Mathematics Software[†]

Patrick W. Thompson
Illinois State University

Barbara Bowen has asked us to discuss examples of decisions we have made that address a number of issues. These issues (e.g., use of multiple representations, "hooks" into the software, etc.) are important approaches to consider in software design. Before speaking specifically to items in her list, however, I will speak to the themes amplified by them.

Also, I am sure that most people attending this conference will have made design decisions that strike a balance among these issues: too strong an adherence to any one can preclude adherence to one or more others. I will discuss (my) self-imposed constraints which shape any balance achieved among these issues.

Self-imposed Constraints

First, the programs I design are meant to be used as *pedagogical* tools. This means that I always assume that a teacher is in control of instruction, and the programs he or she uses will be used as tools to achieve some set of learning objectives.[1] Sometimes a program will be used by a teacher as a device to instigate discussions; sometimes a program will be used by a teacher to establish a set of constraints within which students are to solve problems. Nevertheless, a program being used is *used*. It does nothing on its own unless directed by a person, and it is the person who chooses the problem (pedagogical, mathematical, or both) a tool is used to solve.

Second, these programs are meant to assist in students' achievement of *cognitive* objectives in some conceptual field in mathematics. This means that a lot of cognitive analysis needs to be done beforehand, and even more needs to be done during the design-implement-redesign cycle. It also means that these programs are not "generic," in the way a word processor is a generic tool for composing prose.

Third, these programs are meant to enhance the subjective experience of the student in mathematically substantial ways. This means that mathematics is not meant to be "shown." It is meant to be "felt." The aim is for students to construct mathematical knowledge through interactions with software, in the context of problems defined within the metaphor of the software, but which are defined independently of it (e.g., through problem sheets or discussions). As such, the mathematics "shown" by a program is both more and less than the mathematics that is intended

[†] Research reported in this paper is supported by NSF Grant No. MDR 87-51381 and by a grant of equipment by Apple Computer. Any opinions expressed or implied are those of the author and do not necesarilly represent positions of the NSF or Apple Computer.

[1] I do not mean by this that I assume the teacher is always teaching. Rather, I assume only that a teacher will make all substantive instructional decisions. The software might constrain those decisions, but it does not dictate them.

to be "seen" by students. There is more foundational mathematics shown these programs than is seen by students, but there is less mathematics availed by them than is intended to be constructed by students. It is more difficult than it sounds to keep from availing too much mathematics to students in a program designed to assist in the teaching of mathematical concepts and skills.

Fourth, I try to design programs that are transparent to students using them. The reason for this constraint is simple: I want students to be students of mathematics, not students of some program. Thus, these programs are meant to provide intuition-building experiences that students can formalize independently of the programs. That is, computer programs of this genre are meant to be transitional devices that assist students in constructing mathematical systems meaningfully. The goal is for students to interiorize a program by constructing for themselves the mathematical system programmed into it. But I intend that students will eventually no longer need the program to give meaning to the mathematics.

Imply and Support the User's Full Intelligence

If we are to take this theme seriously, then it must be incorporated into programs in a deeply principled way. Above all, this theme implies that we assume students can and should *reason.* In mathematics, this means that we attempt to make contextual and mathematical constraints explicit, but at the same time it means that we do not attempt to enforce prescribed ways to solve problems within these constraints.

To take this theme seriously and at the same time wear the cap of pedagogue is difficult. It means that software must be designed so that its use assists students in constructing schemes for systematic reasoning and, just as importantly, assists students in constructing the objects toward which reasoning is to be directed. It also means that we need to isolate *cognitive* roots of mathematical concepts, which is quite different from isolating *mathematical* roots of these concepts.[2]

Coordination of multiple representational systems

It is now evident that mathematical competence rests upon one's ability to coordinate multiple, complementary representational systems (Janvier, 1987; Kaput, 1986, this volume). I have used four approaches to the inclusion of multiple, linked representations. In one there is a visual model of some mathematical domain and a set of commands that make things happen within that model. That is, the mathematical meaning of the formalism is defined within the visual model. Problems framed within the metaphor of the visual model are posed; students must solve the problem by casting their solutions or explorations within the command language, which typically

