Computer Programming in the Schools: Emphasizing Structure—An Example Using "Inebriated Turtles"

J. Dale Burnett

Dale Burnett is a professor of computer education at the University of Lethbridge, Lethbridge, Alberta.

The following account should be interpreted at many levels. Life and "artificial life" are too complex to permit simpleminded generalizations and conclusions.

This article itself is a complex product. Obtained by incorporating one software package (Logo) into another (Word 4.0), and with structural flowcharts inserted from a third (More 3.0), it exemplifies some of the possibilities that are now within the reach of students and teachers with access to computers.

The vehicle I used for exploration was the Logo programming language. The activity I describe is thus a Logo programming exercise designed to solve the problem: How can you construct a set of Logo procedures that direct the turtle to perform a random walk within an enclosed space?

I look at random walks because this exercise provides an opportunity to develop an appreciation of randomness. What does it mean to talk about degrees of randomness? Are there different types of randomness? What do you notice when you look at a random path? Looking at randomness is also a way of exploring a new topic. How do you go about it? There are obviously many ways. What follows is a description of my first steps.

These procedures were originally written using Terrapin Logo for use on an Apple II*e*. However, the examples of screen displays that follow were created using Logo (LCSI version 1.0) on a Macintosh computer. Some changes to the procedures may be necessary depending on the version of Logo that you use. This account builds on ideas from *Turtle Geometry* by Abelson and diSessa (pp. 56-57).

The concept of randomness may be approached from either end of the continuum of randomness.. Thus one may either begin with

1. "pure" random behavior and slowly impose constraints, or
2. "pure" nonrandom behavior and slowly impose randomness.

First Approach

Here I follow the first alternative (I leave the second for your consideration).

```
TO    SIMPLE.RANDOM.WALK
  RT    RANDOM 360
  FD    RANDOM 20
  SIMPLE.RANDOM.WALK
END
```

Here are two samples of screen displays that were created by typing SIMPLE.RANDOM.WALK.

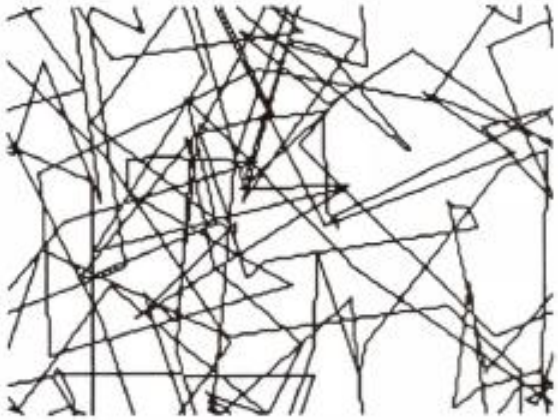
With Terrapin Logo, Ctl-G will stop the stagger. Other versions have a similar program-interrupt feature. The "endless" feature results from the third line in the SIMPLE.RANDOM.WALK procedure, where the program calls itself (forever!). The term for a program calling itself as a subprogram is *recursion.*

A principle for exploration is *wherever there is a constant, substitute a variable.*

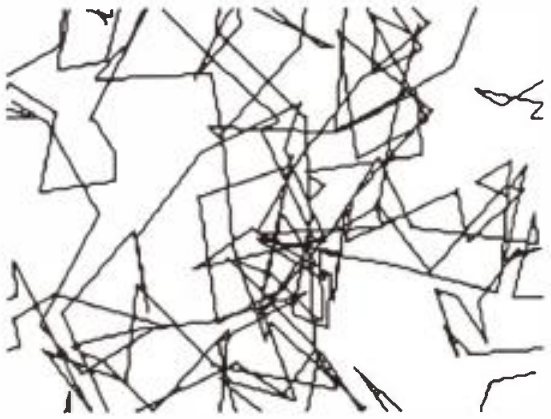
```
TO SRW     :MAX.DEGREE.TURN     :MAX.STEP.SIZE
  RT     RANDOM     :MAX.DEGREE.TURN
  FD     RANDOM     :MAX.STEP.SIZE
  SRW     :MAX.DEGREE.TURN     :MAX.STEP.SIZE
END
```

SRW stands for SIMPLE.RANDOM.WALK. Abbreviations are important when the name of a procedure must be typed repeatedly. :MAX.DEGREE.TURN and :MAX.STEP.SIZE are variable inputs. Meaningful labels for variables are also important, as they help one understand the logic of the procedures and aid in debugging. They should not be shortened. If you are not familiar with the idea of variables, refer to a Logo reference book for more information.

