levels. Perhaps the disjunction allows certain kinds of thinking to emerge that register how writing and coding are particular technologies that generate particular readings and contingencies. Accordingly, this first chapter operates as a broad introduction to live coding alongside speculation on the inherent reflexivity of technological form, in which the process of writing a book on live coding registers its own particular temporality.

What Is Live Coding?

Live coding has been described in terms such as writing software in real time, changing a program while it is running, projecting the screen for the audience to participate in, writing as an improvisatory practice, composing live using textual notation, changing rules while following them, *conversational programming* (conversing with the computer in its own native language), thinking in public, and creating and using bespoke systems tailored for *on-the-fly* or *just-in-time* performance. However, as live coder David Ogborn states, "To define something is to stake a claim to its future, to make a claim about what it should be or become. This makes me hesitate to define live coding."[3] Live coding is a performance practice that operates as an adventure and exploration, deliberately rejecting fixed definitions, remaining heterogeneous in nature, continually challenging its self-understanding through the practice of writing and rewriting—defining and redefining—as a public performance. With no formal definition (or at least only one that includes the possibility of its own redefinition) also comes some resistance to hierarchical control—live coding cannot really become owned by established practitioners or institutions. In asking "What is live coding?," our intention is not to fix or define but rather to explore how live coding *opens up*. Live coding is about people interacting with the world, and each other, in real time, via code.[4] Live coding is about making software *live*.

To make software live is not a brand-new proposition,[5] as over the past two decades a worldwide live coding movement has emerged within a performance context that throws new light on code-as-interface through an explosion of new tools and practices. Live coding asks questions about *liveness*, inviting us to reflect on what it means to be *live*—to have bodies, to communicate, to act. When we write code live, we adapt it to our needs, and it adapts us in return. In this book we consider how the performance practice of live coding has been enabled by an increased capacity for liveness within

computer programming more broadly and in turn how the performance of live coding proposes new ways of operating, posing questions and challenges to some of the underpinning values and ideologies of a wider computational culture.

Live coding involves a critical orientation toward the otherwise conventionalized work of programming and software engineering.[6] For live coder Sarah Groff Hennigh-Palermo, "This way of computing . . . helps me 'unthink' the engineering I do as my day job. It allows for a relationship with computers where they are more like plants, rewarding cultivation and experimentation."[7] Live coding gives us a way to think otherwise about coding—what it *can be*, rather than what it *is*. We might consider this capacity of live coding as a technique of *making strange* (or defamiliarization), in the sense first intended by the Russian literary theorist Viktor Shklovsky, as presenting familiar things in a strange way in order to enhance apperception.[8] Live coding *makes software strange*,[9] allowing us to see beyond routine practices and interpretations of code. This book explores the wider context within which the performance practice of live coding emerges while inviting reflection on the potential of what it might become.

Toward the end of the twentieth century, computers were becoming omnipresent across many areas of arts practice, and communities of musicians and artists were beginning to form around specific programming languages (notable are audio programming languages such as Max/MSP, SuperCollider, Pure Data, and CSound). Artists were beginning to use code as a creative material to express music, visuals, choreography, robotics, video, and computer games. With code as the material and the laptop as a studio, the traditional divisions between art, music, architecture, design, and game development blurred, and educational institutions began to respond with new interdisciplinary approaches to learning how to program computers. By the year 2000, prototypical live coders began to arrive on the scene with a distinct approach to live programming, straddling performance and creative computing. These coders asked: Why not simply use the technology to *show* what we are doing? Why not *show the screen* and be more transparent about how we are conversing with our machines in our own languages?

Live coding involves showing the screen or making visible the coding process as part of a live performance. Broadly speaking, it describes the improvisatory real-time composition of predominantly computer-generated audiovisual material, in which the writing of code itself (or other executable instructions) is presented as a live event for an audience. Alongside witnessing the coder engaged in the live act of coding (laboring at their laptop), the code itself is also presented—typically projected—in real time as it is being worked on as a visible part of the performance. In one sense the imperative to show the screen might be considered a response to the audience's frustration or need to *see* something during coding performances, or even a way of foregrounding the

technical credibility (even virtuosity) of creative coders. However, for many live coders the practice of showing the screen is a critical gesture in its own right.

Despite reluctance to be constrained by definitions, one popularly cited formulation is that live coding is best characterized as *thinking in public*, a phrase derived partly from the etymology of programming as *writing in public* (Greek *programma*, from *prographein*, "write publicly"). For some live coders, making these private intellectual and creative activities publicly visible has a liberating effect, while at the same time introducing an imperative of clarity within coding and turning each algorithm into artwork accessible through simplicity of purpose and expression. Live coding unveils the underlying operational layer of activity beneath the more familiar, readable gestures of computational performance. Commitment to a mode of thinking that is open and creative, especially when made public, can thus acquire the status of a political act.

