

Logo Exchange

Journal of the ISTE Special Interest Group for Logo-Using Educators



THE BEAUTY OF MATHEMATICS

INSIDE

- 20 Reasons to Use Boxer (Instead of Logo)
- Simulating Artificial Life with Logo
- Giving Meaning to Mean (and Standard Deviation)
- Meaning and Math
- 21st Century Logo Quilts
- Chipping Away at Mathematics
- Research and Mathematics Education Standards
- Book Review, Teacher Feature, The More Things Change

Logo Exchange

Volume 17 / Number 3

Editorial Publisher

International Society for Technology in Education

Editor-in-Chief

Gary S. Stager, Pepperdine University
logoexchange@moon.pepperdine.edu

Copy Editing, Design, & Production

Ron Richmond

Founding Editor

Tom Lough, Murray State University

Design, Illustrations & Art Direction

Peter Reynolds, Fablevision Animation Studios
pete@fablevision.com

Contributing Editors

Dr. Douglas Clements, SUNY Buffalo
Dr. Carolyn Dowling, Australian Catholic University
Alan Epstein, Metasoft
Dr. Brian Harvey, U.C. Berkeley
Daniel E. Kinnaman, *Curriculum Administrator Magazine*
Dr. Julie Sarama, Wayne State University

International Editor

Jeff Richardson, Monash University, Australia

International Editor Emeritus

Dennis Harper, Olympia, Washington School District

SIG Logo Officers

Stephen Sesko, President
Jeff Richardson, Vice-President
Hope Chafian, Secretary/Treasurer
Gary S. Stager, Editor

SIG Coordinator

Tom Magness

1998-1999 ISTE BOARD OF DIRECTORS

ISTE Executive Board Members

Lynne Schrum, President *University of Georgia-Athens (GA)*
Heidi Rogers, President-Elect *University of Idaho*
Cheryl Lemke, Secretary *Milken Family Foundation (CA)*
Michael Turzanski, Treasurer *Cisco Systems, Inc. (MA)*
Chip Kimball, At Large *Lake Washington School District (WA)*
Cathy Gunn, At Large *Northern Arizona University*

ISTE Board Members

Larry Anderson *Mississippi State University*
Jose Calderoni *ILCE, Mexico*
Penny Ellsworth *Western Springs School District 101 (IL)*
Marianne Handler *National-Louis (IL) University*
Dennis Harper *Olympia School District (WA)*
Jorge Ortega *FACE/Leon County SD (FL)*
Neal Strudler *University of Nevada, Las Vegas*
Sue Waalkes *Upper Dublin School District (PA)*
Peter Wholihan *Sts. Paul & Peter School, Virgin Islands*

ISTE Committees

Lajeane Thomas *Accreditation and Standards*
Dave Brittain *Awards*
Cathy Gunn *Distance Learning*
Michael Turzanski *Finance*
Paul Resta and Gerald Knezek *International*
Jenelle Leonard *Minority Affairs*
Lary Smith *Policies and Procedures*
M. D. Roblyer *Publications*

ISTE Executive Officer for Research and Development

David Moursund

Logo Exchange is published quarterly by the International Society for Technology in Education Special Interest Group for Logo-Using Educators. *Logo Exchange* solicits articles on all aspects of Logo use in education.

Submission of Manuscripts

Manuscripts should be sent by surface mail on a 3.5-inch disk (where possible). Preferred format is Microsoft Word for the Macintosh. ASCII files in either Macintosh or DOS format are also welcome. Submissions may also be made by electronic mail. Where possible, graphics should be submitted electronically. Please include electronic copy, either on disk (preferred) or by electronic mail, with paper submissions. Paper submissions may be submitted for review if electronic copies are supplied on acceptance.

Send surface mail to:

Gary S. Stager
21825 Barbara St.
Torrance, CA 90503 USA

Send electronic mail to:

logoexchange@moon.pepperdine.edu

Logo Exchange is published quarterly by the International Society for Technology in Education (ISTE), 1787 Agate St., Eugene, OR 97403-1923, USA; 800.336.5191.

ISTE members may join SIG/Logo for \$24. Dues include a subscription to *Logo Exchange*. Non ISTE member subscription rate is \$34. Add \$10 for mailing outside the USA. Send membership dues to ISTE. Add \$4.00 for processing if payment does not accompany your dues. VISA, MasterCard, and Discover accepted.

Advertising space in *Logo Exchange* is limited. Please contact ISTE's director of advertising services for space availability and details.

Logo Exchange solicits articles on all topics of interest to Logo-using educators. Submission guidelines can be obtained by contacting the editor. Opinions expressed in this publication are those of the authors and do not necessarily represent or reflect official ISTE policy.

© 1999 ISTE. All articles are copyright of ISTE unless otherwise specified. Reprint permission for nonprofit educational use can be obtained for a nominal charge through the Copyright Clearance Center, 27 Congress St., Salem, MA 01970; 508.750.8400; Fax 508.750.4470. ISTE members may apply directly to the ISTE office for free reprint permission.

POSTMASTER: Send address changes to *Logo Exchange*, ISTE, 480 Charnelton St., Eugene, OR 97401-2626 USA. Periodicals postage paid at Eugene, OR. USPS# 000-554. ISTE is a nonprofit organization with its main offices housed at the University of Oregon. ISSN# 0888-6970.

This publication was produced using Aldus PageMaker®.

Contents

ARTICLES

Twenty Reasons Why You Should Use Boxer (Instead of Logo)
A. A. diSessa 7

A Note from Andy diSessa, creator of Boxer
A. A. diSessa 17

Simulating Artificial Life with Logo
Bill Engel and Pat Greene 20

Giving Meaning to Mean (and Standard Deviation, Too)
Tom Lough 34

COLUMNS

EDITORIAL

The More Things Change . . .
Gary S. Stager 2

QUARTERLY QUANTUM
 Meaning and Math
Tom Lough 3

TEACHER FEATURE
 Stephen Costa
Gary S. Stager 4

BOOK REVIEW
 Making Numbers Count
Carolyn Dowling 5

IN THEIR OWN WORDS
 Using Spezeski's
 Poly.Round Procedure
John Hayes 23

STARTING WITH STARLOGO
 Auto Maze
Alan Epstein 25

LOGO: SEARCH AND RESEARCH

Research and Mathematics
 Education Standards
Douglas H. Clements
and Julie Sarama 27

FOR BEGINNERS
 21st Century Logo Quilts
Gary S. Stager 31

THE LAST WORD
 Chipping Away at Mathematics:
 A long-time technophile's worries
 about computers and calculators
 in the classroom
E. Paul Goldenberg 36

The More Things Change . . .



School math . . . Few words strike such fear in the hearts of the public. (Although I did once see an exhibit at my local shopping mall celebrating “Mobile Army Dentistry.” Does the Army actually believe that this will boost recruitment?) For me, Logo has been the antidote for the years of psychic damage I endured at the hands of school math teachers.

School math should not be confused with the actual field of mathematics. There may be no other school subject whose teaching and curriculum bears so little resemblance to the actual discipline. School math is mechanics, mathematics is mystery. School math is a topic per week with test on Friday, mathematics is a way of understanding the world. School math is about marks on paper, mathematics is about beauty. Logo offers many children an opportunity to acquaint themselves with that beauty, mystery, and epistemology.

The 1990 National Council of Teachers of Mathematics Standards contain a statement that has challenged my thinking for several years now. The Standards state that “More than 1/2 of all mathematics has been invented since WW II.” Where is this stunning piece of news reflected in most math curricula? To ignore this progress is to deny lots of kids access to exciting new branches of more playful, experimental, visual mathematics such as chaos,

fractal geometry, number theory, topology, and cellular automata. These emerging topics may provide a port of entry to the beauty and power of mathematics for learners, like myself, who were not moved by solving dozens of identical quadratic equations.

The Standards go on to explain the causes of this explosion in mathematical progress.

1. Science and technology make ever new demands on mathematics for assistance.
2. Each new, completed result becomes the potential starting point for several new investigations. The new technology not only has made calculations and graphing easier, it has changed the very nature of the problems important to math and the methods mathematicians use to investigate them. (1990 NCTM Standards, page 8)

When I discussed this at the recent NCTM conference, people in the audience predicted that closer to 80 or 90% of all mathematics has now been invented since World War II. I sincerely hope that the 2000 NCTM Standards (take a look at drafts at www.nctm.org) will acknowledge the rapid advances in mathematical knowledge as well as the reluctance of the math education community to seize upon the

exciting potential this progress holds for learners of all ages.

This issue is dedicated to thinking about mathematical thinking and teaching. There are wonderful ideas for classroom projects, as well as provocative essays challenging us to declare our intentions for the future. Logo pioneers Andy diSessa and Paul Goldenberg, both Logo users since the 1970s, have contributed thoughtful pieces to this issue. Andy proudly announces the long-awaited release of Boxer and states that we should all be using it because it’s better than Logo. You decide and let us know what you think! Paul contributed part of a provocative paper in which he muses over what may actually be lost as we use computers and software, like Logo, to learn mathematics. Please think about their hypotheses and share your views with us at *Logo Exchange*.

I look forward to hearing from you! **LX**

Gary

Gary Stager, Editor-in-Chief
logoexchange@stager.org

See **LETTERS TO THE EDITOR** (Page 19)



Meaning and Math

Occasionally, it happens to each of us. You know what I mean. One of those cosmic connections, a serendipitous chain of events with ideas that seems to happen on their own, without warning.

For me, it came about as a result of two widely separated actions that seemed to have little relationship to each other. One was an appointment with my college dean, and the other was the simple act of reading my mail at home. Let me sketch them out for you.

Our college hired a new dean several months ago. One of the first things he did was to invite each faculty member for a one on-one appointment. I arrived a bit early for my meeting, and took advantage of the extra time to survey the books in his professional library. A slim volume with the title of *Logo Learning* practically leaped off the shelf and into my hands! Immediately after my meeting, I signed the book out and promised myself I would read it.

Several weeks later, I opened my mail to discover the February 1999 issue of *Phi Delta Kappan* magazine. The cover article for that issue was "The Mathematical Miseducation of America's Youth." I thumbed through the pages and promised myself I would give it a read sometime soon.

For one reason or another, I did not accomplish either reading. Then, when Gary Stager mentioned recently that he was collecting mathematics articles for this *Logo Exchange* issue, I remembered these two events. At last, things started into motion; I took out the publications and began to read.

The *Phi Delta Kappan* issue had two articles that caught my eye. In the fea-

ture article, "The Mathematical Miseducation of America's Youth: Ignoring Research and Scientific Study in Education," Michael Battista contrasts the traditional mathematics teaching style with that being urged by current educational reform. Lamenting that "the only time that Americans pay any attention to mathematics teaching is when educators attempt to improve it," he sketches out the formidable barriers to reform.

But then he focuses the reader on the nature of learning mathematics and its relationship to a constructivist point of view, suggesting that students personally construct mathematical meaning from their experiences. (Meaning. Hmm. Where have I seen that term before?)

A second article in the same issue (positioned outside the mathematics theme area) has the engaging title of "*Shazam! You're a Teacher: Facing the Illusory Quest for Certainty in Classroom Practice.*" Author Selma Wassermann develops the idea that one of the most important tasks teachers do is to make meaning of events in the classrooms. By sizing up a situation and reflecting on what it means, teachers are able to choose appropriate actions. She goes on to suggest that the ability to make meaning is a learned set of skills, and outlines a way this might be done. (More meaning. Hmm. How is all this coming together?)

Aha! The *Logo Learning* book! I was disappointed when I first thumbed through it to discover that it was not about Logo! Perhaps that is why I did not rush to read it. But now I remembered the subtitle: *Searching for Meaning in Education*. In less than 150 pages, author Dale Parnell sketches out the importance of meaning in teaching and learning.

He defines Logo Learning as an educational philosophy and an educational strategy that centers on enabling students to find meaningfulness in their education. One of the major tasks of Logo Learning teacher is to "broaden the student's perceptions so that meaning becomes visible and the purpose of learning immediately [becomes] understandable."

I found my thoughts flashing back through the years to a book I remember that was about mathematics and about students performing learning tasks that were personally meaningful. Yes, *Mindstorms* had all the elements even then!

I am still marveling at how a non-Logo book named *Logo Learning*, an issue of *Phi Delta Kappan* magazine, and a computer language named Logo all came together in my life, each focusing on the importance of meaning in mathematics.

When a student asks, "Why do I have to learn this stuff?" I want to have an answer ready. In the meantime, I have a lot to think about!

FD 100!

LX

Reference

Parnell, Dale. (1994) *Logo Learning: Searching for Meaning in Education*. CORD Communications, Waco, Texas.

**Tom Lough, Founding Editor,
Murray State University
Department of Elementary and
Secondary Education,
PO Box 9, Murray, KY 42071.
phone: 502.762.2538
fax: 502.762.2540
tom.lough@coe.murraystate.edu**



Stephen Costa

by **GARY S. STAGER**

Steve Costa is acting head of the Junior School at Methodist Ladies' College (MLC), in Melbourne, Australia. He has been an upper-primary teacher and administrator at the school for more than a decade. In 1989, Steve became perhaps the first teacher in the world to teach a class of children in which every kid had a personal laptop computer. MLC achieved international acclaim for their commitment to constructionism and personal computing. Steve played a major role in that success and welcomed thousands of educators from across Australia into his classroom to observe children learning with laptops and Logo. Few teachers have had more impact on their peers than Steve Costa.

Q: How did you get started with Logo?
In 1981 I began using an Apple computer. I quickly discovered how to turn it on, load a floppy disk and enjoy the excitement of playing a "computer game." After playing low-res, green screen, shoot em ups or spelling words to zap "killer bugs;" Logo was the first truly educational package I stumbled upon. Logo was "hard." It made me think but it was fun 'cause I was in control. I could have fun while learning and was able to "see" a graphical representation of my abstract commands.



Q: What is the most satisfying thing you and/or your students have done with Logo?
This sounds like a cop-out, but there would be numerous satisfying moments I have had with my students. One line of satisfying moments for me comes under the heading of "Eureka." This occurs when students are working on a "serious" project and they discover something by accident. They type some commands and the outcome is totally different from what they expected. The "unexpected outcome" instills in them a desire to find out

why! This self-directed, self-imposed, serious research into unraveling their commands is an exciting time for all involved. The learning is real, purposeful and helps to instill a sense of discovery and an enjoyment in "thinking" and "acting like a turtle," and a knowledge that they are being creative, in control.

Q: What did the laptop bring to the Logo experience?

The introduction of the laptop program provided an ideal setting to help foster a true sense of community and a setting where a collaborative learning environment could flourish. As each student had access to a computer, they became more willing to share. They not only shared their work, but their ideas and skills as well. Students began to gather around "interesting problems."

Discussions, suggestions and debates on the best way to do something, or how best to solve a problem sprung up around the room. No longer was one's own individual work the only important aspect or priority. Students were aware they had "time" to learn. Logo takes time to understand, enjoy and become familiar with so magic can happen. An ability to have time to learn—a chance to learn when they

See **TEACHER FEATURE (Page 6)**



BOOK REVIEW

Making Numbers Count

by CAROLYN DOWLING

***Humble Pi: The Role Mathematics Should Play in American Education*, by Michael K. Smith, 1994, Prometheus Books, New York**

***Math: Facing an American Phobia*, by Marilyn Burns, 1998, Math Solutions Publications, Sausalito, CA**

Those of my acquaintance who claim to love mathematics could be accurately counted on the fingers of a two-toed sloth—and even then, the activity for which they profess such passion seems to bear little resemblance to math as it is taught, and all too frequently loathed and feared, in school and for a lifetime thereafter. In his foreword to Michael K. Smith's *Humble Pi*, Papert makes a similar observation in relation to the distinction between what he would regard as true mathematics and those school based activities described by Smith as “mathematics,” but which Papert would dismiss as mere “math” (p. 8). Such semantic and conceptual differences aside, education systems in many countries of the world share an anxiety, often very publicly expressed, concerning the effectiveness of the mathematical learning which is taking place in schools.

There can be little doubt as to the extent of the damage to individual self-esteem that has been wrought by “bad experiences” in the mathematics classroom. But is mathematics simply an easily identifiable scapegoat for a range of far broader concerns about educational principles and practices, or is

there special cause for disquiet, particularly in regard to both popular and systemic perceptions of the importance of mathematical competence as a basis for understanding the underlying structures of the world and of society, as a necessary preparation for dealing with the practicalities of everyday life and, perhaps most importantly of all, as a predictor of future success in advanced study and in a vast range of occupations? It is these three assumptions, together representing a widely held belief in the “supremacy” of mathematics, with which Smith takes issue, particularly in regard to certain consequences. These include the weighting given to scores in the mathematics component of the SAT test, and in other tests and examinations regarded as indicators of the competency of the nation at large.

As Seymour Papert writes in the foreword to this book, “People who write truly iconoclastic books cannot expect unqualified agreement” (p. 7), and there are many who, while agreeing with the broad directions of Smith's case, would suggest that he takes it too far. Nevertheless, he does succeed in mounting some compelling arguments in support of the notion that the study of mathematics in its current form at school level is not necessarily of benefit either to the individual or to society at large.


