

Twenty Things to Do with a Computer

Seymour Papert and Cynthia Solomon

When people talk about computers in education they do not all have the same image in mind. Some think of using the computer to program the kid; others think of using the kid to program the computer. But most of them have at least this in common: the transaction between the computer and the kid will be some kind of "conversation" or "questions and answers" in words or numbers.

In the real world, computers are used in many different ways. Some are programmed to fly airplanes—not to tell a human pilot what to do, but to pull the levers with their own electro-mechanical effectors and read the altitudes, airspeeds and what-not with their own electronic sensing devices. Computers are programmed to generate music or to condition dogs by ringing bells and delivering meat powder while the modern-day Pavlov is happily asleep. Some computers are programmed to control lathes and milling machines in industrial plants; others generate pictures for animated film cartoons.

Why then should computers in schools be confined to computing the sum of the squares of the first twenty-odd numbers and similar so-called problem-solving uses? Why not use them to produce some action? There is no better reason than the intellectual timidity of the computers-in-education community, which seems remarkably reluctant to use the computers for any purpose that fails to look very much like something that has been taught in schools for the past centuries. This is the more remarkable since the computerists are students of a momentous intellectual and technological revolution. Concepts from the sciences of computation—"cybernetics," "information theory," "artificial intelligence" and all its other names—have deeply affected thinking in biology, psychology and even the philosophy of mathematics. Machines from its engineering branches are changing our way of life. How strange, then, that "computers in education" should so often reduce to "using bright new gadgets to teach the same old stuff in thinly disguised versions of the same old way."

Our purpose here, however, is not to complain of what other people have not done, but to tell of some exciting things you can do with the computer you have now or with the one you will be incited to get by the suggestions that follow. More than half the suggestions we are about to make have been implemented and tested in our elementary school teaching program. This does not

imply that they are not of equal or greater value at other levels of education; on the contrary, we are convinced that they give a glimpse of the proper way to introduce *everyone*—whatever age and whatever level of academic performance—to programming, to more general knowledge of computation and, indeed (we say courageously steeling ourselves for the onslaught), to mathematics, to physics and to all formal subjects including linguistics and music.

Each section of this article describes something one can do with a computer. Most of these "things to do" assume that your computer can spin motors, activate electromagnets, switch lights, read the state of light-sensitive cells and so on. The amazing fact is that it is very easy to make your computer do all these things! The last section of this article says something about how to make it do so if it doesn't already. While reading the article you need not (and should not, it is a distraction) think about how the commands we describe will produce their effects. As you read on you will be learning a computer language called LOGO. In order to use a computer language, you do not need to know how the computer works—no more than you would need to know how a human brain works in order to give a person instructions. In both cases you need only know how to describe what you want in an appropriate language.

1. Make a Turtle (Figure 1)



Figure 1 shows one of our turtles—so named in honor of a famous species of cybernetic animal made by Grey Walter, an English neurophysiologist. Grey Walter's turtle had life-like behavior patterns built into its wiring diagram. Ours have no behavior except the ability to obey a few simple commands from a computer to which they are attached by a wire that plugs into a control-box that connects to a telephone line that speaks to the computer, which thinks it is talking to a teletype so that no special system programming is necessary to make the computer talk to the turtle. (If you'd like to make a fancier turtle, you might use a radio link. But we'd like turtles to be cheap enough for every kid to play with one.)

The turtle can send signals back to the computer. These signals appear to the computer just like the signals

Seymour Papert and Cynthia Solomon are with the Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

from a teletype—so, again, no special system programming is necessary to make a turtle talk to a computer. Where do the signals come from? They are generated by sense organs attached to the turtle. Our turtles do not have a fixed set of sense organs. Rather, they have inlets into which one can plug wires to attach any sense organs one is clever enough to make. Touch sensors, light-sensitive cells and sound detectors are obvious examples that require very little cleverness. Accelerometers and tilt detectors lead to more sophisticated fun.

Turtles can have effector organs as well. The activities described here use only a simple one—a pen located at the turtle's center, which can be lowered to leave a trace of the turtle's path, thus turning it into a remarkable geometric instrument.

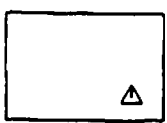
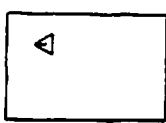
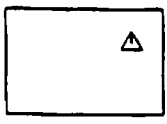
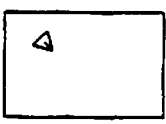
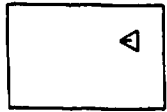
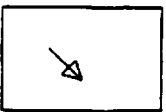


2. Program the Turtle to Draw a Man

A bad way to use a turtle is to know just which character symbols will cause the turtle's motors to move. A better way is to design a good language. This means deciding on a set of intelligible commands and building these into the computer language. For example, we can type `LEFT 90` on the console keyboard and thereby cause the turtle to rotate 90° about its central axis in the left (i.e., counter-clockwise) direction. Obviously this is better than having to figure, every time one wants to use the turtle, the number of steps of the stepping motors one needs to produce the desired movement and writing a complicated instruction to send out control characters to produce these steps.

