COURSEWARE DESIGN IN THE TOOLKIT AGE

Jesse M. Heines, Ed.D.

University of Massachusetts Lowell
Associate Professor
Dept. of Computer Science
Lowell, Massachusetts 01854

Telephone: (978) 934-3634
Internet E-Mail: heines@cs.uml.edu

presented at the

The 33rd International Conference of the

Association for the Development of Computer-Based Instructional Systems

Marriott Pavilion Hotel
St. Louis, Missouri

November 10-14, 1991

Abstract

The extensive capabilities of today's systems have made the development of state-of-the-art courseware a formidable task. To make these sophisticated capabilities accessible to people without extensive programming backgrounds — and to significantly speed up program development even for people with such backgrounds — rich software toolkits have been developed. The Macintosh toolkit has been around for several years, but Windows 3.0 for DOS systems has only recently come onto the market. In the workstation arena, we are even seeing high level toolkits become available that are built on lower level toolkits, as Motif is built on Xwindows.

The paper discusses how five standard toolkit design elements can be applied to computer-assisted instruction applications: pulldown menus, popup windows, scrollable windows, dialog boxes, and help tools. The author strongly encourages the use of these elements to standardize user interfaces, whether through programming languages or menu-driven authoring systems. He feels that doing so will allow courseware to obtain the standard "look and feel" of other applications and thus help students to concentrate on the subject matter they are trying to learn rather than the mechanics of running the course.

Keywords:

- computer-assisted instruction
- user interfaces
- screen design
- courseware design and development
- software toolkits

Issues in Courseware Design

In 1983 I authored a small book entitled Screen Design Strategies for Computer-Assisted Instruction. One of the major issues that book tried to address was to provide a basis for response to James Martin's 1973 lament:

> As yet, no acknowledged sense of style has developed for CAI. ... In the meantime, however, some singularly unstylish CAI programs are being written (Martin, 1973).

Style remains one of the major issues in courseware design to the present day. While human factors research has helped us gain insight into the best ways to use design components such as functional areas, icons, and menus, the capabilities of today's systems have expanded so quickly that researchers have been hard-pressed to keep up with the myriad of developments. The net result is that even as our knowledge grows about what constitutes good courseware design today, we are continually asked to reevaluate our knowledge in new contexts. There are far more choices for courseware designs today than there were when I wrote my book in 1983, and the problem is orders of magnitude more complex than when Martin found it so lamentable in 1973.

Only one factor remains constant: our desire to develop courseware that meets the needs of our students. With today's systems it is easy to become so enamored of the hardware and software capabilities that we lose sight of this desire. The "KISS" (Keep It Short and Simple) principle is often violated as developers fall victim to afflictions such as "font-itis and color-itis" (van Dam, 1990) — using too many fonts or colors on the screen at one time. As far back as 1971, researchers found that the excitement experienced by instructors in preparing programs for classroom use did not necessarily transfer to their students (Desmaisons *et al*., 1971).

The main issue, therefore, is really the same as it was when Pressey introduced his first "simple apparatus which gives tests — and teaches" in 1926: how can we

harness the truly fantastic capabilities of today's systems to serve the needs of our students? Clearly, there is no simple answer to this question. I attempt in this paper, however, to point the way to the multiple answers provided by the *de facto* standards evolving out of the programs now available for today's systems.

The Toolkit Revolution

It has long been argued that to educators interested in using computers in instruction, programming is at best an undesirable chore and at worst an insurmountable obstacle. Some feel that all instructional developers should have at least a basic knowledge of programming, while others feel that such knowledge may actually be counterproductive.

One of the strongest advocates of programming is Paul Tenczar, President of Computer Teaching Corporation, which markets the TenCORE family of products. The TenCORE language is a modernized and expanded version of Tutor, the language that Tenczar developed for the PLATO system in the early 1970s. Tenczar argues:

> While authoring systems requiring little or no computer literacy can open the field to a wider pool of authors, a "programmerless" authoring environment is as limited as a doctorless hospital. ... Programming is one of the elements required by a professional courseware team... (Tenczar, 1990)

Robert Becker expresses a similar view:

> Perhaps the most underrated function on an interactive training development team is programming. Programming is generally equated with *implementation* — an uncreative, mechanistic class of behavior that is supposed to follow instructional design and development in lock step...
>
> [Some advertisements foster] the idea that interactive multimedia training can be created without programmers. Because their work is merely procedural rather than creative, why not automate it by using an authoring system?