When interpreted in the context of cultural experience, live coding is not only fundamentally an "open work"[10]—inasmuch as constituents of it are left open to the laws of improvisation, chance, or intervention by the public—promising an aesthetic experience that matches the networked online community aspirations of the internet era (specifically free software principles) but, significantly, a broader DIY, punk, and post-punk ethos. The ambitions of such work are far-reaching for technologies that support new styles of conversational programming—that are nonlinear and routinely interactive, with software infrastructures that are not static monuments to the achievements of the past but dynamic processes of engagement with the future. The temporality that is implicit in technologies of control becomes subverted by software in the *present tense*. The distinction between human action in the moment and systems-based planning has been a constant tension for the theorists of human-computer interaction. As we explore in the later chapters of this book, the intersection of these modes of thinking with the socioeconomic and corporate infrastructure of the software industry is unavoidably political.

The capacity of live coding for making visible counters the *smart* paradigm in which coding and everyday life are drawn together in ways that become imperceptible. The invisibility here operates like ideology, where lived experience appears increasingly programmed, and we hardly notice how ideology is working on us.[11] If we follow this logic, then we do not *use* computers; they use *us*. Along with the disappearance of the computer, as one of the tenets of contemporary interface design established through technical programs of pervasive and ubiquitous computing,[12] artist Olia Lialina identifies that the *user* is disappearing as well—both the phenomenon and the term.[13] This is a problem for her (and us) inasmuch as *Big Tech* wants computers to be invisible so our experience of using them becomes seemingly natural.[14]

Against received wisdom that would tend to be skeptical of the term *user*,[15] Lialina reclaims the possibility of what she calls the *Turing Complete User*, referring to users who have more autonomy over computer use regardless of the primary purpose of an application or device. As Lialina states: "Being a User is the last reminder that there is, whether visible or not, a computer, a programmed system you use."[16] It is in this sense, too, that we invoke the user in the title of this book.

Critical-Creative Contexts

Live coding emerges through a critical relation to the wider programming and software-engineering context. However, it has also evolved in dialogue with the various creative practices that find new expression in and through live coding, including choreography, the visual arts, poetry, and especially music. In one sense, in the context of live coding, music often becomes a metonym for other performance practices. However, this is not meant to reduce or homogenize all other art forms, and within this book we explore how ideas, practices, and theories emerging from wider interdisciplinary contexts open up new ways of conceiving live coding's distinctiveness. Still, the musical analogy warrants attention, for it invites particular consideration of the specific things that live coding *does*.

For musicians in the classical notated tradition, every note they play is predetermined by the composed score, to which live performance adds expressive adjustments. In contrast, performers of jazz, folk, and rock music might follow a chord sheet but often have the freedom to choose notes within those chords. Free jazz improvisers have even more freedom. But a live coding musician is like an improvising composer, able to transform the whole structure of the piece with a few keystrokes. The code is the score, and the computer performs the music (in a sense) as the audience watches it being written. The live coder breaks down and works outside the established dichotomies of score/performance, composer/performer, and composition/improvisation.

The technologies of music-making are diverse in their acoustic, electronic, and digital forms. Established genres of recorded music control many layers of adjustment, from basic acoustic vibrations to electronic filtering, sampling, mixing, and remixing using tape loops, turntables, or disk drives. As digital alternatives have become available at all of these different layers of sound production and manipulation, all are accessible via code, turning the laptop into a kind of universal *instrument* whose own capabilities and boundaries can in turn be redefined.

The capacity of live coding to have an impact on the production of music is clearly evident. However, we argue that all performance practices (including music) offer a special

way of understanding software and that this has radical potential for all software—not only for artists, their audiences, and art theorists but also for engineers, philosophers, and activists. Scripting a performance is like programming a computer because both involve ordering events in time. Music and dance are art forms concerned with movement over time, from fluctuating waveforms or movements making up individual actions to sequences of actions and the rhythmic patterns in which they play out. Everybody knows that performances can be captured digitally, but it is not so widely understood (or even accepted by computer scientists) that computers and algorithms are fundamentally concerned with time or that using computers and making software is a kind of performance in itself. In these terms, live coding helps us to gain new insights about what software can be.

Programming languages, like languages in general, offer ways to describe things compositionally, through a structured system of communication. Programming languages are often used to describe algorithms, but we can also describe algorithms with human languages. Language comes alive through the body, through communication with others, and through writing systems. In this book we argue that live coding can transform human experiences of technology through these same connections.

