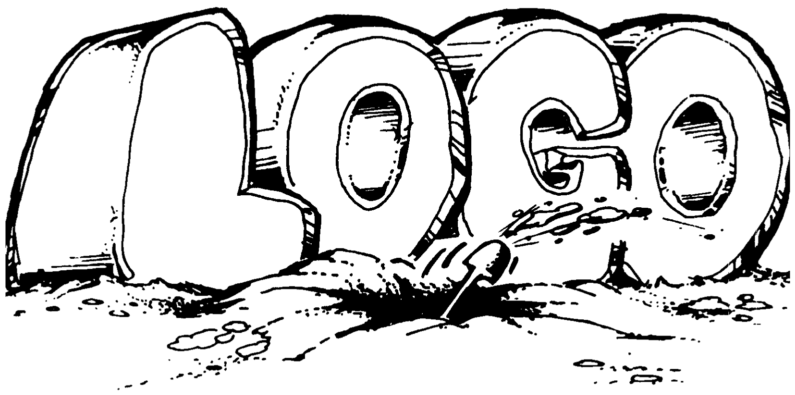
Chapter 13. What’s Next

Believe it or not, you’ve just started to dig into Logo.



There is so much more to discover. But now you have a few tools to use on your adventure. But before we go, here’s a few more things to think about.

From Two to Three Dimensions

Here's a challenge for you…put the soccer ball pattern on the screen.

The first thing you see, looking at a soccer ball, is a bunch of hexagon shapes. When we asked some young computer club members to draw this pattern on the screen, they thought it was easy…

```
TO SOCCER.BALL :DIS
REPEAT 6 [REPEAT 6 [FD :DIS RT 60] FD :DIS LT 60]
END
```

The boys thought that all they had to do was draw a series of hexagons. Try their SOCCER.BALL procedure. It's not quite right, is it?

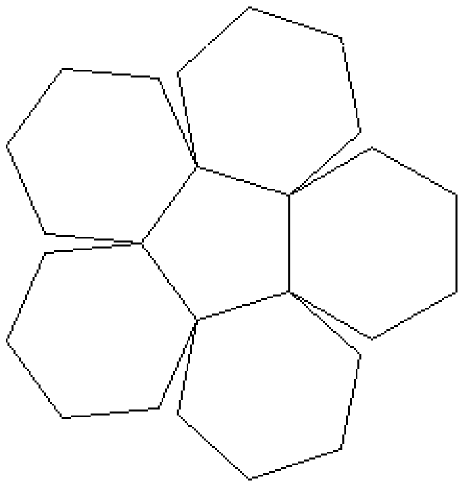
The girl's team was the first to figure out that they could not make the soccer pattern on the screen as it appears on the ball. They had to flatten it out.

At first, they thought this procedure was wrong. But it was really correct.

```
TO SOCCER :DIS
REPEAT 5 [REPEAT 6 [FD :DIS RT 60] FD :DIS LT 72]
END
```

The girls printed twelve of their patterns, colored them, taped them together, and made their own soccer ball. When they were finished, they decided it made a better pinata.

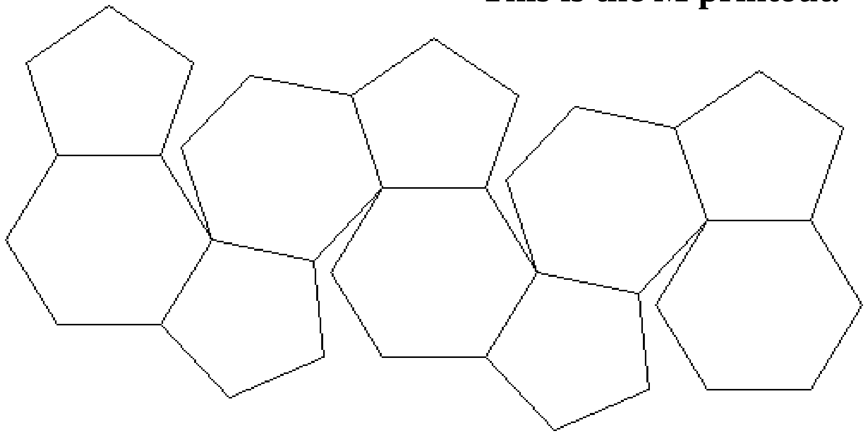
So they filled it with candy and had a party.



