LOGO AND EXTENDED DEFINITION

BRENDAN J. DESILETS

I was not looking forward to teaching extended definition, though I was not much bothered by the widely-circulated research findings that seemed, at least at the time of their publication, to debunk such "rhetorical" approaches to teaching writing. In fact, my experience with this approach had led me to believe that extended definition was as useful as any skill I could help my rather academically-inclined high school students develop, and also that the teaching of this method of development lent itself quite readily to the justly-popular process approach to writing instruction. The problem was not a lack of materials either. Over a period of years I had collected a variety of helpful exercises in development by definition, ranging from a well-structured and fairly easy lesson built around a definition of a teaspoon (Blickhahn) to the usual lessons in *Warriner's*, to some more advanced, but still high-school appropriate, work in Brooks and Warren's *Modern Rhetoric*. Still, I knew the dilemma I would face. If I stayed with the exercises only as long as I could keep the students really interested in them, about one third of them would have trouble producing essays developed by definition, largely because of confusion between the kind of brief definition they knew from their work with dictionaries and the more exhaustive sort of definition that serves as a method of development. What I needed was a tool that would let me focus on the process of extended definition, stressing the differences between extended and brief definition, in a form that was simpler than the end product I wanted to reach, a good essay, and yet captivating enough to keep the students' attention for a period of a week or so. I found the answer in a surprising place, the computer language Logo (Harvey; Papert).

Now, I am not much of a programmer, in Logo or any other

language. In fact, the Logo procedures listed here are among the simplest and most commonly used ones. They appear in Harold Abelson's *Logo for the Apple II* (all within the first fifty pages) and in many other Logo manuals and tutorials. I have used them mostly with Terrapin Logo for the Apple II series, but they work almost identically in all versions of Logo. For people who are familiar with Logo, these procedures will seem quite commonplace, though their use in teaching extended definition will not. For those of us who are not accustomed to using Logo or other programming languages, the procedures will be easy enough to provide a starting point for an exploration of the use of programming to help students improve their thinking and writing. These exercises, then, demand little programming skill on the teacher's part and none on the students'; what the teacher does need is an understanding of the skills involved in extended definition and of the role of each programming task in developing these skills.

The first programming task allows the teacher to introduce the Logo commands needed for the later tasks and reinforces the importance of precision in developing an extended definition (Carter). If many computers are available, as in a computer lab, each student or group of students should probably work at her own computer, though the project also works well with just one computer with a large display monitor, as I used it.

With Logo in its "draw" mode, start by introducing the students to the "turtle," a marker, usually triangular in shape, that leaves a trail as it moves around the screen. Next, show the students the FORWARD command, abbreviated FD, which tells the turtle to move forward, leaving a line as its trail. For example, typing FD 50 (and then pressing the "Return" or "Enter" key) will cause the turtle to move forward fifty "turtle steps" and will cause a line of the same length to appear on the screen. The only additional specialized Logo word or "primitive" the student will need is RIGHT, abbreviated RT, which causes the turtle to turn to the right. RT 90, for instance, makes the turtle turn ninety degrees to the right. As their first LOGO problem, ask the students to write a series of commands that will draw a square on the screen. Most will quickly come up with an appropriate sequence, such as FD 50 RT 90 FD 50 RT 90 FD 50 RT 90 FD 50 RT 90.

Once students have developed this sequence, show them how these commands can be used to define a procedure which the

computer will remember. Type TO BOX, and note that the Logo editor screen will appear, with TO BOX at its top. Then define the procedure called BOX by typing in each of the commands, pressing Return or Enter after each command. The screen will now look like this:

```
TO BOX
    FD 50
    RT 90
    FD 50
    RT 90
    FD 50
    RT 90
    FD 50
    RT 90
    END
```

Next, leave the Logo editor in such a way as to define the BOX procedure, (by pressing the Control and C keys simultaneously on an Apple II computer). Now, typing BOX and pressing Return or Enter will cause the turtle to draw a box on the screen.

