Heuristics for Designing Enjoyable User Interfaces: Lessons from Computer Games

Thomas W. Malone
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

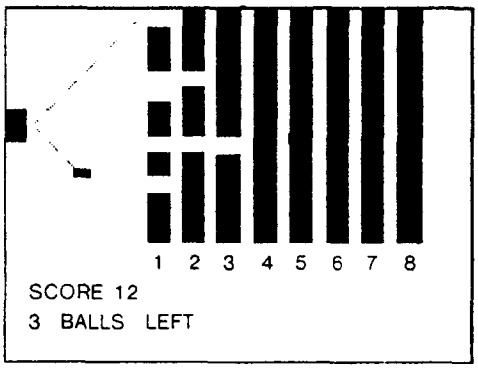
In this paper, I will discuss two questions: (1) Why are computer games so captivating? and (2) How can the features that make computer games captivating be used to make other user interfaces interesting and enjoyable to use?

After briefly summarizing several studies of what makes computer games fun, I will discuss some guidelines for designing enjoyable user interfaces. Even though I will focus primarily on what makes systems enjoyable, I will suggest how some of the same features that make systems enjoyable can also make them easier to learn and to use.

STUDIES OF ENJOYABLE COMPUTER GAMES

To help determine what makes computer games so captivating, I conducted three empirical studies of what people like about the games. All of these studies are described in more detail elsewhere ([8], [9], [10]) and are only briefly summarized here. The primary purpose of these studies was to help design highly motivating instructional environments, but they also have important implications for designing other user interfaces.

Darts. To illustrate the methodology used, I will briefly describe one of the studies. This experiment analyzed a game called Darts that was designed to teach elementary students about fractions [4]. In the version of the game used, three balloons appear at random places on a number line on the screen and players try to guess the positions of the balloons (see Figure 1). They guess by typing in mixed numbers (whole numbers and/or fractions), and after each guess an arrow shoots across the screen to the position specified. If the guess is right, the arrow pops the balloon. If wrong, the arrow remains on the screen and the player gets to keep shooting until all the balloons are popped. Circus music is played at the beginning of the game and if all three balloons in a round are popped in four tries or fewer, a short song is played after the round.



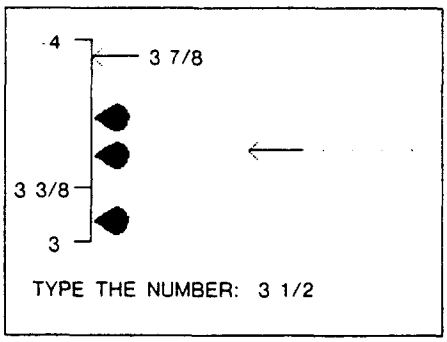


Figure 1. Display format for the Darts game.

To find out what features contribute most to the appeal of this game, I constructed 8 different versions of the game by taking out, one at a time, features that were presumably motivational. The features removed included: the music, the scorekeeping, the fantasy of arrows popping balloons, and different kinds of feedback (see Figure 2).

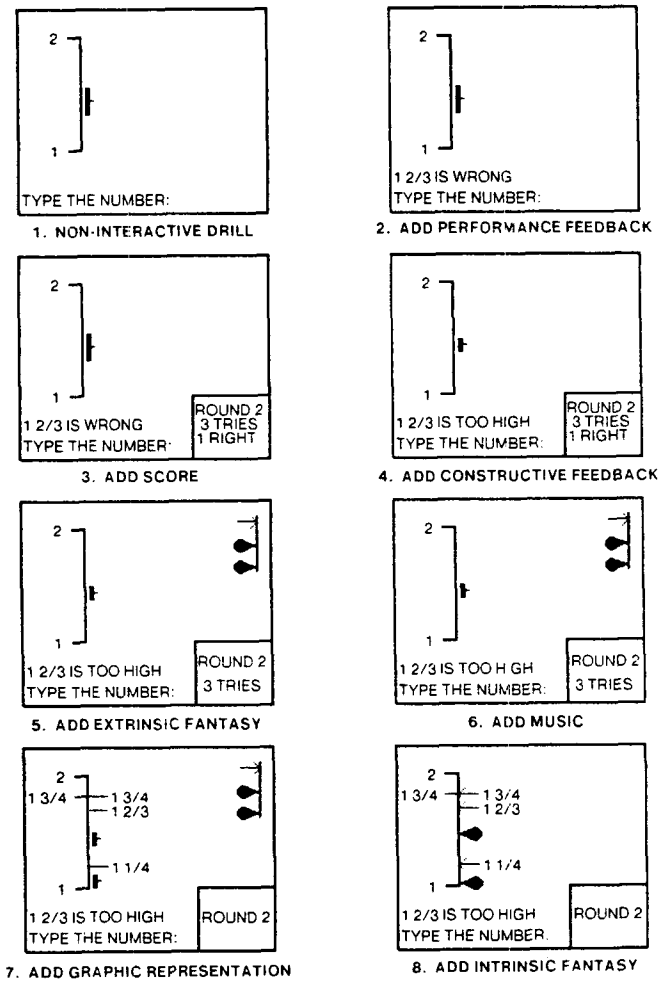


Figure 2. Different versions of the Darts game.