When this story was published in *Turtle News*, different groups and classes around the world had fun with it. Here's one young person's response. His challenge was to reduce the pattern to the fewest number of parts. Two was the best he could do.

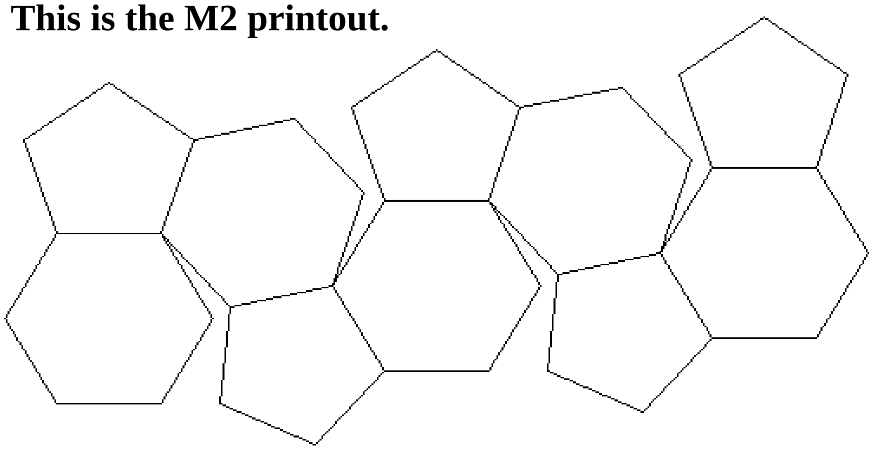
```
TO ADAM'S.SOCCER.BALL
CT
PR [This Logo procedure is from Adam Johnson,]
PR [age 12, The Computer Learning Center,]
PR [Caldwell, Idaho 83605.]
END

TO C
RT 120 FD 30 LT 60
END
```

This is the M printout.



```
TO F
LT 120 FD 60 RT 60
END

TO H
REPEAT 6 [FD 60 RT 60]
END

TO K
FD 60 RT 72 FD 60 LT 180
END
```

This is the M2 printout.



```
TO K2
LT 72 BK 60 RT 108
END
TO L
FD 60 RT 60 FD 60 RT 42
END

TO M
RT 90 PU FD 180 PD
H FD 60 LT 90 P
K H L P K2 H P2 P
K H L P K2 H P2 P
END

TO M2
PU LT 90 FD 240 RT 180
PD H FD 60 LT 90
P RT 108 H P2 P K H
L P K2 H P2 P K H L P
END
```

```
TO P
LT 90
REPEAT 5 [FD 60 RT 72]
END

TO P2
LT 60 BK 60 RT 210
END

TO TOP
P RT 108 H RT 120
H C H C H C H
END
```

Rabbit Trail 21. Folded Paper Fun

Making the soccer ball out of paper is just one of many things you can do with MSW Logo and paper. You can do all sorts of things with three dimensional objects.

How about a simple cube? This takes you from the two dimensional square to a three dimensional cube.

```
TO CUBE  :D :X1 :Y1
PU SETXY :X1 :Y1 SETH 0 PD
REPEAT 4 [SQUARE :D SETX :X1 + :D MAKE “X1
     XCOR]
MAKE “X1 XCOR - ( :D * 3 )
MAKE “Y1 YCOR - :D
PU SETXY :X1 :Y1 PD
REPEAT 3 [SQUARE :D FD :D]
HT
END
```

```
TO CUBES  :D :X1 :Y1
CUBE :D :X1 :Y1
CUBE :D :X1 + ( :D * 4 ) :Y1
END

TO SQUARE  :D
REPEAT 4 [FD :D RT 90]
END
```

What about three dimensional shapes using triangles?

```
TO MOVEL  :D
LT 60 FD :D RT 60
END


TO TETRAHEDRON  :D
RT 30 TRI :D MOVER :D TRI :D
MOVEL :D TRI :D
END

TO MOVER  :D
RT 60 FD :D LT 60
END

TO TRIS  :D
PU SETX - 120 PD
RT 30
REPEAT 6 [TRI :D MOVER :D]
END

TO TRIR  :D
RT 60 FD :D TRI :D
END
```