Try different values for :MAX.DEGREE.TURN and :MAX.STEP.SIZE. This permits you to control the degree of randomness. You may wish to save/print some pictures.

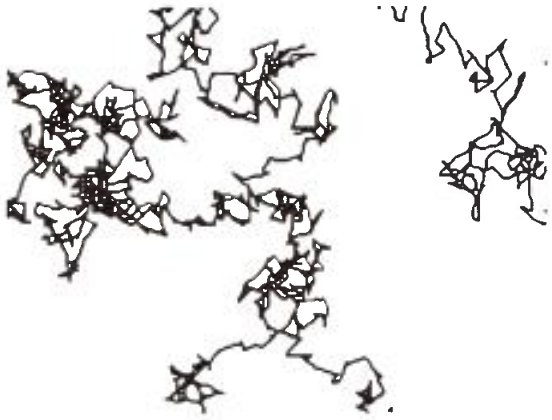


SRW 360 100

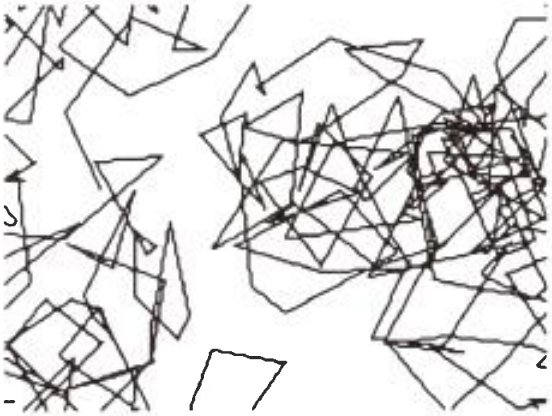


SRW 360 50

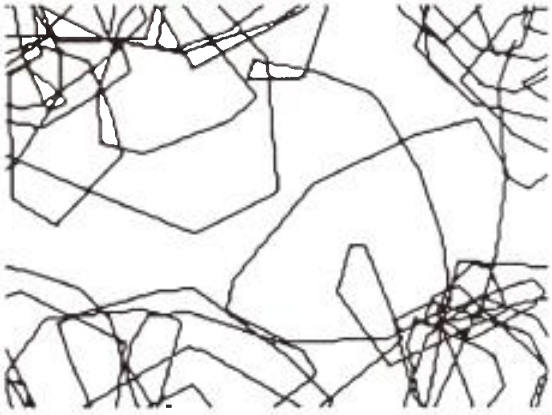
SRW 360 25



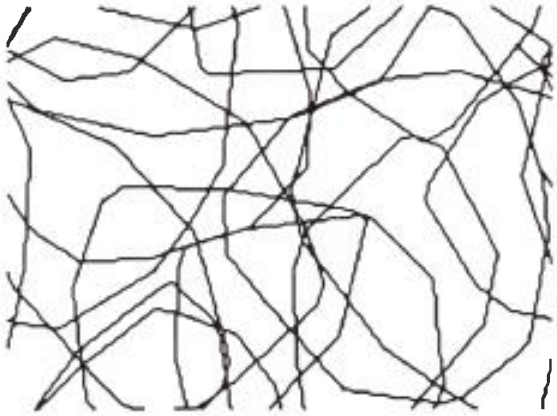
SRW 360 10



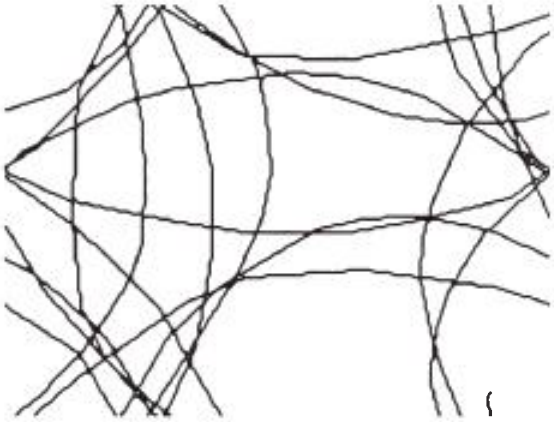
SRW 180 50



SRW 90 50



SRW 45 50



SRW 20 50

Notice how the turtle's path begins to approximate a curved line as the degree of randomness in the turn is restricted. If one were to restrict the randomness in the right turn to zero, the path would become a straight line. This example indicates a weakness in the randomness associated with the orientation of the turtle. It is better in the intuitive sense if the randomness were to fluctuate about zero (that is, permit right and left turns to occur with equal frequency) rather than about a restriction between 0 degrees and some other (positive) number. This is easily accommodated.

```
TO SRW    :MAX.DEGREE.TURN    :MAX.STEP.SIZE
  RT    ( :MAX.DEGREE.TURN / 2) - RANDOM    :MAX.DEGREE.TURN
  FD    RANDOM    :MAX.STEP.LENGTH