Open Potential and Interdisciplinary Perspectives

As stated at the outset, this book is not a conventional user's manual. The title borrows from the combinatory literature of Georges Perec and his 1978 book *Life: A User's Manual*, a collection of interwoven stories based on the lives of the inhabitants of a fictitious Parisian apartment block.[17] More than a passing reference to Perec, the experimental writers group OuLiPo (Ouvroir de Littérature Potentielle, of which Perec was part) draws attention to the careful, even programmatic, use of language and in turn has helped us better understand our own endeavor of writing this book on live coding. For example, the French term *ouvroir* refers to a place where people work together on a difficult task, deriving new techniques; more precisely, the example is given of a sewing circle, which we believe is a good metaphor for the practice of live coding. The *littérature potentielle* of OuLiPo speaks of a search for new forms of writing, as Perec states: "We call potential literature the search for new forms and structures which may be used by writers in any way they see fit."[18]

We see live coding (and our attempt to write *with* it as much as *about* it) operating in this spirit, as a search for new arrangements of form and structure while staying mindful of the formal rules and constraints associated with computing in dialogue with creative practices such as music, choreography, and the visual arts. By appropriating

Perec's title, we indicate our ambition to open up the forms and structures of live coding to close description, analysis, and experimentation, revealing it to be in a perpetual state of potential transformation into something else. For us, live coding has something of this sense of reinvention, an ability even to unsettle the relationship between code and life—and that is what this book is about.

The composition of the authorship of this book also reflects OuLiPo's interdisciplinary perspective (which in its case included mathematics, computer science, and literature) by bringing together different disciplines and orientations within a shared project. Our own writer's group embraces diverse practices and theories, as well as conventions and histories, drawn from music, interdisciplinary design and software development, software studies, and network culture alongside areas of artistic research, contemporary art, and performance. Live coding operates at the threshold of these different practices and disciplinary perspectives, and we wanted to reflect that in the style of the book itself. We write from *within* the field of live coding, as practitioner-scholars writing from the insider's perspective, from the viewpoint of live coding performers and software innovators, as well as through wider interdisciplinary scholarship that considers how live coding relates to a wider contextual field, including situating live coding within philosophical, political, and performance-based practices. Our interdisciplinary exchanges have not only happened on the page and in the process of writing the book. Rather, over several years (and throughout the duration of the book's development) we have talked, discussed, and explored live coding in dialogue with various research projects, networks, and conferences, as well as through live encounters within numerous festivals and algoraves.[19]

Each and every chapter has been coauthored in various ways. Certainly, this has been a complex undertaking. At times, different contributing voices become tangible through a perceived shift in style or semantics or through the meeting of different—even seemingly incompatible—references or ideas. Over many years we have gradually gathered fragments of thoughts and ideas. In places, the transition from one voice to another might appear smooth and seamless; elsewhere, the reader might notice the sudden break or change in tack. Through this approach to coauthoring the book, we draw attention to the etymological relation of *text* to *textile*: from the Latin *textus*, literally "a thing woven," or from *texere*, meaning to weave or interweave, to braid or fit together.[20] In doing so we reflect the collaborative principles of live coding itself and the desire and potential therein for bringing different systems into dialogue. In writing together we have also become attuned to the potential of writing the book itself as an improvised, collaborative performance.

Live coding necessarily draws attention to the writing machine—including laptops, source code editors, and so on—and the material production of text or code, book, or

even computer hardware.[21] It is also connected to the body of the author and reader (producer, worker), and it becomes clear that writers, readers, texts, coders, machines, and codes all have bodies and that these become operative under particular sociotechnical conditions. Informed by the practice of live coding—and our skepticism about some of the conventions of academic writing and its singular authoritative voice—in the process of writing this book we began to explore some aspects of liveness, experimentality, and reflexivity within our collaboration.[22] To focus attention on the performative dimension of live writing as analogous to live coding calls into question some of the static assumptions of communicative forms and instead foregrounds material conditions and their effects, as well as the wider sociotechnical infrastructures through which they are served. It is also worth remembering that all writing *is* technology, and live writing, like live coding, is able to activate slippages of meaning and ultimately demonstrate how writing subjects and objects become thoroughly entangled.[23]

In writing this book, we also reflected on the jumbled experience of time that the writing of books conduces.[24] By writing about writing as we write, we identify a parallel to the way that live coders edit code as it runs. In this way, examining live coding and live writing together usefully confuses any strict definitions between them, as well as between the act of writing and its execution, allowing us to rethink some of our preexisting notions of notation, liveness, temporality, and knowledge production, which in turn become the focus for the various chapters (4–7, respectively) of this book. While we have only gestured toward this experimental and reflexive attitude toward writing about live coding, it does suggest another book yet to be written (and one at times we thought we were writing here). For now, the book that you are reading takes a more conventional form.