The following diagram explains the main commands in our turtle language.

Turtle Language

At any time the turtle is at a particular *place* and facing in a particular *direction*. The place and direction together are the turtle's geometric *state*. The picture shows the turtle in a field, used here only to give the reader a frame of reference:

<p>(1) <code>FORWARD 100</code></p>  <p>The triangular picture shows the direction.</p>	<p>(1) <code>FORWARD 100</code></p>  <p>The turtle advanced 100 units in its new direction.</p>
<p>(2) <code>FORWARD 50</code></p>  <p>The turtle advanced 50 units in the direction it was facing.</p>	<p>(3) <code>LEFT 135</code></p>  <p>The turtle rotated left 135 degrees.</p>
<p>(3) <code>LEFT 90</code></p>  <p>The turtle's position remained fixed. It rotated 90 degrees to the left. So its direction changed.</p>	<p>(4) <code>PENDOWN</code></p>  <p>(Produces no visible effect. But the next FORWARD instruction will leave a trace.)</p>
<p>(4) <code>FORWARD 70</code></p>  <p>The effect of PENDOWN is to put the turtle in a state to leave a trace: the pen draws on the ground.</p>	<p>(5) <code>FORWARD 70</code></p>  <p>The effect of PENDOWN is to put the turtle in a state to leave a trace: the pen draws on the ground.</p>

To make the computer do anything more complicated you have to write a program. For example (using our language, LOGO, in a way that should be self-explanatory), one might type into the computer the following definition:

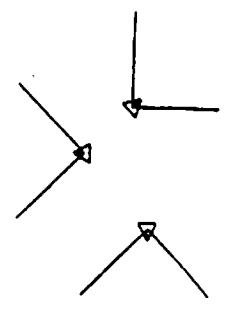
```
TO DRAW :DISTANCE
1 FORWARD :DISTANCE
2 BACK :DISTANCE
END
```

TO says that a definition follows. DRAW is the command being defined. :DISTANCE says that the command will have an input and in the definition its name will be "DISTANCE".

Now if we type the command `DRAW 100` the computer will say to itself, "How do I DRAW?" Well, the definition says, "TO DRAW 100, first go forward 100 units, then go back 100 units and that's all." So if the turtle is in PENDOWN state it will draw a line and come back to its starting position. Now, using DRAW as a sub-procedure, let's give the computer a new command VEE, by typing the following definition:

```
TO VEE :SIZE
1 LEFT 50
2 DRAW :SIZE
3 RIGHT 100
4 DRAW :SIZE
5 LEFT 50
END
```

A defined command can be used in defining new commands just as if it were a primitive LOGO command like FORWARD or LEFT.



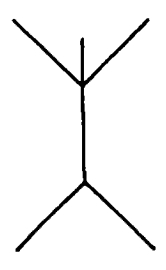
The command `VEE 100` will now cause the turtle to draw V's as shown in the figures. The starting and finishing positions of the turtle are shown by the usual triangle.

Now

```
TO MAN :SIZE
1 VEE :SIZE
2 RIGHT 180
3 FORWARD :SIZE
4 VEE :SIZE
5 FORWARD :SIZE/2
END
```

We now use the previously defined command in making a new command. In other words, DRAW is a sub-procedure VEE; VEE is a sub-procedure MAN.

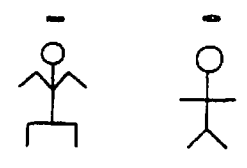
MAN 100 will draw



MAN 20 will draw



Here are some other drawings the fifth grade kids made the turtle draw.

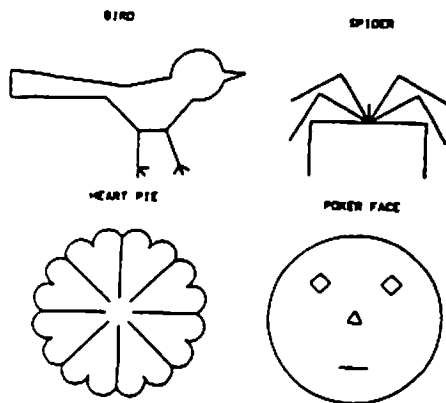


3. The must organ in the animal that a device should be able to make. L. behavior shall TOUCH. going back a for "I about outside promise promise

TO M I TI ? IF I FC I M, END

The eight b style of a LOGO name ide

TO MA I IF I ? FO I GO END



Turtle Biology

To make the turtle more like a living creature we must give it behavior patterns. This involves using sense organs. A well-conceived turtle should be very flexible in this respect: instead of fixed sense organs like real animals, it should have a number of sockets (we find eight is good) into which you can plug any on-off device such as a micro-switch, or light detector or whatever you think up. Such devices are easy and cheap to make.

