Computer Science Logo Style
Advanced Techniques

Brian Harvey

Computer Science Logo Style

SECOND EDITION

Volume 2
Advanced Techniques

The MIT Press
Cambridge, Massachusetts
London, England

For information on program diskettes for PC and Macintosh, please contact the Marketing Department, The MIT Press, 55 Hayward Street, Cambridge, Massachusetts, 02142.

This book was typeset in the Baskerville typeface.

The cover art is an untitled mixed media acrylic monotype by San Francisco artist Jon Rife, copyright © 1996 by Jon Rife and reproduced by permission of the artist.

Library of Congress Cataloging-in-Publication Data

Harvey, Brian, 1949–
Computer Science Logo Style / Brian Harvey. — 2nd ed.
p. cm.
Includes indexes.
Contents: v. 1. Symbolic computing. — v. 2. Advanced techniques — v. 3. Beyond programming.
ISBN 0–262–58151–5 (set : pbk. : alk. paper). — ISBN 0–262–58148–5 (v. 1 : pbk. : alk. paper). — ISBN 0–262–58149–3 (v. 2 : pbk. : alk. paper). — ISBN 0–262–58150–7 (v. 3 : pbk. : alk. paper)
1. Electronic digital computers–Programming. 2. LOGO (Computer programming language) I. Title.
QA76.6.H385 1997
005.13′3—dc20 96–35371
CIP

Contents

Appendices

Preface

This is the second volume of *Computer Science Logo Style,* a three-volume series that uses the Logo programming language as the medium for a presentation of a range of topics in computer science. The main audience I had in mind for these books was high school students, but it's turned out that they have also been used in teacher training, and to some extent by independent adult learners.

In the first edition, the first volume was a complete Logo tutorial, explaining all of the features of the language; the second volume was entirely devoted to programming projects. (The third volume, then and now, is a sampler of topics from undergraduate computer science courses.) My idea was that students would spend their first year in an intensive programming course, and would then pursue their own programming projects on an independent study basis, using my projects as examples.

As it turned out, people found the first volume both too hard and too easy. It was too hard because it arrived too soon at the more advanced and complicated features of Logo; it was too easy because the actual programming examples were all short enough to fit on a page. Such tiny examples didn't help the learner extrapolate to the design of a program that could actually do something interesting. This deficiency may have encouraged some readers to conclude that Logo is just a toy, and that serious projects should be done in a "serious" language such as Pascal or C++.

In this second edition I've rearranged things. The first volume now teaches only the core features of Logo, the ones every programmer must understand; it also includes three of the projects that were originally in the second volume. This volume is now a more advanced programming text; it alternates tutorial chapters on advanced language features with example projects that demonstrate those features.

The project chapters serve two purposes at once. First, each project is an example of something you might actually want to do. The emphasis is on getting the computer

to do something fun and interesting. Each of the projects in this book is here because I thought I'd enjoy writing it myself, not because it fit some subtle pedagogic purpose. The projects are offered as case studies, as examples to inspire your own creative efforts.

At the same time, I *am* a teacher, and in this book I'm trying to teach some ideas about programming technique and programming style. Often there is an easy way and a hard way to achieve a certain result, and you're better off if you know the easy way. Nobody has a complete list of such techniques; you'll be learning new ones for as long as you maintain your interest in computer programming. The ones I discuss in this book are the ones that came up in these particular projects. Ideally, as your teacher, I would look over your shoulder while you're working, and I'd tell you about the techniques that apply to *your* projects. I can't do that in a book, and so instead I'm presenting some projects of my own and discussing them as I would discuss your projects if I knew you personally.

With one exception, each example chapter comes after a tutorial chapter that has introduced a new Logo programming technique, and that technique is used in the project. (The exception, the pattern matching project, is an advanced programming technique in its own right, and is used in a later project.) But the technique from the previous chapter is rarely the most important aspect of the project! Each project exhibits many different techniques, and the project chapters describe some of them.

This book does not make much explicit reference to the first volume, but to understand the discussion here, you should be familiar with the ideas presented in Volume 1: evaluation, procedures, locality, iteration, recursion, mapping, predicates, operations, and so on.

Teaching and learning, by the way, don't necessarily imply a classroom in a school. I like to imagine you curled up with this book in front of your home computer, playing around with one of these projects just for the fun of it. Pretend I'm a friend or relative who happens to be a professional computer scientist. On the other hand, if you *are* reading this for a course in a school, you have the advantage of a living teacher who can provide the kind of individual attention to your specific projects that I can't. There are advantages and disadvantages either way.

