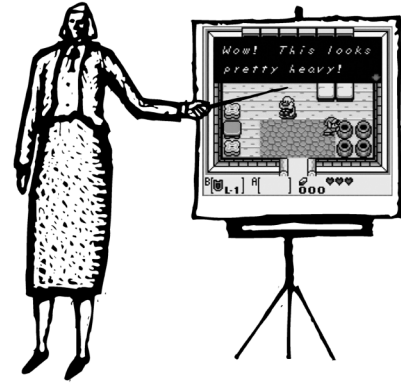


# It's All Part of The Game

## Applying Video Game Interaction Design and Business Performance

by Ara Shirinian



For all the varied applications we can create out of software, the one domain that seems to be most underappreciated is that of the game. That's understandable: You don't visibly accomplish anything by playing a video game. It's just a curious diversion that many kids enjoy and many adults avoid. A game's sole purpose is to entertain you. It won't help you solve your real-world problems, though it might help you forget about them. So it's easy to see why people don't look toward video games when they want to study things like performance support, human-computer interaction, and interface design.

There is something important to be learned from video games, however, especially when it comes to interaction design. Video games are quite closely related to things like a spreadsheet program, a tax software program, or an electronic performance support system. In all cases, we have software, and then we have human interaction with that software.

Video games are especially interesting, because interaction exists without specific goals outside the system. The only other important difference between a video game and some other software is the metric that measures its efficacy as far as the user is concerned: Is it fun to play? Will continued play be worthwhile? Of course, my goal while muddling through TurboTax® this past April was not to have fun, but it's really the same situation. The same types of interactions can take place in these ostensibly disparate types of software. The user can experience frustration, confusion, confidence, comfort, anger, and, to varying extents, pleasure. Due to the extreme nature of the video game, one can feel some additional emotions. But what about frustration, confusion, and anger in a video game? If you don't believe this happens, try picking up some new game and see how long it takes for you to experience any of those three.

What is the point in designing software such as TurboTax? In general, the point is to enable the user to complete the task (filing tax returns) as efficiently as possible. When the software allows the user to become angry, frustrated, or confused (either directly or indirectly), his or her performance suffers. However, when the user feels in control, comfortable, and confident, those feelings usually correlate with improved performance.

Additionally, the software may actually be enjoyable to use. This situation mirrors that of video games exactly, except that we are more interested in the emotional responses of the user than in how well some task is performed. For us, the efficiency of the task being completed is manifested as a side effect, instead of the other way around.

A user can experience any number of different interactions within a video game. We can examine his or her emotional responses to those interactions and categorize them. We say that those interactions that encourage the player to continue playing are good, and those interactions that discourage the player from continued play are bad. Admittedly, it does sound a little insidious. Our goal is to create an interactive product that will only promote further use, but that is often the measure of a good game. In fact, the term *addictive* is commonly used to describe many players' favorite games.

## Video Games: Definition and Scope

There are all kinds of electronic games, but not all of them are video games.

There is no clear definition of a video game that everyone can agree on. It is the kind of terminology that can spur endless discussion. In the interest of avoiding that, I'm now going to explicitly state what is included and excluded by the use of the term *video game* in this article.

The term *video games* encompasses console games and arcade games only. A console is a customized computer whose design is solely to play games at home. Recent examples of consoles are Sony's PlayStation 2, Nintendo's N64, and Sega's Dreamcast. Arcade games are the same thing as console games, but they are housed in large cabinets and are designed for playing in some public location.

It is important to mention that this discussion excludes all computer games—that is, all games designed to be executed on general-purpose computers. These are commonly referred to as PC games, Mac games, etc. This article also excludes all electronic games that do not use some video monitor or television as an output device.

To restrict the subject even further, the article will only address video games in which users control some on-screen representation of themselves (henceforth known as an *avatar*). The avatar can be as simple as a cursor navigating within the world of a menu interface. Also, interaction must take place between the avatar and objects in the game world. Finally, the user may only interact with the computer, not other users playing the game. Fortunately, this still allows us to talk about nearly all video games.

