Exploratory learning with graphs

Licklider was interested early in using computers for expressing multiple linked modes of representation of concepts and phenomena, especially including visual representations. In 1961 he developed a program, *Exploratory Learning with Graphs*, that enabled a user to specify a polynomial equation, such as the quadratic $y = a(x - b)^2 + c$, and assign values to the coefficients. The computer would generate the graph of the function. The user could then change the coefficients and the computer would generate the corresponding graph on the same screen. The intent was that, by exploring the effect on the graph of changes in the coefficients, and by investigating the operation of the program for a variety of polynomial functions, a student would develop a better intuitive understanding of the relationship between symbolic and graphic representations of functions.

Socratic System and the Mentor Language

Computer scientist Wallace Feurzeig came to BBN in 1962 to work with Licklider on the development of interactive computation facilities ("thin-skinned" computing) and user-oriented programming languages. After initial work programming acceptance routines for the newly arrived research computer, the Digital Equipment Corporation PDP-1, Feurzeig was invited by psychologist John Swets to collaborate on a CAI research project. The proposal called for the development of a conventional CAI system, very much like the drill-and-practice program described above. Feurzeig and Swets proposed an alternate approach. They wanted to extend the versatility and instructional power of then-current CAI systems by enabling computer support for complex learning tasks that allow students greatly enhanced capabilities for exploration and investigation.

In 1963, Feurzeig designed and implemented a CAI system with the following capabilities. Students would not be limited to responding to questions asked by the program. They could take the initiative by *asking* the questions — that would not be the sole prerogative of the program. This sharing of control between the program and the student was subsequently dubbed "mixed-initiative interaction." Further, the program would not have to make a fixed response to the student's inputs. Its response could be conditional on history (i.e., what had happened during the interaction thus far and thus, presumably, what the student had learned) as well as on the context within which the inputs occurred.

Swets named the program the *Socratic System* because of its ability to support sustained investigative dialogues between the student and the program. In a typical application, the program presents a problem to a student and engages him in a mixed-initiative dialogue in support of his attempt to solve the problem. The initial applications, designed to test the operation of the system, included an alphabet character recognition game and an electronic troubleshooting problem with a simple circuit. The major application, which demonstrated the power and potential usefulness of the system, was a differential diagnosis problem in clinical medicine. The application was inspired by a thought piece of Swets titled "Some Possible Uses of a Small Computer as a Teaching Machine." Here is an excerpt from that 1959 BBN memorandum.

> Let's say we want to make good diagnosticians out of our blossoming M.D.'s. So we have lots of cases in a computer, A student comes into the computer room, selects a card out of a file, and learns that John Doe has a medical history of thus and so, that some intern has "worked him up" on his recent admittance thus and so. What's John's problem? The student sits down at an available typewriter, and decides what else he wants to know. He wants to know if John has urea in his urine, so he asks the computer and the computer tells him the answer is "yes." "Aha, then how many

> white corpuscles does he have?" Answer: "150." "Well," he tells the computer, "this is clearly a case of mononucleosis." The computer replies: "Don't you think you ought to know whether John shows a Bobinski before deciding such?" "Yeah," says the student, "I guess so, does he?" Answer: "Yes." "Now I'm sure it's mononucleosis." "But" says the computer, "you are forgetting that John's pulse is normal, which you well know, is inconsistent with your diagnosis."...

In a *Socratic System* medical application (Feurzeig et al, 1964, Swets and Feurzeig, 1965), the student is given a statement of the problem (the patient's complaint and other presenting information) and a list of the questions and assertions that can be input by the student in the course of the interaction. Allowable questions include standard medical items: the patient's condition (e.g., general appearance?), physical examination (e.g., auscultation?), and requests for laboratory tests (e.g., rbc?). Allowable assertions include diagnoses (e.g., appendicitis), and justifications for a given diagnosis (e.g., evidence from urine culture). The list can be extensive. The student can do the history, physical exam, request lab reports, and make diagnoses in any order. As in real life, lab results need not be immediate, they may be reported back some time after they are requested. The program may respond differently to the same input, depending on occurrences in the interaction that should affect the student's knowledge. The problem can embody a situation that changes with time as the interaction develops. For example, the patient's condition may worsen or improve.

The system can respond to the student in the manner of a personal tutor, commending thoughtful questions, reproving unwarranted tests or foolish conclusions, acknowledging perceptive decisions, and questioning the grounds of diagnostic inferences. It can respond to a question by posing a question. The pedagogic strategies are not built in, they are specified by the instructor who designs the problem. Directed by these strategies, the system can develop interesting contingencies to considerable depth. The following excerpt from a student interaction with the system, illustrates some of these capabilities. The student's inputs are indented and capitalized to set them off from those of the system. The student's preliminary questions have been deleted.

Here is the problem statement.

> It is mid-December. Late this afternoon a patient was admitted to the hospital. He is a well-developed, well-nourished, 34-year-old coal miner. He was brought in by his wife who reports that he has had "the grippe" for about 4 days. During the past 3 days she has given him fluids and aspirin every 6 hours with no sign of improvement. Today he suddenly became much worse. For the past 2 days he had complained of feeling very warm and of pain in his belly. This morning he had a severe shaking chill that was so bad his teeth rattled.
>
> [*Deleted: student's initial requests for vital signs and other physical exam items.*]
>
> Before proceeding with the reexamination, it might be instructive to pause and consider the diagnostic implications of the information at hand. Perhaps you think it too early to arrive at a definitive conclusion. On the other hand, you may be prepared to make the diagnosis now. Are you?
>
> NO.
>
> Fine. Nevertheless, we've got a fair amount of information-at least enough to raise some, perhaps several, diagnostic possibilities. Have you any in mind?
>
> YES.
>
> Name one.
>
> PNEUMOCOCCAL PNEUMONIA.
>
> Good. The general signs of a pneumonic process are present. Of course, you haven't isolated the bug. Do you think the belly pain and flank pain reinforce the case for pneumococcal pneumonia?

NO.
Right. Though the white count is consistent, isn't it?
YES.
I think so too, but a later reading is indicated to confirm this finding. We've considered one possibility. Do you have another diagnosis in mind?
NO.
In that case, I'd like to talk about pulmonary infarction. What I like about that diagnosis is the presence of high fever, rusty sputum, and shaking chills. I don't like the dyspnea, pleural pain, and signs of consolidation-do you agree?
NO.
You're quite correct. I was just trying to see whether you were alert-and you were. Now let's see how the patient is today. Begin the physical when you're ready.
[*Deleted: student requests for vital signs and other physical exam items.*]
PNEUMOCOCCAL PNEUMONIA.
Yes. I remember you liked this diagnosis before. Are you absolutely sure?
NO.
What would help you nail down the diagnosis.
EVIDENCE FROM SPUTUM SMEAR AND CULTURE.
Good. Anything else?
EVIDENCE FROM X-RAY FINDINGS.
Good. X-ray shows area of consolidation at right lower lobe, consistent with a pneumonic process. Lab reports gram positive organisms in the sputum smear and the presence of pneumococci in the culture. Had you thought to rule out things like appendicitis and acute pyelonephritis? Apparently you weren't thrown off by the referred abdominal pain. In any case, you've made the correct diagnosis.

In this example the student was fairly insightful. Less-thoughtful students may make ill-informed diagnostic guesses. The program is more demanding when the evidence for their diagnoses is absent or weak.

Feurzeig designed and implemented a user-oriented programming language, *Mentor*, for developing *Socratic System* applications. One application was a parody of the Agatha Christie mystery, expressly designed to demonstrate the capabilities of both the *Socratic System* and the *Mentor* language. Other applications were made in diverse areas including classroom scheduling, management decision-making, and electronics troubleshooting. The work in medical diagnosis spearheaded further work in medical applications within BBN (e.g., the NIH hospital time-sharing project). It also led to extensive work on computers in medical education elsewhere in the 1970s and 1980s (Clancey, 1982).

Scholar program

Scholar was the first attempt to use a semantic network for knowledge representation as the basis of a teaching program. BBN computer scientist Jaime Carbonell designed the program in 1970. He distinguished frame-oriented CAI systems, which require that the designer anticipate all questions and answers in advance, from information-structure-oriented CAI systems, which represent domain information as a knowledge network from which specific system responses can be constructed. He realized that it was useful to have separate representations of the domain knowledge to be taught and the teaching strategy that was to be employed. Thus, *Scholar* has two components: a knowledge base to incorporate domain information, and a program for directing the tutorial interaction with the student. He saw the development of Scholar as "a step in the direction of CAI programs that know what they are talking about, the same way human teachers do."

The program was implemented in LISP. Its knowledge base was organized as a

semantic network (Quillian, 1968). The program's initial knowledge base contained geopolitical information about South America. For each country it stored such information as population, land area, capital, major cities, major rivers, major mountains, major religions. What could be taught by the approach was limited to a set of facts: "The area of Peru is approximately 400,000 square miles. The major cities in Brazil are Sao Paulo, Rio de Janeiro, Brasilia, . . . " The natural-language semantics and syntax required for treating such information are highly constrained. An excerpt from a representative student dialogue with Scholar follows (Carbonell, 1970). The student's inputs are indented and enclosed by asterisks.

What is Chile?
    a country
That's fine.

Select an alternative from the list: 214,000; 800,300; 49,883; 63,690;
For the question: approximately what is the area of Chile?
    800,300
Your answer is incorrect. The correct answer is 214,000.

The population in Chile is approx 850,000. True or False?
    true
You are correct. (sic!)

    Tell me something about Peru.
Wait a minute, I'm checking my files.
Let's proceed now.
Peru is a country.
It is in South America.
The population is approx 7,000.000 people.
The capital is Lima.

    Please tell me more about Peru.
The area of Peru is approx 48,000 square miles.
The language is Spanish.

    How many square milea are there in Brazil?
The area is approx 3,600,000 square miles.

Approximately what is the area of Argentina?

    What is the principal religion in Uruguay?
The principal religion in Uruguay is Catholicism.

Now answer the question you didn't answer before.
    The area of Argentina is 1,100,000 square miles.
Approximately true, but the exact value is 1,200,000 .

The eastern bordering country of Peru is . . . ?
    Catholicism.
Wrong. Catholicism is not a country but a religion.
The correct answer is: Brazil.

    Is Asuncion hot?
Since Asuncion is part of Paraguay, yes.

    What is the most important difference between Peru and Paraguay?
The primary difference is: the population of Peru is larger by a factor of 7.8.

Just before Carbonell's untimely death, he and BBN educational psychologist Allan Collins sought to use the *Scholar* system to explore a number of issues in natural language semantics (Carbonell and Collins, 1973). They had begun to consider how

to implement some human reasoning capabilities in *Scholar*, such as the ability to make deductive, negative, and inductive inferences. They also intended to incorporate teaching strategies like those used by human tutors. Collins and colleagues sought to identify such strategies by recording and analyzing dialogues of human teachers tutoring students on South American geography (Collins, Warnock, and Passafiume, 1974). They identified six tutorial strategy categories: topic selection, interweaving questions and presentations, questioning about basic concepts, reviewing, use of hints, and responses to errors. They looked for common strategies across teachers and identified several general principles. For example, tutors appeared to introduce new questions when they thought the student knew the answer to previous ones, and to present additional information otherwise. They then attempted to program these strategies into Scholar.

One version of *Scholar* incorporated the capability to generate and display annotated maps (Collins, Adams, and Pew, 1978). It enabled the display of the entire continent of South America or sections appropriate to particular questions. For example, it could access the relevant map section and highlight the locations of cities in response to student requests such as "blink Brasilia." *Scholar* work with the South American geography knowledge base was augmented by two additional *Scholar* applications. One dealt with the structure of the ARPA computer network, the precursor of the Internet; the other with a text-editing system called NLS. The latter work resulted in the development of a version of *Scholar* called *NLS-Scholar* (Grignetti et al, 1974). *Scholar* proved influential in fostering research and development in the area of artificial intelligence applications to education, both outside and within BBN.

Why program

Why is a continuation of the work on *Scholar*. It was developed by Collins and fellow BBN psychologist Al Stevens after Carbonell passed away (Collins and Stevens, 1980). The major advance was in changing the character of the instruction from purely factual exchanges to causal reasoning interactions, as exemplified in the domain of meteorology. Here is an excerpt from a *Why* dialog.

Do you think the Amazon jungle has heavy or light rainfall?
Heavy rainfall

Why does the Amazon have heavy rainfall?
Because it has mountains nearby

Do you think that any place with mountains has heavy rainfall?
Yes

Southern California has mountains.
Why doesn't Southern California have heavy rainfall?

The interaction illustrates the application of explicit teaching rules for generating the questions. These fall in the category of predictions, particular cases, prior causes, insufficient causes, and general rules. For example, the first question above asks the student for a prediction about a particular case. The second question asks for prior causes. The third question asks the student about a general rule. The last question introduces a counter-example to the student's insufficient causal response, and asks for prior causes.