> The simple answer is that programming is not merely procedural. It is also among the most creative tasks in multi-media development. A project team that does not solicit design recommendations and guidance from a capable programmer is almost certainly doomed to produce mediocrity — even with the best authoring system in the world. (Becker, 1991)

Not surprisingly, advocates of iconic and object-oriented systems take a somewhat different view. One of the strongest advocates of programmerless systems is Michael Allen, Chairman of Authorware, Inc., which markets the Authorware Professional family of products. Authorware is an outgrowth of the *PLATO Courseware Design, Development, and Delivery (PCD3)* authoring system, developed by Control Data Corporation in the mid-1980s. Allen claims that:

> Through evolution of an approach that is graphic, not based on programming, and uses object-based technology, we feel it is actually a conservative statement to say the Authorware products are at least a generation ahead of even the most costly systems, including PLATO. (Allen, 1988)

It must be recognized that although their approaches differ, both Tenczar and Allen have the same goal: to simplify the process of creating quality computer-assisted instruction (CAI). Allen is no more an advocate of having computer neophytes develop CAI than Tenczar is of having courseware developers work in assembly language (Allen, 1990; Tenczar, 1990). On closer examination of their respective products, one notices an interesting regression to the mean: Computer Teaching Corporation now markets TenCORE Producer, a tool that provides many of the basic capabilities of the TenCORE Language Authoring System through a menu-driven interface. And among Authorware Professional's standard set of icons, one notices that within the framework of the deceptively simple "calc" icon, one can actually write code that looks very much like C. In addition, both systems have long acknowledged the desirability of being able to call out to and be called by external programs.

These developments are extremely important to courseware developers. We certainly want to use the powerful capabilities of systems such as TenCORE and Authorware Professional for organizing instruction, managing multimedia presentations, evaluating student input, and tracking student progress, but we cannot afford to be "locked out" when we need additional capabilities that these systems do not yet provide. We need to be able to pick and choose among all options for improving the quality of our on-line instruction. As Gloria Gery (1986) has said, "I want it all!"

This is where the concept of toolkits comes in. Software development on today's systems is indeed a highly skilled task. It is so complicated, in fact, that only the most diehard bits and bytes programmer would relish writing a program with a state-of-the-art user interface "from scratch." Imagine writing a Macintosh mouse device driver and a complete set of menu subroutines each time you wanted to write an application that uses a pulldown menu! To avoid this, the Macintosh provides a "toolkit" of such routines that high-level programs can call. A similar toolkit is now provided for DOS systems in the Microsoft's Software Development Kit for Windows 3.0. In addition, we are now seeing higher-level toolkits that make these routines even easier to use. For example, it wasn't long after MIT made the Xwindows toolkit available to workstation programmers that a higher level toolkit was developed: Motif.

At the user level, the toolkit concept is also critically important, because few people today use only one software package. As the number of packages that any one person uses on a regular basis continues to grow, it becomes increasingly important that these packages:

(1) share a common user interface so that it is easy to move among them, and

(2) can exchange textual and graphic data so that redundant efforts can be eliminated.

Macintosh users have long enjoyed relatively seamless boundaries between software packages, and DOS users are now starting to enjoy the same benefit with Microsoft Windows 3.0. Donald Norman has stated:

> ... the development of the internal toolbox has caused developers to be consistent in their use of screen, mouse, and key-board, even when their natural tendencies would have led them elsewhere. As a result, most new programs can be used immediately, with little or no study, often without even opening the manual. (Norman, 1990)

Thus, programming toolkits have made possible application toolkits, and already some users are demanding that all applications they consider for use conform to high-level toolkit standards for user interfaces and data exchange. When courseware applications achieve these goals, they can become part of the complete set of software tools that people in all levels of education and training need to get their jobs done. On-line instruction can now become part of a total application package rather than a separate add-on.

Toolkit Design Elements

Is it possible to serve the dual masters of student needs and toolkit standards? I think it is. Today's toolkits are rich in interaction styles, most of which lend themselves very well to instructional purposes. The window metaphor is particularly useful, as instructional designs often call for overlaying graphics or the output of other programs with explanatory text. Let us look at five basic toolkit design elements to examine how they can be applied in instructional sequences.