Let's give the turtle a ridiculously simple piece of behavior based on using four touch sensors which we will call FRONTTOUCH, BACKTOUCH, LEFTTOUCH and RIGHTTOUCH. The behavior consists of going straight ahead until it touches the wall, turning back and so on. (The "and so on" illustrates the need for "loops" or "recursion" in the procedure we are about to write. To prepare yourself for the concept, consider the plight of a person who never fails to keep a promise and who has been tricked into saying, "I promise to repeat what I have just said.")

```

10 MARCH
20 TEST FRONTTOUCH
30 IF TRUE RIGHT 180
40 FORWARD 1
50 MARCH

```

The TEST will be "TRUE" if FRONTTOUCH is "TRUE"; i.e., if the front touch sensor is activated.

IF TRUE depends on the TEST. The turtle does an about face if the front touch sensor is touching the wall.

In any case the turtle takes a little step forward.

MARCH is the name of this procedure. It is also used as a command in the procedure. This is recursion. When the computer gets this, it starts to carry out the directions on how to MARCH. So, it starts again at line 1.

The next definition explains this idea in a way that might be clearer for people who are used to another style of programming. It also illustrates some flexibility in LOGO by showing other LOGO idioms to express the same idea:

```

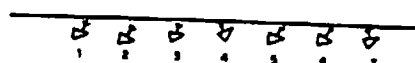
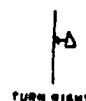
10 MARCH
20 IF FRONTTOUCH RIGHT 180
30 FORWARD 1
40 GO 1

```

This is equivalent to lines 1 and 2 above.

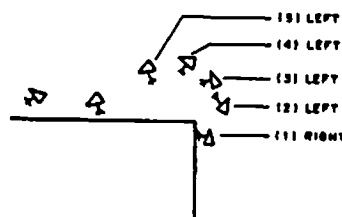
"GO 1" instructs the computer to go to line 1.

A more interesting behavior is to go to the wall and circumnavigate the room. Getting the turtle to find the wall is easy: just as in MARCH. To make it follow the wall we use the important concept of *feedback*. The idea is this. Imagine yourself walking next to a wall on your left with your eyes closed. Every now and then you put out your left hand. If it does not touch the wall, you say to yourself, "I'm wandering into space, better turn left a little." If you do feel the wall you say (slightly perversely), "Maybe I'm getting too close, better turn right a little." The result is that you will follow the wall perhaps in a slightly wavy line.



Your Progress

Interestingly, this procedure would make you circumnavigate a house walking on the outside. Watch what happens at the corner:



To circumnavigate the room from the inside one could use FRONTTOUCH to know when to turn. A small extension of the procedure could enable the turtle to find a door and escape from the room, or to explore a maze.

Using light sensors one can imitate a moth's flight to the candle, and cause turtles to pursue one another or to engage in dances or fights.

4. Make a Display Turtle

In our fifth grade class a turtle that walks on the floor is called a "turtle turtle." Another kind is called a "display turtle." This kind exists on a "scope" (i.e., cathode ray tube; a TV-like screen) as a picture just like the triangle we have used to illustrate the same commands, leaving a line of light as a trace when given the command PENDOWN. The disadvantage of the display turtle is that it cannot move physically about the world, touching, pushing and playing, but it has advantages for some purposes. One is that it is very fast and accurate. Another is that one can command it to draw a line which will last only for a stated length of time—say, a tenth of a second. Thus it can make moving pictures.

In LOGO the command `PEN :NUMBER` causes all lines to appear for `:NUMBER` tenths of a second. Thus `PEN 10` makes all lines last a second before vanishing.

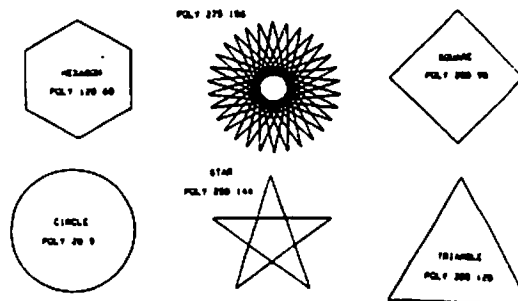
The command `FLY` will cause a bird to move across the screen if the following procedure has been written, as well as the procedure `BIRD`, which draws a bird.

```
TO FLY
10 PEN 2
20 BIRD
30 FORWARD 5
40 WAIT 2
50 FLY
END
```

This procedure draws a bird. The picture of the bird will last 0.2 seconds.

It waits 0.2 seconds. By this time the bird has vanished.

This causes the whole action to repeat as the machine gives itself the command `FLY`.



