Feurzeig, W. (2011). Educational technology at BBN. In D. Walden & R. Nickerson (Eds.),
A culture of innovation. East Sandwich, MA: Waterside Publishing.

Chapter 13

Educational Technology at BBN

Wallace Feurzeig

Since the early 1960s, BBN mathematicians, scientists, and engineers have explored ways to use computer and communications technologies to improve teaching and learning. BBN staff members have conducted basic research in human cognition and learning, developed innovative software tools to extend and enrich the traditional curriculum, and provided professional development to help educators make effective use of the new technologies. They have helped schools employ networking facilities to connect teachers and students with each other and with national and global resources.[1]

13.1 From drill-and-practice to "intelligent" CAI

No one, in the early sixties, saw the enormous potential of computers for education more clearly than J. C. R. Licklider. At that time, the prevailing model for the use of technology in education was drill-and-practice. The computer, like earlier electromechanical teaching devices, directed the interaction; it posed questions (typically multiple choice) and assessed the correctness of the student's answers. The student's role was purely responsive.

Paired-associate drill

An early example of this kind of computer-aided instruction (CAI) was the *paired-associate drill tutor* developed by Licklider in 1960 (Coulson, 1962). the program was used to provide practice in learning German language vocabulary. However, it could be used to provide practice in learning any kind of paired-associate material, that is, material that is organized in pairs, the first item of which is treated as a question, the second as an answer. The program worked roughly as follows. On each trial, the program would type the first item of a pair (i.e., the question) and wait for the student to type an answer. If the student typed the correct answer (i.e., the second item of the pair), the program would indicate that the response was correct, and move on to the next question. If the student gave an incorrect answer, the student was allowed to try again. If the answer was still incorrect, the program gave the correct answer, and proceeded to the next question. In commenting on the program's tendency to hold the attention of its users, Licklider made the following observation. "It seems possible, by exploiting the computer's constant capability for quick response and reinforcement, to develop techniques of instruction and intellectual exploration that will 'trap' the attention of students, and divert their energies from less constructive pastimes, to education."

[1] Editor's note: Color is important to many of the illustrations in this chapter. These may be seen at `www.walden-family.com/bbn/feurzeig.pdf`

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.

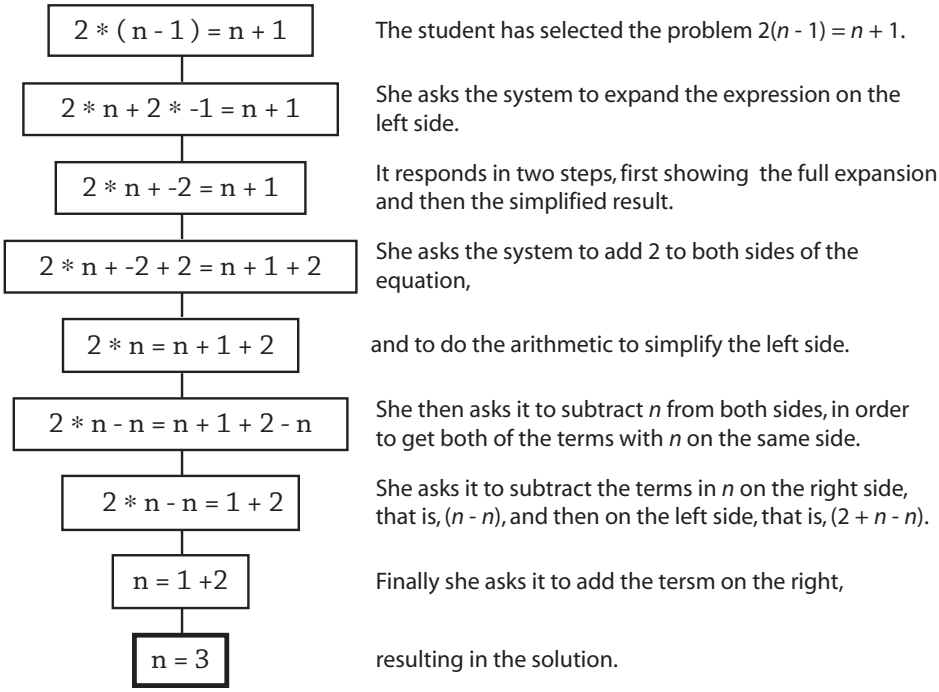


Figure 13.1 Transcript illustrating use of the *Algebra Workbench*.

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.

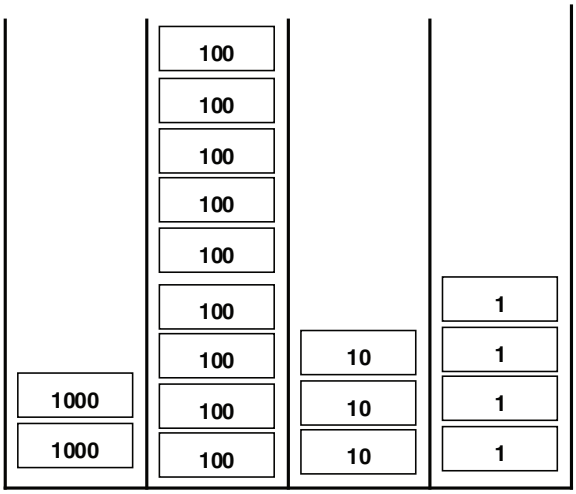


Figure 13.2 The *Summit* bin model.

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.

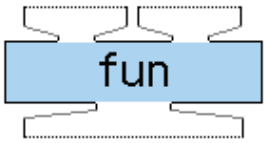
The second program in *Summit* demonstrated the standard (right-to-left) algorithms for addition and subtraction. It displayed a problem on the screen and talked its way through to the solution, explaining its steps along the way. The demonstrations were sequenced, starting with single-digit problems 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.

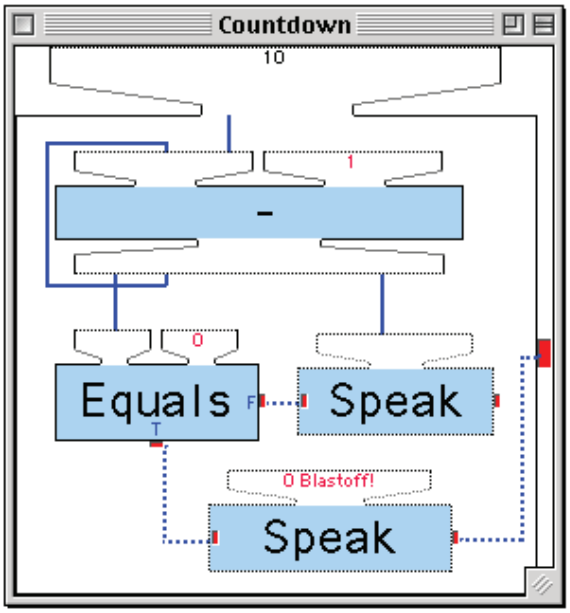


Figure 13.3 ***Function Machines*** **Countdown program.**

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.

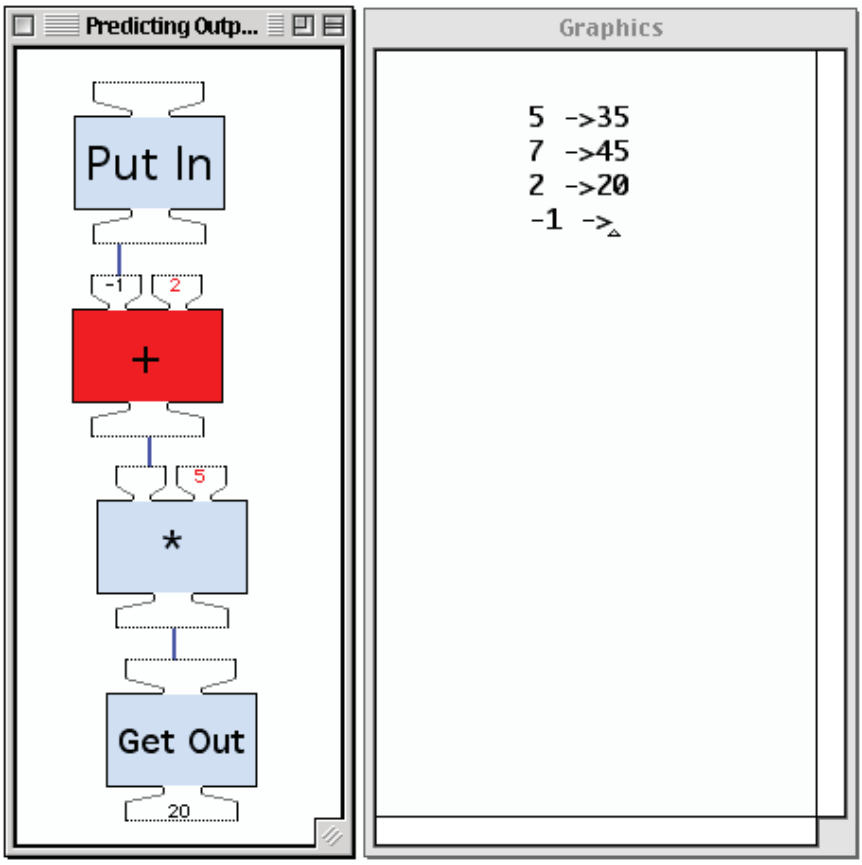


Figure 13.4 Predicting Results *Function Machine.*

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.)

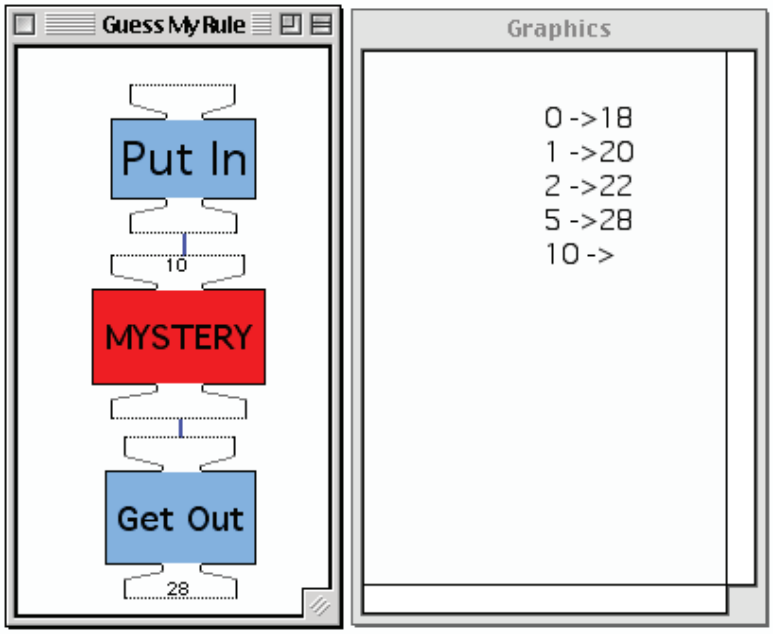


Figure 13.5 Guess My Rule *Function Machine.*

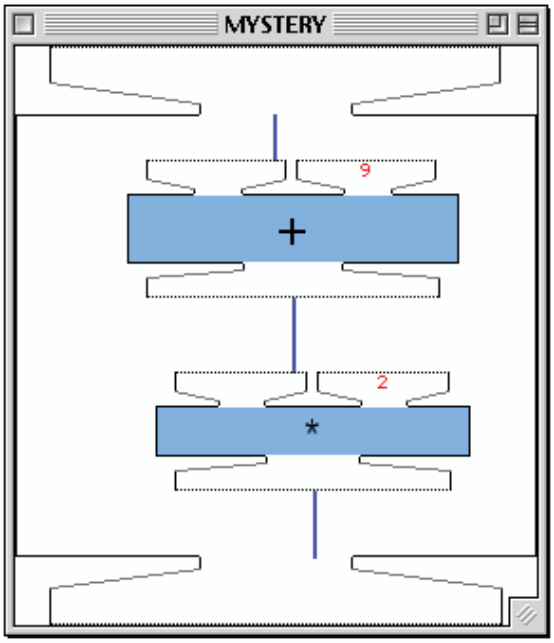


Figure 13.6 Inner contents of the *Mystery* Machine.

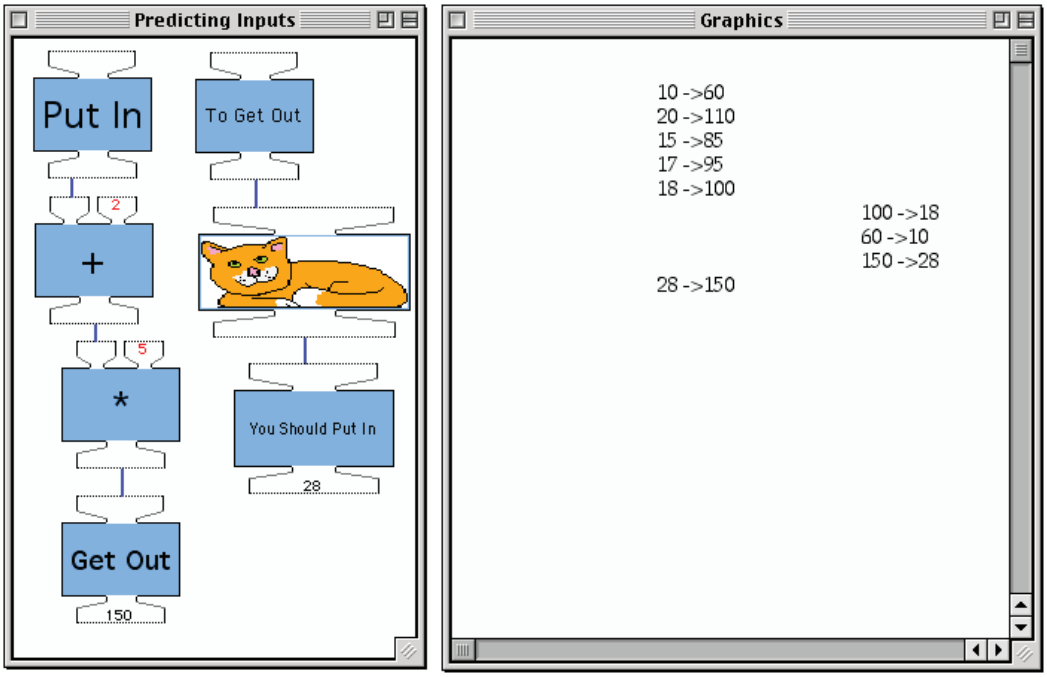


Figure 13.7 Predicting Inputs machine.

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.

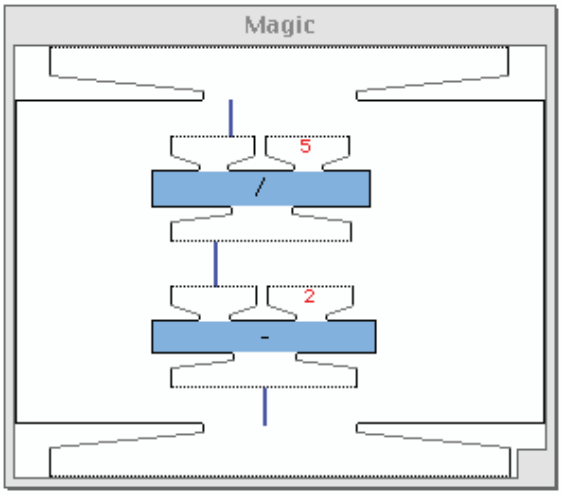


Figure 13.8 Magic machine.

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.

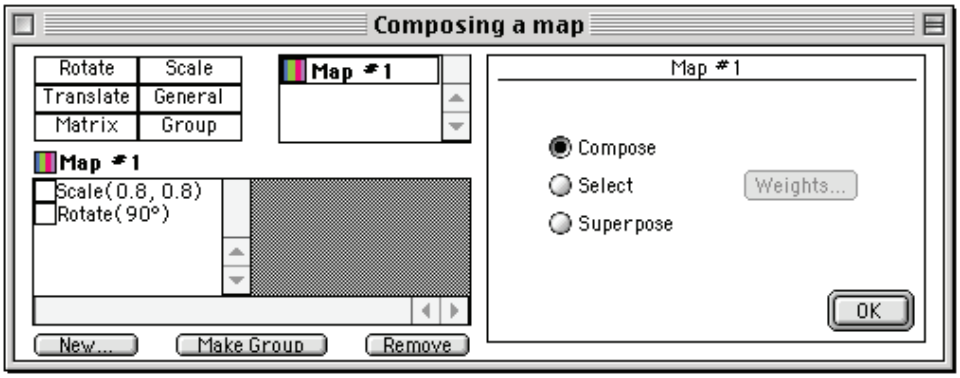


Figure 13.9 Composing a map.

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.

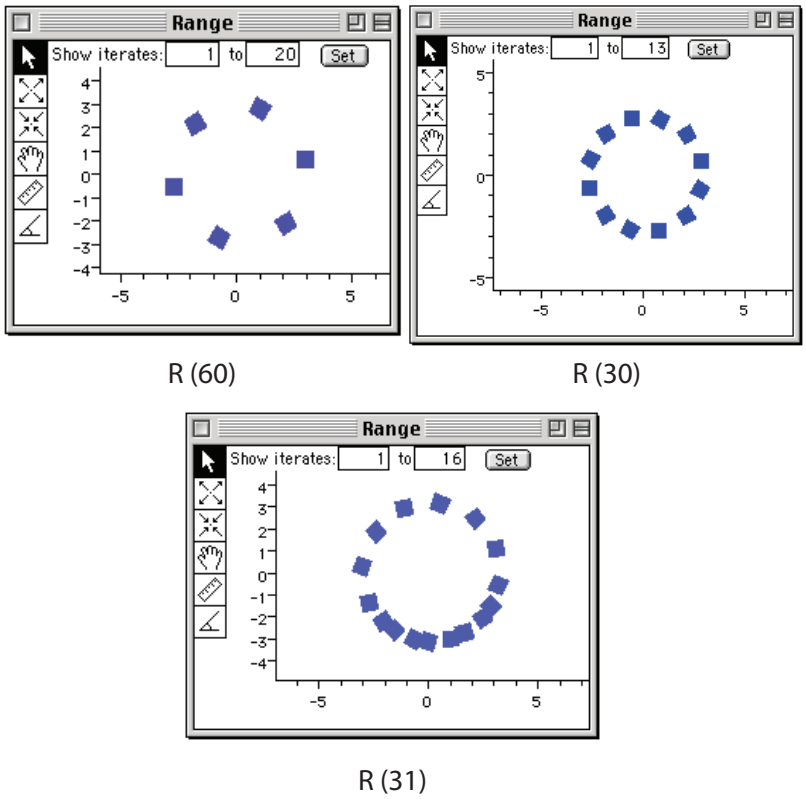


Figure 13.10 A *MultiMap* session.