By contrast, Burns' book sits far more easily within the comfort zone of most readers. While acknowledging the existence of severe problems in the area of the teaching and learning of math-

ematics (and hence providing some justification for the feelings of inadequacy that many of us experience in this regard), she does not question the fundamental “usefulness” of mathematics, which she describes in her Introduction as “a subject so important to our lives,” nor the connection of competence in this particular discipline with broader cognitive capacities. She writes, in fact: “Employers in all fields of work have issued the same request across the country. Send us employees who can think, reason, and solve problems. The cry is loud and the call is reasonable. Children must be helped to learn mathematics in a better way than we were, so that mathematical limits do not shut them out of certain life choices and career options” (p. xi).

There is little in this book to threaten the long-term interests of the mathematical establishment and lobby group. There is, however, much to engage the attention of those who would favour evolutionary rather than revolutionary change in the nature and role of the mathematics that is taught in schools. Her emphasis on an active, exploratory, situated approach to numeracy is refreshingly in keeping with current thinking in a range of other disciplines. Somewhat in common with Smith, she advocates students engaging with problems possessing a complexity which results from their location within real world contexts rather than from artificially imposed constraints, and in which considerations from many “disciplines” may impinge

upon the solution—or solutions. Such solutions may indeed vary as appropriate in nature, in precision and in the pathways by which they are achieved, according to the demands of the particular situation and in line with individual cognitive preferences. In keeping with such an approach, Burns devotes considerable attention to the importance of developing skills and confidence in different methods of calculation including “in our heads”, “pen and paper” methods and the use of calculators, along with an appreciation of differing needs for accuracy in different contexts. She also argues persuasively for the importance of removing pressures of time on students (such as timed tests), so as to encourage exploration of alternative methods and solutions as distinct from the rote imposition of formulae.

Neither author is denying that a degree of numeracy is a valuable, possible essential tool for functioning effectively in today's world if only, as many of Burns' examples in particular would suggest, in the interest of ensuring that we are able to recognise value for money! The question of precisely which mathematical skills we have most need to master, and how this can be achieved in such a way that the interest and confidence of students is not destroyed in the process, remains at issue. Each of these books in its own way makes an important contribution to our search for answers, one by asserting our collective right to ask the “unaskable” questions, the other by fruitfully combining the insights of a math enthusiast (the author) with the experiences of “the rest of us” in the quest for ways in which we can all succeed in learning and using mathematics.

Read Carolyn's past reviews and browse the books at www.stager.org/lxbooks.html. 

Carolyn Dowling
Australian Catholic University
412 Mt Alexander Rd
Ascot Vale, Victoria 3032
AUSTRALIA
c.dowling@mercy.acu.edu.au

want to, or when they feel like working on their Logo activities. Time becomes more flexible and can be used to their advantage and not a constrictor or limit to the excitement of using a computer.

Q: How does Logo support/enhance the personal computing experience for kids?

Logo makes the user personally responsible for her own actions! If you tell the computer to do something it does it! If what turns out wasn't what you expected, you need to think about what needs to be changed. Or ask, “how is the computer interpreting your commands?” You need to be precise and clear in your thinking and in the procedure you write. The personal satisfaction comes in creating something that you started out to create. You set the challenge for yourself, you know if you have reached it.

Q: Can you tell us about some of the cool things kids have done with Logo?

I can remember a LogoWriter project a Year 5 (fifth grade) girl did in 1991. It had the entire “Princess saved by a Prince” story with animation, sound and music. The helpless Princess is attacked by dragon. Prince rides off on horse in pursuit of dragon. The “Dragon fight” with a prince (Dragon “breaks” with red streaming colors). During the marriage ceremony, complete with guard of honor, the computer played the “Here Comes The Bride” song as the bride and groom walked down the aisle. Sounds easy but at this time in Logo history, music was made with the Tone #pitch #duration commands.

Drawings were done using Setpos commands. Copying setpos pictures and editing the Xcoordinate or Ycoordinate to move over a bit generated big animations! The girls could not get enough of doing Logo. The laptop allowed them to spend “their time” at home, at school, at recess even,

if they were wished doing THEIR WORK.


Another was making Polygons... The girls worked on the 360 Theorem and understood how to divide the 360-degree trip by the number of corners! This helped to produce many nice triangles, squares, hexagons etc. But the seven-sided figure was tricky because they hadn't been introduced to repeat or how to use RT 360 / 7.

One girl's solution was to have 4 forwards and 4 turns of 51 degrees followed by 3 forwards and 3 turns of 52 degrees. Total trip 360 degrees and it looked like a Septagon! Lots of thinking went into this. A personal problem that was happily solved using Logo.

Q: What have you learned from Logo or teaching with Logo?

Kids are amazing, creative, and dying to discover new things. They are able to take a simple suggestion or idea and build upon it. They can take themselves to places well beyond the teacher's expectations.

Logo provides an environment where risk taking is encouraged. There are many ways to produce a similar end product. Never believe that they CAN'T do something, they will always amaze you, and raise your expectations and admiration for their ability to learn, think and achieve.

I have personally learned to be more patient. To allow more time for the kids to develop their skills and understanding of what Logo can do and what they can do. I constantly preach to teachers that they need faith. Faith that wonderful things will happen when using Logo if they stop worrying about what the kids are missing! They might need to give a bit more time to Logo and forget the timetable for a while. In the long run, the value of what is learned, produced and the concepts gained will make up for the maths, spelling or writing lesson not covered in a more conventional way. 



FEATURE ARTICLE

Twenty Reasons Why You Should Use Boxer (Instead of Logo)

by A.A diSESSA

This work was funded, in part, by the National Science Foundation, in grant number RED-9553902. The opinions expressed are those of the author, and not necessarily those of the Foundation.

Reprinted from: diSessa, A. A. (1997). Twenty reasons why you should use Boxer (instead of Logo). In M. Turcsányi-Szabó (Ed.), *Learning & Exploring with Logo: Proceedings of the Sixth European Logo Conference*. Budapest, Hungary, 7-27.

Introduction

This paper explains and advocates Boxer as a computational environment for educational purposes. I intend mainly to speak to the Logo community. However, I hope not to produce a paper for “insiders” only. Instead, Logo stands in for many open educational computing environments as, arguably, the best of the lot. When I say “Logo” in this paper, I mean to include successors like LogoWriter and Micro-Worlds.

Boxer and Logo share a great deal in terms of philosophy and purposes. Indeed, the design of Boxer emerged over a dozen years ago, from within the Logo project at MIT. It was an attempt to design a successor to Logo, capitalizing on all we had learned using Logo with children and teachers. Both Logo

and Boxer aim to provide the simplest but most powerful and unconstrained computational resources possible to “just plain folks” in the service of enhancing learning. Both projects believe that programming, in some form, is essential in truly liberating the computer’s power as a learning tool. Differences at the level of philosophy exist and are interesting, but are not the main issue here.

The differences between Logo and Boxer that are relevant to this paper are technical. That is, they have to do with the designed structure of the Logo and Boxer environments and their associated programming languages. “Technical,” however, does not mean either esoteric or unimportant. To go back to the beginning, the difference between text and a programming environment are “technical” in exactly the same sense. But, I think everyone would agree that the difference between what you can say and do with text versus a programming language is substantial.

Logo’s principal claim to fame was that it made programming easier and more accessible. It basically adopted a subset of the capabilities of a “difficult” programming language, Lisp, and changed the way people saw and used that environment. For example, Logo had a more friendly syntax. In view of this central and well-advertised advance, it is stunning that the basic form

of Logo’s presentation to the user has remained essentially identical from the late ’60s when it was originally designed. This lack of change is even more surprising in view of two other facts. First, Logo was plainly constrained by the teletype terminals that it originally used. There was no choice except using characters, words, and lines as basic structuring devices. Bitmapped graphics and even mice were yet to become serious design possibilities. Second, I believe it was evident even in those days that the restricted structural possibility afforded by a “typewriter” interface caused difficulties. I wrote a memo to the Logo Group in the late 1970s about these limitations and made some suggestions that, eventually, became Boxer. By now, I believe these limitations are even clearer and empirically verified, especially in contrast to Boxer. I’ll make reference to many particular limitations and some of the data that confirms these in the remainder of this paper.

I don’t mean to imply that Logo hasn’t changed in its nearly 30-year history. But, the changes are not, for the most part, in its basic structure. Logo has not taken advantage of dynamic, graphical display possibilities in its core. Programs are still words in sequence, broken into lines, and so on. Instead, changes have included having some internal application-like features (draw program features), some canned

interface “widgets” (like buttons and sliders), and most notably, parallel processing. Even the latter significant change, however, involves no change in the basic presentation of the language—only a new command, **launch**.¹

On this background, let me sketch 20 reasons why you should use Boxer (instead of Logo).

1. Lower Threshold

Teachers who have taught both Logo and Boxer tell me that the first stages of introduction to Boxer invariably go more quickly and smoothly. One of my most reliable sources (and, at times, a friendly critic), who is a high school teacher and long-time Logo user, tells me that he can now get through his basic introduction to programming in more like two days compared to two weeks with Logo. This includes learning commands, procedures, iteration, and at least a little about variables.

A second measure of lower threshold is how long it takes for students to get into really interesting, self-directed projects. A benchmark for me in this regard also comes from the above-mentioned teacher. In a mathematics course that lasted only 10 weeks, he brought a not-particularly brilliant class from the state of being non-programmers to where they each produced both cogent and personally fulfilling projects. This is significantly beyond anything I have seen accomplished with Logo.²

```
to square
repeat 4 [fd 50 rt 90]
end
```

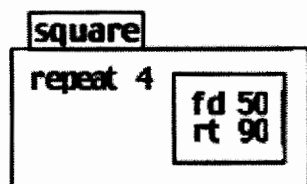


Figure 1. A square procedure definition in Logo (top) versus a visually transparent Boxer presentation (bottom).

This is not magic, but it results from some basic properties of Boxer. For example, procedures appear as visual entities, boxes. Consult Figure 1. Little simplifications, such as extremely clear visual boundaries, make a lot of difference to beginners. You don’t have to learn the special meanings for a little piece of text, **end**, as the boundary of an object. In Boxer, you can’t have a beginning (**to . . .**) without an **end**, and you can’t misspell syntactic boundary markers.

Logo has always, unfortunately, distinguished the mode of creating procedures from the mode of executing them in one way or another. Early in Logo’s development, you entered a special mode for procedure creation during which you couldn’t execute. Later, there was a separate editor, which became the “flip” side of the page, or the procedures page. All of these separations cause difficulties, especially for beginners. Most notably, you cannot easily—or at all!—see the effects of a procedure at the same time that you look at its form. This makes *learning by inspecting* difficult; you have to flip back and forth between different areas to see a procedure and its effects. In addition, it rules out a mode of *learning by interacting* with pieces of code, which is very powerful and characteristic of Boxer. For example, if you look at a line of code in Boxer and wonder what it will do, you can just double-click on that line, and it will be executed. This also turns out to be an extremely powerful debugging technique.³ If something goes wrong, you can just step through the process by executing one line at a time. That is, the inherent inspectability of Boxer is extended with “pokability.” Without easy visual interpretability, inspectability, and pokability, is it any wonder that Logo beginners frequently just throw away old procedures without re-using them (rather than figuring out what they do)? Or they simply start again with a new procedure definition instead of debugging what exists.

```
print :x
make "x 35
```

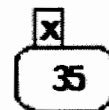


Figure 2. Contrasting Logo interactions with a variable via a conversational paradigm (left), versus direct visibility and access (right), characteristic of Boxer.

Variables are well-known to provide difficulties for Logo beginners. The problem is that Logo variables are very abstract. In Boxer, in contrast, variables are boxes just like procedures (although they are different kinds of boxes; procedures are *doit* boxes, and variables are *data* boxes). So, in Boxer you can literally see variables. Consult Figure 2. You can see their current values, and if a procedure changes a variable, you can see that change. Furthermore, if you want to change a variable, you can just edit its visual presentation as if it were plain text.

Logo never lets you see a variable—only its value—and you must interact with it “conversationally” rather than directly.⁴ To see its value, you must execute a command like **print :x**, and to change it, you must similarly execute a command. Actually, Boxer is extremely unusual in this regard, as—as far as I know—no other “serious” language provides an interactive notation (you can edit it directly!) for the fact of a variable having a value, as opposed to a notation for the fact of changing a value or the fact of doing something with it. Spreadsheets are another context that shows the power of concreteness (visibility) and the ability to change “variables” (cells) directly.

2. See for Yourself

The reasons that Boxer is better for beginners, such as those explained above, are not accidental or particular to the way we chose to design a notation for procedures and one for variables. Instead, these are part of a larger plan, which accounts for very many of the advantages that Boxer has over Logo

and other programming environments. In particular, there are two overarching principles that guided Boxer design. First, there is the principle of *naive realism* (or the principle of “concreteness”). Here, the idea is that the user of a computer system can pretend that what appears on the screen is the computer system. That is, you don’t need to do a lot of mental work interpreting an abstract presentation that relates only indirectly to the fact of the matter (as, for example, imagining something called a variable that is changed or accessed by commands). Instead, naive realism means that everything in the system must have a visual presentation that allows easy interaction with it, including creating it, changing it, moving it, and deleting it.

One of the wonderful successes of Logo was that it developed the turtle, whose spatial state was always visible for learners to contemplate. So, the practice of semi-programming was born. Students can execute commands one at a time and inspect and think about the state of the world thus created. This is a tremendous boon to beginners as it frees them from the need to imagine a complex state created in the midst of a complex process. But, because Logo has no principle of naive realism for computational objects, as opposed to for the turtle (or other graphical “side effects” of computation), semi-programming can’t work to support more abstract programming. In Logo, semi-programming can’t work where only a set of variables are changing. With Boxer, students can simply watch variables change just as they might watch a turtle. They learn “abstract” programming nearly as easily as they learn turtle programming. This is not just an advantage for beginners; it also helps experts watch their programs in action, and to debug them.

The second powerful and general principle of Boxer is the *spatial metaphor*. You can think of this, once again, as extending an excellent idea of Logo to apply more broadly. In this case, the Logo turtle allows students to use their

very well-developed spatial understanding to become engaged in programming. Every child intuitively understands certain spatial relationships. Children can instantly see if the turtle is in the correct place, facing in the appropriate direction. And they can reason through what they want to have happen next. But, the turtle is not computational structure, per se. Boxer uses space and spatial relations systematically to represent aspects of “abstract” computation. In particular, Boxer has a wonderfully transparent hierarchical structure of boxes inside of boxes that represents huge ranges of computational meanings.

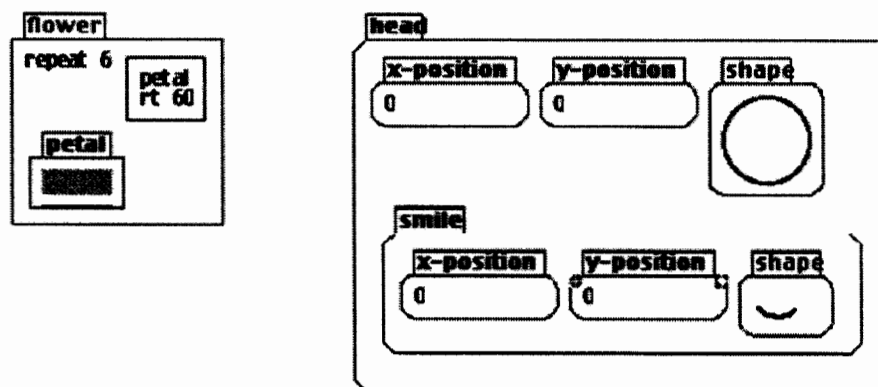


Figure 3. Boxer uses its natural hierarchical structure of boxes inside boxes to represent important computational semantics. On the left, a procedure, **flower**, contains its subprocedure, **petal**. On the right, a subsprite, **smile**, behaves as part of its supersprite, **head**, when the supersprite is moved, but it can also be independently addressed.

For example, procedures inside other procedures represent the “belonging” of a subprocedure to a procedure. See Figure 3. Similarly, the boxes that represent turtles (we call them sprites) may be nested one inside the other. A subsprite thus created moves with its supersprite when the supersprite moves, but moves independently within the frame of the supersprite when you direct commands to the subsprite. Subsprites make excellent components of objects, for example, arms or eyes of a person or animal that you want to have move with the complete animal. Yet they can also have independent actions, like raising an arm or winking an eye.

Boxer’s spatial hierarchy represents literally dozens of computational relationships in a way that we have found is very natural. Look for continuing examples in the sections below. The spatial metaphor has proven much more successful than even we initially believed. For example, we thought people might get lost in a maze of boxes inside other boxes. One of the first utilities that we designed before we tried Boxer out with people was a map utility that showed the structure of your universe and provided a “you are here” indication of your current location. As a matter of fact, this proved completely unnecessary. After just a

very little experience with Boxer, students never get lost. Better said, if they get lost, they understand how Boxer is organized well enough to find themselves without difficulties.

3. Higher Ceiling

In addition to a lower threshold, Boxer provides a higher ceiling than Logo. This is, in part, a difference of orientation. Logo was designed for children. But Boxer is designed to grow with children into adulthood. One of the disconcerting things I found with Logo was that teachers learned it only for their students. It didn’t really serve any of their adult purposes.

