COMPUTER-ASSISTED INSTRUCTION and INTELLIGENT TUTORING SYSTEMS

Shared Goals and Complementary Approaches

Edited by
JILL H. LARKIN
RUTH W. CHABAY

LEA

COMPUTER-ASSISTED INSTRUCTION and INTELLIGENT TUTORING SYSTEMS:

Shared Goals and Complementary Approaches

TECHNOLOGY IN EDUCATION SERIES

Edited by
Raymond S. Nickerson

Nickerson/Zodhiates • Technology in Education: Looking Toward 2020

Larkin/Chabay • Computer-Assisted Instruction and Intelligent Tutoring Systems: Shared Goals and Complementary Approaches

Bruce/Rubin • Electronic Quills: A Situated Evaluation of Using Computers for Writing in Classrooms

COMPUTER-ASSISTED INSTRUCTION and INTELLIGENT TUTORING SYSTEMS:

Shared Goals and Complementary Approaches

Edited by
JILL H. LARKIN
RUTH W. CHABAY
Carnegie Mellon University

LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS
1992 Hillsdale, New Jersey Hove and London

Carol Scheftic

Editor for Content and Style
during the crucial early period of the book's development

Lawrence Erlbaum Associates, Inc., Publishers
365 Broadway
Hillsdale, New Jersey 07462

Library of Congress Cataloging-in-Publication Data

Computer-assisted instruction and intelligent tutoring systems :
shared goals and complementary approaches / edited by J.H. Larkin, R.W. Chabay.
p. cm.
Includes bibliographical references and index.
ISBN 0–8058–0232–0.—ISBN 0–8058–0233–9 (pbk.)
1. Computer-assisted instruction. 2. Intelligent tutoring systems. I. Larkin, J. H. (Jill H.) II. Chabay, R. W. (Ruth W.)
LB1028.5.C5288 1991
371.3'34—dc20 91–15081
CIP

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Contents

Introduction 1
Jill H. Larkin and Ruth W. Chabay, Editors

The purpose of this book *1*
Complementary Approaches *1*
Shared Principles *6*
Value of Interaction *7*
Acknowledgements *7*
Summary of Chapters and Programs *9*

1. **The Design of Computer-Based Mathematics Instruction** 11
Sharon Dugdale

Introduction *11*
Example 1: Sort Equivalent Fractions *16*
Example 2: Darts *22*
Example 3: Green Globs *28*
Example 4: A Hypothetical Fractions Tutor *36*
Summary and Conclusion *42*

2. **From Syntax to Semantics in Foreign Language CAI** 47
Gerald R. Culley

Syntax: Generative Computer Assisted Instruction *48*
Feedback *49*
"Invisible" Review *50*
Efficiency of Development *50*
Limitations *52*
Semantics: Interaction with Meaning *54*
The Adventure Game Format *55*
Instructional Features *56*
How Does the Program "Understand" the Student? *58*
Recognition *59*
Spawning and Unspawning Entities *62*
Action *63*

3. LISP Intelligent Tutoring System: Research in Skill Acquisition 73
Albert T. Corbett and John R. Anderson

LISP and an Illustrative Interaction with LISPITS *74*
LISPITS: Assessment and Current Use *81*
Theoretical Principles Underlying LISPITS *82*
Model Tracing: Implementing LISPITS *88*
Skill Acquisition Research with LISPITS *93*
Model Tracing and Student-Controlled Feedback *99*
Conclusion *107*

4. Knowledge Representation and Explanation in GIL, An Intelligent Tutor for Programming 111
Brian J. Reiser, Daniel Y. Kimberg, Marsha C. Lovett, Michael Ranney

Problem Solving Knowledge in Intelligent Tutors *111*
GIL: A Programming Tutor That Constructs Its Own Explanations *122*
Conclusions *144*

5. A Practical Guide for the Creation of Educational Software 151
Ruth W. Chabay and Bruce A. Sherwood

Purpose *151*
A Unique Medium with Unique Design Challenges *152*
Displays *155*
Interactivity *170*
Input analysis *175*
User Control *179*
Hints for Development *183*
Conclusion *186*

6. Realizing the Revolution: A Brief Case Study **187**
George Brackett

Slide Shop *189*
Issue: Hardware Requirements *190*
Issue: Time, Effort ... and Money *192*
Issue: The Effects of Market Orientation *195*
Conclusion *197*

7. SHERLOCK: A Coached Practice Environment for an Electronics Troubleshooting Job **201**
Alan Lesgold, Susanne Lajoie, Marilyn Bunzo, Gary Eggan

SHERLOCK's Task Domain *203*
Principles Guiding SHERLOCK's Development *205*
Examples *212*
Implementation of the System *217*
Some Pedagogical Issues *235*

8. From PROUST to CHIRON: ITS Design as Iterative Engineering; Intermediate Results are Important! **239**
Warren Sack and Elliot Soloway

Introduction: Motivations and Goals *239*
Description of PROUST *241*
Can PROUST Teach What it's Supposed to Teach? *248*
Can PROUST Perform the Task it Teaches? *254*
CHIRON: Answers to Questions Raised by Proust *261*
Conclusions *270*

Author Index **277**

Program Index **278**

Subject Index **279**

Introduction

Jill H. Larkin and Ruth W. Chabay, Editors
Center for Design of Educational Computing, Carnegie Mellon University

THE PURPOSE OF THIS BOOK

Two groups of individuals share a vision that computers can provide excellent instruction for large numbers of students. The first of these groups we call developers of computer-assisted instruction (CAI). This group consists predominantly of experienced teachers and educational researchers, with strong backgrounds in the subject matter of their programs. The second group we call developers of intelligent tutoring systems (ITS). This group consists predominantly of researchers in cognitive psychology and computer science who develop principles of learning and apply them in instructional programs.

