Introduction

“I dare to imagine the general public learning how to write code. I do not mean that knowledge of programming should be elevated to the ranks of the other subjects that form basic literacy: languages, literature, history, psychology, sociology, economics, the basics of science and mathematics. I mean it the other way around. What I hope is that those with knowledge of humanities will break into the closed society where code gets written: invade it.”

— Ellen Ullman, Life in Code
This collaboratively authored work, much like the object that has inspired it, is nonlinear and modular. It has been compiled together from several smaller component parts. We invite you to read this book accordingly. We have provided a series of experimental readings — just a few of what we believe to be the numerous explorations of the creative possibilities found within the confines of a rigidly constructed formal language that was once used to facilitate the breaking of existing spatial boundaries. We intend each section to expose new horizons of interpretation and exploration for understanding the poetics of code.

Throughout this book, we seek to show that software, or more specifically computer code, in excess of its bare functionality or its use value as an instrument to achieve some planned and programmed goal, also has numerous aesthetic properties and creative features. The aesthetic features of computer code — often characterized by a rigidly formal, restricted syntax, and numerous paralinguistic dimensions — sometimes have a supplemental character; they appear, at times, almost ornamental in their sheer excess beyond the functional elements and programmed goals. At other times, these features are an intrinsic and necessary part of the code. We believe that these special properties of computer code make possible imaginative uses or misuses by its human programmers and that these properties and features justify our exuberant readings, misreadings, translations, and appropriations.

At its base, this book is a poetic and philosophical meditation on the idea of computer code and the affordances and limitations of a language that is machine-oriented yet human-authored. The ordered instructions of this technological language work overtime to keep at bay the disorder of the world and the imprecision found in human language and thought. At the same time, this book is also a work of cultural analysis that examines what we will show to be the intersections of several distinct discourses that are all registered in this now obsolete and obscure computer language: the dreams and aspirations of 1960s computer and space science, the Cold War ideologies that enabled these technologies, the knowledge gained from the application of these technologies that was then used to advance and exercise imperial military power, and the traces of a counter-cultural language that emerged to supplement and at times resist components of the sparse, stripped-down syntax of these other discourses. Recovering, uncovering, and decoding these imbricated discourses requires
the resources of multiple fields and approaches — methods both specialized
and radically undisciplined.

Together, we take up a fascinating and now monumentally important
historical source text for our critical and creative readings: the source
code for the guidance computer that powered both the command and lunar
modules for the Apollo Project, and specifically the version or edition of the
code as used in the legendary Apollo 11 mission from July 16 to July 24 1969.
This book appears during the fiftieth anniversary of this historic flight and
we want to use this moment and our work to commemorate and critique this
scientific and cultural event. This code was one of the technologies that
made space travel possible; it would not be wrong to say that we wrote our
way to the Moon. The Apollo Project, with its grand ambitions and aims, has
inspired countless students, scientists, and engineers to dream big, to find
and follow their vocations into the sciences and the arts, and to launch
their own large-scale imaginative projects. Yet one of the most crucial
newly developed technologies that enabled the astronauts to land on and
return from the Moon, the digital computer that provided these astronauts
with guidance data and assisted in the control of the Lunar and Command
Module, has remained somewhat cloaked in obscurity. Unavailable and un-
interpretable to the larger public, the text of the code powering this revo-
lutionary computer remained locked within what we might call its base or
bare functionality.

Each section of this book highlights and illuminates different aspects
and dimensions of the Apollo Guidance Computer (AGC) code and the cultur-
al moment that enabled its construction. We are producing code commen-
tary — remarking and remixing the code. We intend no single account of
the code to be definitive; our purpose in presenting critical commentaries
alongside poetry is to interrupt the desire to fix and re-instrumentalize our
source text. Instrumentalization, in part, involves the flattening of a tech-
nology into a mere tool and the privileging of what we might term the an-
thropological account of a technology as a means by which to accomplish
some goal. In reading an object that one might assume to be the province
of one culture through the tools and methodologies of another, we want
to show that this division, the now entrenched separation of the sciences
and the humanities, itself has already been called into question by the in-
vention of code. Proceeding from here, we provide wide ranging readings, responses, and interpretations of the code that we believe will aid our readers in thinking broadly about exploration, collaboration, and computation. Moonbit will not get you to the Moon, but seeks to re-claim the text that did this, as a site for artistic exploration.

It is in this spirit that we write this book as a collaborative project. Inspired by the collective work of over four hundred programmers, writers, engineers, project managers, and others who worked on the various Apollo 11 digital computer systems — both hardware and software — not to mention the hundreds of thousands participating in the larger 1960s space program itself, we “compiled” this book from a critical reading and what could be called a deformation of its source text into a collection of poems and expansive commentary. We would like to think of this project as a set of remarks — here we use remarks in order to riff on the term for the existing formalized commentary supplied by the original authors of the code and included within the body of the code — on the code that frame and elaborate the meaning of the code at its point of origin in 1969, its longer historical context of the development of computing and scientific exploration, and the code’s meaning for our present moment.