Simply put, students who stick with

Boxer get a lot further than those who continue with Logo. I have several benchmarks. First, in all of our experience with Boxer, essentially every student in courses that last more than a few weeks has managed to produce a cogent project at the end of that time. This was not my experience with Logo. In addition, exceptional students have substantially surpassed what appears possible with Logo. For example, in a Boxer class we gave, a pair of sixth grade students who were certainly clever—but clearly not prodigious—managed to create a huge “graphing adventure game.” The game presented players with dozens of graphs to interpret, kept score and had included help and a “reward” video game to play. This program contained hundreds of boxes and was larger than the average construction I program myself in Boxer. Exceptional high school students have created immensely complex programs. One example was a “molecular toolkit” that contained tools to analyze organic molecules, to display visual presentations of their structure (given only their chemical formula), to name them automatically, and so on (Ploger & Lay, 1992).

Some of these accomplishments, and the higher ceiling for Boxer generally, come about for completely obvious reasons. Boxer provides many more advanced facilities compared to Logo, including different styles of programming (see below), a much more flexible and reconfigurable environment, advanced structures (e.g., compound graphical objects, the subsprites described briefly above) and so on. I will describe some of these in more detail in subsequent sections. However, other reasons that Boxer has a higher ceiling are more subtle, although equally important. These reasons are what I wish to discuss in Boxer advantage No. 3.

I already said, but it bears emphasizing: The characteristics of Boxer, naive realism and spatial metaphor, which, in part, make it comprehensible to beginners, also help relative experts. Inspectability and pokability help one

understand and manage complex programs, as well as understand how simple programs are created. It’s a lot easier to inspect the state of your program by watching its variables than to try to imagine what is going on. And executing a little piece of a large program is such a powerful part of debugging that making that very easy—as it is in Boxer—pays huge dividends.

A problem that occurs with Logo is a set of plateaus that appear regularly with respect to structured programming. First, students hardly ever begin programming in a structured style on their own. Instead, they produce “spaghetti code” programs, if ever they create large ones. This is not a cognitive limitation or even bad instruction (at least, not entirely), but it is a case of the expressive environment not facilitating effective organization. Contrast the visually clear capability of Boxer to put local procedures and local variables directly inside a superprocedure. (Again, consult Figure 3.) In contrast, Logo subprocedures at best follow their superprocedure, and there is no automatic and evident visual connection. Complex procedures with many subprocedures tend to become a disorganized jumble, unless one takes great care and invents ways of associating who belongs to whom. Obviously, what goes for local procedures also goes for local variables—except, arguably, the situation is even worse.⁵ Again, you

must use a “conversational” technique of declaring a local variable, rather than just putting one where you want it. And if a subprocedure calls a local variable that is not in that subprocedure, you have to do a complex process of guessing and finding superprocedures that call your subprocedure to see which one contains the local variable. In Boxer, you can often just scan visually outward to find the superior box that contains the relevant variable.

We find that Boxer beginners often begin “accidentally” to structure code reasonably without instruction. For example, they find themselves wanting a subprocedure in the middle of writing the code for a superprocedure, so they just interrupt writing the main procedure and write the subprocedure right there, within the superprocedure. Accidentally doing the right thing is a great, facilitating effect of well-designed environments. Similarly, students automatically assume that different box environments are independent. (Boxes can contain entire environments, and it is not uncommon to have several such box-environments in your Boxer world at the same time.) That is, they expect the procedures and variables in one box-environment to work independently of others. This happens to work just fine, even before we teach students how variables “scope.”

Of course, accidental facilitations only go so far. To manage really large

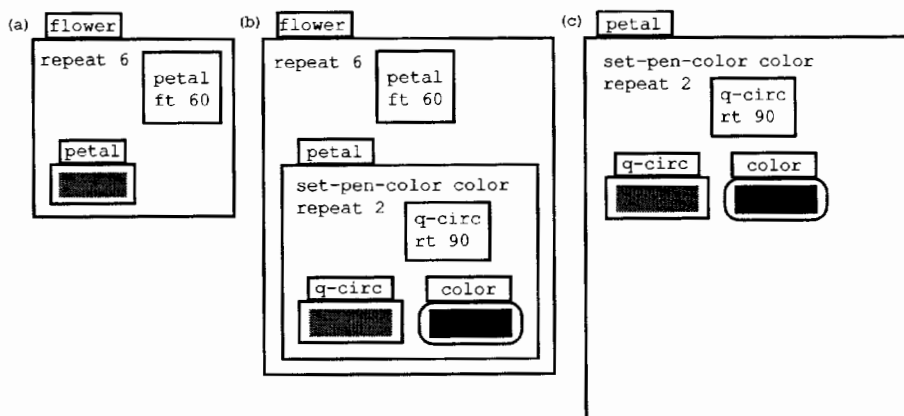


Figure 4. (a): A procedure contains a subprocedure, which is shrunken to hide its details. (b): Clicking on the subprocedure opens it to reveal its contents. (c): If the contents are complex, the user can click to expand the subprocedure to full-screen size. A click to shrink the box brings one back to (b).

and complex programs, you have to do some work. Boxer provides several resources for effectively managing complexity. In the first instance, boxes within boxes are a great organizing feature that is automatically provided and easy to understand. In order to further control complexity and facilitate viewing a complex environment, the visual presentation of boxes is easy to manipulate. Any box can be shrunk to a small, gray box, hiding its contents and making more of the surround visible and easy to see without distraction. Consult Figure 4. Or, if a box is complicated and the student wants to focus his or her attention right there, the box may be expanded to full-screen size, effectively hiding the context. It takes a little practice to use these capabilities effectively, but, in our experience, they are easy to understand, and not much direct instruction is necessary.

In addition to controlling complexity by adjusting the visual presentation, Boxer allows a wide variety of ways to distribute code into semi-independent, meaningful units. For example, since visible graphical objects (Logo turtles or Boxer sprites) have a directly inspectable and modifiable box form, you can put code for behaviors that belong to that sprite right inside him. If you want a turtle to dance, you can (and probably should) put the `dance` procedure right inside him. Then, if he dances funny, you know where to look. I will discuss other methods of meaningfully distributing code to make programming and debugging easier in some other sections.

The final method that I will mention by which Boxer facilitates the construction of large programs also comes back to the fact that it has such a useful, spatial presence. In particular, in order to join two programs, or even two complete environments, the first step is usually trivial. Simply cut and paste the pieces so that they appear together, in the same place. Then, you can gradually integrate the procedures of the two parts so that they work together fluidly. I first noticed this process in the

construction of the graphing adventure game by the two sixth-grade students, mentioned above. One of them had a complete and working "video game" in a box, and wanted to make it part of the graphing adventure game. The first step was trivial. Just cut the video game box and paste it into the middle of the graphing adventure box. Gradually, the two young programmers integrated the code so that, for example, you could not enter the video game box before you managed successfully to complete some number of graphing challenges. They also added code so that your score on the video game affected what happened on following graphing challenges.⁶

4. Structure, Structure, Everywhere

In the prior sections, I emphasized how the spatial metaphor and Boxer's principle of naive realism provide advantages for beginners and also for more advanced Boxer users. In this section, I want to show just a little bit of how these same features provide ease of

example, specifying in advance that you needed only a sequence of three parts in your list. Logo, essentially, made no changes in this structure.

Boxer, on the other hand, was driven by what is easy to see and manipulate with modern display technology. The basic data object is a box, containing arbitrary elements in a two-dimensional display, like a sheet of paper. The two-dimensionality of data boxes gives one tremendous flexibility in considering how you want to configure and think of your data. First, you can, of course, think of your data as a list, a sequence of items reading left to right, top to bottom. But you can display your list horizontally or vertically, or mixing the two by grouping a number of elements on each row. Or, you can consider your box of data to be an array, and use array indices to fetch or to change parts of it. Or, you can ask for elements of your box, or change them, one row or column at a time. Figure 5 shows some examples of different organizations of box data.

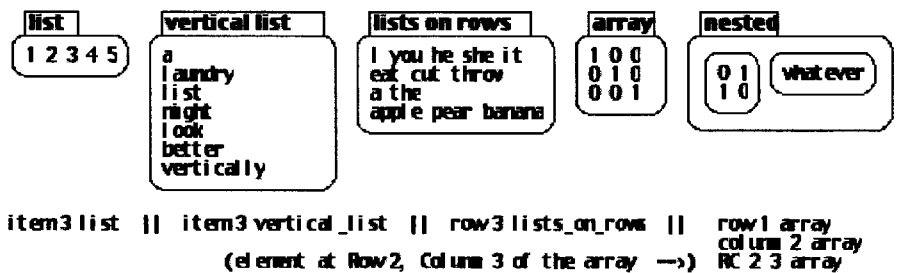


Figure 5. Some different arrangements of Boxer data and, below each, Boxer expressions selecting a part of the data.

construction of, and wide flexibility in, the data structures that one can develop in Boxer.

Logo's data structures are patterned on Lisp's, which in turn emerged from what is easy for computers to do. In particular, Lisp capitalized on the idea of a linked list, where each data item has a unique successor. The innovation of Lisp over other programming environments was that each element in the linked list could be an arbitrary object, for example, another list. And, one did not have to reserve a fixed amount of space associated with each object, for

Boxer allows much greater flexibility than Logo in terms of what kinds of things one can place inside data. In fact, Boxer places no restrictions whatever on what you can place in a data object. So, for example, Figure 6 is a record in a database that may easily be constructed just by making boxes, naming them, or cutting and pasting anything you can find in Boxer. The named subboxes can be addressed by name. For example, if this entry has been assigned to the variable `entry`, then `entry.last_name` provides access to the `last_name`. Notice also

that pictures (graphics boxes) can be contained in this compound data object, and also colors (appearing as colors, not as a name or numeric code). Finally, procedures are also first-class objects, and my favorite fractal procedure can, like anything else, be placed in a compound data object.

Let me give a couple of specific examples. The first relies on the fact that boxes saved as files can remain present in any Boxer environment. So, when you open a Boxer file, you might see a number of subfiles scattered through it. These subfiles initially appear as black boxes, and they

A final example shows how the flexibility you get when every aspect of your environment is computational can pay off in surprising ways. During an early attempt to create on-line (Boxer) documentation for Boxer, the project coordination went somewhat awry. Multiple people created multiple versions documenting the same command or structure, and different people used different formats for the units of documentation. In order to straighten this out, I just collected all the pieces of documentation and dropped them into a single box. Then I used a little search program that I had on hand to collect related documentation elements so that I could select the best, or cut and paste best features. As I used a documentation element, I just deleted it from the box-database. In a similar manner, when I finished the complete, hierarchically organized documentation of Boxer, I wrote a simple program to prune out all the details, leaving a nice hierarchical index. Try to do either of these things in any ordinary programming environment, including in Logo.⁸

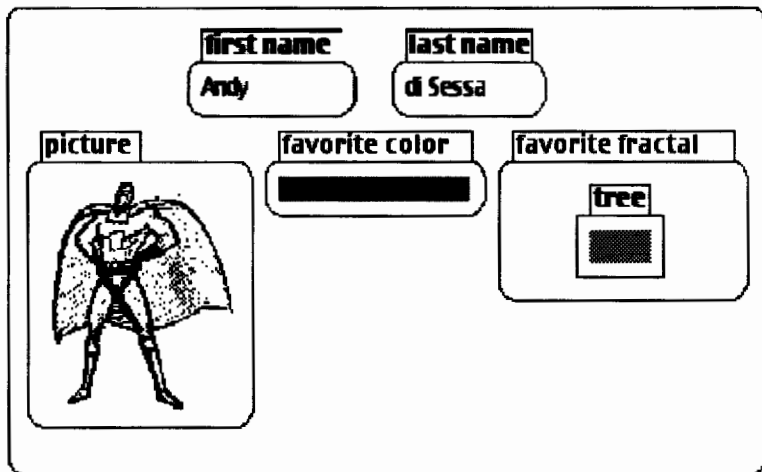


Figure 6. An entry in a database can contain any kind of Boxer data, including pictures and procedures.

5. A REAL Work Environment

A critical test for a life-long learning environment is not whether it is a nice place to visit, but whether you would like to live there. Boxer is designed to provide a flexible and practical work environment in which you can collect and integrate a set of tools and information to suit particular needs. Because Logo does not have the flexible structure of text anywhere, organized by boxes inside boxes, it lends itself more to a presentation environment for a single program. For example, in Boxer you can collect a number of box-tools (see reasons 8, 10 and 15, below) in the same place and surround them with working data. The tools can use and operate on surrounding data. You can write notes to yourself about what you want to do, which is, what I call emblematically, “scribbling on the desktop.” Indeed, because everything in Boxer is computational, you can write a little program to re-organize your “desktop” as easily as you could write a program to manipulate any other kind of data.

read themselves in whenever you click on them to open them as you would any closed (shrunken) box. You could type whatever annotation you want around these files, and thus create a well-documented “directory” structure. The point of this example is that by combining generic Boxer structure of boxes and text, you can create very many kinds of organizations to suit particular purposes—in this case, a personalized file organization.⁷

6. Build It Yourself

List processing is another area with which many people never achieve competence using Logo. This is a place where Boxer made a small innovation, which, nonetheless, has proved very valuable in eliminating a plateau in learning that was evident with Logo. Instead of a fairly complicated collection of ways of assembling and disas-

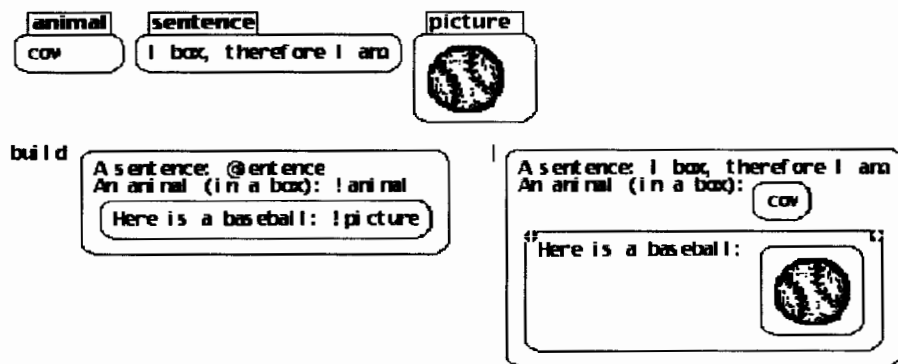


Figure 7. Top: A set of variables. Bottom: Build takes a spatial template and fills in parts marked with ! And @.

sembling compound data objects, **list**, **sentence**, **fput**, **lput**, etc., Boxer has essentially just one command. **Build** takes a spatially-organized template as input, and creates an output of exactly the same form, except that every part of the template preceded by an @ is replaced by the contents of the box that follows @, and, similarly, ! means to insert the full box referred to at that point. Consult Figure 7. Removing two levels of impossibility (lack of graphical data, lack of two-dimensional structuring), the equivalent Logo expression is still very difficult to produce. Try it!

Build is one of the areas that has had independent and convincing empirical study. A study by two European researchers (Schweiker & Muthig, 1986) showed that learners achieved competence with **build** about three times as quickly as the equivalent constructs in Logo. And, after achieving competence, subjects were about three times as fast to create fairly complex objects, and to debug faulty expressions. This demonstrates once again the power of visual, concrete structures, as opposed to invisible processes that you have to imagine.

7. Port Yourself to Infinity

Let's return to some parts of Boxer that were designed for more mature users, rather than for beginners. A *port* is quite similar to a regular box, except that its insides are identical to some other box, called the port's *target*. If you change either the port or its target in any way (editing or via a program), both are instantly changed. So a port

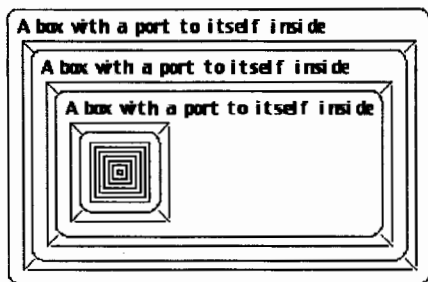


Figure 8. A box containing a port to itself is an infinite structure.

provides access to a box that might exist remotely from the port, say, deep inside a complex program.

Let me start with a little "parlor trick" one can perform in Boxer, which, nonetheless, suggests the power of ports. Suppose you have a box containing a port to the box, itself, that contains the port. What you see inside the port will be the box, which contains the port, which contains the This is an infinite structure (Figure 8) that can be created in Boxer with a few keystrokes and mouse clicks. You can even "climb down" far into the infinite structure, in case you believe it is just a pretty picture.

Ports make great user interface devices, in addition to their numerous programming uses. For example, as suggested above, you can provide easy access (both for viewing and for changing) to variables or programs whose natural place might be otherwise buried deeply within the box hierarchy. Ports also make excellent hypertext links. You can directly see and use (or change) things that don't exist locally. Boxer is a hypertext environment, the first hyperprogramming environment, a colleague once said. One of the built-in Boxer uses of ports is that error messages, when possible, include a port to the offending procedure. That means you can frequently correct errors like misspelling or missing inputs immediately, without flipping the page or wandering around to find the procedure's definition.

Ports implement data sharing in a natural way. For example, if you want to have a database where two people share the same phone number (and changing one should change both), then ports are the right thing.