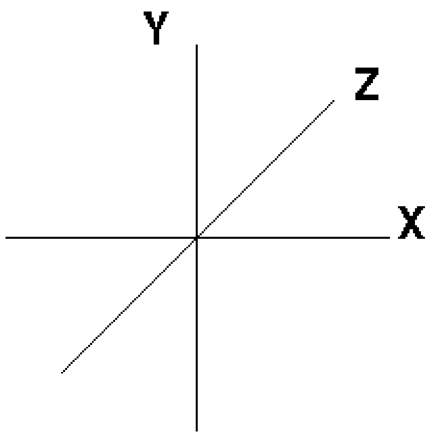
```
TO TRI  :D
REPEAT 3 [FD :D RT 120]
END

TO OCTAHEDRON  :D
LT 30 TRI :D RT 30 TETRAHEDRON :D
LT 60 TRI :D TRIR :D TRIF :D
END

TO TRIF  :D
FD :D RT 60 TRI :D
END
```

These procedures are just the beginning of what you can do with Logo and a printer. Go ahead. Try these. Print them. Fold them up. And then design your own figures.

More 3-D Logo

When some junior high students saw the work that the third grade students had done creating and folding soccer ball patterns, they wondered if it would be possible to work in three dimensions on the Logo screen. They were thoroughly familiar with the two dimensions of the x - y coordinate system.

Could this be expanded to serve three dimensions: X, Y, and Z?

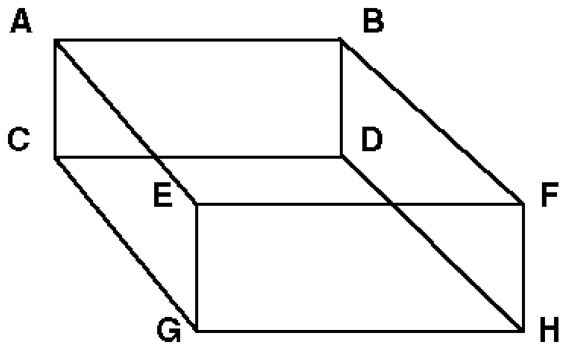


Yes, it can.

In addition, the resulting procedure offers a good look at property lists, an often confusing feature of Logo.

In the procedure listed below, the basic unit is the coordinate point as defined by the POINT procedure. Points have letter names and x, y, and z coordinates to position them in three-dimensional space.

Once you have defined all the required points by name and position, you can construct shapes like this.



The FIGURE procedure takes the shape name and a list of two-point lists; for example, [[A B][A C][A E][B F][B D][C D][C G]…]. The two-point lists represent the line segments of the shape with each letter representing an endpoint.

The procedure allows you to create as many shapes as you want, but only one can be manipulated at a time.

Once you have defined your shape, you can expand it or contract it, rotate it, magnify it, shrink it, and then restore it to its original shape. To expand a shape, use EXPAND. Tell the procedure which shape to expand, which axis the expansion will operate on, and how much to expand it.

MAGNIFY is very similar to EXPAND. However, you don't specify an axis since the figure is magnified in all directions.

ROTATE operates on a plane, xy, xz, or yz. Specify the shape, the plane, and the degrees of rotation you want to see.

As you move your shape through space, the turtle remembers the position of your shape and moves it from its last position. When you want to start over with a new shape, or start from your shape's original position, use…

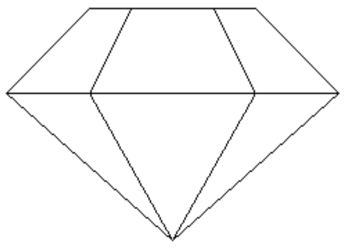
RESTORE "<*figure name*>.

To get you started, there are two examples provided in the procedure below: a diamond and an econobox-like car. Type DIAMOND or CAR to see a front view of the figures. Then rotate the figures using commands such as:

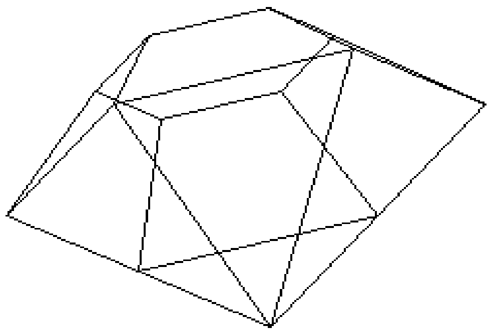
```
ROTATE "<figure name> "XY 45
ROTATE "<figure name> "YZ 30
```

