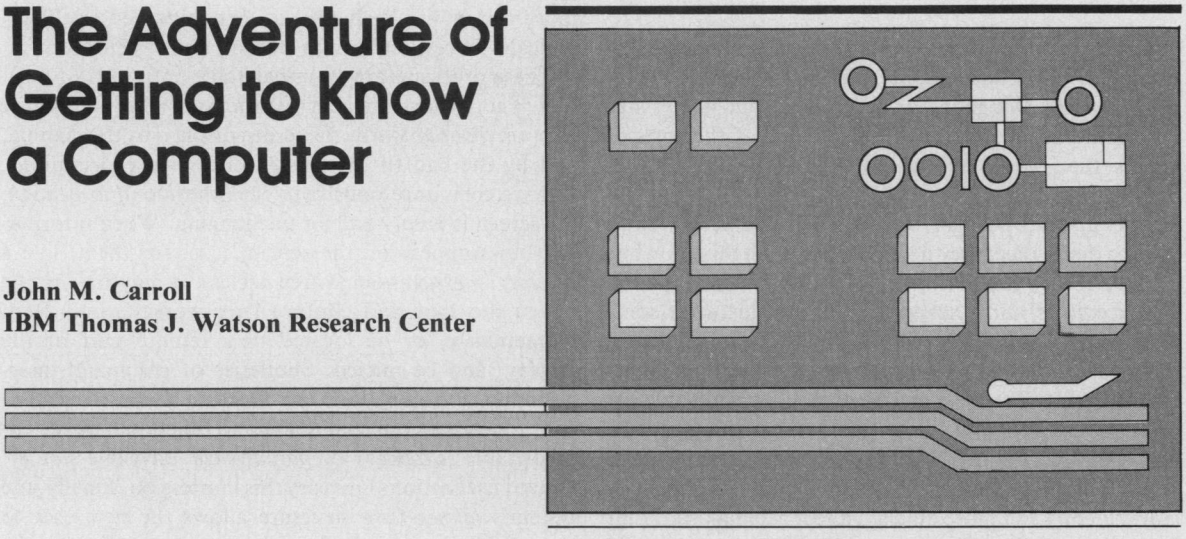
Making text editors more like computer games may seem ridiculous on the surface, but these "games" use basic motivational techniques—something designers of application systems have overlooked.

The Adventure of Getting to Know a Computer



John M. Carroll
IBM Thomas J. Watson Research Center

It is suppertime, and most everyone in the computing center has already gone home. But framed against one bright gray wall sits a lone explorer. In his mind, he is deep underneath the surface of the earth in a dark and dangerous cave—but he is entering into a world of color, imagination, action, and general amazement. He is playing Adventure. On the screen is a cryptic message: "The bird was unafraid when you entered, but as you approach, it becomes disturbed and you cannot catch it." He stares intently, his thoughts are almost tangible, leaping between his head and the screen. He is going over every variation of every possibly relevant parameter of the situation, and he will do this over and over again until he has wrung out *at least* 470 points. It's just a game.

The very next morning, someone else is sitting in the same chair, directly in front of the same console, staring into the very same tube. Despite the background bustle, this person also sits quietly mesmerized by the cryptic message on the screen: "Task not applicable at this time." Like her predecessor, this person is silently examining her knowledge and her hypotheses about the system: "What is a task?" "Does 'not applicable' mean I don't need to do this or that I have to do it, but not this way?" "If I sit here and wait for some time to pass, will it work then?"

But unlike her predecessor, this person is not navigating a cave far below the surface of the earth—she is trapped between two menus in a text editing facility. Also unlike her Adventure counterpart, she will probably not win all 470 points, at least not for a long time, because she can resolve the task-not-applicable problem more quickly than the other can resolve the bird problem—she simply decides that "Task not applicable at this time" means nothing very much at all and goes on. She gets to the next menu, or the prior menu, or she just turns the machine off and starts fresh—the lessons, whatever they might have been in the prior session segment, are sacrificed. The goal, after all, is to learn how to get a letter typed out—this is no game!

Here we have two people, a player of Adventure, a well-known and popular computer game, and a user who is trying to learn an application system. Despite their obvious differences, they have much in common: both are struggling to cope with an unfamiliar environment, and both are experiencing certain types of learning difficulties. To a large extent, these difficulties are inevitable characteristics of human-computer interaction and as such are potential problems in any system. In Adventure, however, problems are turned into challenges; whereas in application systems, they are burdens—even severe learning obstacles at times. This difference may partially explain why people master computer games with no useful (practical) goal, while they fail to become even accomplished novice users of application systems like text editors.

Studies of office personnel learning to use text editors show that the type of learning environment affects how the user perceives the system and how easily he learns to use it.[1-3] A computer game like Adventure has a conceptual, mazelike learning approach, which I call an exploratory environment, that makes the player *want* to overcome the problems, even invent ways to use them to his advantage. The application system, on the other hand, has a more passive, prescribed environment that seems to frustrate more than help. What I hope to show in this arti-

0018-9162/82/1100-0049$00.75

cle is that by examining the similarities and differences between the Adventure player and the inexperienced user, we can find some insights to use in designing application systems that are easier for the user to learn.

The common basis

Computers are freeing us from paper and pencil. For example, Adventure is quite similar to the paper-and-pencil game, Dungeons and Dragons, but games like Adventure are masters at keeping track of the sorts of details that would render paper and pencil versions unbearably tedious. Players don't have to wait to find out what happens to them in the course of a move; they don't have to throw dice; they don't have to write things on bits of paper. They just get the good stuff.

Text editors, and other application systems, are in some ways even better.* They relieve people of having to scrawl things on paper—even people who are writing a letter! Once in the system, text can be revised, manipulated, or printed out. Repetitive tasks, like mass mailing, can be automated by the use of variables: once input, forever done. Using a text editor can help a child learn to write more fluently and more quickly, and it can make an adult feel more comfortable and confident about writing.[5,6] Moreover, learning to use a text editor is easy—at least relative to learning to write or type. Text editing is really only a step away from typing and actually reenforces this skill.