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."

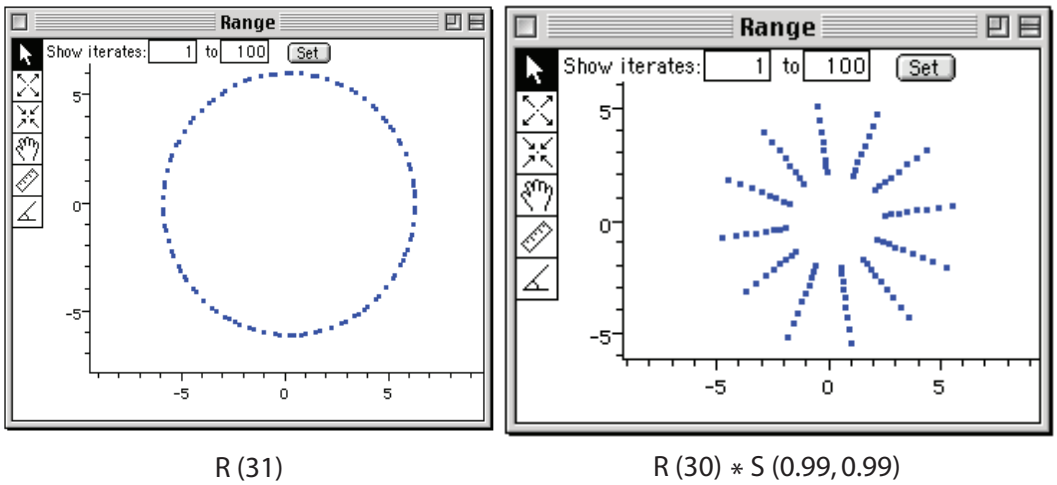
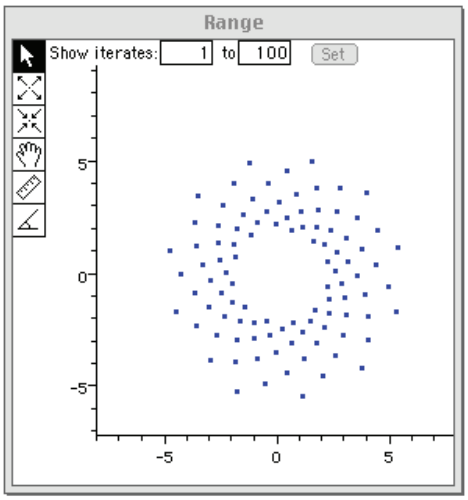


Figure 13.11 Continuing the session.

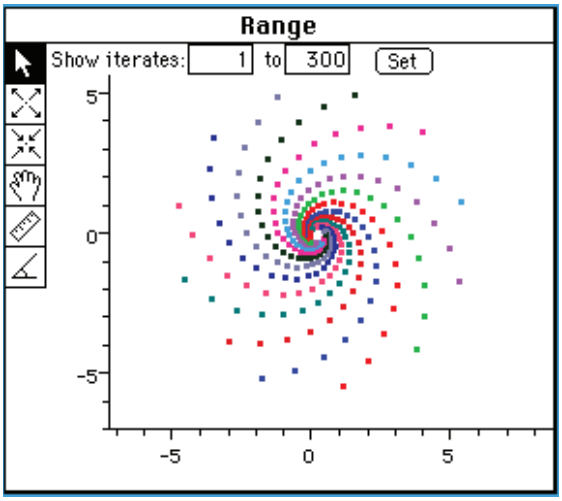
The instructor agreed and asked what would happen if the rotation were 31 degrees instead of 30. Fred replied "It would be almost the same but the points would not be on straight lines. They would curve in a little each time." He tried this. The result is shown in the top left of Figure 13.12. Kate said "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."

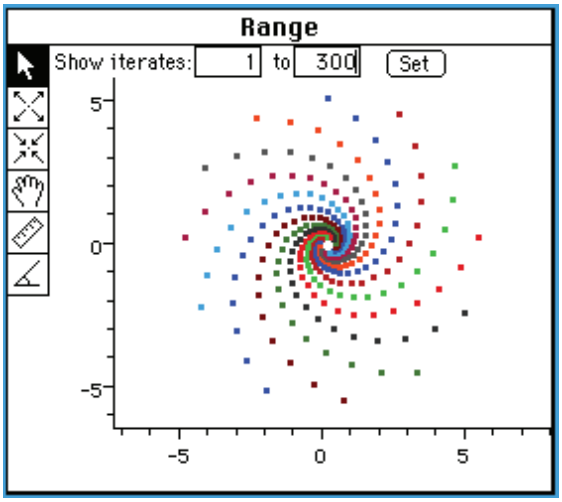
Instructor: "It might be that we don't have enough detail — let's get a more detailed picture by changing the scale from .99 to .999, and increasing the number of iterations from 300 to 600. See if that makes a difference." The result, after 600 iterations, is shown at the bottom left of the figure. 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.



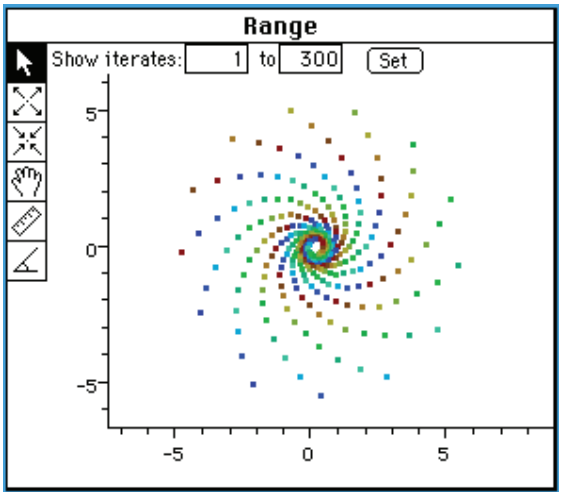
R (31) * S (.99, .99)



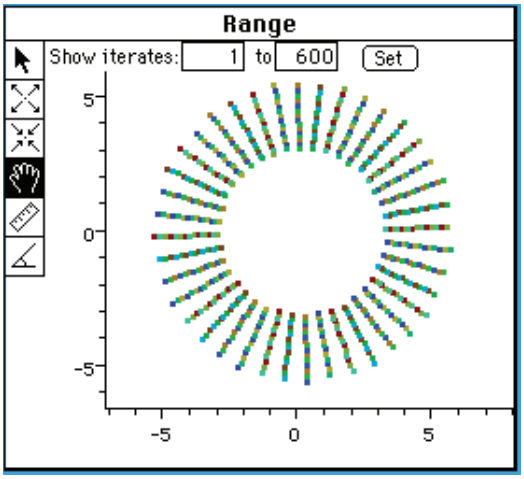
R (31) * S (.99, .99) 12-color ramp



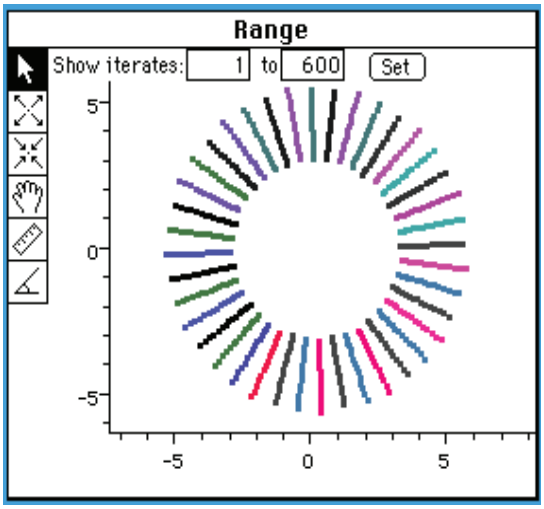
R (29) * S (.99, .99) 12-color ramp



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.

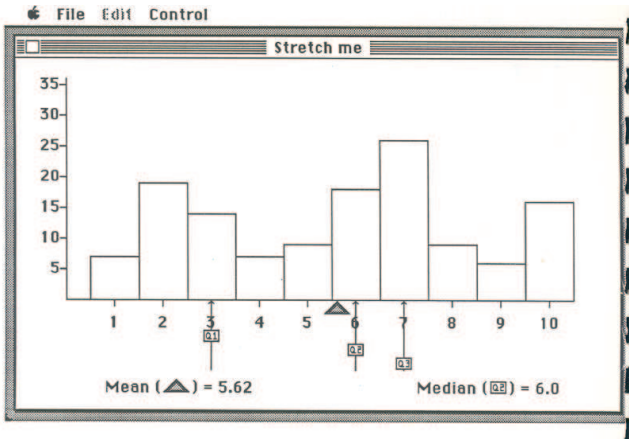


Figure 13.13 Stretchy Histograms.

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.

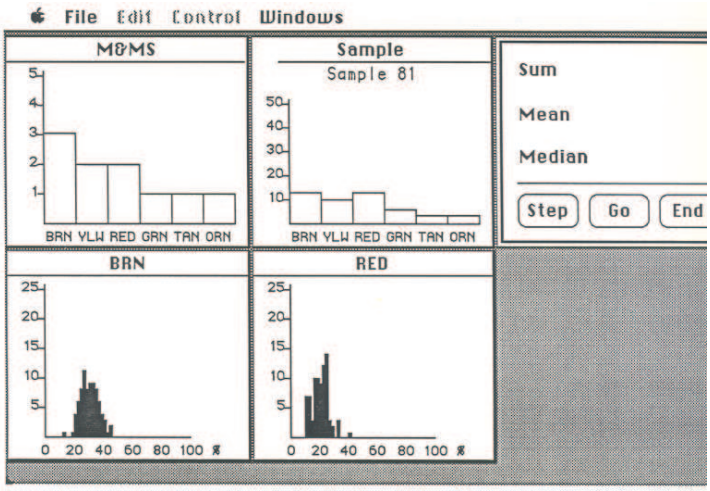


Figure 13.14 Sampler.

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.

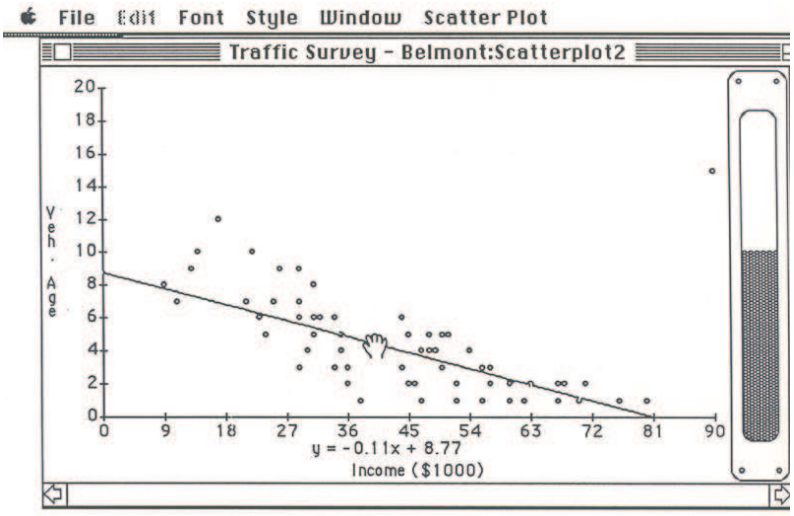


Figure 13.15 Shifty Lines.

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.

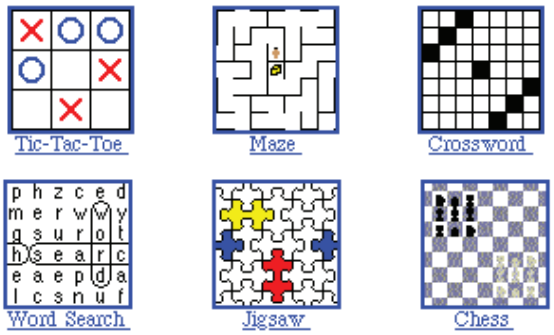


Figure 13.16 Entry points to games.

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.

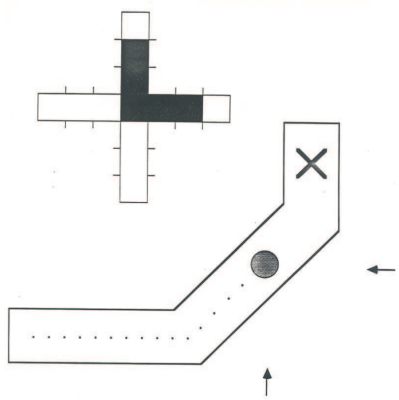


Figure 13.17 *ThinkerTools* game.

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.

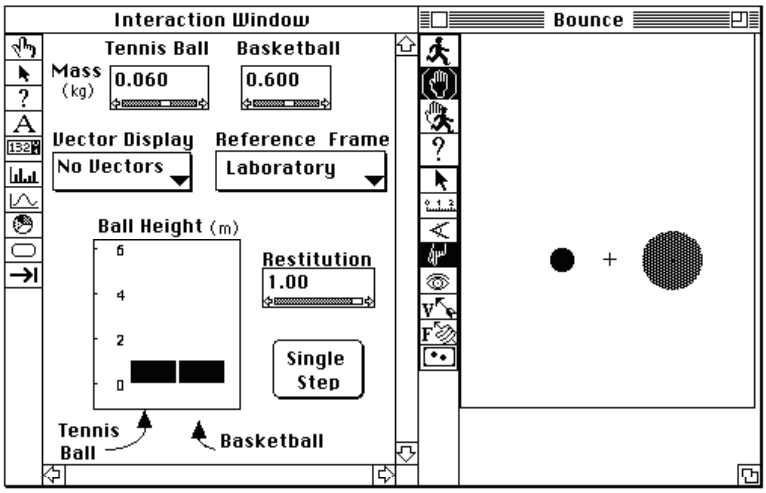


Figure 13.18 Tennis ball and basketball interaction.

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.

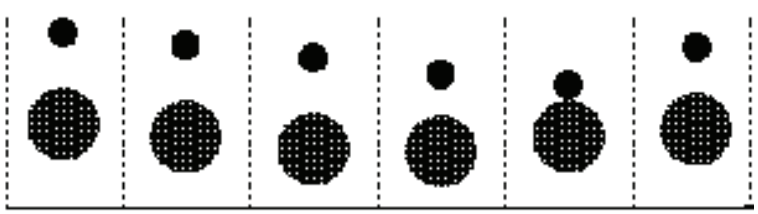


Figure 13.19 Sequence of frames.

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

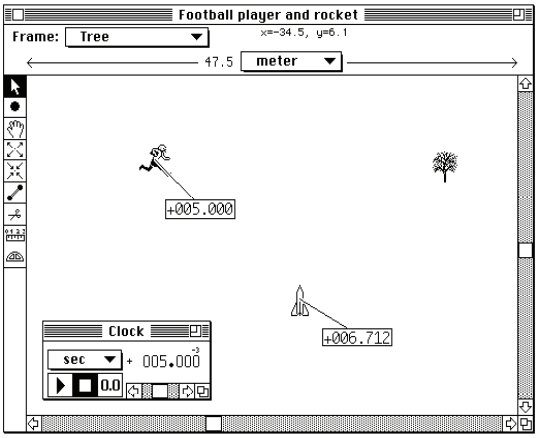


Figure 13.20 Representative *RelLab* screen.

the same speed as the reference object. *RelLab* scenarios often involve astronomical distances. Since there are no obvious indications of the space scale (the icons, which represent point objects, do not change size when the screen is zoomed), this can be confusing to students. Thus, *RelLab* provides a continuously available indication of the space scale, in the form of arrows that span the top of the screen and indicate (in units selectable by the student) how far it is across.

Time is represented directly in *RelLab* through animation. As a result of another explicit design decision, *RelLab* does not allow the user to alter the rate at which time passes: there are no time-lapse or slow-motion displays. When animated, every scenario runs in "real time" — one second on the computer being exactly equivalent to one second in the scenario itself. Early in the design of *RelLab* it was decided not to allow students to build simulations in which the speed of light is altered. This was done not only to avoid possible confusion, but also because such a fundamental alteration of the laws of physics would lead to internal inconsistencies. Instead, *RelLab* demonstrates relativistic effects, which are ordinarily too small to be observed, either by allowing objects to move at very high speeds or by enabling students to make extremely precise measurements of low-speed scenarios. This, in turn, required us to provide exceedingly fine measuring tools, and indeed *RelLab* allows students to measure distances, such as the separation between objects, and time intervals to a precision limited only by that of the computer.

In addition to representing and measuring time and space, *RelLab* provides students with powerful tools for manipulating these quantities. The *RelLab* screen may be scrolled effectively any amount in any direction, and its width may be set to represent any distance from a few millimeters to many light-years. Time can also be set to any value. The clock that displays time in the current reference frame also gives students control over that time. Any alteration in the time displayed by this clock generates an immediate update of the positions of all objects. By observing such changes on the screen, for example, students can determine the distance traveled during a nanosecond (a billionth of a second) by a rocket traveling at nearly the speed of light.

Representations can be as important for what they conceal as for what they display. For instance, *RelLab* does not represent extended objects; each icon is simply a graphic representation of a single point. The reason for this constraint stems directly from the nature of relativity itself. A rigid object (one that retains its spatial configuration when its velocity changes) is an impossibility in relativity, in part because the speed of

sound in such an object would exceed that of light. But students are accustomed to thinking of any extended (solid) object as infinitely rigid, in the sense that an impulsive force applied to one side is transmitted instantly across it. This sort of "pre-relativistic thinking" is likely to lead to confusion, so *RelLab* does not admit the construction of objects that have finite spatial extent. It does, however, allow one to associate point objects that have a semantic association (for instance, the front and back ends of a lance carried by a relativistically galloping knight). Objects of this kind may be connected by drawing straight lines between them. The lines are drawn in gray, however, to convey the fact that they do not correspond to anything physical, and they stretch or shrink if the separation of their endpoints changes. They are analogous to the fictitious lines often drawn between stars to represent the constellations — they denote logical groupings of fundamentally independent objects and imply nothing about the presence of forces between them.

