www.logofoundation.org

Logo Overnight

by

Mitchel Resnick

Introduction

When most people think about Logo projects, they think of programs that take only a few seconds (or maybe a few minutes) to run. But this need not be the case. There are many interesting projects that need hours of computing time. These projects are not necessarily more complex or more difficult than standard Logo projects. They simply take a long time to run.

Running these programs during the school day could be a problem, since each program would monopolize a machine for hours. But there is an easy solution: Let the programs run through the night. Students could start running the programs when they leave school and see the results the next morning.

Logo Overnight projects are particularly useful for exploring ideas involving large numbers, estimation, extrapolation, probability, and randomness. What is more, these projects can provide insights into the scientific process. Much of experimental science involves long-running experiments. Scientists must plan an experiment, wait for it to run, and analyze the results. Then, based on the results, they make changes in the experiment and run it again. Logo Overnight projects often involve the same sort of process.

Perhaps most important, I think that Logo Overnight projects are lots of fun. It's always exciting to go the computer the next morning to see what has happened overnight.

Here are some ideas for Logo Overnight projects. Obviously, I hope that students and teachers will come up with project ideas of their own, but these examples might help them get started.

Counting Sheep
Imagine that your computer "counts sheep" at night, while you are asleep. How many sheep would the computer count overnight?

Here's one way to investigate this question. If you use the variable **sheep** to keep track of the number of sheep, then you can write a procedure like this:

```
to count-sheep
make "sheep :sheep + 1
cc show :sheep
count-sheep
end
```

To start the experiment, first initialize the **sheep** variable (type **make "sheep 0** in the command center). Then start the **count-sheep** procedure.

Can you estimate the number of sheep the computer will count overnight? Does it make any difference whether the computer prints out each number as it is counting?

Monkeys at the Keyboard
If a bunch of monkeys typed randomly at computer keyboards for a long time, what are the chances that one of them will luckily type out Hamlet?

This Logo Overnight project will give you some sense of how likely (or unlikely) it is for the monkeys to succeed. Instead of actually getting a bunch of monkeys, you can write a Logo program to randomly generate words. And it makes sense to start with an easier task: Rather than trying to get the computer to type out all of Hamlet, how long will it take the computer to randomly type a single word, like "cat"?

The program for matching "cat" could look something like this:

```
to monkeys
make "letter1 pick-letter
make "letter2 pick-letter
make "letter3 pick-letter
make "guess (word :letter1 :letter2 :letter3)
cc show :guess
make "number-of-guesses :number-of-guesses + 1
if :guess = "cat [show :number-of-guesses stop]
monkeys
end
```

The pick-letter subprocedure should report a random letter from the alphabet. Here's one way to write the procedure, based on the fact that the letters a through z have "ascii values" of 97 through 122:

```
to pick-letter
output char 97 + random 26
end
```

To start the experiment, first initialize the number-of-guesses variable to 0 by typing the instruction **make "number-of-guesses 0**, then run the **monkeys** procedure.

One time, I helped a group of fourth-grade students run this experiment. They watched as the computer printed one three-letter "word" after another: "yhe", "rwd", "urt", and so on. We kept waiting to see the word "cat". Suddenly, the computer printed the word "dog". All of us had the sensation that the computer must be getting close!

If you run the experiment more than once, will you get similar results? If you run the experiment 100 times, what is the average number of guesses? What is the maximum? The minimum? If the computer tries to guess a four-letter word (like "logo"), rather than a three-letter word, how many guesses does it need (on average)? What's the longest word that the computer can reliably guess overnight? What if you used an alphabet with only 10 letters instead of 26 letters? What if the alphabet had 100 letters?

Random Dots
If you randomly place dots on the screen, how long will it take before the entire screen is filled with dots? You could use the following procedure:

```
to dots
seth random 360
pu fd random 1000
make-a-dot
dots
end

to make-a-dot
pd fd 0 pu
end
```

To start the experiment, clear the screen, then run the **dots** procedure. If you run the experiment several times, does it take different amounts of time to fill the screen? It would be nice if the program "knew" when it was finished filling the screen. How could you do that?