But although text editors are potentially "easy" to learn to use, people are still having tremendous difficulty learning to use them. In most cases, they are learned incompletely and after significant confusion and error. A variety of severe problems beset people who are learning to use contemporary text editors.[1-3,7] From these, I have selected eight specific problems that are shared by a person learning to use an application system and a person playing Adventure. These learning problems, which are summarized in Table 1, may seem bizarre and exotic to those unfamiliar with text editors, but they are fairly typical with respect to both learners and systems.

*Malone[4] distinguishes sharply between application tools (e.g., text editors) and puzzles, which include computer games like Adventure. But his distinction rests on an error. He says, " . . . in the case of text editors, the central problem is not how to use editing commands, but how to improve the document being edited." In studies of those learning to use text editors, we at the research center have *never* come across a subject who thought of the task in these narrow terms. The problem our learners spontaneously—and quite stubbornly—orient to is "how does this system work?"

Table 1.
Typical problems faced by new users and Adventure players.

Disorientation	The user/player doesn't know what to do in the system environment
Illusiveness	What the user/player *wants* to do is deflected towards other, perhaps undesired, goals.
Emptiness	The screen is effectively vacant of hints as to what to do or what went wrong.
Mystery messages	The system provides feedback that is useless and/or misleading.
Slipperiness	Doing the "same thing" in different situations has unexpectedly different consequences.
Side effects	Taking an action has consequences that are unintended and invisible, but cause trouble later.
Paradox	The system tells the learner/player to do something that is clearly inappropriate.
Laissez-faire	The system provides no support or guidance for overall goals (e.g., "winning," "typing a letter").

These problems are an unpalatable potpourri of troubles. People have difficulty getting started at all because they are *disoriented* by the screen display, by the manual, and by the bad fit of both to their own expectations. The system is unresponsive to what they do (*illusiveness*); the screen is *empty* and/or unchanging. When information does appear on the screen, it is, for them, like a *mystery message*—and often useless. It may stay on the screen too long and confuse later work; it may flash momentarily, or be located in a remote part of the display, and be missed. Subtleties of command interpretation and command architecture make the causal connection between commands and functions appear unpredictable (*slippery*) or *paradoxical*. Invisible *side effects* of user actions intensify this impression. Finally, the system's *laissez-faire* structure allows the new user to become lost in a maze of mystery messages, commands, and side effects.

Disorientation. Computing systems are often very alien to the backgrounds of new users. When a novice enters a computing environment, the first reaction is disorientation. Learners my colleagues and I have studied spend interminable amounts of time just trying to sign on—and then trying to figure out what happened when they did sign on. Suppose they manage to log into the local system, and even to enter the text editing environment. They see at the top of the screen:

EDIT: 'TEXT SCRIPT A1' TABS: 1 6 11 16 21 26 31

In the middle of the screen, at the left:

EOF:

And at the bottom of the screen, at the right:

YKTVMV RUNNING

"OK, now what?" the users ask, and naturally, help is not available in this initial state.

Likewise, the player enters an environment like Adventure with no idea of what will be there. Presumably *some* sort of adventure will occur, but from the start, he doesn't know what sort of adventure or where to find it. The first message from the game to the player is simply

> You are standing at the end of a road before a small brick building. Around you is a forest. A small stream flows out of the building and down a gully.

Where is the adventure? In the building? Surely not in the stream. Perhaps in the forest . . . The answer just isn't there.

Illusiveness. Computing systems, even the ones data processing professionals consider elementary, are complex. The new user may sign on with a goal, like typing a

memo. But very quickly that goal is all but lost in a morass of decisions and subgoals that develop and must be confronted during the session's interaction. First, the user might find out that what he calls "typing in a letter," is known to the system only as Create Document or Input Mode. ("Ah-ha," the user thinks, "so *that's* what I wanted to do!") But first, he must dispose of the Create Document Menu, which asks for a "document name"; whether the document will be "shared access" or "private"; what "retention period" is desired; which "Document Format Source" is to be invoked; and what "Charge Number" will be used. By the time the user has dealt with all these matters, he will be lucky indeed to remember the content of the letter!

Adventure is again closely parallel. Because the task situation of the game changes constantly from one instant to the next, the overall goal of winning (like typing a letter) evolves endlessly into a succession of more immediate, lower level goals. At one point, a player's goal is FIND TREASURE, but he then stumbles into a maze—the new goal is GET OUT. Suddenly, a little dwarf attacks with knives, and the player's goal becomes KILL DWARF. Note that *the player* has not done anything to change the game situation. From the player's perspective, while trying to accomplish *X,* something unforeseen occurred that changed things, impelling him to accomplish *Y.*

Emptiness. To protect the user from what's going on inside the application system, designers typically opt to display little or nothing to define what has just happened or what is now happening. Frequently during a learning session, the new user is staring at a blank, or at least static, screen. For example, when each new command is issued, the display is typically cleared of prior commands. The learner is thus deprived of any trace of the session, which he may need in planning further action. Indeed, many actions that a user initiates have no visible consequence on the screen at any time—although inside the system they are being executed. This lack of response is often true for both appropriate and inappropriate commands. Wildly wrong keypresses, for example, those that have no meaning anywhere in the system, may only trigger a reset condition, which is signalled by a tiny light on the side of the screen (where learners often fail to notice it). Maybe you're right; maybe you're wrong; maybe you're making progress; and maybe you're not—but the machine's not telling.

Here, the Adventure parallel breaks down just a bit. For although the screen per se is frequently empty and static in Adventure, the game does put forth some response to every user action—albeit brief and cryptic: "You can't be serious!" or "I don't understand that!" Moreover, the language that the system employs is extremely vivid; the world that Adventure constructs is one of extraordinary visual intensity, even though the scenario of the game is in caves, which in the real world are dimly lit places of limited color. In Adventure, when you kill a dwarf, "The body vanishes in a cloud of greasy black smoke." When you enter a room, it seems to light up:

> You are at the east end of the two-pit room. The floor here is littered with thin rock slabs, which make it easy to descend the pits. There is a path bypassing the pits to connect passages from east to west. There are holes all over, but the only big one is on the wall directly over the west pit where you can't get to it.