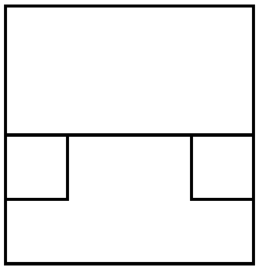
Now you're ready to start off on your own.

```
TO DIAMOND
POINT "A [60 0 60]
POINT "B [0 51.961 51.961]
POINT "C [30 51.961 103.92]
POINT "D [90 51.961 103.92]
POINT "E [120 51.961 51.961]
POINT "F [90 51.961 0]
POINT "G [30 51.961 0]
POINT "I [30 81.961 51.961]
POINT "J [45 81.961 77.941]
POINT "K [75 81.961 77.941]
POINT "L [90 81.961 51.961]
POINT "M [75 81.961 25.98]
POINT "H [45 81.961 25.98]
```



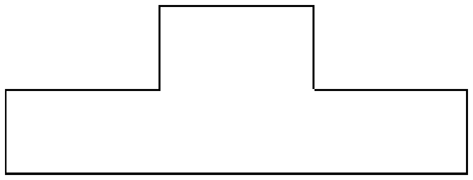
```
FIGURE "GEM [[A B] [A C] [A D] [A E] [A F] [A G]~
      [B C] [B I] [C J] [C D] [D K] [D E] [E L] [E F]~
      [F M] [F G] [G H] [G B] [I J] [J K] [K L] [L M]~
      [M H] [H I]]
END
```





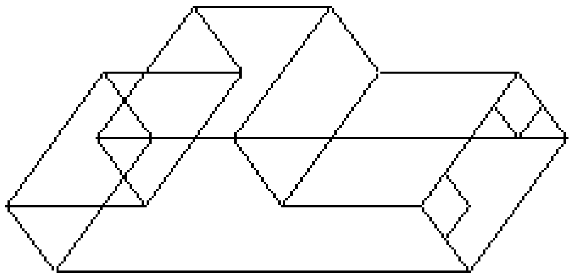
Front View

```
TO CAR
POINT "A [0 0 0]
POINT "B [0 20 0]
POINT "C [40 20 0]
POINT "D [40 0 0]
POINT "E [0 20 40]
POINT "F [0 40 40]
POINT "G [40 40 40]
POINT "H [40 20 40]
POINT "I [0 20 80]
POINT "J [0 40 80]
POINT "K [40 40 80]
POINT "L [40 20 80]
POINT "M [0 0 120]
POINT "N [0 20 120]
POINT "O [40 20 120]
POINT "P [40 0 120]
POINT "Q [0 10 0]
POINT "R [10 10 0]
POINT "S [10 20 0]
POINT "T [40 10 0]
POINT "U [30 10 0]
POINT "V [30 20 0]
```



ROTATE "AUTO "XZ 90

```
FIGURE “AUTO [[A B] [A D] [A M] [B C] [S R] [R Q]~
        [B E] [C D] [T U] [U V] [C H] [D P] [P O] [P M]~
        [O L] [O N] [N M] [N I] [I J] [I L] [J F] [J K] [K L]~
         [K G] [G H] [G F] [F E] [E H]]
END
```