What if the computer only had to fill a square, not the entire screen? How long would it take to fill the square with dots? What if the sides of the square were twice as long? How much more time would it take to fill the square?

A Random Walk Down Turtle Street
Suppose that the turtle is confused about where it is going. With each step, it is equally likely to take one step forward or one step back. How quickly will the turtle get anywhere?

Let's say you start the turtle at the middle of the screen. Where do you think the turtle will be by the next morning? What is the furthest that the turtle will have wandered from "home" during the night? If the turtle steps were twice as large (or half as large), how would the answers be different?
You could use a procedure like this:

```
to walk
ifelse (random 2) = 0 [fd 1][bk 1]
if ycor > :ymax [make "ymax ycor]
if ycor < :ymin [make "ymin ycor]
walk
end
```

The variables **ymax** and **ymin** keep track of how far "up" and "down" the turtle has wandered. You should initialize each with a value of 0 by typing the instructions **make "ymax 0** and **make "ymin 0**.

Actually, there is a bug in this procedure, since the turtle could "wrap" around the screen during its walk. How can you fix this bug?

When's Your Birthday?
If you are in a room with 30 people, what are the chances that two of the people have the same birthday? Are the chances better than 50/50?

Here's one way to investigate this question. Start the Logo turtle in the middle of the screen, heading at 45 degrees. Pick a number between 0 and 364 (representing a day of the year), move the turtle forward twice that distance, and make a dot. That dot represents one person's birthday. Then repeat the process, making dots for the other 29 birthdays. If the turtle ever puts two dots on the same spot, then two of the birthdays match. (Note: the turtle moves forward *twice* the random number, to make sure that nearby dots, representing nearby dates, do not "overlap" by accident.)

Here's a Logo procedure implementing this strategy. (The procedure assumes that the variable **people** has been initialized to 0. You may do this with the instruction **make "people 0**.)

```
to any-matches?
make "people :people + 1
if :people > 30 [output "false]
pu home seth 45
fd 2 * random 365
if colorunder = 1 [output "true]
pd fd 0 pu
any-matches?
end
```

What if you run this experiment many times? It's as if you keep entering new rooms, each with 30 people. What fraction of the rooms will have "birthday matches"? The birthday procedure runs the experiment over and over, keeping track of the number of matches:

```
to birthday
make "experiments :experiments + 1
enter-new-room
if any-matches? [make "matches :matches + 1]
birthday
end

to enter-new-room
rg
make "people 0
end
```

To start the investigation, initialize **experiments** and **matches** to 0, then run the **birthday** procedure. After a long time, what is the value of **matches** divided by **experiments**? Do more than half of the experiments result in matches? What if there were 25 people in each room? What if there were 40 people in each room? What if there were 367 people in each room?

More Experiments

These examples are intended to give you some ideas, to help you get started with Logo Overnight. The real challenge is to think of your own Logo Overnight projects.

It's a waste to let your computers sit idle at night. So start putting them to work!

Appendix - Different Versions of Logo

The projects described in this paper may be carried out in almost any version of Logo. However, the programs were developed using the MSDOS version of LogoWriter and may not work exactly as written in other versions of Logo, or even the Macintosh or Apple IIe versions of LogoWriter. Although the required changes are minor, they may be essential.

Here are some ways in which you might need to modify the programs in order for them to work in your version of Logo:

1. You cannot use procedure names like **count-sheep** or variable names like **number-of-guesses** in most versions of Logo because of the **-** character.
2. The command **cc** to clear the command center is used only in LogoWriter. In other versions you might need **cleartext**, or **ct**.
3. The syntax of **if** and **ifelse** varies from version to version.
4. The procedure **make-a-dot** may not work as written. In some versions of Logo, **fd 0** does not leave a dot. You could use **fd 1**. Your version may have a **dot** procedure. To put a dot at the turtle position you may need to use the instruction **dot pos** or **dot xcor ycor**, depending upon your version.
5. Instead of **rg** (reset graphics) to clear graphics from the screen, you may need to use **clearscreen**, **cs**, or **draw**.
6. **Colorunder** is not available in all versions of Logo. Alternatively, you may be able to use **dotp** or **dot?** to detect a dot on the screen.