Eighty fifth grade students were each assigned to one of the 8 versions and then allowed to play with either their version of Darts or with a version of Hangman that was the same for all students. The primary measure of appeal of the different versions was how long students played with their version of Darts in comparison to Hangman. This measure was also highly correlated with how well students said they liked the game at the end ($r=.30$, $p<.01$).

Somewhat surprisingly, there was a significant difference between boys and girls in what they liked about the game. An analysis of variance of the time spent playing Darts revealed significant effects of condition ($F(7,48)=2.21$, $p<.05$) and of the sex by condition interaction ($F(7,48)=4.84$, $p<.001$). A detailed interpretation of the differences (shown in Table 1) is given in [8] and [10]. Briefly, the girls' dislike of the intrinsic fantasy of arrows and balloons (Condition 7 vs. 8) appears to be because they dislike the arrows and balloons fantasy in the first place and the fantasy is more salient in the intrinsic than the extrinsic version. Furthermore, the differences between Conditions 3 and 4 for boys and between Conditions 5 and 6 for girls appear to be less reliable than the others because they are not significant when the time measures are scaled according to a plausible model of choice behavior.

In summary, the primary result of this experiment was that boys liked the fantasy of arrows popping balloons, and girls appeared to dislike this fantasy. The results also showed that fantasy made more difference in the appeal of the game than did simple feedback. In other words, even though responsiveness is often mentioned as an important reason why computers are captivating, the simple feedback in the game was not as important as the fantasy in making this game fun.

I think the most important implication of this experiment is that fantasies can be very important in creating intrinsically motivating environments but that, unless the fantasies are carefully chosen to appeal to the target audience, they may actually make the environment less interesting rather than more.

Table 1
Interest in different versions of the Darts game

Condition	*Time playing Darts (0 - 40 mins.)*	
	Boys	*Girls*
1. Non-interactive drill	20.5	15.5
2. Add performance feedback	18.8	20.2
3. Add scoring	24.2	19.8
4. Add constructive feedback	16.2 *	22.2
5. Add extrinsic fantasy	25.8 *	20.8
6. Add music	21.8	30.0 *
7. Add graphic representation	28.3	29.8
8. Add intrinsic fantasy	34.5	19.8 **
Average	23.4	22.0

* $p < .05$, for comparison with previous condition
** $p < .01$, for comparison with previous condition

Other studies. Another game, called Breakout, was studied in a similar way. In this game, the player controls a paddle and tries to hit a ball so that it knocks all the bricks out of a wall. The visually compelling goal of knocking bricks out of the wall was found to be the most important of the features varied in this game. Finally, in a survey of the computer game preferences of 65 elementary school students, the features that were most strongly correlated with game popularity were the presence of an explicit goal, score-keeping, audio effects, and randomness.

IMPLICATIONS FOR DESIGNING ENJOYABLE USER INTERFACES

In this section, I will outline a general framework for analyzing the appeal of computer systems based on three categories: challenge, fantasy, and curiosity (see Table 2). The primary purpose of this framework is to serve as a checklist of heuristics for designing enjoyable user interfaces. One purpose of this paper is to show how this framework, which was developed elsewhere [10] for analyzing instructional environments, can be applied to more general user interfaces.

The motivational processes discussed in this paper are, in many ways, less well-understood and subject to much larger individual differences than many of the cognitive processes involved in human-computer interactions. Accordingly, the heuristics in this section should be viewed as suggestions, not as requirements. Many of them are only appropriate for some people in some situations and they must be applied with care.

Toys and tools. It is important, in describing this framework, to distinguish two different uses of computing systems: *toys*--systems used for their own sake with no external goal (e.g., games), and *tools*--systems used as a means to acheive an external goal (e.g., text editors, programming languages, etc.).

As discussed below, good toys and good tools are similar in the ways they can use fantasy and curiosity, but in an important way they are opposite with respect to their requirements for challenge. Since most user interfaces are for tools, not toys, much of the motivation for using the system depends on the user's motivation to achieve the external goal. In cases where the external goal is not highly motivating (e.g., is routine and boring), the toy-like features discussed below can be especially useful in making the activity enjoyable.