5. Play Spacewar

Spacewar is a famous computer game invented at Massachusetts Institute of Technology in the days when display programming was new and unusual. Two people play it. On the "scope" appear two spaceships, together with background frills such as stars, the sun, etc. There are two players; each controls a spaceship and may cause it to turn, go forward or shoot out a stream of rockets. Whoever destroys the other ship wins. The excitement of the game is increased by such dangers as getting caught by the sun's gravity and vanishing in a brilliant explosion.

When our fifth grade class visited M.I.T., they were caught up by the fun of playing the game. (It really is orders of magnitude better than non-computerized pin-tables.) Unlike most people, our children could go back to school the next day and get caught up by the even greater fun of programming their own versions of spacewar.

6. Differential Geometry

The "turtle language" provides a very remarkable formal system for describing many geometric objects; we think it is vastly superior to Cartesian coordinates as an introductory path into geometry. To see this let's study a very simple procedure, known in our fifth grade class as `POLY`. In its simplest form `POLY` has two inputs called "STEP" and "ANGLE." In LOGO it is written:

```
TO POLY :STEP :ANGLE
1 FORWARD :STEP
2 LEFT :ANGLE
3 POLY :STEP :ANGLE
END
```

The following pictures show the effect of invoking this procedure with different inputs (the first input is the side size; the second is the angle).

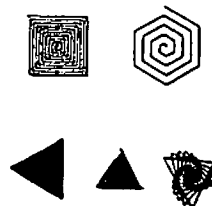
7. Draw Spirals

To change the procedure called `POLY` so as to

draw spirals we make a very small addition to line 3. we can also change the name—but this is, of course unnecessary.

```
TO POLY :STEP :ANGLE
1 FORWARD :STEP
2 LEFT :ANGLE
3 POLY :STEP :ANGLE
END
```

```
TO POLYSPI :STEP :ANGLE
1 FORWARD :STEP
2 LEFT :ANGLE
3 POLYSPI :STEP+5 :ANGLE
END
```

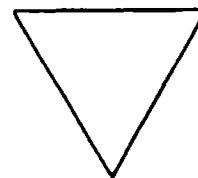


8. Have a Heart (and learn to DEBUG)

Making a procedure to draw a heart required the following steps.

Step 1: Find something like a heart that we know how to make. Idea: a triangle.

```
TO TRI :SIZE
1 FORWARD :SIZE
2 RIGHT 120
3 FORWARD :SIZE
4 RIGHT 120
5 FORWARD :SIZE
END
```



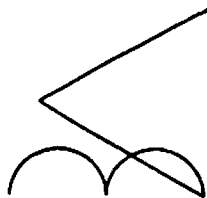
Step 2: Make a plan to modify `TRI`. Idea: Make a procedure `TOP`.

```
TO TOP :SIZE
1 SEG :SIZE/2
2 RIGHT 180
3 SEG :SIZE/2
END
```



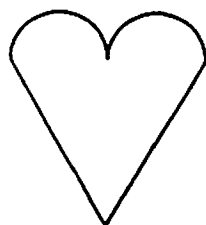
Then replace line 1 in TRI by 1 TOP :SIZE. This is easy but the result is:

HEART WITH BUG



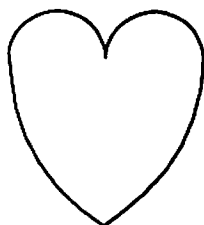
Step 3: Debug. Trying out this idea produced a bug. Why? Because replacing "FORWARD" by "TOP" in line 1 of TRI has side effects we did not anticipate! And is therefore typical of almost all good ideas in (almost all good projects.) To remedy this we must change line 2 as well; and while we are about it, let's change the name to "HEART1."

```
TO HEART1 :SIZE
  TOP :SIZE
  RIGHT 30
  FORWARD :SIZE
  RIGHT 120
  FORWARD :SIZE
END
```



Step 4: Consider: is this a good enough abstract model of a heart. No. Let's curve its sides. After a little debugging we get:

```
TO HEART2 :SIZE
  TOP :SIZE
  SEG 2*:SIZE 60
  RIGHT 60
  SEG 2*:SIZE 60
END
```



THEOREM: A heart can be made of four circular segments.

Growflowers

A computer program to draw this flower uses the geometric observation that petals can be decomposed (rather surprisingly!) as two quarter circles. So let's assume we have a procedure called TO QCIRCLE whose effect is shown by the examples. Some of them show initial and final positions of the turtle, some do not.

QCIRCLE 50

QCIRCLE 100



Now let's see how to make a petal.

