NICK MONTFORT, PATSY BAUDOIN,
JOHN BELL, IAN BOGOST, JEREMY DOUGLASS,
MARK C. MARINO, MICHAEL MATEAS,
CASEY REAS, MARK SAMPLE, NOAH VAWTER

10 INTRODUCTION

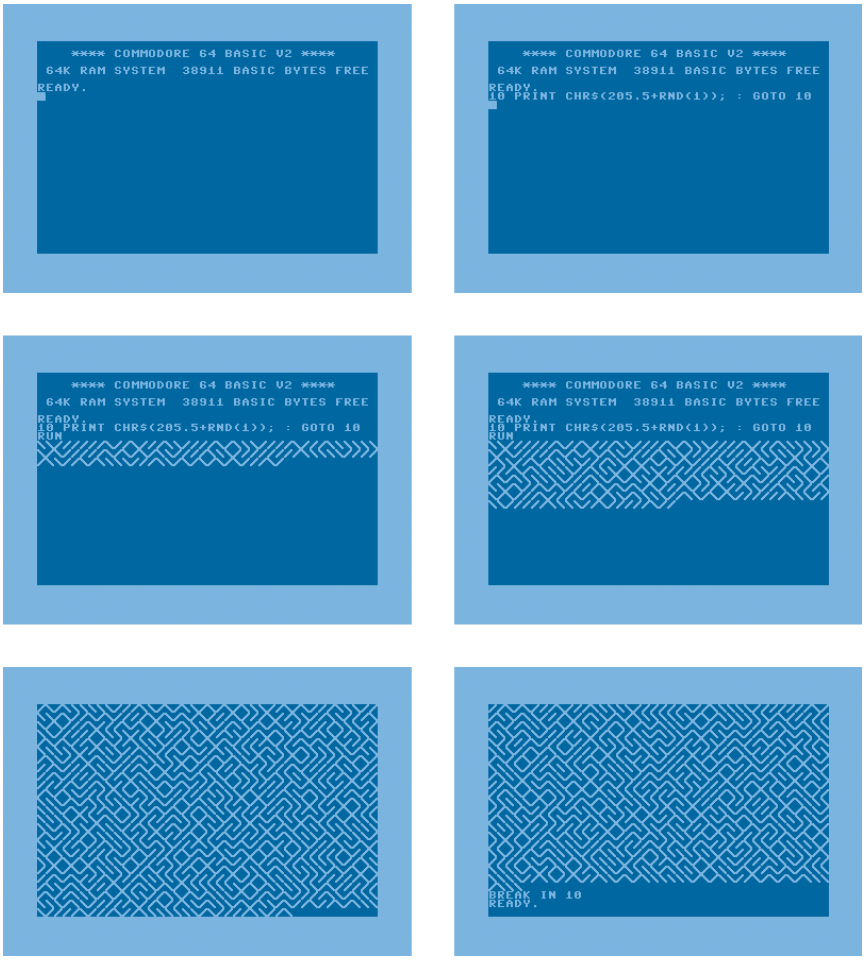

Figure 10.1

From left to right and top to bottom, the `10 PRINT` program is typed into the Commodore 64 and is run. Output scrolls across the screen until it is stopped.

```
(2)     10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

Computer programs process and display critical data, facilitate communication, monitor and report on sensor networks, and shoot down incoming missiles. But computer code is not merely functional. Code is a peculiar kind of text, written, maintained, and modified by programmers to make a machine operate. It is a text nonetheless, with many of the properties of more familiar documents. Code is not purely abstract and mathematical; it has significant social, political, and aesthetic dimensions. The way in which code connects to culture, affecting it and being influenced by it, can be traced by examining the specifics of programs by reading the code itself attentively.

Like a diary from the forgotten past, computer code is embedded with stories of a program's making, its purpose, its assumptions, and more. Every symbol within a program can help to illuminate these stories and open historical and critical lines of inquiry. Traditional wisdom might lead one to believe that learning to read code is a tedious, mathematical chore. Yet in the emerging methodologies of critical code studies, software studies, and platform studies, computer code is approached as a cultural text reflecting the history and social context of its creation. "Code . . . has been inscribed, programmed, written. It is conditioned and concretely historical," new media theorist Rita Raley notes (2006). The source code of contemporary software is a point of entry in these fields into much larger discussions about technology and culture. It is quite possible, however, that the code with the most potential to incite critical interest from programmers, students, and scholars is that from earlier eras.

This book returns to a moment, the early 1980s, by focusing on a single line of code, a BASIC program that reads simply:

```
10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

One line of code, set to repeat endlessly, which will run until interrupted (figure 10.1).