For most video games, the user's input device is called a controller. It usually consists of one or more directional con-

trols (either a joystick or a set of buttons corresponding to different directions) and a number of action buttons. One directional control is used to move the avatar within the space of the video game world, while additional directional controls serve to adjust the avatar's orientation among other things. Action buttons allow the avatar to perform commands, such as attack, jump, block, or other miscellaneous functions.

## Positive Video Game Experiences

What kinds of experiences result in user reactions that we associate with a good game design? People often look for different things in games. Some particular aspect of a game that might be appreciated by one user can be detested by another. But that doesn't mean it's time to throw your hands up in the air and say that everything about a video game is subjective. There is a set of common experiences that will encourage the user to continue playing. Four are repeatedly encountered in well-designed games:

1. The user learns the rules of the game easily. Few people want to play a game if they have to expend serious effort just to understand the rules. All games have rules, but the best games either have very simple rules or present new rules to the user in a gradual fashion, well before it becomes crucial to master them. Also, rules must be conveyed to the user in an unambiguous manner.
2. The user is able to perform some action more skillfully than what his or her perception of ability suggests. First, this can apply to games where the avatar is controlling some complex action that is representative of reality but still beyond the player's ability. Second, it can apply to situations where, purely within the context of the game, the player is able to surmount some challenge beyond his or her expectations.
3. The user can perceive an improvement of skill during game play. One of the greatest influences on continuing play is the perception of improvement. If the player thinks he or she is getting better at the game, he or she will want to keep going. Conversely, if the player doesn't observe any improvement or seems to be getting worse, he or she won't want to play for much longer.
4. The user feels like he or she is in direct control of the avatar. The importance of direct control cannot be understated. The more direct the perception of control, the more comfortable and confident the user is in his or her position. On the other hand, if the control feels sluggish and indirect, the opposite happens.

The avatar is an extension of the user. When the avatar does not respond as directly as the user expects it to, it's a recipe for frustration and confusion. Additionally, direct control facilitates all of the previous three experiences.

The kinds of interactions we design can lead to either a positive emotional state or a negative one. We have already

identified what kinds of experiences are desirable, so now it remains to examine what kinds of designs can create them.

### **Immediacy of Feedback**

Almost everything the user does in a video game is based on visual feedback. Auditory and tactile feedback are also present, but for the vast majority of games, visual feedback dominates and the other two senses are usually relegated to a supplementary role.

The most important feedback received by the user concerns the state of the avatar. After all, the user *is* the avatar in the context of the game. Control of the avatar must be simple and easy. It is absolutely necessary that the user feel comfortable and confident in controlling the avatar. Anything other than immediate feedback in this case is deceptive and places an unnecessary burden on the player.

For example, suppose our avatar is some fat-nosed plumber that we can move right and left on the screen. If we start to move it right, it should move right immediately, regardless of its previous state. Now, if we change our minds and decide to move it left, it should turn around and move left just as immediately. What if our game included a “turning animation” to show the avatar turning around before it actually moves left? This is a poor design, because the result is an effective delay of feedback exactly corresponding to the duration ( $n$  seconds) of the animation.

Why is this undesirable? It doesn't really matter what causes the delay. When the user wants to tell the avatar to turn around and go left, he or she expects that the action will happen immediately. But that's not what happens. The action is really executed  $n$  seconds later. The user is forced to compensate by trying to estimate when his avatar will need to turn around, then attempting to press the button  $n$  seconds before that. It's a highly unnecessary and highly confusing complication.

### **Allowing Graceful Recovery From Mistakes**

When playing any video game, the user is bound to make mistakes and use poor judgments. This is unavoidable, so the question is only one of degree. However, the possibility of recovering from mistakes varies considerably from game to game.