Challenge

Goal. For an activity to be challenging, it needs to have a *goal whose outcome is uncertain.* As described above, computer games without explicit or easily generated goals were less enjoyable than games with goals. In other words, a challenging toy must either build in a goal or be such that users can easily create their own goals for its use. A good tool, on the other hand, is designed to achieve goals that are already present in the external task.

For both toys and tools, however, users need some kind of *performance feedback* to know how well they are achieving their goals. In games, this performance feedback is provided by things like the missing bricks in Breakout and the position of the incorrect arrows on the number line in Darts. There may be similar ways to incorporate

Table 2
Heuristics for Designing Enjoyable User Interfaces

I. Challenge

A. *Goal.* Is there a clear *goal* in the activity? Does the interface provide *performance feedback* about how close the user is to achieving the goal?

B. *Uncertain outcome.* Is the outcome of reaching the goal uncertain?

1. Does the activity have a *variable difficulty level*? For example, does the interface have *successive layers of complexity*?

2. Does the activity have *multiple level goals?* For example, does the interface include *score-keeping*?

II. Fantasy

A. Does the interface embody *emotionally appealling* fantasies?

B. Does the interface embody *metaphors* with physical or other systems that the user already understands?

III. Curiosity

A. Does the activity provide an *optimal level of informational complexity?*

1. Does the interface use *audio and visual effects:* (a) as decoration, (b) to enhance fantasy, and (c) as a representation system?

2. Does the interface use *randomness* in a way that adds variety without making tools unreliable?

3. Does the interface use *humor* appropriately?

B. Does the interface capitalize on the users' desire to have *"well-formed" knowledge structures*? Does it introduce new information when users see that their existing knowledge is: (1) *incomplete*, (2) *inconsistent*, or (2) *unparsimonious*?

performance feedback for the external task into tools. For example, the Writer's Workbench developed at Bell Laboratories [7] measures various stylistic features of manuscripts such as word length, sentence length, percentage of sentences using passive voice, and so forth. These rudimentary kinds of performance feedback for the external goal of producing a readable manuscript may enhance the challenge of using the tool.

Uncertain outcome. The most important difference between toys and tools occurs with respect to the uncertainty of outcome of reaching a goal. If a user is either certain to achieve a goal or certain *not* to achieve it, the activity will not be very challenging. For an activity to be challenging, the outcome of achieving the goal must be uncertain.

One way of making the outcome of a computer game uncertain for a wide range of players, or for the same player over time, is to have a *variable difficulty level.* For example, in the Breakout game, after a player hits the ball correctly five times in a row, the ball speeds up. As Nolan Bushnell, the founder of Atari, Inc., has been quoted as saying, "A good game should to easy to learn, but difficult to master."

A good tool, on the other hand, should be both easy to learn and easy to master. Since the outcome of the external goal (writing a good letter, getting a program to work) is already uncertain, the tool itself should be reliable, efficient, and usually "invisible". In other words, the tool users should be able to focus most of their attention on the uncertain external goal, not on the use of the tool itself. In a sense, a good game is intentionally made difficult to play, but a tool should be made as easy as possible to use. This distinction helps explain why some users of complex systems may enjoy mastering tools that are extremely difficult to use. To the extent that these users are treating the systems as toys rather than tools, the difficulty increases the challenge and therefore the pleasure of using the systems.

In spite of the differences between toys and tools, there is a way tools can use variable difficulty levels to increase challenge and, at the same time, probably improve learnability as well. I have heard many system design arguments in which the fundamental conflict is between, on the one hand, a desire to have the system be simple and easy to learn for beginning users, and on the other hand, the desire to have it be powerful and flexible for experienced users. Many of these arguments could be resolved by consciously building in a logical progression of *increasingly complex microworlds* for users at different levels of expertise [5].

For example, a multi-layered text editor could be designed so that beginning users need only a few simple commands and more advanced users can use more complicated and more powerful features of the system. Ideally, this system should be internally consistent at each level so that the error messages for users of the first level would never assume any knowledge of concepts used only in more advanced levels. In fact, some commands that might make sense if made by an advanced user should probably be treated as errors if made by a beginning user.

The point here is that a multi-layered system could not only help resolve the trade-off between simplicity and power, it could also enhance the challenge of using the system. Users could derive self-esteem and pleasure from successively mastering more and more advanced layers of the system, and this kind of pleasure might be more frequent if the layers are made an explict part of the system.

Another way of providing uncertain outcome in computer games is to have *multiple level goals* all present in the environment at the same time. For example, in the Breakout game, long before there is any hope of a beginning player breaking out all the bricks, the player can still be challenged by lower level goals like breaking out any brick in the third row or breaking out all the bricks in the first row. Or in the Darts game, players who are certain they can pop all the balloons can still be challenged by trying to pop all the balloons in as few tries as possible. In general, *score-keeping* and *timed responses* are two common ways of enhancing multiple level goals in computer games.