Unsophisticated programmers can skip to the next section at this point. This is for relative experts: Here is a somewhat esoteric but extremely powerful use of ports. If you name a port to a procedure instead of the procedure itself, then you get the effects of "lexical scoping." That is, when you use the procedure by name, any variable names

used inside that procedure refer to variables accessible from the location of procedure definition, rather than from the place the procedure is executed. Lexical scoping in this way is a natural consequence of the meaning of ports (which I won't explain in detail). If you don't know about the great debates about the merits of lexical versus dynamic scoping (dynamic scoping is the usual way Boxer scopes, and the only way Logo scopes), suffice it to say that both have their advantages in different situations. In particular, lexical scoping is superior in that it always does the same thing, in contrast to dynamic scoping, which accesses one variable or another depending on where the procedure that contains the variable happens to be called.

Finally, ports implement "object access" in Boxer. Logo really doesn't have objects at all in that you cannot tell the difference between a data object and a copy of that data object. If you don't know much about these issues but wish to understand their importance, consult Abelson & Sussman (1985).

8. Beg, Borrow, or Steal

Boxer was not designed, particularly, to be a collaborative environment. But I have been surprised to see how much better collaboration has gone in Boxer, compared to my experience with Logo. During the several years Hal Abelson and I ran an NSF-sponsored summer program for bright high school students, one of the constant difficulties we had was that, typically, one student always took over the programming for a collaborative project. Other students became very dependent on the "programmer" of the group, since they couldn't even use the program very well without him.

Our experience with Boxer has been the opposite. Again and again we have seen students, even students of very different programming capabilities, working extremely well together. For example, we have "exit video tapes" of students from a summer program

course in Boxer where each student is interviewed about the project. Even quiet students produced excellent explanations of how the projects were programmed, and, frequently, they would stop in mid-stream to make a small edit or two to improve performance.

In retrospect, some of this improved collaboration comes from evident differences between Boxer and Logo—in fact, from differences I have already discussed. When a program is easily inspectable, pokable and changeable, it is much easier to share. Anything you miss in the construction can be made up by playing with the code. Reviewing video tapes of students collaborating reveals another very important mode of keeping everyone in sync. Boxer's very rich visual presence on the display screen means that it is easy for students to configure the screen in order to point and explain what a program does. "Look, this says to increase the variable *X* each time. Watch *X* while I execute that line. See, this procedure calls that one; here, let me open (expand) it."

9. Reconstructible Interface

This set of features is for more advanced users, or for teachers and developers to prepare easy-to-use environments in Boxer for less sophisticated students. Logo has innovated, just a little, with respect to allowing the user to customize and reconfigure its own user interface. For example, you can create buttons and sliders to begin activation of a program or control a variable. Boxer takes that a step further. Rather than providing a small set of canned elements, Boxer provides resources to *create* these sort of things. For example, you can redefine what it means to click the mouse button anywhere in Boxer. You can define what a mouse click means on a particular sprite, on a particular graphics box, in a particular box, or on all sprites and graphics boxes, etc. You can similarly define what keystrokes do—either everywhere, or in a particular place.

The mechanism for doing this is really very simple. Every user interface action executes a procedure of a certain name. Clicking the mouse in a box executes **mouse-click**. Clicking on a graphics box or on a sprite executes **mouse-click-on-graphics**, or **mouse-click-on-sprite**. If you define a new procedure by that name, then it will be executed instead of the default Boxer action within the box in which you define that new procedure (and also in all subordinate boxes that don't have their own versions of that command).

To make a box that serves as a button, you just define a **mouse-click** procedure in that box. Another simple feature of Boxer allows you to make your button "pretty." If you define a graphics box called **boxtop**, then when you shrink the box containing **boxtop**, it appears as the graphic contained in **boxtop**. So, a pretty button is nothing more than a shrunken box with a **mouse-click** procedure and a **boxtop** image inside it.

These resources make it easy to create many kinds of interface objects, in addition to buttons and sliders. One of the advantages is these are all easily inspectable (to learn how they work) and, of course, changeable to suit your particular purposes. Boxer comes with some sample objects, including buttons, sliders, pulldown menus and "clickers." Clickers, in fact, are one of my favorite interface objects. They look like (on) or like (off). If you click on them, they reverse their state. You can distribute clickers in the midst of code in order to allow easy turning on or off particular segments. I use clickers all the time as part of the user interface of micro-worlds and tools to turn various options on or off. Mixing code, data, and interface objects like clickers is a hallmark of Boxer and impossible in Logo.

10. First Class, Interactive Objects

The generalization of clickers turns out to be one of the most powerful kinds of objects in Boxer. You can make in-

teractive objects that respond to mouse clicks and other interface actions, and that have the following properties:

- a) They have all their "works" inside, so anyone can cut, copy and paste them anywhere work needs to be done.
- b) For the same reason, they can be opened to inspect them to see how they work, or to modify and extend them.
- c) They can be used as part of a program; just put one in the midst of code in an appropriate place.
- d) Similarly they can be used as inputs to procedures, or they can be created as outputs from procedures. The latter makes it easy to create procedures (which I call "factories") that create specialized interactive objects and return them directly to users to be cut and pasted where they are needed. For example, you can make a button factory that returns a fully functioning button to your specifications.⁹

Objects that can be placed in a data structure and can be used as inputs and outputs of procedures are called "first class" objects in the parlance of computer science. Boxer is almost alone among computer languages in allowing first class objects that are both graphical and interactive, in addition to fully functional in the language. Logo has no means to redefine interaction, and graphical objects can't exist in data or as inputs and outputs of procedures.

Figure 9 shows two interesting interactive Boxer objects. The first is a vector that you can control by dragging the arrow tip of its graphical presentation around. On the inside of the vector are its coordinates, which you can set by hand or by program. Vectors can be added, as in **add vector1 vector2**, which returns the sum vector. (If you don't understand vectors, for now just realize they are powerful quantities representing things like force, velocity, and acceleration in phys-

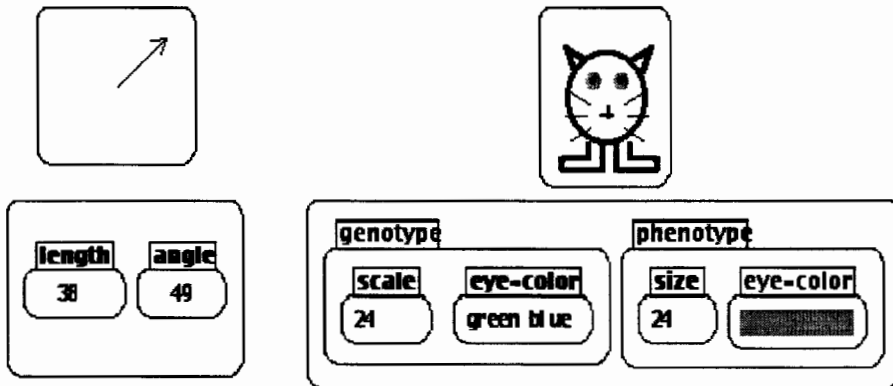


Figure 9. A vector and its flip side (left); a "scat" and its flip side (right).

ics.) In addition, you can command sprites to move with the speed represented by a vector, and simultaneously you can change the vector to see how that affects the motion of the sprite. One of the nice things about Boxer vectors is that they are so useful and so much fun in creating and controlling motion that children spend a lot of time programming with them. This leads to a lot of learning of things that are usually considered "advanced." Students in one sixth-grade experimental class we ran learned about vectors and motion in this way (diSessa, 1995b).

The second graphical, interactive object in Figure 9 is a creature called a scat. The insides of the scat include a representation of its genotype along with a computational version of its phenotype. Students can play with changing the genetic characteristics of the scat, and teachers or students can develop simple programs to experiment with breeding scats.

11. Can Your Turtle Do This?

Boxer turtles (sprites) have a number of advantages over Logo turtles, some of which are already apparent, above.

- a) In addition to all the usual attributes—shape, x and y coordinates, heading, pen-width and pen-color—Boxer turtles have a couple of other handy attributes. These include an overall size parameter and a home position where the sprite goes when you clear the graphics box in which the sprite resides.

- b) All those attributes are directly inspectable if you "flip" the graphics box to see its "logical" rather than graphical side. This is just the principle of naive realism. If there is a computational structure, you should be able to see and modify it. Of course, you can name sprites in the same way you name all boxes.
- c) You can put as many sprites as you want in a graphics box. You can even have a program add new sprites.
- d) You can easily add new procedures or attributes to any sprite or collection of sprites. In Logo, there is no logical (as opposed to graphical) representation of turtles, so this is impossible or, at minimum, awkward.
- e) Sprites are sensitive to mouse clicks, as explained above, so you can define their behavior when clicked.

12. Skeletons in the Closet?

Because Boxer makes things so visible and present, we have had to be somewhat inventive about allowing people to put things out of sight when they don't want to see them. One of the chief ways of doing this is with closets, which are part of every box in Boxer. Closets may be opened or closed at will. For example, when you look at a sprite (in its logical presentation), you see usually only the most-used attributes. But in the closet of the sprite, you can find all the other attributes. Similarly, if you look back at vectors and scats, you see only the most necessary parts. The rest is accessible in their closets.

A closet is an excellent place to hide the works of a microworld that users will ordinarily not need to see or change. Closets are a good place also to hide things like boxtops and key—and mouse—redefinitions for particular boxes.

13. Object-oriented Programming

Boxer allows other paradigms of programming, in addition to the usual procedural paradigm supported by Logo. This and the next section very briefly treat two other paradigms.

In Logo, you can **ask** a turtle to do something. But, it's awkward, at best, to teach a particular turtle new tricks, and very difficult to add new attributes to it. Most distressing, turtles are about

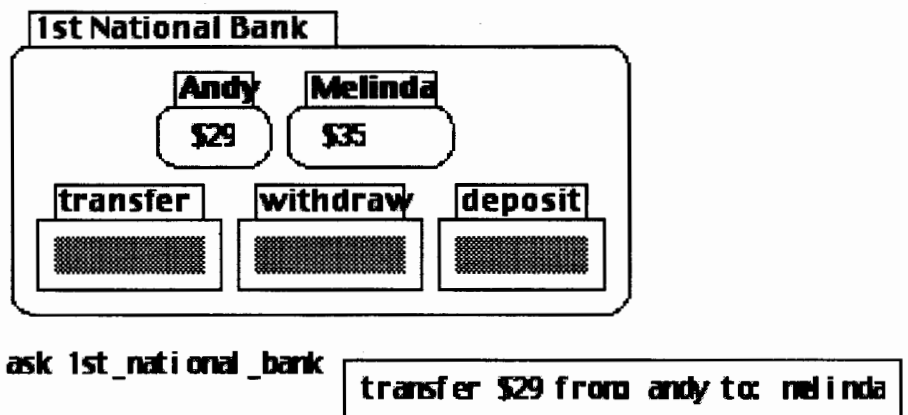


Figure 10. A bank has local data, knows how to do things like transfer, and can be asked to perform these functions.

the only thing you can **ask**. In Boxer, you can ask any data box to do things, and, of course, you can fill that box with whatever local data and procedures you find convenient. Thus, Boxer gets most of the important features of object-oriented languages like Smalltalk and Object Logo, but in a very concrete, spatial-visible way.¹⁰ Object-oriented programming, using objects and messages, has a number of advantages over plain procedural programming. First, it is a better, more modular organization of data and procedures to have meaningful chunks of them grouped together. It leads to systems that are easier to understand, and systems that are easy to extend, even if you don't understand everything about them. As important, objects are the natural way to think about and model many physical situations. Creatures running around a graphical display (sprites!) are one notable example, but there are many others.

Figure 10 shows part of a bank object. Banks, of course, know how to do things like deposit, withdraw, and transfer among accounts.

14. Activation-oriented Programming

Having a program automatically executed on certain conditions is a very useful way to program. You don't have to explicitly write the action and its conditions into other code that may have nothing much to do with it. For example, you may want to update some display any time a particular variable is changed. The display may just be a graphical presentation of the variable, say, a thermometer that shows a **temperature** variable. You don't want to have to put the graphic-changing procedure in every possible place where the temperature variable may be changed. Another example of activation orientation is a spreadsheet. If you change a cell, the recompute function is automatically executed by the very fact of that change.

Boxer has several activation triggers. You can set a trigger to execute when

any box is changed by the user directly, or by a program. Similarly, you can set triggers to execute whenever you enter or leave a particular box. Activation-oriented programming is sometimes tricky because you can easily generate surprising and unintended chains of triggering. That's the negative side of the simplicity of saying "anytime this happens, also do something else." But, sometimes it is exactly the right thing, for example: with a spreadsheet-like uses of boxes; to keep logical and graphical representations in sync; or to initiate actions, say, starting a microworld box program executing when you enter it.¹¹

15. Tool Building and Sharing

Building on reasons 8 and 10, Boxer makes an excellent environment in which a community can build a flexible set of tools to share with one another. As developers, my group of graduate students and I have experienced the kind of tool sharing that I never experienced with Logo. One of the seemingly little—but, after the fact, important—features of Boxer is that tools may exist just as a single box, with all the "works" inside to inspect, modify and extend. So, to start, you can just copy the box and use it directly; then, as you become familiar with it, you can open it up to modify and extend. One of the best kept secrets about tool-sharing in electronic environments is that nobody ever wants to use exactly the same tool as anybody else. If it's not inspectable and modifiable, it isn't much good.

Vectors turned out to be a marvelous general resource for our group when we designed several editions of a physics course. First, it is completely trivial to write simple tutorials that show how vectors work using working vectors! Second, vectors are great tools to build other tools and exercise microworlds. One of many tools we built in very short order with vectors was an analysis tool where students analyzed scanned stroboscopic images of balls flying through the air to find out

things like whether their horizontal speed decreased (as most students expect) and how the vertical speed behaves. I already mentioned how vectors served as a tool for student projects.

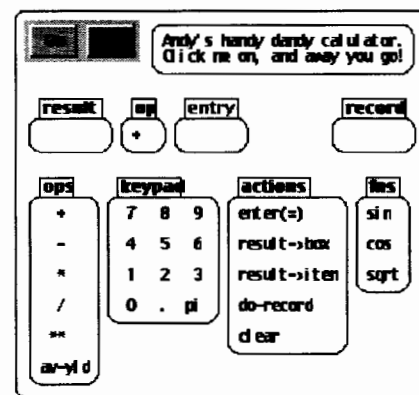


Figure 11. A simple Boxer tool—a calculator. See text for description.

Figure 11 shows a more familiar tool. It is just a little calculator written in Boxer. However, unlike a regular calculator (or a computer calculator!) it has the following Boxer-like characteristics.

- It is visible. You can see the "internal" registers for both numbers (**result**, **entry**) and for the operation (**op**).
- It is easily modifiable. You can just edit the **keypad**; it's just text. And you can easily add any functions (to **ops** and **fns**) that happen to be useful to you.
- It has "permeable boundaries." You can click on numbers outside the calculator, as well using the **keypad** to enter them. And, you can "export" numbers to the surrounding Boxer environment (the **result->box**, etc., commands). There is no need to worry about limits in the number of registers!
- It keeps a runnable and editable program (**record**) as a written history of the actions you perform.
- It is "scriptable." That is, you can ask it to perform any of its available operations. (Well, maybe a

scriptable calculator is computational overkill. But, to have a graphing tool that you can call upon to draw a graph and return it to you is a better example of the power of using Boxer tools as objects to **ask**.)

More details about Boxer tools and toolsets appear in diSessa (1997).

16. Completely Integrated Mail and Network Browsing

You can send and receive mail from within Boxer. You can transparently send and receive not only text, but any Boxer object, such as a microworld. Consider how nice it would be to sort mail in various boxes (with ports in one place to all your important mail), or to be able to write a simple program to sort mail for you. Non-Boxer file attachments or Web URLs appear as

icons inside Boxer that you can double-click on to open.

Boxer also has net boxes. These are almost exactly like the file boxes described in reason 5. That is, they appear as black boxes, or as their iconic **boxtop**, if one is defined. Then, when you click on them or use them in any other way, their "insides" are read in over the network. Net boxes, like any Boxer objects, have full integration with the rest of the Boxer environment. This means:

- a) Boxer's familiar and easy-to-use browsing capabilities also browse net boxes. The hierarchical structure means it is easier to keep track of where you are.
- b) You can put net boxes wherever you wish in the Boxer environment. Your whole world is your browser, and you can even write

programs to manage your net boxes, if you like.

- c) Anything you make can be a net box or appear in a net box—programs, tools, complete environments. Put a trigger in a box if you want a moving graphic. (Java "applets" are a great idea—unless you want to program them.) There is no conversion necessary to put something up on the Boxer Web.

17. New Social Modes of Materials Development


This section is unlike others in that it is not a demonstrated superiority of Boxer over Logo. Instead, it represents a hope. It represents a hope that the cumulative effect of multiple advancements may change the very presumptions about who makes what for whom in educational computing.

A note from Andy diSessa, creator of Boxer

This is an exciting time for Boxer. Technologically, we have a full-featured Boxer implementation that is freely available, downloadable from the Web. In a few months, we expect to have a Windows Boxer to complement the existing Macintosh version so that many more people can join in. After years of working in laboratories with expensive machines, it is now true (and has been for a few years) that any new computer has much more than sufficient power to run Boxer well.

