ISSN 0734-1717

The National LOGO Exchange

Volume 1 Number 3 FORWARD 100! November 1982

Molly Watt Writes for Teachers

The October issue of Creative Computing magazine contains an article well worth reading. "What Is LOGO?" by Molly Watt, appearing on page 112, is an excellent composition which presents the point of view of a teacher who has used LOGO in many different situations.

If you have been on the lookout for LOGO documentation to give to decision makers in your school systems, plan on reading Watt's article. It might be just what you have been needing.

She starts with an overview of LOGO and a bit of the historical development. Then she presents some examples of typical beginners work.

Next, she shares some of her personal experiences as a teacher in Amherst and Cambridge, Massachusetts. Her students had a wide age range, from 8 to 17 years.

Her case studies of the individual students reveal the true potential of LOGO, both of the language itself, and its accompanying teaching philosophy.

She then directs her attention to the development of the role of the LOGO teacher, and offers several specific techniques which were effective for her.

Lastly, she makes an extremely interesting comparison: what LOGO teachers say they teach versus what LOGO students say they learn. There are probably no big surprises in this comparison. But, the reader is likely to make a few new discoveries which may not have been apparent before.

In addition to the Watt article, Robert Lawler's "LOGO Ideas" on page 111 of the same issue contains a discussion of the passing of parameters which clarifies several points often misunderstood.

He also discusses the virtues of planning how to break a programming problem into parts, operating on each individually. ▷

Hawaiian Students Love LOGO Overlays

based on an idea submitted by Elaine Blitman

Acetate transparencies can be used to create LOGO overlays which have many applications. Elaine Blitman, a supervisor for Punahou School in Honolulu, Hawaii, and her teachers and some 800 children are exploring the many possibilities offered by this idea.

To make a LOGO overlay, simply obtain a sheet of acetate such as that used for overhead projectors. Draw directly on it with a grease pencil or acetate marker, and place it on the computer monitor screen.

In most cases, the normal static cling will be sufficient to hold it in place. Otherwise, turning the monitor off then on once will usually help.

What sort of thing can be drawn on the overlay? That is left to the teacher (and the children). Mrs. Blitman suggests mazes, game boards, targets, and other similar shapes for starters.

Once the overlay is in place, the children can guide the turtle around, through, over, to, along, or behind the parts of the drawing, depending upon its nature and invitation.

A progression of overlay patterns suggests itself as part of the all-important development of turtle control. Perhaps the first overlay could be one with three or four numbered dots on it. The children are invited to move the turtle from HOME to dot #1, then to #2, and so on in succession.

Next could come an overlay with a simple closed curve in the center. The children are challenged to figure out how to make their turtle go around the outside of the closed curve one time, without necessarily trying to come close to the sides of the curve.

Perhaps next would be an overlay with a more complex numbered dots pattern, even one with a connect-the-dots picture which will develop as the turtle follows the correct path.

continued on page 2

The National LOGO Exchange, copyright 1982 Posy Publications, a part of The Posy Collection, all rights reserved. Published monthly September through May, $25 per year, $45 for two years, mailed FIRST CLASS from Charlottesville, VA. Add $5 per year for addresses outside the United States. Single copy price $3. Opinions expressed by the authors are not necessarily those of The National LOGO Exchange. Address all editorial correspondence to the editor at The National LOGO Exchange, Post Office Box 5341, Charlottesville, VA 22905. ISSN 0734-1717

Editor......................Tom Lough

Hold Your Horses

A colleague asked me recently, "Do you know what's the hardest thing for me to do when I'm teaching LOGO?" I didn't receive a chance to respond before she continued, "I'll tell you what. It is to not intervene. Do you understand?"

I encouraged her to continue. She said, "When the children are working so hard with LOGO, and they have gotten bogged down with a problem, I have this overwhelming urge to rush right in with the solution. It is so hard to wait or to just ask a question instead."