The program was able to detect obvious student misconceptions. It was not used for carrying on extended dialogues nor did it claim to diagnose students' underlying misunderstandings or erroneous models of weather processes. Its major advance was

the introduction of a tutorial strategy that employs a systematic logical approach for formalizing the questioning methods.

How the West Was Won

From 1973 through 1980, computer scientists John Seely Brown, Richard Burton, and their colleagues in the BBN Intelligent CAI group did advanced instructional research and software design leading to the implementation of tutorial systems incorporating powerful artificial intelligence facilities.

In 1975 they developed a paradigm for tutorial systems with capabilities for providing automatic feedback and hints in a game environment (Brown and Burton, 1975; Burton and Brown, 1976). They demonstrated the paradigm by implementing a computer coaching system,*West*, based on the children's game "How the West Was Won," a variation of the classic game "Chutes and Ladders." *West* was designed to teach computational skills through game playing strategy. There are two opposing players, one of whom may be the computer. The objective of the game is to land exactly on the last town on the game-board map. On each turn, a player spins three spinners to produce three numbers which he then combines using two of the operations addition, subtraction, multiplication, or division, possibly with parentheses. The value of the arithmetic expression thus generated is the number of spaces he gets to move. (Negative values result in backward moves). There are towns and shortcuts along the way to the goal. The rules specify the effect of a move landing on a town (moving to the next town), landing on a shortcut (advancing to the end of the row), or landing on the same place as his opponent ("bumping" him back two towns).

The system uses a computer-based "expert" player. It tracks and evaluates a student's moves and constructs a "differential model" that compares the expert's performance with that of the student. Procedural specialists assess the conceptual constraints that might prevent the student's full utilization of the environment. These help the tutor decide whether and when to suggest better moves to the student. For example, the student may be unaware of the benefit of bumping his opponent, e.g., of evaluating whether it is more advantageous to send her opponent back m places or to get ahead of her by n places. This assumes, of course, that she knows desirable values for m and n, and also how to construct appropriate arithmetic expressions that compute m and n from the three numbers selected by the spinners. Thus, a poor move might be due to the student's failure to consider a better alternative or to an incorrect computation of a move, a distinctly different kind of difficulty that calls for a qualitatively different instructional treatment.

Sophie

The intent of *West* was to turn a "fun" game into a productive learning environment without diminishing the student's enjoyment. The performance analysis in West identifies weaknesses in the student's play, but it does not diagnose the underlying difficulties that are responsible for them. From 1974 through 1978, the ICAI group undertook a considerably more ambitious effort, the development of an "intelligent" instructional system, *Sophie*, (for SOPHisticated Instructional Environment). Unlike previous CAI systems that employed AI methods to emulate a human teacher, *Sophie* sought to create a "reactive" environment that fosters a student's learning while he tries out his ideas working on a complex electronics troubleshooting task (Brown, Burton, and Bell, 1975). *Sophie* supports a student in a close collaborative relationship with an "expert" who helps the student explore, develop, and debug his own ideas.

Sophie incorporates a "strong" model of the electronics knowledge domain along with heuristic strategies for answering a student's questions, critiquing his current solution paths, and generating alternative theories to his current hypotheses. Its expertise is derived from a powerful inferencing scheme that uses multiple representations of knowledge, including simulation models of its electronics circuit domain, procedural specialists for using these models, and semantic nets for encoding factual knowledge. *Sophie* was designed to demonstrate the feasibility of using AI techniques to construct an instructional system which, *on its own*, could reason, answer unanticipated questions, evaluate a student's hypotheses, and critique the student's performance behaviors, while carrying on an intelligent tutorial dialogue (Brown and Burton, 1978a).

In the basic scenario, *Sophie* acts as a lab instructor interacting with a trainee who attempts to debug a malfunctioning piece of equipment. The trainee can perform any sequence of measurements, ask questions about the implications of these measurements or more general hypothetical questions, and ask for advice about what to do next. *Sophie* may encourage the trainee to make a guess about what may be wrong given what he has found thus far. *Sophie* will evaluate his hypothesis by considering what he should have been able to derive from his current measurements, and if this information is contradicted by the hypothesis, *Sophie* will identify and explain these contradictions, just as in the Socratic System. *Sophie* can determine whether a new measurement is logically redundant and whether it rules out possible circuit faults which were still in contention. These capabilities require deep logical inferencing. They would be difficult or impossible to achieve by the use of pre-stored decision trees, since the sequence of measurements and the hypotheses any trainee will make is unknown in advance.

The electronic instrument used in *Sophie* is based on the Heathkit IP-28 power supply, a fairly sophisticated circuit with both current-limiting and voltage-limiting behavior. These two interrelated feedback loops make troubleshooting this circuit a nontrivial task. The following brief excerpt illustrates the kind of interaction made possible by *Sophie*. The student's inputs are preceded by two carats. Reviewer annotations are parenthesized. The student has asked *Sophie* to insert a fault in the circuit. *Sophie* randomly selects a fault and inserts it into the model of the circuit.

The instrument has a fault and its present control settings are:
CC = 1.0 (Current Control - from 0.0 (Min Current) to 1.0)
CR = HIGH (Current Range Switch, HIGH = 1 Amp, LOW = 0.1 Amp)
LOAD = 20 (Load Resistance in Ohms)
VC = 1 (Voltage Control - from 0.0 (Min Voltage) to 1.0)
VR = HIGH (Voltage Range Switch, HIGH = 30, LOW = 10 Volts)

>>What is the output?
The output voltage is 11.7 volts.

>>What is it in a working instrument? (uses dialog model for interpreting pronouns)
In a working circuit the output voltage is 19.9 volts.

>>What is the voltage between R8 and Q1?
The voltage at N1 is 47.6 volts.

>>What is the base emitter voltage of the voltage limiting transistor? (=transistor)
The base emitter voltage of Q5 is .037 millivolts.

>>Is it possible that Q6 is open? (hypothesis assertion, questioned by *Sophie*)
Is the complete fault mode EB/EC/BC = OP/OK/OP what you intended?

>>Sure. (*Sophie* constructs a counter-example)
When the settings were: CC=1.0, CR=HIGH, LOAD=20, VC=1.0, VR=HIGH,
that fault conflicts with the following measurement. You observed that the

output voltage was 11.7 volts. If Q6 had fault mode EB/EC/BC = OP/OK/OP
it would be 23.9 volts. In a working circuit it is 19.9 volts.

(*Sophie* also determines what is "good" about the student's hypothesis.)
That fault does not explain any of the symptoms you observed but it is
consistent with the values for the base emitter voltage of Q5 and the voltage at N1.

Sophie employs a semantic grammar incorporating extensive domain-specific knowledge (Burton and Brown, 1979). Here are some representative examples of student inputs that are effectively and efficiently parsed by *Sophie*.

What is the voltage across the base emitter junction of the current limiting transistor?
What is the VBE of Q6?
What is current through the base of Q5?
What is the voltage between node 1 and the positive terminal of C6?
What is the dynamic resistance of R11?
What is the beta of the voltage limiting transistor?
In a working circuit what is the output voltage of the power reference transformer?
Change the output load to 10 megaohms.
Let C2 be leaky.
Set the current control to maximum.
Suppose the BE junction of Q6 is shorted.

Sophie has been used in a two-person gaming situation where one student introduces a fault into the circuit and predicts the consequences and the other student is challenged to discover the fault. The roles are then reversed. In another version of the game, one student introduces a circuit modification and the other requests measurements which the first student answers as best he can on the basis of his earlier prediction of the effects of his modification on circuit behavior. The system could monitor the operation and interrupt if a mistake could result in a serious compounding of misunderstandings.

The understanding capabilities in *Sophie* were largely based on its use of a general circuit simulation model (SPICE), together with a Lisp-based functional simulator incorporating circuit-dependent knowledge. These facilities were essential for inferring complex circuit interaction sequences such as fault propagation chains. *Sophie's* capabilities for modeling causal chains of events formed the basis for its explanation and question-answering facilities. *Sophie* used the simulator to make powerful deductive inferences about hypothetical, as well as real, circuit behavior. For example, it determined whether the behavior of the circuit was consistent with the assumption of specified faults and whether a student's troubleshooting inferences were warranted, i.e., whether the student had acquired information of the voltage and current states of relevant circuit components sufficient to unambiguously isolate the fault.

Sophie could infer what the student should have been able to conclude from his observations at any point, e.g. the currently plausible hypotheses and those that were untenable. However, because *Sophie* did not determine the reasons underlying the student's actions, e.g. the hypotheses he was actually considering, it was unable to diagnose the student's underlying conceptual difficulties in understanding and diagnosing circuit behavior. Despite this limitation, *Sophie* was one of the first instructional systems capable of supporting compelling and effective knowledge-based interactions, and it had enormous influence on other work in the ICAI area during the 1970s and 1980s.

13.2 Learning and teaching mathematics

Wallace Feurzeig founded the BBN Educational Technology Department (ETD) in 1965 to further the development of improvements in learning and teaching made possible by interactive computing and computer time-sharing. Time-sharing made feasible the economic use of remote distributed computer devices (terminals) and opened up the possibilities of interactive computer use in schools. The ETD work shifted from the development of tutorial environments to the investigation of programming languages as educational environments. The initial focus of the group was on making mathematics more accessible and interesting to beginning students.

Stringcomp

BBN programmers implemented the TELCOMP language in 1964 (Myer, 1966). It was modeled after JOSS, the first "conversational" (i.e., interactive) computer language, which had been developed in 1962-63 by Cliff Shaw of the Rand Corporation. TELCOMP was a FORTRAN-derived language for numerical computation. BBN made it available as a time-sharing service to the engineering market. Shortly after TELCOMP was introduced, Feurzeig extended the language by incorporating the capability for non-numerical operations with strings, to make it useful as an environment for teaching mathematics. The extended language was called *Stringcomp*.

In 1965–66, under U.S. Office of Education support, Feurzeig and his group explored the use of *Stringcomp* in eight elementary and middle school mathematics classrooms in the Boston area, via the BBN time-sharing system. Students were introduced to *Stringcomp*. They then worked on problems in arithmetic and algebra by writing *Stringcomp* programs. Experiencing mathematics as a constructive activity proved enjoyable and motivating to students, and the project strongly demonstrated that the use of interactive computation with a high-level interpretive language can be instructionally effective.

Logo educational programming environment

Feurzeig's collaborators in the development of *Logo* were BBN scientists Daniel Bobrow, Richard Grant, and Cynthia Solomon, and consultant Seymour Papert, who had recently arrived at MIT from the Piaget Institute in Geneva. The positive experience with *Stringcomp*, a derivative language originally designed for scientific and engineering computation, suggested the idea of creating a programming language expressly designed for children. Most existing languages were designed for doing computation rather than mathematics. Most lacked capabilities for non-numeric symbolic manipulation. Even their numerical facilities were typically inadequate in that they did not include arbitrary precision integers (big numbers are interesting to both mathematicians and children).

Existing languages were ill-suited for educational applications in other respects as well. Their programs lacked modularity and semantic transparency. They made extensive use of type declarations, which can stand in the way of children's need for expressing their ideas without distraction or delay. They had serious deficiencies in control structure, e.g. lack of support for recursion. Many languages lacked procedural constructs. Most had no facilities for dynamic definition and execution. Few had well-developed and articulate debugging, diagnostic, and editing facilities, essential for educational applications.

The need for a new language designed for, and dedicated to, education was evident. The basic requirements for the language were:

1. Third-graders with very little preparation should be able to use it for simple tasks
2. Its structure should embody mathematically important concepts with minimal interference from programming conventions
3. It should permit the expression of mathematically rich non-numerical algorithms, as well as numerical one

Remarkably, the best model for the new language (*Logo*) turned out to be *Lisp*, the lingua franca of artificial intelligence, which is often regarded by non-users as one of the most difficult languages. Although the syntax of *Logo* is more accessible than that of *Lisp*, *Logo* is essentially a dialect of *Lisp*. Thus, it is a powerfully expressive language as well as a readily accessible one.

The initial design of *Logo* came about through extensive discussions in 1966 among Feurzeig, Papert, and Bobrow. Papert developed the overall functional specifications, Bobrow did the first implementation (in *Lisp* on a Scientific Data Systems SDS 940 computer). Subsequently, Feurzeig and Grant made substantial additions and modifications to the design and implementation, assisted by Solomon and BBN engineers Frank Frazier and Paul Wexelblat. Feurzeig named the new language *Logo* ("from the Greek $\lambda o \gamma o \sigma$, the word or form which expresses a thought; also the thought itself," Webster-Merriam, 1923). The first version of *Logo* was piloted with fifth-and sixth-grade math students at the Hanscom Field School in Lincoln, Massachusetts in the summer of 1967, under support of the U.S. Office of Naval Research. (Feurzeig and Papert, 1968).