Just as representations can constrain students' emerging conceptions, design choices may entail a conscious decision not to allow students to perform a particular manipulation. One example in *RelLab* is the constraint that objects may not be assigned speeds exceeding the speed of light, conventionally designated by the lowercase letter "c." Attempts to do so bring up an error box. This may seem an arbitrary limitation to students, but their annoyance may lead them to discover that it is not as arbitrary as it may appear. For example, an enterprising student who attempts to bypass it by firing a high-speed projectile from the nose of a rocket moving at close to the speed of light soon discovers that this does not work. No matter how fast the projectile moves (provided it is less than c) in the reference frame of the rocket, its speed never exceeds c in the frame in which the rocket is moving. Relativistic velocities do not add as ordinary ones do.

Another, more subtle constraint arises from the inability of *RelLab* to express cause-and-effect relationships in terms of action at a distance. Every change in a *RelLab* scenario takes place at an event — a particular point in space and time — and has consequences that are local in the sense that they affect only objects in their immediate vicinity. The scripting language that underlies the definition of events in *RelLab* does not allow "if — then" constructions that imply instantaneous action at a distance. For example, a command such as "When the light reaches Andromeda, launch the rocket from Earth" is impossible to express in *RelLab*. Such commands are improperly posed because they imply simultaneity of spatially-separated events and thus can be carried out only in a special subset of reference frames. This frame dependence of simultaneity is a very subtle and completely counter-intuitive concept — perhaps the hardest one for students of relativity to accept and understand. *RelLab* does not explicitly teach this, but because it is built into the very syntax of the event language, the program constrains students to think in purely local terms and prevents them from constructing improperly-posed cause-and-effect relationships.

RelLab won two awards in the 1992 EDUCOM national educational software competition: one for Best Natural Science Software (Physics), the other for Best Design. Using *RelLab* in the classroom, teachers have found that high school students can achieve a qualitative understanding of relativity comparable to that of graduate students.

GenScope

GenScope was developed by Paul Horwitz, Eric Neumann, and Joyce Schwartz from 1993-1996 as the key tool in the NSF project Multi-Level Science (Horwitz, Neumann, and Schwartz, 1996). The goal of the project was to give middle-school and high-school students an understanding of the reasoning processes and mental models

that characterize geneticists' knowledge, together with an appreciation for the social and ethical implications of recent advances in the field. *GenScope* is an open-ended computer-based exploratory environment for investigating the phenomena of genetics at several levels (DNA, gene, cell, individual organism, family, and population) in a coherent and unified fashion. Each level offers visual representations of the information available, as well as tools for manipulating that information. The information flows between the levels, linking them in such a way that the effects of manipulations made at any one of them may instantly be observed at each of the others. The levels thus combine to form a seamless program. The software presents the complex, linked, multi-level processes of genetics visually and dynamically to students, making explicit the causal connections and interactions between processes at the various levels. The underlying genetic model is itself linked, via a software structure called a "hypermodel," to a variety of data objects, including video sequences of cell division, visualizations of protein and DNA structure, and organism phenotypes.

To illustrate genetic phenomena, *GenScope* starts with dragons — simple, fictitious creatures that are useful for teaching purposes and do not prematurely raise such sensitive issues as the pros and cons of genetic engineering or the uses of genetic screening tests. Two *GenScope* dragons are shown in Figure 13.21. Students are introduced to

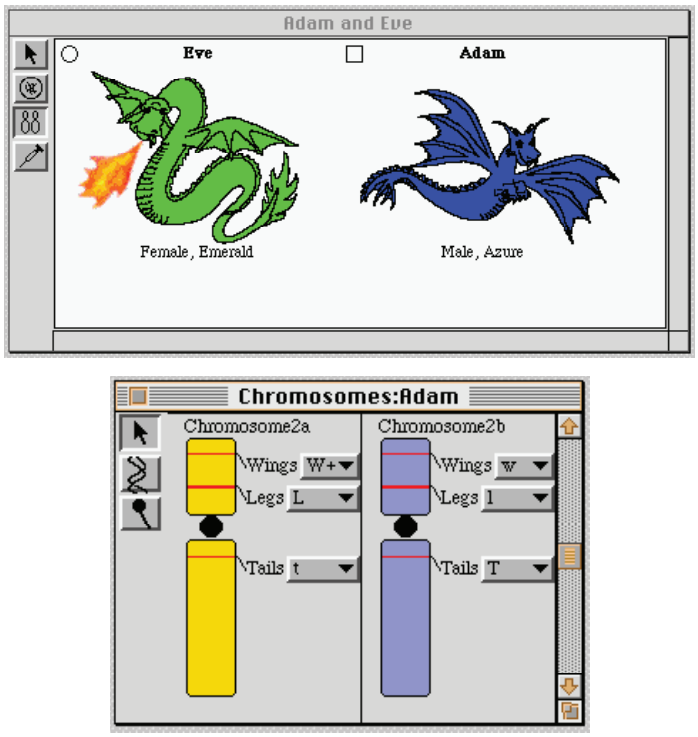


Figure 13.21 ***GenScope*** **screens.**

GenScope at the organism level, which displays the organisms' phenotypes (physical traits), but gives no information on their genetic makeup. Using a *GenScope* tool, however, they may move to the chromosome level to observe a pair of chromosomes for an organism, as shown at the bottom of the figure.

The chromosomes, in turn, are made up largely of DNA, which is observable at *GenScope*'s molecular level and carries the genetic information of the particular organism. Genes may be manipulated at either the chromosome or the DNA level, and the results

of such manipulations, if any, are immediately observable in the affected organism.

When two organisms mate, their genes are shared by their offspring through two processes which take place at the cell level. The cells can be made to undergo either mitosis, in which process they simply reproduce themselves, or meiosis, whereby they produce a new kind of cell, called a gamete, which possesses only half the chromosomes of the parent cell. The gametes produced through meiosis can then be combined in the central panel of the window, to produce a fertilized cell containing the usual complement of chromosomes. This cell, in turn, will grow into a dragon possessing genetic material from each of its parents. Each of these processes is represented graphically by the computer, as shown in the two illustrations in Figure 13.22. The first figure shows one cell each from Eve and Adam, the two dragons. The spaghetti-looking things in the centers of the cells are chromosomes, the carriers of the genetic information within the cell.

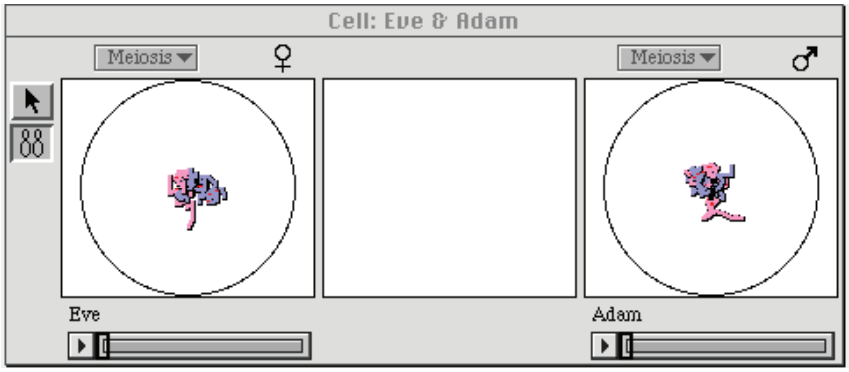


Figure 13.22 Example of mating organisms.

These cells can be made to undergo meiosis (division into four gametes, each of which contains only half the genetic material of the parent cell) or mitosis (ordinary cell division into two identical cells. When meiosis is invoked, the computer runs a randomized simulation of gamete formation. In Figure 13.23 a snapshot of the cell window, meiosis is in process. Adam's cell, on the right, has already produced the four gametes; Eve's cell, on the left, has completed the first division and is halfway through the second.

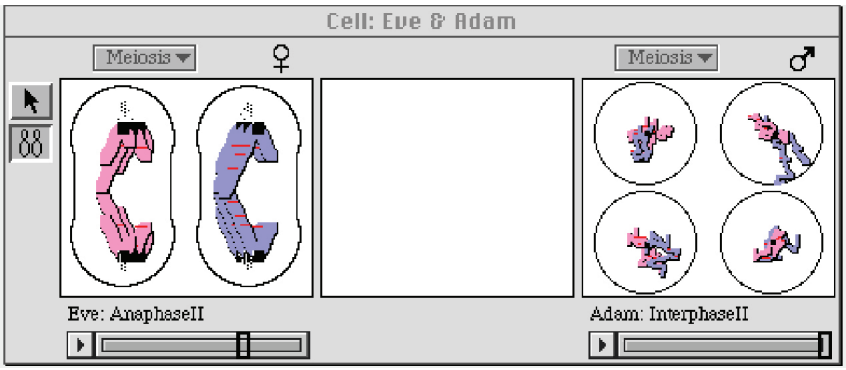


Figure 13.23 Meiosis proceeds.

At *GenScope*'s pedigree level students create "family tree" structures of related organisms in order to observe and investigate such inheritance patterns. The population

level displays groups of organisms moving about and randomly mating. Different portions of the screen can be assigned different "environments," which selectively favor one or another phenotype. The resulting "genetic drift" alters the distribution of gene types in the environments. The true nature of the genetic mechanism resides at the molecular level. *GenScope* enables students to drop down to this level to explore the DNA molecule that resides within each chromosome. For example, Figure 13.24 depicts Eve's two genes for wings, showing what the W+ and w alleles look like at the DNA level. The left window shows the dominant and wild type (normally found in the population)

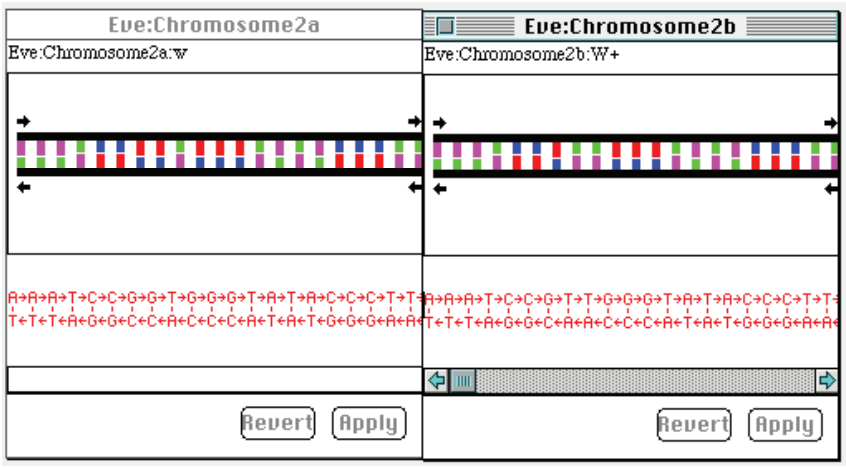


Figure 13.24 More about Eve.

W+ allele, the right window the recessive w allele. They differ by a point mutation — a single base pair substitution.

Just as the informational representation of a gene can be manipulated, via pulldown menus, so the information representation of a DNA strand can also be altered, simply be deleting or inserting the appropriate letters, typing them in as one would do with a word processor. Thus, alleles can be altered at the DNA level, and the changes will be reflected in the organism just as though the gene had been changed directly on the chromosome through the pulldown menu. Mutations created at the DNA level are treated as new alleles. They can be named and used just as the pre-defined ones can.

GenScope was designed to induce students to think at multiple levels. It does this by offering them a set of increasingly difficult challenges, and by careful choice of the set of things they can see and things they can do. On the very first day of class, for example, students are formed into pairs, and issued a simple challenge: "by directly manipulating its genes, try to make this dragon blue, with two legs and no horns." This requires them to explore and experiment with a dominant/recessive trait (horns), a co-dominant one (legs), and a sex-linked, polygenic one (color). Initially, they are given the ability to manipulate the genes directly. In a later activity, the students may be asked to turn their two-legged dragon into a four-legged one, but now their ability to alter the genes at the chromosome level has been disabled. This forces them to work (and therefore to think) at the DNA level, carefully observing the difference between two genes, and then altering one of them appropriately to accomplish the assigned task.

Cardio

Cardio was one of several visual models for science investigation developed in the 1989-1992 NSF-supported project Visual Modeling: A New Experimental Science (Feurzeig,

1994b). BBN project staff included Wallace Feurzeig, Paul Horwitz, John Richards, Ricky Carter, Barry Kort, Eric Neumann, and Donald Redick. *Cardio* is an interactive visual simulation environment for investigating the physiological behavior of the human heart while providing insight into the dynamics of oscillatory processes — particularly coupled oscillators which are fundamental to the operation of living systems. *Cardio*, which was developed by computational biologist Neumann, focuses on the heart's electrical system.

Cardio was designed to permit students to observe the deterministic heart dynamics produced by the cardiac electrical system and to study the effects of changes to specific heart component parameters. As the simulation runs it generates several displays. The major schematic display is the graphic animation of the heart model which shows in real time the rhythmic pulsation of the heart chamber accompanied by the sound of the closing of the heart valves. Another display shows the electrical schematic diagram of the pacemaker nodes and conductance paths. These interactive displays can be simultaneously viewed with EKGs and phase plots showing heart dynamics. During a simulation, the student can record and plot various time-dependent dynamic variables, including EKGs and chamber contraction. This is useful in comparing the dynamics of systems with different parameter values. The *Cardio* screen is shown in Figure 13.25.

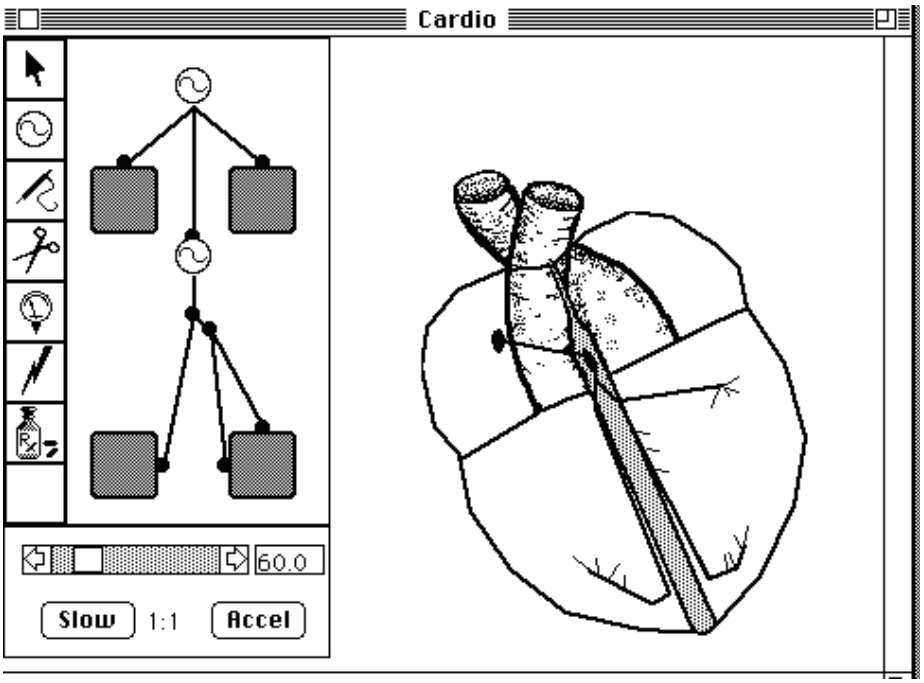


Figure 13.25 *Cardio* screen.

The system dynamics result from the run-time interactions of its individual components. These components include pacemaker nodes, conductance paths, and heart chamber muscle. Students can select from several pre-defined heart dysfunctions and dysrhythmias which alter the component parameters and investigate the resulting dynamics. Because the heart model derives its behavior from component interaction, students can change the parameters of any component or add their own components to form ectopic pacemakers and anomalous conduction paths. In fact, *Cardio*'s visual modeling environment enables students to graphically create new components by selecting them from a palette, specifying their parameters, and connecting them to

existing components. This is made possible by using new instances of pre-compiled objects and inserting them into the component list, circumventing the use of a slower and less efficient interpretive structure. Thus, students can create their own heart models and investigate their behaviors.

EKG graphs are constructed and displayed in real time from the 3-dimensional dipole field generated by the four chambers. The depolarization wave of the myocardium creates a positive deflection on the EKG trace as the wave approaches a lead, a negative deflection as the wave recedes, and no deflection if the wave moves orthogonally to the lead. The "L" leads represent the difference between each pair of Einthoven's triangle vertices (i.e., right arm, left arm and feet). Based on the interpretation of at least three different EKG leads, sophisticated users are able to reconstruct the 3-dimensional electric vector time-dependent sweep of the heart. Conduction delays between the atria and ventricles will appear on the tracings as delays between the deflections. EKGs are useful in identifying pacemaker characteristics, conduction rate changes and myocardium anomalies (e.g., ischemia and infarcts).

However, because EKGs are the result of the combined electric fields of each chamber, it is not easy to elucidate from EKG plots the complex and asynchronous patterns of chamber depolarization that continuously evolve over time. Such patterns may arise when the heart does not return to the same state-space after a single pacemaker cycle (as is the case for myocardium which is still in the refractory state caused by the previous pulse). To help visualize such complex behavior, phase plots of the contractions or electric fields of one chamber plotted against those of another chamber illustrate the dynamics by means of orbit paths. For instance, a plot of right atrium contraction vs. left ventricle contraction shows a limit cycle whose eccentricity depends on the phase difference between the two chambers. Complex dysrhythmias can be generated, observed and analyzed in this user-defined phase-space. For example, a second-degree block introduced by the user results in ventricle rhythm that does not always follow the sinus pacemaker, producing multiple orbit paths like those shown in the Figure 13.26 phase plot.