Navigating the Book

The structure of the book is broadly bipartite. The first part is more practice focused, offering an account of the origins, development, and aspirations associated with the evolution of live coding alongside presenting documentation and examples of live coding practice. The second part is more speculative and conceptual in its register, allowing space for discussion of the many ways live coding reflects and informs wider cultural and political concerns. In this sense we identify live coding as a *critical technical* and *aesthetic practice*, able to activate sensemaking across interdisciplinary fields.[25]

Chapter 2, "Partial Histories," informed by research interviews and primary sources, focuses on the history of live coding, taking our point of departure from the TOPLAP manifesto of 2004.[26] Far from the intention of establishing a canon here, we take live

coding to be a way of addressing some of the assumptions of historicism more broadly and of the progressive technological development that we develop in later chapters. It highlights some of the wider conditions from which live coding has evolved, including the recent history of computer-based music (since the 1980s) and the interest in digital culture within art schools around the mid-1990s, alongside an increased wider cultural interest in creative coding (since the turn of the millennium). We acknowledge various technological catalysts and precursors to live coding, the influence and significance of specific individuals and collaborations, and the importance of receptive and proactive venues and festivals in creating a context for the emerging practice of live coding. We chart the arc of this burgeoning practice from its inception and instituting moments to its becoming established within various institutional frames while arguing how the insurgent and irreverent imperative of live coding has continued to thrive through the new genre of algorave.

The third chapter, "Expositions," presents live coding in its singularity, exposing the specific approaches through which live coding manifests within a diversity of different practices. Forming a parallel history of the field, it includes diverse voices from live coders themselves as reflective performers and innovators. Occupying a central position within the book, the focus on distinctive practices spans the different waves of live coding, from the more tentative explorations preceding the formation of TOPLAP in 2004 to the focused campaigns of popularizers and the cultural transformation of live coding as it has spread around the world.[27] It includes both the creators of new live coding tools and those who have developed artistic practices through using those tools. We do not attempt to synthesize a single narrative for this chapter but provide an opportunity for the diversity of the field of live coding to be represented through individual voices, through rich accounts of practice itself. We refer to these presentations of practice as *expositions*, a term from the field of artistic research for describing how the epistemological contribution of a given practice is exposed, often through a combination of the practice, its documentation, and writing.[28] The epistemological dimension of live coding and the question of "what live coding knows" is further explored in chapter 7.

Following the practice-based expositions is a shift in orientation toward the wider issues that live coding raises in relation to notation, liveness, time, and knowledge, as well as its future. Chapter 4, "Notation," focuses specifically on the notational nature of the live coding performance. The generative relationship between a notation and what is notated is what drives generative creativity in the live coding field. The perceptual gap between the notational map and the generated territory allows the live coder to reach beyond their imagination and work with notation not to efficiently realize a ready-made idea but to *follow* an idea to see where it takes them. 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 music itself as notated by the live coder. This chapter unpacks these ideas further, exploring how the momentary nature of a live coder's notation might be closer to speech than to text. Or rather, live coding practice finds itself caught between two worlds, as it is too ephemeral to be score-based culture and yet too centered on text to be oral culture. The focus is therefore not only on the notation but what is notated and the activity of notation—the nature of the deterministic algorithms that live coders work with and the dynamic ways in which they are crafted. By examining how live coding challenges tensions and dichotomies such as those between determinism and liveness, between musical improvisation and composition, and between oral and written culture, we find new approaches to notation as a dynamic, live medium.

This quality of liveness is explored further in the chapter 5, "Live Coding's Liveness(es)." It explores the implications of live coding for our understanding of liveness by asking: What kinds of liveness are produced within live coding, and with what effects? What does this indicate about the relations between technology, performance, and even *life* that live coding suggests? The chapter draws on different theoretical ideas of liveness and performativity to demonstrate that live coding does not sit easily within any singular theoretical framework: its liveness must be apprehended from more than one epistemological and ontological perspective in which the *hierarchy of liveness* conceived in computational terms collides with a wider discourse on embodiment, vitality, and performativity. The chapter explores how complex conceptualizations of liveness are negotiated through the human-machine relations central to live coding, in turn raising wider contemporary concerns about what it means to be alive, operative, and actively present in the world. Live coding presents a direct challenge to the conventional understanding of liveness and to the lived experience of time, both with respect to the role of the programmer and user and, crucially, through the way the machine understands time, whether in a wiki edit, a keystroke log, a page render, or a print queue.