[2] For example, David Tall differentiates between the concept of "local straightness" as a *cognitive* root of derivatives and the concept of limit as a *mathematical* root of derivatives.

resembles traditional mathematical notation (Thompson, 1985a, 1985b, 1987a; Thompson & Dreyfus, 1988). A second approach is to have two representational systems, either of which can be active. Problems are posed to students within the framework of one of the systems; they must achieve a solution by acting within the framework of the other (Thompson, 1987b, 1989c). A third approach is to have three representational systems, where one is completely traditional, one is graphic, and the third is a trans-system set of actions that have consequences in the other two systems. Students act within the trans-system set of actions and apply those actions to the graphic representational system (Thompson & Thompson, 1987; Thompson, 1989a). The fourth approach is to have two systems, one of which is completely traditional, the other graphic, and the student acts completely within the graphic system. The traditional system (e.g., arithmetic or algebraic expressions) corresponds to consequences of actions within the graphic system (Thompson, 1989a, 1989b).

In all four approaches, a program is meant to set a *context* for students' problem solving and for teachers to initiate discussions of substantial cognitive issues. Curriculum is kept independent of software, but programs are designed to support a cognitive curriculum (Thompson, 1985a, 1989c).

An Example

Base-ten numeration is one of the most foundational mathematical systems students will learn in school. Every numerical algorithm is founded ultimately upon some system for representing numbers, and concrete aggregates beyond those we can perceive directly may be accessed only through some numeration system. It is also evident that too many students understand our numeration system only to the extent that they can "read" numerals (NAEP, __; McKnight et al., 1987). They need to understand that a numeration system constitutes a set of constraints within which methods for naming the result of an operation are invented.

I will draw on one program and its associated pedagogy to make concrete the idea of making contextual and mathematical constraints explicit while leaving it up to students to construct methods for solving problems within those constraints. In this program, called BLOCKS MICROWORLD (or BLOCKS, for short), the major constraint is that students can represent quantities only within a base numeration system (in this example, the base is ten). The topic of this example is decimal numeration, and the operation is division.

Sue, the student from whose work these illustrations are taken, was a fourth-grader. Prior to working with division of decimal numbers, Sue participated in instruction on decimal numeration and addition and subtraction of decimal numbers (Thompson, 1989d), and instruction on division of whole numbers. Sue was reminded continually that she was free to solve problems any way she wished; her main task was to construct a method by which she could record the state of a problem

each time she acted on BLOCKS' display. This approach was suggested by Vergnaud (1982). The intent was that students first develop "theorems in action" (i.e., intuitive schemes) and then transform them into "theorems in thought" (i.e., students formalize their schemes by organizing and representing them systematically).

In this example, Sue was asked to name the result of dividing 93 by 8. She created the display in Figure 1 by dragging copies of blocks (9 tens and 3 ones) into the "block" area. She then used a menu to have BLOCKS display 8 containers. That is, Sue translated "What is 93 ÷ 8?" into the problem of sharing 93 ones among 8 containers so that each container receives the same number.