Even when the ambient light is low and things are shadowy, the descriptions are still vividly intense.

Mystery messages. System messages are designed to be succinct and precise, but even when they are they may be of little use to the new user. Quite often messages that the system delivers to the user are puzzles rather than prompts, problems rather than assistance. For example, one system says "Task not valid at this time," which is really too vague to be helpful. The message is also a bit jargony, since few new users have figured out what a "task" is, and it stays on the screen almost permanently, worrying the user about a continuing error. One subject we studied interpreted the message as meaning that her request could not be completed at that time and thought that if she waited, it would later work all right. "Task not applicable in this situation," would be somewhat of an improvement, but it, too, is jargony and conveys only that *something* is amiss.

Adventure is full of these mystery messages. For example, early in the game one approaches a little bird.

> The bird was unafraid when you entered, but as you approach, it becomes disturbed and you cannot catch it.

Now even a player who has barely managed to get this far in the game knows that the bird must be captured or taken along somehow, but something is still *not* being said—just the something that the player needs to know.

Slipperiness. Computing systems require that commands be entered exactly. A slight deviation in spelling or syntax renders a command meaningless, or worse, changes the meaning to something unintended and entirely different. Commands also frequently function differently when issued in different system states. The learner, then, may not notice a slight misspelling amid all the disorienting confusion, and he typically cannot recognize the difference between relevant system states. From the user's viewpoint, and from the player's as well, the very same action mysteriously elicits unpredictably different consequences when employed on different occasions. In one text editor, a buttonpress command, "Cancel," is used to escape from many situations (for example, menus). Learners invariably try to use Cancel to escape from a typed page display, for example, when they have completed a memo and want to store it or print it out. However, Cancel does nothing; instead, the mystery message "Task not valid at this time." appears. To escape from the typed page situation, the user must employ a different command entirely.

Adventure is also slippery. Early on, the adventurer comes upon a rod, which he can wave, but a mystery message says, "Nothing happens." Only when he waves the rod in a certain place in the cave does something happen: a crystal bridge appears at the great fissure. Another example is the Pirate's treasure, which does not materi-

alize in the cave until all other treasures are collected. Adventure is slippery because of its probabilistic aspects. When a dwarf attacks a player, he thinks, "If I am prudent and lucky enough to have an axe with me, I can throw the axe at the dwarf. Having done all this, I might kill the dwarf and save myself, or I might miss and have to make a death-defying attempt to retrieve the axe and try again." Success in killing a dwarf depends on doing the right things, but also on random luck. The game is slippery here; the player can do everything right and still end up dead.

Side effects. Each command causes a variety of effects, only some of which are visible to the user. Nevertheless, the other, invisible, effects of a command can become relevant to the user's goals later on. Quite often the inexperienced user or the Adventure player can do something that seems harmless enough—perhaps even right—at the time but that later produces a negative side effect. For example, printing a document may cause it to become inaccessible for re-editing (while it is on the print queue). To the new user, this side effect is unpredictable, and the system may give no hint that it has occurred. So when the learner asks to have the file again for reformatting, the mystery message "Document Unavailable" comes up. Sometimes users believe that someone else has appropriated their document. Sometimes they believe that the machine has temporarily lost the document. They never guess that *they* caused this undesirable situation.

In Adventure a player encounters a magic rod, which he generally picks up and takes along. Later he encounters a little bird. However, he cannot now catch the bird. As the message (given above) indicates, the bird is now afraid. What the player doesn't know is that while he is holding the rod, the bird is afraid. This is an odd side effect of the rod and an unpredictable obstacle to progress in the game. As a tricky consolation, the game allows him to kill the little bird (which he is inclined to do after not being able to capture it):

The little bird is now dead. Its body disappears.

But side effects from this action will occur: killing the bird will make progress impossible later in the game.

Paradox. New users come to a system with a rich, and often idiosyncratic, body of knowledge and experience. They use this background to interpret each message and instruction that is presented to them. When prior knowledge is generalized correctly, it can be useful, but it can also be trouble. Often new users and players are presented with actions or choices that seem clearly wrong. Consider an instruction manual that says "Backspace to erase." The typical learner, probably thinking of typewriters, reads this as a paradox. Backspace doesn't erase on a typewriter; it just moves the typing point back. Indeed, I have watched learners come to a complete standstill during an exercise and stare vacantly at this instruction. The paradox—real though it may be to the novice—is only apparent, because if the user throws intuition to the wind and backspaces anyway, he finds that backspace does delete prior text. In the same system, CHAR DEL can also delete blank *lines*—since the system views them as characters in the data stream.

Adventure is a game about collecting treasures, yet at one point the player encounters a troll who demands one treasure be given up in payment for crossing his bridge, which the player rightly suspects will lead to more treasures. How can this be? Of course, the lucky player can foil the troll and avoid giving up a treasure, but even so, he is faced with the immediate paradox. Another example is that, at the end of the game, the player must decide whether or not to blow up the cave through which he has so carefully navigated.

Laissez-faire. Systems are finally *tools*, and as such they are intended to be used. Although initially the user may be led through programmed learning exercises, ultimately the user will direct the system. Except for truly intelligent systems, the system must play a limited role in directing and supporting the user's goals. Thus situations will inevitably occur in which each command issued is correct in the short term—but also perfectly useless since the command fails to advance the user's overall goals. Indeed, these situations are commonplace of novices.

Not only is laissez-faire inevitable, but it is also a bit cruel. Computer systems, like anything complex, can be conceptual mazes—I have often observed learners wandering hopelessly and without end. The system just accepts each new command, letting the lost learner race onward to nothing at all. The game scenario in Adventure is quite literally maze learning, and Adventure doesn't merely *allow* the player to wander; it traps and compels him to wander. For example, one region of the Adventure cave consists of "twisty little passages all alike."