In 1967-68, the ETD group designed a new and greatly expanded version of *Logo*, which was implemented by BBN software engineer Charles R. Morgan on the DEC PDP-1 computer. BBN scientist Michael Levin, one of the original implementers of *Lisp*, contributed to the design. From September 1968 through November 1969, the National Science Foundation supported the first intensive program of experimental teaching of *Logo*-based mathematics in elementary and secondary classrooms. (Feurzeig et al, 1969). The seventh grade teaching materials were designed and taught by Papert and Solomon. The second grade teaching materials were designed by Feurzeig and BBN consulting teacher Marjorie Bloom. The teaching experiments demonstrated in principle that *Logo* can be used to provide a natural conceptual framework for the teaching of mathematics in an intellectually, psychologically, and pedagogically sound way.

Classroom work to investigate the feasibility of using *Logo* with children under ten years old was first carried out at the Emerson School in Newton, Massachusetts in 1969. The students were a group of eight second-and-third graders (ages seven to nine) of average mathematical ability. The children began their *Logo* work using procedures with which most children are familiar. Examples included translating English into Pig Latin, making and breaking secret codes (e.g., substitution ciphers), a variety of word games (finding words contained in words, writing words backwards), question-answering and guessing games (Twenty Questions, Buzz, etc.). Children already know and like many problems of this sort. Children think at first that they understand such problems perfectly because, with a little prodding, they can give a loose verbal description of their procedures. But they find it impossible to make these descriptions precise and general, partly for lack of formal habits of thought, and partly for lack of a suitably expressive language.

The process of transforming loose verbal descriptions into precise formal ones becomes possible and, in this context, seems natural and enjoyable to children. The value of using *Logo* becomes apparent when children attempt to make the computer perform their procedures. The solutions to their problems are to be built according to a preconceived, but modifiable plan, out of parts which might also be used in building

other solutions to the same or other problems. A partial, or incorrect, solution is a useful object; it can be extended or fixed, and then incorporated into a large structure. Using procedures as building blocks for other procedures is standard and natural in *Logo* programming. The use of functionally separable and nameable procedures composed of functionally separable and nameable parts coupled with the use of recursion, makes the development of constructive formal methods meaningful and teachable.

The work of one of the seven-year-olds, Steven, illustrates this course of development. Steven, like all second-graders in the group, was familiar with the numerical countdown procedure accompanying a space launch. He had the idea of writing a COUNTDOWN program in Logo to have the same effect. His COUNTDOWN program had a variable starting point. For example, if one wished to start at 10, he would simply type COUNTDOWN 10, with the following result:

```
10 9 8 7 6 5 4 3 2 1 0 BLASTOFF!
```

He designed the program along the following lines. (The English paraphrase corresponds, line by line, to Logo instructions.)

```
TO COUNTDOWN a number
Type the number.
Test the number: Is it 0?
If it is, type "BLASTOFF!" and then stop.
If it is not 0, subtract 1 from the number, and call the result "newnumber".
Then do the procedure again, using :newnumber as the new input.
```

Steven's program, as written in Logo, followed this paraphrase very closely. (The colon preceding a number name designates its value. Thus, :NUMBER is 10 initially.)

```
TO COUNTDOWN :NUMBER
1 TYPE :NUMBER
2 TEST IS :NUMBER 0
3 IFTRUE TYPE "BLASTOFF!" STOP
4 MAKE "NEWNUMBER" DIFFERENCE OF :NUMBER AND 1
5 COUNTDOWN :NEWNUMBER END
```

(Note that the procedure calls itself within its own body — it employs recursion, however trivially.) Steven tried his procedure. He was pleased that it worked. He was then asked if he could modify COUNTDOWN so that it counted down by 3 each time, to produce

```
10 7 4 1 BLASTOFF!
```

He said "That's easy!" and he wrote the following program.

```
TO COUNTDOWN-3 :NUMBER
1 TYPE :NUMBER
2 TEST IS :NUMBER 0
3 IFTRUE TYPE "BLASTOFF!" STOP
4 MAKE "NEWNUMBER" DIFFERENCE OF :NUMBER AND 3
5 COUNTDOWN :NEWNUMBER
END
```

He tried it, with the following result,

```
COUNTDOWN-3 10
10 7 4 1 -2 -5 -8 -11 -14 -17 . . .
```

and the program had to be stopped manually. Steven was delighted! When he was asked if his program worked, he said "No." "Then why do you look so happy?" He replied "I heard about minus numbers, but up till now I didn't know that they really existed!"

Steven saw that his stopping rule in instruction line 2 had failed to stop the program. He found his bug — instead of testing the input to see if it was 0, he should have tested to see if it was negative. He changed the rule to test whether 0 is greater than the current number,

```
2 TEST IS :NUMBER LESS-OR-EQUAL 0
```

and then tried once more.

```
COUNTDOWN-3 10
10 7 4 1 BLASTOFF!
```

And now COUNTDOWN-3 worked.

Steven then worked on an "oscillate" procedure for counting up and down between two limits by a specified number of units. Two special and characteristic aspects of programming activity are shown in Steven's work — the clear operational distinction between the definition and the execution of a program, and the crucial mediating role served by the process of program "debugging."

Logo-controlled robotturtles were introduced in 1971, based on work of BBN and MIT consultant Mike Paterson. Screen turtles were introduced at MIT around 1972. BBN engineer Paul Wexelblat designed and built the first wireless turtle in 1972. He dubbed it "Irving." Irving was a remote-controlled turtle about one foot in diameter. It was capable of moving freely under *Logo* commands via a radio transceiver attached to a teletype terminal connected to a remote computer. Irving could be commanded to move forward or back a specified increment of distance, to turn to the right or left a specified increment of angle, to sound its horn, to use its pen to draw, and to sense whether contact sensors on its antennas have encountered an obstacle.

Children delighted in using Logo to command Irving to execute and draw patterns of various kinds. An early task started by having them move Irving from the center of a room to an adjoining room — this typically required a sequence of ten or fifteen move and turn commands. After Irving was somewhere out of view, the child's task was to bring Irving back home, to its starting point in the original room. This had to be done only through using *Logo*, without the child leaving the room or peeking around the doorway! The child had a complete record of the sequence of commands she had used since each command she had typed was listed on the teletype printer.

This task was fascinating and, for all but the most sophisticated children, quite difficult. The notion that there is an algorithm for accomplishing it was not at all obvious. They knew that Move Forward and Move Back are inverse operations, as are Right Turn and Left Turn. But they didn't know how to use this knowledge for reversing Irving's path. The algorithm — performing the inverse operations of the ones that moved Irving away, but performing them in the reverse order — is an example of a mathematical idea of considerable power and simplicity, and one that has an enormous range of application. The use of the turtle made it accessible to beginning students. Once the children were asked how to make Irving just undo its last move so as to get to where it had been the step before the last, most children had an "Aha" experience, and immediately saw how to complete the entire path reversal. The algorithm was easily formalized in *Logo*. This paved the way to understanding the algebra procedure for solving linear equations. The algorithm for solving a linear equation is to do the inverse of the operations that generated the equation, but in the reverse order — the same algorithm as for path reversal.

These examples illustrate the kinds of interactions that have been fostered through work with *Logo*. There is no such thing as a typical example. The variety of problems and projects that can be supported by *Logo* activities, at all levels of mathematical sophistication, is enormous. *Logo* has been the center of mathematics, computer

science, and computational linguistics courses from elementary through undergraduate levels. (Feurzeig and Lukas, 1972; Abelson and DiSessa, 1983; Lukas and Lukas, 1986; Goldenberg and Feurzeig, 1987; Lewis, 1990; Cuoco, 1990; Harvey, 1997)

In 1970, Morgan and BBN software engineer Walter Weiner implemented subsequent versions of *Logo* on the DEC PDP-10 computer, a system widely used in universities and educational research centers. Throughout 1971-74, BBN made DEC 10 *Logo* available to over 100 universities and research centers who requested it for their own research and teaching. In 1970, Papert founded the Logo Laboratory at MIT, which further expanded the use of Logo in schools. The advent of micro-computers, with their wide availability and affordability, catapulted *Logo* into becoming one of the world's most widely used computer languages during the 1970s and 1980s and, especially in Europe, currently.

Algebra Workbench

Introductory algebra students have to confront two complex cognitive tasks in their formal work: problem-solving strategy (deciding what mathematical operations to perform in working toward a solution) and symbolic manipulation (performing these operations correctly). Because these two tasks — each very difficult in its own right for beginning students — are confounded, the difficulties of learning algebra problem solving are greatly exacerbated. To address these difficulties, Feurzeig and BBN programmer Karl Troeller developed the *Algebra Workbench*. The key idea was to facilitate the acquisition of problem-solving skills by sharply separating the two tasks and providing students automated facilities for each (Feurzeig and Richards, 1988).

The program includes powerful facilities for performing the symbolic manipulations requested by a student. For example, in an equation-solving task it can add, subtract, multiply, or divide both sides of the equation by a designated expression, expand a selected expression, collect terms in an expression, do arithmetic on the terms within an expression, and so on. This enables students to focus on the key *strategic* issue: choosing what operation to do next to advance progress toward a solution. The program will carry out the associated manipulations. The *Workbench* has a variety of facilities to support students' work. It can advise the student on what would be an effective action at any point; it can check a student's work for errors, either at any point along the way, or after the student completes his work; and it can demonstrate its own solution to a problem.

The *Algebra Workbench* was designed for use with formal problems in the introductory course, e.g., solving equations and inequalities, testing for equivalence of expressions, factoring, simplification, etc. It can provide a student with a set of problems, such as: $(n-1)/(n+1) = 1/2$, or $(16x+9)/7 = x+2$, or $10y-(5y+8) = 42$. It can accept other problems posed by the student or teacher. In demonstrating the working out of a problem, it employs pattern recognition and expression simplification methods at a level that can be readily emulated by beginning students. Its facilities for expression manipulation, demonstration, explanation, advice, and critical review are available at the student's option at any time during a problem interaction. See Figure 13.1.

Another student, who worked on the same problem, replaced $2*(n-1)$ by $2n-1$ on the left side of the equation, transforming it into $2n-1 = n+1$. When the student announced that he had solved the equation with the result $n = 2$, the system reviewed his work and pointed out his incorrect expansion of the left-side expression during the initial step.

A commercial version of the *Algebra Workbench* was developed by BBN scientist John Richards and Feurzeig under support of Jostens Learning Corporation in 1993.

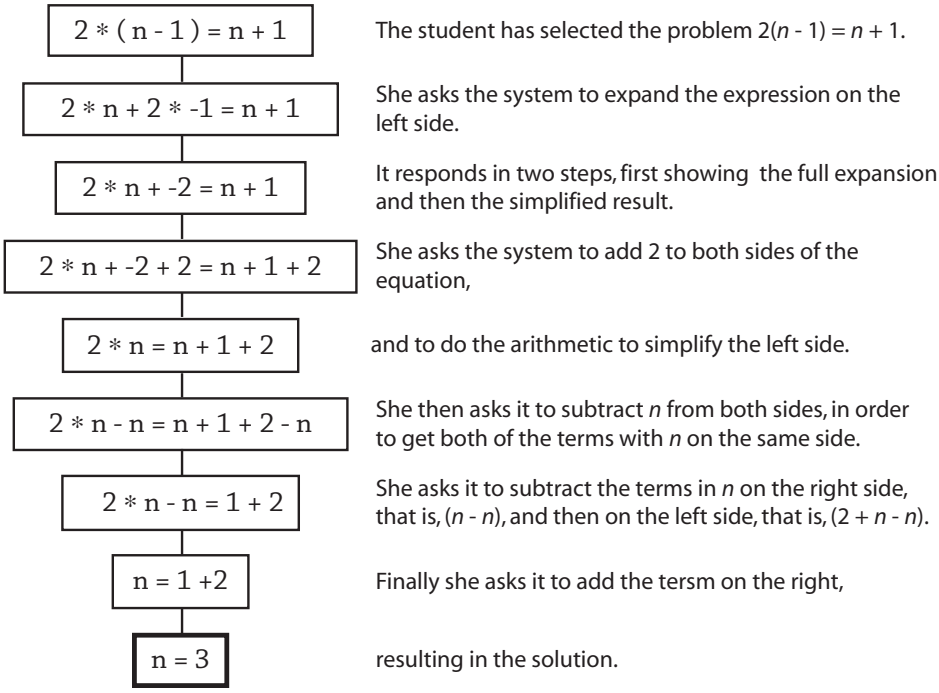


Figure 13.1 Transcript illustrating use of the *Algebra Workbench*.

The system was never released because Jostens was absorbed into a larger publishing firm that was not interested in educational software products.

Non-prescriptive learner-centered systems with instructional supports (like the *AlgebraWorkbench*) can help students acquire the knowledge and skills they need to use mathematics. The learning benefits of these tools are further enhanced when they are augmented by open-ended learner-driven tools like Logo that enable students to undertake projects with mathematically rich content in broader contexts.