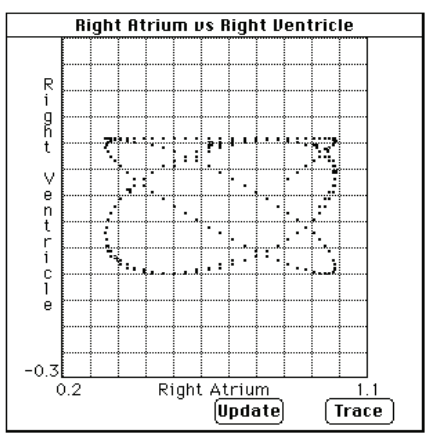


Figure 13.26 Phase plot.

Other types of dynamics can also be created and studied in *Cardio*. Mechanical, electrical, and chemical disturbances of many kinds can be introduced and their effects on heart behavior observed and analyzed. Multiple (i.e., ectopic) pacemakers can be modeled in several ways (e.g., resetting and non-resetting). Combined with the intrinsic refractory limit of the conduction system, these yield complex echo and skip beats.

By saving heart parameters as files, *Cardio* enabled students to compare, model and test various heart conditions and to determine the state-space domains of complex and chaotic rhythms.

Object-Object Transformation Language (OOTLs)

OOTLs is a visual modeling environment for describing dynamic phenomena (Neumann and Feurzeig, 1999). It was developed by Eric Neumann, Wallace Feurzeig, and consultant Peter Garik to help students acquire experience and skill in formulating problems involving dynamical processes. Events in *OOTLs* are conceptualized as interactions among the key objects in the model processes. The *OOTLs* language supports the description and simulation of phenomena for which the law of mass action holds. It applies to "well-stirred" systems composed of large numbers of dynamically interacting objects.

OOTLs has application to an extensive variety of phenomena in many areas of science including epidemiology (contagious disease spread), population ecology (competition, predation, and adaptation), economics (market dynamics), physics (gas kinetics), chemical dynamics (reaction-diffusion equations) and traffic flow. It provides students with a parser to construct equations describing interactions between objects. The objects, which are represented as graphic icons, may represent chemical species, gas molecules, or humans. Objects interact with each other at specified rates. The equations describe the transformations resulting from the object interactions. Objects may be created or consumed (e.g., for chemical reactions there are sources and sinks for reactants; for a biological problem, birth and death of species; for a model of an economy, imports and exports, or innovation and obsolescence). Equations are specified simply, by dragging graphic icons into windows.

OOTLs enables students to study the time behavior of the reactions before they have the mathematics necessary to understand the underlying differential equations. The number of coupled reactions and the number of participating objects are not limited. Objects are assigned arbitrary colors — red, blue and green — which mix to form other colors on the screen. Thus, as the reactions progress the color of the reaction products changes. Concentrations of all constituents, and any mathematical combinations of them, can be graphed in real time. *OOTLs* also models diffusion processes. Multiple reactors can be created and linked in linear or two-dimensional arrays. Diffusion constants can be specified, and the resulting dynamics displayed by means of animated colors. Since the diffusion constants of the different constituents need not be the same, the effects of variation in this important parameter are directly observable. *OOTLs* can function as a gateway to many different topics in various areas of science and mathematics.

The following example illustrates the use of *OOTLs*. The application describes a classic situation in epidemiology: the spread of disease in a large population concentrated in a local geographic area. A familiar example is mononucleosis (the "kissing disease") spread among students living close to each other in university dormitories. The basic model assumes that most students will eventually contract the disease through contact with a student who is infected, and that each student who becomes infected will eventually recover and acquire immunity. Thus, there are three sub-populations of students at any time. There are the *Susceptible* students, those who have not yet caught mononucleosis but who will catch it if they come in contact with an infected student; the *Infected* students, those who are currently ill; and the Recovered students, those who have been ill and are now immune.

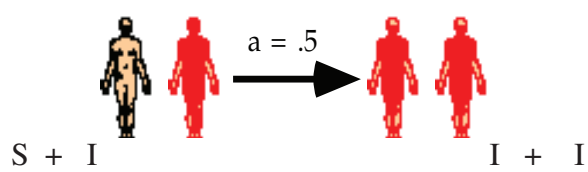
The system of ordinary differential equations (ODEs) describing this dynamic model

involves three populations of individuals. It is defined as follows (where a is the transmission rate, the fraction of the individuals in the susceptible population that becomes infected per encounter per day; and b is the recovery rate, the fraction of the individuals in the infected population that recover per day):

(1) $dS/dt = -a * S * I$ =change in Susceptible
(2) $dI/dt = a * S * I - b * I$ =change in Infected
(3) $dR/dt = b * I$ =change in Recovered

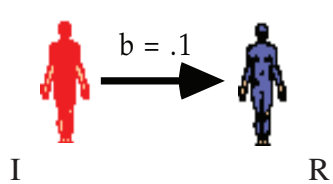
For each susceptible individual that gets ill, S is decreased by the same amount as I is increased, thus the term $a * S * I$ appears twice, once negative, once positive. The same applies to the recovery rate term, $b * I$, though it is offset by only one equation. Our experience, and that of other investigators, is that most high-school students are unable to formulate these rate equations.

This is how students might build the same spread of disease model using *OOTLs*. They begin by identifying the types of objects that are relevant. In this instance they identify two kinds of objects — individuals who are currently infected (denoted I), and those who are healthy but susceptible (denoted S). They then describe the possible interactions between such individuals that can give rise to the observed behaviors — transmitting or "catching" the disease. In this case, the students identify a single interaction: "When a susceptible individual meets an infected one, the healthy individual becomes infected also." They specify an interaction rate, a. They then define and select the icons specifying susceptible and infected individuals, and arrange them to form the following causal *OOTLs* interaction equation, describing what occurs before and after the two types of individuals come into contact.



Once this transformation equation has been input via the *OOTLs* graphical interface, students can simulate the system based on the initial conditions they choose. If they start with a small number of sick people and a large number of healthy ones, over time all the healthy individuals will "turn into" sick ones, reaching a stable final state, though the dynamics involved in attaining this are not trivial.

Students are then asked whether this is the actual outcome that describes what happens in the real world. Their considered answer is: "No, people do not stay sick forever. They get better." The issue they now address is: how do people stop being sick, and how is this to be represented? One way to extend their model is to simply allow for sick individuals to become healthy again after a period of time. This requires creating a new type of object (denoted R), for individuals that have recovered and are immune to further infection. Then, a second transformation equation is added to the model, expressing recovery: sick individuals eventually recover at some rate b.



This is known as a first-order decay, and produces exponential diminution over time. The result of simulations with this new two-equation model now yields a peak level of infection, with the number of infected dropping thereafter, followed by a new stable state in which not all the original healthy (susceptible) individuals may become sick. Students can extend the model by adding additional transformations of increasing complexity, such as the addition of a rule to allow recovered healthy players to again become susceptible to infection over time. Alternatively, recovered individuals could still be carriers without any outer symptoms, thereby infecting healthy individuals. And finally, students might incorporate population dynamics, allowing individuals to reproduce, die, and form sub-populations with different rates for growth and death.

In realizing these models, the appropriate mathematics is handled by the *OOTLs* graphics language preprocessor. Notice that, while the differential equations (DE) representation employs three equations, one for each possible health state of the individual, in *OOTLs* only two process equations are required — the DE form is redundant, and beginning students are often confused by the significance of its terms. The dynamics of the DEs are fully captured by *OOTLs*, as illustrated by the simulation output in Figure 13.27.

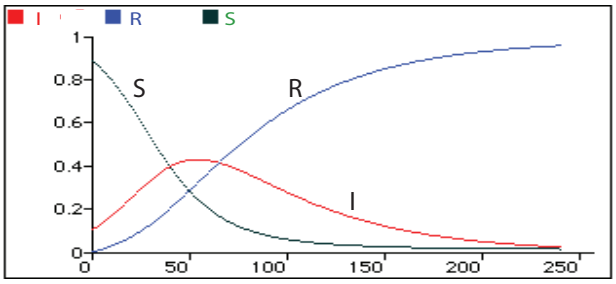


Figure 13.27 Simulation output.

The dynamics resulting from this formulation display the classic onset and course of an epidemic, with the number of infected peaking at a certain time, and then diminishing as the number of recovered increases asymptotically. Note however, that not all susceptible individuals will necessarily get ill. If the rate of spread is not as fast as the recovery, then some individuals escape infection. However, decreasing the rate of recovery (i.e., lengthening the incubation-illness period) has the effect of ensuring that more individuals will get the disease. This important concept is very easily explored in the process-specific form embodied in *OOTLs*. The *OOTLs* system provides its own DE simulation engine. However, *OOTLs* can also be used as a language front-end to drive other simulation engines, including those employing discrete and stochastic mechanisms as well as those employing continuous dynamics.

13.4 Learning and teaching language and reading

BBN research in educational technology has covered a wide range of issues related to language processing and comprehension. Applications have included teaching language and reading skills to beginning learners and to those with severe hearing impairments.

Second language learning

People who learn a second language as adults often speak it with a “foreign” accent all their lives, in spite of using it daily. One explanation for this is that in the course of

learning one's native language, one loses the ability to make certain auditory discriminations or articulatory movements that are not characteristic of that language. Thus if the second language requires such discriminations or movements, one may not only have difficulty making them, but may be unaware of the fact. To one's own ear one's pronunciation sounds correct, even though to the ear of a native speaker of the second language it does not.

The question naturally arises as to whether distinctions that are difficult to make by ear might be more susceptible to training if they could be made by eye. A computer-based system was developed at BBN in order to explore this possibility. The system was built around a DEC PDP-8L computer equipped with a bank of analog filters to pre-process incoming speech, a tape recorder with a five-second loop to maintain a continuous recording of the last five seconds of speech input, and a cathode ray tube on which the results of various types of speech analysis could be displayed. The system, which was called the Automated Pronunciation Instructor (API), was used in a series at studies aimed at developing better procedures to teach the correct pronunciation of a second language. The second languages used in these studies were English for native speakers of Spanish and Mandarin Chinese for native speakers of English.

Several displays were developed, each emphasizing some particular aspect of pronunciation that was deemed by the investigators to be particularly relevant to the training objectives. One such display showed a schematized representation of tongue position during vowel production. Tongue position here means roughly the two-dimensional position of the tongue hump within the vocal track as viewed from the side. (The vowels in Spanish and English are not quite the same and native speakers of Spanish often substitute the nearest equivalent Spanish vowel for the correct English vowel when speaking English.)

Inasmuch as vowel quality is determined, in large part, by the position of the tongue body in the mouth, it was hypothesized that a display that permitted the student to compare the actual tongue position with the desired tongue position for specified vowels would facilitate correct pronunciation. Actual tongue position was represented by dots in a large rectangle on the display. The desired position (more accurately, the region of acceptable positions) was represented by a small rectangle within the large one. The student's task was to produce a vowel in such a way that the dots fell inside the small rectangle. Tongue position was inferred from certain sum and difference calculations performed on the outputs of the individual analog filters. Figure 13.28 shows the tongue position display as it might appear for both a correctly (on the left) and an incorrectly pronounced vowel (on the right). The word the student was attempting to pronounce was <u>bow</u>.

A second difficulty that native speakers of Spanish have in learning to speak English, is that of producing initial aspirated stops /p,t,,k /. A native speaker of English delays voicing onset following these initial stop consonants for a few tens of milliseconds. A common error made by a native speaker of Spanish is to initiate voicing too soon, thus making what should be unvoiced consonants sound like their voiced counterparts. Therefore, a program was also written to display aspiration time, shown in Figure 13.29.

The horizontal line at the bottom represents the segment of the utterance during which voicing occurred. The vertical line represents the point at which the stop was released. The dots that form a more-or-less parabolic curve represent aspiration intensity at successive ten millisecond intervals. The API system was used in a variety of experimental training situations (Kalikow and Swets, 1972).

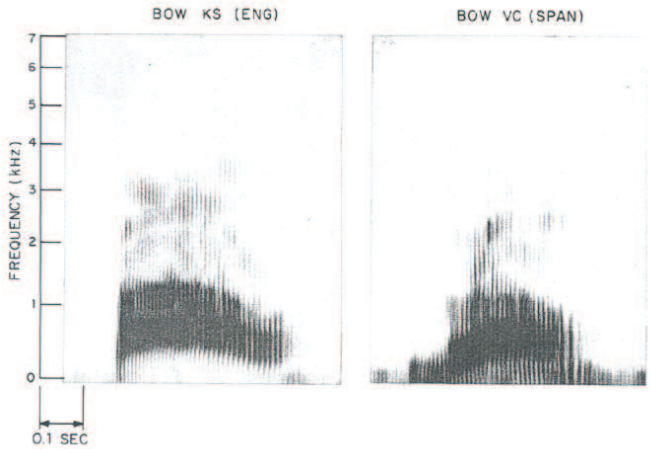


Figure 13.28 Tongue position displays.

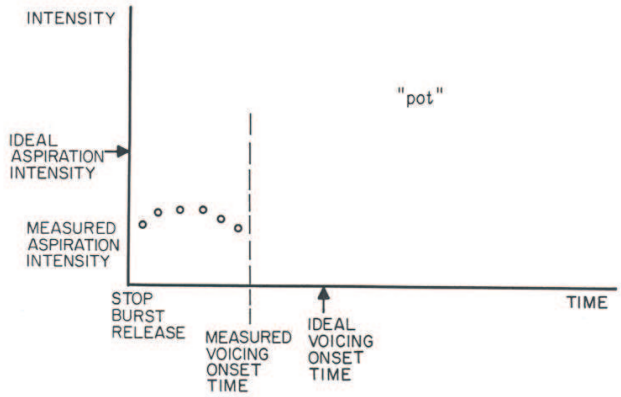


Figure 13.29 Aspiration time display.

Speech training aids for the deaf

An outgrowth of the work on second language learning was the development of an experimental computer-based system of visual displays that could be applied to the problem of improving the speech of pre-lingually deaf children. Children who are born deaf or who become deaf during the first couple of years of life do not develop normal speech capability, because they are deprived of the feedback channel through which hearing children learn to speak by comparing their utterances to those of other people around them. The inability to communicate via speech has profound implications for educational, social, and vocational development. The use of visual displays to help teach the correct pronunciation of a foreign language quite naturally led to the thought that such displays might also be useful in teaching speech to deaf children. The displays would not necessarily be the same, of course, but the basic idea of analyzing speech in a variety of ways, representing the results of those analyses visually, and providing students with visual targets to match seemed transferable to the new problem context. A system similar in many respects to the Automatic Pronunciation Instructor was designed and built (Nickerson and Stevens, 1973; Nickerson, Kalikow, and Stevens, 1976). The computer was a DEC PDP-8E. The configuration of peripherals was slightly different, but, like the earlier system, this one also contained a CRT display. Precisely what to display by way of speech properties was not clear a priori. It is not as though the speech of deaf children typically needs a little fixing. The problems tend to be numerous and complex (Nickerson, 1975; Nickerson and Stevens, 1980; Nickerson et al., 1983). They cannot be worked on all at once and there was very little in the literature to give guidance regarding where best to start. With the intent of providing a basis for exploration, the BBN system was programmed to produce several types of displays. Some of these were intended to support vocal exercises in game-like situations; others provided continuous feedback regarding one or more specific speech parameters during the emission of connected speech. The properties or speech that the system could display included amplitude, fundamental frequency, nasalization, and spectral distribution. One game-like display is illustrated in Figure 13.30. It shows a ball moving at a constant speed from left to right across the screen.

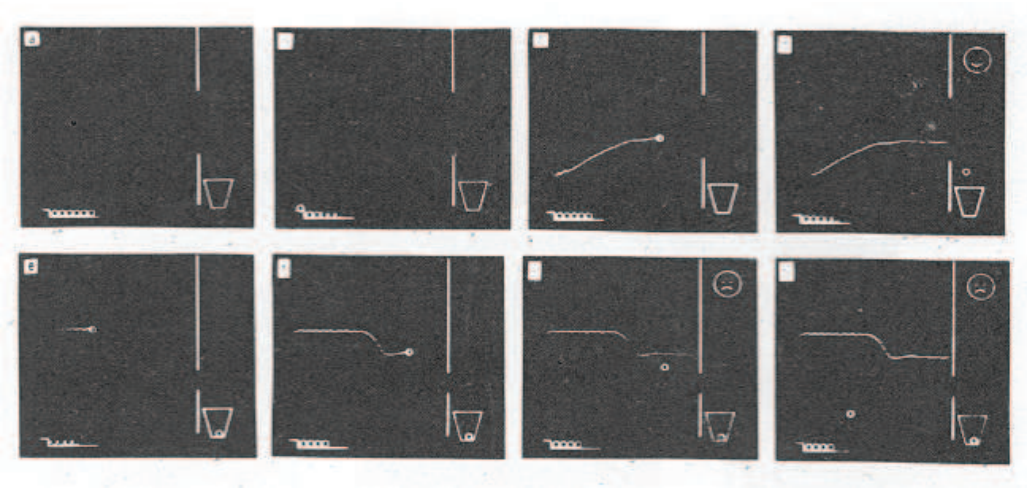


Figure 13.30 Pitch-controlled game display.

The height of the ball was controlled by the pitch of the speaker's voice. A vertical line positioned toward the right side of the screen represented a "wall" with a "hole" in

it. The student's task was to adjust the pitch of his voice so that when the ball arrived at the wall it would be at precisely the right height to pass through the hole. If it did, it then dropped into a basket on the far side of the wall and a smiling face appeared in the upper right corner of the display. It the ball arrived above or below the hole, it rebounded to the left. Both the height and width of the hole could be adjusted by turning a control knob. The top sequence shows a successful trial; the bottom one an unsuccessful one.

In a more complicated version of the game, two walls were used, separated by an adjustable difference. The heights of the holes in these walls could be different, thus forcing the student to change the pitch of his voice during a short time period, in order to get the ball though both holes. This game was used to teach students to control the pitch of their voices and in particular, to drop the pitch at the end of an utterance, which is something hearing speakers spontaneously do, but prelingually deaf children typically do not.