  SRW    :MAX.DEGREE.TURN    :MAX.STEP.SIZE
END
```

Do you appreciate what is happening in the first line of the procedure? Let's see what effect this change has on the paths of the turtle.

SRW 360 50

SRW 180 50

Do you see the difference? Do you understand the difference? Do you have a preference for one version of the SRW procedure? Why or why not? What would happen for SRW 20 50?

Here is what actually happened.

Are you surprised? Why? Why not?

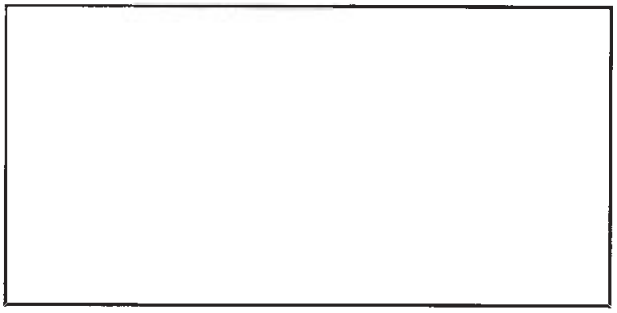
Many people find the wrap feature, where the turtle disappears off one side of the screen and reappears on the other, to be annoying during such an exploration. This can be obviated by creating a room and placing the turtle inside it.

Second Approach

Making a (rectangular) room is easy.

```
TO    DRAW.ROOM    :LEFT.SIDE    :RIGHT.SIDE    :TOP    :BOTTOM
  CLEARSCREEN
  HT
  PU
  SETXY    :LEFT.SIDE    :BOTTOM
  PD
  SETXY    :RIGHT.SIDE    :BOTTOM
  SETXY    :RIGHT.SIDE    :TOP
  SETXY    :LEFT.SIDE    :TOP
  SETXY    :LEFT.SIDE    :BOTTOM
  PU
END
```

```
DRAW.ROOM    -100    100    50    (-50)
```



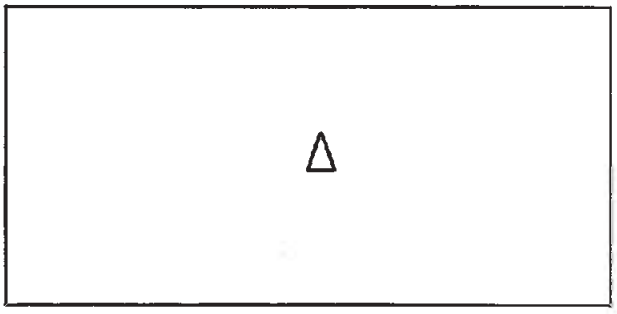
Note the use of variables! It is now easy to change the size of the room.