Buggy and Debuggy

Buggy was a computer game based on the diagnostic interactions of a teacher and a simulated student. The teacher is presented with an arithmetic problem that the "student" has done incorrectly, due to an underlying bug in one of the arithmetic subprocedures. The teacher's task is to diagnose the bug by providing strategic test problems for the "student" to solve in order to discover the bug. The problems given by the teacher are answered by the "student" using the buggy procedure. When the teacher thinks he knows the answer, *Buggy* asks the teacher to describe what he thinks the bug is by responding to a five-problem test in the same way that the student would do them, i.e., by simulating the student's bug (Brown and Burton, 1978b).

The following is a protocol illustrating the use of *Buggy* by a team of teachers. The teachers' inputs are preceded by vertical lines on the left.

I HAVE CHOSEN A BUG. HERE IS AN EXAMPLE OF THE BUG.

```
 17
+5
---
 13
```

NOW YOU CAN GIVE ME PROBLEMS TO DETERMINE THE BUG.

```
|   18     43
|  +6    +79
|  ---   ---
|   15     23
```

| Got the bug!
HERE IS A PROBLEM TO TEST YOUR THEORY ABOUT THE BUG. WHAT IS:

```
|   21
|  +39
|  ---
|   51
```

THAT'S NOT THE BUG I HAVE. USING MY BUG:

```
|   21
|  +39
|  ---
|   15
```

TRY GIVING ME SOME MORE EXAMPLES.

```
|     51   99    68
|  +1707  +99   +9
|     ---  ---  ---
|     21   36    23
```

| Got the bug!

PLEASE DESCRIBE THE BUG YOU THINK I HAVE.
| The student is adding all the digits together.

HERE ARE SOME PROBLEMS TO TEST YOUR THEORY. WHAT IS:

```
|   33  1091    8   28    90
|  +99   +60  +34  +70    +6
|  ---   ---  ---  ---   ---
|   24    17   15   17    15
```

VERY GOOD. MY DESCRIPTION OF THE BUG IS: THE STUDENT
ALWAYS SUMS UP ALL THE DIGITS WITH NO REGARD TO COLUMNS.

The *Buggy* data base incorporated a substantial number of typical student bugs in addition and subtraction, based on empirical studies of the buggy behaviors of elementary students of arithmetic. The data base consisted of 20,000 problems performed by 1300 students (Brown et al, 1977).

The work on *Buggy* motivated the development of *Debuggy*, a diagnostic modeling system for automatically synthesizing a model of a student's bugs and misconceptions in basic arithmetic skills (Brown and Burton, 1978). The system introduced procedural networks as a general framework for representing the knowledge underlying a procedural skill. Its challenge was to find a network within this representation that identified the particular bugs in a student's work as well as the underlying misconceptions in the student's mental model.

Summit

In 1983, Feurzeig and BBN cognitive scientist Barbara White developed an articulate instructional system for teaching arithmetic procedures (Feurzeig and White, 1984). *Summit* employed computer- generated speech and animated graphics to aid elementary school children learn standard number representation, addition, and subtraction. The system comprised three programs. The first was an animated bin model to help students understand place notation and its relationship to standard addition and subtraction procedures. It could display up to four bins on the screen: a thousands bin, a hundreds bin, a tens bin, and a ones bin. The bin model in Figure 13.2 represents the number 2934.

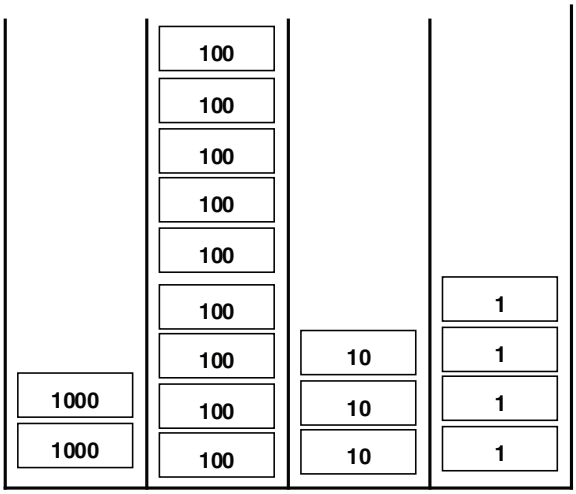


Figure 13.2 The *Summit* bin model.

The student could give commands such as ADD 297 or SUBTRACT 89 and the program would cause the appropriate numbers of icons to be added or subtracted from the appropriate bins graphically, in a manner analogous to the standard procedures for addition and subtraction. When a bin overflowed, the program animated the carrying process. Similarly, in subtraction when there were not enough icons in the appropriate bin, the model animated the process of "borrowing" (replacing). The program explained its operations to the student using computer-generated speech.

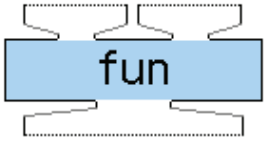
The second program in *Summit* demonstrated the standard (right-to-left) algorithms for addition and subtraction. It displayed a problem on the screen and talked its way through to the solution, explaining its steps along the way. The demonstrations were sequenced, starting with single-digit problems and working up to four-digit problems. They included explanations of the more difficult cases, such as borrowing across zeros.

The third program in *Summit* gave students an opportunity to practice addition and subtraction problems aided by feedback and guidance. A student, using a computerized work tablet, worked through a problem using keyboard arrows to control the positioning of number entries. If the student made an error, he was presented with the choice of trying it again or seeing a *Summit* demonstration of how to solve the problem.

Summit was an exploratory research project. Few elementary schools had computers, and virtually none had computer-generated speech facilities. The *Summit* programs were written in Logo on the Apple II computer. The system was tested with fourth-grade students in a Cambridge, Massachusetts school in 1983 and proved effective in helping students learn place notation and the standard algorithms for addition and subtraction.

Function Machines

Function Machines is a visual programming environment for supporting the learning and teaching of mathematics. It is a dynamic software realization of a function representation that dates back to the 1960s — the notation that expresses a function as a "machine" with inputs and outputs.



Function Machines employs two-dimensional representations — graphical icons — in contrast with the linear textual expressions used for representing mathematical structures in standard programming languages. The central *Function Machines* metaphor is that a function, algorithm, or model is conceptualized as a machine, displayed as shown in the figure, where fun represents a function. The two funnel-shaped objects at the top of the figure are called input hoppers; the inverse one at the bottom is called an output spout.

Machines can have multiple inputs and outputs. The output of a machine can be piped from its output spout into the input hopper of another machine. A machine's data and control outputs can be passed as inputs to other machines through explicitly drawn connecting paths. The system's primitive constructs include machines corresponding to the standard mathematical, graphics, list processing, logic, and I/O operations found in standard languages. These are used as building blocks to construct more complex machines in a modular fashion. Any collection of connected machines can be encapsulated under a single icon as a higher-order "composite" machine; machines (programs) of arbitrary complexity level can be constructed (Feurzeig, 1993, 1994a).

Execution is essentially parallel — many machines can run concurrently. The operation of recursion is made visually explicit by displaying a separate window for each instantiation of the procedure as it is created, and erasing it when it terminates. The hierarchical organization of programs implicit in the notion of a composite machine fosters modular design and helps to organize and structure the process of program development. Like a theater marquee, the system shows the passage of data objects into and out of machines, and illuminates the active data and control paths as machines are run. Thus, it visually animates the computational process and makes the program semantics transparent.

Function Machines supports students in a rich variety of mathematical investigations. Its visual representations significantly aid one's understanding of function, iteration, recursion, and other key computational concepts. It is especially valuable for developing mathematical models. To understand a model, students need to see the model's inner workings as it runs. At the same time they need to see the model's external behavior, the outputs generated by its operation. Function Machines supports both kinds of visualizations. The use of these dual-linked visualizations has valuable learning benefits.

Function Machines was designed in 1987 by members of the BBN Education Department including Feurzeig, Richards, scientists Paul Horwitz and Ricky Carter, and software developer Sterling Wight. Wight implemented *Function Machines* for Macintosh systems in 1988. A new version, designed by Feurzeig and Wight, was completed in 1998. It is implemented in C++ and runs both on Windows and Macintosh systems.

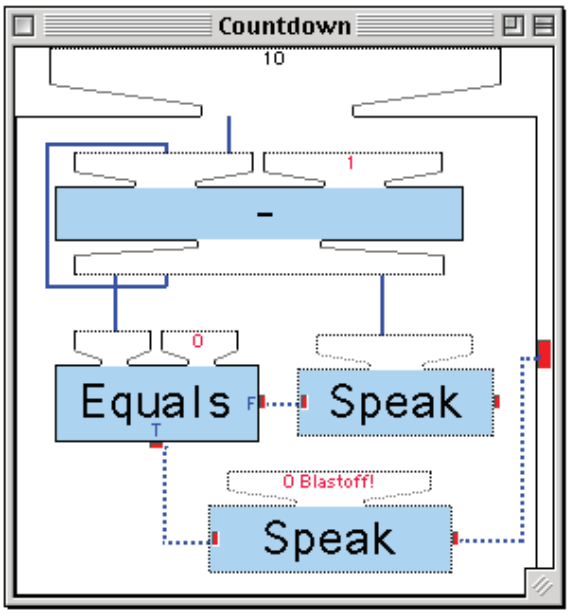


Figure 13.3 ***Function Machines*** **Countdown program.**

Figure 13.3 shows a Function Machines Countdown program, corresponding to the first of the Logo Countdown programs written by second grader Steven, which was illustrated on page 292.

The Countdown composite machine comprises four machines: a subtraction machine (−), an Equals machine, and two Speak machines. The Speak machines use speech generator software to "speak" their inputs. Countdown has as its input the value 10, which is sent to the left hopper of the −machine. The right hopper of that machine has as its input the constant 1 (the decrement of the subtraction operation). The output of the −machine is sent to three places: the left hopper of the Equals machine, the hopper of a Speak machine, and back to its own left hopper as its next input. When Countdown runs, it tests to see if the output of the −machine is equal to 0. If not, it "speaks" that output value; otherwise (if the output is 0), it speaks "0 Blastoff!" and triggers the stop button (in red, on the right border). During each iteration, the output of the −machine is decreased by 1, and the process is repeated, terminating when the output becomes 0.

Function Machines has been used in elementary and secondary mathematics classrooms in the United States, Italy, and Germany (Feurzeig, 1997, 1999; Cuoco 1993, Goldenberg, 1993). The following activities from a fifth-grade pre-algebra sequence (Feurzeig, 1997) illustrate the spirit and flavor of the approach. Students begin using the *Function Machines* program Predicting Results shown in Figure 13.4. Students enter a number in the Put In machine. It is sent to the +machine, which adds 2 to it; the result is then sent to the ∗machine, which multiplies it by 5; that result is sent to the Get Out machine.

The display window on the right shows a series of computations; thus, when students put in 5 the machine gets out 35. The program uses speech. When 5 is entered, the program gives the spoken response Put In 5 and, as the result of that computation is printed, the program responds Get Out 35. Similarly, the input 7 yields 45, and 2 yields 20. The figure shows the beginning of a computation with Put In −1. The +2 machine is ready to fire. Perhaps some students will predict that the calculation will result in Get Out 5.

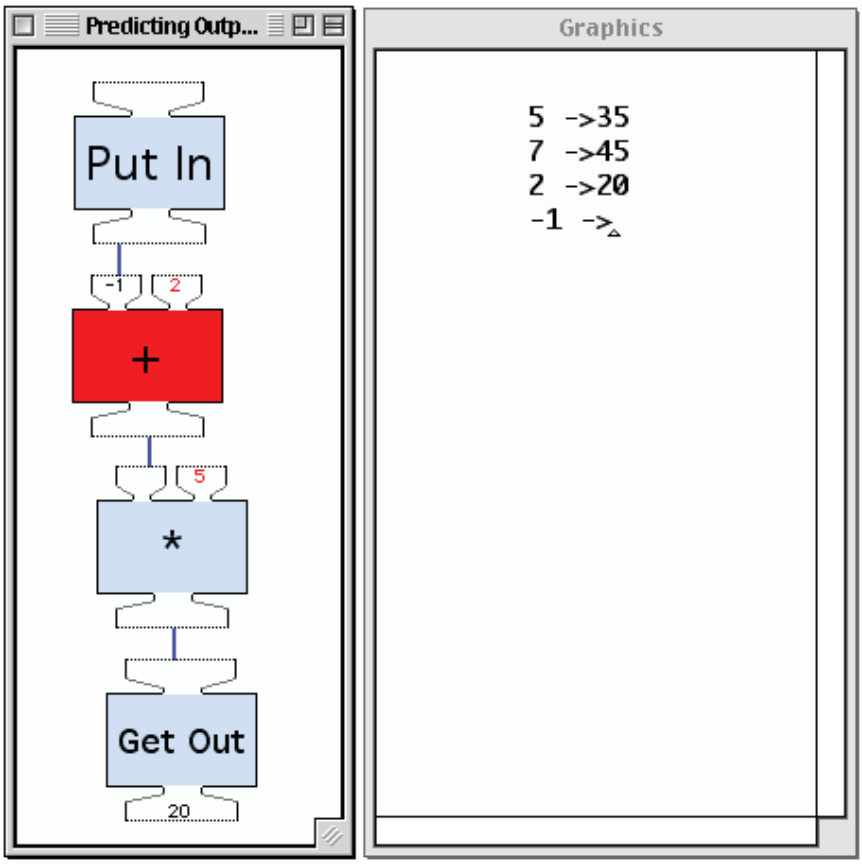


Figure 13.4 Predicting Results *Function Machine*.