Unfortunately, these groups have had few vehicles for sharing ideas or programs. Different backgrounds and settings have meant reading different journals and attending different conferences. The purpose of this book is to foster a mutual understanding of shared issues and complementary approaches so as to further powerful educational applications of computing.

The following pages first summarize the complementary, but distinct, approaches of CAI and ITS, and then discuss shared issues toward which these complementary approaches are directed.

COMPLEMENTARY APPROACHES

Central Aims

The aim of CAI programs is to address existing needs of particular groups of students. The CAI developer wants to produce the program that has the best chance of teaching effectively and applies to this end all available experience and expertise. CAI programs are specific and hand-crafted for the domain, topic, and

students addressed. With these specifically tailored programs, CAI pushes the frontier of the best that can be done with current technology and imaginative techniques.

In contrast, the aim of ITS developers is to implement in programs a set of instructional principles sufficiently general to provide effective instruction for a variety of teaching tasks. ITS programs are strongly rooted in research on the psychology of learning. With large programs that provide instruction for many tasks (e.g., a significant part of a course), ITS pushes the frontier of knowledge about what general instructional techniques work and why.

Instructional Models

CAI programs do not follow a single theoretical model of instruction. In many CAI programs, the instruction emulates (in a form appropriate to the computer medium) interactions that might occur between a student and an excellent teacher. Other programs attempt to create an engaging, motivating environment that encourages purposeful exploration in a domain. A rich diversity of environments and problems is often a goal in CAI, and a suite of programs developed for a single course may vary significantly in goals, tasks, and style.

CAI programs reflect their developers' experienced beliefs about good teaching and good design for the computer medium. Some of the most interesting programs derive from developers' intuitions about activities well-suited to the medium, rather than from traditional instruction in the domain. Learning goals can be implicit in CAI, and activities may or may not be explicitly related to the tasks the student will be expected to perform after instruction.

ITS programs, in contrast, contain an explicit computer-implemented model of instruction. It is this model, and not the hand-crafted code of the developer, that determines how the program responds to the student. This model of instruction consists of two parts: (a) a *performance model* capable of performing the tasks the student is learning to perform, and (b) a *teaching model* that compares the student's actions with those of the performance model, and determines what (if any) reactions the ITS will provide to the student.

The performance model usually consists of a large set of fine-grained inference rules (although other techniques may be used in conjunction with these rules). Each rule consists of actions together with conditions under which these actions should occur. It is important that the rules be small in scope, because then they can be combined flexibly, in different ways, to do different activities. When an ITS is running, these rules are not invoked in any fixed order. Instead, at each point in time, the program searches for the rule most likely to contribute useful information to the current situation. This rule-based program architecture

has had repeated success in building computer programs that apply large amounts of knowledge flexibly to a variety of tasks.

The teaching model in an ITS consists of engaging the learner in a reasoning task, and comparing the student's actions with the set of actions that can be generated by the performance model. This comparison lets the ITS identify when the student does something either wrong (according to domain knowledge) or useless in pursuit of the current task. The teaching model then determines what (if any) advice is to be offered. Furthermore, the tasks given to the student are designed so that (a) all inference rules in the performance model are required to solve the full set of tasks, and (b) each inference rule is applied in several contexts. Thus, the student has multiple opportunities to practice and acquire the knowledge represented by each rule.

In summary, CAI programs reflect experience in teaching in a particular domain, and consist of varied activities designed to help students increase domain knowledge and apply that knowledge in different contexts. ITS programs have an explicit model of the knowledge required for a domain of tasks, and the activities they include provide systematic practice in using all relevant knowledge.

Program Structure

CAI programs are interaction centered, reflecting the CAI model of instruction in which the computer is an interactive medium for instruction characteristic of excellent teachers. CAI programs are therefore built to provide a specific kind of student interaction with the computer screen. These programs start with a vision of this interaction, and then programming is done to make that vision real. CAI developers see the computer screen as a programmable interactive communications medium.

Single CAI programs are usually relatively brief (although some can be used in many ways) and deal with only a few aspects of a domain (e.g., comparing the size of fractions). A set of related programs, carefully coordinated by their developer(s), can provide instruction throughout an entire course.

In contrast, *ITS programs are knowledge centered*, reflecting the explicit model of performance knowledge on which they are based. Interaction with a student consists of the model sending a message to the interface which in turn uses its own knowledge of graphics and layout to convey the information to the student, and to convey student input back to the central model.

