TrainingNews
Volume VI, Number 11 July 1985

## THE CBT CRAFTSMAN

# ¿Queue Parser?

CBT authoring systems are appealing for their ease of use, but one of the reasons they're so easy to use is that they're not designed to do very complicated things.

**Jesse M. Heines**

The best CBT programmer I ever worked with had red hair, a red beard, and a Cuban wife. When I first met his better half, I quickly tried out the only thing I knew how to say in Spanish: "Eh! Que pasa?," which I'm told means "Hey! What's happening?" She wasn't too impressed, but then she didn't seem offended either. Anyway, the phrase became a kind of password between this superprogrammer and me, and many detailed code design sessions began with one of us walking into the other's office and chirping "Que pasa?" He never once said that an interaction I had dreamed up couldn't be implemented. He did, however, often quote the old adage, "The impossible we do immediately; miracles take a bit longer!"

One of the first problems this programmer and I addressed together was that of handling optional student responses, that is, student input that is not in direct response to a question. Such optional responses include requests for help, access to an on-line glossary, a look at the objectives, a review of the previous display, and so on. We wanted these options to be available at all times, but unfortunately, we were not working on PLATO with its built-in HELP, TERM, DATA, and BACK function keys and complementary language constructs. The options had to be available at all times, regardless of whether the program was expecting a single keypress or a series of keystrokes followed by RETURN. This called for us to design a generalized student response *parser*, a subroutine which accepts and interprets student input. The parser grew and grew as we programmed it to accept more and more optional student responses. As the parser became more elaborate, response time slowed down. In our efforts to speed up response time, we spent many hours hunched over line printer listings of the code making it more efficient. Whenever the listings got too marked-up to follow, we'd call for a break and queue a new listing of the parser to the line printer. "Que pasa?" thus soon evolved into "Queue parser?"

**Proponents of authoring** systems may look down on the decision to program a parser to handle student responses and say that one of the main reasons authoring systems were developed was to eliminate the need for CBT course developers to program complex parsing routines. I will agree, but only to a point. All authoring systems provide a way to get, store, and analyze student input, but many limit analysis of that input to checking for "right" and "wrong" responses and branching to another section of code—or, worse yet, another "frame"—based on the student's response. While a specific authoring system may make programming this type of interaction very easy, it may at the same time make programming a more complex interaction (or branching decision) literally impossible. This philosophy represents the worst aspect of some authoring systems: they make the simple trivial, but they make the complex impossible.

Consider, for example, the ubiquitous multiple choice question. No matter how sophisticated you may make your course design, there is always some point at which the author wants to ask the student to choose from a fixed set of options. This format may be clearly evident in a common multiple choice test item, or it may appear more subtly as a menu with a choice of options. All of today's authoring systems provide some easy method for accepting a single student response to multiple choice items. The problem arises, however, when we begin to examine the *style* of that interaction:

- Does the authoring system limit the number of alternatives you can present, or are you free to present any number?
- Does it force you to mark the alternative with letters, or can you use numbers as well?
- Can you make the program respond as soon as a key is pressed, or must you require the student to press RETURN after typing his or her response?
- Are there canned responses for unanticipated student input, e.g. a "6" when you have only five choices, or can you specify how the program should respond?
- Does the authoring system give the student a predetermined number of chances to respond, or can you control how many times the student is allowed to try the question?
- Is there any way to ask the student to select *more than one* of the items on the list?
- Can you accept "coded" alternative responses, such as a press of the F6 key to back up to an explanatory display or a response of "H" for help, in addition to the identifiers for each of the alternatives in the question?
- Can students use the arrow keys to move a pointer indicating their choice, or must they type a response? (Use of the arrow keys is very beneficial for non-typists.)

**Some authoring systems** will provide all of these options, while others will be much more rigid. The rigid systems not only stifle creativity and the development of crafted courseware, but they can also seriously affect the program's instructional effectiveness. Such rigidity forces course authors to adapt their teaching strategies to the capabilities of the authoring system, thus simplifying the range of interactions in the name of simplifying the program implementation. At their best, programs with restricted interactions are boring; at their worst, they're instructionally stagnant.

Not everyone is as fortunate as I have been to work with a good programmer who implemented interactions as I designed them rather than as dictated by the constraints of some authoring system. Response handling is the most visible of these constraints, but others exist, particularly in the display of graphics. For example, I become very frustrated when I know that my computer's basic input-output system provides sophisticated graphics functions that I cannot access from an authoring language. Alfred Bork, the reknowned pioneer of computer-assisted instruction in physics, and his colleagues at the University of California at Irvine develop all of their courseware in Pascal. The reason for this choice, according to Bork, is not that Pascal is a particularly good authoring language. It is that Pascal is simply a good language.

Remember that naive courseware authors, like naive computer users, stay naive for only about two weeks. Sooner or later they outgrow the capabilities of almost all authoring systems and want to do more sophisticated processing like parsing. Another old adage—"make it simple enough that any idiot can use it and only idiots will"—holds true for authoring systems as well as applications programs. I am all for ease of use, but not at the expense of functionality. Instead of trying to shoehorn your instructional design into a restrictive authoring system, get yourself a real CBT programming language with a rich set of subroutines—and a red-haired programmer if necessary—and you'll create CBT courseware that is instructionally sound, visually stimulating, and comfortable for students to use. □

*Jesse M. Heines is an independent consultant based in Chelmsford, Massachusetts. He develops custom CBT programs and provides training and consultation on CBT courseware development. Dr. Heines is also an assistant professor of computer science at the University of Lowell in Lowell, Massachusetts, and the author of* Screen Design Strategies for Computer-Assisted Instruction.