Remedial actions

Having defined these problems, we can now look for ways to treat them. The first approach that comes to mind is using common sense. A serious common-sense analysis of the new user's plight may provide some insight, but as the past bears out, it alone is not likely to solve the problem. The more fundamental point is that people want to *do* things with computers and, particularly when they are learners, they make errors. These errors tangle up the "pure" forms of the problems in Table 1 and are impossible to anticipate (or to diagnose) by mere common sense. The bottom line is that people do not, and possibly cannot, learn passively—although many current system designs incorporate this learning approach.

Common sense. I have presented the learning problems in their extreme form, which may not be something we have to live with. Indeed, we should be able to resolve the extreme cases fairly quickly. For example, messages can be purged of gratuitous jargon merely by adding some common sense to the design process. In one system, menus are defined as displays containing "items." Subsequently, an item is called a "parameter" in a menu. Now this label would be reasonable if parameters were not always items, but they are. To complicate things further,

the command that moves the cursor between items/parameters in a menu is VAR ADV (Variable Advance)—even though all variables are parameters. What we have, then, is item-parameter-variable, three terms instead of one. Even worse, the system recognizes only parameter and variable, which are the two terms learners don't know.

Getting this kind of common sense into system design is evidently easier said than done, but for the sake of argument, suppose we could. I'm not certain that even then we could resolve the troublesome usability concomitants of these problems. Systems are complex, and any action we take will have side effects. Perhaps these side effects can be made more visible to the user, but they will be there, nonetheless. To resolve emptiness, we need to strike a balance between empty or static screens and screens so cluttered with explanation and aids that they are potentially confusing. However, we cannot simply "solve" this problem.

The problems of laissez-faire and illusiveness have a certain inevitability that is, surprisingly, somewhat desirable. If the application system is to be a tool and not a task master, then endless wandering is always possible, at least until we can anticipate user goals. And any task involving complex and dynamic goal elaboration has a certain degree of illusiveness. However, Bandura and Schunk[8] and Csikszentmihalyi[9] argue that these problems may be beneficial. The possible short-term goals that can emerge from the interaction of agent and environment are crucial in motivating a user and keeping him oriented to a task.

Allowing for user error. Many problems in Table 1 have a hybrid form that develops through interaction with user error. Thus, pure slipperiness is probably something that a serious understanding of psychological consistency can ease considerably.[10] But slipperiness can also be apparent, that is, caused by error. In learning the text editor referred to above, users often picked up use of the Cancel key quickly. Although in many situations, Cancel is essentially an undo or escape key, it can also be a Request if the Code key is not held down. Since Cancel and Request are the same physical key, whose function depends on whether or not the Code key is held down, we can see how new users could get into trouble. Suppose a user wishes to escape from an environment. All he needs to do is press the Code key and then the Cancel/Request key. But what if he presses Code too late and actually issues a Request?

Request causes the cursor to move to the Request line and allows the user to issue a request. However, the user who struck Request by mistake didn't want any of this, so he hits Cancel, which he supposes is the right thing to do. Unfortunately, Cancel in this context cancels only the request. Hence, after the failed Cancel and the successful Cancel the user is looking at the original screen. In his opinion, Cancel has failed twice. Later, he will probably learn about environments and will realize that thinking of Cancel only as escape is reckless. But for now the user simply concludes that Cancel is slippery—sometimes it works and sometimes it doesn't.

Even if we managed to get large quantities of common sense into the design process and to solve the troubles associated with all the problems in Table 1, we would still have to deal with the pervasive difficulties that result from interactions of systems characteristics and learner error. Naturally, we cannot reasonably envisage any application system that can prevent learners from making errors of the Cancel/Request sort. Therefore, if people are always going to make errors we need to motivate them to be active problem solvers *as they are learning*.

Active learning. Why is laissez-faire a problem for new users? Why would anyone faced with these disorienting, illusive, slippery, secretive, and paradoxical mystery makers even hope to be given guidance in their personal goals? One user I observed believed that the screen's Reset/Help light, which came on when she entered illegal characters or keypresses, was the machine telling her that in its view she needed some help: "It thinks I need help," she said. This perception was doubly unfortunate, since not only was the machine not worrying about her progress towards high-level goals, but also when she inadvertently got the light to go off by trying Help uncoded (Reset), she concluded that the machine felt she was now on the straight and narrow again. Too bad.

One answer to the laissez-faire puzzle is that while the systems are indeed fraught with troubles for the learner, they also implicitly promise to be traditional environments: the exercises of their self-instruction manuals promise this; their reference-book Help facilities promise this; and even their jargon, paradoxes, side effects, slipperiness, and all-around inscrutability promise this. However, the systems are *not* like traditional learning environments. These depend crucially on the special design skills of a human teacher, while the systems are notoriously stupid without even rudimentary problem-solving capabilties. Also, traditional learning environments fail most decisively to encourage active learning.[11]

Exploratory environments

So how does Adventure come off so well for people even though it causes all the problems we've discussed? Part of the answer is surely that Adventure is recreation whereas application systems are work. In our culture, we sharply isolate work from recreation, and we consequently may have quite different response modes to these two situations. Adventure is dealing in fantasies, while application systems are dealing in accounting, word processing, and the like. Text editors are not about dragons or treasure; they are about documents, printers, and libraries. (I may not be totally secure in this argument, since Adventure is often played by programmers, and other computer professionals and hobbyists, while application systems are typically learned and used by nonprogrammer adults. Possibly, these differences make comparison of the two experiences invalid, but to be certain, we would need a more empirical study, perhaps of secretaries learning Adventure.)

This contrast between work and play should not be viewed as monolithic, however; indeed, we can learn

much about work by examining play environments. We can see how particular aspects of recreation and work coexist in complex human activities and experiences and can use that knowledge to structure work environments that are based on the organization of play. What Adventure does *not* share with a typical application system makes it compelling to learn, even though it has some of the same problems that application systems suffer. This difference is what I call the *exploratory environment* (Table 2).

Ironically, Adventure is a parody of applications systems *because* it presents the learner with an exploratory environment. Consequently, the Adventure player is not frustrated by the laissez-faire problem, but rather *expects* it. Neither is the player frustrated, or beaten into passivity, by the other problems in Table 1; rather he fights back, actively extracting each secret from the game.

