tive designs. We repeat: Design should be left to designers!" [1].

To further clarify, let's take an example of one of the concepts listed above—pace. We've defined pace in the past as "the rate at which players experience new challenges and novel game details."[2]. This is a very high-level definition, but spans across most, if not all, game genres. The way that a game designer will address pace is going to vary widely depending on the genre, their vision, and the experience they want to create. Let's look at two examples.

1. A tennis game. Pace can manifest itself in a number of ways, including the length and number of the cut scenes (or short movie clips or replays) that occur between points and between games. Pace can also manifest itself with ball and player movement speed or the mechanics for hitting the ball. Commercially successful tennis games have varied the pace on both of these dimensions. Therefore, before we know which aspect of pace to focus on, we need to understand the vision of the game designer. If the vision is a frenetic, high-action packed game (as opposed to a simulation), then we may focus on usability and gameplay issues that help speed up the action. Can users get right in and start playing? Do users really want to see a replay after every point? Do users want to see animations of their players walking back and forth between the sides of the court (as in real tennis)?

2. A First-Person-Shooter game (FPS). Pace can be affected in several ways in an FPS, such as the amount of chaotic action in the game or amount of tension that may be experienced. If the designer wants to create a gameplay experience that is pure adrenaline-driving action, pace will be affected if the intended chaotic action was actually perceived as being chaotic, or if the behavior that triggers the chaos is not achievable because the game objectives were not clear. Alternatively, if the design intention was more about stealth and tension, then pace will be affected very differently.

So what methods do we use to begin to address these kinds of issues? It depends on what exactly we are trying to achieve. A think-out-loud usability technique will be very useful to determine whether or not users are able to figure out how to skip scenes in the tennis game. Large sample surveys would be good for measuring the perceived "chaos" in a shooter game. The point is, we must define the problem before we can discuss the appropriate methodologies, and that starting point, the identification of objectives, most often comes from the vision of the game designer.

REFERENCES

1. Overbeeke, K., Djajadiningrat, T., Hummels, C., Wensveen, S., and Frens, J. (2003). Let's make things Engaging. In M. Blythe, A. Monk, K. Overbeeke, and P. Wright (eds.), *Funology: From Usability to Enjoyment* (pp. 7-17). Netherlands, Kluwer Academic Publishers.
2. Pagulayan, R. J., Keeker, K., Wixon, D., Romero, R. L., & Fuller, T. (2003). User-centered design in games. In J. A. Jacko & A. Sears (Eds.), *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications* (pp. 883-906). Mahwah, NJ: Lawrence Erlbaum Associates.
3. Pagulayan, R., Steury, K., Fulton, B., and Romero, R. (2003). Designing for Fun: User-Testing Case Studies. In M. Blythe, A. Monk, K. Overbeeke, and P. Wright (eds.), *Funology: From Usability to Enjoyment* (pp. 137-150). Netherlands, Kluwer Academic Publishers.

Computer Games as Interfaces

By Dennis L. Chao
Department of Computer Science
University of New Mexico
dlchao@cs.unm.edu

We can learn a lot from computer games. While most application software can be a source of confusion and frustration for users, people consistently find enjoyment in their games. One way to make everyday software more fun is to incorporate game-like elements, such as action, narrative, and interactive graphics. The problem with this approach is that the individual features that make a game entertaining might not work out of context. For example, a cartoon character stranded in a serious application is not only incongruous, but it could feel condescending to the user. So why not use a complete game as a front-end for a "serious" application? Would this hybrid be as fun to use as the original game? To answer these questions, I created PSDoom, a system administration tool that takes its interface from the popular first-person shooter Doom.

From Doom to PSDoom

I wanted to add an interface to the standard set of tools used to manage programs running on a computer, such as the Windows task manager (Figure 1). These provide data about running programs and allow the user to terminate those that have stalled or crashed. What is interesting about this mundane task is the vivid language used to describe it. People talk about "killing" programs or "blowing them away," "fighting for resources," and letting "daemons spawn." I chose to borrow the interface of Doom to reflect this aggressive language.