Pulldown Menus

Virtually all instructional programs provide their users with choices via menus.

Some common applications of menus are:

- selecting the unit one wants to study within a specific course
- switching to another course
- exiting the courseware system altogether
- requesting help on using the courseware system
- requesting help on the course subject matter
- looking up words in a glossary
- responding to multiple choice test items

Over the years, courseware developers have come up with a large number of creative ways to present options such as these to students. Indeed, many different techniques have been shown to be effective. With the advent of toolkits, however, the use of pulldown menus has far outstripped all other techniques. In this technique, a list of main menu options (called a "menu bar") appears horizontally across the top of the screen (or window). When an option is selected, a list of suboptions appears below the main option (see Figure 1).

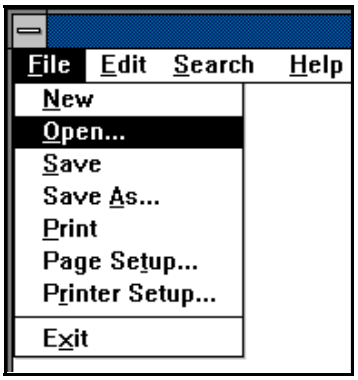

Figure 1. **A menu bar and pulldown menu.**

This type of menu is probably so familiar to most readers that they wonder why I am even mentioning it. That is precisely the point. This type of menu is so familiar because so many programs use it. It is a *de facto* standard, and it can be generated by routines available in virtually all menu toolkits available on DOS and Macintosh systems. While it is impossible to argue that this is the *only* viable format for designing menus

today, I can argue that unless one has a specific reason that warrants designing menus differently, one should use this format. The advantage is two-fold:

(1) the menu format is so familiar to most students that they need no training in its use — on the contrary, many students now consider use of menus in this format intuitive, and

(2) the format is easily implemented through routines available in most toolkits.

Not only is the menu format shown in Figure 1 a *de facto* standard, but the words chosen for the options, the order in which they appear, and their associated "hot keys" (underlined in the figure) have also become standardized. The interested reader may find extensive discussions of these issues in, for example, Apple's *Human Interface Guidelines* (1987), IBM's *Common User Acess Advanced Interface Design Guide* (1989), and the Open Software Foundation's *Motif Style Guide* (1990). Note that these guidelines also include standards for "pullright" menus — submenus of the individual entries on the pulldown menus.

It is interesting to consider what options courseware designers should put on their menus and what actions these options should take to adhere to these toolkit standards. Following is one possibility. Each of the boldface words listed below would appear as an option on the menu bar, with the words in normal text appearing as options on their pulldown menus.

File

New	*switch to a different unit or course*
Open	*temporarily switch to a different unit or course with the possibility of returning to the current unit and course*
Save	*save all data stored on my work up to this point so that I can restart this course at this point after I exit*
Save As	*let me specify the name of the file in which my student data will be stored*
Print	*provide a hard copy of the current page or other data relevant to the course*
Page Setup	*adjust various screen parameters such as colors and type sizes*
Printer Setup	*let me specify the type of printer I want to use*

Exit	*leave the courseware system entirely, with the system saving the data on my work if it has not already done so*
Edit	
Undo	*ignore my last response and let me try again*
Copy	*copy the material I have highlighted to a file so that I can examine it later, perhaps with another software tool*
Paste	*take input from a file I have already created*
Calculate	*give me a calculator that I can use*
Call	*let me call another program and return here*
Support	
Detail	*provide additional details on this topic*
Example	*show an example of the concept being discussed*
Summary	*summarize this section*
Exercise	*present exercises related to the current section*
Test	*test me on this material*
Help	
Subject	*I don't understand this; explain the current subject matter in another way*
Index	*show an index of all help topics*
Keyboard	*provide help on using the keyboard and the meanings of any special key stroke combinations*
Commands	*provide help on the commands available to me at this point*
Using Help	*provide help on using the help system*
About	*tell me the date and version number of this course*

All of the items in this menu structure would not, of course, be relevant to all courses, and others would be needed for some courses. This sample structure does demonstrate, however, that the major properties of the standardized pulldown menu structures we see in other applications can easily be adapted for computer-assisted instruction applications.