For example, in the game *Pitfall!*, once you decide to make Pitfall Harry (your avatar) jump forward, he'll jump forward at a fixed height and distance, and there's nothing you can do about it until Harry completes the jump. On the other hand, in the more recent game *Klonoa 2: Lunatea's Veil*, you have the ability to move Klonoa (your avatar) right and left while jumping at the same speed as if you were just walking back and forth. The difference here is that in *Pitfall!*, if you wanted

Harry to jump, you were forced to commit to your decision. In the latter game, it is possible to start a jump, realize that it was a bad idea, then move back to your original position without consequence, so long as you began to move back before the midpoint of the jump. In both cases, the player learns the same thing: “I shouldn't have jumped at that moment.” But in *Pitfall!* the player was penalized, and in *Klonoa 2* the player had an opportunity to recover without penalty.

If the player is penalized too often, he or she may become frustrated. It's important to give the player opportunities to realize mistakes and correct them, instead of immediately dispensing penalties with every imperfect action.

The issue boils down to how much control the player is allowed over the avatar. In a game like *Pitfall!*, you are basically surrendering all control while your avatar is jumping. But a game like *Klonoa 2* allows you to retain control over your avatar, even while it is jumping. That particular design is superior, because it allows more direct control over the avatar, and that makes the player more comfortable and confident.

One could argue that the fixed jump in *Pitfall!* was deliberately implemented to add an element of strategy to jumping. The player has to consider the consequences of jumping at a particular moment before actually jumping. Allowing further control during a jump would destroy that strategy. Indeed, it would, to an extent, but not everything in a game needs to be laced with strategy. It is too much to require the player to continuously strategize such basic maneuvers.

This is not to say the player should be able to get out of every mistake without facing penalties. Even if you jump the wrong way in *Klonoa 2*, it takes a degree of skill to realize it early enough to do something about it. It adds more to the game than it detracts, because it takes at least as much skill (only of a different type) to correct a mistake as it does to think ahead, knowing there's no possibility for recovery. It makes for a faster and more exciting game if the player has such latitude in manipulating the avatar.

### **High-Quality Feedback**

Feedback is the only way to communicate to the user the consequences of his or her actions. The game designer must judiciously choose how much and what kind of feedback to provide.

Feedback must be obvious enough that the player notices it easily, and feedback about one thing must be easily distinguishable from feedback about something else. For example, in some games the avatar can be in a number of different health states, like “normal,” “poisoned,” “cursed,” etc. Common practice is to turn the avatar's color to green or purple when it's in a poisoned state. That's fine, so long as



Figure 1. *Gauntlet* (©1985, Atari Games).

it is distinguishable from the normal status. But it's not always a sufficient degree of feedback. What if the player doesn't know what that means? At least for the first time, the game should inform the player a bit more overtly about what it means to be poisoned.

Conversely, too much feedback can be a nuisance. What if the game opened up a dialog box explaining to you all the details about the poisoned state every time your avatar was poisoned? You would appreciate it the first time, but perhaps not the 10th or 15th time.

A great example of this is the classic arcade game *Gauntlet* (see Figure 1). The avatar is a character running around in a maze, battling creatures and collecting items and treasure. Whenever you pick up some new item, the game stops and an explanation appears, describing what the new item does. But when you encounter the same thing a second time, the avatar just picks it up without explanation or interruption.

An example of misused feedback is in the game *The Legend of Zelda: Link's Awakening DX* (see Figure 2). In this game, the avatar Link may find pots within the game world. The pots are too heavy to manipulate, but later on a special item allows Link to lift and move them. Unfortunately, without that special item, whenever Link touches a pot the following message appears on the screen: "Wow! This looks pretty heavy! You won't be able to lift it with just your bare hands...." If Link touches a pot in the same room again, the message doesn't appear, but if Link walks between rooms

the message pops up the very next time a pot is touched. The player frequently will have Link touch pots and other functionally similar heavy objects by accident. Each time the user is subjected to the same message. Worse, the message appears on the screen slowly, letter by letter. The dialog box stays on the screen for a minimum of three seconds and the player must press a button three times to dismiss the message. After the first few times, the idea is already ingrained in the user's head. Subsequent encounters only serve to frustrate.