ROTATE "AUTO "XY 45
ROTATE "AUTO "XZ 45

```
TO POINT :POINTNAME :COORDS
MAKE :POINTNAME :COORDS
PPROP :POINTNAME “POINT “TRUE
PPROP :POINTNAME “ORIG :COORDS
END

TO FIGURE :SHAPENAME :LP
IF (GPROP :SHAPENAME “POINT) = “TRUE~
        [(PR :SHAPENAME [IS ALREADY A POINT~
        NAME.]) STOP )]
MAKE :SHAPENAME :LP
PPROP :SHAPENAME “FIGURE “TRUE
MAKE “SX (WORD :SHAPENAME “PTS)
MAKE :SX [] MAKE “N9 1
REPEAT COUNT :LP [TSF :N9 TSL :N9~
        MAKE “N9 :N9 + 1]
MAKE “MATRIX [1 0 0 0 1 0 0 0 1]
DRAW:SHAPENAME
(PR :SHAPENAME [IS NOW A SHAPE.])
END
```

```
TO DRAW:FIGURE
MAKE “S9 THING (WORD :FIGURE “PTS)
REPEAT COUNT :S9 [MAKE “P9 FIRST :S9~
        MAKE :P9 (LIST (ITEM 1 :MATRIX) * ~
        (ITEM 1 THING :P9) + (ITEM 2 :MATRIX) *~
        (ITEM 2 THING :P9) + (ITEM 3 :MATRIX) *~
        (ITEM 3 THING :P9) (ITEM 4 :MATRIX) *~
        (ITEM 1 THING :P9) + (ITEM 5 :MATRIX) *~
        (ITEM 2 THING :P9) + (ITEM 6 :MATRIX) *~
        (ITEM 3 THING :P9) (ITEM 7 :MATRIX) *~
        (ITEM 1 THING :P9) + (ITEM 8 :MATRIX) *~
        (ITEM 2 THING :P9) + (ITEM 9 :MATRIX) *~
        (ITEM 3 THING :P9)) MAKE “S9 BF :S9]
CS ST MAKE “S9 THING :FIGURE
REPEAT COUNT :S9 [PU SETPOS BL THING ~
        (FIRST FIRST :S9) PD SETPOS BL THING~
        (LAST FIRST :S9) MAKE “S9 BF :S9]
END

TO RESTORE :FIGURE
IF NOT (GPROP :FIGURE “FIGURE) = “TRUE~
        [(PR :FIGURE [IS NOT A SHAPE.]) STOP]
MAKE “N9 THING (WORD :FIGURE “PTS)
REPEAT COUNT :N9 [MAKE FIRST :N9 ~
        GPROP (FIRST :N9) “ORIG MAKE “N9 BF :N9]
MAKE “MATRIX [1 0 0 0 1 0 0 0 1]
DRAW:FIGURE
END

TO ROTATE :FIGURE :AXIS :AMT
IF NOT MEMBERP :AXIS [XY XZ YZ] [PR ~
        [THE AXIS MUST BE XY, XZ, OR YZ.] STOP]
IF NOT (GPROP :FIGURE “FIGURE) = “TRUE~
        [(PR :FIGURE [IS NOT A SHAPE.]) STOP]
```

```
IF :AXIS = “XY [MAKE “MATRIX (LIST~
        (COS :AMT) 0 - (SIN :AMT) 0 (SIN :AMT) ~
        (COS :AMT) 0 0 0 1)]
IF :AXIS = “XZ [MAKE “MATRIX (LIST ~
        (COS :AMT) 0 0 - (SIN :AMT) 0 1 0 (SIN :AMT)~
        0 (COS :AMT) 0)]
IF :AXIS = “YZ [MAKE “MATRIX (LIST 1 0 0 0 ~
        (COS :AMT) 0 - (SIN :AMT) 0 (SIN :AMT) ~
        (COS :AMT))]
DRAW:FIGURE
END

TO MAGNIFY :FIGURE :AMT
IF NOT (GPROP :FIGURE “FIGURE) = “TRUE ~
        [(PR :FIGURE [IS NOT A SHAPE.]) STOP]
MAKE “MATRIX (LIST :AMT 0 0 0 :AMT 0 0 0 :AMT)
DRAW :FIGURE
END

TO EXPAND :FIGURE :AXIS :AMT
IF NOT MEMBERP :AXIS [X Y Z] [PR ~
        [THE AXIS MUST BE X, Y, OR Z.] STOP]
IF NOT (GPROP :FIGURE “FIGURE) = “TRUE ~
        [(PR :FIGURE [IS NOT A SHAPE.]) STOP]
IF :AXIS = “X [MAKE “MATRIX ~
        (LIST :AMT 0 0 0 1 0 0 0 1)]
IF :AXIS = “Y [MAKE “MATRIX ~
        (LIST 1 0 0 0 :AMT 0 0 0 1)]
IF :AXIS = “Z [MAKE “MATRIX ~
        (LIST 1 0 0 0 1 0 0 0 :AMT)]
DRAW :FIGURE
END
```

```
TO TSL :N9
IF NOT MEMBERP LAST (ITEM :N9 :LP) ~
        THING :SX [MAKE :SX FPUT LAST ~
        (ITEM :N9 :LP) THING :SX]
END

TO TSF :N9
IF NOT MEMBERP FIRST (ITEM :N9 :LP) ~
        THING :SX [MAKE :SX FPUT FIRST ~
        (ITEM :N9 :LP) THING :SX]
END
```

Understanding Property Lists

At the beginning of each Apple and IBM Logo procedure is a line that for the longest time, seemed to make no sense whatsoever…

```
PPROP ".SYSTEM "BURY "TRUE
```

Logy and Morf soon found that this line could be read as…

Put the PROPerty of BURY, which has the value of TRUE, with the name .SYSTEM. For those versions of Logo that do not include property list primitives, they can be written as procedures.

```
TO PPROP :NAME :PROPERTY :VALUE
MAKE (WORD :NAME CHAR 32 :PROPERTY)~
        :VALUE
END
```

For the line above, the procedure combines .SYSTEM, CHAR 32, and BURY to make the word ".SYSTEM BURY." The word is then given the value of TRUE.