At first glance this is a much more cumbersome procedure than the familiar local geometry procedure of REPEAT 4 [FD 50 RT 90]. Looking ahead, we are going to need some way of testing whether or not the turtle is "hitting a wall." Logo does not have a pixel-detection primitive, so we will have to trick the computer into acting as if it had such a primitive. This can be accomplished by testing the *x* and *y* coordinates of the turtle's position against the *known* values of the boundaries of the room. Thus we want to make these coordinates explicit.

Placing the turtle inside the room is also easy.

```
TO    PLACE.TURTLE    :TURTLES.X.COORD    :TURTLES.Y.COORD
  PU
  SETXY    :TURTLES.X.COORD    :TURTLES.Y.COORD
  PD
  ST
END
```

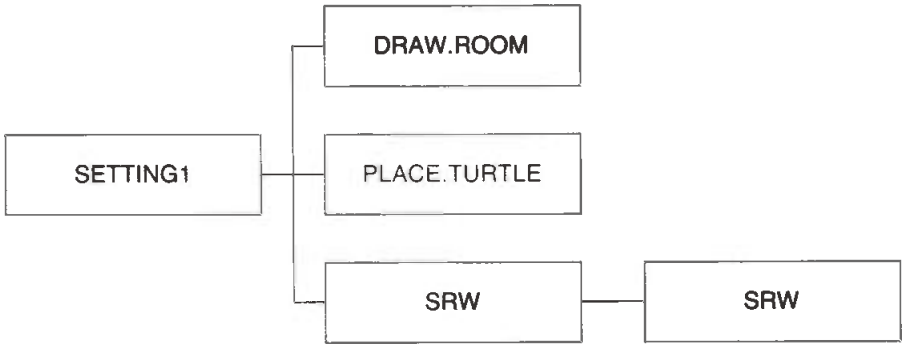
```
PLACE.TURTLE    0    0
```



Now set up a master procedure that calls these three ideas.

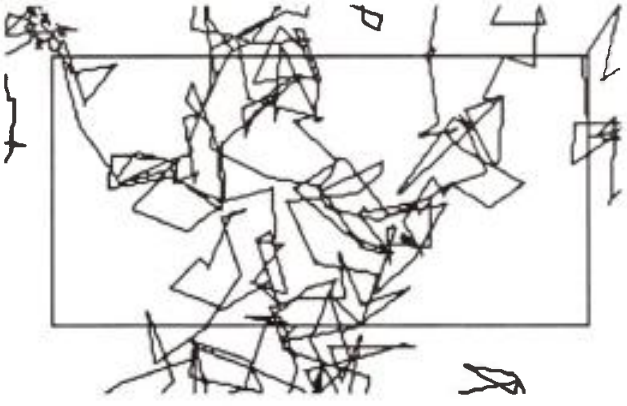
```
TO    SETTING1
  DRAW.ROOM      -100    100    50    (-50)
  PLACE.TURTLE    0      0
  SRW    360     30
END
```

It is often useful to visualize the structure of such a procedure. Here is an example of a chart that does this.



This diagram is read as follows: procedure SETTING1 calls procedure DRAW.ROOM, then procedure PLACE.TURTLE and then procedure SRW. Procedure SRW in turn calls itself (recursion). In Logo jargon, the task has been broken into "mind-size" chunks. Conceptually, the idea is to call one procedure, SETTING1, which then calls a sequence of other procedures successively to draw a room, place the turtle in it and then iterate a random walk.

Try this by typing SETTING1.



At least two problems immediately become apparent:

1. Ctl-G is a clumsy way to stop the turtle.
2. The walls have no effect!

Let's tackle the easy one first.

Third Approach

One way to stop a turtle is to have it run out of gas (alcohol?)! Thus you must alter the main procedure to include a variable indicating the number of turtle steps you want the little fellow to take.