The experience is common to many. We all have seen the excitement on the faces of children which comes from making discoveries or accomplishments all by themselves. And, we all at some time have seen slightly crestfallen looks when we offer the solution to a problem too soon.

Wait time is important. Children need time to think. In a situation as simple as directing a question to a child in class, it is difficult to wait through long agonizing moments for an answer. How often do we yield to the temptation of asking another (perhaps quicker) student before the first one has had a chance to think?

With LOGO, wait time is particularly important. We all need to work hard to identify that special moment between "I hate this stupid computer anyhow!" and "Don't tell me! I want to figure it out for myself!"

It is then, and perhaps only then, that the comment, the question, the hint, or even the answer will be most effective.

It is hard to hold your horses. But let's all try. ▷

LOGO Overlays continued

Next could come simple mazes and obstacle courses. By then, the children would have devised several of their own design, of course.

Perhaps the turtle could become a space rocket and visit various planets drawn on the overlay. Or, maybe the turtle is really an automobile navigating a street map on the screen.

Sprites could also be used in conjunction with the LOGO overlays, if you have the Texas Instruments version. The children will learn quickly that reasonable values for SETSPEED must be selected! However, if you TELL SPRITE 1 to CARRY :BALL and SETCOLOR :YELLOW, there appears on the screen a creature bearing a fairly close resemblance to PAC-MAN™ which many children can move about quite skillfully!

The following TI LOGO listing will provide children with a means of controlling a moving sprite:

```
TO E
  SETHEADING 0
  END

TO S
  SETHEADING 270
  END

TO D
  SETHEADING 90
  END

TO X
  SETHEADING 180
  END

TO CONTROL
  CALL READCHAR "K
  RUN SENTENCE :K [ ]
  CONTROL
  END
```

By entering CONTROL, children can then guide a sprite through a maze by pressing the E, S, D, and X keys at appropriate points. Note, however, that neither the sprite nor the turtle can "sense" what is drawn on the overlay.

For schools without a printer or a version of LOGO which will save the contents of the screen, the LOGO overlay idea could be used in reverse. Once a child has produced something on the screen, it can be saved and taken home to parents by placing a blank piece of acetate (or tracing paper, for that matter) on the monitor screen and copying the drawing directly with a marker.

Also, if the drawing is particularly challenging, it can be placed on other children's monitors with an invitation to try to copy it with their turtle!
continued on page 8

TIPPS for TEACHERS

by
Steve Tipps

Patterns and Repetition II

The previous column on patterns and repetition suggested many ways which teachers can encourage students to find and describe patterns in the environment. The REPEAT command serves as the way which LOGO allows the programmer to draw patterns and consolidate understanding of direction and distance. The amount of time spent with consolidating control of the screen turtle will be rewarded as children move toward procedures.

REPEAT allows the programmer to see the results of the same turtle movement done again and again. The simplest form of this is a single command inside the brackets:

```
REPEAT 3 [ FORWARD 20 ]

REPEAT 17 [ FORWARD 50 ]

REPEAT 100 [ BACK 6.7 ]
```

The additive nature of the turtle's microworld is clearly demonstrated by these simple commands. The turtle can also be made to pivot with a REPEAT command in a game called Spin the Turtle. The only command inside the brackets is a LEFT or RIGHT command:

```
REPEAT 20 [ RIGHT 11 ]

REPEAT 3 [ LEFT 15 ]

REPEAT 300 [ LEFT 20 ]
```

As children make the turtle spin they find that combinations of very large numbers only result in the same directions as smaller numbers. Very large numbers are superfluous because the direction of the turtle is limited.

Some children may stumble across the magic number of 360 which is the number of turtle turns in the Total Turtle Trip Theorem. (See, for example, page 24 of Turtle Geometry by Abelson and diSessa.)

Students can get involved in guessing games about the final orientation of the turtle as a result of REPEAT statements. The ability to use repeat with direction and distance separately will increase the possibility of using them together and with other commands.