Popup Windows

One of the most difficult aspects of courseware design is the management of screen space, simply because there is never enough of it. In my 1984 book I devoted an entire chapter to the issue of functional areas, a term I used to refer to the practice of always putting messages of a certain class in the same area of the screen. For example, I recommended setting aside functional areas for instructions, student entry prompts, help

feedback, and error messages. It wasn't long after the book was published, however, that I modified my recommendation to include the use of windows (Heines, 1985).

Figure 2 shows the use of a window in a course I developed on writing. This window is used to explain the objectives for the current module, overlaid onto a screen explaining how to use the arrow keys in a word processor. Note how the use of a window in this situation maintains the instructional context and lets the student relate the information provided in the window to the task at hand. In addition, it would be impossible to reserve a functional area large enough to display the information in this window. When the student presses Enter, the underlying screen is restored exactly as it was when the student pressed the hot key calling up the objectives display.

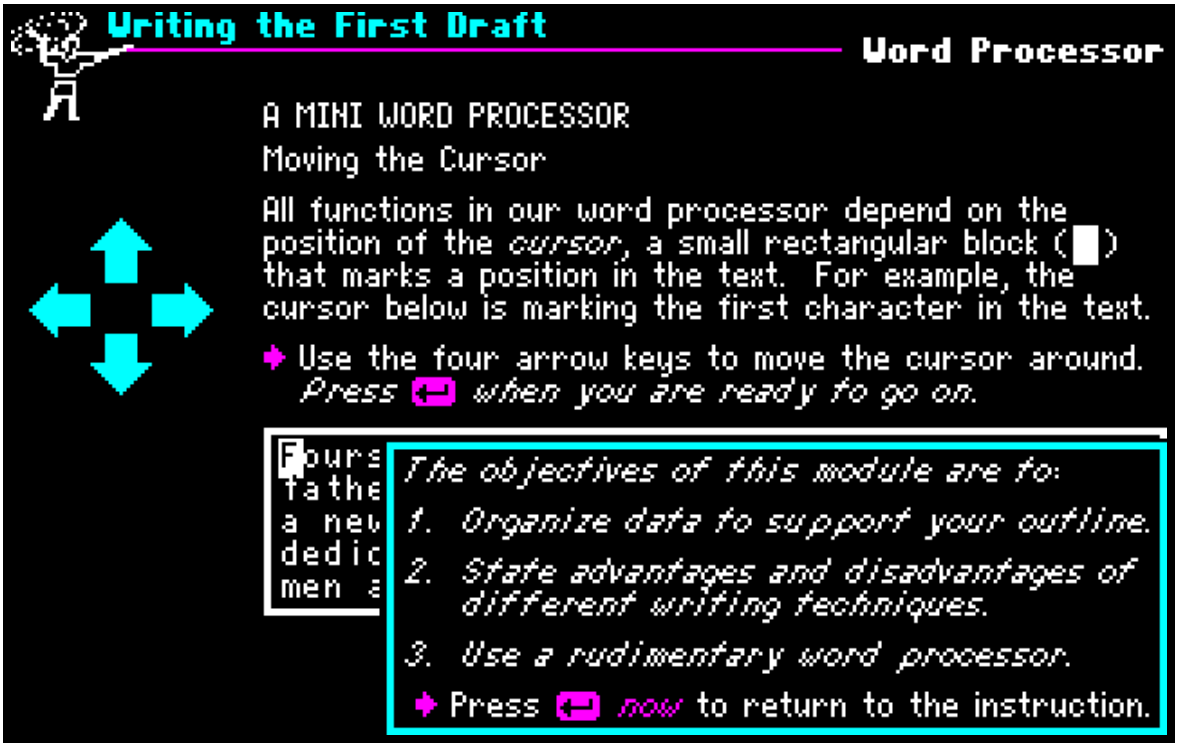

Figure 2. **A popup window showing the objectives for the current module.**

The windowing capabilities of today's toolkits are considerably more sophisticated. Consider, for example, the properties of the standard Windows 3.0 information popup window show in Figure 3, which displays an error message for an improperly configured printer. On first examination, this window appears merely to contain four visual elements missing from the window shown in Figure 2:

(1) a title bar

(2) a system pulldown menu at the extreme left of the menu bar

(3) an icon (the exclamation point) indicating the type of its message

(4) an OK button to remove the window from the screen

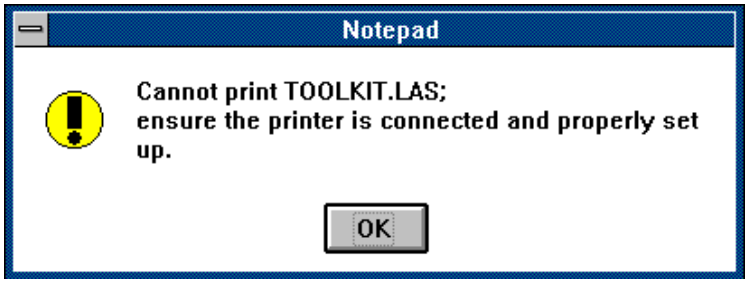

Figure 3. **A Windows 3.0 popup window.**

It has two other important properties, however, that cannot be seen visually but that are even more important:

(5) it can be moved about the screen through standard Windows 3.0 protocols

(6) it can remain on the screen while the user takes the actions suggested — it is only removed from the screen when the user clicks on OK

This last property is critically important in instructional situations, where if students request help, they typically want it to remain on the screen while they answer a question.

Scrollable Windows

Another common courseware design problem raised by my 1984 book is that no matter how interactive and graphic-based we strive to make our courseware, at some point we are always faced with the need to present more text than can comfortably fit on a single screen. In 1984 I argued against pausing between successive displays unless the student had a clear and easy way to get back to the text that was erased when new text was displayed. As much as all proponents of interactivity deplore excessive press-return-to-continue scenarios, I felt that this technique was preferable to erasing information before students may have had a chance to digest it.

Once again modern toolkits come to the rescue by providing us with scrollable windows (see Figure 4). These text presentation tools allow students to view any amount of text in a convenient package. They can advance or backup through the text a line at a time (by clicking on the up and down arrow keys on the vertical scroll bar at the right of the window) or a screenful at a time (by clicking on the scroll bar above or below the position marker). Students can also quickly advance to any point in the text by dragging the position marker up or down.

It is interesting to note that the use of a scroll bar also provides important orientation information recommended in my book — students can easily tell how far they are through the text by the relative position of the marker. This type of orientation information has been implemented in the Teletutor series of telecommunications courses from Cooper & Associates in Portsmouth, New Hampshire. Their *Introduction to Telecommunications* course (1988) used a diamond-shaped position marker on a line at the bottom of the screen to indicate the student's position in the course (see Figure 5). Unfortunately, this course was developed before toolkits were available that allow students to move the marker and actively adjust their position in the course.