By editing SETTING1, and changing its name to SETTING2, I am able to modify the original procedure without having to retype the entire procedure. Altering the name (by changing the suffix to 2) leaves the original procedure intact in case I wish to go back to it later. It also gives me a form of development trail as new ideas occur to me.

```
TO    SETTING2    :TOTAL.NO.OF.STEPS.TO.BE.TAKEN
  INITIALIZE
  DRAW.ROOM
  PLACE.TURTLE    0    0
  SRW2    :TOTAL.NO.OF.STEPS.TO.BE.TAKEN
END
```

The new procedure incorporates two ideas. One is maintaining a counter that will keep track of the number of steps that the turtle takes. The second is to create a separate procedure that will establish all the necessary initial values for running the procedures.

```
TO INITIALIZE
  MAKE    "NO.OF.STEPS.TAKEN    0
  MAKE    "LEFT.SIDE    (-100)
  MAKE    "RIGHT.SIDE    100
  MAKE    "TOP    50
  MAKE    "BOTTOM    (-50)
  MAKE    "MAX.DEGREE.TURN    360
  MAKE    "MAX.STEP.SIZE    30
END
```

The input variable TOTAL.NO.OF.STEPS.TO.BE.TAKEN specifies the value of the number of steps that you want the turtle to take and the variable NO.OF.STEPS.TAKEN acts as the counter that will be incremented by one each time the turtle takes a step. LEFT.SIDE, RIGHT.SIDE, TOP, and BOTTOM refer to the sides of the room. MAX.DEGREE.TURN and MAX.STEP.SIZE refer to the parameters controlling the degree of randomness the turtle will exhibit.

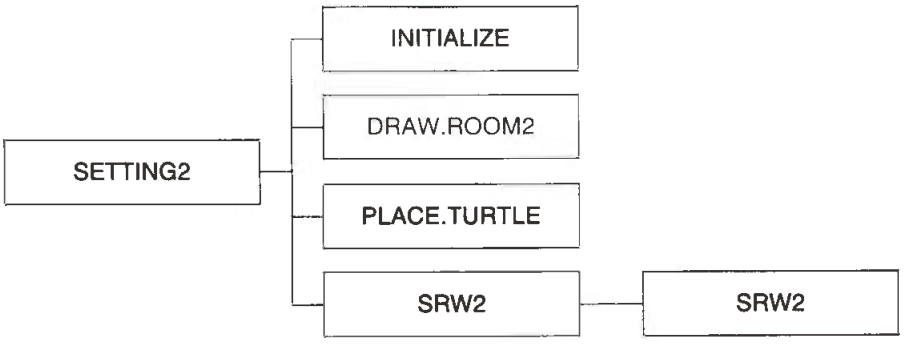
DRAW.ROOM must also be modified to change the way the room's dimensions are specified.

```
TO    DRAW.ROOM2
  CLEARSCREEN
  HT
  PU
  SETXY    :LEFT.SIDE    :BOTTOM
  PD
  SETXY    :RIGHT.SIDE    :BOTTOM
  SETXY    :RIGHT.SIDE    :TOP
  SETXY    :LEFT.SIDE    :TOP
  SETXY    :LEFT.SIDE    :BOTTOM
  PU
END
```

Finally, SRW must be modified to keep track of the number of steps that the turtle takes.

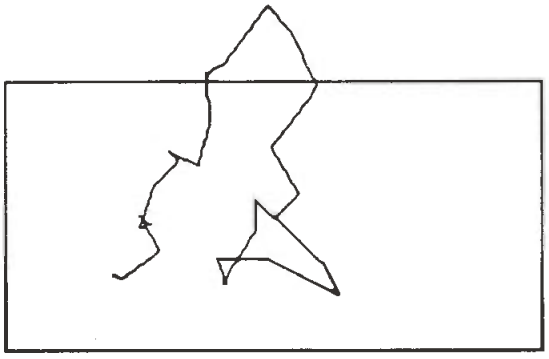
```
TO SRW2    :NO.OF.STEPS.TO.BE.TAKEN
  RT    (:MAX.DEGREE.TURN / 2) - RANDOM    :MAX.DEGREE.TURN
  FD    RANDOM    :MAX.STEP.SIZE
  MAKE    "NO.OF.STEPS.TAKEN    :NO.OF.STEPS.TAKEN + 1
  IF  :NO.OF.STEPS.TAKEN  =  :NO.OF.STEPS.TO.BE.TAKEN    STOP
  SRW2    :NO.OF.STEPS.TO.BE.TAKEN
END
```

The corresponding procedural flowchart is:



Although there has been a number of changes to the Logo procedures, the logical flow of the task remains essentially the same.