The key is motivation. In an exploratory environment the learning experience belongs to the learner. Extrinsic authority, like a teacher, manual, or system, is absent, making the learner the sole control.* Since extrinsic penalties are ruled out, the usual sort of learning anxiety just can't arise. Extrinsic rewards are also ruled out, which is not necessarily a bad thing.[16] Rather, the environment affords, encourages, and even demands conceptual and empirical experiment. We can have no unengaged learner in such an environment, and this motivational orientation overcomes the cognitive learning problems in Table 1.

*The consensus on learner control at the moment seems to be that it improves task-oriented motivation and may reduce anxiety but often slows down learning rate. Students tend to choose the easy problems over the difficult ones.[12-15] However, I caution against applying these conclusions to the context presented here. First, learner control as a real-world issue in instructional design may have quite a different complexion from what it has in these studies, which are very limited in scope. The most extensive study only tracked learning for 15 days of half-hour sessions. Second, most of these studies focused on learning unintegrated material by rote or on refining arithmetic skills. This focus is quite different from what is involved in learning to use application systems and in playing Adventure—namely, acquiring integrated conceptual structures. Finally, these studies focus exclusively on children, who would most likely be pretty bad at designing the character of their own instructional environments. Results from this type of study might be quite irrelevant to adult learning.

Table 2.
Typical characteristics of an exploratory environment.*

Responsiveness	When you do something, you get some feedback (at least informational).
Benchmarks	You can tell where you are within a given episode or session. You have the means for assessing achievement and development of skill.
Acceptable uncertainty	Being less than fully confident of your understanding and expertise is OK.
Safe conduct	You cannot do anything *too* wrong.
Learning by doing	You *do* so that you can learn to do: you design a plan; you do not merely follow a recipe.
Opportunity	Most of the things you learn to do work *everywhere*. You can reason out how to do many other things.
Taking charge	If progress stagnates, something new is suggested or happens spontaneously.
Control	*You* are in control, or at least have the illusion of being in control.

*The properties in Table 2—and for that matter in Table 1— are clearly a nonexhaustive, somewhat arbitrary set of contrasts and similarities. I make no claim to the contrary. I can't call on any substantive psychological theory of either computer games like Adventure or text editors to clarify my list, since no theory really exists. Hence, even if I'm basically right about the learning benefits of so-called exploratory environments, I may be overlooking the role psychological factors play and may have the right conclusions for the wrong reasons.

Responsiveness. Even though the screen display is often empty and static, Adventure tends to provide *some* reaction for every user action; it is not always helpful or supportive but it is at least some reaction. The player cannot issue a command that will fail to elicit some message (however mysterious) from the system. If you issue a command that is syntactically and situationally correct, you will see some sensible consequences immediately. If you issue a command that is *incorrect,* you will still see some consequence—even if it is only "Huh?" This may not seem like a lot, but it keeps players responding. The one thing that a learner cannot seem to overcome is the silent treatment. Informing the learner of consequences *on the occasion of learning* can help him recognize when to engage this learning in the future.[17,18]

For a prime example of *un*responsiveness, pick any text editor. Even a major command like Print often elicits nothing from the system. On one text editor, a print message is sent to the user but held in a message reservoir. The unsophisticated user doesn't recognize what has happened and quite often doesn't realize that anything has been printed. If the user fails to examine the print message when it is sent and waits—or simply discovers it later—he might end up getting a print confirmation message for File 1, having just created or even printed File 2.)

Benchmarks. Adventure allows the player to assess his progress in skill and achievement. The continuum from novice to seasoned expert is finely and intricately segmented. At any point in an Adventure session, the player may get a score, reflecting the success he has had in dispatching the various tasks of the game scenario. This score is also a basis for classifying the player at the end of an adventure. The challenge of such clear mappings of personal progress increases the learner's task-oriented motivation.[4,11,19]

Again, we have a contrast because application systems eschew benchmarks. For example, when a user first signs on, the text editor does not display any message like "You have successfully signed on"; rather, it immediately brings up the task selection menu. The sign-on challenge, "do you want to use the system and are you authorized to do so?" is replaced by the task challenge, "What do you want to do?" The shrewd user will of course recognize that a new menu *is* a benchmark, but then again the shrewd user isn't the one who needs to know. Perhaps the saddest case of this sort is signing off from the first session. When a user successfully signs off, the *sign-on* menu comes up immediately. Again the experienced user knows that the system is saying good-bye and hello, but the new user is quite apt to see it only as another request to fill in an inscrutable menu, which he then does—inadvertently signing back on again. Is there no end to this misery?

Acceptable uncertainty. In Adventure, having low, even virtually no, confidence in one's strategy or understanding of the game's inner workings doesn't matter a bit. In fact it adds to the fun, to the uncertainty, and to the indeterminacy (as the player's mental model of the game shifts and changes dynamically). Indeed, the game's interface is written with the obvious expectation that players will often be struggling. In contrast, the interface and training for a real application system presumes and implies that the user will succeed perfectly on the first try. No wonder our subjects have exclaimed, "It makes me feel stupid!!"

The new user of a text editor gets messages on various wavelengths that the experience is *not* fun. Unfortunately, the general context in which work is performed in our society quite typically presupposes a disjoint relation to fun. True, the system is not to blame but it doesn't do even a little to change this learning bias. Training materials are stuffy, stilted, and oriented towards "correct performance," never towards personal growth and discovery.

People often prefer to be uncertain, at least of their actual possibilities of success or failure. Weiner[20] found that people prefer goal-driven activities in which their probability of success is near 0.5. Berlyne[21] argues extensively that the conceptual conflict due to the uncertainty of outcome is a fundamental source of curiosity. In other words, games like Adventure keep players interested by keeping them uncertain.