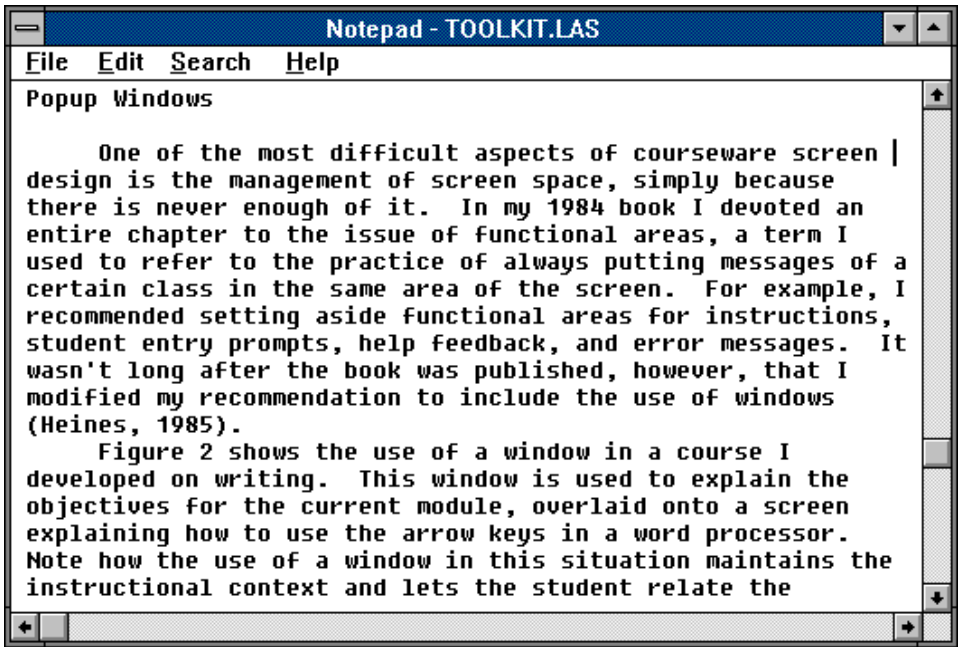

Figure 4. **A Windows 3.0 scrollable text window.**

Figure 5. **Use of a marker similar to that in a scrollable window to provide orientation information on a student's position in a course.**
The marker is the diamond at the left of the line at the bottom of the screen.

Dialog Boxes

Today's toolkits handle the majority of *structured* data entry through tools known as "dialog boxes." These tools provide a large number of interaction styles, three of which are shown in Figure 6.

(1) The **Find What** field is a *text entry field* or *edit box*. The cursor appears as a vertical bar and may be positioned using the mouse or the arrow keys.

(2) The **Match Upper/Lowercase** option is a *toggle* or *check box*. Clicking in the box turns this option on, signified by the presence of an "X" in the box. Toggle box options are usually independent; each toggle box may be turned on or off without affecting the state of other toggle boxes.

(3) The Search Direction (**Forward** and **Backward**) options are *radio buttons*. Clicking in either of the circles turns the corresponding option on, simultaneously turning all other radio buttons in the same group off. Radio button options are usually mutually exclusive; only one of each set may be on at any given moment.

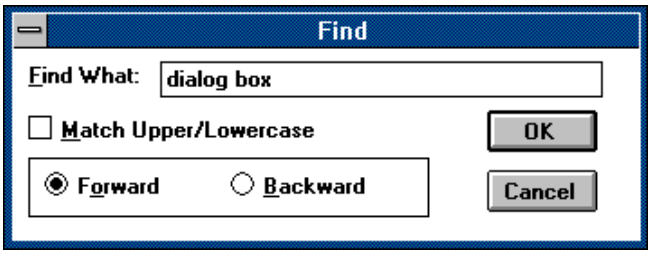

Figure 6. **A Windows 3.0 dialog box.**

Other basic dialog box interaction styles include *slider scales*, in which values are specified by sliding an icon left and right (or up and down) on a numeric scale, and *menu lists*, in which choices are presented in a menu and the user highlights the option he or she desires. Detailed information on the use of these styles is provided in Murray and Pappas (1990), Young (1990), and Open Software Foundation (1990).

The interactions in dialog boxes are extremely intuitive, especially when one is using a mouse or other such pointing device. In addition, it is easy to find applications of

these techniques in instructional settings. For example, multiple choice questions lend themselves perfectly to radio buttons, while questions such as "indicate all possible values of the discriminant when A=1, B=2, and C=3" lend themselves perfectly to a set of toggle boxes. Use of these toolkit techniques can foster valuable consistency between courses, making it easier for students to concentrate on subject matter rather than the mechanics of specifying their responses.

Help Tools

The last courseware design toolkit technique I want to discuss is on-line help. The help available on the Macintosh and Windows 3.0 today are highly sophisticated software tools. In fact, writing help for sophisticated applications using these systems is itself a skilled art. These help systems allow users to get context-sensitive help on the task at hand, access an index of all help topics available, backtrack to topics they have seen in the past, browse forward and backward through help on other subjects, and search for help on topics containing specified key words (see Figure 7). In addition, the text may contain "hot spots" — graphical or textual — on which users may click to jump to linked help messages in a hypertext manner. (Such hot spots are underlined in Figure 7.)

Help tools will surely continue to develop until they evolve into complete computer-assisted instruction authoring systems. Indeed, a number of the com-panies with which I consult are looking at having their software call courseware written in TenCORE, Authorware Professional, or other such tools to provide learning environments for their products. I believe that this level of integration will represent a major shot-in-the-arm for computer-assisted instruction, and truly allow software systems to make possible what Allen has called "Just In Time Learning" — the ability "to provide instruction on the user's selected topic on demand" (Allen, 1989).