Another way to look at the PPROP procedure is to use some more familiar terms.

```
PPROP "MUSIC "COUNTRY "GUITAR
PPROP "MUSIC "ROCK "LOUD
PPROP "MUSIC "CAROLS "CHRISTMAS
```

Put the property, COUNTRY, which has the value of GUITAR, with the name, MUSIC. Put the property, CAROLS, which has the value of CHRISTMAS with the name, MUSIC. The other example speaks for itself.

Now that you have the properties defined, what can you do with them? For one thing, you can recall them using this procedure…

```
TO GPROP :NAME :PROPERTY
OUTPUT THING (WORD :NAME CHAR 32~
    :PROPERTY)
END
```

This outputs the THING (the value) defined in the PPROP procedure. For example:

```
GPROP "MUSIC "COUNTRY
```

results in [GUITAR]. The 3-D procedures are one good use of property lists. Play around with them on your own. You're bound to find other uses.

Bury and Unbury

BURY is one of those Logo primitives that is often ignored. But it can be very useful.

Let's try something.

1. **Load any procedure.**
2. **Type BURYALL and press Enter.**
3. **Type EDALL and press Enter.**

Where'd the procedures go?

4. **Try to run the buried procedure. What happened?**
5. **Now load another procedure.**
6. **Type UNBURYALL and press Enter.**
7. **Type EDALL and press Enter.**

Both the procedures are now visible in the Editor, aren't they?

What this means is that you can bury certain conditions and then erase everything else. If you are writing an adventure game, you can have your character carry things from one situation…one procedure…to another.

Experiment with these commands. You'll find lots of uses for them.

Logo and AI

The whole idea of artificial intelligence gets very confusing. It makes you wonder…what is intelligence? And how can it be artificial?

One of the things that makes humans intelligent is the ability to learn. And if we can teach a computer to learn, then maybe it's intelligent.

But computers don't really learn. It is the programs that computers run that make them look like they can learn. So the learning is really kind of artificial.

So let this be our short and simple definition of artificial intelligence.

Have you ever played the game States and Capitals? Someone names a state. You have to name the capital of that state.

Here’s a game of States and Capitals that shows how the computer can look like it’s learning.

```
TO ASK
MAKE “ITEM ((RANDOM COUNT :SLIST) + 1)
MAKE “CAP PICK :ITEM :SLIST
PRINT SENTENCE [What's the capital of] FIRST :CAP
END

TO CHECKANSWER
MAKE “CHECK READLIST
IF EMPTYP :CHECK [OP “TRUE]
IF :CHECK = [QUIT] [OP “FALSE]
TEST :CHECK = LAST PICK :ITEM :SLIST
IFTRUE [PRINT (SENTENCE [That's right,] :NAME)]
IFFALSE [PRINT (SENTENCE “Sorry,~
        :NAME “It's LAST PICK :ITEM :SLIST)]
END

TO GREET
CT PRINT [What's your name?]
MAKE “NAME READLIST
PRINT (SE “Hi, :NAME “. [Welcome to States
        and Capitals!])
PR [I'll tell you the name of a state.  You tell me the
capital.]
PR [To end the game, type QUIT.]
PR [Do you want to add more states and capitals,
        or play the game (Add/Play)?]
MAKE “REP FIRST RL
IFELSE :REP = “A [SETUP][QUIZ]
END

TO INIT
MAKE “SLIST []
END

TO PICK :N :LIST
TEST :N = 1
IFTRUE [OUTPUT FIRST :LIST]
IFFALSE [OUTPUT PICK (:N - 1) (BUTFIRST :LIST)]
END

TO QUIZ
ASK CHECKANSWER QUIZ
END

TO SETUP
; SETUP erases any States and Capitals you have used
; before and allows you to teach the system a new set of
; facts.  Maybe you want to change the game from States
; and Capitals to Countries and Capitals.  Or maybe to
; famous husbands and wives.
CT PR [Do you want to erase the current list?]
MAKE “ANS FIRST READLIST
IFELSE :ANS = “Y [INIT TEACH][TEACH]
QUIZ
END

TO START
GREET QUIZ
END
```

```
TO TEACH
CT
PRINT [What is the state?]
MAKE “QUEST READLIST
PRINT []
PRINT (SENTENCE [What is] :QUEST ['s capital?])
MAKE “ANSWER READLIST
MAKE “GROUP []
MAKE “GROUP LPUT :QUEST :GROUP
MAKE “GROUP LPUT :ANSWER :GROUP
CLEARTEXT
TYPE (SE [The state is] :QUEST “. CHAR 32 )]~
        PR (SE [The capital is ] :ANSWER “.)
PRINT [Shall I add them to the list ( Y or N ) ?]
MAKE “ANS FIRST READLIST
IF :ANS = “Y [MAKE “SLIST LPUT :GROUP :SLIST]
PRINT []
PRINT [< < < < NEW LIST > > > >]
PRINT []
SHOW :SLIST
PRINT []
PRINT [More to add (Y or N)?]
MAKE “ANS FIRST READLIST
IF :ANS = “Y  [TEACH]
END
```