STACKING COMMANDS and DEBUGGING

Distance, direction, and pen commands are still being used in the graphics (immediate) mode, so that feedback in the form of turtle movement is immediate with hitting <RETURN>. When children first learn that they can stack commands on a line:

```
FD 20 RT 45 LT 33 PU BK 60 PC 3
PD FD 90 LT 45 PC 6 FD 70 RT 80 BK 66
FD 9 BK 91 RT 56 FD 81
```

they will often adopt this garbage approach to turtling. The initial fun of having the turtle do all these actions rapidly and in sequence can be counterproductive to using and recognizing patterns which do interesting things.

The teachers should encourage the child to look at single commands or at most two related commands on a line. This is very early debugging practice which encourages "mind sized" chunks of programming.

I would not say that a child should never stack up a line of commands. The idea that LOGO does whatever it is told in the order which is specified and that the sequence is unlimited for all practical purposes (save 255 characters on a line) are important LOGO understandings.

But children should recognize that it is usually very difficult to figure out which part of a very long line of commands was responsible for a particular part of a complex turtle drawing.

As children begin to draw things one command at a time in the graphics mode, they will begin to see that repeating the same combination of commands results in some particularly interesting designs. Closed figures and designs result if a pattern of distance and direction is created.

```
FD 38 RT 33          FD 100 LT 144
FD 38 RT 33    or    FD 100 LT 144
FD 38 RT 33          FD 100 LT 144
etc.
```

The patterns call upon the teacher to introduce REPEAT, enabling the child to explore the combinations of distance and direction commands at a faster rate.

```
REPEAT 22 [ FD 38 RT 33 ]

REPEAT 22 [ FD 100 LT 144 ]
```

The teacher must remember that s/he is not teaching formulas or giving out recipes for squares, stars or circles. Instead, s/he is only providing the format of the command which allows the students to create squares, stars and circles without

continued on page 4

typing each and every command separately. The recipe process is called "repetitive and redundant" by Glen Bull, while what you want to encourage is a process which is only "repetitive."

Collapsing many actions into a REPEAT command is the transitional step to understanding what programming is. Children will also try out putting garbage into the brackets. Again this may produce some interesting effects which the child can clean up.

My approach to this stage is to ask the child to describe or draw what is going to happen as the sequence is repeated. If they show understanding of the effect of the command and the sequence of action, I'm not concerned. They will be able to do other more coherent programming when they need to. If, on the other hand, the student seems to be mystified and is using REPEAT in some mindless way, I tend to guide them back to one or two commands in the brackets.

It would be nice to pretend that a teacher could only show REPEAT to those children who demonstrate a need for it. However, the classroom grapevine wll pass along trade secrets. REPEAT gives so much power to the beginner that they will not be denied even if they are not in complete control. The teacher's job, as I see it, is to guide their thinking toward understanding the command.

ZIGZAG: AN ILLUSTRATIVE EXAMPLE

The next step is to control more commands within REPEAT. Zigzag patterns are particularly good because they have many variations of direction and distance -- almost all of which result in pleasing designs.

I would suggest that zigzags be used to reinforce the ideas of body syntonicity and control which are at the foundation of LOGO philosophy. At a number of teacher workshops, when participants were invited to create a zigzag pattern, they began by creating a closed figure like those above.

Instead, they needed to walk through the pattern using conversation while playing turtle:

Walk forward,
turn to the right,
walk forward some more,
turn to the left.

Several of them tried to go directly to REPEAT statements rather than working out a design command by command.

```
FORWARD 50              FORWARD 40
RIGHT 39       or       RIGHT 120
FORWARD 77              FORWARD 40
LEFT 22                 LEFT 120
```

Working ideas out in the immediate mode is very important and is a prime advantage of LOGO. Only after a design is achieved should the REPEAT statement be used.