Programs that function exactly like this one were printed in a variety of sources in the early days of home computing, initially in the 1982 *Commodore 64 User's Guide,* and later online, on the Web. (The published versions of the program are documented at the end of this book, in "Variants of 10 PRINT.") This well-known one-liner from the 1980s was recalled by one of the book's authors decades later, as discussed in "A Personal

Memory of 10 PRINT" in the BASIC chapter. This program is not presented here as valuable because of its extreme popularity or influence. Rather, it serves as an example of an important but neglected type of programming practice and a gateway into a deeper understanding of how computing works in society and what the writing, reading, and execution of computer code mean.

ONE LINE

This book is unusual in its focus on a single line of code, an extremely concise BASIC program that is simply called `10 PRINT` throughout. Studies of individual, unique works abound in the humanities. Roland Barthes's *S/Z,* Samuel Beckett's *Proust,* Rudolf Arnheim's *Genesis of a Painting: Picasso's Guernica,* Stuart Hall et al.'s *Doing Cultural Studies: The Story of the Sony Walkman,* and Michel Foucault's *Ceci n'est pas une pipe* all exemplify the sort of close readings that deepen our understanding of cultural production, cultural phenomena, and the Western cultural tradition. While such literary texts, paintings, and consumer electronics may seem significantly more complex than a one-line BASIC program, undertaking a close study of `10 PRINT` as a cultural artifact can be as fruitful as close readings of other telling cultural artifacts have been.

In many ways, this extremely intense consideration of a single line of code stands opposed to current trends in the digital humanities, which have been dominated by what has been variously called distant reading (Moretti 2007), cultural analytics (Manovich 2009), or culturomics (Michel et al. 2010). These endeavors consider massive amounts of text, images, or data—say, millions of books published in English since 1800 or a million Manga pages—and identify patterns and trends that would otherwise remain hidden. This book takes the opposite approach, operating as if under a centrifugal force, spiraling outward from a single line of text to explore seemingly disparate aspects of culture. Hence its approach is more along the lines of Brian Rotman's *Signifying Nothing* (1987), which documents the cultural importance of the symbol 0. Similarly, it turns out that in the few characters of `10 PRINT`, there is a great deal to discover regarding its texts, contexts, and cultural importance.

By analyzing this short program from multiple viewpoints, the book

explains how to read code deeply and shows what benefits can come from such readings. And yet, this work seeks to avoid fetishizing code, an error that Wendy Chun warns about (2011, 51–54), by deeply considering context and the larger systems at play. Instead of discussing software merely as an abstract formulation, this book takes a variorum approach, focusing on a specific program that exists in different printed variants and executes on a particular platform. Focusing on a particular single-line program foregrounds aspects of computer programs that humanistic inquiry has overlooked. Specifically, this one-line program highlights that computer programs typically exist in different versions that serve as seeds for learning, modification, and extension. Consideration of `10 PRINT` offers new ways of thinking about how professional programmers, hobbyists, and humanists write and read code.

The book also considers how the program engages with the cultural imagination of the maze, provides a history of regular repetition and randomness in computing, tells the story of the BASIC programming language, and reflects on the specific design of the Commodore 64. The eponymous program is treated as a distinct cultural artifact, but it also serves as a grain of sand from which entire worlds become visible; as a Rosetta Stone that yields important access to the phenomenon of creative computing and the way computer programs exist in culture.

CORE CONTRIBUTIONS

The subject of this book—a one-line program for a thirty-year-old microcomputer—may strike some as unusual and esoteric at best, indulgent and perverse at worst. But this treatment of `10 PRINT` was undertaken to offer lessons for the study of digital media more broadly. If they prove persuasive, these arguments will have implications for the interpretation of software of all kinds.

First, to understand code in a critical, humanistic way, the practice of scholarship should include programming: modifications, variations, elaborations, and ports of the original program, for instance. The programs written for this book sketch the range of possibilities for maze generators within Commodore 64 BASIC and across platforms. By writing them, the `10 PRINT` program is illuminated, but so, too, are some of the main plat-

CRITICAL CODE STUDIES, SOFTWARE STUDIES, PLATFORM STUDIES

Critical Code Studies (CCS) is the application of critical theory and hermeneutics to the interpretation of computer source code, as defined by one of this book's authors (Marino 2006). During an online, collaborative conference, another of this book's authors challenged the 2010 Critical Code Studies Working Group to apply these methodologies to the one-line program that is this book's focus (Montfort 2010). Until then, a number of exemplary readings had taken up software and other encoded objects possessing considerably more code, clear social implications (for example, a knowledge base about terrorists), and more free space for writing of human significance in the form of comments or variable names. Members of the working group had demonstrated they could interpret a large program, a substantial body of code, but could they usefully interpret a very spare program such as this one? What followed, with some false starts, was a great deal of productive discussion, an article in *Emerging Language Practices* (Marino 2010), and eventually this book, with those who replied in the Critical Code Studies Working Group thread being invited to work together as coauthors.