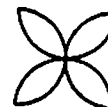
```
TO PETAL :SIZE
  1 QCIRCLE :SIZE
  2 RIGHT 90
  3 QCIRCLE :SIZE
END
```

PETAL 100



```
TO FLOWER :SIZE
  1 PETAL :SIZE
  2 PETAL :SIZE
  3 PETAL :SIZE
  4 PETAL :SIZE
END
```

FLOWER 100



```
TO STEM :SIZE
  1 RIGHT 180
  2 FORWARD 2*:SIZE
  3 RIGHT 90
  4 PETAL :SIZE/2
  5 FORWARD :SIZE
END
```

STEM 100



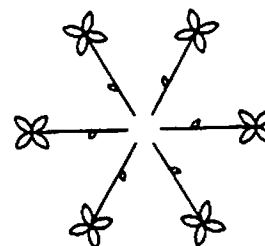
```
TO PLANT :SIZE
  1 PENDOWN
  2 FLOWER :SIZE
  3 STEM :SIZE
  4 PENUP
END
```

PLANT 100



Now let's play a little.

```
TO HEXAFLOWER :SIZE
  1 RIGHT 90
  2 FORWARD 4*:SIZE
  3 PLANT :SIZE
  4 FORWARD :SIZE
  5 RIGHT 30
  6 HEXAFLOWER :SIZE
END
```



10. Make a Movie

We describe how to make a very simple movie, in which the whole plot consists of a flower growing.

A flower can be drawn as well by the physical turtle described in idea number 1 as by a display turtle. Movies need a display turtle. The following commands in LOGO allow us to take advantage of a special feature of CRT drawings—their ability to vanish! We recall that the command PENDOWN causes the turtle to leave a trace. The commands PEN 50 (or PEN 10, etc.) cause a trace that will stay for 50-tenths of a second (or 10-tenths of a second, etc.) and then vanish. The command WIPE causes everything to vanish instantly.

Now let's try making successive frames of our little movie. First we do it by direct commands, rather than writing a new procedure.

PENDOWN
PLANT 5
WIPE
PLANT 10
WIPE
PLANT 30 etc.

This can be automated slightly by

PEN 50
PLANT 10
PLANT 20
PLANT 30

PEN 50 causes the picture to vanish after 5 seconds. So WIPE is not needed.

We give the commands PLANT 10, PLANT 20, PLANT 30 immediately after the previous picture vanishes.

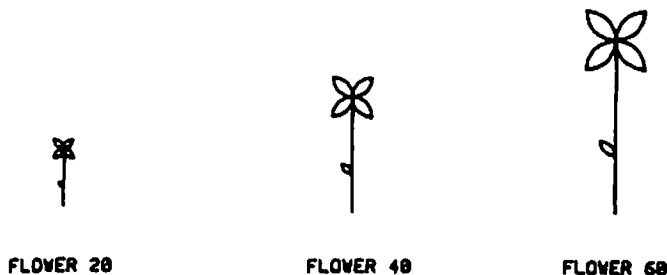
To automate the process further we build a procedure around the central action command:

PLANT :SIZE
WAIT 5

A pause of half a second occurs, so that the next round does not rush in before the previous plant is seen. PEN 50 would be chosen to match WAIT 5.

Now make some more exciting movies!

A superprocedure to issue these commands will be called MOVIE. It will make successive frames appear at half-second intervals.



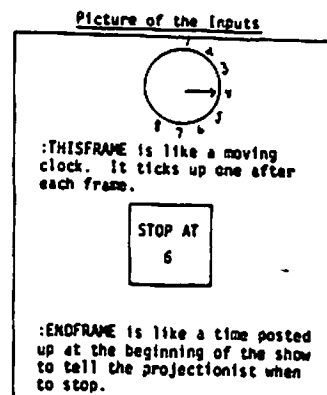
To command itself recursively at any given time, it must know the appropriate input for PLANT. It also needs to know its frame number so as to know when to stop. We notice that remembering the frame number eliminated the need to remember separately the input to PLANT—this is merely the frame number multiplied by 10. So the little movie program is:

```
TO MOVIE :THISFRAME :ENDFRAME
1 IF :THISFRAME = :ENDFRAME STOP
2 PLANT :THISFRAME*10
3 MOVIE :THISFRAME+1 :ENDFRAME
END
```

The meaning of these inputs is explained below.

We think of a movie as a process. As it goes on we need to know two things: where we are and where we are going. The two inputs are set up for this. The first is :THISFRAME. It starts at 1 and increases by 1 on each round. It is the frame number. The second input remains constant during the showing of the movie.

How to Think about the Inputs



11. Make a Music Box and Program a Tune

A music box is a device for making sound under control of a computer. Our style of music box "listens" to the signals sent by a computer to a teletype. Just as the teletype "decodes" them as instructions to print particular characters, and the turtle decodes them as movements, the music box decodes them as instructions to emit particular sounds. It is only a slight technical frill to give the music box several "voices" that will play simultaneously.