In 1999, Id Software released Doom's source code, making it easy for me to turn Doom into the program manager PSDoom. I added code to create one monster in the Doom environment for each program running on the user's computer. Each of these new monsters is labeled with the name of its associated program (Figure 2). For example, as I write this article, I can see zombies representing my word processor, a Web browser, and several terminal windows waiting patiently for me in a large room. PSDoom allows the user to affect the running programs by inflicting damage on the monsters. A light wound lowers the corresponding program's priority to give it fewer CPU cycles, causing the program to run more slowly on the computer. If the monster is killed, the associated program is terminat-

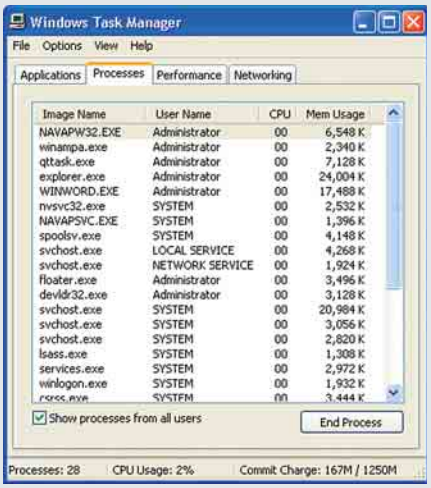

Figure 1: The Windows XP task manager. Users can terminate running programs by selecting them from the list and clicking the "End Process" button.

ed (Figure 3). The user's character can also sustain damage, and when it is killed the character is restored to health and loses any items, such as powerful weapons, that he or she had previously acquired. Because part of the fun of Doom is to find and use more powerful weapons, the fear of losing them is a strong incentive to be careful with one's character.

PSDoom appeared to be an immediate success. Only a few weeks after the application was written, tens of thousands of people visited the project's Web page and thousands downloaded the code. The computer community seemed ready to have an interface to match the violent language used for system administration tasks. Many system administrators, who would be the most likely to perform such tasks, have played Doom, so its interface is a familiar one. However, out of the hundreds of e-mails I received from people who loved the interface metaphor, only a handful had actually used PSDoom. These users found it much more satisfying to "shoot" a misbehaving program than to click on it in the task manager or to type kill -9 at a UNIX prompt. Unfortunately, they also enjoyed shooting at everything in PSDoom, killing critical programs and causing the computer to crash in surprising and sometimes spectacular ways. So despite its entertaining interface and compelling metaphor, PSDoom never became a practical application.

Metaphors can initially help users become familiar with a system but will inevitably mislead when the metaphor and the system differ. In this case, the metaphor breaks down when one considers the goals in Doom (the game) and PSDoom (the application). The goal of the game is to kill as many monsters as possible, while the goal of the system administration task is to kill processes only when they need to be killed. The PSDoom interface gives no information about which monsters should be attacked, so users often attempt to kill all of them. Even if there were such information, the user, surrounded by hostile-looking monsters, would probably try to shoot them all anyway. Allowing anything to survive would be antithetical to the Doom narrative.

Conclusion

Game-like interfaces have not entered the workplace, but they have entered the home. Developers of children's software embrace computer games, though not always intelligently. In *Interface as Mimesis*, Brenda Laurel criticizes educational software that interleaves educational drills and short game segments. Writing about a hypothetical program that allows a child to play a game for 20 seconds if he or she solves three math problems, she asserts:

> *Either the math problems or the game segments are gratuitous, depending on Jimmy's point of view. The proper solution is either to eliminate one of the activities, or re-shape the context so that it includes both; e.g., a starfighter simulation in which Jimmy must naturally solve math problems in order to operate the ship.*

Adults at work are in the same situation as Jimmy—surreptitiously playing a few rounds of computer solitaire or browsing the Web as a reward for using a boring application. Is there a way to integrate work and fun? It will be difficult, but the overwhelming response to PSDoom shows that it would be greatly appreciated.

REFERENCES

1. Chao, D.L. Doom as an interface for process management. (2001). In J.A. Jacko, A. Sears, M. Beaudouin-Lafon, & R. Jacob (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 152-157). New York: ACM Press.
2. Laurel, B. K., Interface as mimesis. (1986). In, D.A. Norman & S.W. Draper (Eds.), *User centered system design: New perspectives on human-computer interaction* (pp. 67-85). Hillsdale, NJ: Erlbaum.

RELATED URLS

Information on PSDoom can be found here:
www.cs.unm.edu/~dlchao/
Source code for PSDoom (for Linux) can be found here:
http://psdoom.sourceforge.net/

Figure 2: Programs are represented as monsters in PSDoom.

Figure 3: Killing programs in PSDoom. The user is terminating emacs, a text editor, with a shotgun.