The next activity, Guess My Rule, is shown in Figure 13.5. The Put In and Get Out machines are as above. The Mystery machine, however, is new. It has concealed inside it two calculation machines (+, -, *, or / machines).

The activity proceeds as follows. Students are organized into groups of five. Each group has a computer with the Guess My Rule program installed in it. Group A creates two calculation machines and conceals them inside the Mystery machine in Group B's computer. The task of Group B is to run Group A's *Guess My Rule* program with a variety of inputs, and to determine from the resulting outputs, which two calculation machines are inside the *Mystery* machine. At the same time, Group B makes a pair of calculation machines and conceals them inside the *Mystery* machine of Group A, and the kids in Group A try to determine what's inside. This "guessing" game exercises students' thinking about arithmetic operations. Most kids found this challenge a great deal of fun. The figure on the right above shows a session of *Guess My Rule* in progress. The display shows that inputs 0, 1, 2, and 5 produce outputs of 18, 20, 22, and 28 respectively. The current input, 10, is about to be run by the *Mystery* machine. Can the students guess what output it will generate?

Figure 13.6 shows the inner contents of the *Mystery* machine used in the above session- the two machines $+9$ and $*2$. It is not a trivial task for fifth-graders to infer this or other possible solutions. (For example, instead of the machines $+9$ and then $*2$, an equivalent *Mystery* machine is $*2$ and then $+18$.)

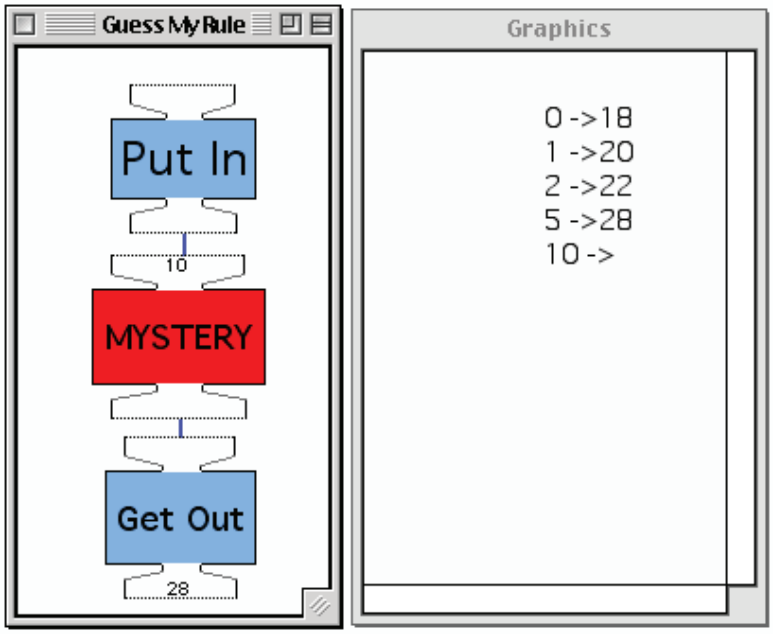


Figure 13.5 Guess My Rule *Function Machine*.

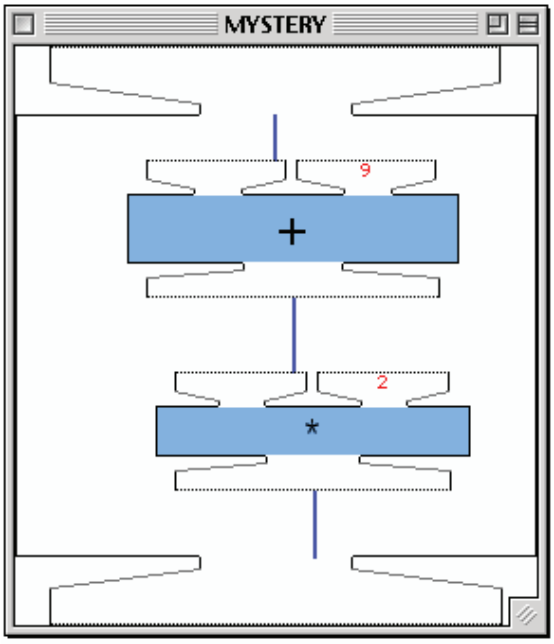


Figure 13.6 Inner contents of the *Mystery* Machine.

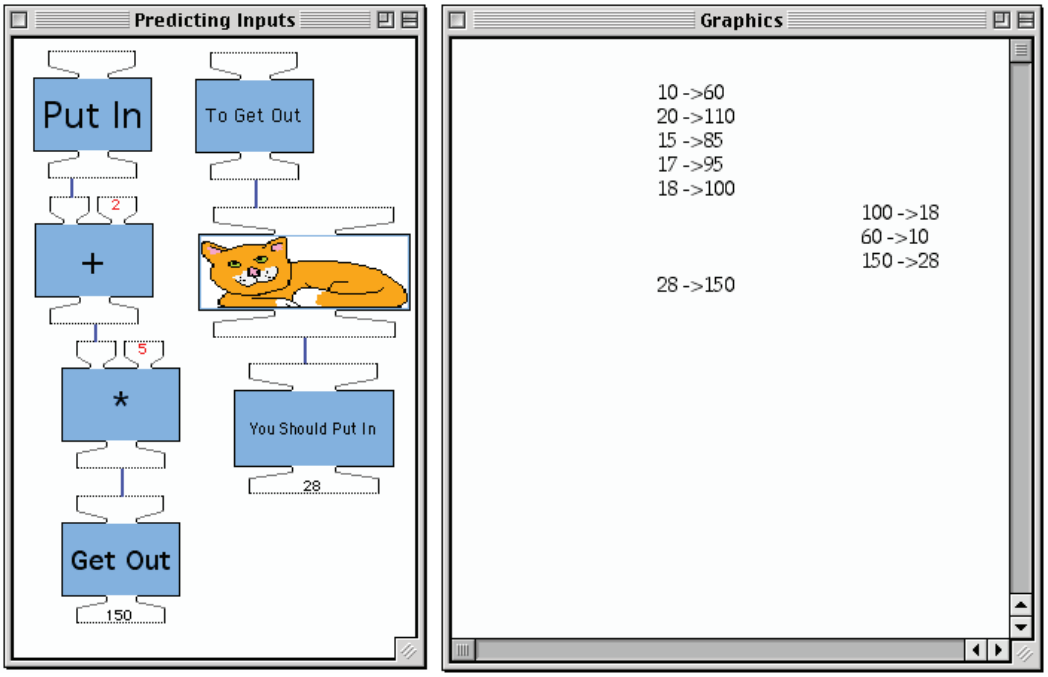


Figure 13.7 Predicting Inputs machine.

Figure 13.7 shows the introduction of a new problem. The left window shows the same sequence of machines used in the first figure in the instructional sequence. However, the new challenge from running these machines, is not to determine what output will be produced by a given input, as previously. *Instead*, the task is to answer the *opposite* question: what number must be input to produce a specified output?

For example, using the given +2 and ∗5 machines, what number must be Put In to Get Out 100? Understandably, this seems to the kids a much more difficult problem. They are encouraged to approach the task by trial and error. The display screen on the right shows the results of a trial-and-error sequence aimed at finding what input yields an output of 100. The input 10 yields 60, so perhaps a larger input is needed. An input of 20 yields 110. So perhaps some input between 10 and 20 is called for. 15 yields 85; 17 yields 95, and then-"Hooray! 18 works!" The sequence required five trials. Using the same pair of machines with other output targets, kids attempt to determine the correct inputs in as few trials as possible. They are delighted when they can zoom in on the answer in three or four trials and sometimes even two!

At this point, the two new machines in the left window, To Get Out and You Should Put In, are introduced. The inputs and outputs printed by these machines are shown on the right half of the display screen. Just after the printout on the left was produced, showing that an input of 18 produces an output of 100, the two new machines are run. An input of 100, the target output in the previous problem, is given to the *To Get Out* machine. This input is piped to the *You Should Put In* machine which produces the printout of 18 shown on the display. Somehow, using this pair of machines, all one had to do was give it the desired output and it produced the corresponding input-and in just a single trial! Also, as the subsequent printout from this machine shows, it confirms what was shown before, that to get an output of 60 requires an input of 10. And it could be used to confirm the other pairs also-that an output of 110 calls for an input of 20, etc. Instead, however, it is used to assert a new claim, that to get an output

of 150 requires an input of 28. The last printout on the left of the display, generated by running the four machines on the left, confirms this-an input of 28 indeed yields an output of 150.

The problem for the students is: how does the *You Should Put In* machine know, right from the start, what input one needs to put in to get a desired output? What kind of magic does it use? The kids are not expected to fathom the answer. Instead, they are shown the inner contents of the machine, seen in Figure 13.8.

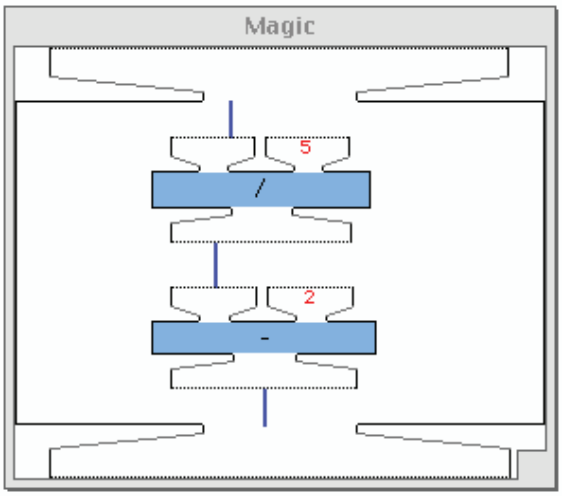


Figure 13.8 Magic machine.

As the figure shows, the solution to the problem is to do the inverse computations in the opposite order. (This is the same algorithm used for turtle path reversal in *Logo*. The original computation sequence was: put in a number (say N), add 2 to it, multiply the result by 5, and get out the answer. To undo this sequence, one computes operations in the reverse order from the original sequence, replacing the original operations with their inverses, as follows. Put in the desired answer, divide it by 5, subtract 2 from the result and get out N. This is essentially the algorithm for solving linear equations in algebra, and in this context, fifth-graders can understand its sense and purpose.

MultiMap

MultiMap is a software tool for introducing experimental mathematics into the pre-college curriculum through investigating the dynamics of planar maps. It aimed to introduce students to the concept of a map, seen as a transformation of the plane onto itself. The *MultiMap* program transforms figures on the computer screen according to rules (i.e., *maps*) specified by the user. Linear maps can be created from primitive operations such as rotation, scaling, and translation. The rules can also be expressed as mathematical functions. Though *MultiMap* is accessible to middle-school students, it also provides professional mathematicians an environment in which they can encounter challenging questions. It was originally designed as a general-purpose tool to support a high-school curriculum in mathematical chaos. It was developed by Horwitz, Feurzeig, and MIT consultant Michael Eisenberg, and implemented on the Macintosh by BBN software developer Bin Zhang.

MultiMap has a direct manipulation iconic interface with extensive facilities for creating maps and studying their properties under iteration. The user creates figures

(such as points, lines, rectangles, circles, and polygons), and the program graphically displays the image of these figures transformed by the map, possibly under iteration. *MultiMap* allows one to make more complex maps out of previously created maps in three distinct ways: by composition, superposition, or random selection of submaps. It includes a facility for coloring maps by iteration number, a crosshair tool for tracing a figure in the domain to see the corresponding points in the range, a zoom tool for magnifying or contracting the scale of the windows, and a number of other investigative facilities. *MultiMap* also enables the generation and investigation of nonlinear maps that may have chaotic dynamics. The program supports the creation of self-similar fractals, allowing one to produce figures that are often ornate and beautiful. Using *MultiMap*, and with minimal guidance from an instructor, students have discovered such phenomena as limit cycles, quasi-periodicity, eigenvectors, bifurcation, fractals, and strange attractors (Horwitz and Eisenberg, 1991).

When *MultiMap* is called up, the screen is divided into three windows. The domain window enables the user to draw shapes such as points, lines, polygons and rectangular grids, using the iconic tools in the palette on the left. The range window is used by the computer to draw one or more iterates of whatever shapes are drawn in the domain. The map window specifies the transformation of points in the domain that "maps" them into the range. The user controls what the computer draws in the range window by specifying a mapping rule, expressed in the form of a geometric transformation. The map is to be performed on the entire plane, a user-definable portion of which is displayed in the domain window. For example, the default transformation, called the "identity map," simply copies the domain one-for-one into the range.