The display used most often with students was one that showed speech amplitude as a function of time. This display was used in training sessions aimed at improving the temporal properties of the children's speech. The need for such training is illustrated by the fact that one characteristic of the speech of deaf children is a lack of differentiation between stressed and unstressed syllables. In the speech of hearing speakers, stressed syllables may be slightly louder than unstressed syllables, and almost invariably are considerably longer in duration. The amplitude-versus-time display was used to help the deaf children modify the temporal characteristics of their speech, bringing them more in line with the temporal patterns produced by hearing speakers. The usual approach was for the teacher to illustrate the appropriate timing of an utterance by making the utterance and displaying its temporal pattern on the top half of the display. This pattern would remain in view as the student attempted to produce one on the bottom half of the display that would approximately match it.

Two of these systems were built and installed in two schools for the deaf — the Clarke School for the Deaf in Northhampton Mass., and the Lexington School for the Deaf in New York City where they were used on an experimental basis for several years. Some formal experiments were done to determine whether training procedures based on the use of specific displays would be effective in modifying the speech of deaf children in desired ways, and in particular with respect to nasalization, fundamental frequency, timing, and voice quality. This work was documented in a series of reports (Nickerson and Stevens, 1980; Stevens, Nickerson, Boothroyd, and Rollins, 1976).

While the speech of most of the participating students was modified in ways targeted in their training objectives, measured improvements in intelligibility were not consistently realized. One general conclusion that came out of the project was that there is a need for greater knowledge of how the intelligibility of speech depends on its objectively measurable properties. It is relatively easy to specify various ways in which the speech of a particular deaf child differs from the norm. However, given our current state of knowledge, it is difficult to say which aspects of the speech one should attempt to change in the interest of affecting the greatest improvement in intelligibility during limited training time.

In addition to being used in formal experiments, the systems were also employed at the schools where they were installed for a variety of other purposes. These included making measurements on children's speech for purposes of diagnosis, self tutoring (some children used the systems on their own to help wok on specific aspects of their speech), and teacher training. Several additional efforts to apply computer technology to the problem of enhancing the speech of deal children have been initiated since the completion of this project, both in this country and elsewhere.

Reading Aide

The number of adults in the population with unacceptable levels of literacy is enormous. Illiteracy costs the United States over 225 billion dollars annually in corporate retraining, lost competitiveness, and industrial accidents. The implication is clear: our goal of providing a modern competitive workforce hinges very directly on our ability to achieve a massive improvement in adult functional literacy during the next decade. This cannot be accomplished through the use of human teaching alone. There simply are not enough reading instructors in the country. Their teaching must be augmented by the creation and widespread application of an effective technology for automating literacy tutoring. More than one out of five adult Americans is functionally illiterate and their ranks are swelling by about 2.3 million persons each year. Nearly 40 percent of minority youth and 30 percent of semiskilled and unskilled workers are illiterate.

Although for a small fraction of illiterates the ability to read is impeded due to neurological problems, and for others there are learning difficulties that are not associated with sensory or motor problems, the primary cause of illiteracy among Americans is a failure to learn to read. For most adult illiterates, a major obstacle to effective reading development lies in two simple facts. The human resources do not exist to provide the teaching support that is needed, and there is no way of adequately increasing their number to provide such support during the next several years. We cannot develop a sufficient force of trained professionals and paraprofessionals at the level of expertise required, even with a massive injection of funding. The only option we have is the effective introduction of appropriate technology.

Learning to read requires time and practice. Research indicates that once the basics of learning to read are in place, a grade-level gain in reading ability takes approximately 100 hours of engaged literacy training. Further, at beginning levels of reading, individual feedback, motivation, and guidance are critical. Studies show that students need 4-10 minutes each day of supported reading to progress for 1st to 2nd grade, and 20 minutes per day to progress from 3rd to 4th grade. In 1996, Marilyn Adams, Richard Schwartz, and Madelyn Bates developed a computer-based *Reading Aide* to address the early reading problem. It incorporates capabilities for advanced speech recognition and sophisticated speech analysis. It operates as follows. The computer displays a page from a book, indicates a sentence for the child to read, listens to the child reading, highlights words as they are read, detects dysfluencies, and responds accordingly. Students read at their own level and at their own rate.

The *Reading Aide* can detect a wide range of dysfluencies. Examples include incorrect words, skipped words, repeated words, stutters, starting over, getting stuck, hesitation, and mechanical reading. It responds appropriately — for example, by moving on, asking for a retry, reading to the student, and providing help when necessary. It can delegate some control of the system to the student. It has numerous modes, from a line at a time to "read me a story." The student can navigate within a story either forward or back, play a word or sentence, or request help. The intent is to maximize the detection of dysfluencies and to minimize false alarms. However, speech recognition is imperfect. Moreover, many early readers have immature elocution, some children have strong accents, and new readers' oral reading is not fluent. The system incorporates an explicit model of dysfluencies. The sentence grammar includes distinct probabilities for skipping, repeating, starting over, and getting stuck. The single word grammar includes distinct probabilities for arbitrary phoneme sequencing, stuttering, and likely substitutions. The system avoids making responses that are confusing. For example, telling a child he made a mistake when he didn't can be extremely damaging. Instead, its responses are designed to be informative (useful) but noncommittal. The system

keeps a log of the confidence of its responses for later analysis. It maps the responses into a decision tree, annotating each sentence with the acceptability of *each* possible response. Figure 13.31 shows the system architecture (Gifford and Adams, 1996).

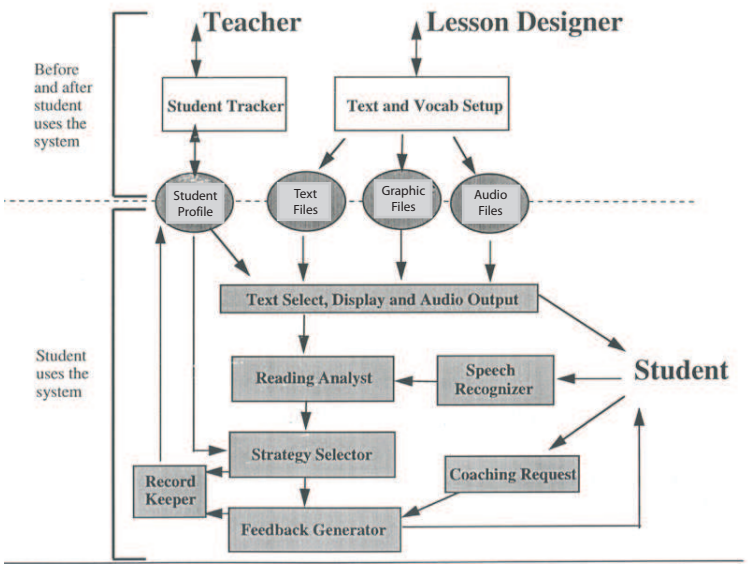


Figure 13.31 ***Reading Aide*** **system architecture.**

ILIAD

Dr. Lyn Bates, was Co-Principal Investigator with Dr. Kirk Wilson of Boston University on the project "Interactive Language Instruction Assistance for the Deaf," funded by the Bureau of Education for the Handicapped. The research was motivated by the fact that children who are prelingually deaf often never master standard English and usually lag far behind their grade level in reading English. Bates and Wilson hypothesized that one major reason is that they had not been exposed to many examples of certain key English structures, such as passive sentence forms. (An example: "The car was hit by the truck." as contrasted with the active form "The truck hit the car.") Readers need experiences with these kinds of English structures to understand how the syntactic structure affects the meaning of sentences.

To address this problem, they constructed *ILIAD*, a computer system that employed a transformational grammar to generate English sentences with random components. Particular features (such as passives, possessives, plurals, and various irregular forms) could be presented by settings under the control of the teacher. The system generated sentences instead of running through a fixed list, thus it never ran out of examples; students were not bored by having to repeat the same material over and over again.

ILIAD provided students with several game-like exercises. The system was used in classes at the Boston School for the Deaf (Bates and Wilson, 1981). The project won a Merit Award from Johns Hopkins University in the area of Personal Computing to Aid the Handicapped in 1981.

13.5 Training technology

From its early years BBN has been engaged in research and development involving the application of technology to technical training in complex task domains. Much of the work has focused on the introduction of new approaches employing sophisticated computer-based instructional technology based on methods derived from artificial intelligence and computational modeling. This section describes several such training applications in complex systems operation and maintenance tasks as well as aircraft piloting and tactical decision making tasks requiring support for real-time responses.

STEAMER

STEAMER was an advanced computer-based interactive graphics-oriented expert system for training operation and maintenance of complex steam propulsion power plants. It was developed by Bruce Roberts, Albert Stevens, Larry Stead, Albert Boulanger, and Glenn Abrett, under support of the Navy Personnel Research and Development Center in 1978–1983. A Navy steam propulsion plant is a very complex physical system consisting of thousands of components interconnected by miles of pipes, and requiring the operation of a team of 16 to 25 individuals. Years of instruction and experience are required to develop the understanding and skill for competent operation of a plant. The driving idea behind *STEAMER* was to enhance operator training through the development of a propulsion plant multi-level simulation with a color graphics interface and an intelligent tutoring component capable of instruction, guidance, and explanation of plant operation, operating procedures, and underlying operational principles (Stevens, Roberts, and Stead, 1983; Stevens et al., 1981; Hollan, Hutchins, and Weitzman, 1984).

Using an AI model of the propulsion plant, *STEAMER* generated interactive graphical diagrams of the entire plant and individual plant components at different levels of detail. The propulsion plant comprises many subsystems. *STEAMER* graphically depicted the flow of water or steam through these systems and the effects that various types of operator actions and system malfunctions would have on the operation of the plant. A screen shot of the *STEAMER* main steam cycle is shown at the left of Figure 13.32. An interactive diagram of the main engine lube system in *STEAMER* is shown on the right.

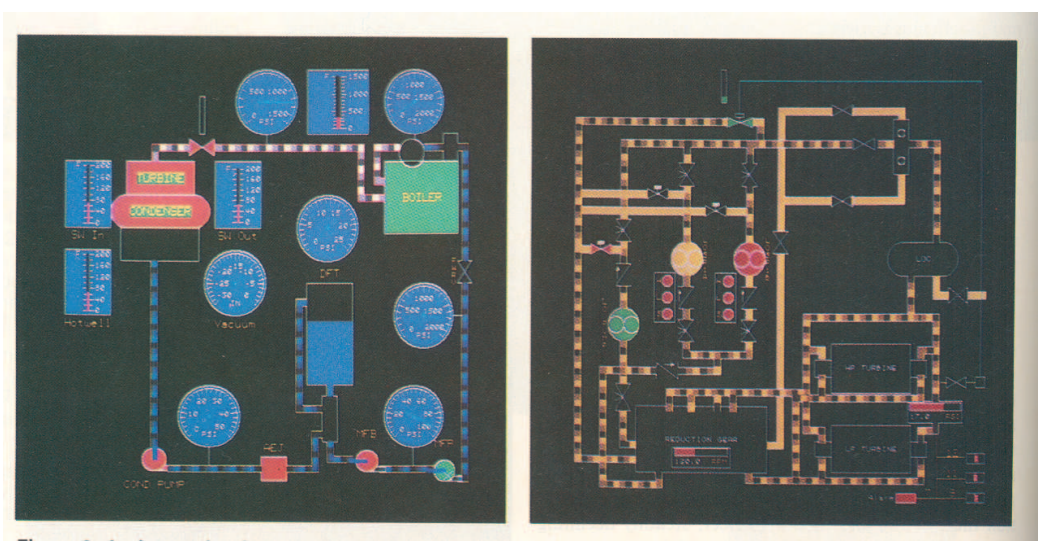


Figure 13.32 ***STEAMER*** **screen shots.**

The *STEAMER* instructional system provided a structured tutoring mode that presented problems to the student and guided him through a lesson. It supported exploratory learning activities that enabled students to perform "what if" experiments to discover the consequences of various operator actions. It could generate explanations of the operation of the plant, of what is happening, as the simulation is run. Thus, it could teach not only the plant's operating procedures, but also their underlying rationale. In describing a procedure for draining a chamber, it could explain the reason for the order of operations, e.g., why it is necessary to align the chamber's drain valves before opening an input valve to the chamber. (Because, otherwise, the water that is left in the chamber will mix with the steam, and high-energy water will get thrown downstream.)

STEAMER served as a compelling demonstration of the great potential of animated graphics representations driven by AI simulation models for making visually clear and understandable the dynamic interactive operation of complex physical systems comprising large-scale multi-level logical, electrical, and material components. A prototype *STEAMER* system was tested in a Navy training course. The system enabled students to inspect and operate a propulsion plant at various conceptual levels. Students found it easy to use, and programmers and curriculum developers found that its graphic editor readily enabled them to add and modify *STEAMER* diagrams. It was well received by users within the Navy training command.

ORLY

The *ORLY* flight training simulator was developed and employed in flight performance analysis research in 1974-1978 under support of the Naval Training Equipment Center and the Air Force Office of Scientific Research. The *ORLY* system development and instructional research was performed at BBN by computer scientists Wallace Feurzeig, George Lukas, Joe Berkovitz, Bill Huggins, Dan Stefanescu, Marty Schiff, and consultants Dan Cohen, Ken MacDonald, and Pat McHugh (Lukas, Feurzeig, and Cohen, 1975; Feurzeig, Lukas, and Stefanescu, 1980). The goal of the *ORLY* project was the development of computer-based methods for diagnostic performance analysis. The major research product was a performance analysis system for providing very specific characterizations of student pilots' performance on a variety of instrument flight control tasks. It enabled, not only the detection of performance errors, but also the generation of diagnoses characterizing the students' underlying difficulties. The first versions of the *ORLY* flight simulator were implemented on the DEC PDP-15 and PDP 1/PDP-10 computer systems by Cohen. The final version and the associated performance analysis facilities were implemented at BBN on a DEC GT-44 graphics display system with a PDP-11 CPU.

ORLY presented a realistic and fairly complete set of stylized but fairly realistic instruments on the bottom half of the display. The panel provided standard presentations of attitude, airspeed, altitude, heading, rate of climb, rate of turn, power, time, a compass rose/digital compass, an automatic direction finder (ADF), and an instrument landing system (ILS). The outer marker was at the ADF and a beeping and flashing of the corresponding instruments indicated passage over it and over the middle marker. A schematic and sparse cockpit window view occupied the top half of the display screen. The objects presented were the crossed airport runways, block structures corresponding to airport structures and antenna towers, and a graduated cloud ceiling. Distant mountains provided horizon information. The window view was primarily used for take off and landing. Figure 13.33 shows the *ORLY* instrument display and window view at three successive stages of a final approach.

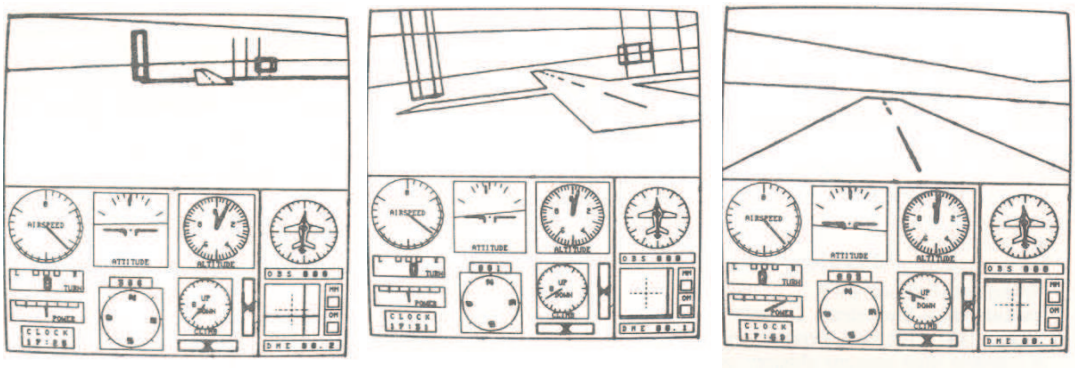


Figure 13.33 *ORLY* instrument display and window view.

The performance analysis system employed state-of-the-art computer feature extraction and pattern recognition methods expressly designed to mirror the analysis procedures used by expert instructors. In the first phase of the analysis, pilot performance data on task-dependent flight parameters (such as glide path, heading, rate of turn, etc. for instrument approaches) are fitted by a connected sequence of line segments. In the second phase, each segment is labeled by a set of attributes that characterize its performance relative to prescribed course and tolerance regions associated with that parameter in the flight plan. A segment is characterized by its location with respect to the tolerance region, by its length (duration), and by its slope. In the third phase, sequences of labeled segments, both within and across parameters, are interpreted as control patterns. Error and correction patterns of various types are identified. The control patterns include under-corrections, over-corrections, oscillations, and stabilizations. These patterns are specified in the program by formal procedures. In the last phase, this set of partial descriptions is integrated to produce an analysis narrative describing salient features of the pilot's performance during the task. Errors, error patterns, and contextual information to help explain difficulties are identified.

These methods were successfully applied to the analysis of basic instrument flight tasks, e.g., closed patterns such as figure 8s and cloverleafs, incorporating climbs, descents, timing variations, and airspeed changes. Figure 13.34 shows two such patterns.

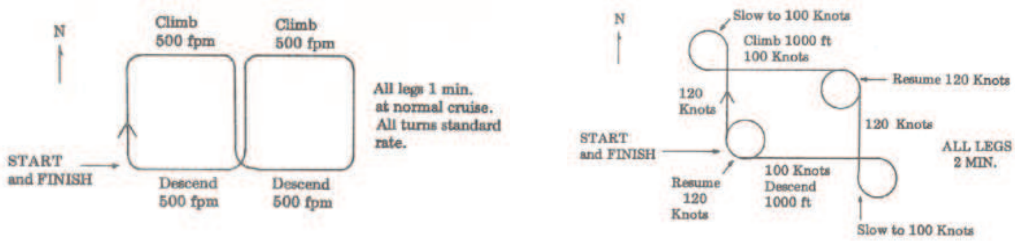


Figure 13.34 Basic flight patterns.