If feedback results from a player's mistake, it should show the player why and how he or she made the mistake and possibly how it could be corrected. Players commonly make mistakes without knowing what they did wrong. Learning is only possible if the player can see why his or her action was a poor one. We want to facilitate learning for the player as much as possible, so the game must apply feedback that is useful, at least for such basic actions.

### Input Device Mappings

In video games, the interface that connects the player to the game provides the association, or "mapping," between the buttons on the input device (controller) and the avatar's movements or actions.

Video games are unique because in most instances, the same generic controller must be used for a wide variety of games.



Figure 2. *The Legend of Zelda: Link's Awakening DX* (©1993, 1998 Nintendo of America, Inc).



Figure 3. *Dance Dance Revolution* Menu Selection Screen (©2001 Konami of America, Inc).

This explains the arbitrary labeling of buttons, like A, B, X, Y, etc. These buttons' function cannot be labeled, because they often do very different things from game to game. In fact, there are a few conventions for usage of these buttons, but most of the conventions are arbitrary as well.

For a directional control, there exist two natural, easy-to-understand mappings. One corresponds to moving the avatar on a two-dimensional plane parallel to the television screen (vertical plane). Up is up, right is right, etc. The other corresponds to moving the avatar on a two-dimensional plane parallel to the floor (horizontal plane). Up moves the avatar far away from you, and down moves it closer to you; left and right are the same.

It is completely reasonable for the player to expect to move the avatar left when he or she presses left on the directional control. However, it is totally unreasonable for the player to watch the avatar move to the right in response. It sounds silly, since no sane designer would want to reverse the natural mapping, right? But it's not as trivial as it seems.

A common confusion arises in certain menu interfaces. Because left and right on the directional control can be made to move or do anything, it is sometimes difficult to tell whether you're supposed to be moving a cursor (our avatar in this scenario) within a menu or moving the entire menu in relation to a stationary cursor. This confusion is mainly a side effect of the highly stylized menu designs in many video games.

In most games, the correspondence between pressing a button on the directional control and movement of the cursor is straightforward.

For example, in *Dance Dance Revolution* (see Figure 3), at the song selection screen the user uses left and right on the directional control to select a song out of a list. However, pressing left seems to move the cursor to the right! In actuality, the user shifts the menu left relative to the cursor and the cursor never moves. But there's no cue at all to suggest that the user will be moving the entire menu around instead of just the cursor. As a result, some people perceive this as a cursor movement in the wrong direction!

On the other hand, in *Tony Hawk's Pro Skater 2* (see Figure 4), the main menu consists of a graphic of a skateboard wheel. Menu items are written on different sections of the wheel. Whichever menu item is in the top position is highlighted. In this case, if we press left, the wheel rotates clockwise and the cursor stays put. If we were only rotating a wheel, this mapping might seem totally backward, but because we have a cursor, it's more straightforward for the user to imagine the cursor moving within a stationary environment instead of displacing the environment around the cursor.

## Interface Complexity

One can describe the interface between player and game as simultaneously occurring within two layers. The first layer concerns the player, who presses buttons on the controller to manipulate the avatar. I call this the user-avatar interface. The second layer concerns the avatar and how it manipulates various objects within the universe of the game. I call this the avatar-game world interface.

This kind of division is useful because it allows us to differentiate between manipulating the interface and actually playing the game. In a perfect world, the user-avatar interface would always be transparent, so the user would only need to worry about his or her actions in game world terms. Unfortunately, as there is always a certain degree of arbitrariness in our input devices, our brains have to think about what buttons we are pressing and how that translates into actions in the game. One possible exception to this applies to expert players who have heavily practiced the same type of game over a period of many years. In some of those cases, the player has become so familiar with the first layer of interface that he or she doesn't need to think about it at all.

Regarding each layer, we can also speak in terms of their complexity. When the user-avatar interface is complex, the player has at his or her disposal a large number of different functions that control the avatar. It takes longer for the player to process and choose the appropriate one at any given moment. When the avatar-game world interface is complex,

the avatar may face a large number of elements in the game world with which it must either interact or react.