By now you should have little if any trouble figuring out how this procedure works. Sure, it might take some time. But you can do it.

The main feature of this game is lists within lists within lists.

First, there is the list of States and Capitals…SLIST.

Secondly, there is a list that matches each state with its capital…GROUP.

Thirdly, there is a list of each state…QUEST…and one for each capital…ANSWER.

Together, these look like this…

```
MAKE "SLIST [[[Oklahoma] [Oklahoma City]] ~
    [[New York] [Albany]]~
    [[Texas][Austin]]~
    [[Massachusetts][Boston]]~
    [[California][Sacramento]]]
```

Logo "learns" new states and capitals from the TEACH procedure.

The first thing that TEACH does is ask you to create the variables, :QUEST and :ANSWER. It then creates a new empty list named GROUP.

```
MAKE "GROUP []
```

Next, it adds the state (:QUEST) to the :GROUP list.

```
MAKE "GROUP LPUT :QUEST :GROUP
```

LPUT and FPUT are interesting commands. They are used to add words or other lists to a list. For example…

```
LPUT "Logo [MSW]
```
results in the list [MSW Logo].

```
FPUT "MSW [Logo]
```
also results in the list [MSW Logo].

In the case of States and Capitals, LPUT tells Logo to add :QUEST at the end of the list :GROUP. But :GROUP is empty. So does it make any difference whether you put :QUEST at the beginning or at the end of :GROUP?

Does it make any difference whether the line uses FPUT or LPUT?

```
MAKE "GROUP FPUT :QUEST :GROUP
```

Once you have the state listed, you need to add the capital.

```
MAKE "GROUP LPUT :ANSWER :GROUP
```

This line adds :ANSWER as the second list within the list :GROUP.

```
MAKE "SLIST LPUT :GROUP :SLIST
```

And finally, this line adds the list of two lists to the master list :SLIST.

Time to experiment.

Change the TEACH procedure to add a third element to the GROUP list …maybe the county of the capital or the population of the state?

How would you change the other procedures to ask about that third element?

Go ahead…try it. It's really not that hard.

Rather than look for the LAST element of the GROUP list, you might want to look for the LAST BUTLAST element…or the FIRST BUTFIRST element…or select an ITEM from a list that matches something else.

More AI Applications

Some time ago, a private school for developmentally disabled young people explained a problem they were having with Logo.

They were using an INSTANT procedure…a single-key procedure…with young children to help them focus on following instructions and solving problems. The problem was that the children got nervous with the teachers watching them all the time.

Was there a way to record the actions of the children so that the teachers would not be "invading the children's space?" The teachers could then replay the actions to see what the children did. Did they follow instructions or did they just randomly give the turtle directions?

Take a look at INSTPLAY.LGO. As children created their drawings, they also created a "History" file of all the commands they gave the turtle. These were saved along with the drawing procedures.

The difference between the History files and the drawing procedures was that the History files included *all* commands made by the children including *all* the debugging they did, *all* the lines and commands that were changed. So the teachers got an accurate record…not just a finished product.

INSTPLAY and STATES are but two very simple examples of how software can "learn" and how the learned information can be used. As you become more familiar with Logo, you'll begin to see how much larger knowledge and rule-based AI systems might be used.

But before we leave the subject of artificial intelligence, there is another realm to explore…simulations. These are ways to get the computer to act out your "What if?" wishes.