ITS programs are typically comprehensive, with one program, in a single format, providing most or all of the instruction for a large fraction of the material in a course. The knowledge encoded for early lessons is re-used when appropriate for later lessons, much as the student is intended to use knowledge acquired earlier, along with new knowledge to address more complex tasks. An

ITS systematically teaches a body of knowledge by providing multiple and varying opportunities for the student to learn each inference rule in the performance model.

Necessary Experience

Because CAI is interaction centered and because it takes time to learn the sensitive use of a new and difficult medium (e.g., film and television as well as the computer), many of the best CAI programs are written by people with years of experience. The authors in this book, for example, have up to 20 years experience in the computer medium.

In particular, three of the CAI chapters reflect work which began in the PLATO[1] instructional computing environment originating at the University of Illinois. This environment provided a graphics screen with nearly twice the number of pixels as the standard Macintosh, a touch interface, and the ability to back-project microfiche—all available in the early 1970's! The size of the PLATO system allowed very large courses to require several hours of computer work each week. Thus new programs could accrue hundreds or thousands of hours of student use each semester.

In contrast, ITS is based on efforts to represent explicitly the knowledge humans use in performing a set of tasks. Building such programs is an extremely difficulty task, and stresses existing principles and techniques in both psychology and computer science. Therefore, developers of ITS are usually researchers in psychology or computer science, and have extensive experience in both fields. In particular, they draw on about 20 years of research on detailed processes of the human mind, and ways of characterizing these processes through computer-implemented models.

Computer Implementation

The aim of CAI is to provide practical instruction, consisting of interactive programs that teach effectively. Therefore CAI programs are developed under heavy computational and other constraints (eloquently described in the chapter by Brackett). They must respond rapidly (to support interaction and varied graphic interactions). Without graphics and fast interaction, no CAI program described in this book could exist in anything like its current form. Yet these programs must run on affordable and widely available machines (low end Macintoshes and PCs, and the educationally ubiquitous Apple II), machines with severe memory and speed limitations.

---

[1]PLATO® is a registered trademark of The Roach Organization, Inc. The PLATO system is a development of the University of Illinois.

CAI programs are generally algorithmic in structure (i.e., describable by a flow-chart) with user input often determining branching. Programs are organized around the screen display, and interactions with students center on the display, which changes in response to various inputs from the student. Algorithmic programs can be faster and simpler than those with more complex architectures. In CAI, the most-used languages are algorithmic languages, e.g., versions of BASIC, PASCAL, and assembly language, as well as descendents of TUTOR (the PLATO programming language).

In contrast, ITS programs exploit models of knowledge and teaching. Because the human mind is complex, computational models of its functioning are complex. Implementing them requires powerful machines and the most sophisticated techniques of computer science research.

ITS programs can not be algorithmic because humans have abilities far more varied than those of any algorithmic program. Instead ITS programs have a logic based on repeatedly applying knowledge (encoded as rules) to react to a current situation. Furthermore, content of this knowledge consists not of numbers, but of symbols (words, phrases, rules). ITS systems are therefore almost always written in languages that support processing of symbols and lists (e.g., LISP, Prolog) and their rule-based extensions (e.g., GRAPES, OPS5). ITS systems use extensively techniques of "artificial intelligence," that is techniques for building large programs that can incorporate large amounts of knowledge, and use it flexibly. For efficiency, ITS programs are sometimes written (or rewritten) in C, one of the most powerful modern algorithmic languages.

Implications

The differing aims, constraints, and instructional models imply the following substantial differences between CAI and ITS programs.

CAI programs are hand-crafted using deep knowledge of the domain, the students, and the computer medium. In CAI, the interface is intuitive, and using it is part of the instructional process. Good CAI has clarity and charm. Despite their algorithmic architecture, the clever hand-crafting of interactions gives these programs the feeling of freedom of interaction. Good CAI typically has been used for many thousands of hours by varying students, although quantitative studies of its effectiveness are rare.

In contrast ITS programs are the product of an applied science. They are based on, and explicitly include, psychological principles of learning. The instruction aims to provide systematically opportunities for learning all knowledge in a carefully defined domain. These programs are often meticulously tested with student groups, and their effectiveness can be dramatic (although there are few demonstrations of continued utility in widespread use).

SHARED PRINCIPLES

The chapters in this book, each describing the development and nature of one or more instructional programs, make concrete the contrasts discussed above. However, they also illustrate a rich set of largely shared design principles for good educational computing. The following paragraphs summarize these shared principles, which appear as continuous themes throughout the chapters in this book.

Engage the student in active work on central tasks

For example, Corbett & Anderson's principles of intelligent tutoring include providing instruction in the context of active problem solving. Chabay and Sherwood, summarizing their experience-based guidelines for CAI, give the mandate "Ask, don't tell," and go on to discuss techniques for designing programs that consistently engage the student in active work. This principle is reflected by every instructional program described in this book. Students do not passively read screens of text, or simply watch graphic simulations.

Use the interface to suggest appropriate reasoning

Displays and interfaces have long been a primary focus of CAI. In particular, Dugdale provides an excellent description of the intricate interaction between the domain being taught and the intricate displays that teach it. Only recently have ITS developers given significant attention to the difficult task of designing principled and effective graphic interactions. The chapters by Reiser et. al, and by Lesgold et. al., discuss this work.