The sum of complexity in both layers together can be understood as what people call the *difficulty* of the game. When the total complexity exceeds a certain threshold, the game simply becomes too difficult to play successfully. Of course, this threshold varies from person to person, depending on how skilled the player is and how much practice he or she has had playing the game in question.

A video game should begin with an overall low interface complexity to accommodate players who are not familiar with the rules. However, there is as much danger in making a game too easy as there is in making it too hard. If a game is too easy or too simple, the player may lose interest. Conversely, if a game is too hard, the player may give up. The best situation is to provide an environment in which the player feels challenged but is able to make progress in the game in spite of it. This is difficult to implement because of the large variance of video-game skill among different people.

### Difficulty Curves

If a game is imagined to be a linear series of challenges, the difficulty curve is a make-believe function that can be graphed by plotting the “difficulty level” on the *y* axis with respect to each successive challenge in the game, from left to right on the *x* axis.

Although no quantitative metric has been established to describe game difficulty, we can still discuss the idea in relative terms. The classic difficulty curve can be described as

starting off with low difficulty at the beginning of the game, then strictly increasing as the user completes each section of the game.

Today, games with all kinds of difficulty curves exist. Some exhibit almost constant difficulty throughout, while others seemingly become easier or harder at random. A game can be well designed with any one of a number of very different difficulty curves. Good designs avoid sudden increases in difficulty. When a game suddenly becomes very difficult, it confuses and frustrates the player. It's irresponsible for the designer to lead a player up a hill and then ask him to scale a wall along the way. In these situations, the player may be ill prepared to take on such a challenge. On the other hand, sharp decreases in difficulty are a little different. It's refreshing to give the player a break once in a while. Some games implement this idea by presenting “bonus stages” to the player at regular intervals, where the player may progress without any fear of penalty.

### The Interface Complexity Explosion

In many classic game designs, the complexity of the user-avatar layer is fixed at a low level: Usually there is just one directional control and perhaps one or two action buttons to manipulate the avatar. Difficulty is increased by steadily adding to the complexity in the avatar–game world layer. For example, in the game *Kaboom!*, the player can only move the avatar left or right—for the entire duration of the game. In the avatar–game world layer, interaction takes place between the avatar and a set of bombs steadily falling from the top of the screen to the bottom. The player is penalized for allowing any bomb to fall past the avatar. At first, there are few bombs and they fall at a slow rate. The avatar–game world complexity is low. As the player continues to play successfully, the number and rate of descent of the bombs increases. In other words, it is the avatar–game world complexity that is increasing.

In more recent games we see something different. One can immediately observe that the complexity of the user-avatar layer is quite high right from the beginning. This is evidenced by how our input devices have become increasingly complex over the years: In 1977 the Atari 2600 console included an input device that consisted of one joystick and one functional button. By the 1990s, game consoles sported controllers boasting four to ten separate functional buttons. Most recently, the Sony PlayStation 2 and the upcoming Microsoft Xbox and Nintendo GameCube consoles use controllers with no less than 3 distinct directional controls and anywhere from 7 to 12 functional buttons.

Furthermore, with the addition of game interaction taking place within three dimensions of space, the



Figure 4. *Tony Hawk's Pro Skater 2* Menu Selection Screen (©2000 Activision, Inc.).