Here is an example of running SETTING2 30.



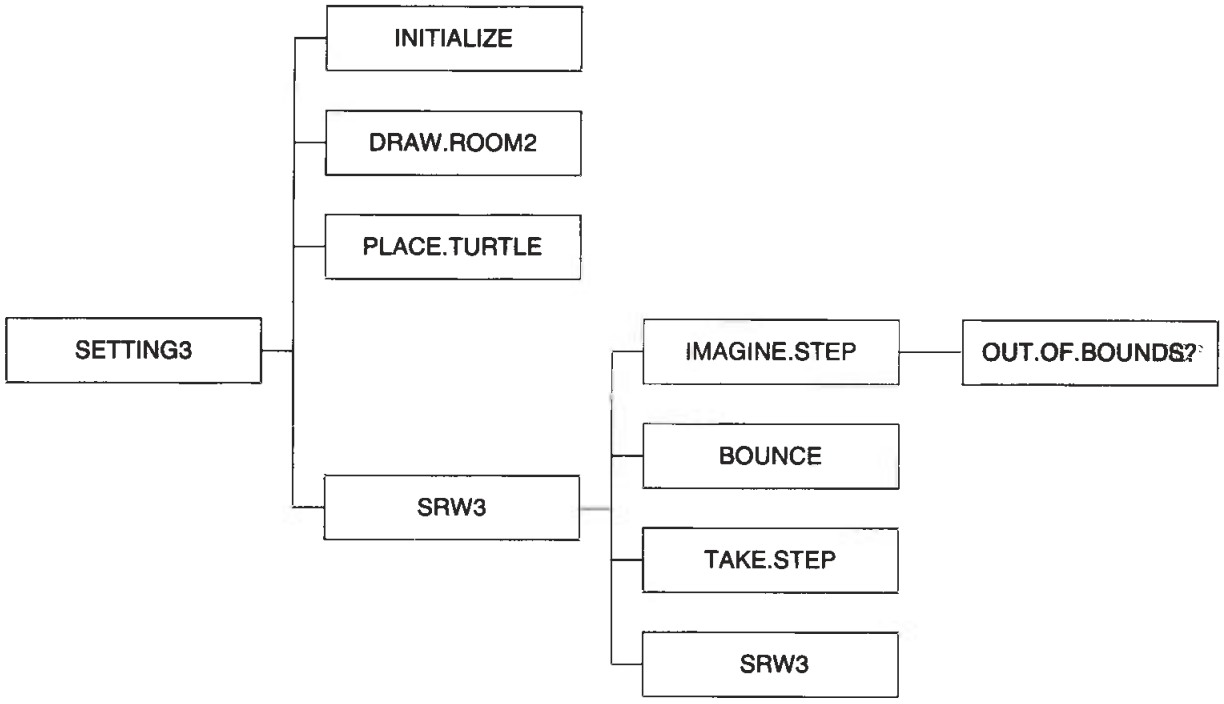
The "running out of gas" feature is working, but the walls still need more substance.

Fourth Approach

Let's now return to the problem of penetrable walls.

One way to approach the problem is to take an imaginary step first and then test if the potential position is outside the wall. If it is, you want to do something about it. Otherwise you simply take the step. Taking an imaginary step is a way of tricking the computer. The idea is to hide the turtle, put the pen up (so the turtle does not leave a trail), take a step and determine whether or not you are still inside the room. If you are, then the step will be okay, so you back up, put the pen down, show the turtle and retake the step. If you are outside the room, then do not take the step. Instead, one possibility is to back up, rotate the turtle by 180 degrees, put the pen down, show the turtle and try again.

Let's look at the structural chart first.