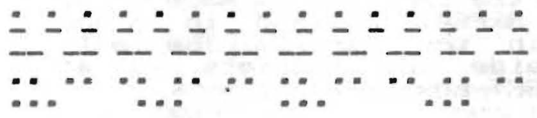
```
REPEAT 900 [ FORWARD 26 RIGHT 98
FORWARD 40 LEFT 100 ]
```

All manner of zigzags, wiggles and steps are created with the REPEAT format. The initial direction of the turtle can transform the appearance of the pattern. Overlaps and cross-hatching are often quite striking. As zigzags are created, programmers should become very familiar with the three elements of the REPEAT command and the effects of changing the repeater or the pattern.

PROJECTS WITH REPEAT

If the teacher has prepared students with pattern finding and making, and if the students are very comfortable with REPEAT statements, they should be ready to use REPEAT to create many of their own patterns.

I like to have the class create a set of patterns on tag board or sentence strips. Among the most frequent are dotted and dashed lines using PENUP, PENDOWN, and different distances in a REPEAT command.



Later, the dots and dashes could turn into Morse code or an idiosyncratic personal code which children can program.

Other designs might incorporate turtle moves such as resetting the turtle at the side of the screen, flipping it upside down, and spinning it to create a star effect or rays from a center position. (See "LOGO on Wheels" elsewhere in this issue, for example.)

Each child should keep a record of different designs and statements. Challenge designs can be posted as puzzles for other children to solve. Of course, not all the solutions will be exactly alike, but this will help the children compare their different approaches.

So far, positioning has been included in the REPEAT. However, recognition that the position and the pattern serve two different purposes

continued on page 8

MICROWORLDS

by

Glen Bull

Is LOGO a tool?

LOGO was intended to be a language with a low threshold and a high ceiling. It has been used effectively in classes ranging from kindergarten to physics courses at the university level. This is a wide range, but use of a computer language as a tool represents an even higher threshold of use.

LOGO has been used by many children to satisfy personally meaningful goals. In that sense, it is a meta-tool. It might be said to be a tool for learning how to use tools. However, this begs the question. A more meaningful question might be whether LOGO can be used as a tool in the context of adult goals in the work-a-day world, in the sense that COBOL is commonly used to develop business applications.

SOME CONSIDERATIONS

The physically and communicatively handicapped constitute a population where dramatic gains in quality of life could be achieved. The use of technology to gain control of the environment and establish communicaiton with the world is potentially one of its most humane uses.

The results of research in this area have not filtered down to the bulk of the population it will ultimately help. For example, far less than ten percent of the cerebral palsied population with communicative handicaps has access to technology potentially available to them. The limits on the real utility of a prosthetic device are dependent upon accessibility. If access to, say, eye glasses were limited by a price of $30,000 a pair, many of us would lead far less productive lives!

One key to driving down the cost of devices such as augmentative communication systems is utilization of general-purpose hardware. The economics of large-scale production runs is well known. As a matter of simple fact, a device developed for a specialized population may require an end cost ten times higher than a similar device developed for the general population.

A great deal of off-the-shelf hardware for interacting with and controlling the environment has been developed in the past two years. Devices which plug into the bus of the Apple computer, for example, control electric appliances and light switches. A reasonably intelligible speech synthesizer with an unlimited vocabulary is available for about $300. In contrast, the cost of an equivalent synthesizer in 1980 might have been $3000.

LOGO LIMITATIONS

LOGO is a language which should be rich with possibilities for interaction with the outside world. In the LOGO working papers (Bibliography of LOGO Memos, MIT LOGO Group, 545 Technology Square, Cambridge, MA 02139), there are references to dancing floor turtles, control of music boxes, use of robot arms to control yo-yo's, and other external devices. In this conception of LOGO, there should be ample capabilities for controlling general-purpose hardware which might be used by special populations.

This concept has not always survived translation into implementations on inexpensive microcomputers. In some cases, it has been necessary to sacrifice communication between LOGO and the outside world, except through the keyboard. In these instances, speech synthesizers, joysticks, and other external devices are available, but there is no way to tell LOGO about their existence.