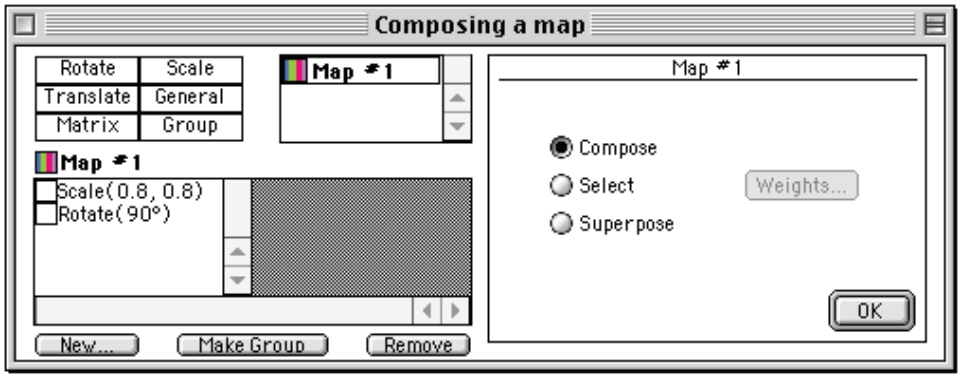


Figure 13.9 Composing a map.

In Figure 13.9, the user has entered a rectangle in the domain window and has then specified a map composed of two submaps, a scale and a rotation. Scale (0.8, 0.8) scales the rectangle to 0.8 of its original size in both x and y. Rotate (90 °) rotates the rectangle 90 degrees about the origin. In a composition map such as this, the transformations are performed in order. Thus the rectangle is scaled and then rotated. This is an iterated map. The user has specified that the map is to be performed 4 times (after including the identity map), with a distinct color for successive iterations (light blue, green, red, and pink). The range window shows the result of the mapping.

Using *MultiMap*, students from local high schools created and investigated simple maps built on the familiar operations of rotation, scaling, and translation. They then investigated the behavior under iteration of more-complex maps, including maps that produce beautiful fractals with self-similar features at all levels, random maps that generate regular orderly structures, and maps that, though deterministic, give rise

to unpredictable and highly irregular behaviors (chaos). Students were introduced to rotation, scale, and translation maps during their first sessions, and to their properties under composition and iteration.

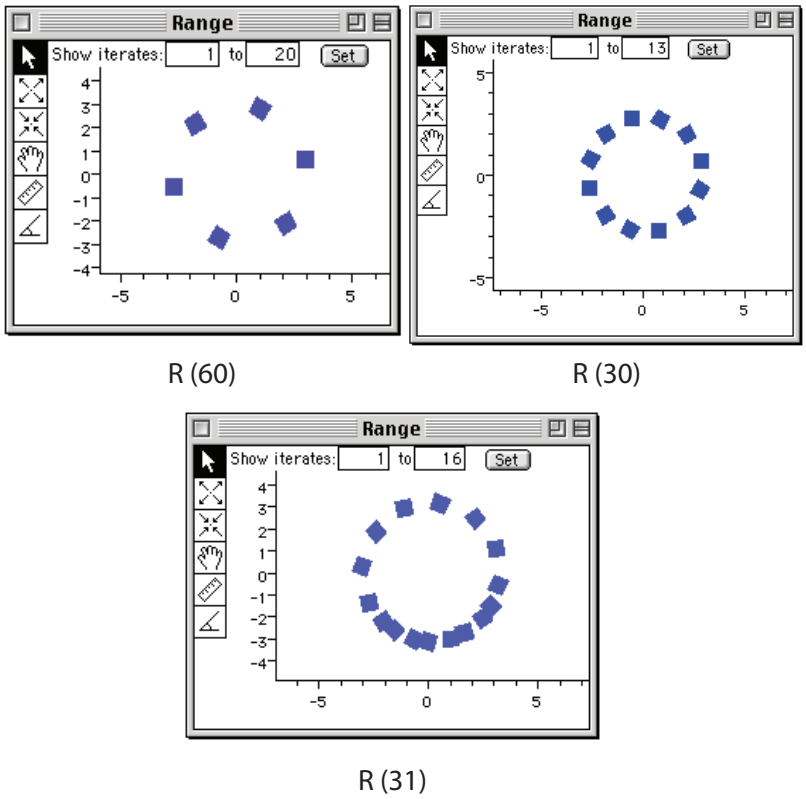


Figure 13.10 A *MultiMap* session.

The session shown in Figure 13.10 illustrates the use of *MultiMap* by two students , Kate and Fred, working together on an investigation of rotational symmetry (Horwitz and Feurzeig, 1994). They began by drawing a square and rotating it by 60 degrees, as shown in the top left screenshot. They noted that the 6 copies of the square lay around a circle centered at the origin, and that, though the map was iterated 20 times, after the first 6 iterations the others wrote over the ones already there. They were then asked what the result of a rotation by 30 degrees would be. Kate said that there would be 12 copies of the square instead of 6, no matter how many iterations. They confirmed this, as shown in the top right screenshot. The instructor then asked "What would happen if the rotation angle had been 31 degrees instead of 30?" Fred said "There will be more squares-each one will be one more degree away from the 30 degree place each time, so the squares will cover more of the circle." *MultiMap* confirmed this, as shown in the bottom screenshot.

Instructor: "The picture would be less crowded if the square was replaced by a point." Fred made this change. The result, after 100 iterations, is shown on the left of Figure 13.11. Since there was still some overlap, the instructor said "After each rotation let's scale x and y by .99. That will bring the rotated points in toward the center a little more at each iteration." Ann then built an R(30°)S(0.99, 0.99) composite map. The effect of the scaling is shown on the right.

Fred said "Now the points come in like the spokes of a wheel with 12 straight arms."

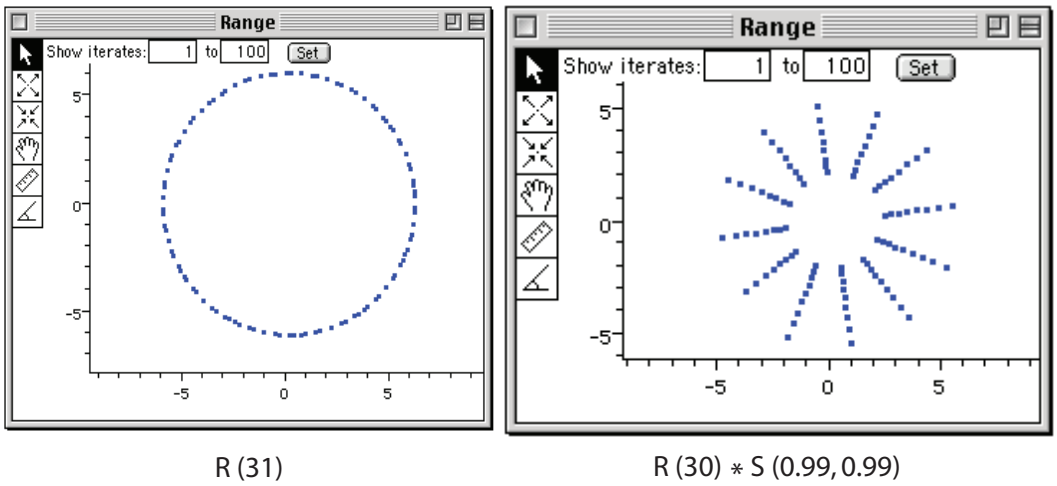
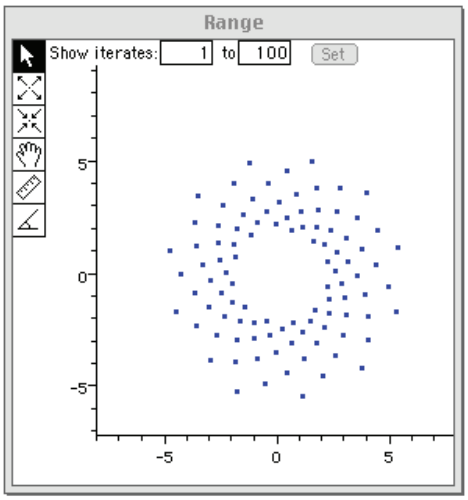


Figure 13.11 Continuing the session.

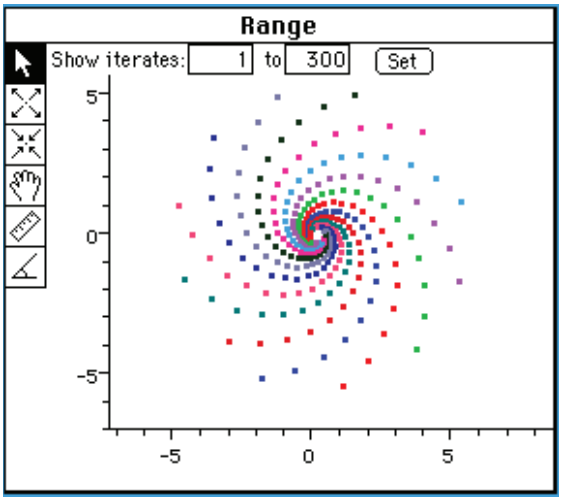
The instructor agreed and asked what would happen if the rotation were 31 degrees instead of 30. Fred replied "It would be almost the same but the points would not be on straight lines. They would curve in a little each time." He tried this. The result is shown in the top left of Figure 13.12. Kate said "The spokes have become spiral arms." When asked how many arms there were, she said "It looks like 12." The instructor said "Let's check that visually by making the points cycle through 12 colors repeatedly so that successive points have distinct colors." The result is shown in the top right of Figure 13.12. Kate: "Oh, how beautiful! And now each arm of the web has the same color." Instructor, "Right, so we can clearly see that the web has 12-fold symmetry."

Instructor: "What do you think will happen if the rotation is 29 degrees instead of 31 degrees?" Kate: "I think it will be another spiral, maybe it will curve the other way, counter-clockwise. But I think it will still have 12-fold symmetry. Here goes!" The result is shown in the middle left of Figure 13.12. Instructor: "Right! It goes counter-clockwise and it does have 12-fold symmetry. Very good! Now let's try a rotation of 27 degrees. What do you think will happen?" Kate: "I think it will be about the same, a 12-fold spiral web, maybe a little more curved." The result is shown in the middle right of Figure 13.12. Instructor: "So what happened?" Kate: "It looks like a 12-fold spiral web but why aren't the colors the same for each arm?". Instructor: "Right! It goes counter-clockwise and it does have 12-fold symmetry."

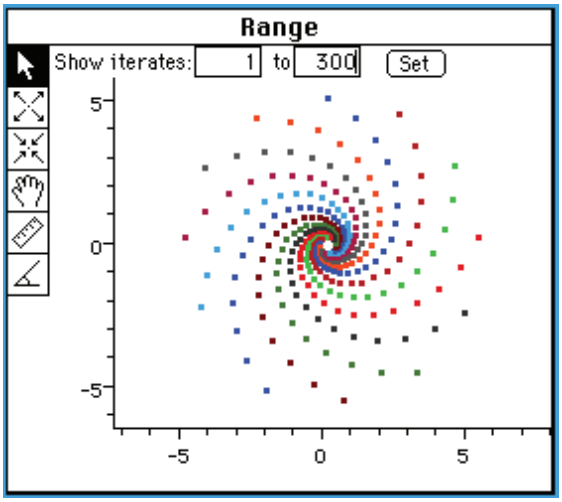
Instructor: "It might be that we don't have enough detail — let's get a more detailed picture by changing the scale from .99 to .999, and increasing the number of iterations from 300 to 600. See if that makes a difference." The result, after 600 iterations, is shown at the bottom left of the figure. Kate: "Wow, it looks very different now! There are many more than 12 arms, but they're all straight, and each arm still has many different colors." Instructor: "There's obviously much more than 12-fold symmetry here. Any idea what it is?" Fred: "120." Instructor: "Why do you say that?" Fred: "Because 360 and 27 have 9 as their greatest common divisor. So 360 divided by 9 is 40, and 27 divided by 9 is 3, and 40 times 3 is 120." Instructor: "What do you think, Kate?" Kate: "I don't know but I counted the arms and it looks like there are 40." Instructor: "Let's see if that's right. Reset the color map so that the colors recycle every 40 iterations instead of every 12 iterations." The students changed the color ramp. The result, after 600 iterations, is shown below on the right.