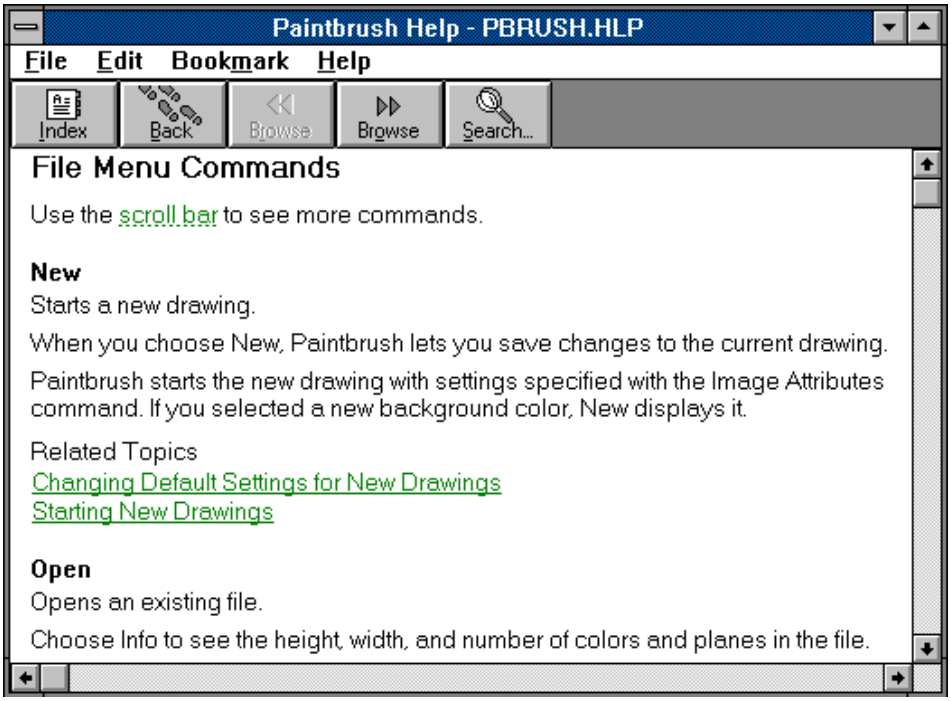

Figure 7. **A Windows 3.0 help window.**

Using Toolkits Creatively

Occasionally one hears the assertion that toolkits suppress creativity. "Why should we use toolkits that almost force us to conform to a user interface standard that is still evolving?" the resisters ask. "Sure, it makes implementation easier," they admit, "but the price of restriction on our creativity is too high." There are two answers to such arguments, and both are expressed in the main issue I raised at the beginning of this paper: "how can we harness the truly fantastic capabilities of today's systems to serve the needs of our students?"

The first answer is that today's toolkits are quickly growing in depth and scope, providing an increasingly rich set of user interface techniques that can be applied in highly creative ways. One of the best examples of such creativity may be found in

computer games. Figure 8 shows a screen from *Hero's Quest I* (Cole *et al.*, 1989), a game which makes heavy use of pulldown menus, dialog boxes, and other such toolkit techniques that game enthusiasts move through with amazing speed to get to the competition. Such users are seldom interested in implementation issues; they simply want to know the rules and begin competing as quickly as possible. The creativity in these user interfaces can be found in their layout, choice of type fonts, use of graphics, and the options that they provide. The interaction techniques themselves are very secondary — especially to the 8-12 year old boys who form the vast majority of these programs' users — but they are essentially the same as those discussed in this paper.

Figure 8. **Use of pulldown menus in a computer game.**
Copyright 1989, Sierra On-Line, Inc.

The second answer is that we must constantly remind ourselves that the purpose of courseware is to meet the needs of students, not instructors. Most students care very

little about whether they select an answer by pressing a key or by clicking a mouse button. They are vastly more concerned with the correctness of their responses than the techniques they use to indicate those responses. The consistency provided by toolkits can virtually eliminate the need for modules that teach how to take a computer-assisted instruction course, allowing students to concentrate exclusively on the subject matter.

We must also keep in mind that the programming/programmerless authoring system argument is of absolutely no interest to students. Their only concern is with the courseware they must try to learn from, regardless of how it was developed. It is interesting to note that few of us who make a living developing courseware have actually ever taken a CAI course as a student. We have all tried courses out as students, but how many of us have actually taken a course for academic credit in which CAI is the main instructional medium? Courseware development is the construction of a piece of software. The building blocks are computer subroutines, and one may join them together by writing code or by arranging icons representing them in a logic diagram. The important point is not to "reinvent the wheel," but to use standard toolkits so that the product has a standard "look and feel" and can be completed in time to allow revisions when it is tested with students.