In other implementations, it has been possible to include provisions for access to input and output (I/O) ports to the outside world. However, some of these implementations make no provision for calls to assembly language subroutines. This limitation makes it difficult for the user to develop specialized I/O drivers. To put it another way, lack of this capability places a very definite ceiling on the use of LOGO to interact with the outside world.

All current LOGO implementations on microcomputers (to my knowledge) have another constricting factor that limits use of LOGO as an adult tool: they are somewhat cramped for memory space. LOGO is more sophisticated than many languages, and hence occupies considerable memory. In fact, it was difficult to squeeze LOGO into the address space of an eight-bit microcomputer at all.

However, this necessary limitation on the size of programs which can reside in memory at a given time does hamper some of the more ambitious uses of LOGO as a tool for the handicapped.

continued on page 6

Much of the original work with LOGO was done on Digital Equipment Corporation minicomputers. It might be possible to evade some of the limitations I have mentioned by acquiring this version of LOGO. However, use of a minicomputer would defeat the purpose of increasing access to technology for the handicapped, since minicomputers carry hefty price tags.

At this stage, LOGO in some implementations on microcomputers is tantalizingly close to being an effective tool for real-world applications, but falls just short. The sensible thing would appear to be to develop these types of applications in some other language, such as Forth or Pascal.

LOGO UTILIZATION

However, there is another aspect which must be considered. The needs of the disabled vary considerably, and, in the best of all possible worlds, prosthetic devices would be customized. In fact, many such devices are customized of necessity since it is the only means of making them remotely usable. Customization in the case of a microcomputer implies an individualized program for the patient.

Currently, there are not enough engineers and programmers to go around, and there may not be for the forseeable future. This drives the cost of specialized programs up, if it is possible to obtain one for a handicapped individual at all. The cost may be less than that for equivalent individualized modifications in hardware, but the cost is still unaffordable in many applications.

It is not possible to teach clinicians to program interactive devices in Pascal. Many of them are capable of learning how, but there is not room for inclusion of this information in the curriculum. Computer scientists require several semesters of work to reach this stage. By the time a clinician took the required number of courses, it might be more sensible to go on and get a degree in computer science! Of course, some clinicians do add these classes to their regular courseload, but this is not realistic for most students.

However, LOGO is powerful enough and has a structure which permits a clinician to develop augmentative devices for the disabled in less than a semester of work. Memory limitations hinder expansion of these devices, but judicious use of the disk as virtual memory makes it clear what the potential could be.

Even when the programming concepts require prevent the clinician from completely developing a project in finished form, the process of working through some aspects provides insight into the task of the engineer. This, in turn, leads to improved communication between the clinician and the engineering team.

Currently, there are a number of 16-bit microcomputers which have expanded memory capabilities. It also would probably be easier to implement LOGO on these microcomputers, since development of a program becomes increasingly difficult as the capacities of a machine are stretched to the limit. The increased power of chips such as the Motorola 6800 microprocessor should decrease the difficulty of adapting LOGO to these systems.

On the other hand, the sophistication of LOGO makes it a complex language to develop, and hence more difficult to implement than a simpler language such as BASIC. This, in turn, decreases the probability that it will be developed on the next generation of microcomputers.

If it is developed for one of the 16-bit generation of machines, the development team may not understand the importance of making it possible for LOGO to interact with the outside world. Even if they do, technologic quirks may make it prohibitively expensive to develop some of the more desirable features of LOGO on the system adopted.

BACK TO TOPLEVEL

Thus, in answer to the question "Is LOGO a tool?" the reply must be "It almost is." or "It could be." LOGO in its abstract form as a programming philosophy foreshadows an eminently usable tool. In the practical requirements of the real world, the capabilities of 8-bit microcomputers on which LOGO has been implemented to date fall just short of providing the power required.