The authors of another study and exploration of “old” and obsolete code, 10 PRINT CHR$(205.5+RND(1)); : GOTO 10, faced a much larger task than ours at present: convincing their readers that their singular titular line of BASIC code for the Commodore 64, a popular home computer produced during the 1980s, was an important cultural artifact and one worth engaging with in the present and that their interpretations and readings had value for software studies. They write:

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

---

1 In “New Methods for Humanities Research,” his 2005 Lyman Award Lecture, prominent digital humanities scholar John Unsworth cites Bill Wulf, a former president of the National Academy of Engineering, as arguing that “computer science should really be considered one of the humanities, since the humanities deal with artifacts produced by human beings, and computers (and their software) are artifacts produced by human beings,” http://people.virginia.edu/~jmu2m/lyman.htm.
prove persuasive, these arguments will have implications for the interpretation of software of all kinds.\textsuperscript{2}

The examination of obsolete code, whether written, modified, and used by hundreds of thousands of hobbyist home computer owners or developed in secret for the US nationalist project of space exploration, brings elements of the past into the present and reveals how this obscure computational past might, to riff on William Faulkner, not even be past. We believe that the AGC code is of as much historical and cultural interest, if not more, as the memoirs, recordings, and documents that serve to record and shape our understanding of the inception and development of the US space program.

Source code appears throughout this book, sometimes with extensive commentary that draws out the implications, assumptions, and desires of the authors, and other times lines of code appear as suggestions or provocations. We do not expect the reader to be familiar with the specific language used or to have studied computer science. We present code as an interpretable object. This is because this particular code text, while restricted to the confines of the fixed format dictated by 1960s coding standards and requirements, contains a rich set of meta-commentary that explains as it codifies — that attempts to account, in a series of remarks, for the many decisions made and choices selected within the code. Code, it might surprise you to learn, is not written just for a computer; code, as we will show, has many audiences and can be shaped into several different forms. Code is not just what is executed by the computer, but a language, a discourse, with creative and functional possibilities. Contrary to the common perception of programming, code is not just a set of instructions, it is not just math. Even in the earliest and simplest of computer languages, written code is frequently imaginative and has the capacity to be wildly playful. Code contains within it a poetics of its very own. There is an aesthetics to be found within the construction of code but these aesthetic features sometimes exceed their functional value. We believe the AGC code to be truly remarkable code.

This book, in part, seeks to provide an introduction to the theory and practice of critical code studies. We seek to outline a more capacious version of critical code studies that takes up all manner of imaginative decodings and recodings of our object of analysis. In introducing some of the

\textsuperscript{2} Nick Montfort et al., \textit{10 PRINT CHR$(205.5+RND(1)); : GOTO 10} (Cambridge: MIT Press, 2013), 5.
major existing approaches to the study of code and culture, we attempt to provide multiple readings of the source code along with an explanation and theorization of the way in which the Apollo Guidance Computer code works, as both a computational and a cultural text. We tend, however, to privilege the cultural rather than technical meanings of the code as we unpack, deform, and explicate. There are a number of existing accounts of the AGC hardware and software and while we will explore some of the functional purposes of this “antique” code, we are finally more interested in the way in which the code can become meaningful to its human readers. This is to say that we believe the code makes and contains interesting cultural commentary that we can read in relation to the historical moment in which the code was developed and used.

We draw out buried meaning and recode what was punched out through several interpretive and creative methods, including erasure. The AGC code provides rich source material that is about motion as much as it is about communication — complete with scatological jokes in the commentary. This code put people on the Moon and continues to inspire discovery. Erasure poetry, like the source language that it borrows from, offers itself as a way to memorialize or monumentalize while also making something new. The erasure method begins with a complete source text — really any sort of object — and removes much of it, creating a new text, a poem entirely wrought from some other primary textual source. Jen Bervin’s Nets takes Shakespeare’s sonnets as its source and erases most of the words, carving entirely new poems out of canonical literature. In contrast, Tom Phillips, in his art book A Humument, takes an unknown Victorian novel, A Human Document by W.H. Mallock, and erases most of it. Phillips makes each page into an original work of art, with only a few of Mallock’s original words remaining. M. NourbeSe Phillip’s Zong! makes a coherent cacophony of what remains from a massacre of one hundred and fifty slaves who were pushed off the slave ship Zong, so that the Zong’s investors could re-coup what they lost in a failed venture in the form of insurance money. This case left behind a legal legacy of barely five hundred words. Phillip’s book-length Zong! poem gives voice to those massacred people and distressingly, but correctly, offers the reader no consolation.3