It may be possible to incorporate similar kinds of multiple level goals into tools, as well. For example, I think some users of a text editing system would be challenged by having the system automatically maintain scores like typing speed or number of corrections made. If the text-editing task is boring or routine for the user, this challenge might increase the pleasure of using the system. (It would almost certainly *not* increase the pleasure of using the system, however, if such scores were used for surveillance by organizational superiors, however.)

Another way of providing multiple level goals in a system is by having a lot of user programming capabilities. If users can write procedures to do subcomponents of their routine on-line tasks, then they can continue to be challenged by trying to make their system more efficient for the tasks they do. For example, if a text-editing system allows users to define their own macros, people who prepare many similar documents can be challenged by constructing macros to make this process more efficient.

Fantasy

Fantasy is probably the most important feature of computer games that can be usefully included in other user interfaces. By a system with fantasy, I mean a system that evokes mental images of physical objects or social situations that are not actually present. For example, the Breakout and Darts games evoke images of physical objects like balls, bricks, darts, and balloons; and the omnipresent computer Adventure game evokes images of caves, dwarves, birds, and so forth. I think fantasies have two important aspects for designing user interfaces: *emotions* and *metaphors*.

Emotions

Fantasies in computer games almost certainly derive some of their appeal from the emotional needs they help to satisfy in the people who play them. It is very difficult to know what emotional needs different people have and how these needs might be partially met by computer games. As the Darts experiment described above suggests, there are large differences among people in what fantasies they find appealing. Designers of computer systems that embody fantasies should either be very careful to pick fantasies that appeal to their target audience or they should provide several fantasies for the same system so that different people can select different fantasies.

One use of fantasy in computer systems might be to give different "personalities" to different parts of a system. For example, the operating system might have one personality, different application programs might have other personalities, and file servers on a network might have still other characteristics. But the personalities of the different parts of the system could be different for different users. Some users might like to work in a world of wizards, dragons, and trolls, others might prefer a world of dogs, cats, and rabbits, or even a world populated by characters from Star Trek or Charlie's Angels. Not only could these fantasies increase the emotional appeal of the systems, they could also be useful metaphors to help users learn the difference between different parts of a system--something that is not at all trivial for beginning users.

Carroll and Thomas [3] suggest another use of fantasy in "reframing" routine information processing tasks to make them more interesting. For example, they suggest that certain kinds of factory control operations (e.g., monitoring a steam engine) could be presented to the user as more captivating "virtual tasks" such as flying an airplane full of passengers onto a dangerous landing field. Measurements in the factory control space could be translated into the airplane metaphor, and actions taken in the airplane fantasy could be translated into actions in the factory.

This kind of reframing would presumably be appropriate only if the original task was boringly routine. Fantasies could make such routine tasks more enjoyable. But unless the outcome of reaching the goal is made uncertain (e.g., with an adjustable difficulty level or multiple level goals), the fantasy tasks could become boring as well.

Metaphors

In addition to being emotionally appealing, fantasies that are analogous to things with which the users are already familiar, can help make the systems easier to learn and use (see [3] and [6] for extended discussions of this point). For example, I think one of the reasons for the popularity of the VisiCalc system [2] is the fact that the program is very analogous to the kind of paper "spread sheet" that was already widely used by many of the business analysts who purchased VisiCalc systems.

The user interface for the Xerox Star workstation is another example of a system that makes extensive use of metaphors. Much of the manipulation of information takes place by moving icons around on a "desktop" that is simulated on the screen. The icons are pictorial representations of familiar objects like in-baskets, file folders, and filing cabinets. To the extent that this fantasy is analogous to real desktops, it presumably makes the system easier to learn and use.

Curiosity

The final category of features that make computer games appealing includes features that evoke the users' curiosity. Environments can evoke curiosity by providing an *optimal level of informational complexity* ([1] and [11]). In other words, the environments should be neither too complicated nor too simple with respect to the user's existing knowledge. They should be *novel* and *surprising*, but not completely incomprehensible. In general, an optimally complex environment will be one where the learner knows enough to have expectations about what will happen, but where these expectations are sometimes unmet.

One important way computer games evoke what might be called *sensory curiosity* is by using audio and visual effects. Audio and visual effects can be used (1) as decoration, (2) to enhance fantasy, and, perhaps most importantly, (3) as a representation system. Examples of using audio and visual effects as representation systems include (1) using different tones for errors and for successful entries (2) using graphs instead of numbers, and (3) using icons to represent different parts of a system (such as in-baskets and out-baskets) and different commands.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.