Center work on appropriate reasoning

Most of the time the student should be working smoothly and successfully on important tasks. In Dugdale's *Green Globs*, for example, students are free to write whatever equations they like, with the aim of producing a graph that will pass through "globs" placed on the screen. Any engagement in the program means thinking about the relation between equations and their graphs. The actual graph that does appear is itself immediate feedback on whether the student's thinking about this relation was accurate or not. In the ITS world, relevant, successful, active thinking is achieved by tracing inference rules used by the student, and by intervening with a correction or hint if the student moves more than a step or two away from an appropriate reasoning sequence. This does not mean the student can solve a problem in just one "correct" way—the program's inference rules always attempt to support any correct solution, and often to provide strategic advice on incorrect reasoning.

Provide prompt feedback focusing attention on erroneous thinking

For example, Dugdale's elegant CAI programs are designed so that the interface gently prohibits many irrelevant actions, and the results of relevant actions are immediately apparent. All of the ITS programs are designed to track the student's reasoning and to give increasingly detailed feedback when the student does something wrong or irrelevant. In particular, the chapter by Reiser, et., al., discusses the use of knowledge in the ITS to formulate this feedback.

Adapt instruction to individual student knowledge

Corbett & Anderson describe how ITS programs can adapt through selecting the inference rules required by the task—a small set at first, more later on. In contrast the ITS *SHERLOCK* (Lesgold, et. al.) provides a single set of problems, but uses hints to suggest inferences the student finds difficult. Dugdale's CAI adapts to students' knowledge by letting students set their own tasks, within a cleverly designed environment where involvement in any task is likely to increase skill and ability to design and solve more complex problems. Many CAI programs (e.g., Culley's earlier grammar programs), provide the student with unlimited practice, together with information on success rates. Thus students with varying initial capabilities can readily use the program as a tool to build skill.

VALUE OF INTERACTION

Although the shared instructional principles discussed above are, of course, implemented quite differently in ITS and CAI programs, there are also the beginnings of joint implementations. For example, the chapter by Culley describes how a CAI developer began explicitly to need the techniques used in ITSs. Sack and Soloway, ITS developers, discuss both their needs for the evaluation techniques of educational research, and for a program structure that provides greater student interaction.

The purpose of this book is to acquaint both ITS and CAI developers with a broad range of approaches to their common vision of broadly effective educational computing.

ACKNOWLEDGEMENTS

We wish to acknowledge, with great appreciation, the central contributions of the following individuals and organizations:

The James S. McDonnell Foundation which provided funding for the conference, held in March, 1988, on which this book is based, and for preparation of the manuscript. John Bruer, President of the Foundation, provided

intellectual and moral support to complement the financial support of his organization. During initial planning for the book and conference, Jill Larkin was supported by a fellowship from the Guggenheim Foundation and by a National Science Foundation Visiting Professorship for Women.

Neither the conference nor the book would have been possible without the resources and working environment provided by Carnegie Mellon University and its Center for Design of Educational Computing. Additionally the Software Engineering Institute provided excellent facilities for a technologically complex conference.

Stanley Smith of the University of Illinois challenged and enlivened the workshop with his demonstration of innovative chemistry programs, which allow students to see and analyze real chemical phenomena using interactive videodisc technology.

Carol Scheftic served ably as the central contact with authors and as editor during the first difficult phases of manuscript preparation. Carol Scheftic and Ruth Chabay also took full responsibility for the conference leading to this book, while Jill Larkin managed to attend the sessions despite pneumonia.

Alice Huston patiently and meticulously read all of the ITS chapters, giving invaluable suggestions for making these chapters readable to an audience outside of the computer science community.

Finally, Mary McCallum provided patient expertise in proofreading, correcting endless versions of manuscripts, tracking down detailed information, and transferring materials (physically and electronically). She cheerfully took on endless extra tasks both contributing directly to this book and giving time for the editors to edit — a task more difficult than either of us had imagined.

Jill H. Larkin and Ruth W. Chabay
Pittsburgh, October, 1991

SUMMARY OF CHAPTERS AND PROGRAMS

Ch. No.	*Author*	*Program Type*	*Age Level*	*Subject matter*
1.	Dugdale	CAI	Pre College	Mathematics
2.	Culley	CAI/ITS[2]	University	Latin
3.	Corbett & Anderson	ITS	University	Programming in LISP
4.	Reiser et al.	ITS	University	Programming in LISP
5.	Chabay & Sherwood	CAI	All Levels	Design issues in CAI
6.	Brackett	CAI	Pre College	Slide presenter & scientific reasoning
7.	Lesgold et al.	ITS	Military	Electronic device troubleshooting
8.	Sack & Soloway	ITS[2]	University	Programming in PASCAL

[2]These two chapters describe programs which are not "tutors", but which use techniques of artificial intelligence.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

1 The Design of Computer-Based Mathematics Instruction

Sharon Dugdale
Division of Education, University of California, Davis

INTRODUCTION

Overview

This chapter illustrates, by example, some design goals and instructional techniques devised and used during a fifteen-year software development and implementation effort for mathematics instruction.