The limitations of memory space and insufficient processor speed are hardware restrictions rather than conceptual difficulties. Recent history suggests that hardware restrictions can be erased by technologic advances. From that point, it is a matter of creating a version of LOGO which makes use of this newfound capability. ▷

Glen Bull is a professor at the University of Virginia, and teaches LOGO courses at both the graduate and undergraduate level.

LOGO on Wheels

The power of LOGO to help in the investigation of phenomena present in the physical world was made evident to me recently.

As I was driving home one night, I saw ahead of me a strangely-behaving light crossing the road. It bobbed and weaved up and down in a herky-jerky fashion which commanded my attention.

It turned out to be a reflector mounted on the spokes of a bicycle wheel. (Whew!) Aside from making sure that I saw him, the bike rider also gave me something to think about.

What if a child observed the same motion and wanted to investigate it further with LOGO? How might the process take place?

By observing a bicycle wheel in slow motion, a child might note that the center of the wheel moves smoothly in the direction of travel while the spokes, reflector, rim, and tire rotate around it. The first challenge might be to make the turtle move like the axle.

By playing turtle, the child discovers that, by moving a little toward an object, and then turning some, then moving toward the object again (even if not facing the object), and so on , something close to the desired motion is produced. A spinning dancer crossing a room has a similar motion.

```
TO MOVE :STEP :TURN
     PENUP
     RIGHT :TURN
     SETX XCOR + :STEP
     MOVE :STEP :TURN
     END
```

The above MIT LOGO (Terrapin/Krell) listing might be one way to show this. MOVE 5 5 will make the turtle behave as if it were glued to the axle of an invisible wheel which is rolling across the screen. The turtle spins (rotates) and moves linearly (translates) in a uniform manner.

However, the bicycle reflector was not on the axle, but near the rim. How could this be represented with LOGO?

The turtle might mark the <u>location</u> of the reflector at successive moments. The turtle on the axle could be imagined to be facing the reflector at all times. Thus, it could dart forward, mark where the reflector is, and return to the axle in time to turn through the angle.

```
TO DOT
     PENDOWN
     FORWARD 1
     BACK 1
     PENUP
     END
```

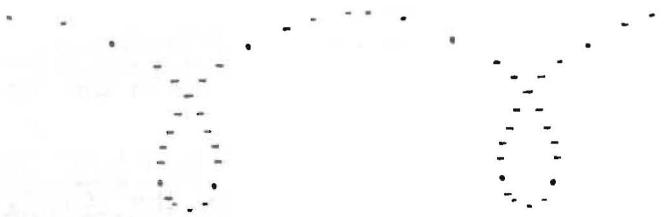
The above listing will make the turtle draw a dot at its location. Now to have the dot drawn at the location of the reflector.

```
TO REFLECTOR :DISTANCE
     PENUP
     FORWARD :DISTANCE
     DOT
     BACK :DISTANCE
     END
```

The REFLECTOR procedure needs to know how far the reflector is from the axle. This is designated as the variable :DISTANCE. Now let's integrate what we have done into the MOVE procedure.

```
TO MOVE :STEP :TURN :DISTANCE
     REFLECTOR :DISTANCE
     RIGHT :TURN
     SETX XCOR + :STEP
     MOVE :STEP :TURN :DISTANCE
     END
```

MOVE 5 10 40 will draw a picture similar to this:



This is just about what the path of the bike reflector looked like. The child has taken a reasonably complicated physical situation and has represented it well and quickly with LOGO in familiar terms.

Additionally, there has been ample opportunity for exploration along the way. Extremely interesting pictures and patterns can be generated by varying the parameters. It is also fun to choose values for the parameters, sketch out the predicted result, and then run the procedure. There are usually a few surprises!