3 Jen Bervin, Nets (Brooklyn: Ugly Duckling Presse, 2004); Tom Phillips, A Humument (London: Thames and Hudson, 1980); M. NourbeSe Philip, Zong! (Middletown: Wesleyan Univer-
Poems about space travel crave white space on the page. Here the white space represents the unknown cosmos or white light from the stars or perhaps the white face of the Moon itself. Erasure creates white spaces. Erasure creates room to breathe and space to think by finding holes within the source text or creating holes by erasing existing marks and larger textual structures. It navigates through these gaps, found or created, within the source text to bring something new into being. This debris may be of use. While there are computational methods for automatically producing erasure poetry, the poems in this book follow no program. They are human responses to code written by other humans. As William W. Cook argues of Frederick Douglass's understanding of learning to write by "writing in the spaces left" in a source text: "In the spaces left he finds those uninscribed topoi necessary to his own creation. He writes a hand similar to, but not identical with, that of his model preparatory to taking full control of the text itself. Imitation and repetition lead here to creativity and liberation." The AGC code itself contains multiple languages, multiple worlds. It contains subroutines to alter our orientation, to translate our coordinates, to alter its internal representation of space in terms of the Earth and the Moon. Erasure, for Brian McHale, engages in a cycle of "making and unmaking" that, in the case of James Merrill's work, "structure (and deconstruct) the world, or rather the worlds in the plural." Applied to this multiple-worlded text at the limits of modernity, erasure reinserts the hand into the machine to liberate the poet and this text and in the process, destabilize the inscribed formal relations among the represented bodies.

The four hundred programmers and engineers working on the Apollo Guidance Computer were employed by the Draper Laboratories, later the MIT Instrumental Lab, in Cambridge, Massachusetts. The code was not developed in isolation; it builds upon prior knowledge and expertise, collabora-


tors within other organizations and departments, and the contributions of consultants and industry partners. These programmers and engineers developed the code with instructions from NASA, editing and debugging from Cambridge, while astronauts departed from Earth with their code, their creation, within the lunar and control modules.

The advent of the digital computers placed in the Apollo Lunar Module marked an incredibly important development in the history of digital computing and space flight. In the past few years there has been an increasing amount of interest in these systems and the people behind the development of this early code. Images of the printed code were scanned and uploaded to the information sharing site iBiblio and optical character recognition (OCR) software (along with some manual editing) was used to render these images of printed text legible in digital form. The archives of the code enabled hobbyists and space enthusiasts to explore and play with the AGC code but it remained difficult for browsers to understand the larger code project in its entirety until the text of the code was made available in a new form. It was, then, in 2016 that an intern at MIT uploaded the AGC code to the code repository Github, enabling global and easy access to the code along with the collaborative editing, commentary, and revision tracking system provided by the site. In the process of moving the code into Github, the code was segmented into separate files and presented in a form that would work with the Github conventions for displaying code, including the transformation of certain code features that formerly belonged in fixed positions into a contemporary, less structured form.

While the conversion of the AGC code into Github drew our attention to this code, our primary driver for exploring the code was the growing attention to the work of one particular MIT Instrumental Lab staff member that coincided with the Github “publication” of the Apollo code. In “compiling” our readings and responses into this book we seek, above all, to recognize and acknowledge the contributions of Margaret Hamilton, lead programmer on the Apollo Guidance Computer project. Hamilton was one of the few women working in the nascent field of computer engineering and the only female senior staff member. In November 2016, President Obama awarded Margaret Hamilton the Presidential Medal of Freedom. In his citation, Obama wrote of her many contributions, all of which were first imagined and explored in the text of the AGC code examined by this book: “Hamilton contributed to concepts of asynchronous software, priority scheduling and priority displays, and human-in-the-loop decision capability, which set the
foundation for modern, ultra-reliable software design and engineering.” Hamilton’s work on the Apollo project and that of many others helped to establish the field of software engineering and legitimized new discursive practices. Her work and imagination inspires our own flights of fancy as we produce numerous readings of the code that she committed to the Apollo Project.

7 President Barack Obama’s citation reads as follows: “Margaret H. Hamilton led the team that created the on-board flight software for NASA’s Apollo command modules and lunar modules. A mathematician and computer scientist who started her own software company, Hamilton contributed to concepts of asynchronous software, priority scheduling and priority displays, and human-in-the-loop decision capability, which set the foundation for modern, ultra-reliable software design and engineering.” Office of the Press Secretary, “President Obama Names Recipients of the Presidential Medal of Freedom,” The White House, November 16, 2016, https://obamawhitehouse.archives.gov/the-press-office/2016/11/16/president-obama-names-recipients-presidential-medal-freedom.