The example instructional programs illustrate a variety of styles, sharing a common goal of encouraging students to become active, creative, independent learners. To this end, the programs minimize rote repetition of algorithms and procedures in favor of more attention to analyzing problems and developing strategies.

Three programs are presented in some detail with discussions of their features, goals, and classroom dynamics. A hypothetical fourth example is used to extend the instructional perspective to the teaching of more routine skills and to suggest considerations for defining appropriate roles for computer-implemented tutors. In conclusion, the chapter contrasts three styles of feedback in response to student errors and offers suggestions for the design of instructional materials in general.

An Introductory Remark

This chapter is about mathematics in particular, due to reservations about discussing instructional design independent of a subject matter context. Imagine entering the kitchen of a good restaurant and asking the chef, "What food preparation method do you use?" The chef would likely respond with a blank

stare—not out of incompetence, but simply because the choice of a food preparation method is so obviously dependent on what particular food is being prepared, and for what audience. Similarly, in preparing knowledge for consumption, it is important to take a careful look at what is to be served, understand its inherent qualities and characteristics, and then select or devise a presentation method which best brings out its unique flavor and character.

This is not to say that a basic knowledge of general instructional design is unnecessary, any more than a basic knowledge of cooking is unnecessary for the good chef. On the contrary, a knowledge of a *variety* of methods seems useful, and the basic sense of where to use (and not use) these various methods is at least as important as the methods themselves. The sense of where to apply what methods is probably rooted more in an understanding of their function than in a strict recall of their form, and the designer who is led by a more functional approach may be better able to set aside the recipe box and do some inventive things.

Of course we can point to examples of educational software that could be improved by the strict application of generic design rules—just as we sometimes encounter food that is better doused with ketchup. This chapter is not expected to remedy such fundamental deficiencies, nor is it intended to address efficient production of routine materials. (Though there is undeniably much to be said for the consistent quality, reliability, and economy of a good fast food operation.) The purpose here is, rather, to explore issues beyond those addressed in the basic recipe books. This purpose requires a specific content, although it is hoped that the examples and discussions presented will provide insights applicable beyond the content covered here.

Background

The Fractions Curriculum

The first two examples represented here, *Sort Equivalent Fractions* and *Darts*, are from the *Fractions Curriculum* of the PLATO® Elementary Mathematics Project[1] (Dugdale and Kibbey, 1980). Developed for use on the PLATO network (Smith and Sherwood, 1976; Lyman, 1984), this curriculum covers common fractions, mixed numbers, and an introduction to decimal notation. It includes approximately 100 individual computer programs, a teacher's manual, and booklets and worksheets for students.

The management system allows a teacher to assign topics and individual activities. Based on the teacher's assignments and on individual student

[1] PLATO is a registered trademark of The Roach Organization, Inc. The PLATO system is a development of the University of Illinois.

background information (either stored by the programs during interaction or entered by the teacher), the management system constructs personalized student sessions. Although many session structures are possible, a typical student session might include:

1. a few minutes of review from previous material,
2. 15-20 minutes of work from a menu of currently assigned topics,
3. 5-10 minutes of work from an expanded menu which includes several general experience programs in addition to the currently assigned topics. The general experience programs are typically games and exploratory activities which provide practice with previous material, expansion of current topics, or initial exploration of new areas.

The management system provides an extensive on-line data reporting facility with which teachers and students can follow individual progress.

The curriculum offers students extensive experience using fractions and mixed numbers to manipulate interactive models, establishing a variety of representations for rational numbers and links between those representations. The models usually allow many different ways of solving problems. This provides opportunities for exploration, helps students attend to the meaning of what they are doing, and helps avoid the notion that mathematics is simply the repetition of dictated algorithms.

The curriculum provides a mix of instruction, review, practice, experience, and exploration. In many of the programs, the difficulty and complexity of the task is automatically adjusted (up or down) based on the student's performance. Completion of these programs depends on reaching a "mastery level," that is, demonstrating proficiency with problems of a complexity deemed necessary for use in succeeding materials. The mastery level number and the student's current level number are continuously displayed to keep the student informed about progress through the program.

As a pioneering effort in the use of computer networks to encourage sharing of ideas among students, the curriculum includes:

1. inter-terminal programs which allow students to interact with each other in real time from widely separated terminals, and
2. "libraries" which allow students to store their work and see products generated by other students who are working not only at other places but at other times.

Through these network facilities students share their work and ideas not only with their own classmates, but also with students in other classes and other schools (Dugdale, 1979).

Beginning in 1973, the fractions curriculum was developed in conjunction with use by students in public school classrooms. This permitted the authors to tailor the materials to the needs of students and teachers in realistic educational settings. During the 1974-75 and 1975-76 school years the curriculum was pilot tested as a regular part of the mathematics program in about a dozen classrooms, serving a total of approximately 300 students each year. The curriculum reached essentially finished form during the second of those years.

A summative evaluation done by Educational Testing Service proved these materials highly effective in both achievement and attitude improvement (Slottow, 1977; Swinton 1978). Further work and adaptations were done in conjunction with special needs programs, including work with hearing impaired students and those with other physical, emotional, or learning problems (Dugdale and Vogel, 1978).