About the Projects

Although I now have the projects linked with tutorial chapters, in the first edition I organized them into five categories, based not on the programming techniques used but rather on the purposes of the programs. The projects reflect aspects of my own

character: I came to computers by way of an early interest in mathematics; my computing background is in artificial intelligence and in systems programming; I tend to think in words, not in pictures. I think it may give the collection of projects a more coherent feel if I explain the categories in which they were written, even though the book is no longer organized around those categories.

The first is cryptography. One of the first books I can remember buying, as a child in elementary school, was about secret codes. Besides the universal appeal of knowing a secret, cryptography was interesting to me because it's a *mathematical* sort of puzzle, like those logic problems about who lives in the yellow house. The Cryptographer's Helper project in this volume includes a very small effort at artificial intelligence: the program makes some guesses, on its own, to start solving a cryptogram. The Playfair Cipher project, now moved to the first volume, deals with a more complicated technique for encoding a message, but it doesn't try to break such a code.

The second category is games. I'm not a video game enthusiast; hand-eye coordination isn't my strong point. (I never really learned to ride a bicycle!) Anyway, writing video game programs depends too much on the particular hardware of your computer, so I can't do it in this general book.* Instead I've written two simple strategy games. In the first volume is a program that plays tic-tac-toe. This game is extremely trivial for a human being, but it's surprisingly hard to formulate strategy rules that are simple and precise enough to embody in a computer program. Also, it's an opportunity to throw in a little bit of graphics programming, to draw the board and fill it with Xs and Os. In this volume is a program that deals out a hand of solitaire and maintains the display of the layout as you play the hand. Before I wrote this program, I had been feeling bored and lonely for an extended period, and I was wasting a lot of time playing solitaire myself. I figured it would be more productive to write a computer program!

The third category is mathematics. I once spent some time working as a systems programmer at a computer music research center in Paris, and this volume includes a project about Fourier analysis, the mathematical basis of computer music. The project demonstrates graphically how a complex waveform, representing the texture of a sound, can be built up from much simpler elements. In the first volume is a program to solve the kind of problem, often found on IQ tests, in which you are given pitchers of certain sizes and asked to use them to measure a given amount of water by pouring back and forth. This project illustrates the idea of searching through a "solution space" of possible pouring steps.

---

* You can find a video game that I wrote in the collection *LogoWorks: Challenging Programs in Logo*, edited by Solomon, Minsky, and Harvey (McGraw-Hill, 1985).

The fourth category is that of utility programs. This is actually the area of programming I know best: writing things that are not complete applications in themselves, but rather tools to help in the creation of even larger projects. For the second edition I've replaced the original projects in this category with two new ones. The project Finding File Differences is a utility program that can be used to compare two versions of a file to see what's changed from the old one to the new one. Then there is a compiler for the BASIC programming language; besides illustrating the idea of program as data—the compiler generates new Logo procedures to carry out the instructions in a BASIC program—this project may help to prepare the reader for the more complicated Pascal compiler in the third volume.

The fifth category is pattern matching. This category combines my interests in systems programming and artificial intelligence. The first project is a tool, like the ones in the utilities category, but it's a tool designed specifically for artificial intelligence applications: a pattern matcher. This program compares a particular list with a general template, or pattern. Instead of checking for exact equality like `equalp`, the pattern matcher checks for a kind of "fill in the blanks" partial equality. The second project in this category uses the pattern matcher to implement `doctor`, another famous artificial intelligence program that simulates a conversation with the user.

About This Series

Computer Science Logo Style is intended to bring to the hobbyist audience a particular point of view about computer science: the artificial intelligence view. This way of looking at computers is quite different from the more usual software engineering approach. In that approach, you are always dealing with a very well-defined problem, and are looking for the best way to solve it. Software engineers like to start with a formal problem statement, and then design a computer program to fit. They believe that the design process should be *top-down*; you should start with the overall structure and work down to the details. Their preferred programming language is Pascal.

In artificial intelligence, the problems are not usually so well defined. Starting with a vague problem statement like "develop a good strategy for playing chess," AI programmers can't begin with a rigid program specification. Instead, they build *tools:* program fragments that can be pieced together to form larger programs. The programming process involves writing code, testing, coming up with new ideas, and modifying the program interactively. This process is encouraged by an interactive language like Lisp or Logo.