Educationally, we are beginning to capitalize broadly on what we've learned in laboratory studies. Our early work on teaching motion to elementary school children is being replicated at sites that do not have extraordinary computer resources, nor a daily presence of the Boxer research team, nor even experienced "leading edge" teachers. In Florida, whole school districts are adopting Boxer-based elementary mathematics curriculum. What's powering this last effort is teacher creativity and innovation—given a suitably powerful and flexible framework consisting of the right tools in an open, changeable environment. The role of teachers in the emerging Boxer culture is most gratifying.

I am just putting the finishing touches on a book about Boxer and the educational goals in which it is steeped. Look for *Changing Minds: Computers, Learning and Literacy* late this year. I hope this book can do for Boxer what *Mindstorms* did for Logo—explain to a broad range of practitioners and researchers what we have wanted to do and why.

The article published here, "Twenty reasons," is really an invitation and challenge to the Logo community to understand why we felt it necessary to abandon Logo technologically, and what advantages we gained from doing that. The article is, perhaps, a little confrontational—"why Boxer is better than Logo." But it is written, I hope, in a community spirit, assuming common goals for future learning. Please join us to explore a "new" computer software system that supports the intelligence and creativity of teachers, students . . . and the rest of us. 

Resources

<http://www.soe.berkeley.edu/boxer/>

- Sample chapters from *Changing Minds*.
- Other papers, references and resources.
- A new release of Mac Boxer will appear in late May.
- Windows Boxer, mid to late 1999.

diSessa, A. A. (in press). *Changing Minds: Computers, Learning and Literacy*. Cambridge, MA: MIT Press.

Building particularly on 15 and 16 (which, in turn, build on other Boxer properties), I have gotten enthusiastic recently about opening the development of educational software more seriously to teachers. In particular, I would like to engage a community of likeminded people, including teachers and students, in the construction of physics materials for learning. Unlike almost all prior software, this will be a flexible toolset that supports a wide range of ways to use it. Of course, the toolset will come with pre-made materials and activities, but the fundamental idea is that it is always open to innovation and change. So many times in our experience, the real brilliance of a tool was in a little change a teacher made to suit her or her students, or a new idea about how to use an old, familiar tool. Logo might have accomplished this, but it made it too hard for teachers to learn to use and modify tools. It makes tools that are brittle and isolated, hard to combine. Boxer's inspectability, pokability, and management of complexity may just cross a threshold that allows, not only open materials, but an open process of creating and experimenting with tools. It takes a long time and a lot of experience to create a good tool or toolset. As a developer, I need serious help.

I imagine starting a smallish collaborative of folks using, commenting on, and modifying some of the tools we have built ourselves to teach physics. A Boxer activity database should evolve. Imagine that each tool has built-in net-box links to the activity database, and to the current discussions of core collaborators. I happen to think a book, or several, will also be necessary to support practical use.

Can this happen? Can it help professionalize, empower and re-energize teachers? Will we really be able to muster the effort to build curriculum that is simultaneously effective by any measure, usable by real teachers, and also true to the open learning principles that inspired Logo? Ask me in a couple of years.

18. But What About . . . ?

This section has some brief comments on some special things Logo (or MicroWorlds) has that you may think you can't live without:

- a) a draw program: Boxer provides all the necessary "hooks" to write your own draw program. There are two interesting, if very simple examples distributed with current Boxer. As the Boxer community expands-it has barely begun to mature-you can expect very many general tools like this to be produced and to become available.
- b) buttons and sliders: Again, check the Boxer demonstration files. You may find you can't live without Boxer clickers, pulldown menus and the many other kinds of user interface objects that are constructible in Boxer.
- c) QuickTime movies: Audio-video boxes should be coming soon, if they are not already available. Upgrade your old Boxer.
- d) parallel processing: In many instances, it is better to retain control of parallel processes in order to make sure things run in synchrony. For example, we frequently write programs with multiple sprites (or other objects) that are all driven by a **tick** messages sent from a single controller. Boxer has its own parallel processing system, but we have not found it useful enough to debug it thoroughly. If you need thousands of turtles and just can't do with only scores, stay with StarLogo for now.

Are there any things I envy about current Logos? Sure. They are pretty, slick and require less memory than Boxer. These are things that only commercial efforts can manage.

19. Upgrade Your Skills

It's fairly easy to start doing Boxer if you know Logo. We have been careful not to change things gratuitously just to be

different. But be warned: It may be easy to fall into the trap of thinking Boxer is mostly just like Logo, and to use it to program just like Logo. As many of the items and examples above should have demonstrated, that would be a serious under-use of Boxer's power. Constructs like ports, object-activation-oriented programming, etc., can make things that are difficult in Logo much easier.


20. It's Free!

See:

www.soe.berkeley.edu/boxer

or send mail to boxer-inquiry@soe.berkeley.edu.

Conclusion

Logo began with a grand image of the computer transforming learning from an often painful, alienating and awkward process into a more natural-feeling and empowering one. But, I believe Logo tripped by not realizing its transitional nature-born of teletypes and printers. Instead, it pursued glitz and contemporary-looking features rather than changing its infrastructure. This has left users able to do nice-looking things quickly, but without the kind of deep penetration into learning cultures that we need. With Boxer, we started from scratch designing an environment that uses display technology to make things easier to do and easier to understand. This paper invites you to explore new possibilities. I have tried to explain what I believe to be advances of Boxer, and why these could make a big difference. 

References

- Abelson, H. & Sussman, J. (1985). *Structure and Interpretation of Computer Programs*. Cambridge, MA: MIT Press.
- diSessa, A. A. (1995a). Collaborating via Boxer. In L. Burton and B. Jaworski (Eds.), *Technology—A Bridge between Teaching and Learning Mathematics*. Bromley, Kent, UK: Chartwell-Bratt, 69-94.
- diSessa, A. A. (1995b). The many faces of a computational medium: Learning the mathematics of motion. In A. diSessa, C. Hoyle, R. Noss, & L. Edwards

- (Eds.), *Computers and Exploratory Learning*. Berlin: Springer-Verlag.
- diSessa, A. A. (1997). Open toolsets: New ends and new means in learning mathematics and science with computers. In E. Pehkonen (Ed.), *Proceedings of the 21st Conference of the International Group for the Psychology of Mathematics Education*, Vol. 1. Lahti, Finland, 47-62.
- Ploger, D., & Lay, Ed. (1992). The structure of programs and molecules. *Journal of Educational Computing Research*, 8(3), 347-364.
- Schweiker, H., & Muthig, K. (1986). Solving interpolation problems in Logo and Boxer. In P. Gorny & M. J. Tauber (Eds.), *Visual Aids in Programming*. Heidelberg: Springer-Verlag.

Endnotes

¹Actually, there is at least one interesting exception. The invention of "pages" and "flip sides" are directly along the lines of some Boxer innovations that I will be discussing. However, pages and flip sides didn't penetrate the language, even if they are an excellent innovation in the environment. It may be surprising, but Boxer's version of these innovations already existed when Logo "invented" them.

²I am describing the "infinity" class, whose work and final projects are distributed with the demonstration programs available with Boxer. Go see for yourself if you believe you could accomplish the same amount starting with non-programming students!

³Of course, executing a piece of code that depends on a context created by the execution of other code makes debugging simply by pointing to an arbitrary line of code sometimes difficult. However, Boxer makes some of these cases easier to deal with in that if you execute an input line, you get local variables in which you can insert test values.

⁴Yes, sliders sometimes provide some of Boxer's inherent concreteness and visibility. But "sometimes" is a long way from "always." Consider how often one wants non-numerical data, and how often a slider is just more work than having a variable.

⁵Here is a good little research project. How many Logo users even know about

local variables? In Boxer, people begin using them even without instruction because the idea of putting a variable where it belongs is so obvious.

⁶More on this graphing game and how the students managed to accomplish such a complex programming feat appears in diSessa (1995a).

⁷A slight modification of this "directory" idea makes for a very useful organization for teachers. One can make a file box with an "auto read" property, so that it is read in automatically when another a box containing it is read in. So, a teacher can put a file box containing a set of tools inside each student's Boxer world. Whenever students read their worlds, they get the teacher's most recent set of tools. (When a box containing a file box is saved, the subfile box is not itself saved-only the "pointer" to the file.)

⁸The result of this work is, in fact, the online documentation of Boxer we supply with the current release. You can look to see how useful the hierarchical index is, and recall that it was generated from the full documentation by a simple program (which appears elsewhere in the set of Boxer demonstrations).

⁹See the button factory in the Boxer demonstrations included with the current Boxer release.

¹⁰When I first programmed in Smalltalk, I was terribly distressed to find out that objects, including graphical objects, don't "live" in a particular place the way they do in Boxer. But, Smalltalk didn't have either the principle of naive realism or the spatial metaphor to make its objects easily comprehensible.

¹¹Sprite attributes work in this precise way! In the closet of each sprite attribute there is a modified-trigger that changes the visual presentation of the sprite when you change that attribute by editing it, or under program control.

Andrea A. diSessa
Graduate School of Education
University of California
Berkeley, CA 94720
disessa@soe.berkeley.edu
www.soe.berkeley.edu/boxer

Letters to the Editor

Dear Gary,

I am writing to you as representative of all the people who contributed articles about me and my work to say how touched I was. I was also stimulated to do a lot of new thinking and will express some of it in your columns very soon.

With warmest collegial greetings,
 Seymour Papert

Dear Editor:

I read with interest your piece in *Logo Exchange* entitled "Never Satisfied, Only Gratified" and agree with almost all of it. I was disappointed, however, when you failed to mention our book, along with that of Druin and Solomon, as books that DO NOT ignore Logo and research on Logo.

Our book published by Allyn and Bacon is in its second edition (third in preparation), is entitled *Educational Computing: Learning with Tomorrow's Technologies* (Maddux, Johnson, & Willis) and contains many references to Logo, including an entire chapter (chapter 15) entitled "Logo: A Unique Computer Language."

Cleborne D. Maddux, Ph.D.
 Professor, Counseling and Educational Psychology, University of Nevada, Reno



Simulating Artificial Life with Logo

by **BILL ENGEL AND PAT GREENE**

Background

John Conway developed one of the first artificial life activities in the sixties. Conway's *Game of Life* simulates the birth and death of organisms based on certain rules. The simulation takes place on a grid and has been implemented with computer software.

Computer strategy games use artificial intelligence to simulate computer players. Computer programmers must develop artificial players that make intelligent moves. Sometimes these intelligent moves involve complex movements such as a dogfight with an airplane.

More recently, a number of virtual pet software programs have been developed. One can raise dogs, cats, or pigs that move around on the computer screen. One must feed and care for the pet, or the pet will die.

Virtual pets have moved from the desktop computer to stand alone devices. One branch of computer science involves the development of computer programs that simulate life. Computer software, based on A-life, has been developed to simulate such activities as bacteria growth, bird flights, and bee swarms.

In this article, a logo software program is developed to simulate a turtle that wanders around on the computer screen. This activity could be used by a mathematics or computer science teacher in grades 6 through 12. The teacher could use the suggestions in this article to help students develop their own artificial life program.

Why Logo?

Logo is an ideal programming language for constructing artificial life. In fact, Logo was developed by Papert, who was involved with the Artificial Intelligence Laboratory at MIT. The turtle graphics commands in Logo serve as a foundation for the development of an artificial turtle that can move around on the screen like a real turtle. Logo programming encourages the modular development of procedures. In this article small procedures are developed first that can be tested independently. The program can be expanded easily by adding new procedures.

There are many different versions of Logo. The program described in this article was developed with LCSl's MicroWorlds Logo. Most versions of Logo are fairly standard, and the pro-

gram can be transported from one version to another with only small modifications. It is assumed that the teacher and students have some familiarity with Logo programming.

The Goal

The goal is to develop a Logo program that simulates the movement of a turtle on the computer screen. We want the turtle to perform a variety of tasks. Some of the more obvious tasks are to move, turn, sleep, and eat. One needs to determine what variables might effect the turtle such as age, energy level, and location. In order to keep the initial simulation simple, interactions with the human user will not take place in this implementation. Food, in the form of green trees will be randomly distributed before the turtle starts moving. If the turtle is on top of a green tree and is hungry then the turtle will take a bite out of the tree.

Initial Conditions

At the beginning of any computer program, one must set constants and variables to initial values. It may be necessary to clear the screen and

command center. The start procedure for this program sets some constants and variables, distributes food as green trees around the screen, and establishes a repeat loop that loops for the age of the turtle. The start procedure can not be tested alone, since it calls the **food** and **decide** procedures that are not developed at the beginning of the program. If one deletes the two lines that call **food** and **decide**, then one can run the start procedure to determine if there are any errors. One can put in some print statements to determine if the variables are set to the proper values. For example: **print :energy**.

```
to start
cc ct cg
make `maxenergy 100
maxenergy, maxage, and
maxfood are constants.
make `maxage 200
make `energy :maxenergy ;Set
variables to initial values.
make `age 0
food :maxfood ;Go to food
procedure with :maxfood input.
setc 12 home ;Set initial
conditions of turtle.
repeat :maxage [decide
recycle]
print "Died
end
```

Recycle is a primitive logo command in LogoWriter that clears the stacks.

Distribute Food

In order to distribute food randomly around the screen, a **rand** procedure is used to select a random number between a lower and upper limit. The Logo random command produces a number from 0 up to the parameter. For example, **random 5** produces the numbers from 0 to 4 inclusive. The **rand** procedure requires two input parameters.

```
to rand :x :y
output :x + random (:y -
:x + 1)
end
```

The **rand** procedure is tested by typing **print rand 5 10** in the command center. A random number between 5 and 10 inclusive should be printed on the screen. The development of this procedure illustrates one of the powerful capabilities of Logo. It is possible to define new procedures to add to the Logo language.

One must determine the dimensions of the Logo screen. One can move the turtle to the limits and then type **print xcor** or **ycor** in the command center to determine the values of the edge of the screen. In the case of LogoWriter, the left is -150, the right is 150, the top is 70, and the bottom is -70.

The shape of the tree has a value of 8, and the green color has a value of 7. The input parameter represents the number of trees that are placed on the screen.

```
to food :x
setsh 8 setc 7 ;Set shape to
tree and color to green.
repeat :x [setx rand -150 150
sety rand -70 70 pd stamp pu]
setsh 0 ;Set shape back to
turtle.
end
```

Type **food 20** in the command center to test the food procedure. Twenty trees should appear on the screen in a random pattern.

Turtle Tasks

Most of the time, in software development, it is easier to start with small tasks, check them out, and then develop a main procedure that calls the individual tasks. The turtle simulation lends itself to this "bottom up" programming style. There are four main tasks the turtle will perform: move, turn, sleep, and eat.

In order to make the turtle move forward, the **forward 40** command could be used, but the turtle would move too fast. In order to make the turtle appear to move slowly, it is necessary to establish a repeat loop. The animation will appear much more realistic.

Note that the **rand** procedure that was developed for the food procedure is used as input for the **move repeat** loop. In this case, the turtle will move randomly between 10 and 30 steps. The **check** procedure is developed later to determine if the turtle has moved off the screen or is on top of food. The **energychange** procedure will decrease the energy level of the turtle by .5. The wait time can be adjusted to create a more realistic moving turtle.

```
to move
repeat rand 10 30 [check fd 1
energychange -.5 wait 1]
end
```

In order to test the **move** procedure, one should remove the **check** and **energychange** procedures and type **move** in the command center. The turtle should move forward a random amount each time. Since the **check** procedure is not in use, the turtle will wrap around the screen.

In the **turn** procedure, one can use the fact that a right turn of -30 is really a left turn of 30. Just like in the **move** procedure, it is necessary to use a repeat loop to animate the turn. Note that the **rand** procedure is use in two different places in the **turn** procedure.

```
to turn
make `t rand -2 2 ;The
variable t is assigned -2 to 2.
repeat rand 1 90 [check rt
:t energychange -.2 wait 1]
end
```

One can test the **turn** procedure by removing the **check** and **energychange** procedures and typing **turn** in the command center. Note if **t** happens to be 0, the turtle will not turn.

The **sleep** procedure is really just a random wait along with an increase in energy.

```
to sleep
repeat rand 5 10 [check
energychange 5 wait 2]
end
```

The `eat` procedure produces a sound, stamps out the green color to simulate eating, and changes the energy level to a maximum. There is no reason to access the `check` procedure from `eat`, since the turtle does not move or use energy to eat.

```
to eat
tone 800 2
setc 0 pd stamp pu
make "energy :maxenergy
end
```

Energy Change

The `energychange` procedure has one parameter input that provides a means to increase or decrease the energy level. The procedure does not allow the energy level to increase above the maximum. If the energy level drops below zero, the turtle dies. In addition, the `energychange` procedure changes the color of the turtle depending on the amount of energy.

```
to energychange :x
make "energy :energy + :x
if :energy > :maxenergy [make
"energy :maxenergy]
if :energy < 0 [print "Dies
stopall]
if :energy < 25 [setc 8 stop]
;Stop exits this procedure only.
if :energy < 50 [setc 9 stop]
if :energy < 75 [setc 6 stop]
if :energy < 90 [setc 4 stop]
setc 12
end
```

One can test the `energychange` procedure by first typing `make "energy 44` in command center then

`energychange 5`. Now type `print :energy` and see if the `energychange` procedure works. The turtle should also change to the appropriate color.