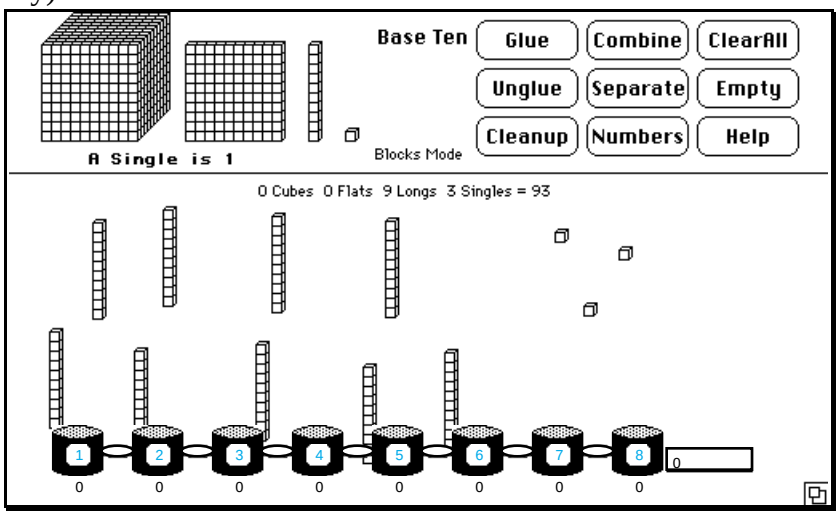

Figure 1

Sue's method of sharing was to try to share blocks of the smallest value first. Here, her method immediately created a problem: how to share 3 ones among 8 containers, where "one" is represented by a single block.[3] She decided to change her representation of one, so that a "long" block represented one (see Figure 2). Were she simply to have changed the unit, her display would have represented 9.3 instead of 93. Sue needed to decide to chose a menu item that *both* increased the unit and increased the size of each block.

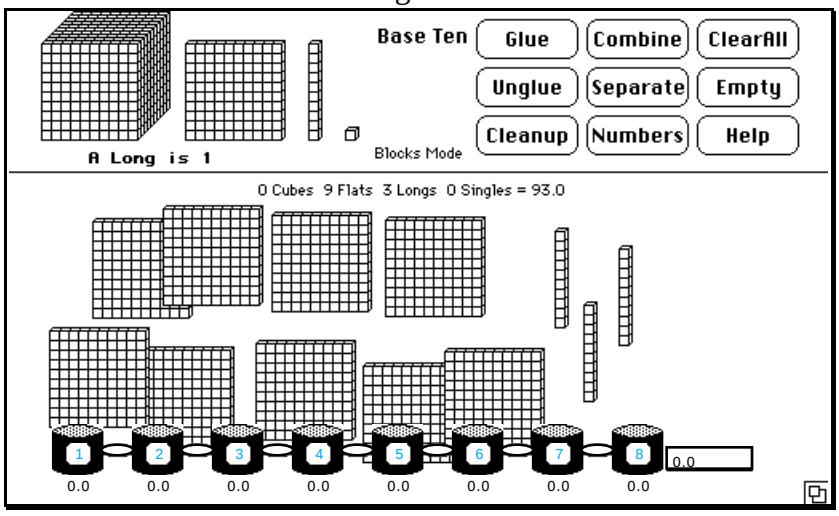

Figure 2

[3] The program refused to put any singles into containers because the containers were linked. The links reflected the constraint that if any container received blocks, then every container must receive the same number of the same kind of block.

Sue's next step was to break apart the ones (into tenths) and share as many tenths among containers as she could. To execute this step, she selected the three ones (by drawing a selection rectangle) and then clicked the **Unglue** button, which caused each one to break apart into tenths (Figure 3). Sue then dragged 3 groups of 8 tenths each to the containers, ending up with Figure 4.

