

Logoed Mottos

by Judi Harris

Can Logo be used to solve "real-world" problems, or must Logo challenges be contrived and solved from within simulated contexts, such as turtle graphics or LEGO/Logo? Recently, I received an interesting request by electronic mail that showed me that Logo can, in certain situations, function as *more* than an educational tool.

The Challenge

The electronic message was from Talbot Bielefeldt, the patient and witty associate editor for the International Society for Technology in Education (ISTE). In part, he asked,

Would you be interested in a programming problem for dear old ISTE? I am part of a committee charged with coming up with a new tagline or motto for the organization. The current one is "Educational Leadership in the Age of Information Technology." Quite a mouthful. It's so cumbersome, we never use it.

After an initial brainstorming session, we came up with a list of concepts, which I expanded [upon] with the aid of an electronic...thesaurus. Our next step is to see how these might be constructed into some mottos. It seems to me there are several syntax patterns appropriate for a tagline. They include:

participle/noun

([example:] "Making Waves")

participle/preposition/noun

([example:] "Working for Change")

preposition/adjective/noun

([example:] "For Better Education")

noun/participle/noun

([example:] "Teachers Helping Students")

preposition/verb/noun

([example:] "To Change Education")

All essentially complete the sentence "ISTE is..."

At this point, like any Logo fan(atic), I was smiling and already planning how I might procrastinate meeting my less interesting deadlines so that I could begin work on this special Logo challenge.

Jumping In

Talbot had included five brainstormed word lists (nouns, verbs, adjectives, prepositions, and participles) at the end of his message. I downloaded the message, copied the word collections to a new *LogoWriter* page, and turned each into a procedure that OUTPUTs a list that contains all words in a particular category. These procedures are listed below.

TO NOUN

```
OUTPUT [education change future
world technology solutions oppor-
tunity challenge vision questions
issues earth society humanity
outlook prospect resolution an-
swer key innovation transforma-
tion difference guidance knowl-
edge learning teachers learners
students teaching advocate voca-
tion passion force power light
liberation renewal creation des-
tiny individual center] END
```

TO VERB

```
OUTPUT [guide know learn expect
imagine envision transfigure
change transform resolve answer
solve serve summon dare question
confront help motivate kindle
encourage inspire awaken excite
spark advocate light ignite lib-
erate creating]
```

END

TO ADJECTIVE

```
OUTPUT [educational effective dy-
namic meaningful true relevant
instructional inspiring produc-
tive current strong constructive
practical useful active expert
exciting rightful authentic cer-
tain positive destined central
far farthest]
```

END

TO PARTICIPLE

```
OUTPUT [guiding knowing learning
expecting imagining envisioning]
```

transfiguring changing transform-
ing resolving answering solving
serving summoning daring ques-
tioning confronting helping
striving]

END

TO PREPOSITION

OUTPUT [of by for to in into within
inside [out of] with]

END

That was the easy part. I paused to marvel at how fascination with language structures can translate into Logo programming opportunities as easily as can fascination with geometric figures or the physics of movement.

Choosing a Method

In another part of his message, Talbot had asked me,

...do you have some code lying around that you could modify to generate a heap of sample taglines? The program would use the syntax elements as variables and the word lists as input.

Not bad for an only-occasional Logo tinkerer, eh? Talbot had recognized a "Logo-like" problem and mentally constructed a tentative blueprint for the procedures that could give the committee their desired output. He went on to predict,

Many of the combinations would be garbage. A few might be quite amusing. But it would be one way [for us] to get a start...

At this point, I realized that Talbot was requesting output from a set of Logo procedures that would generate possible *permutations* of words according to each of the five syntactical structures mentioned earlier in his message. Those combinations could be generated one at a time and at random, or *all* possible permutations could be generated and listed.

Simple Sampling

Telling Logo to generate random combinations according to predetermined syntactical structures is an easy task. The PICK tool can be used to make random selections from within each of the five word type lists (nouns, verbs, adjectives, prepositions, and participles):

```
TO PICK :LIST
  OUTPUT ITEM (1 + RANDOM COUNT :LIST)
  :LIST
END
```

Combined with a simple PPN procedure that concatenates random selections from named word lists, the PICK tool and the primitive PRINT quickly form possibilities for the ISTE motto in a participle/preposition/noun format:

```
TO PPN
```

```
  OUTPUT (SENTENCE PICK PARTICIPLE
          PICK PREPOSITION PICK NOUN)
```

```
END
```

Typing PRINT PPN may cause the computer to produce

```
ENVISIONING WITH INNOVATION
```

or...

```
SUMMONING TO CHALLENGE
```

or...

```
GUIDING INTO LIBERATION
```

Procedures similar to PPN could be coded to yield potential ISTE mottos in other formats. For example, TO PVN could be used to generate random preposition/verb/noun permutations:

```
TO PVN
```

```
  OUTPUT (SENTENCE PICK PREPOSITION
          PICK VERB PICK NOUN)
```

```
END
```

Of course, concatenation tools such as PVN could output

```
INTO RESOLVE EARTH
```

as easily as they could produce

```
TO GUIDE TRANSFORMATION
```

The disadvantage to giving tools such as these to Talbot and the motto committee is, of course, that someone would have to keep typing some version of the PRINT PICK (concatenation type) procedure execution until Logo randomly generated meaningful phrases that could be used as motto candidates. I suspected that the committee wanted a more time-efficient way to review motto ideas than using procedures that produced one option at a time.