A typical flight map generated by a student pilot flight trial of the first pattern above and the chart record generated by *ORLY* for this flight are shown in Figure 13.35. The student's rate of climb, rate of turn, heading, altitude, airspeed, and power are charted

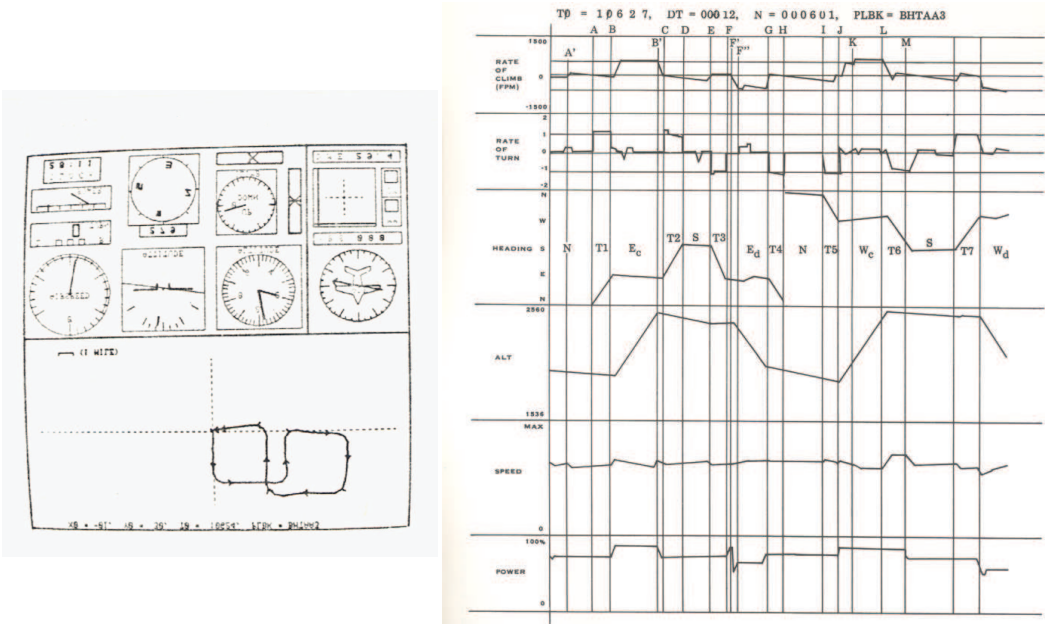


Figure 13.35 Typical flight map and chart record.

across all flight segments. A multi-level, context-sensitive, task-dependent procedure employing a pattern recognition grammar then performs a detailed analysis of the chart data. It recognizes standard performance patterns as well as canonical errors such as ballooning, diving, beginning a turn in the wrong direction, waiting too long to begin roll-out on a new heading, improper use of pitch and power, over-banking on turn entry, increase or decrease of bank during a turn, erratic bank control, shaky turn, beginning roll-out in the wrong direction, and climb/descent instead of descent/climb.

During the last phase of the analysis, the system produces a coherent summary of the significant features of the student's performance in language familiar to instructor pilots. The system was used in the analysis of 150 closed turn patterns flown by 16 student pilots. Comparisons of instructor analysis of these flights established that all unequivocal errors and error correction patterns were found and correctly identified by the system (Lukas, Berkovitz, and Feurzeig, 1977). For example, the system's description of the student's heading control during the first straight leg in the figure 8 trial shown above was: "The pilot established the heading and maintained the course for 53 seconds. An uncorrected drift then occurred while the pilot was having difficulty with altitude control." An instructor pilot's description of the same leg was: "Pilot drifts from heading until he is well outside the tolerance range. The pilot is apparently occupied by the altitude adjustments he has to make when the drift occurs." Instructors judged the performance summaries to be correct and essentially complete at that level of description.

Subsequent work with *ORLY* involved more complex types of maneuvers involving navigation components as well as instrument control. Among these tasks were holding patterns and ADF and ILS approaches, including missed approaches. These tasks entailed new and more complex errors involving, for example, glide slope and localizer parameters. In some phases of this work, pilots were given a great deal more latitude in their choice of flight path and in their mode of execution of the maneuvers. Thus, the unambiguous determination of certain errors was more difficult than for tasks with

completely prescribed plans. Despite the increased task complexity, the methods used for analysis of turn pattern tasks also appeared effective in the analysis of navigation tasks.

TRIO

TRIO (Trainer for Radar Intercept Operations) is an expert instructional system for training F-14 interceptor pilots and radar officers in dynamic spatial reasoning and the basic tactics of high-speed air intercepts. It was developed by Wallace Feurzeig, Frank Ritter, William Ash, Barbara White, and Michael Harris under support from the Navy Training Systems Center in 1983-1988 (Feurzeig, Ash, and Ricard, 1984; Ritter and Feurzeig, 1988). *TRIO* was designed to provide training in the effective conduct of air intercept operations by an F-14 radar intercept officer (RIO) in defense of an aircraft carrier or other naval asset. The *TRIO* task environment supports simulations of airborne radars, interceptor and target aircraft operations, and weapons models. It provides dynamic displays of heading, bearing and displacement vectors, radar screens, flight instruments, intercept parameters, radar and missile envelopes, and interceptor/target aircraft ground tracks. It incorporates real-time speech recognition and synthesis subsystems including advanced capabilities for recognition of naturally articulated utterances from an extensive lexicon. *TRIO* supports three instructional modes: demonstrations by the *TRIO* expert program, student practice with optional guidance, and performance analysis and student debriefing following student practice (Panagos, Feurzeig, and Ritter, 1987).

TRIO was the first successful application of intelligent tutoring system technology to real-time tactical task domains. In the *TRIO* environment, trainees participate in simulated engagements under the guidance of expert software tools that incorporate knowledge of the task and the training issues. These programs can demonstrate correct intercept tactics, provide assistance to correct trainee misconceptions, evaluate trainee task performance, and adaptively generate reasoned explanations of effective strategies. During these engagements, trainees observe indicators of system function (including simulated sensor output) and manipulate standard aircraft system controls.

An expert program in *TRIO* is capable of performing the same intercept tasks that it trains. The *TRIO* expert is articulate — as it performs air intercept engagements it explains its performance along the way. Each time it takes an action (e.g., calls for a change in heading, altitude or airspeed, selects or fires a weapon, or changes the radar display presentation) it can state the reason for the action, not only what the action is intended to accomplish but also why this is desirable in terms of its current goal. The goal structures of the tactics employed in performing intercepts are explicitly represented in the rules that drive the *TRIO* expert. This enables rapid evaluation and execution of the rules and facilitates real-time intercept performance in rapidly changing air battle situations. It also aids in the generation of tactically-based explanations for the expert's actions, to better motivate the sense and purpose of the strategic thinking and spatial reasoning involved.

The articulate expert capability is central to *TRIO*'s capability for instructional demonstrations. In the *TRIO* demonstration mode, the expert program runs *TRIO* to perform an intercept in very much the same way the trainee is expected to perform it. The intercept problem is usually assigned by an instructor or generated by *TRIO*, but problems also may be posed to the expert by the trainee. The expert explains its actions and the underlying reasons for them in terms of its current subgoal structure. The knowledge is represented using a special form of production rule system — continuous running, interrupt-driven, goal-directed rules — to operate the articulate expert program.

The expert performs intercepts in real-time and explains its actions and reasoning along the way. It uses the identical information the student sees and drives the simulation through the same interface. The intent is to provide the trainee with concrete models that prepare him for his own attempt to do similar intercepts.

After a trainee has seen the articulate expert fly an intercept to demonstrate a new tactical procedure or the application of a familiar tactical procedure to a new situation, he typically tries to do it on his own using *TRIO*'s guided practice facility. His performance is monitored and recorded for subsequent analysis. The trainee may try to perform the intercept without help. Otherwise, *TRIO* is able to intervene throughout the run to provide specific guidance to aid his performance, such as advance warnings to help trainees notice and avert major errors that threaten the success of the intercept. Rapidly changing tactical situations such as those occurring in air battle engagements impose very intense attentional demands. So the guidance offered by *TRIO*, which may come when the trainee is very absorbed in the intercept task, must be communicated in a clear and non-intrusive manner without stopping or slowing the action or breaking the trainee's concentration and thought processes. In real-time tasks with high cognitive loads, guidance must be presented in a way that allows a trainee to maintain his attention on the tactical situation while noticing and assimilating instructional communications. This is accomplished in *TRIO* by the use of "demons."

A demon is a continuously active rapidly executing program that monitors the state of task-critical parameters to detect a specific event, such as the imminent loss of radar contact or missile threshold. Demons are used primarily to detect and report errors in time for correction by the trainee. If its event occurs, a demon takes two actions. It records the event on the history list for use in the post-flight performance analysis debriefing narrative and it alerts the trainee to the need to take timely corrective action. The alert is communicated as a short message in a demon display window, possibly accompanied by flashing or by alerting sounds generated by the speech output device.

During an intercept exercise, *TRIO* employs speech recognition capabilities to simulate the voice communications between the RIO and the simulated pilot. The *TRIO* speech recognition facility is capable of real-time recognition of naturally spoken English messages from a specified lexicon of allowable RIO utterances, including fairly complex flight directives such as "Come starboard hard as possible to a heading of two four zero degrees." The pilot carries out the flight directives spoken by the RIO trainee as he directs the intercept. The pilot's flight control operations and the interceptor's flight dynamics are simulated by programs. *TRIO* acknowledges each RIO directive and executes it exactly as a human pilot would (subject to limitations of flight dynamics and response time) by making appropriate changes in the instrument and radar displays.

At the conclusion of an exercise, *TRIO* replays the relevant segments of the intercept, reproducing the displays along the way, with an added ground track display that shows the effect of the trainee's actions, and comments on the trainee's performance at each action time. *TRIO* debriefs the trainee verbally, reviewing the development of the engagement, recalling trainee actions, evaluating the quality of trainee actions, indicating appropriate actions at each decision point, and providing a reasoned explanation in terms of established intercept doctrine to support the recommended action. The expert's actions and explanations are given as spoken utterances, using the real-time speech synthesis device. The use of speech as the output mode is necessary because the RIO trainee has to attend closely to the radar scope and tactical information displays that were generated during the rapidly changing air battle. This poses a heavy visual workload; the use of another modality that does not distract the trainee's monitoring of the displays or otherwise hamper his visual performance, is essential.

The analysis of the trainee's performance is based on the use of pattern matching

methods that compare the trainee's actions to allowable performance behaviors defined by a solution state space. The solution space represents alternative solution paths during each phase of the intercept as permitted by the prescribed engagement rules and procedures. These paths allow considerable variation in the kind, number, and timing of trainee actions over those demonstrated by the expert program in its execution of an intercept. The analysis identifies faulty action sequences, i.e. those that could not be effective in realizing the appropriate subgoals in the intercept solution space, and determines very specific reasons for their unacceptability in terms of their adverse effects on the intercept. The analysis enables *TRIO* to generate explanations of what the trainee did wrong, where it happened, why it was wrong, and what he should have done instead.

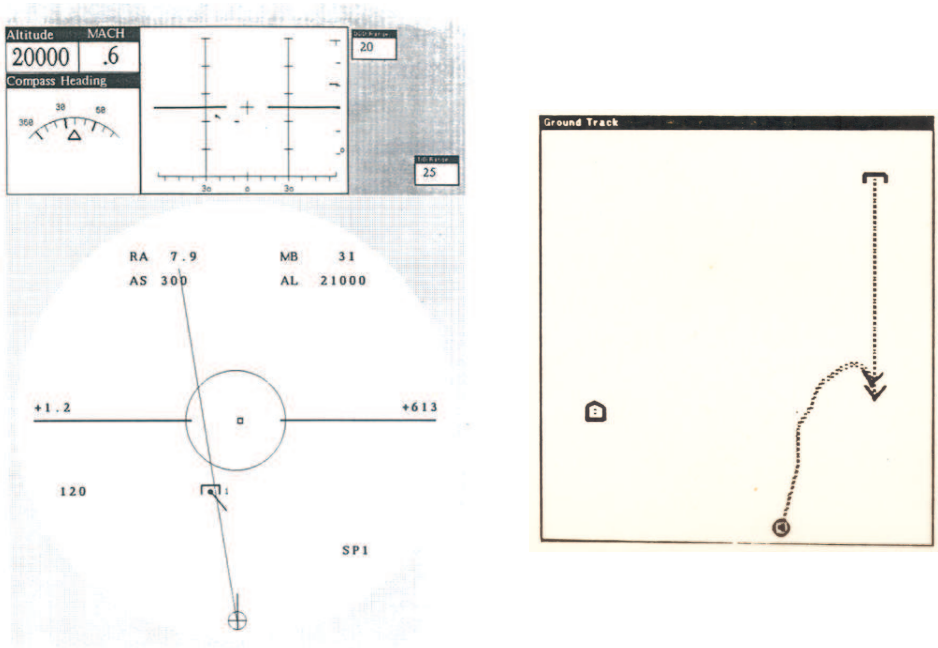


Figure 13.36 *TRIO* screen.

The *TRIO* screen is divided into a number of windows, as shown on the left side of Figure 13.36. These include displays of the RIO instruments (e.g., altitude, airspeed, heading, raw radar, and applicable target information). A text window provides the articulate expert's comments to the trainee when these are too verbose to present via the speech channel. The intercept track window is displayed during the debriefing mode following an intercept run. It shows the ground tracks of the RIO and the target aircraft that were generated during the run, as illustrated on the right side of the figure, which shows a successful intercept.

TRIO-based ideas, methods, and technology were incorporated in the BBN *INCOFT* project, described below. An operational version of the TRIO system was developed for training naval personnel at the Radar Intercept School in Pensacola, Florida.

MACH-III

MACH-III was a maintenance training aid computer for the Hawk system — an Intelligent Maintenance Trainer. It was developed in 1985-1989 by Dan Massey, Laura Kurland, Rob Granville, Dawn McLaughlin, Steve McDonald, Yvette Tenney, and Bruce Roberts.

(Massey, et al., 1986; Massey, deBruin, and Roberts, 1988; Tenney and Kurland, 1988; Kurland, Granville, and MacLaughlin, 1992). The work was done under contract to the U.S. Army Research Institute for the Behavioral and Social Sciences (ARI). BBN conducted an extensive series of experiments at the U.S. Army Air Defense Artillery School (AADASCH), Ft. Bliss, TX, to document the cognitive processes involved in successful (and unsuccessful) organizational maintenance of a complex electronic system, the AN/MPQ-57 High Powered Illuminating Radar (HIPIR) of the HAWK air defense system.

The understanding documented through these experiments was incorporated into the *MACH-III* intelligent interactive training system, which employed explanation-based reasoning to tutor trainee radar mechanics interactively in diagnosing and repairing faults in a simulated radar system. Acquiring expertise was a nontrivial task — the radar system comprised a number of subsystems with complex feedback loops, such as shown in Figure 13.37.

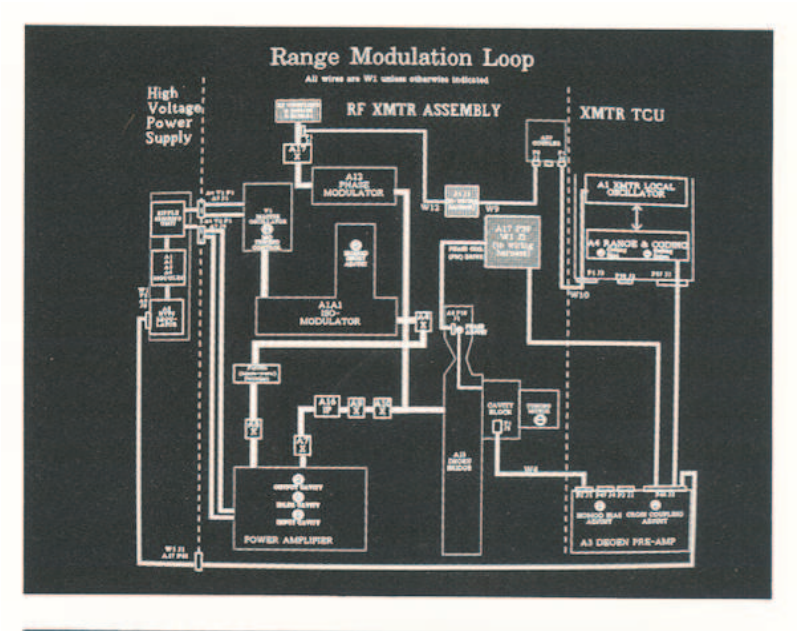


Figure 13.37 *MACH-III* subsystems.

The *MACH-III* system employed a novel approach to qualitative simulation of technical details of internal system functions and malfunctions. Using *MACH-III*, trainees explored the faulty behavior of a simulated system under the tutelage of task and tutorial expert programs. These programs demonstrated correct troubleshooting strategy, provided assistance to correct trainee misconceptions, prompted the recall of relevant knowledge, evaluated trainee performance (in the context of overall instructional goals), and adaptively generated reasoned explanations of system function and proper maintenance strategy.

In the qualitative simulation mode, the system enabled the trainee to manipulate all system controls and to observe all indicators of system function or malfunction. The system generated animated displays of the functional and physical organization of the radar, to help the trainee progress from understanding basic concepts to understanding the operation of the entire system. *MACH-III* included powerful facilities for adaptively generating reasoned explanations of system function and troubleshooting problem-

solving strategies. The system was designed for use in conjunction with standard Government troubleshooting manuals.

MACH-III represented a significant new approach to training organizational maintenance personnel. The system simulation facilities of *MACH-III* made it possible to give the trainee cognitive experiences similar to troubleshooting a real system. Instead of the traditional focus on masses of seemingly unrelated mechanical details and procedures, the powerful simulation models in *MACH-III* gave students both the experience and the conceptual understanding that more closely characterized experts who had years of field experience. Thus, the trainee's time with *MACH-III* was efficiently spent on developing the reasoning skills essential to expert troubleshooting performance. Although *MACH-III* was a prototype system, and was not formally approved for instructional purposes, USAADASCH adopted it as an informal instructional device, supplementing their established training program of lectures and hands-on practice.