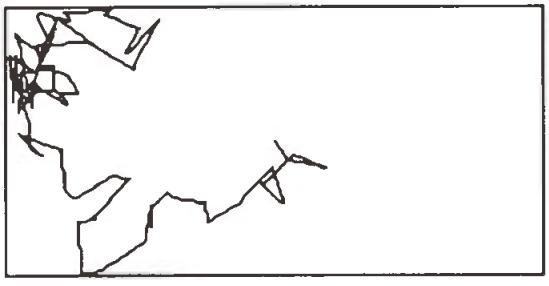
The actual procedures are

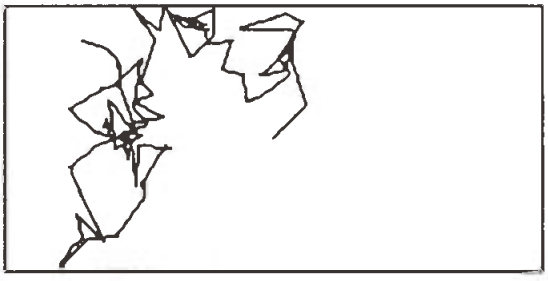
```
TO    IMAGINE.STEP
  MAKE    "PRESENT.HEADING    HEADING
  MAKE    "PRESENT.X    XCOR
  MAKE    "PRESENT.Y    YCOR
  PU
  HT
  FD    :STEP.SIZE
  MAKE    "TOO.FAR    OUT.OF.BOUNDS?
  BK    :STEP.SIZE
  PD
  SETXY    :PRESENT.X    :PRESENT.Y
  SETHEADING    :PRESENT.HEADING
END

TO    OUT.OF.BOUNDS?
  IF XCOR > :RIGHT.SIDE    OUTPUT    "TRUE
  IF XCOR < :LEFT.SIDE    OUTPUT    "TRUE
  IF YCOR > :TOP    OUTPUT    "TRUE
  IF YCOR < :BOTTOM OUTPUT    "TRUE
  OUTPUT    "FALSE
END
```

```
TO SRW3     :NO.OF.STEPS.TO.BE.TAKEN
  RT     ( :MAX.DEGREE.TURN / 2) - RANDOM     :MAX.DEGREE.TURN
  MAKE     "STEP.SIZE     RANDOM     :MAX.STEP.SIZE
  IMAGINE.STEP
  IF     :TOO.FAR     THEN     BOUNCE     ELSE     TAKE.STEP
  IF     :NO.OF.STEPS.TAKEN  =  :NO.OF.STEPS.TO.BE.TAKEN     STOP
  SRW3     :NO.OF.STEPS.TO.BE.TAKEN
END

TO BOUNCE
  RT     180
END

TO     TAKE.STEP
  FD     :STEP.SIZE
  MAKE     "NO.OF.STEPS.TAKEN     NO.OF.STEPS.TAKEN + 1
END

TO     SETTING3     :NO.OF.STEPS.TO.BE.TAKEN
  INITIALIZE
  DRAW.ROOM2
  PLACE.TURTLE     0     0
  SRW3     :NO.OF.STEPS.TO.BE.TAKEN
END
```

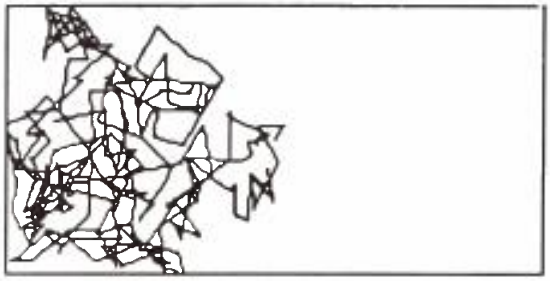
Now we can return to the original problem of studying random motion. The intensity of the randomness is determined by the two input variables :MAX.DEGREE.TURN and :MAX.STEP.SIZE to the procedure SRW3.

Here are a few sample runs of 100 turtle steps, with :MAX.STEP.SIZE 20 and :MAX.DEGREE.TURN 360.