This curriculum and related materials produced by other development teams (Seiler and Weaver, 1976; Cohen and Glynn, 1974) continue to be a regular part of the daily mathematics program for students in approximately 20 local classrooms. The materials were designed for use by students in grades 4-6 in a classroom situation where they could be integrated with the ongoing mathematics instruction on a daily basis. However, most of the activities have found broad applicability in adult education programs and numerous other settings served by the PLATO/NovaNET^{SM} computer network[2]. Because it was developed to take full advantage of the network capabilities, touch-sensitive display, and other features of the PLATO system, the fractions curriculum as a whole has not been transported to microcomputers more common in elementary school classrooms.

Fortunately, the theoretical results of the development effort have transferred more readily to other systems. The following design principles were distilled from analysis of the style and techniques used in many of the programs. Programs displaying these characteristics were labelled *intrinsic models*.

1. The student is given a working model to explore and manipulate.
2. The mathematics to be learned is intrinsic to the model. In other words, the model is a direct expression of the underlying mathematics.
3. Feedback is direct, relevant, diagnostic, and often graphic, so that students can tell at a glance what the error was and how it relates to a correct solution. Unnecessary verbiage and gimmicks unrelated to the mathematics are avoided.

---

[2] NovaNET has evolved from the earlier PLATO system. NovaNET is a service mark of University Communications, Inc.

4. The model provides a rich environment for exploration by students of widely varying mathematical background and ability, but students find that the more mathematics they apply, the more they are able to do.

In short, intrinsic models provide students a rich, mathematically accurate environment and the motivation to manipulate and learn from that environment. Models of this sort can engage students in formulating and using mathematical strategies in ways that were not possible before computers entered the classroom (Dugdale, 1982). Two of the programs presented in this chapter, *Darts* (from the Fractions Curriculum) and *Green Globs* (from a later development effort), are examples of intrinsic models.

Materials on Graphing and Functional Relationships

A later project applied the design principles outlined above to create software to improve students' understanding of functional relationships—concepts fundamental to further work in mathematics, science, and engineering, and an area to which the calculational and display capabilities of the microcomputer proved especially well suited (Dugdale and Kibbey, 1984). This project was one of the first efforts to use microcomputers to involve students in the exploration of graphic representations of functions, including qualitative, as well as quantitative, aspects of graphs. *Green Globs*, the third example presented here, is taken from this development effort.

Initial development of the graphing materials was done on the Color MicroPLATO system (Stifle, et al., 1979). The materials have since been revised and expanded, and derivative works have been created for IBM-PC and Apple II series microcomputers (Dugdale and Kibbey, 1983a, 1983b, 1986a, 1986b).

The materials were developed in conjunction with student use in two public high schools. Students generally worked in pairs or groups of three, actively discussing what they were doing and sharing ideas. Some students, however, became deeply engrossed in working out complicated equations and seemed more intensely involved when working alone. Observation indicated that each of these modes of use (individual and small group) provided different important learning and exploration opportunities.

Evaluation was largely formative, involving on-site observation of students and interviews with teachers and students. The materials underwent several cycles of testing and revision. In an informal two-week trial with the finished materials, students showed gains in performance on test items concerning linear and quadratic graphs (Dugdale and Kibbey, 1983).

Later efforts have extended the work with functional relationships, developing various approaches to understanding algebraic concepts and physical phenomena through functions and their graphs (Dugdale, 1986-87).

EXAMPLE 1: SORT EQUIVALENT FRACTIONS

Description

As shown in Figure 1, this is a practice program in which students sort fractions into equivalence sets. The student is given a collection of fractions and some loops to sort them into by size. A fraction can be moved into a loop or moved from one loop to another by touching the fraction and then touching the desired location on the screen.

After all of the fractions have been placed into the loops and the student has indicated that he or she is finished, the computer checks the equivalence of the fractions within each loop and responds by telling the student how many fractions are not in the correct loop. The student is asked to correct any wrong placements and may move the fractions around the screen until the problem has been solved correctly.

The program adjusts to student performance, increasing or decreasing the difficulty of discriminations, the number of fractions to sort, and the number of loops to sort them into, based on student proficiency in sorting the fractions. Completion of the exercise requires passing a specified difficulty level. The current level is shown at the bottom of the screen to keep the student informed of progress through the activity.

Discussion

This example is included here because the content objectives *appear* simple and obvious—to identify equivalent fractions and group them into equivalence sets. The exercise might be appropriately classified as "drill and practice," and the surface objectives could be met by various other computer or paper-and-pencil activities. However, upon working with the program or observing students using it, it becomes apparent that beyond this surface agenda are several unobtrusive features that foster a richer experience for the student.