At this point, we are ready to challenge the students a bit by presenting them with a task that requires rather careful and precise thinking. This seemingly simple task is to define a procedure that will draw an equilateral triangle. Have the students write such a procedure on paper and then try out their solutions on the screen for all to see. A correct procedure, which will probably take students a few tries to discover, is:

```
TO TRIANGLE
    FD 50
    RT 120
    FD 50
    RT 120
    FD 50
    RT 120
    END
```

Now the students have spent a day or two defining procedures, noting that good definition does require considerable precision and, therefore, in most cases, some rewriting, and are ready to look at a problem that involves a somewhat startling insight.

Their task is to write a procedure that will get the turtle to make the drawing indicated in Figure 1, in such a way that the turtle will continue to retrace the lines for an indefinite period of time. This task is a difficult one for naive Logo programmers, so difficult that some students may complain that they do not know enough Logo primitives (special words) to do the job. Students who have some experience programming in BASIC may be especially adamant with this complaint, but, in truth, the students do have all the primitives and all the knowledge of Logo syntax that they need. Give them time and as many hints as they require, but try to let the students discover a correct solution such as this one:

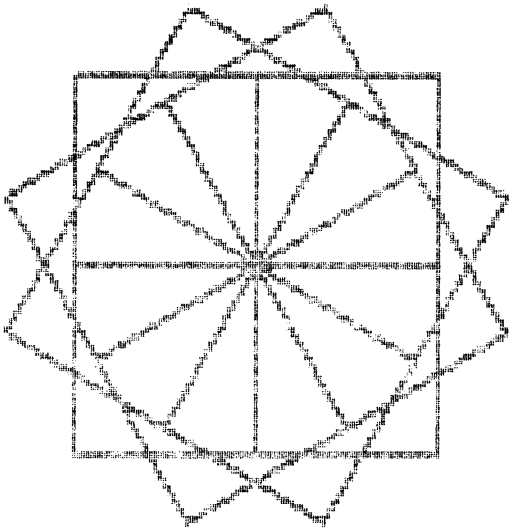

Figure 1. The Manybox Drawing. The turtle will continue to retrace the lines of this drawing until the user interrupts the execution of the procedure, as by pressing "Control" and "G" simultaneously on an Apple II computer. For the drawing to work as presented in this article, the computer must have two procedures in its memory, the BOX procedure and the MANYBOX procedure.

```
TO MANYBOX
    BOX
    RT 30
    MANYBOX
    END
```

At first, this MANYBOX procedure may seem a rather strange bit of thinking. As Harold Abelson puts it, the use of the MANYBOX as part of the definition of MANYBOX seems to be a kind of joke, but it must be logically sufficient to define the drawing since it does work (32-33). In truth, this use of procedure as part of its own definition is an extremely powerful programming idea known as recursion, which can help students understand the distinction between brief definition and extended definition.

Ask students to look closely at the MANYBOX definition and to recall rules about definition they have learned in the past. With a bit of prodding, many students will recall hearing that we should not use a word, or even a form of the word, in its own definition, a rule that the MANYBOX definition flagrantly violates. Is the old rule wrong? Not at all, if we remember the context in which we heard it, a context that involved presenting a brief definition of a new vocabulary word, a word of which we presumably had no previous knowledge. In the case of MANYBOX, though, when the computer reaches the MANYBOX step in the procedure, the machine has some previous "knowledge" of the meaning of MANYBOX; that is, it knows that part of the meaning of the word consists of BOX and RT 30. Using this previous information, the machine is able to use the definition to draw the MANYBOX figure. Most students will quickly see, as they look at the topics usually used in writing extended definitions, that these topics focus on concepts such as freedom or wealth or happiness, with which people already have some familiarity. Thus, in writing extended definition, we may, and usually should, use recursion; that is, we should repeatedly use the word we are defining as we refine its definition.

For some classes, this much work with Logo may be as much as the students can profitably use; with classes of junior high school students I have usually chosen to stop at this point. But, with academically above average high school students, I have found it helpful to move ahead to one more Logo problem, a problem that mimics the process of writing an essay of definition a little more closely.

When we used recursion in MANYBOX, we it used in a way that repeated an identical procedure an indefinite number of times. In extended definition, on the other hand, with each repetition of the word being defined, we hope to add something new to the definition. To mimic this more complex kind of recursion in a Logo problem, we need to introduce the idea of variables in Logo procedures.