Check Procedure

The `check` procedure is used to change the turtles heading if it moves off the computer screen. The `check` procedure determines if the turtle is on food and needs energy.

```
to check
if and (colorunder > 0)
(:energy < .9 * :maxenergy)
[eat]
if xcor < -150 [seth 90]
if xcor > 150 [seth 270]
if ycor < -70 [seth 0]
if ycor > 70 [seth 180]
end
```

Decide Procedure

The `decide` procedure is the main procedure that calls all of the other procedures that have been developed. A random number between 1 and 3 is selected. The turtle will either move, turn or sleep, depending on the selection.

```
to decide
make "choice rand 1 3
if :choice = 1 [move]
if :choice = 1 [turn]
if :choice = 1 [sleep]
end
```

Conclusion

When one types start in the command center, the trees should appear, and the turtle should start moving. One of the major parts of a virtual pet that is missing in this simulation is real time interaction. Instead of having all of the

trees appear at the beginning, the user could provide food by pressing the space bar on the computer. The turtles speed could be connected to how much attention the user paid to the turtle. With a modern version of Logo like *MicroWorlds*, one could develop a number of buttons and sliders to interact with the turtle in a number of different ways.



About the Authors

Dr. Bill Engel is Professor of Education at Florida Gulf Coast University. He is author of several educational software packages and founder of the Florida Center for Instructional Technology. Dr. Engel established graduate programs in Instructional Technology at the University of South Florida and is currently involved in the development of graduate programs in mathematics education at FGCU.

Patrick Greene is an Assistant Professor of Educational Technology at Florida Gulf Coast University. His interests lie in infusing technology into the K-12 curriculum. He has written several articles detailing the use of project based learning methods and student centered education. He has also presented guidelines to improve the technological training of preservice teachers.

Patrick J. Greene
Asst. Professor of Ed Tech.,
School of Education
Florida Gulf Coast University
941.590.7802
941.590.7770 (fax)
pgreene@fgcu.edu



IN THEIR OWN WORDS

Using Spezeski's Poly.Round Procedure

by JOHN HAYES

As I experimented with Spezeski's (1999) poly.round procedure by trying out different values for the sides, length, and radius variables, the teacher in me quickly saw the value for children in such experimenting. I recognised that the microworld format was ideal for facilitating this. I hope that what I have developed will be useful for teachers.

What is a microworld? It is not to be confused with the Logo software of the same name. It is a vehicle recommend by McMillan (1989, 1992) and Yelland (1992-1993, 1995) for teaching mathematics concepts through Logo to young students. According to McMillan (1989) a microworld

- presents a concept or powerful idea to be explored;
- provides a comfortable entry to the concept at the learner's level of understanding;
- motivates the student's learning by focusing on what is inherently interesting to observe and interact with;
- provides an environment for active interaction; and
- has a product or outcome.

The Rounded Polygons microworld meets McMillan's criteria. Besides its obvious application to the study of polygons, it provides a very good setting for experimenting with variables, providing three, and as such it has applicability to algebra and the scientific concepts of variables in experiments.

The microworld is built on Spezeski's set, arcr, and polyr procedures. For the reader's convenience, these are repeated below (arcr is from Spezeski (1996, p.100).

```
TO SET :X :Y :Z
  PU SETXY LIST :X :Y PD
  SETH :Z
```

END

```
TO ARCR :RADIUS :DEGREES
  LOCAL "STEP LOCAL "REM
  MAKE "STEP 2 * :RADIUS * 3.14 / 36
  MAKE "REM REMAINDER :DEGREES 10
  REPEAT :DEGREES / 10 [RT 5 FD :STEP RT 5]
  IF :REM > 0 [FD :STEP * :REM / 10 RT
```

```
:REM]
```

END

```
TO POLYR :N :L :R
  LOCAL "ADJ
```

```
MAKE "ADJ :R * TAN 180 / :N
PD FD :ADJ PD
REPEAT :N [FD :L - 2 * :ADJ ARCR :R 360 / :N]
PU BK :ADJ PD
```

END

A startup procedure, s, sets up the graphics window with setextent "printer so that the students can make A4 size copies of drawings to display and discuss, and prints instructions to the listener window.

TO S

```
CT DRAW SETEXTENT "PRINTER SETFONT
'Tahoma' 8 1
PR SE 'Enter three numbers with a space
between them' CHAR 58
PPR ' the first is a number for the
number of sides of a polygon;'
PPR ' the second is a number for the
length of the sides of the polygon; and'
PPR ' the third is a number for the
radius of the arc forming the corners.'
MAKE "RESPONSE RL
SETUP.DRAWING FIRST :RESPONSE FIRST BF
:RESPONSE LAST :RESPONSE
CT
PPR (SE 'The numbers you used were' BL
:RESPONSE WORD LAST :RESPONSE ".)
PPR 'Experiment to find the effect of
changing just one of the numbers.'
PPR 'Press the spacebar when you are
ready.'
IGNORE RC
CT
PR :RESPONSE
CS ST EXPERIMENTS
```

END

The setup.drawing procedure positions the turtle for a centre-of-the-page drawing of the rounded polygon.

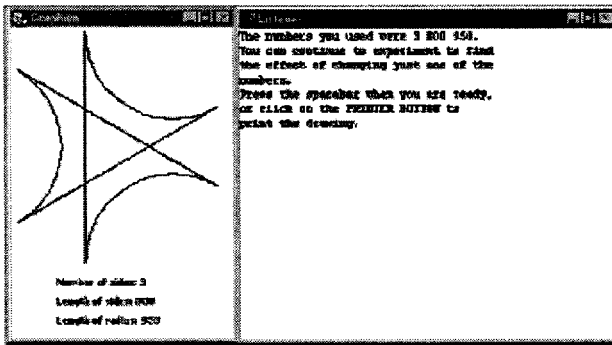
```
TO SETUP.DRAWING :N :L :R
  CS HT SET -0.50 * :L 0 0
  POLYR :N :L :R
```

END

The experiments procedure allows students to try out many different values of :N :L and :R, and to print graphics they want to keep. It calls on label.it to type text to the graphics as an aid to discussion later on.

```
TO EXPERIMENTS
  MAKE "RESPONSE RL
  SETUP.DRAWING FIRST :RESPONSE FIRST BF
:RESPONSE LAST :RESPONSE
  LABEL.IT
  CT
  PPR (SE 'The numbers you used were' BL
:RESPONSE WORD LAST :RESPONSE ".)
  PPR 'You can continue to experiment to
find the effect of changing just one of the
numbers.'
  PPR 'Press the spacebar when you are
ready, or click on the PRINTER BUTTON to
print the drawing.'
  IGNORE RC
  CT CS
  PR :RESPONSE
  ST EXPERIMENTS
END
```

```
TO LABEL.IT
  HT
  SETPC RED
  SET -700 -1000 0
  TT SE 'Number of sides:' FIRST :RESPONSE
  SET -700 -1200 0
  TT SE 'Length of sides:' FIRST BF :RESPONSE
  SET -700 -1400 0
  TT SE 'Length of radius:' LAST :RESPONSE
  SETPC BLACK
END
```



An extension of Spezeski's concept is to draw the regular polygon on which the rounded one is based, and to nest inside it rounded ones with arcs of incrementing radii. An upper limit is placed on the radii. For this nested.polys and Spezeski's poly are needed.

```
TO NESTED.POLYS :R
  IF :R > :U THEN STOP ;U is the upper
limit placed on the length of the radius
  POLYR :N :L :R
  NESTED.POLYS :R + 110
END
```

```
TO POLY :N :L
  REPEAT :N [FD :L RT 360 / :N]
END

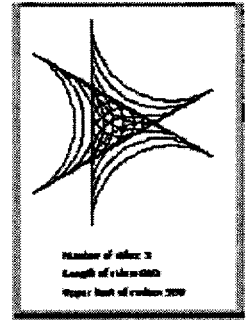
Setup.drawing becomes

TO SETUP.DRAWING :N :L :U
  CS HT SET -0.50 * :L 0 0
  POLY :N :L
  NESTED.POLYS 5
END
```

Adjustments are needed to s and label.it.

In s PPR ' the third is a number to set the upper limit of the radius of the arc forming the corners.'

In label.it
TT SE 'Upper limit of radius:'
LAST :RESPONSE



The resulting graphics are obviously different, as can be seen when the same values are applied as in the example graphic above.

Note

The microworld was written for PC Logo 2. For other dialects of Logo, setextent (printer) would need to be removed, other means of printing text to the graphics screen perhaps used, and PR [] (square brackets) used instead of PPR (or PR) ' ' (apostrophes) for printing mixed-case text to the listener window. LX

References

- McMillan, B. (1989). Teaching with Logo Microworlds. *Computers in NZ Schools*, 2, 49-54.
- McMillan, B. (1992). Logo in the Context of Classroom Learning. In K.W. Lai & B. McMillan (Ed.), *Learning with Computers: Issues and Applications in New Zealand* (pp. 148-174). Palmerston North, NZ: Dunmore Press.
- Yelland, Nicola J. (1992-1993). Introducing Young Children to Logo. *Computing Teacher*, 20, 12-14.
- Yelland, Nicola J. (1995). Encouraging Young Children's Thinking Skills with Logo. *Childhood Education*, 71, 152-155.
- Spezeski, W. J. (1996). Logo. *Models and Methods for Problem Solving*. Cambridge, MA: Harvard Associates.
- Spezeski, W. J. (1999). Polygons and More. *Logo Exchange*, 17(2), 31-32.

About the Author

John Hayes is principal of Te Puke Primary School in rural Te Puke, near the city of Tauranga, New Zealand. The school is predominately for lower socio-economic children aged 5 to 11.

John Hayes

Cameron Rd, Te Puke 3071, New Zealand
jahayes@xtra.co.nz



STARTING WITH STARLOGO

Auto Maze

by ALAN EPSTEIN

Near where I live there is a life size labyrinth constructed from stalks of sorghum. For a nominal fee one can try one's luck as a rat finding its way through the maze, with special bridges on which one can view the pattern from slightly higher than plant tops. There are also sound tunnels with which one can communicate with other seekers in random parts of the maze. A good afternoon of fun and visceral problem solving.

StarLogo is a versatile tool for building and solving mazes. In this column I will show the power of multiple turtles in the construction of a maze, and a first pass at finding a way out.

One way of building a maze is to use a pattern or template to describe the interconnected walls. Unless there is some way of automating the description process, building different mazes this way can be rather laborious.

An alternative is to give each of scores of turtles a set of instructions for building a part of the maze, and turn them loose to self organize. The basic idea is this: distribute turtles randomly to start, have them move forward, occasionally turning to the right or the left, and stopping when they bump into a wall. By biasing their turning to head straight ahead most of the time, the wall patterns end up looking quite reasonable as a maze.

```
to setup
ca crt
builders
setup-builders
setup-fence
setup-maze
place-goal
end
```

'builders' can be a slider variable to allow for user trials and variations.

```
to setup-builders
setc brown
setx random screen-size - 3
sety random screen-size - 3
seth (random 4) * 90
end
```

The maze walls will be brown, created by the builder turtles randomly scattered about the screen, but not too near the walls. Also, the turtles begin with random compass point headings (0, 90, 180, 270) to ensure square walls.

```
to setup-fence
if ((abs xcor) > screen-edge
- 2) or ((abs ycor) >
screen-edge - 2) [setpc
blue]
end
```

For aesthetic purposes, `setup-fence` places a blue border around the

entire workspace, further containing the maze. Use of the absolute value function, `abs`, allows for very brief coding of the fence. While `setup-fence` is applied simultaneously to each patch, the x coordinate is compared against the screen limit, resulting in a blue change if within 2 pixels of the edge. If the x coordinate is negative (as half of them are), the same standard is applied, resulting in blue border bars on the left and the right. The y coordinates are treated similarly.

Now for the fun part. A recursive procedure, `iter`, is initiated by a button called `setup-maze`. `iter` is executed by each turtle continuously until it hits a wall or border and dies. This requires the initial test for existence (`alive?`). A slider for the variable `walllength` allows the user to modify at runtime the lengths of the maze walls, at least generally. Values of 4 to 8 yield decent looking mazes.

The `extend` procedure with the `walllength` argument causes each turtle to repeatedly move forward, stamping the patches brown until it either has moved far enough or has hit a wall. In the latter case, the turtle dies, having done its duty.

Using the left-or-right procedure to randomly add or subtract 90 degrees from the current heading, but only 25% of the time (if `random 4 = 0` [...]).

The other 75% of the time, the heading does not change, allowing the turtle to continue straight ahead on the next iteration. The final, tail recursive iter call continues the process.

```

to setup-maze
iter
end

to iter
if alive? [extend
walllength
if random 4 = 0 [seth
heading + left-or-right]
iter]
end

to extend :count
repeat :count [fd 1
ifelse pc = black [stamp
color] [die] ]
end

to left-or-right
ifelse (random 10) < 5
[output 90] [output -90]
end

```

Finally, a red goal patch is created in one corner of the window, just inside the border. Later the solution-finding turtles will test for red patches to know when they are finished. The place-goal procedure creates this 5 by 5 patch in the upper right hand corner of the window.

```

to place-goal
if ((xcor < screen-edge - 1
and ycor < screen-edge - 1)
and (xcor > screen-edge - 6
and ycor > screen-edge - 6))
[setpc red]
end

```

Note that there are no controls to insure that the maze that is built each time can actually be solved. Since there is no controlling observer, the turtles cannot tell for themselves whether they have created a maze that is soluble. It is therefore possible that there is no way to reach the goal from the starting area. However, this adds to the fun when later the maze-solving turtles are released to find their way to the goal patch.

With the maze generated, the problem of automatically solving the maze can be addressed. One solution would be to let loose a mass of randomly moving turtles whose objective is to find the red goal patch in the opposite corner.

First create a button for the setup-walk procedure. Then set up these turtles to begin the run with place-turtles.

```

turtles-own [win]

to setup-walk
place-turtles
end

to place-turtles
ct crt
turtles
config-turtles
end

to config-turtles
setx 4 - screen-edge
sety 4 - screen-edge
seth (random 4) * 90
if not pc = black [fd 1]
setwin false
setc white
end

```

The turtles variable should be a slider. Each turtle is placed in the corner opposite the red goal patch and given a compass point heading. If the turtle happens to fall on a wall, move it forward one to get off. The win variable is set to false, indicating that this turtle hasn't yet reached the goal.

The iterative rules for the seeker turtles can be as simple as this:

If you are not on the goal patch, move forward by one and test if you are still on a black patch. If you are not, either you have won or you have hit a wall and you should back up. In all cases turn left, right or continue facing forward. A "go" button to begin random-walk should be created.

```

to go
random-walk
end

```

```

to random-walk
if not win [fd 1
if not pc = black
[ifelse pc = red
[setwin true] [bk 1] ]
seth heading + left-or-right
random-walk]
end

```

The seeker turtles find their way over the black patches, backing up when they hit a wall or border, and turning left or right about half the time.

When the turtles reach the goal patch, their win flags are set to true, and those turtles no longer move starting in the next iteration. They will appear stuck to the patch.

Although interesting to watch, these 'dumb' turtles take quite a bit of time to find the goal. Variations on this model might include changing the left-or-right procedure to return random heading change values, such as 14 or 71 degrees instead of 90 degrees. How might this procedure be changed to do this? What effect would this have on how fast the turtles make their way to the goal?

In addition to the variations of random seeking, there are other ways to solve the maze. One way that I will explore next time is the "edge walk", where a turtle follows any connected wall by moving in a way in which the wall is always to one side (left or right). In preparation for next time, think about an algorithm for having turtles perform an edge walk. LX

About the Author

Alan Epstein is the Director of Technology for Watertown Public Schools in Massachusetts. He has also spent 24 years developing software, most recently commercial modeling and simulations systems. He holds an M.Ed from Harvard Graduate School of Education's Technology in Education Program.

Alan Epstein

Alan_Epstein@watertown.k12.ma.us



Research and Mathematics Education Standards

by **DOUGLAS H. CLEMENTS AND JULIE SARAMA**

What role should research play in determining what and how we teach? In a recent article James Hiebert (1999) tackles that difficult question (see the entire article at www.nctm.org/jrme/abstracts/volume_30/vol30-01-hiebert.html). Here, we summarize his major points. [And, as usual, put some of our own comments and additions in square brackets like these.]

Problems in Relating Research and Practice

Many people's expectations put research between a rock and a hard place. On the one hand, researchers are called upon to resolve issues that really are about values and priorities, not about research. On the other hand, research is ignored when empirical evidence could direct decision making. Even when research is used, what it can and cannot say is often misunderstood. Hiebert attempts to clarify what we can expect from research, especially in shaping educational standards, such as the math Standards (National Council of Teachers of Mathematics, 1989; 1991).

Relationships between Research and Standards

Can we say, or even ask, if "research supports the Standards?" Yes, but the

answers are not simple. First, standards are not and can not be based solely on research. Second, research does not address all topics and questions equally, or even at all. Hiebert concludes that, where research is relevant, it is consistent with the Standards.

What We Can Expect from Research

Why can't research just tell us if the Standards are right or wrong? One main reason is the standards are statements about values and goals. [This can have a fundamental impact. There have been reform projects which parents have tried to stop because they believed that teaching children to think for themselves—and in mathematics yet!—is socialist and anti-American.] Research may help you teach written computation better, but it can't tell you how much such skills should be emphasized.