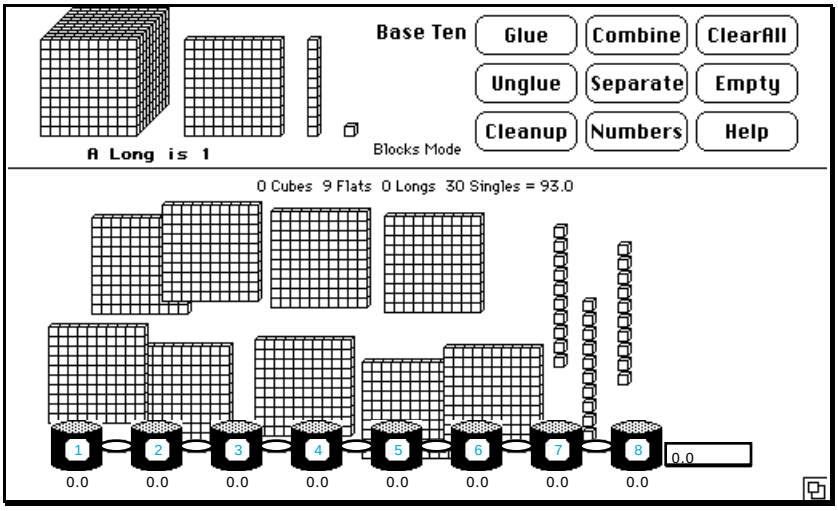

Figure 3

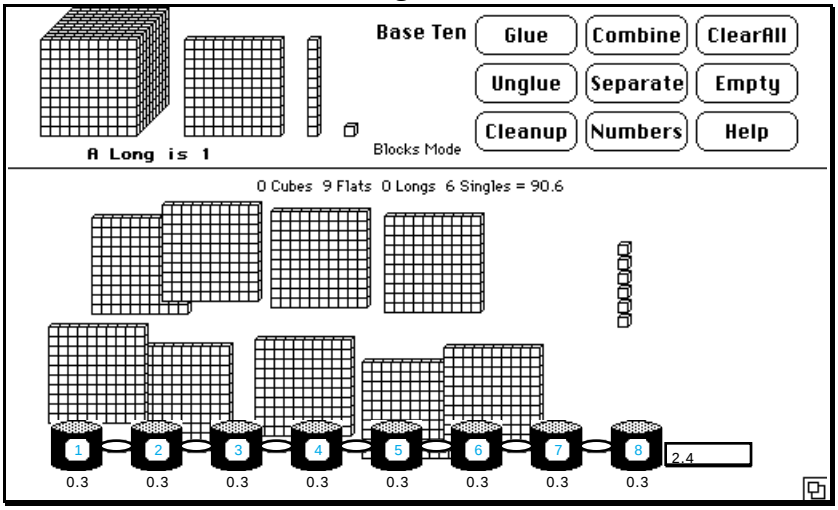

Figure 4

In the remainder of her solution, Sue continued her method of working first with the smallest block numerous enough to be shared, and increased the unit each time she needed to break apart one or more singles.

After having solved many division problems with BLOCKS, Sue was given the additional task of devising a scheme by which to record steps in her solutions. Her scheme, applied to 93 ÷ 8 = __, is shown in Figure 5. With her notational scheme Sue could solve division problems without using BLOCKS while at the same time referring to her experiences with it. Sue's use

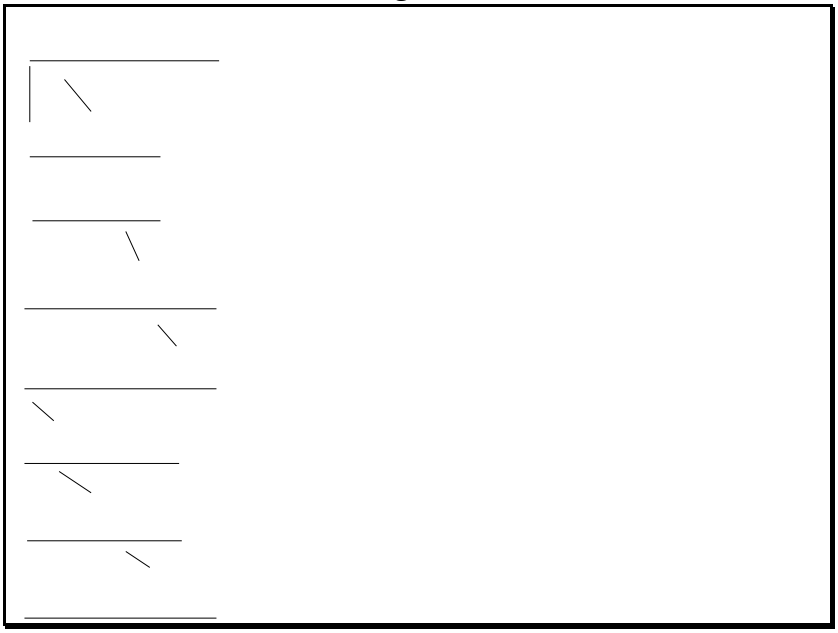

Figure 5

of BLOCKS established a semantic base—a mental model—that she could "think with" as she solved division problems symbolically.