It has been shown that few people read manuals before they use familiar devices such as radios, lawn mowers, and microwave ovens. (When was the last time you rented a car in which you could even *found* the owner's manual?) The user interfaces to everyday devices have become so standardized that their use is intuitive to most children as well as adults. Courseware design based on toolkits will allow us to achieve the same intuitiveness in on-line instruction. We can then focus on our students' needs and spend our time and energy experimenting with creative solutions to problems dealing directly with their mastery of the subject matter.

References Cited

Allen, Michael W., 1988. Sorry, but this story is unbelievable. *coAction Magazine,* 1:1, pp. 4-5. Authorware, Inc., Minneapolis, MN.

Allen, Michael W., 1989. CBI could be just in time. *Authorware Magazine,* 2:1, pp. 4-5. Authorware, Inc., Minneapolis, MN.

Allen, Michael W., 1990. "No experience needed!" — what's the purpose of an authoring system? *Authorware Magazine,* 2:2, pp. 4-5. Authorware, Inc., Minneapolis, MN.

Apple Computer, Inc., 1987. *Human Interface Guidelines: The Apple Desktop Interface*. Addison-Wesley Publishing Company, Reading, MA.

Becker, Robert S., 1991. How to build an authoring environment. *Instructional Delivery Systems,* 5:2, pp. 6-15. Communicative Technology Corp., Warrenton, VA.

Cole, Lori Ann, *et al.*, 1989. *Hero's Quest I* (interactive computer game). Sierra On-Line, Inc., Coarsegold, CA.

Cooper & Associates, Inc., 1988. *Introduction to Telecommunications* (computer-assisted instruction course). Cooper & Associates, Portsmouth, NH.

Desmainsons, Robert E., Maury P. Hepner, and Robert J. Dirkman, 1971. *The BRAIN System.* In Blum, Richard (ed.), *Computers in Undergraduate Education,* pp. 121-135. Commission on College Physics, College Park, MD.

Gery, Gloria, 1986. I want it all! *Data Training,* March 1986. Weingarten Publications, Boston, MA.

Heines, Jesse M., 1984. *Screen Design Strategies for Computer-Assisted Instruction.* Digital Press, Bedford, MA.

Heines, Jesse M., 1985. I do windows. *Training News,* 7:3, pp. 5-6. Weingarten Publications, Boston, MA.

IBM Corporation, 1989. *Common User Access Advanced Interface Design Guide.* International Business Machines Corporation, Purchase, NY.

Martin, James, 1973. *Design of Man-Computer Dialogues.* Prentice-Hall, Inc., Englewood Cliffs, NJ.

Murray, William H., III, and Pappas, Chris H., 1990. *Windows Programming: An Introduction.* Osborne McGraw-Hill, Berkeley, CA.

Open Software Foundation, 1990. *OSF/Motif Style Guide.* Prentice-Hall, Englewood Cliffs, NJ.

Norman, Donald A., 1990. Why interfaces don't work. In Laurel, Brenda A., ed., *The Art of Human-Computer Interface Design*, pp. 209-210. Addison-Wesley Publishing Company, Inc, Reading, MA.

Pressey, Sidney L., 1926. A simple apparatus which gives tests — and teaches. *School and Society*, 23:586, pp. 35-41. As reprinted in Lumsdaine, A.A. and Robert Glaser, 1960, *Teaching Machines and Programmed Learning: A Source Book*, pp. 59-65. National Education Association of the United States, Washington, DC.

Tenczar, Paul, 1990. Dear readers. *Tenfold*, 3:2, p. 2. Computer Teaching Corporation, Champaign, IL.

van Dam, Andries, 1990. *Electronic Books: User-Controlled Animation in a Hyper-media Framework.* Lecture presented at *User Interface Strategies '91,* a teleconference broadcast from the University of Maryland, December 5, 1990.

Young, Dennis A., 1990. *The X Window System, Programming and Applications with Xt, OSF/Motif Edition.* Prentice Hall, Englewood Cliffs, NJ.