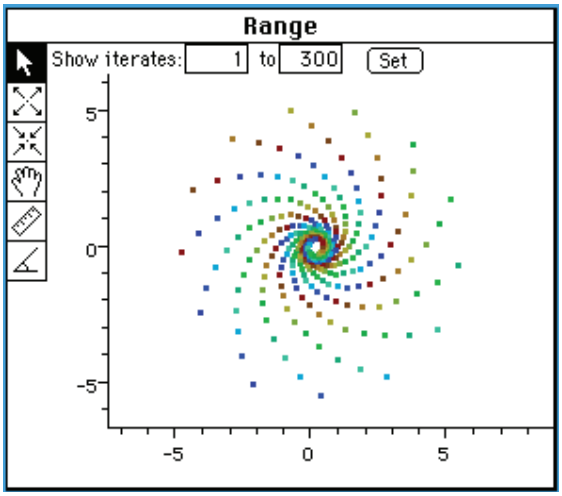
R (31) * S (.99, .99)



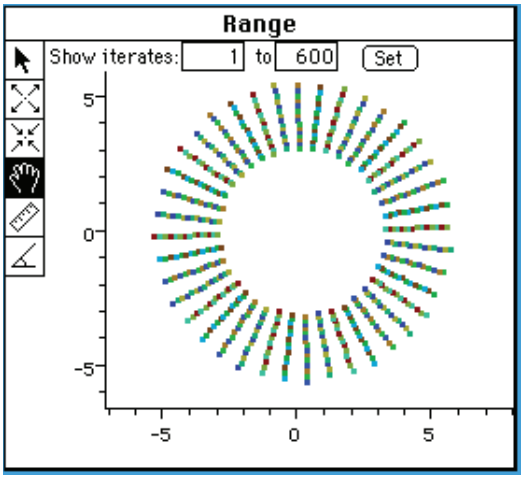
R (31) * S (.99, .99) 12-color ramp



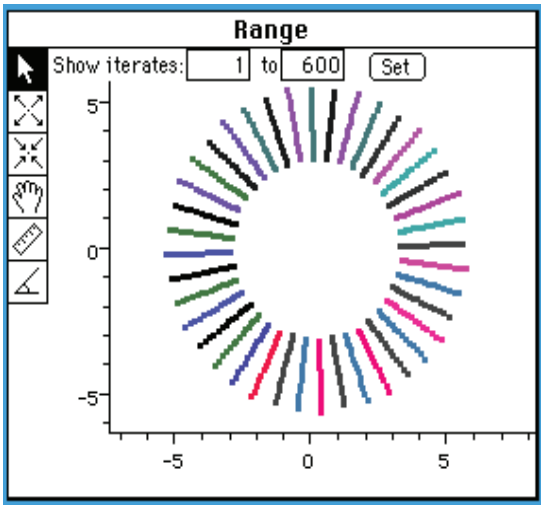
R (29) * S (.99, .99) 12-color ramp



R (27) * S (.99, .99) 12-color ramp



R (27) * S (.999, .999) 12-color ramp



R (27) * S (.999, .999) 40-color ramp

Figure 13.12 More of the session.

Kate: "Now each arm is the same color. So there is 40-fold symmetry." Fred: Is 120 wrong? Instructor: "No, 120 isn't wrong but it's not the only or the best answer. 240 and 360 would work and so would any other multiple of 120. But the real question is: what is the smallest one? The way to view the problem is this: what is the least number of times you have to go around a circle in 27-degree increments to come back to where you started? Or, to put it another way, what is the smallest integer N such that the 27 times N is an exact multiple of 360? The answer is 40 because 40 times 27 equals 1080, which is 3 times 360. No integer less than 40 will work." Fred: "I understand. Now I can do the problem for any angle."

MultiMap was developed in the NSF project "Advanced Mathematics from an Elementary Viewpoint" (Feurzeig, Horwitz, and Boulanger, 1989). Its use enabled students to gain insights in other visually rich mathematical explorations such as investigations of the self-similar cyclic behavior of the limiting orbits of rotations with non-uniform scaling (Horwitz and Feurzeig, 1994).

ELASTIC

The *ELASTIC* software system is a set of tools for exploring the objects and processes involved in statistical reasoning. It was developed for use in a new approach to teaching statistical concepts and applications in a high-school course called Reasoning Under Uncertainty (Rubin et al, 1988). *ELASTIC* supports straightforward capabilities for entering, manipulating, and displaying data. It couples the power of a database management system with novel capabilities for graphing histograms, boxplots, scatterplots, and barplots. *ELASTIC* provides three special interactive visual tools that serve students as a laboratory for exploring the meaning underlying abstract statistical concepts and processes. One tool, Stretchy Histograms, shown in Figure 13.13, enables students to manipulate the shape of a distribution represented in a histogram.

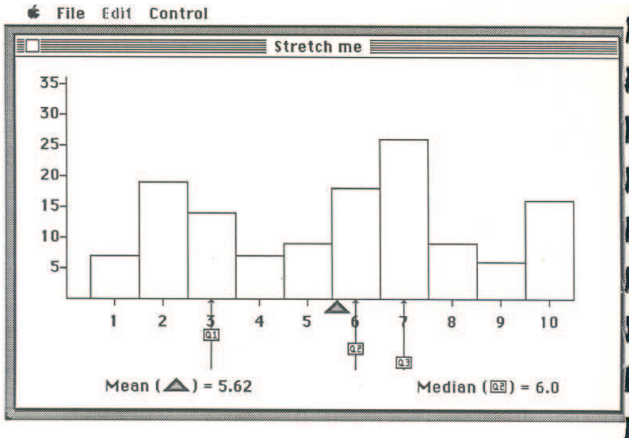


Figure 13.13 Stretchy Histograms.

Using the mouse, they can stretch or shrink the relative frequencies of classes in a distribution and watch as graphical representations of mean, median, and quartiles are dynamically updated to reflect those changes. In this way, students can explore the relationships among a distribution's shape, its measures of central tendency and its measures of variability. They can also use the program to construct histograms that represent their hypotheses about distributions in the real world. Another tool, Sampler, shown in Figure 13.14, is a laboratory for exploring the process of sampling.

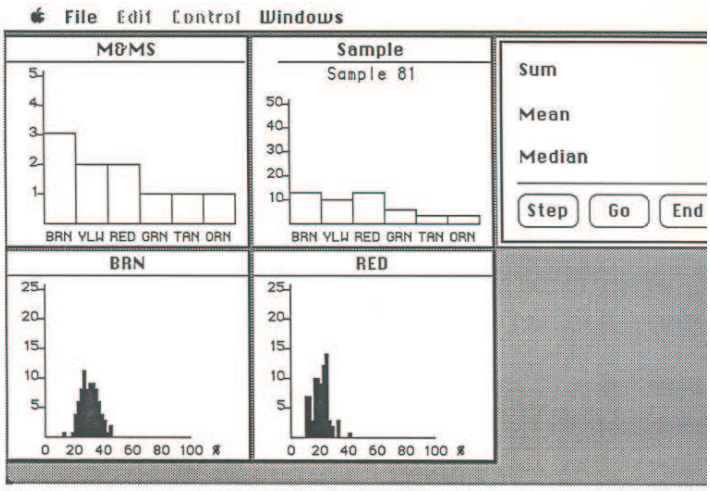


Figure 13.14 Sampler.

A student or teacher can create a hypothetical population and then draw any number of samples of any size from it. Sampler displays graphs of the population model, the sample that is currently being drawn, and summary data on the set of samples, including a distribution of sample means. Students can use Sampler to run experiments, for example by taking repeated samples and watching as the distribution of sample means or medians grows. They can also compare the distribution of samples to real world samples they have generated. A third tool, Shifty Lines, shown in Figure 13.15, is an interactive scatterplot that enables students to experiment with line fitting, adjusting the slope and y-intercept of a regression line, and observing the resulting fit of the line to the data points.

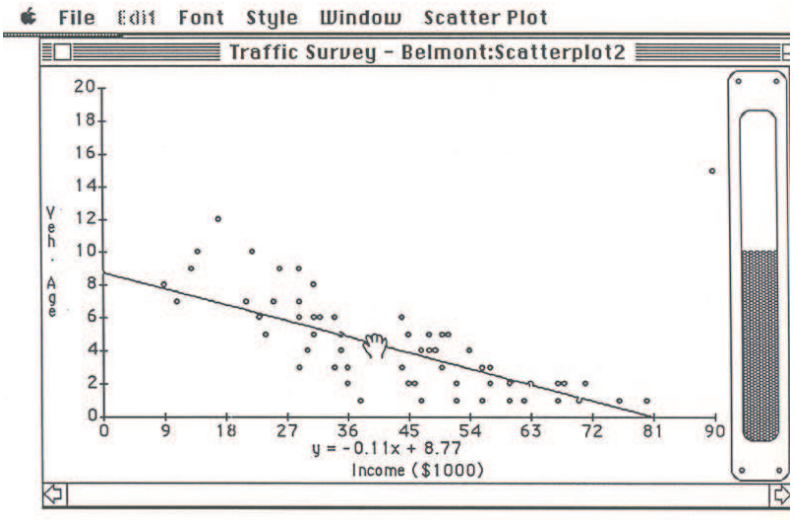


Figure 13.15 Shifty Lines.

The software provides students with several kinds of information: 1) dots for each data point, 2) a straight line that represents an hypothesis about the x-y relationship, 3)

a thermometer icon that displays the "goodness of fit" of the line to the points, 4) marks on the thermometer that show the best fit achieved so far and the best theoretical fit, and 5) an equation for the current straight line.

As students adjust the regression line, the sum of squares "thermometer" changes to reflect their actions. They can temporarily "eliminate" points from the scatterplot and watch as the other representations are automatically updated. They can query a point on the graph and receive information about it from the database. Thus, using Shifty Lines, students can explore multiple sources of information about multivariate data.

The project involved the following BBN scientists and support staff. Andee Rubin, Ann Rosebery, Bertram Bruce, John Swets, Wallace Feurzeig, Paul Biondo, Willian DuMouchel, Carl Feehrer, Paul Horwitz, Meredith Lesly, Tricia Neth, Ray Nickerson, John Richards, William Salter, Sue Stelmack, and Sterling Wight. The software was published as the Statistics Workshop by Sunburst Communications Inc. in 1991.

Topology software

From 1995 through 1997, Feurzeig and BBN mathematicians Gabriel Katz and Philip Lewis, in collaboration with consultant Jeffrey Weeks, a topologist and computer scientist, conducted curriculum and software research in the project "Teaching Mathematical Thinking Through Computer-Aided Space Explorations." The object was to introduce some of the most fundamental and central ideas of geometry and topology to a broad population of high-school students through a series of interactive graphical explorations, experiments, and games involving the study of two- and three-dimensional spatial objects. The initial activities included exploratory investigations of the mathematics of surfaces. To help students experience the topology of the torus and Klein bottle surfaces, Weeks developed six software games to be played on these unfamiliar surfaces (Weeks, 1996). Though the games are familiar, playing them when the moves have quite different results from those made in the usual flat world is very challenging. Figure 13.16 shows the screen display for the entry points to the six games.

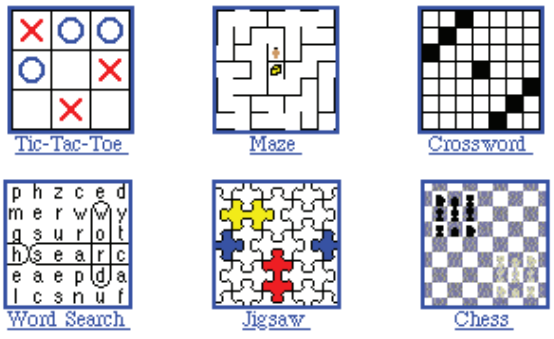


Figure 13.16 Entry points to games.

The illustrations in the figure show the games in the "fundamental domain," as they appear when played on a flat surface. The software enables the games to be viewed as they appear when played on a toroidal or Klein bottle surface. For example, the player can scroll the Tic-Tac-Toe board in the above torus game to see that the X's have won — the third X on the extended toroidal surface is just to the right of the top row, forming 3 Xs in a line. These games are accessible for download at `http://www.geometrygames.org`

Rising Curtain, a software applet, was developed for teaching the Euler formula, which relates the number of vertices (V), edges (E), and faces (F) in planar graphs. In this environment, a web of intersecting straight lines in the plane (the computer screen) is partially covered by a curtain under user control. As a student raises the curtain, local changes (newly appearing vertices, edges, or faces) are revealed. The student's task is to count the number of each of these features as the curtain rises to show the entire graph, and then to determine how these are related, i.e., to discover the Euler invariant for planar graphs ($V - E + F = 1$). Students compute the Euler numbers of a few model surfaces: the torus, Mobius band, and Klein bottle. Rising Curtain is an effective tool for introducing the fundamental mathematical concept of an invariant.

Students were then guided through a gentle, semi-rigorous development of the powerful classification theorem of surfaces, which establishes that all possible surfaces are combinations of a few basic ones, such as the torus and Klein bottle. The final project activities involved computer-simulated journeys in the world of two-dimensional spaces, using *2D-Explorer*, an interactive software tool. This software enables students to explore the surface of an unknown mathematical planet — a closed 2D-surface — with the goal of determining its global structure from local observations. Players piloting a low-altitude "flying machine" undertake voyages to uncharted closed-surface worlds. Given a mystery "planet," they are challenged to answer the question "What is the shape of this universe?" (Weeks, 1997).