Several years ago, Logy and Morf visited Tombstone, Arizona, the town made famous by the gunfight at the OK Corral. That's where Wyatt Earp and his brothers shot it out with the Clantons.

Everything about the fight has been well-documented including where every one involved was standing, who shot first, who killed who.

Billy Clanton was the first to draw his pistol and fire. But because he was such a poor shot, Wyatt didn’t shoot Billy. He shot Billy’s brother with his first shot.

Well, Morf asked, "What if Billy hadn’t missed?"

This raised an interesting question. And so the two friends sat down at the computer and plotted the whole thing out. They made a simulation to answer their questions.

You’ve probably heard about aircraft simulators. These are computer controlled airplane cockpits that allow pilots to train in all sorts of emergency situations without ever leaving the ground.

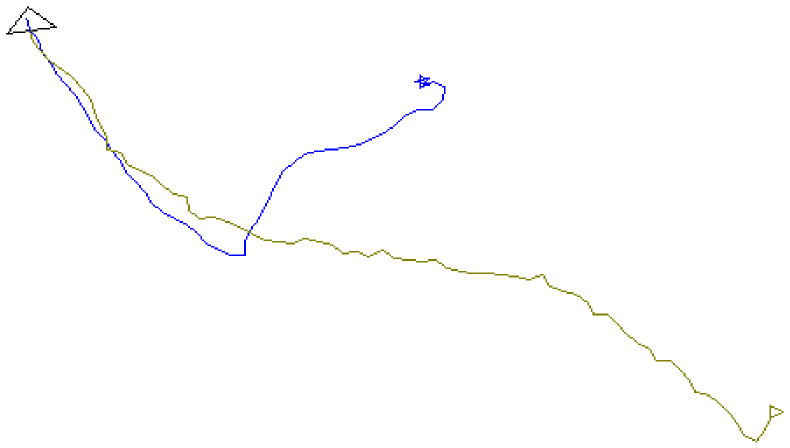
You may have gone to arcades where they have driving or flying games. These are simulators.

And, of course there are the computer flight simulator programs you can buy.

Another interesting use of simulators is in analyzing behavior. For example, how would a mouse find its way out of a maze. One way that researchers discovered was that every time this one mouse came to a wall, it turned right. It eventually found its way out.

You’ll find a behavior simulation on the disk that came with this book…BEHAVIOR.LGO. This offers three animal simulations…Find By Smell, Find by Sight, Chase and Evade.

The "Find" simulations are rather straight forward. The Chase and Evade simulation is fun. Will Find.By.Sight catch Avoid.By.Smell before Avoid can get out of the playing area?



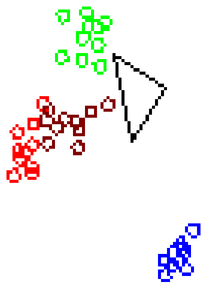
Your Challenge

BEHAVIOR.LGO uses one turtle to simulate the actions of two. Change the procedures to actually use two turtles.

Cellular Behavior

Another interesting simulation can be found in CELLS.LGO. The START procedure lists a message that was posted on CompuServe's Logo Forum. CELLS.LGO is the response.

Three groups of cells are drawn randomly on the screen. The turtle always seeks out the red cells on which additional cells are grown.



Your Challenge

Create an AVOID procedure. Currently, the turtle will move right over the green and blue cells to find the red ones. Your job is to create a procedure that makes the turtle move around the green and blue cells while still seeking the red cells.

Yes…it can be done. Give it a try.

What’s Next

Seems we just got started and here we are. And there are so many other things to do.

And what about all the other games and projects? Well, there has to be something left for you to explore on your own. You can start with the procedures in the Examples directory.

There are many procedures on CompuServe's Logo Forum that you can download and use. Some need a few changes. But you're pretty good at this by now.

This is the part of Logo that Logy and Morf like best…exploring new ways to do things…finding ways to make things work.

In fact, by now you can do just about anything you want with Logo. Doesn’t that make you feel great!

If you have any questions about MSW Logo, don't forget the Online Help file. You can print the Help file as a reference manual if you want.

And don’t forget The Logo Forum on CompuServe. The developer of MSW Logo, George Mills, is available to answer your questions.

Most important…

What ever you do, enjoy your very own

GREAT LOGO ADVENTURE!

MY JOURNAL
OR HOW I DID IT!