Safe conduct. The intolerance with which application systems view diffidence suggests to learners that mistakes will be costly. Text editor learners we have studied invariably ask whether they might accidentally destroy something during an exercise. The policy of our research group is to tell them "no" to put them at ease, but with no trouble at all, a new user can obliterate all the files. The machine itself is saying "look out!" through its interactions with them. For example, in one editor, the user may enter DEL/ anywhere in a file to delete the current line—anywhere, that is, except the end-of-file line. If the user enters DEL/ here, the whole file will be deleted. One user who did this, quickly issuing a FILE command in panic, compounded the problem by saving a consequently empty file instead of his whole day's work.

In Adventure, players cannot really do anything too wrong. The pirate may get your treasure, the dwarves may indeed get you, but the game flow will survive. Players can't accidentally destroy their game, and they know it. This property of Adventure is what Moore and Anderson[11] called the "autotelic principle." This principle states that in initial stages of learning, the learner should be protected from the consequences of mistakes. Early learning should be enjoyed as much as possible for its own sake, and costly mistakes spoil this.

Learning by doing. In Adventure, if you want to know about it, you do it. A miniscule part of its total instruction is what we call "passive." The player customizes an exploratory foray, actively carries it out, and draws what conclusions he can. Nothing comes for free, and almost nothing comes passively. Empirical data support this concept also. Animals in maze learning experiments learn faster and more thoroughly when they actively traverse a maze than when they are passively carted around the maze. A person learns the layout of a new town much faster if he is the driver (active), rather than rider (passive).

Current training programs for application systems rely heavily on traditional passive modes of instruction, such as reading and classes. Fortunately, largely because of the desire to reduce costs, self-instruction approaches are becoming prevalent—but even these are passive. The learner merely follows instructions and does not exert any prerogative as to what to do or how to do it. Moreover, self-instruction approaches apparently fail anyway, and the new user's co-workers end up providing the instruction. When learners were restricted to the self-instruction materials, they had severe and wide-ranging learning problems. A learner who succeeded in following an exercise passively often asked, "What did I do?" upon completing it.

Opportunity. By providing a command architecture that affords action, Adventure supports a learn-by-doing orientation. Virtually all canonical Adventure commands can be issued anywhere in the system. Quite typically, a player can deduce a command, or a proper course of action, on the basis of other experience with the game, like treasure taking and escape. Consequently, the player can always do something; maybe he can't win, maybe he can't do what he wants, and maybe he can't do anything that seems particularly rewarding or useful—but he can do something. Moore and Anderson[11] describe this extendable consistency as a "productive principle."

Things are grimmer with application systems. Much more often than you would think, novice users find themselves in a part of the system from which only *one* command can deliver them, and they don't know that command. Being in this situation cannot really be described, and it isn't pleasant to watch. Of course, they can always switch the machine off and escape to a clean start, but this strategy isn't elegant, and it nullifies at least some of the potential for learning from the previous segment of the session. A "flatter" command architecture might be useful here, since the system is really a myriad of small compartments, each with its own special meanings for commands.

Taking charge. Problems like those in Table 1 can often cause a temporary conceptual paralysis. When the consequences of the user's actions are routinely uninterpretable, he is encouraged to simply stop taking action at all. Adventure does not allow this to happen. If the player founders even a bit, the game will actively intervene:

Adventure: You are in a maze of twisty little passages, all alike.
Carroll: (about 100 moves ensue)
Adventure: Do you need help getting out of the maze?
Carroll: Yes.

> Adventure: I am prepared to give you a hint, but it will cost you 4 points. Do you want the hint?
> Carroll: Yes.
> Adventure: You can make the passages look less alike by dropping things.

Indeed, the game may do more than merely suggest new possibilities. It may actually reconfigure the player's immediate situation. Thus, an explorer who is aimlessly wandering the caves carrying treasure may suddenly be robbed by the cave's Pirate:

> There are faint rustling noises in the darkness behind you. Out from the shadows behind you pounces a bearded pirate! "Har, Har," he chortles, "I'll just take all this booty and hide it away with me chest deep in the maze." He takes your treasure and disappears in the gloom.

The player doesn't ask for this and doesn't want it; it just happens. The chance of being attacked by marauding dwarfs also increases when the player is stuck.

Control. The player learns by doing or can always do something in Adventure, but more important is that the player *directs* what goes on—as the chief instigator. Perhaps this illusion of overall control is what fundamentally drives the aggression fantasies of games like Adventure. Imagine an adventure in which the player is a base camp and can only send up mail and provisions and give advice to the real explorer—who may or may not take it. That scenario just wouldn't do. Zimbardo[22] showed that even the illusion of control can increase motivation to perform a task.

An important distinction about learners of application systems is that not only are they not in control but they are *made to feel* that way. In harsher terms, they are made to feel incompetent. Within the first ten minutes a new user is staring at a message, "Parameter omitted or not valid." She is a very bright woman; she even knows what *parameter* means. However, she does not know what it means here, and indeed it can mean any of the several things displayed on the screen. She explores various of these and concludes that parameter must be the boxlike character directly above the cursor, which identifies a free-key field. She is, of course, wrong, and any illusions she might have had about being in control have been discouraged. (Maybe system designers like to use words like parameter and default so that *they* can feel they're in control.)

So there you have it: people seem to like learning in exploratory environments. Of course most of my evidence is in anecdote and, as such, does not make the case that applications systems ought to be modeled on computer games. Indeed, I am not arguing that they should be. Adventure, for example, has no "undo" key. However, our studies of people learning to use text editors strongly indicates that some such function would substantially ease many of the difficulties learners experience. I am merely trying to say that something might be learned from a study of what's going on in computer games like Adventure and new insights could be applied to the design of application systems. At the very least, we could get some additional perspective on what makes application systems difficult. We might even find new design techniques that make them easier to learn to use. Only a fool looks a gift horse in the mouth.