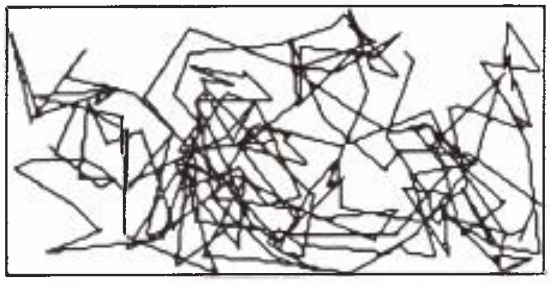


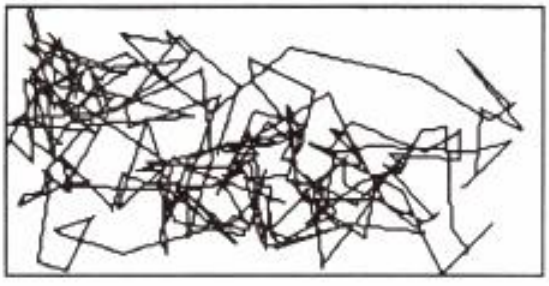


Increasing the number of steps to 300 yields



Increasing the :MAX.STEP.SIZE to 40, gives (for 300 steps)

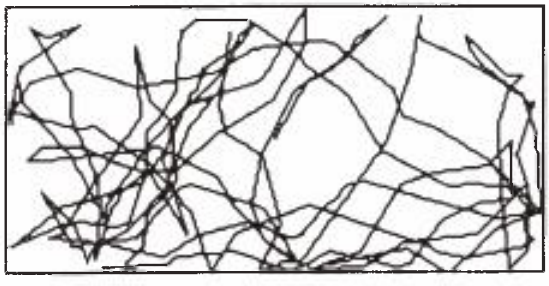




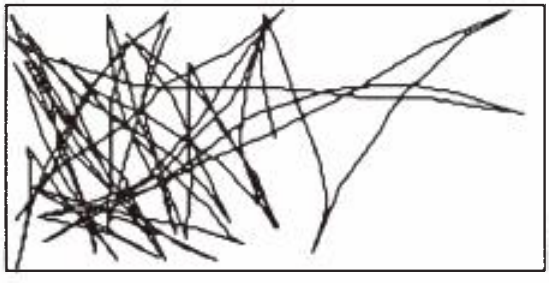
Let's try this with 10,000 steps!!

What conclusions do you draw from this?

Let's return to journeys of 300 steps, leave the :MAX.STEP.SIZE at 30 and vary the :MAX.DEGREE.TURN.



:MAX.DEGREE.TURN 90



:MAX.DEGREE.TURN 30

Additional Approaches

Another area for experimentation occurs when the turtle hits the wall. It is possible to write procedures that have different "bounce" properties. How many can you think of?

As a Logo exercise, write some procedures that have the effect of sobering the turtle up! Can different turtles have different tolerances to alcohol (or to random procedures)?

Meta-Approaches

There are many ways to investigate the turtle's behavior. One approach is simply to run the procedure repeatedly, print out the final path and then examine it. This seems like a good approach until you have 10 or 20 printouts on your lap!

Another approach is to ignore the paths and just note the destination of the turtle after a preset number of steps. You then write a meta-procedure that reruns the main procedure a large number of times, say 500 or 1,000, and examine the resulting array of dots for a pattern.

What began as an apparently simple task has exploded into a number of exciting and substantive investigations. There are at least three distinct levels of investigation:

1. To construct a set of procedures that work properly
2. To use these procedures to investigate another topic (randomness)
3. To construct meta-procedures to run the procedures many times

There is room for improvement and innovation at all levels. You may see a better way of constructing the original set of procedures. You may have a better set of questions to ask about how to examine the nature of randomness. You may have a better way of looking at meta-issues. You may even have a set of considerations for looking at this topic in a fourth (or more!) dimension.

In my case, the sequel was to examine the behavior of "hungry" turtles. One does not live by wine alone!

Postscript

As I indicated at the outset, this account may be interpreted at many levels. From a pedagogical perspective, the same software (More 3.0) that produced the structural flowcharts may be used to produce a series of screen displays suitable for presenting the Logo procedures. Such displays can facilitate class discussion of the various features the procedures incorporate. These displays may also be printed on a laser printer and used as masters for making overheads. Alternatively, with the proper acetate, overheads can be produced directly on the laser printer. Clearly such an approach can be generalized and extended to other programming tasks and languages.

Reference

Abelson, H., and A. diSessa. *Turtle Geometry.* Cambridge, Mass.: MIT Press, 1981.