Their task, as they travel over the unknown planet is to determine the intrinsic global topology of its surface by making local measurements and observations along the way. The program employs graphically rich textures and 3D animation. However, although it presents a 3-dimensional world, the underlying topological connections are only 2-dimensional. An understanding of the characteristic mathematical structure of different surfaces enables the user to establish the topology of the territory she has explored. This permits her to compute the Euler number of the known part of S. By application of the classification theorem, she knows the topology of the part of S that she has explored thus far, and the possible topologies of S that are not yet ruled out. Then, if one day, pushing the final frontier, she fails to discover new territories, she has visited everywhere and her mission is over: she knows the shape of that universe!

From 1997 through 1999, Feurzeig and Katz, in collaboration with mathematics faculty from four universities, developed versions of a new undergraduate course under the NSF-supported project "Looking into Mathematics: A Visual Invitation to Mathematical Thinking." The universities (Brandeis, Clark, Harvard, and the University of Massachusetts, Boston) were pilot sites for the course. The course included units on visual representations of mathematical objects and universes, mathematical maps, curves and surfaces, and topological explorations of "the shape of space." Visual software treating all these topics was developed to support the teaching. The student populations included pre-service elementary school teachers, in-service high-school mathematics teachers, and non-mathematics majors.. Though the four pilot versions of the course differed somewhat in emphasis, as appropriate for their different populations, they had substantial commonality in content and pedagogic approach.

13.3 Learning and teaching science

Much of our understanding of the workings of the physical world stems from our ability to construct models. BBN work has pioneered the development of computational models that enable new approaches to science inquiry. Several innovative software environments that employ interactive visual modeling facilities for supporting student work in biology and physics are described next.

Thinkertools

This 1984–1987 NSF research project, conducted by physicist Paul Horwitz and cognitive scientist Barbara White, explored an innovative approach to using microcomputers for teaching Newtonian physics. The learning activities were centered on problem solving and experimentation within a series of computer microworlds (domain-specific interactive simulations) called *ThinkerTools* (White and Horwitz, 1987; Horwitz and White, 1988). Starting from an analysis of students' misconceptions and preconceptions, the activities were designed to confront students' misconceptions and build on their useful prior knowledge.

ThinkerTools activities were set in the context of microworlds that embodied Newton's laws of motion and provided students with dynamic representations of the relevant physical phenomena. The objective was to help students acquire an increasingly sophisticated causal model for reasoning about how forces affect the motion of objects. To facilitate the evolution of such a model, the microworlds incorporated a variety of linked alternative representations for force and motion, and a set of game-like activities designed to focus the students' inductive learning processes.

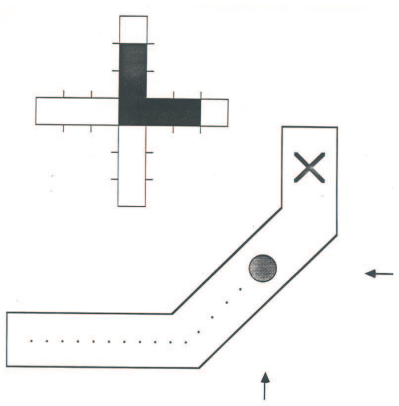


Figure 13.17 *ThinkerTools* game.

Figure 13.17 displays a typical *ThinkerTools* game. The user tries to control the motion of an object so that it navigates a track and stops on the target X. The shaded circle in the middle of the angled path is the object, which is referred to as the "dot." Fixed size impulses, in the left-right or up-down directions can be applied to the dot via a joystick. The dot leaves "wakes" in the form of little dots laid down at regular time intervals. The large cross at the top is the "datacross." This is a pair of crossed "thermometers" that register the horizontal and vertical velocity components of the dot, as indicated by the amount of "mercury" in them. Here the datacross is depicting a velocity inclined at +45 degrees relative to the horizontal. Sixth-grade students learned to use the datacross to determine the dot's speed and direction of motion.

As part of the pedagogical approach, students formalized what they learned into a set of laws which they examined critically, using criteria such as correctness, generality, and parsimony. They then went on to apply these laws to a variety of real world problems. Their investigations of the physics subject matter served to facilitate students' learning about the acquisition of scientific knowledge in general — the nature, evolution, and application of scientific laws. Sixth-grade students using a sequence of *ThinkerTools* problems did better on classical force and motion problems than high-school students using traditional methods.

Explorer Science models

The *Explorer Science* series combined the use of analytical capabilities with scientific models to create simulations for learning physics and biology. The software was developed by BBN scientists John Richards and Bill Barowy with Logal Educational Software, Israel (Barowy, Richards, and Levin, 1992). Animated measuring and manipulation tools complemented the dynamic simulations. Analytic capabilities included graphs, charts, and an internal spreadsheet with automatic or manual data collection. The *Physics Explorer* and *Biology Explorer* software was published by Wings for learning.

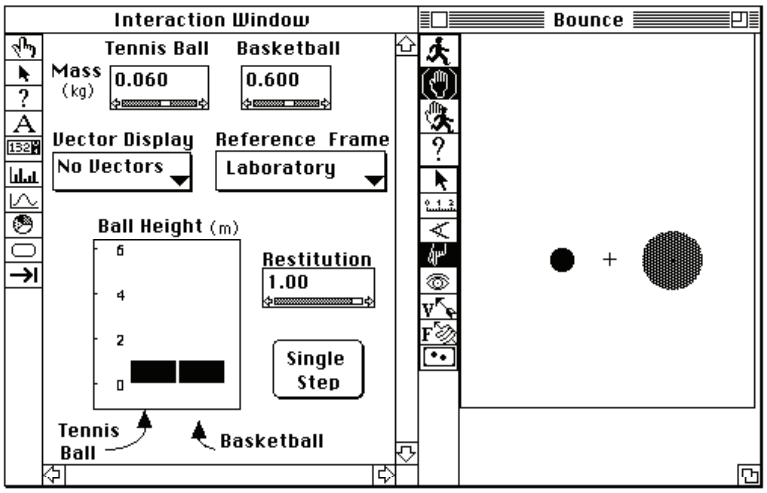


Figure 13.18 Tennis ball and basketball interaction.

The example in Figure 13.18, taken from physics classroom work, illustrates how computer modeling was integrated with laboratory experimentation to foster a coherent approach to science inquiry. The use of the model facilitates analysis and conceptual understanding of the physical phenomena. By dealing explicitly with differences between computer models and the phenomena they simulate it was possible to engage students in fruitful discussions about the strengths and limitations of the models. The students developed a sense of how scientists use models by trying to simulate phenomena themselves.

The example is an exploration of how a tennis ball and a basketball interact when the two are dropped at the same time with the tennis ball directly above and close to the basketball. The class observed both the real phenomena and the simulation with the *Explorer* Two-Body model. In successive stages of the inquiry, the focus of attention alternated between the actual phenomena and the simulation. In the first experiment, the basketball and the tennis ball were held side by side, at about chest height. The instructor asked the students to predict what the height of the first bounce of each ball would be when the two are dropped together. They were asked to explain the reasons for their predictions in order to make their intuitions explicit. After students responded, the experiment was performed. The tennis ball and the basketball each bounced to a height about 3/4 of that from which they had been dropped, which occasioned little surprise. This initial experiment established a baseline observation for checking the credibility of the computer model when it was used to replicate the experiment. The figure shows the lab that was designed to simulate the experiment. The Two-Body model is a mathematical representation of time-dependent interactions between two

circular objects and four stationary walls. The animation generated from the model appears in the model window to the right. A work-window, the Interaction Window, is shown to the left.

Using the model, students investigated the effect of changing the coefficient of restitution. The two real balls were weighed and the masses of the objects in the simulation were adjusted to match. This stage of the investigation focused on the relation of the model to the phenomena. The class discussed how they would determine when they had found a satisfactory simulation.

In the next stage of the activity, the real basketball was held at chest height with the tennis ball about 5 centimeters directly above it. The students were challenged again to predict what would happen when the balls were dropped. Would the basketball bounce as high as it did before? What about the tennis ball? They discussed their ideas and committed their predictions to paper. After several minutes of discussion, when the members of the individual teams had reached a general consensus, the balls were dropped together. Most students predicted that the tennis ball would bounce no higher than the instructor's head. When it is dropped above the basketball from shoulder height, the tennis ball often bounces up to 15 feet in height, and dramatically strikes the ceiling. Students were surprised to find that it bounced much higher then they had expected. They wanted to know why.

The analytic solution to the problem involves solving simultaneous equations, which was beyond the abilities of the students in the class. The computer model, on the other hand, provides the analytic tools to help students acquire a semi-quantitative understanding of the processes that give rise to the phenomena. The experiment was recreated with the software. Two semi-quantitative explanations were developed to account for the phenomena. Both were investigated using the software tools. Both gave reasonable estimates for the maximum height of the tennis ball bounce, though they used different physical principles and *Explorer* tools.

The first explanation was based on energy conservation: whatever energy the tennis ball gains in the collision must be lost by the basketball. This results in a relation between the change in the bounce height of the basketball (ΔH) and the tennis ball (Δh). The relation is $\Delta h/\Delta H = M/m$, the ratio of the mass of the basketball to that of the tennis ball. The change in height for each ball is determined by measuring the distance from the height it was dropped to the maximum height it attains after the interaction. The second explanation uses the principle of momentum conservation and transformations between different frames of reference. This solution was used to supplement the first, to illustrate the possibility of multiple solutions to a problem. By pressing the Single Step button several times, the instructor broke down the motion to enable a frame-by-frame analysis. The sequence of frames is shown in the composite strobe-like diagram of Figure 13.19.

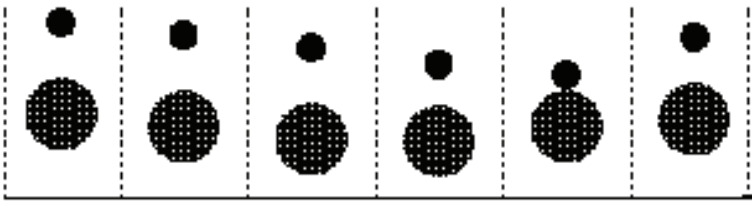


Figure 13.19 Sequence of frames.

Frame-by-frame observations showed that the basketball bounces off the floor first and then, while moving upward, it collides with the tennis ball. Pop-up menus in

Explorer allowed the user to show vector representations for the objects' velocity and acceleration and their components. The students observed that, at the moment of collision, the basketball is moving upward with the same speed as the downward moving tennis ball. Since the basketball is more massive than the tennis ball, it is virtually unaffected by the collision. Therefore, in the laboratory frame, it continued to move upward with approximately the same speed.

Next, the collision between the two balls was viewed from the reference frame that moves upward with the pre-collision speed of the basketball. Just before the collision, the basketball is stationary and the tennis ball approaches with a velocity equal to the difference of the individual velocities. Thus, in this frame, the tennis ball is moving with twice the speed it has in the laboratory frame. The tennis ball rebounds from the basketball with the same speed that it approached the basketball. By transforming back to the laboratory frame, the students estimated that the speed of the tennis ball just after the collision was equal to its speed with respect to the basketball plus the speed of the basketball with respect to the lab. Its speed was greater than it was just before the collision by a factor of three. Students gained an understanding of the surprising result — the "kick" the basketball gave to the tennis ball — and more importantly, familiarity with the use of a powerful science inquiry tool.

The *Explorer* science series has been used in hundreds of school systems throughout the United States. *Physics Explorer* has won numerous awards, including a prestigious Methods and Media Award in 1991.

RelLab

RelLab was developed by Paul Horwitz, Kerry Shetline, and consultant Edwin Taylor under the NSF project Modern Physics from an Elementary Viewpoint (Horwitz, Taylor, and Hickman, 1994; Horwitz, Taylor, and Barowy, 1996). It presents students a computer-based "relativity laboratory" with which they can perform a wide variety of gedanken experiments in the form of animated scenarios involving objects that move about the screen. *RelLab* enables students to create representations of physical objects in the form of computer icons and then assign them any speed up to (but not including) the speed of light. If they wish, they can instruct their objects to change velocity or emit another object at particular instants during the running of the scenario. At any time, as they are building their scenario, the students can run it and observe its behavior in the reference frame of any object. A representative *RelLab* screen is shown in Figure 13.20. The objects in the scenario are a football player and a rocket.

The football player is running at four meters per second. If the animation were run, the icon would move from left to right across the screen, taking approximately 12 seconds to traverse it. The rocket is moving up the screen at two-thirds the speed of light. If the animation were run at normal speed, it would disappear instantly off the top of the screen. Both the football player and the rocket have been given clocks that measure the time in their respective reference frames. The football player's clock matches that of the current frame, but the rocket's shows a different time. This relativistic effect is a fundamental consequence of the constancy of the speed of light, and the reason for it is one of the hardest things for students to learn.

Since relativity deals with time and space, a major consideration in designing *RelLab* was to build into it comprehensive but easily understood representations of these quantities, as well as powerful ways of manipulating and measuring them. All the concepts we wanted to teach could be handled as easily in two dimensions as in three. Thus, every scenario in *RelLab* is viewed as it would appear either from a helicopter looking down on the scene, or horizontally out the window of a train or car moving at