Permutation Possibilities

The *real* challenge, I was soon to discover, was to write a set of Logo procedures that would generate and print all possible permutations of mottos within each format type. Since the numbers of motto options in the participle/preposition/noun structure alone were more than 7,700, I decided to create three smaller lists with which to experiment with Logo permutation.

Inspired by the November ice storm that rages outside my door as I write this column, and the knowledge that many of you will be enjoying spring weather as you read about Logo mottos, I decided to create lists of SEASONS, STATES of being, and DESCRIPTORS:

```
TO SEASON
OUTPUT [WINTER FALL SUMMER SPRING]
END

TO STATE
OUTPUT [[WILL BE] WAS IS]
END

TO DESCRIPTOR
OUTPUT [COMING HERE]
END
```

There are 24 possible permutations of words in a season/state/descriptor format. They are:

```
SPRING IS HERE
SPRING WAS HERE
SPRING WILL BE HERE

SUMMER IS HERE
SUMMER WAS HERE
SUMMER WILL BE HERE

SPRING IS COMING
SPRING WAS COMING
SPRING WILL BE COMING

SUMMER IS COMING
SUMMER WAS COMING
SUMMER WILL BE COMING

FALL IS HERE
FALL WAS HERE
FALL WILL BE HERE

FALL IS HERE
FALL WAS HERE
FALL WILL BE HERE
```

```
WINTER IS COMING
WINTER WAS COMING
WINTER WILL BE COMING
```

```
WINTER IS COMING
WINTER WAS COMING
WINTER WILL BE COMING
```

As you can see, each element from each category must be combined with every other element from every other category. A multiple (embedded) recursive structure, with which as many calls to a COMBINE procedure as exist possible permutations, is one way to solve this challenge:

```
TO COMBINE :ITEM1 :ITEM2 :ITEM3
IF EMPTY? :ITEM1 [STOP]
COMBINE BUTFIRST :ITEM1 :ITEM2
      :ITEM3
IF EMPTY? :ITEM2 [STOP]
COMBINE :ITEM1 BUTFIRST :ITEM2
      :ITEM3
IF EMPTY? :ITEM3 [STOP]
COMBINE :ITEM1 :ITEM2 BUTFIRST
      :ITEM3
MAKE "NEW.COMBINATION (SENTENCE
      FIRST :ITEM1 FIRST :ITEM2 FIRST
      :ITEM3)
IF NOT MEMBER? :NEW.COMBINATION
      :COMBINATIONS [ADD.TO.LIST]
END

TO ADD.TO.LIST
MAKE "COMBINATIONS LPUT
      :NEW.COMBINATION :COMBINATIONS
END
```

As you can see in the last three lines of COMBINE, the computer checks to see if a newly formed combination of single elements from :ITEM1, :ITEM2, and :ITEM3 already exists in the "COMBINATIONS list before adding it with the ADD.TO.LIST procedure. This is a necessary step, since the embedded recursive-calls to COMBINE cause some duplicate concatenations to be formed.

The permutation process is initiated with a superprocedure that I called PERMUTE3:

```
TO PERMUTE3 :ITEM1 :ITEM2 :ITEM3
MAKE "COMBINATIONS [ ]
COMBINE :ITEM1 :ITEM2 :ITEM3
INSPECT :COMBINATIONS
END
```



PERMUTE3 initializes the value of the "combination collector" global variable, "COMBINATIONS, then directs COMBINE to execute, using the lists specified as values for the local variables :ITEM1, :ITEM2, and :ITEM3. Notice that all possible combinations are generated and stored before any are printed for the user to see. INSPECT (adapted from Bull and Cochran's tool with the same name) is the subprocedure that prints the combinations, organizing them into several columns to make viewing easier:

```
TO INSPECT :LIST
IF EMPTY? :LIST [STOP]
INSERT FIRST :LIST
IFELSE (FIRST CURSORPOS) < 50 [IN-
  SERT CHAR 9] [INSERT CHAR 13]
INSPECT BUTFIRST :LIST
END
```

Please note that all procedures in this article are written in *LogoWriter*, but can be easily adapted to function in any full-featured version of Logo.

Time Is All We Need

Excited that the permutation procedures worked so well with my sample season/state/descriptor lists, I eagerly invoked the superprocedure with PARTICIPLE, PREPOSITION, and NOUN as inputs:

```
PERMUTE3 PARTICIPLE PREPOSITION NOUN
```

Forty-five minutes later, my Macintosh SE/30 with 5 megabytes of RAM was still permuting. It is fortunate that my Mac seems to have more patience than I do with repetitive tasks, since it will faithfully generate and store potential ISTE mottoes for hours on end while I am involved in more interesting work.