Research can't even tell you what is the best approach. Why not? Haven't other scientific fields achieved that rigor? Medicine has made great strides. However, it can not tell us what is "best." Is meat good for you? Is it five vegetables and fruits per day or some other number? Is it different for different people? In education and medicine, research is valuable, but it cannot always find a best decision.

Hiebert presents an example that, because it involves technology, we quote in detail.

Is it better for students to use calculators or not to use calculators in elementary school? . . . Shouldn't we be able to prove whether children should use calculators, one way or another? Suppose we try. First, we need to decide what we mean by better and how to measure this construct. Does better mean that students, at the end, understand mathematics more deeply, solve challenging problems more effectively, execute written computation procedures more quickly, like mathematics more? Deciding what better means is not a trivial task. It requires being clear about values and priorities. Suppose, for the sake of argument, that we mean "execute written computation procedures more accurately and quickly." Many people would guess that, if this is the valued outcome, the no-calculator classroom would be the best.

How could we test this hypothesis? How would we set up a fair comparison between the calculator and the no-calculator treatments? A reasonable approach would be to

develop, with our desired learning goal in mind, the best instructional program we could think of with the calculator and the best program without the calculator. Using this approach would mean that students in the two programs probably would be completing different tasks and engaging in different activities, because different activities are possible with and without the calculator. But now we have a problem because we will not know what caused the differences in students' learning. Was it the calculator, the other differences between the instructional programs, or the interactions? Maybe we could solve this problem by keeping the instructional programs identical; just plop the calculators into one set of classrooms and not the others. But into which instructional program should the calculators be plopped—the one designed to maximize the benefits of the calculator or the one designed to function without calculators? Neither choice is good, because the omitted program would not get a fair test. Maybe we should split the differences. But then we have an instructional program that no one would intentionally design.

Does this research design problem mean that all the studies on using calculators, and there have been many, are uninterpretable? No. But it does mean that no single study will prove, once and for all, whether we should use calculators. The best way to draw conclusions regarding issues like this is to review the many studies that have been done under a variety of conditions and look for patterns in the results. . . . As it happens, this kind of review of calculator use has been done and a partial and tentative answer is available (Hembree & Dessart, 1986). The results indicate that using calculators, along with common pencil-and-paper activities,

does not harm students' skill development and supports increased problem-solving skills and better attitudes toward mathematics. This finding does not mean, by the way, that this is what will be found in every classroom, but it does indicate two things: (a) A decision to use calculators wisely during mathematics instruction can be made with some confidence; and (b) when calculators are blamed for damaging students' mathematical competence, it would be useful to check the full instructional program—the problem is likely to be a poor use of calculators, or a feature of instruction unrelated to calculators, and not the calculators themselves.

Research can't decide what your goals are. And even if it compares two methods and finds one more effective, it can't tell you what is "best"—a new alternative may be better than either of the two evaluated.

What *can* we expect from research? Research can influence standards by providing us new ways of seeing and thinking about learning. For example, early in this century, Thorndike's research encouraged a move away from teaching mathematics to "exercise the mind." More recently, research on young children's learning shows us that students can solve arithmetic story problems before school instruction and can invent their own solution methods. Research can show us that students can learn new ideas, or that certain approaches, while attractive, are not effective.

Research can also provide information about how we are doing at the moment. While obvious, this is not often used. In California, for example, many claimed that the new standards led to a decline in achievement. But there was no information on the extent to which the frameworks were influencing mathematics instruction in the state's classrooms! Without such information, the question can't be answered.

What We Can Learn from Research

One thing we *do* know is the current state of classroom teaching. It hasn't changed in a long time. Answers are given for homework. A brief explanation is given of new material. Problems are assigned for the next day. Students work independently on the homework while the teacher answers questions. Most of what is taught is procedures, especially written computation. There is little emphasis on concepts, solving challenging problems, and on mathematical reasoning, communicating, conjecturing, justifying, and proving. ***The research he cited for this was 20 years old, right?

Achievement data shows that students in the U.S. do learn procedures, but their mathematics knowledge is fragile. Hiebert says that traditional teaching approaches are deficient and can be improved." What is disappointing is that people believe that "experimental" approaches are "unproven" and traditional instruction is "proven." Hiebert argues that "presuming that traditional approaches have proven to be successful is ignoring the largest database we have."

What about new teaching methods? There are so many topics and variables that no single conclusion can be drawn. However, one area is a research success story: primary-grade arithmetic. These successful programs share several features that Hiebert describes:

- Build directly on students' entry knowledge and skills. Many students enter school being able to count and solve simple arithmetic problems. . . .
- Provide opportunities for both invention and practice. Classroom activity often revolves around solving problems that require some creative work by the students and some practice of already learned skills. . . .
- Focus on the analysis of (multiple) methods. Classroom discus-

sion usually centers on the methods for solving problems, methods that have been presented by the students or the teacher. Methods are compared for similarities and differences, advantages and disadvantages.

- Ask students to provide explanations. Students are expected to present solutions to problems, to describe the methods they use, and to explain why they work.

Research indicates that such programs can facilitate significant mathematics learning without sacrificing skill proficiency and that students learn new concepts and skills while they are solving problems.

Why don't we have more programs like this? Research confirms that professional development is the answer. Also, such development is demanding, including "(a) ongoing (measured in years) collaboration of teachers for purposes of planning with (b) the explicit goal of improving students' achievement of clear learning goals, (c) anchored by attention to students' thinking, the curriculum, and pedagogy, with (d) access to alternative ideas and methods and opportunities to observe these in action and to reflect on the reasons for their effectiveness. . . . Because most classroom teachers in the United States do not yet have learning opportunities of this kind, it is not surprising that promising alternative methods are not widely implemented."

What About Logo?

Many of Hiebert's points apply to research on Logo, whether in mathematics education or other spheres. There are the same difficulties and cautions. One reason we provided Hiebert's discussion of calculators in such detail is that almost the same argument could be made concerning Logo. We know that Logo is expected to "prove itself" (and there is nothing wrong with that) but non-Logo, traditional approaches do not. We can't always use research

to "prove" what we value about Logo.

Research can tell us a lot about Logo. To provide all these cautions, and then briefly tell what the research says about Logo in mathematics education, would be, well, incautious. But we can provide the conclusions of our recent review (Clements & Sarama, 1997). There we conclude that Logo programming can serve many of the same purposes as the "experimental programs" that Hiebert discusses. Working with Logo, students can construct elaborate knowledge networks (rather than mechanical chains of rules and terms) for mathematical topics. There are several unique characteristics of Logo that facilitate students' learning.

1. The commands and structure of the computer language are consistent with mathematical symbols and structures.
2. Logo promotes the connection of symbolic with visual representations, supporting the construction of mathematical strategies and ideas out of initial intuitions and visual approaches. We believe that Hiebert would agree with the somewhat simplified statement that understanding is making connections.
3. The turtle's world involves measurements that are visible yet formal quantities, helping to connect spatial and numeric thinking.
4. Logo permits students to outline and then elaborate and correct their ideas. Logo helps document student actions, leading the mathematical symbolization.
5. Logo encourages the manipulation of screen objects in ways that facilitates students' viewing them as mathematical objects and thus as representatives of a class.
6. Logo demands and so facilitates precision and exactness in mathematical thinking.
7. Logo provides a mirror of students' mathematical thinking. For teachers willing to work with


and listen to students, such environments provide a fruitful setting. They help take the student's perspective and discover previously unsuspected abilities to construct sophisticated ideas if given the proper tools, time, and teaching.

8. Because students may test the ideas for themselves on the computer, they aid students in moving from naive to empirical to logical thinking and encourage students to make and test conjectures. Thus, Logo facilitates students' development of autonomy in learning (rather than seeking authority) and positive beliefs about the creation of mathematical ideas.

Hiebert warns that you can not conclude that an experimental approach has no effect if you do not know if it is used well. The original developers wanted Logo to serve as a conceptual framework for learning mathematics (Feurzeig & Lukas, 1971; Papert, 1980a). Many studies, however, are built on the assumption that straightforward exposure to mathematical concepts within the context of Logo programming increases mathematics achievement. Research evidence pertaining to this presumption is inconclusive. In contrast, using Logo programming as a conceptual framework is not a method of directly practicing or teaching mathematical ideas. Instead, its effects on mathematical knowledge may result from students' construction and elaboration of schemata that form a structure upon which they build future learning. In particular, Logo may permit students to manipulate embodiments of certain mathematical ideas. Serving as a transitional device between concrete experiences and abstract mathematics, it may facilitate students' elaboration of the schemata for those ideas. Finally, Logo is an environment in which students can use mathematics for purposes that are meaningful and personal for them.

As Hiebert argued, the teacher's role is critical. Teachers must be involved in planning and overseeing the Logo experiences to ensure that students reflect on and understand the mathematical concepts. They need to (a) focus students' attention on particular aspects of their experience, (b) reduce informal language and provide formal mathematical language for the mathematical concepts, (c) suggest paths to pursue, (d) facilitate disequilibrium using computer feedback as a catalyst, and (e) continually connect the ideas developed to those embedded in other contexts. Teachers must provide structure for Logo tasks and explorations to facilitate desired learning. To accomplish all this, teachers need specifically designed Logo activities and environments.

The best use of Logo may involve full integration into the mathematics curriculum. Too much school mathematics involves exercises devoid of meaning. Logo is an environment in which students use mathematics meaningfully to achieve their own purposes. The Logo language is a formal symbolization that students can invoke, manipulate, and understand. Thus, using Logo in mathematics is "teaching students to be mathematicians vs. teaching about mathematics" (Papert, 1980b, p. 177).

Finally, we need continuing research and development to expand our knowledge of what students and teachers learn in various Logo classrooms. Standardized tests do not measure many concepts and skills developed in Logo (Butler & Close, 1989). 

References

Butler, D., & Close, S. (1989). Assessing the benefits of a Logo problem-solving

course. *Irish Educational Studies*, 8, 168-190.

Clements, D. H., & Sarama, J. (1997). Research on Logo: A decade of progress. *Computers in the Schools*, 14(1-2), 9-46.

Feurzeig, W., & Lukas, G. (1971). LOGO—A programming language for teaching mathematics. *Educational Technology*, 12, 39-46.

Hembree, R., & Dessart, D. J. (1986). Effects of hand-held calculators in precollege mathematics education: A meta-analysis. *Journal for Research in Mathematics Education*, 17, 83-99.

Hiebert, J. C. (1999). Relationships between research and the NCTM Standards. *Journal for Research in Mathematics Education*, 30, 3-19.

National Council of Teachers of Mathematics. (1989). *Curriculum and evaluation standards for school mathematics*. Reston, VA: Author.

National Council of Teachers of Mathematics. (1991). *Professional standards for teaching mathematics*. Reston, VA: Author.

Papert, S. (1980a). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

Papert, S. (1980b). Teaching children thinking. In R. Taylor (Ed.), *The computer in the school: Tutor, tool, tutee* (pp. 161-176). New York: Teachers College Press.

About the Authors

Douglas H. Clements, Professor at the State University of New York at Buffalo, has studied the use of Logo environments in developing children's creative, mathematics, metacognitive, problem-solving, and social abilities. Through a National Science Foundation (NSF) grant, he developed a K-6 elementary geometry curriculum, Logo Geometry (published by Silver Burdett, & Ginn, 1991). With colleagues, he is working on the previously mentioned NSF

research grant and is finishing a second NSF-funded project, *Investigations in Number, Data, and Space*, to develop a full K-5 mathematics curriculum featuring Logo. With Sarama, he is co-authoring new versions Logo for learning elementary mathematics. One, *Turtle Math*, is currently available from LCSI. Sarama and Clements are co-PI's on the aforementioned *Building Blocks* project, which is developing mathematics software for preschool to grade 2 children.

Julie Sarama, Ph.D., is an assistant professor at Wayne State University, where she teaches mathematics content courses for pre-service teachers and research courses for graduate mathematics education students. She has studied teachers' use of computer innovations and students development of mathematical constructs while working in computer microworlds. She is co-author of several *Investigations* units and of *Turtle Math* and has designed and programming new versions of Logo and other computer microworlds. She is co-principal investigator on the new *Building Blocks* project.

Douglas H. Clements
SUNY at Buffalo
Dept. of Learning and Instruction
593 Baldy Hall
Buffalo, NY 14260
Clements@acsu.buffalo.edu

Julie Sarama
Wayne State University
Teacher Education Division
Detroit, MI 48202
Sarama@coe.wayne.edu



FOR BEGINNERS

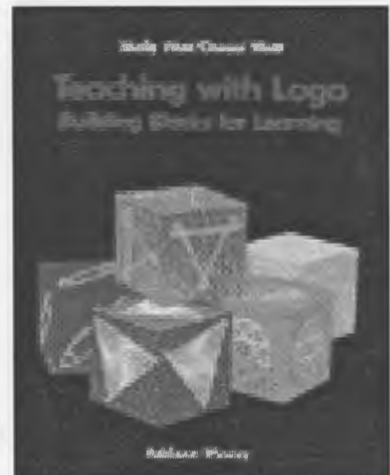
21st Century Logo Quilts

An Activity for Advanced Beginners

by **GARY S. STAGER**

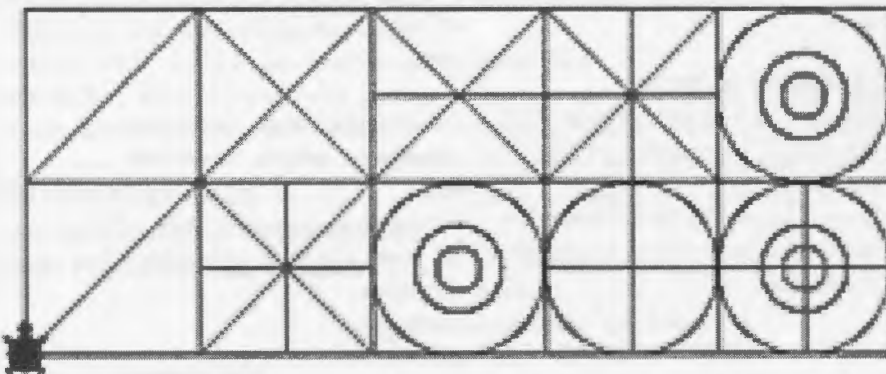
Back in the '80s, a popular Logo activity was to have kids make quilts with turtle graphics. This project lends itself to collaboration, mathematical problem solving, creativity and folk lore. Students can tell stories through their quilts and the act of assembling a Logo quilt requires the communication of geometric relationships to the computer.

3. Write patch procedures in which:
 - a. all of the drawing occurs within the walls of the patch
 - b. the turtle always returns to where it began—heading in the original direction. This *state transparency* allows for the next block to be drawn predictably. In Logo as in life, it is good to return to where you began.



Dan and Molly Watt wrote extensively about teaching and learning with Logo. Their book, *Teaching with Logo—Building Blocks for Learning*, takes an in-depth look at quilting with turtle graphics. Ideas for understanding student learning and assessing that learning may be found in the book listed below.

The following are some sample procedures for making old-fashioned quilts. The illustration above was created with these self-explanatory procedures. The `move.r`, `move.l`, `move.u` and `move.d` procedures are intended to slide the turtle so another patch may be created.



For those of you interested in exploring “old-fashioned” Logo quilt making with your students, follow these instructions:

1. Agree on a size for the basic block. 80X80 turtle steps is a good size.
2. Write a **BLOCK** procedure to draw the outline of the patch.
3. Name each patch procedure with the initials of the programmer and a number, i.e. . . . `gs1`, `gs2` . . . `gs5`, etc. This system of unique naming makes it possible to share procedures to build common quilts.
4. Experiment, have fun, use color
5. Write procedures to randomly assemble new quilts.

Old-fashioned Quilt Starter Procedures

```

to block
setc "black
pd
repeat 4
  [fd 80 rt 90]
end
to gs1
block
rt 45 fd 113
bk 113 lt 45
end
to gs2
gs1
fd 80 rt 90
gs1
lt 90
bk 80
end
  
```

```

to move.r
setx xcor + 80
end
to move.l
setx xcor - 80
end
to move.d
sety ycor - 80
end
to move.u
sety ycor + 80
end
  
```

Quilting MicroWorlds-style

The modern interface of MicroWorlds invites learners to create their own quilting software. In this project we will:

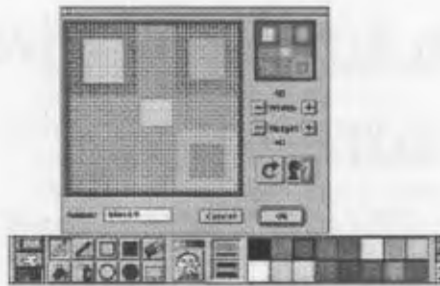
- design geometric (and non-geometric) quilt patches as turtle costumes in the shapes centre
- ask stationary turtles to behave as buttons
- move a patch into position and stamp its shape on the screen in order to assemble a mosaic-like quilt

Students still experiment with and create geometric patterns but have a different relationship to these patterns. Concepts like symmetry may come to the forefront more readily when the user creates with the stamps rather than with turtle geometry. Try it. Let me know what you think. The tool procedures provided are intended as scaffolding so students of all ages and ability levels can express their creativity and achieve programming success.