INCOFT

INCOFT (the Intelligent Conduct of Fire Trainer) was a training system for operators of the Patriot Air Defense system. It was developed in 1986-1989 by Dan Massey, Denis Newman, Wallace Feurzeig, Mario Grignetti, and Mark Gross (Newman, 1991) under contract to the Army Research Institute for Behavioral and Social Sciences (ARI), with sponsorship by the Joint Services Manpower and Training Technology Development Program of the Assistant Undersecretary of Defense for Life and Environmental Sciences. BBN developed *INCOFT* to teach the skills required for making real-time tactical decisions in complex air defense operations. *INCOFT* was designed for USAADASCH, Ft. Bliss, TX, to train Patriot Air Defense Tactical Control Officers (TCOs) and Tactical Control Assistants (TCAs).

A knowledge-based expert system, *INCOFT* demonstrated and explained basic concepts, provided individualized practice time, and evaluated performance. The system prepared trainees for higher performance in initial job assignments in 30 percent to 50 percent less time than existing non-adaptive simulators, which lacked tutorial capabilities. *INCOFT* faithfully reproduced the physical, functional, and tactical conditions related to the specific skills being taught. It provided a trainee workstation that closely replicated the appearance and functionality of Patriot operator workstations. *INCOFT* simulation software mimicked the functionality of TCO and TCA workstations in a realistic engagement, replicating system behavior with sufficient fidelity to support required observations and actions. Trainee actions were monitored in real time during scenario execution, with continuing classification of performance. Immediate feedback and after-action analysis reviews were provided via a speech synthesizer controlled by the intelligent training software.

The *INCOFT* system provided trainees with an easy-to-use interface and an interactive learning environment. It incorporated much of the system architecture, AI methods, instructional strategies, and simulation and communications programs of TRIO, the BBN Trainer for Radar Intercept Operators (Massey, Feurzeig, Downes-Martin, and Ritter, 1985). The system provided multiple instructional modes. Typically, critical operator errors resulted in immediate intervention by the training expert, while less critical errors and omissions were noted during scenario replay for after-action review. Trainees could use the system without constant tutoring by instructors during practice sessions. Instructors could focus their time and energy on more advanced and complex training issues. This resulted in intensified instruction, accelerated learning, improved performance in initial job assignments, and greater operational readiness (Newman, Grignetti, Gross, and Massey, 1992).

QUEST

QUEST (Qualitative Understanding of Electrical System Troubleshooting) was an Intelligent Computer-Aided Instruction system for teaching electrical system troubleshooting (Feurzeig et al, 1983; Feurzeig, 1985; Frederiksen and White, 1984, 1989; White and Frederiksen, 1985, 1986, 1990). *QUEST* used qualitative simulation methods to teach knowledge-based reasoning about circuit behavior and troubleshooting. Humans think about the behavior of phenomena and systems in a qualitatively different way from that used to describe such behavior in mathematical simulation models. Experts in a domain (not only beginning students) use qualitative modes of thought and qualitative models to reason about system behavior. Thus, though it is necessary to employ mathematical simulations to obtain precise detailed descriptions of system behaviors, we also sought to teach conceptually sound qualitative reasoning. The use of qualitative simulation models is valuable for producing understandable explanations and for generating animated displays to show dynamic behavior. This facilitates learning by fostering the student's development of effective mental models for understanding and reasoning about system behavior.

The *QUEST* expert system employed a qualitative simulation model for reasoning about the behavior of RLC electrical circuits composed of batteries, wires, resistors, coils, condensers, lamps, switches, and test lights (Ritter, 1986). *QUEST* was capable of modeling the dynamic behavior of capacitors and inductors in relatively complex circuits. The qualitative simulation included a description of the circuit topology, a runnable functional model for each device in the circuit, rules for evaluating device states at each time increment, and circuit tracing procedures to aid in evaluating conditions for device states. The program generated graphical representations of circuit operation. It was designed to support a dynamic presentation environment within which an expert troubleshooting program could demonstrate troubleshooting concepts and strategy. The expert tutor could be called to solve problems and to demonstrate to students the reasoning involved. *QUEST* also provided an instructional mode for supporting student practice on troubleshooting problems. The program generated explanations of circuit operation in both working and faulted states, employing the same qualitative reasoning principles used in the execution of the expert troubleshooting strategy.

The *QUEST* instructional system provided students with a problem-solving environment within which circuits could be built, tested, and modified. Some circuit problems challenged students to make predictions about circuit behavior or to troubleshoot circuit faults. The qualitative causal simulation was run to illustrate principles of reasoning about circuits. The expert troubleshooter operated in interaction with the simulation program as it demonstrated a strategy for isolating faults. It incorporated the same type of reasoning as that involved in predicting circuit behavior. When solving problems, students could call upon these programs to explain reasoning about circuit operation or troubleshooting logic. Each tutorial program utilized a model that expressed its reasoning at a level of explanation appropriate for that particular stage of instruction. The circuit simulation program explained the operation of circuits in faulted as well as working condition. The troubleshooting expert generated explanations of troubleshooting logic. The *QUEST* project was supported jointly by the U.S. Office of Naval Research and the U.S. Army Research Institute.

QUIMON

During the last months of the *QUEST* project, Feurzeig and Ritter designed and implemented the *QUEST* Instructional Monitor, *QUIMON*, which embodied a novel approach

to cognitive analysis (Feurzeig and Ritter, 1988). The distinctive diagnostic feature of *QUIMON* that set it apart from other ICAI systems was the incorporation of the strategy of eliciting explicit information from the student about his troubleshooting actions throughout the course of the problem interaction. The student states what he hopes to learn from each of his actions on the circuit prior to its execution by the circuit simulator. After its execution he lists any conclusions about circuit faults he draws from seeing the simulator's effect on the state of the circuit. This strategic procedure engages the student as an active participant in facilitating the critique of his own problem work; he contributes valuable primary source information to aid the tutor in making more informed and more valid diagnostic inferences about the student's knowledge, bugs, and learning difficulties. This contrasts with the AI inferencing strategy of attempting to develop a cognitive model based on the student's actions without direct input from the student on the intent of his actions or the implications of their effects.

Here is a brief summary of the application of this strategy in a troubleshooting problem scenario. The student is presented with a (presumably faulty) circuit at a level of complexity appropriate to his current phase of training. As he acquires knowledge of the circuit's behavior, he is asked to develop and maintain a list of suspected faults. Initially, all circuit components may be suspect; at the end of an investigation the list will be reduced to those the student has isolated as faulty. The student investigates circuit behavior through a sequence of actions (e.g., flipping a switch, inserting a test probe, replacing a component). Before each action he is asked what he hopes to learn from performing it. He responds by selecting an item on the pre-action menu. After he responds, he calls the simulation to run. The simulation engine then carries out the requested action, and the student sees the effect of his action on the circuit behavior and state. He is asked what he has learned as a result of performing the requested action. He responds by selecting an item on the post-action menu. Following this response, the three-step process is repeated, continuing with the student's next troubleshooting action. This procedure generates a rich body of diagnostic data for the tutor. It also helps the student structure his approach to problem solving and develop more deliberate and reflective habits of thinking.

The interface is straightforward. The student answers a question by using a mouse to choose a response from a set of responses on the display. Possible student responses to the question "Why do you want to take this action?" on the pre-action menu include the following items: Don't know; To explore general circuit behavior; To test a component; To test the feed to a component; To test the ground side of a component; To replace a component. The student designates the component, feed, or ground of interest by pointing with the mouse. The circuit simulator then performs the requested action and changes the state of the circuit as appropriate.

After the requested action is taken by the circuit simulator, the program asks the student "What did you learn?" The post-action menu includes as possible student responses: Don't know; Identified component that may be faulty; Ruled out component as possible fault; Identified suspect subcircuit that may have a faulty component; Ruled out subcircuit as having a faulty component. Some answers require more than one response, e.g. the first might indicate that the student wants to add an entry to his list of possible faults, the second to point to the component or lasso the subcircuit suspected of being faulty, the third to designate the type of fault.

The elicitation procedure is designed to be non-intrusive and unforced. The student is advised that he does not have to be absolutely certain about the reason for every action he takes along the way to developing hypotheses. The direct manipulation point and click operation allows rapid and easy interaction. The session produces a substantial knowledge base of the student's plans and goals with minimal interference to his

troubleshooting activity. This fine-grained information about the student's intentions, expectations and conclusions can be valuable for understanding his performance and making plausible diagnoses of his misconceptions and difficulties. Moreover, such information can only be elicited from the student: it is, at the very least, extremely difficult for an ICAI system based on present AI methods to infer the student's mental states from his surface behaviors. Thus, we believe that the *QUIMON* work provides an effective starting point for development of more competent student diagnostic models.

The addition of information about the student's intentions, expectations, and plans, as well as his observed actions, is essential to making informed and insightful diagnostic hypotheses. This approach to diagnosis integrates commonsense principles from cognitive science with AI inferencing methods. It enhances the power and reliability of ICAI inferencing. It enables a wide range of applications to complex maintenance and troubleshooting training. The approach has obvious limitations. It assumes the principle of rationality, that problem solving behavior, whether correct or not, is always rational even when based on incorrect knowledge. Further, the elicitation procedure is not applicable in situations where students lack the knowledge or the appropriate vocabulary and language for talking about their problem solving plans and goals. Also, in real-time situations, where tasks have to be performed "on the fly," there is little time available to discuss the student's actions along the way non-distractively. Other instructional methods are required here, like those illustrated in the work on *TRIO* (Ritter and Feurzeig, 1988).

13.6 Educational networking

Soon after the development of computer time-sharing, BBN researchers explored the use of this new communication technology in schools with educational projects such as Stringcomp, which enabled multiple users remote access to interactive computing facilities. Following the development of the ARPANET, BBN began to investigate the application of computer networking technology to provide new ways for people to work together to improve learning and teaching. This section describes representative projects that addressed these new opportunities and challenges.

Co-NECT

In 1992, BBN successfully competed in a national competition sponsored by the New American Schools Development Corporation (NASDC) to design a new generation of "break-the-mold" schools. BBN's winning design, called *Co-NECT*, was selected along with ten others from a field of nearly 700 proposals. *Co-NECT* provided a framework for school-wide reform combining successful teaching practices with a new kind of school organization — all supported by internetworking technologies. The Co-NECT schools project was directed by Bruce Goldberg, Henry Olds, and John Richards.

Co-NECT schools are organized around small clusters of students taught by a cross-disciplinary teaching team. Most students stay in the same cluster, with the same teachers, for at least two years. Working within national, state, and district guidelines, teachers used performance standards defining what graduating students should know and be able to do. The curriculum revolved around "authentic" interdisciplinary projects designed to give students an opportunity to acquire critical skills and understanding.

Faculty representatives of each cluster served on a school design team. Led by the building principal, with input from parents and other members of the community, the team set overall goals and monitored results. A sophisticated communications infrastructure gave Internet access to everyone in the school community. Every *Co-NECT*

school and school district received individual attention from a support team headed by field representatives, consultation with members of the *Co-NECT* design team on an "as needed" basis, and involvement in teleconferences. Schools also took part in the *Co-NECT* Exchange (an Internet-based information service, electronic forum, and support tool), and *Co-NECT* Critical Friends (a program of reciprocal school visits). As a school developed its own internal capacity for sustained educational restructuring and growth, assistance from BBN continued, with increasing reliance on video conferencing and other means of remote support.

The long-range *Co-NECT* technology plan included communications technologies providing students and teachers access to people and information resources both inside and outside the school; ubiquitous access to networked computing tools to provide a solid support structure for project-oriented workgroups; a technology-enriched base of information sources, including video and audio as well as electronically accessible text, data, and graphics; powerful software tools to support many subject matter and skill-building elements of the curriculum; multimedia tools for both exploration and "publication"; networked software used to help manage the scheduling and project development needs of the clusters; and software tools for managing the certificate-based assessment system and for organizing and presenting student portfolios of selected work.

After two years of testing and refinement, the *Co-NECT* design was used by an expanding network of schools and districts around the country, including schools in Juneau, Alaska; Worcester, Massachusetts; Cincinnati, Ohio; Memphis, Tennessee; and Miami, Florida. *Co-NECT* left BBN in 1998 to operate as an independent organization centered in Cambridge, Massachusetts (Morrison, 1997).

National School Network Testbed

With the rapid growth of Internet use in the United States, it became increasingly important to understand how to use these new communication channels and resources most effectively in education. It takes considerable investment to create the technical and organizational infrastructure necessary to support wide participation across a community. There was a need to research and share information about successful models, the benefits as perceived by learning communities, and the investment required. An empirical base of knowledge was needed in order to make sound policy decisions about investment on the part of local, state, and national governments. The National School Network Testbed (*NSNT*) was funded by the National Science Foundation to help develop that knowledge.

Approximately 250 institutions participated in the Testbed, including over 150 individual schools. Phase I of the *NSNT*, conducted over 18 months from 1992 to the spring of 1994, resulted in an understanding of ways schools and other educational institutions could take advantage of internetworking to build their own local information infrastructure in support of desired reforms in education. BBN scientists Beverly Hunter and Denis Newman directed the BBN effort (Newman, 1993; Hunter, 1995).

Phase 2 of the project, which began in the fall of 1994 and continued through 1997, was designed to build models for developing relationships among schools and their communities through the use of telecommunication, so as to enrich the education of both children and adults. The project conducted a longitudinal study to investigate the effect of Internet technology on participating schools over the three-year period. The study collected descriptive and analytic data to determine the extent and nature of changes.

The report identified the need for a product that schools and other organizations

might use to manage their own Internet services. In response to this need, BBN developed the BBN Internet Server, and made it available to schools and other educational organizations. The Data Communications journal gave it an award as Product of the Year in January 1995. Teachers, students, and administrators used the server for communication and data access, both within their organizations and throughout the international Internet. Most significant for educational settings is the fact that users could manage the day-to-day operation of the server without having to use its native environment (UNIX). Instead, a teacher could use an associated educational management program, FrontDoor, that communicated with the server to carry out tasks such as adding new users to group or individual accounts, creating or modifying electronic mailing lists and newsgroups, and publishing Web pages.

MuseNet

The Multi-User Simulation Environment Network project, *MuseNet*, was supported by the National Science Foundation in 1995 under the program "Networking Infrastructure for Education." The research was performed by Wallace Feurzeig, Paul Horwitz, Barry Kort, David Fagan, Kenneth Schroder, and Natasha Cherniak. The goal of the project was the development of scalable distributed networking technology for supporting collaborative environments for science and mathematics education. The project was designed to demonstrate the power of distributed server technology in addressing a key "scale up" problem posed by the dramatic growth of educational traffic on wide area networks. By distributing large, computationally intensive educational applications among multiple heterogeneous servers, *MuseNet* showed how to make considerably more efficient utilization of network resources with consequent improvements in client service and response times (BBN Systems and Technologies, 1996).

MuseNet was both a technology infrastructure demonstration project and an education research project. *MuseNet* sought to make a significant educational contribution by enabling real-time collaboration among users of science simulations and other computationally-intensive applications across the Internet. This called for the development and demonstration both of educational infrastructure for supporting collaborative interactions as well as technology infrastructure for supporting distributed educational applications.

The technology demonstration was built on prior work with the BBN Cronus distributed operational environment. The Cronus system was used to distribute and manage the operation of large-scale applications among multiple servers to sustain smooth operations and optimize response times. *MuseNet* employed the methods of the Cronus-distributed system to optimize server utilization across a large set of Muses dedicated to computationally intensive educational activities.

The participating Muses and their hosts included MariMUSE at Phoenix College in Arizona, MicroMuse at MIT, EcoMuse at theUniversity of Vermont, Bridge Muse at the University of Southern Maine, De Anza Muse at De Anza Community College in California, Graham and Parks Muse at the Graham and Parks public school in Cambridge, CyberMush at CNIDR, and two Muses at BBN, Academy Muse and WindsMare. There was enormous variation in the server load at each of these Muses at different times. Sometimes the level of user activity was extremely high at one site while it was fairly low at another. Further, there was a great difference in user loads and response times at any given site at different times. The disparity in server load within this multi-server community perfectly typified the general problem confronting wide area network management in the era of enormous and rapidly fluctuating network traffic. We showed how the use of *MuseNet* alleviated this problem through dynamic load balancing.

The networking infrastructure in *MuseNet* supported multi-user collaboration in science simulation and modeling activities by making modeling tools and applications operable within a MUSE environment. This new educational infrastructure combined two learning technologies that until then had been separate and unrelated. Work on computer simulation and modeling had been directed at fostering students' development of the "habits of mind" associated with scientific exploration and inquiry. Work on MUSEs had been directed at self-discovery and empowerment through shared encounters in text-based worlds constructed by students. We merged the two technologies by developing *MuseNet* facilities to support student communication and collaboration centered on the use of modeling and simulation tools and applications. Like current MUSEs, the use of this environment enabled students to meet over the net with each other and with teachers and scientists, on virtual field trips to diverse science modeling microworlds. It provided students with powerful facilities for supporting real-time collaboration on joint investigations employing modeling tools and applications.

This development made possible the integration of MUSEs with educational software tools and applications that were designed independently of MUSEs — programs like *GenScope*, the Geometer's Sketchpad, Function Machines, *RelLab*, Interactive Physics, Explorer Science, and other powerful modeling and simulation environments, particularly those that naturally lent themselves to collaborative activities. Students could thus work together, exploring, investigating, building, and modifying computational science structures and processes, using the social and conversational features of MUSEs to discuss their progress and to negotiate their moves in the course of running the programs.

To realize the integration, the following strategy was adopted. A datastream channel and associated multimedia channels on the server control, coordinate, and synchronize the commands for running the software as these commands are decided upon by the users through conversational negotiation on the MUSE. This mode of operation does not require high bandwidth networking — each time the model is run the only data that are transmitted are the commands for changing the state of the program and its current outputs. The integrated *MuseNet* system was demonstrated with two science simulation programs: Space (a 3-D astrophysical simulation of space travel), and *RelLab*, the BBN software for supporting students work in relativity experiments. These demonstrations showed the feasibility and educational benefits of integrating Muses and science simulations, through adding a social dimension to collaborative inquiry activities.

References

Abelson, H., and diSessa, A., *Turtle Geometry. The Computer as a Medium for ExploringMathematics*. MIT Press, Cambridge, MA.,1983.