First, the feedback for student errors simply tells how many fractions are not in the correct loop. Of course, the computer could just as easily tell which particular fractions are misplaced and which loops they belong in, leaving nothing for the student to figure out. The issue of how much and what kind of feedback to provide for student errors is an important one. Here, the limited feedback leaves students to diagnose their own errors. Having just finished scrutinizing each fraction to place it into a loop, few students are eager to methodically check each fraction again in order to find one or two errors. The common tendency is to scan the page and ask, "What is an easy way to spot a misfit?" Observation of students indicates that this diagnostic step is important in encouraging students to step back and look at the problem in a less

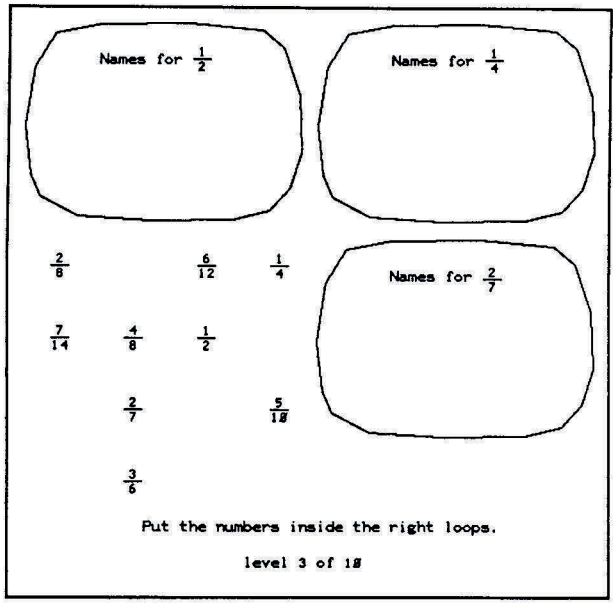

Figure 1a: Initial display from *Sort Equivalent Fractions*.

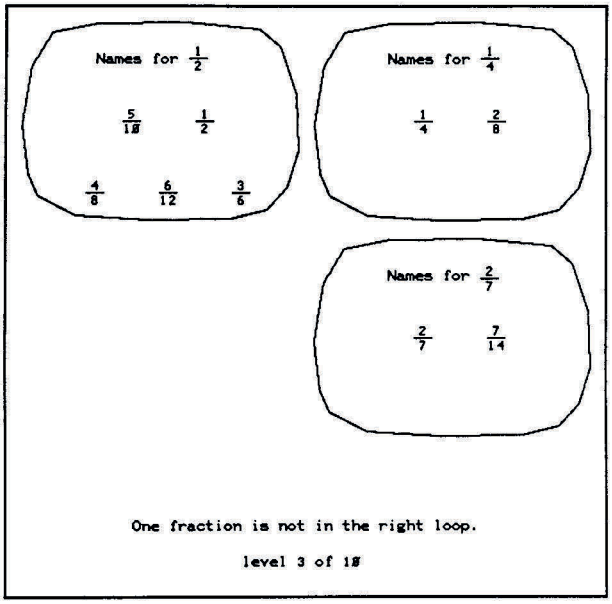

Figure 1b: *Sort Equivalent Fractions*. The student has sorted the fractions into the loops, and the computer has responded that one fraction is misplaced.

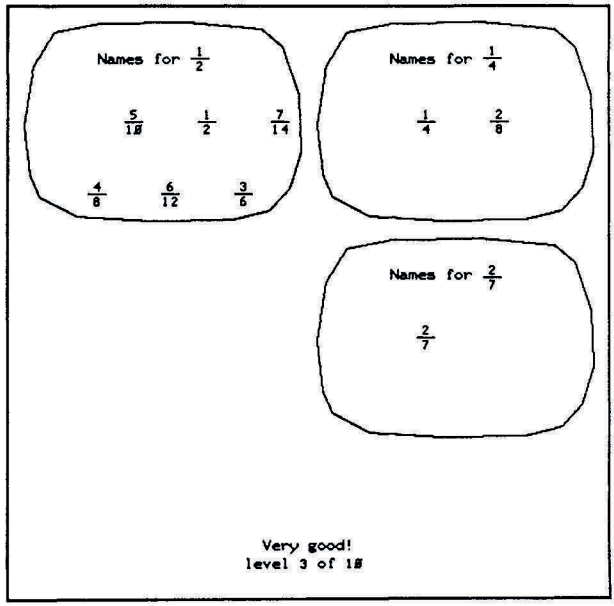

Figure 1c: *Sort Equivalent Fractions.* The student has moved the misplaced fraction to the correct loop.

mechanical way. It can foster multiple perspectives on the problem, encouraging a habit of parallel checking for reasonableness of answers.

Coupled with this incentive to take an overall view of the problem to spot errors, the program provides a strategic choice of items. In the early levels, students sometimes encounter zero in the numerator. Few students find it necessary to apply an algorithmic approach to many zero-numerator fractions before concluding that an algorithm is unnecessary for sorting the zeros into their loop. Some students (and reviewers, too) may chuckle at the lack of sophistication of a program that would know no better than to include such a trivial problem. But this apparent lack of sophistication has planted an important seed—scanning the problem for shortcuts can sometimes avoid a lot of tedious work!

Once a student begins taking an overview of the problem and looking for shortcuts, various strategies can be noticed. For example, in Figure 2 there is only one loop named by a fraction greater than one (i.e., the numerator is greater than the denominator). Since every fraction must go into a loop, no algorithmic checking is necessary to know that all of the fractions greater than one go into

the $\frac{3}{2}$ loop. Although this is hardly an earthshaking mathematical revelation in itself, it can serve as a step toward more interesting ideas.