Implications

The properties that define exploratory environments transform the problems in Table 1—shared by Adventure *and* typical application systems—into challenges for the learner. Now the difference between a challenge and an obstacle can be fine, but as writers like Malone[4,23] and Moore and Anderson[11] extensively argue, it hinges on the character of the learning environment. If the learner's motivation is task oriented and if the learner feels in control of the situation, then obstacles can become challenges. (Malone, in fact, examined computer games from this perspective, but focused on explicitly pedagogical skill games, like Breakout and Darts, rather than on conceptual maze learning games like Adventure.[4])

This transformation of obstacles into challenges can happen in an exploratory environment. The player is immediately apprised whenever he manages to do something. Through explicit functions, the learner is kept informed of his progress and achievement—but he also always knows that it doesn't matter too much if he does make mistakes. A dynamic world is presented in which the learner not only *can* act—without worry about untoward consequence—but *must* act: the learner's exploration is what this world is about.

A person traversing an exploratory environment *expects* laissez-faire and illusiveness; regards paradox, side effects, and slipperiness as intriguing potential keys to the inner logic of the environment; and is unperturbed by disorientation, emptiness, and mystery messages. Each new problem is an intimate and preemptive invitation to learn. In such an environment, the learning truly belongs to the learner.

In any case, if we assume that learners will always make some errors—no matter how good our cognitive solutions to interface design are—then the issue becomes one of motivating learners to actively solve the problems they encounter.

The game metaphor. Adventure may provide a metaphor from which to construct interface concepts for application systems.[24,25] The logic for this is straightforward: the game is similar to the application systems, but the *potential* "troubles" shared by both are *real* troubles only in the systems, not in the game. So why not look at why the two differ and try to make the systems more like the game? We could really go overboard: text editors that award points, or tell you that you're "dead" if you try to print an empty file—but why be silly? To really exploit the game metaphor in making application systems easier to learn will clearly require a considerable and systematic research effort in software psychology and system design. It is serious business.

To get an inkling, in a very small way, of what the metaphor could come to, we may only have to ask people to explore a conventional text editor as if it were an

Adventure-like maze. In ongoing work at the research center, my colleagues and I have been attempting to project the "typewriter metaphor" to ground the subject's understanding of the task and the "game metaphor" to ground the subject's task-oriented motivation. We instructed one typist to read the following instructions on exploring a computer maze:

> We are studying how people learn to use small computing systems. We are interested in how a person goes about such a learning task when *no formal instruction at all* is provided. What we want you to do is to interact just as you please with the computer, to explore its capacities, and to learn to do interesting things with it (HINT: this computer is in many ways a sort of super-typewriter). This, should you decide to accept, is your mission.
>
> We want to emphasize several aspects of this learning task. Most importantly, you should not worry about doing the "wrong" thing. In this situation, the only wrong thing you can do is nothing. The most important thing to do is explore: the computer is in many ways like a maze and you have to find out how to get around (getting lost a little on the way is inevitable). This mazelike quality also means that you should be willing to change your goals from time to time. We have found that in the course of trying to do X, a person very often stumbles onto Y—but you have to be willing to give up on X for the moment!
>
> This super-typewriter-maze idea may sound like a video game in disguise. In fact, we would like you to think of this learning experience as a sort of game. The more you explore of the computer-maze, the higher your "score." If you get stuck, we will answer direct and specific questions. Don't waste your questions, for we will answer only 10 (ten) throughout the entire session. But by all means ask for help when you really need it (otherwise you'll be holding yourself back from further exploration). If you can get by without a question—Bravo!
>
> Because we want to learn how people approach this sort of learning task, we will ask you to try to "think aloud" (as thoughts occur to you, just speak them out loud). To help this along, from time to time we will ask you "What are you thinking just now?" and the like. (You'll get used to it.) In the service of thinking aloud, feel free to ask "rhetorical" questions: we won't charge you for telling us what questions are on your mind—unless you ask us to answer them!
>
> Finally, here are two HINTS.
>
> 1. Try these keys for interesting results: CODE, ENTER, HELP, RESET, REQST, VAR ADV (we will show you where each of these keys is on the keyboard).
> 2. The computer "believes" that your Operator Name is your first name.

These instructions were presented instead of the normal training materials we had used with other subjects. The trade-off here is that while the normal-case subjects were passively led through a series of exercises that presented them with knowledge about the system, our explorer subject was forced to be active, to discover knowledge about the system, or learn nothing. The other important difference is that while normal-case subjects were run in a 16-hour learning procedure, our explorer subject was run for only four hours.

In many ways, the explorer's experience was comparable to that of our other subjects. She failed to notice an indicator light, then tried unsucessfully to type in a reset condition, and became frustrated. She failed to hold down the Code key consistently and was then surprised at the "inconsistency" of the system. She wrongly generalized Cancel and Enter as commands that get you to the next menu—even though they in fact do, but only occasionally and as side effects, etc., etc., etc. These similarities are not really surprising. The system is difficult to learn to use—as work with normal-case subjects made clear. We would have had quite a finding indeed if simply taking away the training materials had made learning easier.

The character of what the explorer did was somewhat different from that for normal-case subjects. The explorer used Help more in her 4-hour session than any of our other subjects had in 16 hours. This difference is important, since the training manual explicitly tells learners how to use Help, and encourages its use. One of the four training modules we used for our prior subjects was concerned centrally with the use of Help. Her use of Help—and of all the information displayed on screens—was qualitatively different too. She read it.

Our normal-case subjects, who had the manual to more or less lead them by the hand, often didn't "see" the screen displays. The most striking case was a subject who, after three days of working with the system, insisted that she had never seen the sign-on menu; her reason was that in trying to passively follow the manual, she had been trying not to look at the screen! The explorer got into areas of the system that the training manual prevented other subjects from discovering. She learned about text blocks, document archiving, margin changes (line format changes), and document duplication. Of course, she also had new problems: because she read the screen information, she was really hurt when it was incomplete, turgid and jargony, or misleading.