One (very bad) way to make the computer play "Frere Jacques" would be to write the following LOGO procedure:

```
TO FJ
1 PRINT "AAA!CCC!EEE!AAA!AAA!CCC!EEE!
  AAA!EEE!FFF!HHHHHHHHH!EEE!FFF!
  HHHHHH!..."
END
```

A better approach is to program the computer to accept descriptions of music in a good notation. An example is the following, which is one of several we are trying experimentally. (This notation and many of the ideas about the musical aspect of our work is due to Ted Winograd* and Jeanne Bamberger.**)

Our music box can play a five-octave range of notes, with as many as four at a time. One octave is chosen as the base, and its twelve chromatic tones are numbered 1 through 12. Notes in the next octave can be indicated either by continuing beyond 12 or by using the sign "*" Thus 13 and 1* represent the same note. A star preceding a number means "down an octave." The LOGO command SING takes a sequence of notes as input and plays them in order. Thus SING "1 3 5 6 8 10 12 1*" will cause a major scale to be played.

*Assistant professor, Department of Electrical Engineering, M.I.T.

**Research associate, Department of Electrical Engineering, M.I.T.

electro-magnet. Make a pile of iron discs, one on top of the other. Program the computer and crane and magnet to play Tower of Hanoi.

14. Make a Super Light Show

The school computer should have a large number of output ports to allow the computer to switch lights on and off, start tape recorders, actuate slide projectors and start and stop all manner of little machines. There should also be input ports to allow signals to be sent to the computer. We leave to your imagination the possibilities that this opens of making "interactive environments" for the next school festivity or even more solemn purposes.

In a similar spirit, but with a little more work, make an array of light bulbs to display the news of the day like they do it in Times Square. Or generate funny cartoons on the light bulb array. Or put up the scores at ball games and track events.

FRERE4
SING MUSIC "1*8 1" AND "2 2 4"

[illegible]

And Then Try Serious Composing

15. Write Concrete Poetry

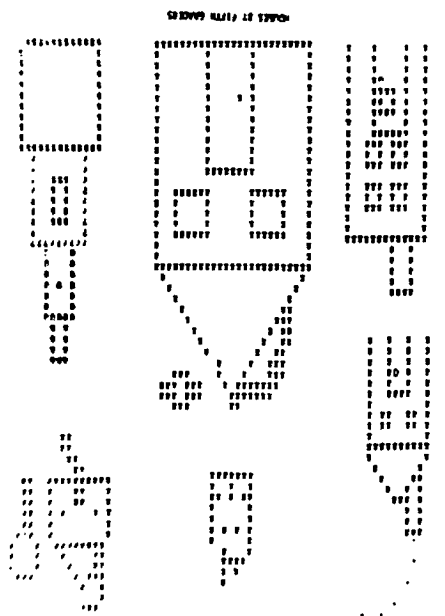
Perhaps we have carried too far our reaction against using computers to write symbols on teletype paper. Here are some examples of teletype output from procedures simple enough for the first weeks of a fifth grade course. We use teletype pictures as an initiation project to learn the very basic principles of using the computer, the terminal, the procedure definition idiom, the ritual for editing procedures and so on. Writing a random sentence generator made a girl exclaim, "So that's why we call words 'nouns' and 'verbs.' " What she meant was: for the first time I see a use for classifying words.

THE FUNNY PROF TALKED WHILE THAT
COOL KID KISSED...
SOME FUNNY PROF WALKED BUT A
BEAUTIFUL KID CLAPPED...
A WILD DONKEY KISSED WHILE THE FUNNY
PROF CLAPPED...
SOME GROSS PROF WALKED ALTHOUGH
SOME COOL KID HUMMED...

Build a Tower of Blocks

To pick up objects make a grab—or use an

ALL GREEN IN THE TWIGS
I GLIMPSE FAINT BIRDS IN THE COLD
WHIZZ THE SUN HAS CRACKED
ALL CURVED IN THE PEAKS
I SEE CLEAR PEAKS IN THE DUSK
WHIZZ THE FLOWER HAS CRACKED
ALL CURVED IN THE PEAKS
I GLIMPSE DARK TREES IN THE DAWN
WHIRR THE STORM HAS CRACKED

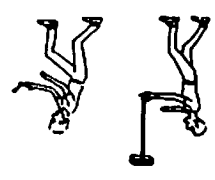


16. Try C.A.I. and Psychology
A slight extension of the sentence generator idea leads to generating mathematical sentences that are *true* (as well as grammatical), though somewhat boring. For example:

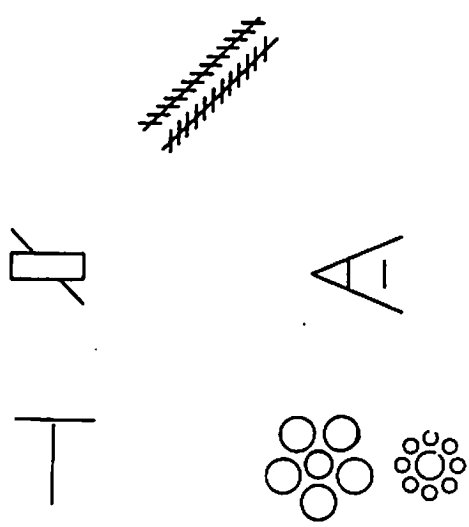
```
TO RANDOMSUM
1 MAKE
NAME "NUMBER1"
THING RANDOM
2 MAKE
NAME "NUMBER2"
THING RANDOM
3 MAKE
NAME "SUM"
THING "NUMBER1 + NUMBER2"
TYPE (SENTENCE:NUMBER1 "+":NUMBER2
"=:SUM)
5 RANDOMSUM
END
```

7 + 4 = 11
3 + 2 = 5
9 + 6 = 15

The effect is something like



17. Physics in the Fingertips
We begin by inviting the reader to carry out the illustrated experiments—or to recall doing something similar.

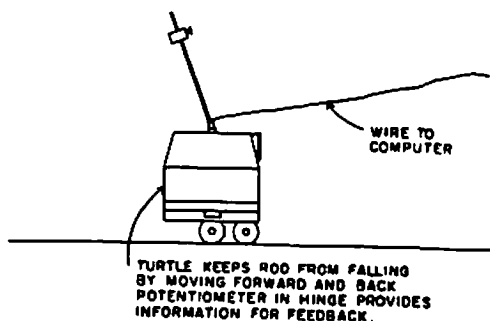
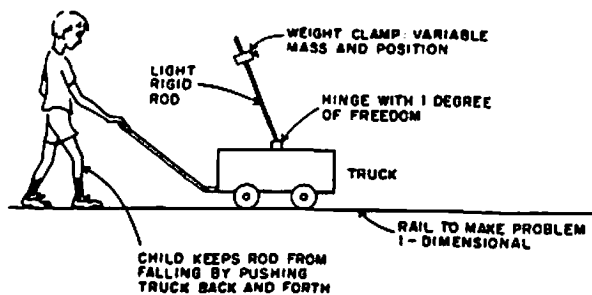


By taking the timing idea more seriously, one can do endless experiments to find out such facts as whether multiplications are hardest (for example: 1×1 is easier, but one might disagree about whether 7×9 is easier than 8×6). Or if one gets bored with teaching arithmetic, one can teach children how to estimate lengths of time, to recognize rhythmic patterns and so on endlessly.
The conclusion from all this is that we have also discovered the true role of C.A.I. in education. Writing C.A.I. programs is one of the twenty best projects for the first semester of a fifth grade computer science course. In a similar spirit, it is fun to do "optical illusion" experiments with the display turtle.

A slight modification will cause the computer to print something like $7 + 4 = ?$ and wait for a human victim to give the appropriate answer. For example:
7 + 4 = ?
ELEVEN (Computer)
IDiot, THE ANSWER IS 11 (Victim)
ELEVEN (Computer)
Even when the procedure has been modified to accept "ELEVEN" we can still tease the victim:

and so on.

One of the goals of this unit of study will be to understand how people do this and particularly to understand what properties of a human being determine what objects he can and what objects he cannot balance. A "formal physical" model of the stick balancing situation is provided by the apparatus illustrated next:



A computer-controlled version replaces the track and the child by a turtle with the angle sensor plugged into its sensor socket. A simple-minded procedure will do a fair amount of balancing (provided that the turtle is fast!):

```

10 BALANCE
1  TEST ANGLE > 10
2  IFTRUE FORWARD 8
1  TEST ANGLE < -10
1  IFTRUE BACK 8
3  WAIT 1
5  BALANCE
END

```

This procedure is written as part of a project plan that begins by saying: neglect all complications; try something. Complications that have been neglected include:

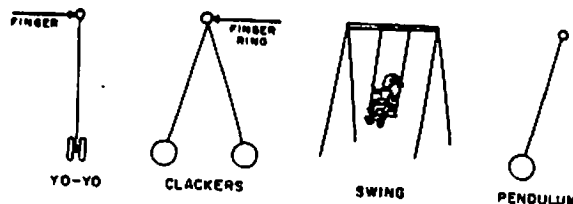
1. The end of the line bug.
2. The overshoot bug.
(Perhaps in lines 2 and 4 the value 8 is too much or too little.)
3. The Wobbly Bug
The TEST in the procedure might catch the rod over to the left while it is in rapid motion towards the right. When this happens, we should leave well enough alone!

One by one these bugs, and others, can be eliminated. It is not hard to build a program and choose instants so that with a given setting of the movable

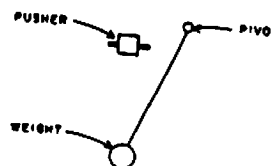
weight, balance will be maintained for long periods of time.