For example, factors and multiples become useful, too. If the denominator of a fraction is not divisible by 7, it does not belong in the $\frac{2}{7}$ loop. But anyone assuming that all fractions with denominators divisible by 7 necessarily do belong in the $\frac{2}{7}$ loop is in for a thought-provoking surprise. (This may have been the error in Figure 1b.) The intent is not to avoid the possibility of errors and misrules, but to treat them as conjectures to be tested and revised. Strategies are developed and refined with experience, and the insights gained in the process can be useful beyond the task at hand.

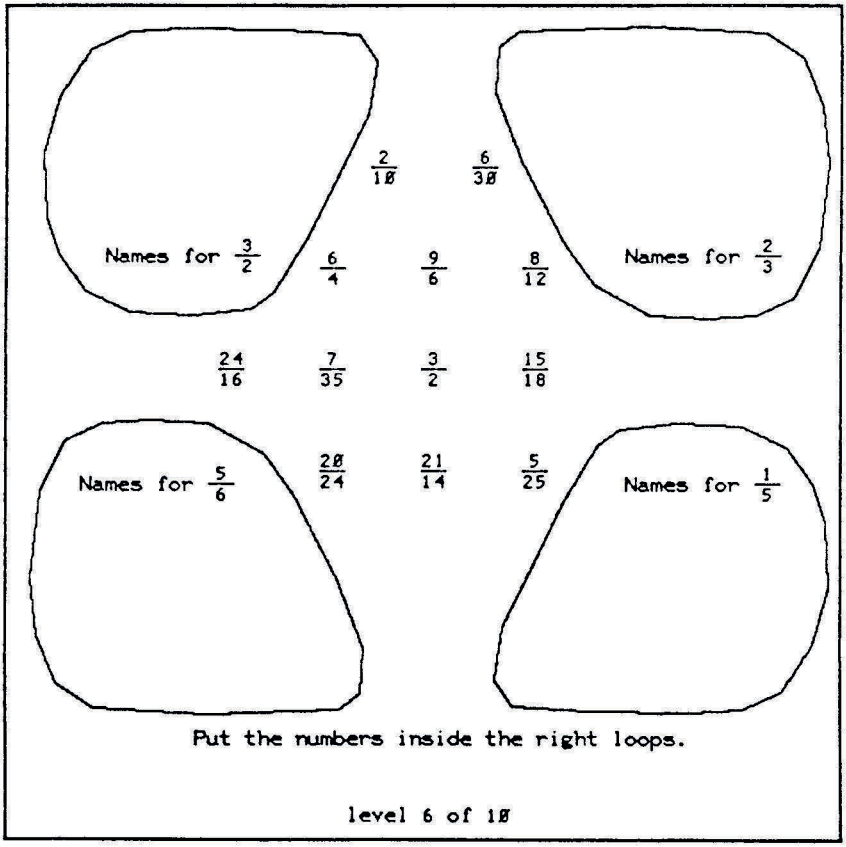

Figure 2: Sorting fractions into loops: just one loop has a value greater than 1.

Of course, different students notice different things. One student looking at the display in Figure 2 might begin by placing all of the fractions greater than 1 into the $\frac{3}{2}$ loop, while another might scan for multiples of numerators and denominators, and still another might notice that any fraction with the denominator more than twice the numerator must go into the $\frac{1}{5}$ loop (the only loop for fractions less than $\frac{1}{2}$). Students can make substantial progress on the problem using any combination of these strategies, but even taken together, these techniques are not sufficient to finish the problem. Something further is needed to decide whether $\frac{20}{24}$ belongs in the $\frac{5}{6}$ loop or the $\frac{2}{3}$ loop.

Students are generally eager to share their insights and discoveries, and students who interact with each other as well as with the computer are likely to learn more strategies than those working in isolation. A critical breakthrough for a student might be simply a classmate's offhand remark, "$\frac{15}{18}$ doesn't belong in the $\frac{2}{3}$ loop, because 15 isn't an even number." Teachers can help by encouraging students to share their ideas.

As Whitney (1986) points out,

> A primary aim of studying mathematics must be to grow in your own natural reasoning powers, especially in domains where precise reasoning is valid. The growth must include creative and critical thinking, increasing control over one's own work, seeing connections with related matters, and raising communication skills.

Whitney then contrasts this aim with what usually happens in the classroom:

> In the usual school mathematics class, the student's aim is to try to remember particular patterns of thought coming from reasoning, but to disregard the reasoning from which they came. Thus the essence of the process is lost.

Indeed, it is not uncommon to see students engaged in tasks in which it is easy to bypass any mathematical thinking by using the available non-mathematical cues. For example, when a textbook introduces an algorithm for multiplying fractions, follows it with a page of practice, then concludes the section with a set of "applications" (i.e., word problems), students do not need to make sense of the word problems to know that the task is to pick out the two fractions in each problem and apply the algorithm.

There are some subtle but critical differences between this sort of mathematics bypass and the following things that happen in using *Sort Equivalent Fractions.*