But the story does not have to end here. The path drawn above is <u>not</u> exactly correct. Assuming the bike wheel rolls without slipping, there is a relationship between the angle and the horizontal distance the wheel travels in each cycle of MOVE.

continued on page 8

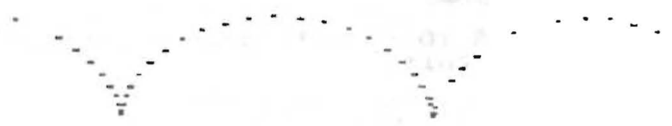
Wheels continued

In particular, the incremental horizontal distance (called :STEP here) is equal to the product of the wheel radius times the angle of the incremental turn (which must be expressed in radians instead of degrees).

For simplicity, let's assume the reflector is located right on the rim of the wheel (that is, :DISTANCE is the wheel's radius). Thus, if MOVE is rewritten accordingly, it would look like:

```
    TO MOVE1 :STEP :DISTANCE
        REFLECTOR :DISTANCE
        MAKE "TURN (:STEP * 180)/
(:DISTANCE * 3.14159)
        RIGHT :TURN
        SETX XCOR + :STEP
        MOVE1 :STEP :DISTANCE
        END
```

Now we have a more accurate representation of the actual physical situation. MOVE1 5 20 will draw a rather nice looking curve similar to the one below.



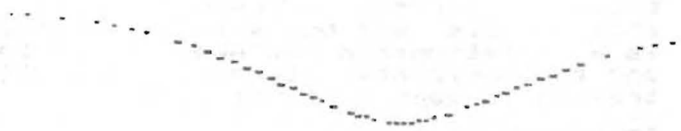
This is the path of a point on a rolling circle, one of a family of curves called CYCLOIDS which can be generated with MOVE1.

A further refinement might be in order. The reflector is usually mounted on the spokes of the wheel, not on the rim. Thus, the :DISTANCE from the axle to the reflector is less than the radius of the wheel.

By using the variable :RADIUS, we can write MOVE3 (my MIT LOGO would not recognize MOVE2 for some unknown reason):

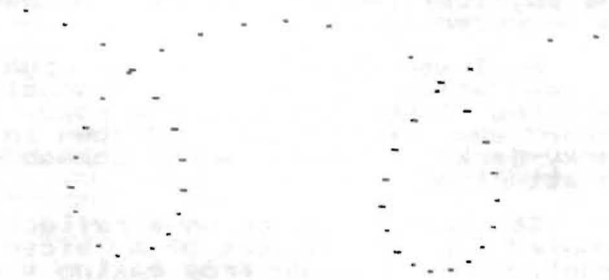
```
    TO MOVE3 :STEP :DISTANCE :RADIUS
        REFLECTOR :DISTANCE
        MAKE "TURN (:STEP * 180)/
(:RADIUS * 3.14159)
        RIGHT :TURN
        SETX XCOR + :STEP
        MOVE3 :STEP :DISTANCE :RADIUS
        END
```

By keeping the :DISTANCE less than the :RADIUS, we can get smoother patterns such as MOVE3 5 20 50:



continued

But, now we can investigate an "un-physical" situation in which the reflector is mounted on the wheel at a :DISTANCE greater than the :RADIUS! MOVE3 5 50 20 produces a picture thus:



Many everyday situations such as this are readily adaptable to investigation with LOGO. Look around. You might surprise yourself. Or, better yet, challenge your children to look around! And get ready!

FORWARD 100! ...... ▷

Overlays continued

The potential of this idea is limited only by the imagination of the teacher and the children. Let us know of any extensions or special applications you invent or use which you would like to share with others.

A special thanks to Mrs. Blitman for sharing this idea. She credits Theresa Overall of the Lamplighter School in Dallas, TX, as the original source of the LOGO overlay. ▷

Tipps continued

will be important as children move toward procedures and more modular programming.

Children who are skilled with REPEAT conceptually and functionally should be ready for the great leap into creating procedures. But, using REPEAT in the immediate mode is a programming and debugging skill which should not be discarded. ▷

Steve Tipps is a professor at the University of Virginia, and conducts LOGO workshops for teachers throughout the eastern United States.

With LOGO, the child becomes the teacher, and the teacher, the child.