Feeding Energy

Again we begin with some fingertip physics by considering some toys:



All these systems will run down unless supplied with energy. How is the energy fed in? A good starting system is the clock pendulum on a rigid rod.



A linear actuator or one of the rotating joints can be used as a "pusher." A simple experiment will show the need for a good phase relationship. When this is understood, proceed to the flexible string and finally the interesting case of the swing in which the source of energy is carried by the pendulum.

A mechanical YO-YO player provides a different setting for similar principles and is an impressive example of a "skill" that can really be achieved quite easily by machines. Moreover, it opens up a huge vista of challenging problems. Causing the YO-YO to SLEEP is a feasible, hard project in our context. The more elaborate tricks like WALKING-THE-DOG or ROUND-THE-WORLD would probably succumb, but would need a lot of work and ingenuity.

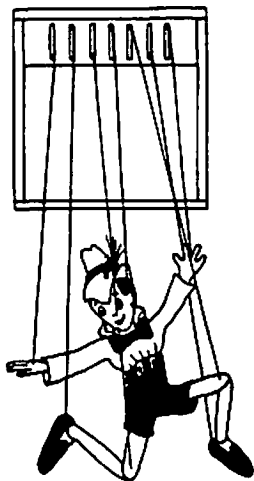
18. Explain Yourself

Building machines to balance sticks did not actually answer the original question about why people can balance broomsticks but not toothbrushes. What property of people determines how long (or short) a stick one can balance? The answer is *reaction time*! Now go back to the balancing machines to give them reaction times similar to those of people (which you will find out by carrying idea number 16 a little further). How good a model can you make of a person? Does this explain you—or at least explain one of your characteristics? Could similar models explain other human characteristics?

19. Puppets

The computer controls enough motors to pull enough strings to manipulate the desired number of marionettes. Like many of these projects, this one has a

great educational property: some effect can be obtained by extremely simple means; extra effort will produce more exciting effects; and to emulate a skilled human puppeteer will require a very thorough understanding of the geometric and dynamic principles of movement.



20. Recursion Line
Think up twenty more things to do.

Epilog: How to Make Those Things Happen
Most of the devices we have mentioned are extremely simple and much cheaper than teletypes. The hardest problem has been getting the computer to communicate with the device. The approach we have developed centers around the concept of a "universal controller." This we define as a black box which looks to the computer like a teletype. So, to use it you would program the computer to print a piece of text which might read "!!(!!!(!!!(!!!(" knowing that the controller will turn "!" and "(" into turtle signals whose effect will be to cause forward and left steps, respectively. Thus any programming language, running on any operating system (including commercial time-sharing services) can be used to control a turtle.

In our image of a school computation laboratory, an important role is played by numerous "controller ports" which allow any student to plug any device into the computer. The ports are protected by fuses and suitable interfaces so that little harm will be done if anyone carelessly puts the main voltage into a computer output port. The laboratory will have a supply of motors, solenoids, relays, sense devices of various kinds, etc. Using them, the students will be able to invent and build an endless variety of cybernetic systems.

This is not the place to discuss strictly practical problems like where to buy good motors. We do, however, expect that very soon someone will supply a full range of suitable things. In any case we would be happy to provide advice and information.

On the Cost of Computation in Schools

A final word about the cost of doing all this. Turtles, music boxes, computer-controlled motors and

the like are less expensive than teletypes. Displays are slightly more expensive but are becoming rapidly cheaper. If computers are being used in a school, there is no good economic argument for accepting the narrowness of the pure teletype terminal.

Some school administrators and town politicians still consider the cost of using computers at all as too high. If you are engaged in battles on this point, write LOGO Information* to be briefed on the latest ideas and prices of equipment. At the moment a good estimate of what computation ought to cost is \$30 per student per year, for one hour per student per week of terminal time. This is based on the assumption that several hundred students will be involved. The price could be halved within a year if several hundred schools would commit themselves to installing identical systems. *Only inertia and prejudice, not economics or lack of good educational ideas, stand in the way of providing every child in the world with the kind of experience which we have tried to give you some glimpses of.* If every child were to be given access to a computer, computers would be cheap enough for every child to be given access to a computer.

*LOGO Information, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

This work was supported by the National Science Foundation under grant No. GJ-1049 and conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under contract No. N00014-67-A-0362-0002.

SUBSCRIPTION AND BACK VOLUMES

Educational Technology Magazine
Englewood Cliffs, New Jersey 07632

☐ Please enter my one-year subscription, at \$18.00 (\$21.00 foreign).

☐ Please forward the following back volumes now available

1964	1965	1966	1967
1968	1969	1970	1971

Note: Volumes from 1964 to 1967 have been reprinted in limited quantities in bound volumes. Volumes from 1968 to 1971 are available in their original form, as single issues. All back volumes are \$24.00.

Name

Address

City State Zip