Barowy, B., Richards, J., and Levin, D., Computer Simulations in the Science Classroom. *Journal of Science Education and Technology*, Vol. 1 No. 1, 1992.

Bates, M. and Wilson, K., ILIAD Project Final Report, BBN Report No.4771, BBN, Cambridge. MA. September, 1981.

BBN Systems and Technologies, MuseNet: A Scalable Architecture for Distributed Educational Applications, Final Report, April, 1996.

Brown, J. S., and Burton, R. R., Multiple Representations of Knowledge for Tutorial Reasoning. In Bobrow and Collins (Eds.) *Representation and Understanding: Studies in Cognitive Science*. New York, Academic Press, 1975.

Brown, J. S., Burton, R. R., and Bell A. G., SOPHIE: A Step Toward Creating a Reactive Learning

Environment. *International Journal of Man-Machine Studies, 7* (5), pp. 67–696, 1975.
Brown, J. S. and Burton, R. R., A Paradigmatic Example of an Artificially Intelligent Instructional System. *International Journal of Man-Machine Studies*, Vol. 10, 1978a.

Brown, J. S. and Burton, R. R., Diagnostic Models for Procedural Bugs in Basic Mathematical Skills. *Cognitive Science*, Vol. 2, pp. 155–192, 1978b.

Brown, J. S., Burton, R. R., and Hausmann, C., Representing and Using Procedural Bugs for Educational Purposes. *Proceedings, National Association for Computing Machinery*, October 1977.

Burton, R. R. and Brown, J. S., A Tutoring and Student Modelling Paradigm for Gaming Environments. *Proceedings of the ACM SIGCSE-SIGCUE Technical Symposium on Computer Science and Education*, February 1976.

Burton, R. R. and Brown, J. S., Toward a Natural Language Capability for Computer-Aided Instruction. *Readings in Natural Language Processing*, pp. 605–625,Morgan Kaufmann, 1979.

Burton, R. R. and Brown, J. S., An Investigation of Computer Coaching for Informal Learning Activities. *Intelligent Tutoring Systems*, pp. 79–98, Academic Press, 1982.

Carbonell, J. R., AI in CAI: An Artificial Intelligence Approach to Computer-Assisted Instruction. *IEEE Transactions on Man-Machine Systems 11* (4): 190–202, 1970.

Carbonell, J. R. and Collins, A. M., Natural Semantics in Artificial Intelligence. In *Proceedings, Third International Joint Conference on Artificial Intelligence*, 344–351, Stanford, Calif., 1973.

Carbonell, J. R. and Collins, A. M., and Warnock, E. H., Semantic inferential processing by computer. In J. Rose (Ed.), *Advances in cybernetics and systems*. London: Gordon & Breach, 1975.

Clancey, W. J., and Shortliffe, E. H., *Readings in Medical Artificial Intelligence: The First Decade*. Addison-Wesley Longman Publishing Co., Inc.. Boston, MA, 1984.

Clancey, W. J., Tutoring Rules for Guiding a Case Method Dialog. In Sleeman and Brown, (Eds.) *Intelligent Tutoring Systems*, Academic Press, London, 1982.

Collins, A. M., and Stevens, A.. Multiple Models of a Complex System. In (Snow, Frederico and Montague (Eds.) *Aptitude Learning and Instruction: Cognitive Processing Analysis*. Hillsdale, N.J., Erlbaum, 1980.

Collins, A. M., Adams M. J., and Pew, R. W., The Effectiveness of an Interactive Map Display in Tutoring Geography. *Journal of Educational Psychology*, 1978, 70, 1–7, 1978.

Collins, A. M., Warnock, E. H., and Passafiume, J. J., Analysis and Synthesis of Tutorial Dialogues. In: Bower (ed.) *The Psychology of Learning and Motivation: Advances in Research and Theory*, pp.49–87, Academic Press, New York. 1975.

Coulson, J. E., Programmed Learning and Computer-Based Instruction. Proceedings of the Conference on Application of Digital Computers to Automated Instruction. John Wiley & Sons, Inc., New York: 1962.

Cuoco, A., *Investigations in Algebra: Combinatorics and Number Theory*. MIT Press, Cambridge MA, 1990.

Cuoco, A., Action to Process: Constructing Functions from Algebra Word Problems. *Intelligent TutoringMedia*, Vol.4, No. 3/4, Learned Information Ltd., Oxford, U.K., November 1993.

Feurzeig, W., The Logo Lineage, book chapter in *Digital Deli*, Workman Press, 1984.

Feurzeig, W., Algebra as Logo. *Proceedings, MIT Logo'85 Conference*, MIT, Cambridge, MA, 1985.

Feurzeig, W., Algebra Slaves and Agents in a Logo-Based Mathematics Curriculum. Book chapter in *Artificial Intelligence and Education Volume One*, Ablex Publishing, Norwood, N.J., 1987

Feurzeig. W., Explaining Function Machines. *Intelligent Tutoring Media*, Vol. 4 Nos. 3,4, Learning Information Ltd., Oxford, U.K., 1993.

Feurzeig, W., Visualization in Educational Computer Modeling. Book chapter in *The Use of Computer Models for Explication, Analysis, and Experiential Learning*, Towne, de Jong, and Spada (Eds.), Springer-Verlag, 1994a.

Feurzeig, W., Visualization Tools for Model-Based Inquiry. Book chapter in Baker and O'Neil (Eds.) *Technology Assessment in Software Applications*. Lawrence Erlbaum, Hillsdale, N.J., 1994b.

Feurzeig, W., Function Machines Pre-Algebra Sequence, *CAETI Newsletter*, Number 14, `http://isx.com:80/CAETI/newsletters/newsletter.14.html`, 1997.

Feurzeig, W. A Visual Language for Model-Based Mathematics Inquiry, book chapter in *Modeling and Simulation in Science and Mathematics Education*, W. Feurzeig and N. Roberts (Eds.), Springer-Verlag, New York, N.Y. 1999.

Feurzeig, W., Ash, W., and Ricard, G., The Use of Artificial Intelligence Methods in Real-Time Tactical Training. *Proceedings, American Education Research Association*, April 1984.

Feurzeig, W., Horwitz, P., and Boulanger, A., Advanced Mathematics from an Elementary Viewpoint: Chaos, Fractals, and Non-Linear Systems. *Proceedings, Computers and Mathematics Conference*, MIT, Cambridge, MA, June, 1989.

Feurzeig, W. and Lukas, G., Logo: A Programming Language for Teaching Mathematics. *Educational Technology*, 1972.

Feurzeig, W., Lukas, G., and Lukas, J. D., *The LOGO Language — Learning Mathematics Through* Programming, Entelek, Newburyport, MA, 1977.

Feurzeig, W., Lukas, G., and Stefanescu, D., Compuer-Based Interactive Flight Training. Final Report, Bolt Beranek and Newman, Cambridge, MA., November, 1980.

Feurzeig, W., Munter, P. K., Swets, J. A., and Breen, M.N.. Computer-Aided Teaching in Medical Diagnosis, *Journal of Medical Education*, Vol. 39, No. 8, 1964.

Feurzeig, W., and Papert, S., Programming-Languages as a Conceptual Framework for Teaching Mathematics. *Proceedings of the NATO Conference on Computers and Learning*, Nice, France, 1968.

Feurzeig, W., Papert, S., Bloom, M., Grant. R., and Solomon, C., Programming-Languages as a Conceptual Framework for Teaching Mathematics. Final Report on the first fifteen months of the Logo project, BBN Report No. 1989, Bolt Beranek and Newman, Cambridge, MA., November 1969.

Feurzeig, W., and Richards, J., Intelligent Tools for Algebra, *Technology and Learning, 2* (31), 1988.

Feurzeig, W., and Ritter, F., Understanding Reflective Problem Solving. In *Intelligent Tutoring Systems: Lessons Learned*. (Psotka, Massey, and Mutter, eds.) Lawrence Erlbaum, Hillsdale, N.J., 1988.

Feurzeig, W., and White, B., An Articulate Instructional System for Teaching Arithmetic Procedures. BBN Technical Report, Bolt Beranek and Newman, Cambridge, MA., January, 1984.

Frederiksen, J., and White, B., Implicit Testing Within an Intelligent Tutoring System. *Machine-Mediated Learning*, Vol. 2 No. 4, 1988.

Frederiksen, J., and White, B., An Approach to Training Based Upon Principled Task Decomposition. *Acta Psychologica*, 71 (1–3), 1989.

Gifford, C., and Adams, M. J., Reading Aide Video. `http://www.iment.com/maida/tv/songvids/mp4/computer/ReadingAide.mp4`.1996.

Goldenberg, E,P,. Learning to Think Algebraically: A Logo and Function Machines Contribution to Solving Word Problems. *Intelligent Tutoring Media*, Vol. 4, No. 3/4, Learned Information Ltd., Oxford, U. K., November 1993.

Goldenberg, E. P., and Feurzeig, W., *Exploring Language with Logo*. MIT Press, Cambridge, MA., 1987.

Grignetti, M. C., Gould, L., Hausmann, C. L., Bell, A. G., and Harris, G., Mixed-Initiative Tutorial System to Aid Users of the On-Line System (NLS). BBN Report 2969, Bolt Beranek and Newman, Cambridge, MA., 1974.

Harvey, B. *Computer Science Logo Style*. MIT Press, Cambridge, MA., 2nd Edition, Feb. 1997.

Hollan, J. D., Hutchins, E. L., and Weitzman, L., STEAMER: An Interactive Inspectable Simulation-Based Training System. *AI Magazine 5* (2), 1984.

Horwitz, P. and Eisenberg, M., MultiMap: An Interactive Tool for Mathematical Experimentation. BBN Report 7651, Bolt Beranek and Newman, Cambridge, MA, September 1991.

Horwitz, P. and Feurzeig, W., Computer-Aided Inquiry in Mathematics Education. *Journal of Computers in Mathematics and Science Teaching, 13*(3), 1994.

Horwitz, P., Neumann, E., and Schwartz, J., Teaching Science at Multiple Levels: the GenScope Program. *Communications of the Association for Computing Machines*, Vol. 38 No. 8, 1996.

Horwitz, P., and White, B., Computer Microworlds and Conceptual Change: A New Approach to Science Education. In *Improving Learning: New Perspectives* (Ramsden, Ed.) Kogan, London, 1988.

Hunter, B., Internetworking and Educational Reform: The National School Network Testbed. a BBN report, `http://www.isoc.org/HMP/PAPER/065/html/paper.html`, Bolt Beranek and Newman, Cambridge, MA, 1995.

Kalikow, D., and Swets, J. A., Experiments with Computer-Controlled Displays in Second Language Learning. *IEEE Trans Audio Electro-Acoustics*. Vol. AU-20, No. 1, March 1972.

Kurland, L. C., Granville, R., and MacLaughlin, D., Design, Development, and Implementation of an Intelligent Tutoring System for Training Radar Mechanics to Troubleshoot. In *Intelligent Instruction by Computer* (Farr and Psotka, eds.) Taylor and Francis, Washington DC, 1992.

Lewis, P,G., *Approaching Precalculus Mathematics Discretely*. MIT Press, Cambridge, MA., 1990.

Licklider, J. C R., Programmed Learning and Computer-Based Instruction. *Proceedings of the Conference on Application of Digital Computers to Automated Instruction*. (J. E. Coulson, ed.) pp. 217–239. John Wiley & Sons, Inc., New York: 1962.

Lukas, G., Berkovitz, J., and Feurzeig, W., Linear Approximation Techniques for Fitting Flight Performance Data. BBN Report 3615, Bolt Beranek and Newman, Cambridge MA, July 1977.

Lukas, G., Feurzeig, W. and Cohen, D., Research on High-Level Adaptive Training Systems, BBN Report 3193, Bolt Beranek and Newman, Cambridge MA, November 1975.

Lukas, G., and Lukas, J., *LOGO: Principles, Programming, and Projects*. Brooks/Cole Publishing Co., Monterey, CA., 1986.

Massey, L. D., Feurzeig, W., Downes-Martin, S., and Ritter, F., TRIO to INCOFT Adaptation Study. BBN Report 6194. Bolt Beranek and Newman, Cambridge MA, 1985 .

Massey, L. D., Kurland, L. C., Tenney, Y., DeBruin, J., and Quayle, K. C., Hawk Mach-III Intelligent Maintenacnce Tutor Design Developoment. BBN Report 6315, Bolt Beranek and Newman, Cambridge MA, Sept.1986.

Massey, L. D., DeBruin, J., and Roberts, B., A Training System for System Maintenance. 1988. In *Intelligent Tutoring Systems: Lessons Learned*. (Psotka, Massey, and Mutter, eds.) Lawrence Erlbaum, Hillsdale, N.J., 1988.

Morrison, D., The Co-NECT School: An Integrated K-12 Design. *Education Reform Digest*. 1997.

Myer, T. H., *Manual for Using TELCOMP Computation Service*, Bolt Beranek and Newman, Cambridge MA., October 1966.

Neumann, E., and Feurzeig, W., Object-Object Transformation Language in Science and Mathematics, Book chapter in Feurzeig and Roberts (Eds,) *Modeling and Simulation in Precollege Science and Mathematics Education*. Springer-Verlag, New York, NY, 1999.

Newman, D., Interpreting and Intelligent Tutor's Algorithmic Task: A Role for Apprenticeship as a Model for Instructional Design. *AI and Society*. Springer-Verlag, London, 1991.

Newman, D., School Networks: Delivery or Access. *Communications of the Association for Computing Machinery*, May, 1993.

Newman, D., Grignetti, M., Gross, M., and Massey, L. D., Intelligent Conduct of Fire Trainer: Intelligent Technology Applied to Simulator-Based Training in Intelligent Instruction, in *Intelligent Instruction by Computer* (Farr and Psotka, eds.), Taylor and Francis, Washington DC, 1992.

Nickerson, R. S., Characteristics of the speech of deaf persons, *The Volta Review*, pp. 342-362, 1975.

Nickerson, R. S., Kalikow, D. N., and Stevens, K. N., Computer-Aided Speech Training for the Deaf. *Journal of Speech and Hearing Disorders*, 41, 1976.

Nickerson, R. S., and Stevens, K. N., Approaches to the Study of the Relationship Between Intelligibility and Physical Properties of Speech. In J. Subtelny (Ed.), *Speech Assessment and Speech Improvement for the Hearing Impaired* (pp. 338–364). Washington, DC: A. G. Bell Association for the Deaf, 1980.

Nickerson, R. S., Stevens, K. N., Rollins, A. M., and Zue. V. W. Computers and Speech Aids. In Hochberg, Levitt, and Osberger (Eds.), *Speech of the hearing impaired: Research, training and personnel preparation* (pp. 313–324). Baltimore, MD: University Park Press, 1983.

Panagos, J., Feurzeig, W., and Ritter, F., TRIO System Documentation. BBN Report to Naval Training Systems Center, Orlando, FL., Bolt Beranek and Newman, Cambridge MA, June, 1987.

Quillian, M. R., Semantic Memory, Book chapter in *Semantic Information Processing*, M.Minsky (Ed.), MIT Press, Cambridge, MA, 1968.

Ritter, F., OREO: Orienting Electrical Circuits for Qualitative Reasoning. BBN Report 6560, Bolt Beranek and Newman, Cambridge MA, 1987.

Ritter, F., and Feurzeig, W., Teaching Real-Time Tactical Thinking. In *Intelligent Tutoring Systems: Lessons Learned.* (Psotka, Massey, and Mutter, eds.) Lawrence Erlbaum, Hillsdale, N.J., 1988.

Rubin, A., Rosebery, A. S., and Bruce, B., Elastic and Reasoning Under Uncertainty. BBN Report No. 6851, Bolt Beranek and Newman, Cambridge MA, June 1988.

Stevens, A.L, and Collins., The Goal Structure of a Socratic Tutor. *Proceedings, Association for Computing Machinery Annual Conference*, 1977.

Stevens, A. L., Roberts, B., Stead, L., Forbus, K. D., Steinberg, C., and Smith, B., STEAMER: Advanced Computer-Aided Instruction in Propulsion Engineering. BBN Report 4702, Bolt Beranek and Newman, Cambridge MA, 1981.

Stevens, A. L., Roberts, B., Stead, L., The Use of a Sophisticated Graphics Interface in Compuoter-Assisted Instruction. *IEEE Explore*, March/April 1983.

Stevens, K. N., Nickerson, R. S., Boothroyd, A., and Rollins, A., Assessment of nasality in the speech of deaf children, *Journal of Speech and Hearing Research, 19*, pp. 394-416, 1976.

Swets, J. A., Some Possible Uses of a Small Computer as a Teaching Machine, Insert in Feurzeig, Conversational Teaching Machine, *Datamation*, vol. 10 No. 6, pp. 38–42, 1964.

Swets, J. A., and Feurzeig, W., Computer-Aided Instruction. *Science*, Vol. 150, No. 3696, pp. 572–576, October, 1965.

Tenney, Y., and Kurland, L. C., The Development of Troubleshooting Expertise in Radar Mechanics. In *Intelligent Tutoring Systems: Lessons Learned.* (Psotka, Massey, and Mutter, eds.) Lawrence Erlbaum, Hillsdale, N.J., 1988.

Webster-Merriam Dictionary, 1923.

White, B., and Frederiksen, J., QUEST: Qualitative Understanding of Electrical System Troubleshooting. *Association for Computing Machinery Sigart Newletter*, July 1985.

White, B., and Frederiksen, J., Intelligent Tutoring Systems Based Upon Qualitative Model Evolutions. *Proceedings, AAAI-86, American Association for Artificial Intelligence.* 1986.

White, B., and Frederiksen, J., Causal Model Progressions as a Foundation for Intelligent Learning Environments. *Artificial Intelligence 42*(1), 1990.

White, B., and Horwitz, P., ThinkerTools: Enabling Students to Understand Physical Laws. *Proceedings, Ninth Annual Meeting of the Cognitive Science Society* (pp.336–347). Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1987.

Wight, S., Feurzeig, W., and Richards, J., Pluribus: A Visual Programming Environment for Educational Research. Proceedings, *IEEE Workshop on Language and Automation*, College Park, MD, The Computer Society, Institute for Electrical and Electronic Engineers, 1988.