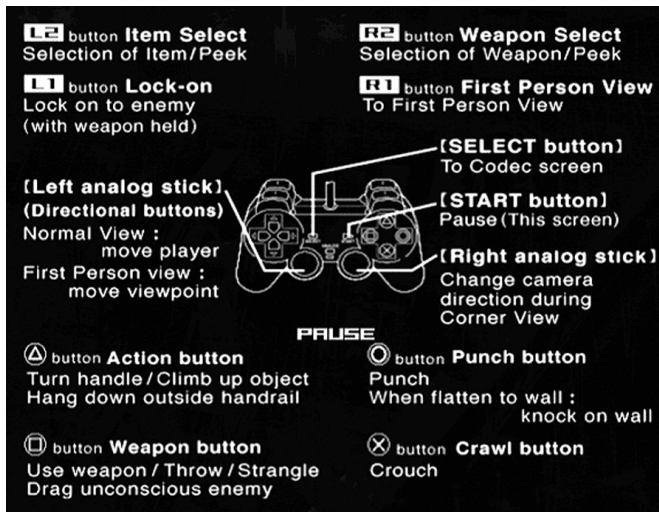


Figure 5. *Metal Gear Solid 2: Sons of Liberty Trial Edition*. This screen appears when the player pauses the game. It describes only some of the functions available for manipulating the avatar from the very beginning of the game (©2001 Konami of America, Inc.).

responsibility placed on the player at the user-avatar layer is staggering. Today's game player has to be concerned with the avatar's position and orientation, plus the position and orientation of an imaginary camera that follows the avatar around. When a game begins with this much complexity, it undoubtedly intimidates some potential players.

This results in an unintentional but dangerous side effect: If the user-avatar layer is complex enough, it takes the player's concentration away from the avatar-game world layer. The player must think about both layers of interaction, yet the user-avatar interface is so complicated that the player's reaction time with respect to the avatar-game world layer is reduced. The game designer must stick to relatively simple interaction in the avatar-game world layer to compensate. Otherwise, the game would be almost impossible to play.

When a game begins with a high user-avatar interface complexity, it unreasonably demands competence of that level from the user right away. For example, *Metal Gear Solid 2: Sons Of Liberty Trial Edition* (a short, one-stage game of a title still in development) uses one of the most complicated user-avatar interfaces ever: two directional controls plus ten buttons, most of which have two or three different functions depending on context (see Figure 5). The player is left to master the entire complex interface all at once. It's like trying to teach algebra to someone who has no knowledge of arithmetic. However, when the game begins with a simple user-avatar interface, the user can more easily master control. The game can then add difficulty by gradually introducing new elements to either the user-avatar layer or the avatar-game world layer.

Within a game, whether the user-avatar interface complexity should remain at a low level or not isn't quite as crucial. Excellent game designs have been created in both cases. However, keeping the complexity fixed at a low level imparts certain advantageous qualities. First, transparency in the user-avatar interface allows the game to be more easily accessible to larger audiences. Second, it permits the game to play faster and more intensely. Third, and most importantly, it maximizes time spent playing the game and minimizes time spent fumbling with the controller.

## Concluding Thoughts

There is no intrinsic quality that attaches a video game to a particular gender, age, language, or culture. The entire medium is on the brink of achieving truly mainstream status. However, if games continue to be designed without proper consideration for the average user, the market will never be able to grow fully beyond the confines of the avid gamer: those few who are willing to endure any and all of the unreasonable demands today's games might impose.

As humans, it's in our nature to play games. The only problem, when video games are concerned, is that there's a necessary additional interface that comes between the game and the player. While the function of this extra layer of interface is to enable interaction in the first place, its overall haphazard implementation has resulted in the alienation and intimidation of potential players. Even among avid game players, emotions such as confusion, frustration, and anger are frequent. The very idea that video games should cause such feelings in the player is absurd.

Only when game designers are prepared to treat the interaction between human and video game as a formal discipline will this medium be able to develop into an activity that everyone can truly enjoy. 🎮

---

**Ara Shirinian** is an Associate Editor at *Tips & Tricks* magazine, a video gaming publication, where he is attempting to "shake up" the industry from the inside with his unique specialties in interface design. Ara first cultivated these interests as a Research Assistant for the Human-Computer Interaction Laboratory at the University of Maryland, College Park, where he developed user interfaces from a human factors engineering philosophy. Later, as a Software Engineer for WPI, Inc., he further developed these skills by designing and implementing custom performance support applications for clients such as the Federal Deposit Insurance Corp. Currently, Ara applies his technical background specifically to the video game industry, developing analyses and methodologies that bring the concepts of human factors, interface design, and PCD to the process of creating better video games. He may be reached at [ara@housesnet.org](mailto:ara@housesnet.org).