When Sharon Yoder, the editor of the *Logo Exchange*, saw the code I had written for making permutations, she suspected, as I did, that there was a simpler, more efficient way to tell the computer to make more than 7,700 combinations. An electronic mail message to Brian Harvey, the "Logo Wizard" <grin>, was all that was needed to provide a much more elegant, economical solution to this interesting problem.

Expertise Is Ultimate Time-Efficiency!

Brian suggested that we use all of the word-list procedures and the INSPECT tool, along with these three procedures that form a new Logo operation:

```
TO COMBINE :LISTOFLISTS
IF EMPTY? :LISTOFLISTS
[OUTPUT [ [ ] ] ]
OUTPUT PREPENDS (FIRST :LISTOFLISTS)
  (COMBINE BUTFIRST :LISTOFLISTS)
END
```

```
TO PREPENDS :TOPS :BOTTOMS
IF EMPTY? :TOPS [OUTPUT [ ] ]
OUTPUT SENTENCE (PREPEND FIRST :TOPS
  :BOTTOMS) (PREPENDS BUTFIRST
  :TOPS :BOTTOMS)
END
```

```
TO PREPEND :TOP :BOTTOMS
IF EMPTY? :BOTTOMS [OUTPUT [ ] ]
OUTPUT FPUT (SENTENCE :TOP FIRST
  :BOTTOMS) (PREPEND :TOP BUTFIRST
  :BOTTOMS)
END
```

To generate, for example, all the possible motto combinations that contain participles, prepositions, and nouns, we would type:

```
INSPECT COMBINE (LIST PARTICIPLE
  PREPOSITION NOUN)
```

One of the most powerful aspects of Brian's solution to this problem is that COMBINE can be used to permute any number of elements into motto combinations, unlike PERMUTE, which must be altered depending upon the number of elements to concatenate. That means that Talbot and his committee can have all of the output they requested with just five requests:

```
INSPECT COMBINE LIST PARTICIPLE NOUN
INSPECT COMBINE (LIST PARTICIPLE
  PREPOSITION NOUN)
INSPECT COMBINE (LIST PREPOSITION
  ADJECTIVE NOUN)
INSPECT COMBINE (LIST NOUN
  PARTICIPLE NOUN)
INSPECT COMBINE (LIST PREPOSITION
  VERB NOUN)
```

As I write this column this evening in Omaha, I am envisioning Talbot and the motto committee's reactions as they receive five very *large* text files by electronic mail when they return to work after a (hopefully) restful weekend. Somewhere in the more than 30,000 possibilities that their word lists were used to create, there just *may* be a new ISTE motto waiting to be discovered and adopted. It delights me that Logo—and more importantly, people's creativity and teamwork—

fueled the search. Why not share a similar collaborative language adventure with *your* students?

Judi Harris works in the Department of Teacher Education at the University of Nebraska at Omaha as an assistant professor of educational technology. Her teaching, research, and service interests include Logo (of course), developmental sequencing in educational hyper-media materials design, telecomputing for K-

12 teachers, and the restructuring of teacher education paradigms.

Judi Harris
514J Kayser Hall
Department of Teacher Education
University of Nebraska at Omaha
Omaha, NE 68182

BITNET: JHarris@unoma1
Internet: JHarris@Zeus.unomaha.edu

Oops!

In the December/January 1990/91 issue of *Logo Exchange*, there was an article entitled "Easy Map Drawing with *LogoWriter*." It included a listing of a program to use to draw a map. Well, drawing using that program certainly was *not* easy. The program contained a number of bugs. Following is a listing of the program that has been tested in Macintosh *LogoWriter* and therefore should run without any changes in any version of *LogoWriter*.

```
to map
  name pos "startingpos
  name heading "starthd
  ht
  pd
  startdrawing [] 0
end

to startdrawing :list :counter
  if :counter = 25 [copylist stop]
  name readchar "x
  if :x = "f [forward 5 (name lput
    [forward 5] :list "newlist)]
  if :x = "r [right 15 (name lput
    [right 15] :list "newlist)]
  if :x = "l [left 15 (name lput [left
    15] :list "newlist)]
```

```
if :x = "g [right 90 (name lput
  [right 90] :list "newlist)]
if :x = "t [left 90 (name lput [left
  90] :list "newlist)]
if :x = "d [if (first last :newlist)
  = "forward [pe back 5 pd name
  butlast butlast :newlist
  "newlist]]
startdrawing :newlist :counter + 1
end

to copylist
  flip
  bottom
  print []
  print [pu]
  (print [setpos] (list :startingpos))
  (print [seth] :starthd)
  print [pd]
  print.list :newlist
end

to print.list :commands
  if empty? :commands [stop]
  print first :commands
  print.list butfirst :commands
end
```