Chapter 6, "Time Criticality in Live Coding," explores how live coding activates the present in all its complexity. The time criticality we speak of points to this inherent processuality of the computer and programming to perform and express dynamic operations in the present: it is both critical on a technical level and on a conceptual level and moves toward *criticality* in the sense that we emphasize practices that actualize potentialities in real time rather than simply expose problems. Across the chapter as a whole, live coding offers a complex multitemporal human and *more-than-human* experience of time—human and machine times become entangled, both in and out of

what we conventionally understand as time. In exemplifying the coming together of multiple temporalities and different types of time, the practice of live coding contributes toward new understanding about our contemporary temporal experience. Moreover, how does live coding *itself* temporalize and actively produce—activate, intervene in, invent, and reorganize—as much as *reflect* different experiential temporalities? Thus, we not only acknowledge that time plays a crucial role in live coding in terms of the unfolding of time in its performance—as in a timeline or score—but also demonstrate how time can be manipulated, and indeed produced, through such means.

In asking "What Does Live Coding Know?," chapter 7 explores the engineering contexts in which the necessary technical knowledge has been generated and applied and the contexts of creativity where *know-how* from several perspectives crosses the boundaries of computer science, craft practices, and artistic research, as well as the sense of indeterminacy that is inherent to the medium. Our intention is to explore *beyond* the knowledge needed to practice live coding, extending to the knowledge that is acquired or that emerges, or is even assumed, in and through that practice. Live coding comprises technical, artistic, and philosophical inquiry, for it not only draws on and from specific fields of knowledge but, as a practice, asks specific epistemological questions. Here, we can draw on the interdisciplinary work of Philip Agre, whose concept of a *critical technical practice* pertains to the philosophical enterprise of artificial intelligence as pursued through technical work,[29] where critical and cultural theories intervene in engineering practices as a way of inviting reflection on the underlying assumptions and ideologies within technological design. Live coding presents a challenge to our categories of knowledge, in both its scholarly and technical-professional realizations. Of specific interest for us is the way in which it operates at the threshold of different species of knowledge, troubling its classification. In posing the question "What does live coding know?," the intention is to expand beyond an anthropocentric and culturally specific understanding. Here, what live coding knows is not synonymous with what the live coder knows but rather refers to the epistemological potential of the practice itself, including the multiple knowledges arising in and through the collaboration with machines. Our suggestion is that the live *intra-actions* of programmer, program code, and the practice of coding defamiliarize knowledge production and expose how making and thinking might escape familiar forms and normative applications of inscription practices.[30]

The final chapter, "What Does Live Coding Want?," shifts the focus from what live coding *knows* to the question of what it *wants*—inviting dual reflection on both the agency of code and the future of live coding. This chapter initially draws out some of the political implications relating to the operations of algorithms. Algorithms, which increasingly manipulate and control our social systems, are not as fixed or as

deterministic as they might first appear but rather are emergent and reactive phenomena, subject to constant *live* updates. They are also part of larger sociotechnical assemblages and infrastructures that are constantly evolving and subject to variable conditions and contingencies that include the lively contributions of coding or writing. Perhaps what is at stake here is a deeper exploration into the ways that space and time are constructed when coding and writing and maintaining the best practices of the arts and the academy in the face of corporate data infrastructures. Live coding is an open-ended performative process that attempts to reject deterministic or previously held certainties over the ways that meanings are generated and disseminated. This provides a counterspace for agency in human and nonhuman entities that is produced through a fuller understanding of the operations of the material-discursive systems that we are written and coded into. In the case of writing this book, we acknowledge that to some extent it also *writes us*.[31]

Live coding conceives of technology as fluid phenomena open to transformation and exploration, including through the live redesign of instruments or the live rewriting of scores. Live coding rejects easy definition, continually challenging its self-understanding through the practice of defining and redefining itself as a public performance—in the way a Wikipedia entry invites rewriting. Such a mode of writing—done with "shaky hands"[32]—expresses the ability to move, change, and explore the unknown. Writing has no beginning or end in this way as it opens itself up to its relations with existing works and its future extensions. The issues that we engage in this book in reference to live coding continue to be developed through ongoing work, inasmuch as all books are provisional and open to multiple interpretations as well as further modification.[33] Thus, any sense of a beginning or end to this introductory chapter (and the book more broadly) is only a temporary holding position in lieu of further thoughts and future developments, which ultimately serves to demonstrate how readers might become writers.