Computer programming is a great intellectual hobby; it provides the same opportunity for creative, concrete work in mathematical thinking that drama or creative writing does for verbal thinking. A learner can have years of intellectual adventure just learning to write better and better programs. Finally, though, there may come a time when the learner gets bored with just writing more and more programs, and seeks a deeper understanding of the issues behind this practical work. The third volume of this series, *Beyond Programming,* addresses the needs of these learners by introducing them to some of the elements of university-level computer science, still in the context of Logo programming.

How to Read This Book

You should have each program actually available to you on a computer as you read about it. These programs are available on diskette from the MIT Press, or can be downloaded from the Internet. Details are in the first volume.

There are many dialects of Logo; this book uses Berkeley Logo, a free version available for PC, Macintosh, and Unix systems. The more fundamental Logo techniques used in the first volume are more or less standard among Logo implementations, but some of the advanced techniques in this volume are unique to Berkeley Logo. It, too, is on the diskette and the Internet.

The programs you see here are essentially the programs I wrote as I was trying to get each project to work. I didn't start with a particular programming style in mind and then invent an example to illustrate the style. It's not always obvious what is the "correct" style for a given problem; sometimes one way is much easier to understand, for example, while a different solution may run much more efficiently. The comments in each chapter sometimes suggest alternative ways in which I might have written some piece of the program. I try to explain why I chose the style I did, although sometimes the real explanation is simply that that's the first thing I thought of. I've modified almost all of these programs for the second edition, and some of the chapters explain my second thoughts.

Each example chapter begins with an explanation of what the project is all about. Remember that these projects were meant to be interesting in themselves, not just as vehicles for a discussion of programming techniques! The discussion in each chapter ends with a return to the purpose of the project, with suggestions for how that purpose might be extended. One source of ideas for projects of your own is to extend someone else's work, and one important purpose of this book is to give you ideas for such starting points. In between comes a technical discussion of the programming techniques used.

What I do *not* provide, generally, is a guided tour of every procedure. One of the things you should learn from this book is the ability to read a long program on your own. You should recognize some of the typical categories of procedures, like ones that apply a given command to each member of a list. In the discussions, rather than explain every detail, I try to focus your attention on the parts of the program that seem to illuminate some more general technical issue. A complete listing of the program is at the end of each example chapter.

The programs in this book are copyright, but you can use, copy, and redistribute them freely; the exact terms are given in the GNU General Public License, which is distributed with the programs and is printed in the first volume of this series. Essentially, the only restriction is that you can't use these programs as the basis for your own commercial programs; if you extend these projects, you can only distribute your extensions on the same free terms. Share ideas, don't hoard them!

Acknowledgments

Cynthia Solomon and Margaret Minsky are the people who got me started at the enterprise of developing exemplary Logo projects. People in the Logo community had been talking for many years about the need for an advanced Logo project book, but nobody got around to it until 1982 when Atari had all the money in the world and used some of it to establish a Corporate Research Department. Cynthia was in charge of the Atari research lab in Cambridge, where many MIT old-timers were gathered. She and Margaret decided that this was the time for the project book. I was one of several people they recruited to contribute projects. The result of that effort is called *LogoWorks: Challenging Programs in Logo* (McGraw-Hill, 1985).

This book is somewhat different from *LogoWorks* in that it's part of a series, so I can make assumptions here about what the reader already knows from having read the first volume. Still, I've benefited greatly from what I learned from Cynthia and Margaret about how to explain the structure of a large programming project.

The people who have read and commented on early drafts of this book include Hal Abelson, Alison Birch, Sharon Yoder, Mike Clancy, Jim Davis, Batya Friedman, Paul Goldenberg, Margaret Minsky, and Cynthia Solomon. As for the first volume, I am particularly indebted to Hal and Paul for their strong encouragement and their deep insights into issues both in computer science and in education. Matthew Wright reviewed some chapters for the second edition.

Berkeley Logo, the interpreter used in this edition, is a collective effort of many people, both at Berkeley and across the Internet. My main debt in that project is to three former students: Dan van Blerkom, Michael Katz, and Doug Orleans. At the risk of missing someone, I also want to acknowledge substantial contributions by Freeman Deutsch, Khang Dao, Fred Gilham, Yehuda Katz, George Mills, and Randy Sargent.

Computer Science Logo Style
Advanced Techniques