CCS is a set of methodologies for the exegesis of code. Working together with platform studies, software studies, and media archaeology and forensics, critical code studies uses the source code as a means of entering into discussion about the technological object in its fullest context. CCS considers authorship, design process,

forms of home computing, as well as the many distinctions between Commodore 64 BASIC and contemporary programming environments.

Second, there is a fundamental relationship between the formal workings of code and the cultural implications and reception of that code. The program considered in this book is an aesthetic object that invites its authors to learn about computation and to play with possibilities: the importance of considering specific code in many situations. For instance, in order to fully understand the way that redlining (financial discrimination against residents of certain areas) functions, it might be necessary to consider the specific code of a bank's system to approve mortgages, not simply the appearance of neighborhoods or the mortgage readiness of particular populations.

This book explores the essentials of how a computer interprets code

 `10 PRINT CHR$(205.5+RND(1)); : GOTO 10`

function, funding, circulation of the code, programming languages and paradigms, and coding conventions. It involves reading code closely and with sustained and rigorous attention, but is not limited to the sort of close reading that is detached from historical, biographical, and social conditions. CCS invites code-based interpretation that invokes and elucidates contexts.

This book also employs other approaches to the interpretation of technical objects and culture, notably software studies and platform studies. While software studies can include the consideration and reading of code, it generally emphasizes the investigation of processes, focusing on function, form, and cultural context at a higher level of abstraction than any particular code. Platform studies conversely focuses on the lower computational levels, the platforms (hardware system, operating system, virtual machines) on which code runs. Taking the design of platforms into account helps to elucidate how concepts of computing are embodied in particular platforms, and how this specificity influences creative production across all code and software for a particular system. This book examines one line of code as a means of discussing issues of software and platform.

In addition to being approaches, software studies and platform studies also refer to two book series from MIT Press. This book is part of the Software Studies series.

and how particular platforms relate to the code written on them. It is not a general introduction to programming, but instead focuses on the connection of code to material, historical, and cultural factors in light of the particular way this code causes its computer to operate.

Third, code is ultimately understandable. Programs cause a computer to operate in a particular way, and there is some reason for this operation that is grounded in the design and material reality of the computer, the programming language, and the particular program. This reason can be found. The way code works is not a divine mystery or an imponderable. Code is not like losing your keys and never knowing if they're under the couch or have been swept out to sea through a storm sewer. The working of code is knowable. It definitely *can* be understood with adequate time

and effort. Any line of code from any program can be as thoroughly explicated as the eponymous line of this book.

Finally, code is a cultural resource, not trivial and only instrumental, but bound up in social change, aesthetic projects, and the relationship of people to computers. Instead of being dismissed as cryptic and irrelevant to human concerns such as art and user experience, code should be valued as text with machine and human meanings, something produced and operating within culture.

```
10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

The pattern produced by this program is represented on the endpapers of this book. When the program runs, the characters appear one at a time, left to right and then top to bottom, and the image scrolls up by two lines each time the screen is filled. It takes about fifteen seconds for the maze to fill the screen when the program is first run; it takes a bit more than a second for each two-line jump to happen as the maze scrolls upward.

Before going through different perspectives on this program, it is useful to consider not only the output but also the specifics of the code—what exactly it is, a single token at a time. This will be a way to begin to look at how much lies behind this one short line.

10

The only line number is this program is 10, which is the most conventional starting line number in BASIC. Most of the programs in the *Commodore 64 User's Guide* start with line 10, a choice that was typical in other books and magazines, not only ones for this system. Numbering lines in increments of 10, rather than simply as 1, 2, 3, . . . , allows for additional lines to be inserted more easily if the need arises during program development: the lines after the insertion point will not have to be renumbered, and references to them (in `GOTO` and `GOSUB` commands) will not have to be changed.

The standard version of BASIC for the Commodore 64, BASIC version 2 by Microsoft, invited this sort of line numbering practice. Some extensions to this BASIC later provided a `RENUMBER` or `RENUM` command that would automatically redo the line numbering as 10, 20, 30, and so on.