Conclusion

One could ask, legitimately, these two questions: (1) *Why use a computer to simulate Dienes' blocks when students could use the real things?* and (2) *Why bother to use computers to teach anachronisms (computational algorithms)?* First, the constraints built into BLOCKS are different from those built into physical Dienes' blocks, while at the same time some physical constraints are relaxed. These differences make for substantially richer experiences when students use BLOCKS in the solution of problems, and these experiences translate into richer meanings being injected into written symbols (Thompson, 1989d). Second, the "reform" movements, which tell us to toss out anything having to do with paper-and-pencil calculations, do not distinguish between two aspects of paper-and-pencil uses. One aspect, and the one that *should* be tossed out, is the ritualistic performance of prescribed procedures, where neither written marks nor actions on them have any meaning for the children "learning" them. Another, the aspect illustrated in Sue's work, is the use of notation to represent a concrete system *and one's actions on it*. This latter aspect is at the heart of intelligent uses of arithmetic and algebra, and is foundational to mathematical modeling. The construction of elementary representational systems and algorithms within them is part and parcel of an approach to mathematics education called *algorithmics* (Hatfield, in press). Through intelligent uses of computers we can combine the spirit of algorithmics with instruction on important conventional representational systems, such as base-ten numeration.

References

Janvier, C. (1987). (Ed.) *Representational systems.* Hillsdale, NJ: Erlbaum.

Kaput, J. (1986). Information technology and mathematics: Opening new representational windows. *Journal of Mathematical Behavior, 5*, 187-207.

McKnight, C., Crosswhite, J., Dossey, J., Kifer, L., Swafford, J., Travers, K., & Cooney, T. (1987). *The underachieving curriculum: Assessing U.S. school mathematics from an international perspective.* Urbana, IL: Stipes.

Thompson, P. W., & Dreyfus, T. (1988). Integers as transformations. *Journal for Research in Mathematics Education, 19,* 115-133.

Thompson, P. W., & Thompson, A. G. (1987). Computer presentations of structure in algebra. In J. C. Bergeron, N. Herscovics, & C. Kieran (Eds.), *Proceedings of the Eleventh Annual Meeting of the International Group for the Psychology of Mathematics Education.*

Thompson, P. W. (1985a). Experience, problem solving, and learning mathematics: Considerations in developing mathematics curricula. In E. A. Silver (Ed.), *Learning and teaching mathematical problem solving: Multiple research perspectives* (pp. 189-236). Hillsdale, NJ: Erlbaum.

Thompson, P. W. (1985b). A Piagetian approach to transformation geometry via microworlds. *Mathematics Teacher*, *78* (6), 465-472.

Thompson, P. W. (1987a). Mathematical microworlds and intelligent computer-assisted instruction. In G. Kearsley, (Ed.), *Artificial intelligence and instruction.* Addison-Wesley.

Thompson, P. W. (1987b). Direct-engagment software for learning and teaching mathematics.

Thompson, P. W. (1989a). Artificial intelligence, advanced technology, and learning and teaching algebra. In C. Kieren & S. Wagner (Eds.) *Research issues in the learning and teaching of algebra* (pp. 135-161). Reston, VA: National Council of Teachers of Mathematics.

Thompson, P. W. (1989b, March). *A cognitive model of quantity-based algebraic reasoning.* Paper presented at the Annual Meeting of the American Educational Research Association, San Francisco.

Thompson, P. W. (1989c, April). *Notes on technology and curriculum reform.* Working paper distributed at the symposium *New Technology's Challenges to Curriculum, Pedagogy, and Evaluation*, Research Presession to the Annual Meeting of the National Council of Teachers of Mathematics, Orlando.

Thompson, P. W. (1989d). *Cognitive effects of multiple representations of mathematical concepts.* Final project report of NSF Grant No. MDR 87-51381.