Step One—Design the Patches

- Change turtle costumes to appear as quilt patches. Be sure to use the same size square in each shape.
- Name the shapes, block1, block2, etc., to reduce confusion.

- Use your imagination, color, creativity and don't forget that you can rotate costumes in the shapes centre. Copy and paste may also reduce your workload while creating new patches.



Step 2: Create the Control Buttons

- Hatch a turtle per patch you created
- Put a different patch shape on each turtle
- Line the turtles up vertically along the side of the page or horizontally along the top/bottom of the page.

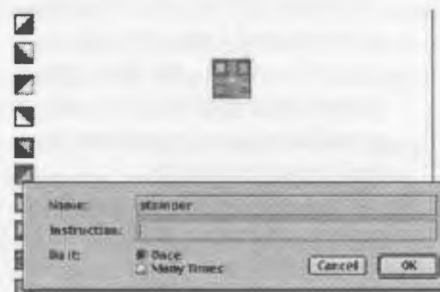
Step 3: Program the Turtle Buttons

- Use the eye tool to give each of these turtles the instruction `Changeblock "block1` where `block1` is the name of the shape attached to this turtle. Check `ONCE` in each turtle's instruction.

Step 4: Create Buttons

- Drop a MicroWorlds button with the `ONCE` instruction `STAMP IT` on the button.
- Drop a MicroWorlds button with the `ONCE` instruction `CLEAR` on the button.

Step 5: Create the Stamping Turtle (cursor)



- Hatch one more turtle.
- Name it `STAMPER`. It does not need any instructions.

Step 6: Create the Navigational Buttons

- Drop four buttons on the page.
- Arrange the buttons in compass directions.
- Give each button the once instruction `North, South, East and West`. (one each) You can't use up and down because right and left are already Logo primitives.

Step 7 (optional): Compose a Click Sound

- Drop the melody tool on the page
- Create a one or two quick note click sound to add to your stampit button.

Type the following procedures on the procedures page:

```

to clear
clean
stamper, pu home
end
  
```

```

to stampit
stamper, stamp
click
end
  
```

```

to changeblock :costume
stamper, setsh :costume
end
  
```

```

to north
stamper,
seth 0
pu fd 40
end
  
```

```

to south
stamper,
seth 180
pu fd 40
end
  
```

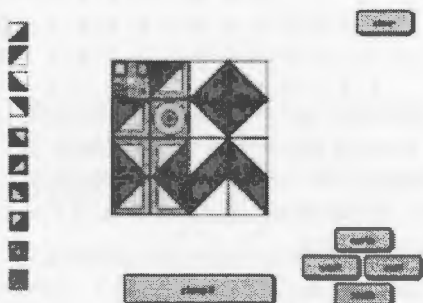
```

to east
stamper,
seth 90
  
```

```
pu fd 40
end
```

```
to west
stamper,
seth 270
pu fd 40
end
```

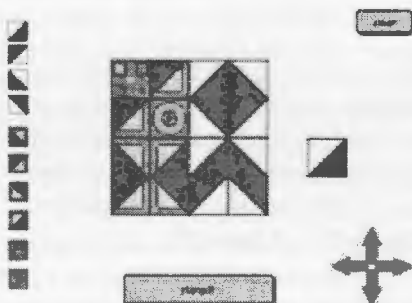
Try my sample quiltmaker below.



- Click Clear to wipe the screen clean.
- Click the navigational buttons to move the stamper.
- Click stampit to leave the patch on the screen.

In the following example, I deleted the navigational buttons (north, south, east, west) and replaced them with turtles wearing arrow costumes. Each of these arrow turtles has the appropriate north, south, east or west instruction. Be sure to tell the turtles to run the instructions only once.

Quiltmaker with navigational arrows rather than MicroWorlds buttons . . .



In the final variation on the Quiltmaker software a slider is added

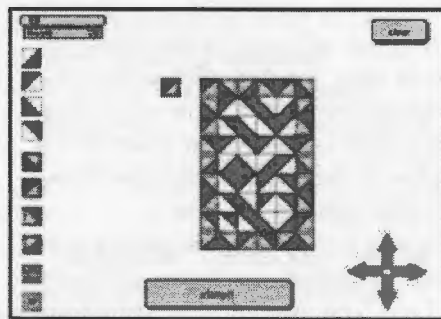
to control the size of each block and the equivalent space between patches.

A slider is a visual variable in MicroWorlds. Sliders have a name and report the value set on the slider. The slider therefore is a Logo reporter/operation.

Instead of asking the turtle to move forward 40 steps each time a navigational button is pressed, the turtle will move by the distance set by the slider. Our slider will be named, LENGTH.

The changeblock procedure has an added instruction for setting the size of the patch to the value set by the slider.

Quiltmaker with slider to determine block size



Step One

Create a slider named, Length, and set its range from 10 to 100.

Step Two

Change the procedures in the procedures page to reflect the changes in italics below. You can also use Find/Replace to swap `fd 40` with `fd length`.

```
to clear
clean
stamper, pu home
end
```

```
to stampit
stamper, stamp
click
end
```

```
to changeblock :costume
stamper, setsh :costume
setsize length
end
to north
```

```
stamper,
seth 0
pu fd length
end
```

```
to south
stamper,
seth 180
pu fd length
end
```

```
to east
stamper,
seth 90
pu fd length
end
```

```
to west
stamper,
seth 270
pu fd length
end
```

Related Links

A terrific source of links on the mathematics of quilting, geometric patterns, related books and quilting itself can be found at: <http://members.aol.com> or hit the Yahoo Quilt Index at <http://dir.yahoo.com/Arts/Crafts/Quilting/>.

Point your browser to www.tessellations.com/ for an amazing assortment of inexpensive, challenging, and beautiful foam tessellating puzzles. LX

About the author

Gary S. Stager is Editor-in-Chief of *Logo Exchange*, Editor-At-Large of *Curriculum Administrator Magazine* and an adjunct professor of education at Pepperdine University. Gary is the recent author of *MicroWorlds Pro Tips and Tricks* and *The North Star Guide to Technology Planning*. Go to www.stager.org to find a collection of resources for progressive educators.

Gary Stager, Editor-in-Chief
logoexchange@stager.org



Giving Meaning to Mean (and Standard Deviation, Too)

by TOM LOUGH

One of the strengths of Logo is its extensibility. Through our ability to choose procedure names that, in turn, become something akin to computer commands, we can give meaning to our work.

However, I am convinced that this capability has never been fully exploited, especially in the area of mathematics. In this article, I would like to offer an example or two of how carefully named procedures can provide an extra layer of meaning to mathematical constructs which are, in themselves, rather abstract.

Let me start with the concept of the standard deviation, and develop procedures in a top-down fashion, starting from a fundamental definition. Mathematically, the standard deviation is often defined as the square root of the variance. (We'll deal with variance in a moment.)

To calculate a standard deviation, we also need a set of data, organized into a list for Logo use. Putting these two ideas together, we can develop a starting procedure such as the following.

```
to standard.deviation :datalist
output sqrt variance :datalist
end
```

In this case, the standard.deviation procedure is a reporter which accepts a list of data as input, and outputs a value calculated by another procedure called variance.

After this promising start, things sometimes get a little complicated be-

cause variance is usually defined by a complicated-looking mathematical formula. However, if the formula is "translated" into English, it sounds something like the following.

Variance is equal to the product of the number of data points times the sum of each squared data point subtracted by the squared sum of all data points, all divided by the product of the number of data points times the number of data points less one.

Believe it or not, this suggests a procedure such as the one below. Compare the Logo language with the definition above.

```
to variance :datalist
output (((n :datalist) *
sum.of squared :datalist) -
(squared sum.of :datalist)) /
((n :datalist) * ((n
:datalist) - 1)
end
```

The variance procedure is a reporter that accepts a list of data as input, and outputs a value calculated by several other procedures, including ones with the names of n, sum.of, and squared.

After examining what each part of the variance definition means, it is possible to write procedures that carry out the appropriate actions. For example, the n procedure simply reports the number of data points in the list of data.

```
to n :what
output count :what
end
```

Things do get a bit complicated with the sum.of procedure. It's purpose is to report the sum of all data points in a list. Without elaboration, here is one way to do this.

```
to sum.of :datalist
ifelse not empty? :datalist
[output (first :datalist) +
sum.of butfirst :datalist]
[output 0]
end
```

The squared procedure is one that needs to perform a double duty. On one hand, it must output a list containing the square of each data point if the input is a list of data. But if the input is a number, the squared procedure must output the square of that number. The procedure below performs both of these functions.

```
to squared :what
ifelse list? :what [output
list.of.squares :what]
[output :what * :what]
end
```

As you have already noticed, one additional procedure is needed in order to produce the list of squares of the data points. Here is one way to do that. Note the similarity of this procedure design to that of sum.of.

```
to list.of.squares :datalist
ifelse not empty? :datalist
[output sentence (squared
first :datalist)
```

```
list.of.squares butfirst
:datalist] [output [ ] ]
end
```

If we have written our procedures correctly, then we can now use them to calculate standard deviations for lists of data points.

```
show standard.deviation [ 2 2
2 2 2 2 3 3 3 3 3 3 3 3 3 3 3
3 3 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 5 6 6
6 6 6 6 6 6 6 6 6 6 7 7 7 7
7 7 7 7 7 7 8 8 8 8 9 9
9]
1.971431
```

But calculation was not our primary purpose, even though it might be useful. What we were exploring here was the ability of Logo to “speak” to us, to make mathematical concepts more understandable—if we name our procedures meaningfully. I hope this example for the standard deviation has given you some ideas of your own to try.

As an extension, here is a procedure for the mean, a frequent companion to

the deviation. Is it meaningful to you? Note the use of previously defined procedures.

```
to mean :datalist
output (sum.of :datalist) / n
:datalist
end
```

Finally, variance is sometimes defined in terms of residuals (differences between the data points and the mean value). The var procedure below was designed according to this definition. As an exercise, can you deduce the English definition of variance from it? The residuals and residual procedures are derived from the definition of a residual.

```
to var :datalist
output (sum.of squared
residuals :datalist mean
:datalist) / ((n :datalist) -
1)
end

to residuals :datalist :value
ifelse not empty? :datalist
[output sentence (residual
```

```
first :datalist :value)
residuals butfirst :datalist
:value]
[output [ ] ]
end

to residual :value1 :value2
output :value1 - :value2
end
```

In closing, I hope that this example of using Logo’s extensibility to make mathematical constructs more understandable will encourage you to try some of your own. By naming procedures meaningfully, we can open the world of mathematics more fully—to ourselves and to our students. **LX**

About the author

Tom Lough is the founding editor of *Logo Exchange* and has taught a variety of mathematics courses at the high school and college levels. He is currently an assistant professor of science education at Murray State University in Murray, Kentucky.

Tom Lough

tom.lough@coe.murraystate.edu.

THE LAST WORD / Continued from Page 36

mathematicians interpret their calculation or technology-based experiment in the light of their background knowledge; students typically are *building* that background knowledge from the experiences they have with the technology. These differences suggest ways that the technology should and should not be used in learning.

Access or excess?

Technology makes some mathematical activities and problem domains much more accessible by removing the drudgery or supplying the computational brawn. Because of this, technology makes it possible for certain important mathematical ideas to take root in intuitive forms earlier than would otherwise have been possible, potentially laying a valuable foundation for the later formalization of these ideas. But sometimes the ideas are genuinely more subtle than they appear, and early access trivializes or distorts them.

Fallout of information age.

Computers force change (remember the Trojan Horse) but they also streamline and amplify what already prevails, and can thereby influence culture without our intent or even our awareness. Troy’s revenge. The greatest influence electronics has had on education is not anything done in a classroom, but rather the shift toward machine-scorable multiple-choice tests. This supports a school epistemology that gives very heavy weight to knowledge of facts.¹ From this perspective, the SAT is the intellectual parent of “200 things your second grader needs to know,” and computers may be abetting the forces against the stated goals of curriculum reform. The rush to the Internet and the push to get data analysis into the curriculum—both made possible only by the new computational technologies—further suggest that the current Zeitgeist focuses on gathered information, rather

than on the systematization of ideas or the fostering and formalization of reasoning. I’m dubious that this is the right way to go, but even if it should turn out to be, our present voyage in that direction is not so much a considered decision as it is a move of momentum. **LX**

¹ Yes, it is possible to write multiple choice tests that call for understanding, but it’s harder, because the testee can often use pattern matching strategies instead of understanding to find answers.

About the author

Dr. Paul Goldenberg is Senior Scientist at the Education Development Center in Newton, Massachusetts. He is coauthor of *Exploring Language with Logo* with Wally Feurzig.

Paul Goldenberg
paulg@edc.org.

Chipping Away at Mathematics:

A long-time technophile's worries about computers and calculators in the classroom

by **E. PAUL GOLDENBERG**



Editor's note: Long-time Logo pioneer Paul Goldenberg recently wrote a paper regarding the marriage of mathematics learning and technology for the NCTM Standards 2000 Draft Committee. He has graciously allowed Logo Exchange to share an abstract of this provocative paper. The entire paper may be found online at: <http://forum.swarthmore.edu/technology/papers/papers/goldenberg/goldenberg.html>

Mathematics, not technology, must lead

Discussions about computational technologies in the schools often stress the rapidity of change and how very hard it is for schools to scramble to keep up with the changes: new software makes old hardware obsolete, new hardware makes old software obsolete, new capabilities change people's expectations. But this is a cart-before-horse perspective. The new computational technologies make certain things easier and other things harder. It is easy to get seduced by the possibilities, constrained by the limitations, and driven by the momentum. These forces are poor guides for educational change. Good decision making must keep technology the servant and not the master.

Ideas have more than one role

Without technology, certain computational techniques were indispensable in order to find answers. But here's what's often overlooked: Some of these techniques had, in addition to their basic computational function, other important benefits that remained largely invisible because they could be taken for granted. One didn't need to think about the side benefits because they came "for free" with the required technique. Chucking a technique because technology has rendered its *computing* function obsolete may also mean chucking these "side benefits," resulting in troublesome gaps in students' mathematical knowledge and understanding. The long division algorithm—often used as the example *par excellence* of a foolish post-calculator teaching—is a case in point.

Empowerment requires control

With the old pedagogies, although most students passed their courses, many of them—even very smart ones—learned just enough to get by. Only a very small number developed what we sometimes call "mathematical understanding." Technology offers the lure of an alternative, by which students can gain access to important mathematical ideas without the protracted skills-acquisition period that used to be the only route and that, by many accounts, failed anyway. But are we making sure that the students whose parents couldn't (or at least *didn't*) master algebra will become true masters of their spreadsheets, dynamic geometry, and other computational technologies? Or will their electronic tool skills remain just barely passable, as were the algebraic skills of their parents, effectively replacing one set of barriers with another? What will actually happen, of course, is an empirical question we must wait to answer, but what we'd *like* to have happen involves a principled decision that we must actively make now.

Computational technology for learning vs. computations for work

Students and professional people bring different backgrounds to their use of technology, and they also bring different questions and needs. For engineers and business managers, it is often the *particular* answer to a *particular* question that is of primary importance. For students, the opposite is most often the case. Likewise, scientists and

See **THE LAST WORD** (Page 35)

When computers and hand-held calculators were first gaining wide currency in classrooms, their introduction was accompanied by great hope (and hype) about what could happen and also by dire predictions about what would happen. But because they were new, what would really occur was all a matter of speculation. Research has suggested some of the answers, but by any reasonable standards the technology is still new, and its effects are (quite naturally) still only partly understood. Over the more than 25 years that I've been involved with computers in mathematics learning, I've shed some old hopes and worries, but I've also developed new ones. Because much of the contribution to this conference is focusing on the hopes and the research, I will (without being bleak or gloomy) confine myself to listing my new worries, worries that are not, to the best of my knowledge, yet well addressed in the research. I'll also try to suggest ways, conjectural though they must be, of avoiding what I see as potential pitfalls.



Logosium '99

SIGLogo's Annual Celebration of Logo Learning
June 21st, 1999

- Hands-on Workshops Exploring the Vast Possibilities of Logo
- Practical Teaching Strategies by Logo-using Teachers
- Exciting Guest Speakers
- Authentic Philly Lunch and Great Dinner before Returning to NECC in Atlantic City

A bus will leave Atlantic City for Philadelphia where this historic event will occur.
Don't miss the best Logo event of the year!

Register at <http://www.neccsite.org>

ISTE BRINGS THE WORLD OF TECHNOLOGY CLOSER TO YOU.

By drawing from the resources of committed professionals worldwide, ISTE provides support that helps educators like you prepare for the future of education.

As an ISTE member, you benefit from a wide variety of publications, national policy leadership, and our work with Teacher Accreditation.

You also enjoy exciting conferences, global peer networking, and graduate-level Distance Education courses.

So if you're interested in the education of tomorrow, call us today.



iste International Society for
Technology in Education

Teachers Helping Teachers Use Technology in the Classroom

WE'LL PUT YOU IN TOUCH WITH THE WORLD.



iste

International Society for Technology in Education

Administrative Office

1787 Agate Street, Eugene, OR 97403-1923 USA

Non-Profit
Organization
US Postage
PAID
ISTE