Suppose that we want to write a procedure that will draw squares of varying size, rather than just producing a square of a single size, as our BOX procedure does. Logo syntax allows us to use variables as in the following example, in which we are calling our variable "SIZE":

```
TO SQUARE :SIZE
    FD :SIZE
    RT 90
    FD :SIZE
    RT 90
    FD :SIZE
    RT 90
    FD :SIZE
    RT 90
    END
```

Now, if we type SQUARE 20, the turtle will draw a square with each side twenty turtle steps long, whereas SQUARE 40 will result in a square with forty-step sides. Armed with this notion of variables, students can try to write a program that will produce the picture shown in Figure 2, and which will continually produce larger and larger squares until we interrupt the program. Once again, we are confronting students with a fairly difficult problem that will require considerable time and effort, and probably a few hints. A correct definition would be:

```
TO GROWSQUARES :SIZE
    SQUARE :SIZE
    RT 20
    GROWSQUARES :SIZE + 5
    END
```

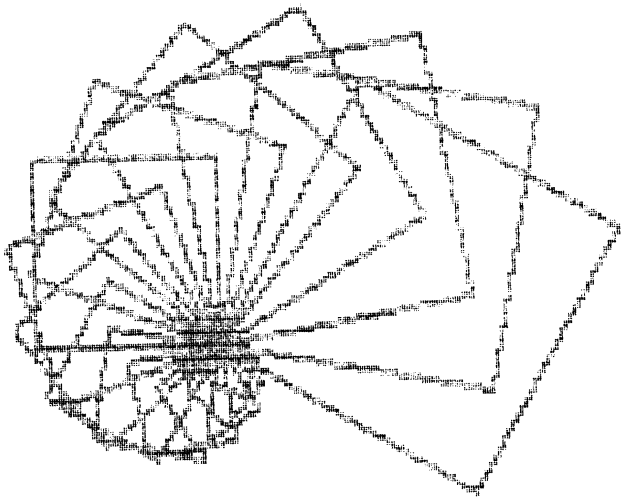

Figure 2. The Growsquares Drawing. The turtle will continue to draw larger and larger squares until the user interrupts the execution of the procedure. For the drawing to work as presented in this article, the computer must have two procedures, SQUARE :SIZE and GROWSQAURES :SIZE, in its memory.

At this point, I have, so far, chosen to stop examining the nature of extended definition through Logo and moved on to more conventional exercises, recognizing that my students have already done some hard thinking about this method of development and its recursive nature. In general, after this Logo unit, students have shown less difficulty in writing papers that really are developed through extended definition. They do not necessarily write better essays immediately, but they do write essays that are developed in the assigned way far more regularly than they do without the Logo work.

But why stop here? The recursion that we see at work in GROWSQUARES, though a bit more complex than that in MANYBOX, still falls far short of the complexity of even the most simple paragraph developed by extended definition. Couldn't we find Logo exercises that would involve still more complicated uses of recursion and thus replicate more closely the process of extended definition? Actually, most of the books on Logo contain numerous projects of this type. The main practical limitation is

the amount of time we want to invest in this investigation. In the future, though, as we encounter more and more students who have long ago explored the GROWSQUARES procedure in elementary school or at home, we will surely find that we can start with these more complicated problems rather than ending with them. Of course, such a situation may demand a little more of us as programmers, but these are the costs of progress.

Brendan J. Desilets currently teaches at the John Glenn Middle School in Bedford, Massachusetts.

WORKS CITED

Abelson, Harold. *Logo for the Apple II*. New York: McGraw Hill, 1982.

Blickhahn, Katherine M., et al. *Writing: Unit Lessons in Composition, 1A*. Boston: Ginn and Company, 1964.

Brooks, Cleanth and Robert Penn Warren. *Modern Rhetoric: Shorter Edition*. New York: Harcourt, Brace and World, 1961.

Carter, Ricky. "The Complete Guide to Logo." *Classroom Computer News* 3 (April 1983): 35-39.

Harvey, Wayne. "Which Programming Language is Right for You?" *Classroom Computer Learning* 4 (April/May 1984): 51-53.

Papert, Seymour. *Mindstorms: Computers, Children, and Powerful Ideas*. New York: Basic Books, 1980.

Warriner, John E.. *Warriner's English Grammar and Composition: Complete Course*. New York: Harcourt Brace Jovanovich, 1973.