Nonconclusions

Generally speaking, simple answers do more harm than good in behavioral science. Luckily, from this vantage point, we cannot generate any simple answers from this discussion. My analysis is fully consistent with the view that an Adventure metaphor for application systems could make them easier to learn to use. But the indications for this are still indirect and our confidence must be qualified accordingly. And even if the empirical and theoretical issues were resolved sharply and favorably, we would still not know to what degree such a metaphor should be implemented in the actual development of in-

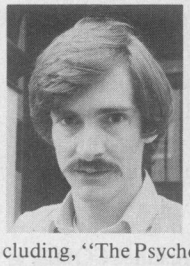
terface designs and instructional materials.[26,27] In guideline definition, we still have so far to go that it's hard to tell when we've taken a step. ■

Acknowledgments

I am grateful to Norman Brenner, Colette Daiute, Fred Damerau, Clayton Lewis, Robert Mack, Don Nix, and John Thomas for helpful comments on earlier versions of this paper. Clayton Lewis, in particular, said several things that totally changed the paper. The original Adventure game was written by Willie Crowther at MIT. It was later revised and extended by Don Woods at the Stanford AI Lab.

References

1. J. M. Carroll and R. Mack, "Learning To Use a Word Processor: By Doing, by Thinking, and by Knowing," to appear in *Human Factors in Computer Systems,* J. Thomas and M. Schneider, eds., Ablex, Norwood, New Jersey.
2. J. M. Carroll and R. Mack, "Actively Learning to Use a Word Processor," manuscript, 1982b, to appear in *Cognitive Aspects of Skilled Typewriting*, W. Cooper, ed., Springer-Verlag, New York.
3. R. Mack, C. Lewis, and J. Carroll, *Learning To Use Office Systems: Problems and Prospects,* manuscript, 1982.
4. T. W. Malone, *What Makes Things Fun To Learn? A Study Of Intrinsically Motivating Computer Games,* Xerox PARC Report CIS-7, 1980, p. 55.
5. C. A. Daiute, "Where Has All the Paper Gone?" *Electronic Learning*, Jan. 1982.
6. J. D. Gould, "Composing Letters with Computer-Based Text Editors," *Human Factors*, Vol. 23, 1981, pp. 593-606.
7. R. Bott, *A Study of Complex Learning: Theory and Methodology,* CHIP report 82, University of California, La Jolla, 1979.
8. A. Bandura and D. Schunk, *Cultivating Competence, Self-Efficacy, and Intrinsic Interest Through Proximal Self-Motivation,* Stanford University, 1980.
9. M. Csikszentmihalyi, *Beyond Boredom and Anxiety*, Jossey-Bass, San Francisco, 1975.
10. J. M. Carroll, "Learning, Using, and Designing Command Paradigms," *Human Learning: J. Practical Research and Application,* Vol. 1, 1982, pp. 31-62.
11. O. K. Moore and A. R. Anderson, "Some Principles for the Design of Clarifying Educational Environments," *Handbook of Socialization Theory and Research,* D. Goslin, ed., Rand McNally, New York, 1969.
12. M. D. Fisher, et al., "Effects of Student Control and Choice on Engagement in a CAI Arithmetic Task in a Low-Income School," *J. Educational Psychology*, Vol. 67, 1975, pp. 776-783.
13. J. P. Fry, "Interactive Relationship Between Inquisitiveness and Student Control of Instruction," *J. Educational Psychology*, Vol. 63, 1972, pp. 459-465.
14. J. B. Hansen, "Effects of Feedback, Learner Control, and Cognitive Abilities on State Anxiety and Performance in a Computer-Assisted Instruction Task," *J. Educational Psychology,* Vol. 66, 1974, pp. 247-254.
15. E. R. Steinberg, "Review of Student Control in Computer-Assisted Instruction," *J. Computer-Based Instruction*, Vol. 3, 1977, pp. 84-90.
16. M. R. Lepper, D. Greene, and R. E. Nisbett, "Undermining Children's Intrinsic Interest with Extrinsic Rewards: A Test of the Overjustification Hypothesis," *J. Personality and Social Psychology,* Vol. 28, 1973, pp. 129-137.
17. E. M. Abernathy, "The Effect of Changed Environmental Conditions upon the Results of College Examinations," *J. Psychology,* Vol. 10, 1940, pp. 293-301.
18. J. A. McGeoach, *The Psychology of Human Learning,* Longmans Green, New York, 1942, pp. 501-505.
19. E. L. Deci, *Intrinsic Motivation,* Plenum Press, New York, 1975.
20. B. Weiner, *Human Motivation*, Rand McNally, Chicago, 1980.
21. D. E. Berlyne, *Structure and Direction in Thinking,* John Wiley & Sons, New York, 1965.
22. P. G. Zimbardo, *Cognitive Control of Motivation,* Scott, Foresman & Co., Glenview, Illinois, 1969.
23. T. W. Malone, "What Makes Computer Games Fun?" *Byte*, Dec. 1981, pp. 258-277.
24. J. M. Carroll and J. C. Thomas, "Metaphor and the Cognitive Representation of Computing Systems," *IEEE Trans. Systems, Man, and Cybernetics,* Vol. SMC-12, 1982, pp. 107-116.
25. H. G. Petrie, "Metaphor and Learning," *Metaphor and Thought,* A. Ortony, ed., Cambridge University Press, New York, 1979.
26. H. A. Simon, *Sciences of the Artificial,* 2nd ed., MIT Press, Cambridge, Massachusetts, 1981.
27. J. C. Thomas and J. M. Carroll, "Human Factors in Communication," *IBM Systems J.*, Vol. 20, 1981, pp. 237-263.



John M. Carroll has been a research staff member in the Computer Science Department of the IBM Thomas J. Watson Research Center in Yorktown Heights, New York, since 1977. His research is in the analysis of basic cognitive skills and capacities that underly complex human behavior and experience. He is the author of *Toward A Structural Psychology of Cinema* and of 45 technical papers including, "The Psychological Study of Design," "Creative Analogy and Language Evolution," "Human Factors in Communication," and "Structure in Visual Communication." He was also coeditor of *Talking Minds* (in press), the proceedings of an interdisciplinary conference held at the Watson Research Center.

Carroll is a member of the Linguistics Society of America and the Psychonomic Society. He received a BA in mathematics and a BA in information science from Lehigh University in 1972 and a PhD in experimental psychology from Columbia University in 1976.