CHAPTER 2

Principles of Game Design

Anybody can recognize problems in a game after it has been created (reviewers are especially good at this), but how do you avoid errors ahead of time? Even though our business is young, the principles of good game design have already been established, and paying attention to them will make yours a superior game.

This chapter deals with design principles that are broadly applicable to all game genres. For design tips about specific genres, see Chapter 3, "Genre-Specific Game Design Issues."

Player Empathy

A good designer always has an idea of what's going on in the player's head.

This empathy for the player is crucial. You must develop the ability to put yourself in the player's shoes and anticipate his reaction to each element of the game. You must be able to close your eyes and see the game unfolding like a movie in your head, all before a single line of code has been written.

At any given point in the game, you must be able to say, "Here's the situation the player faces, and here are the range of choices he can make . . . Now, what will he likely want to do?" Then, your job is to let him try, and to make the game respond intelligently to his attempt, even if it's only to steer him toward a different course of action.

Naturally, no designer has completely accurate foresight. That's one reason you have testers. Testers not only hunt for bugs, but also provide feedback on things they want to try in the game but can't.

One of the hardest things for a designer to do is to keep his mouth shut while watching someone play his game. The urge to tell the tester to go *this* way instead of *that* way can be overwhelming. If you steer the tester in one direction, though, you'll never discover what thousands of actual gamers will encounter when they go the other way instead.

Player empathy not only helps you create good gameplay, but also lets you identify and eliminate problems during the design phase rather than during production, after code has been written and graphics have been created. With good player empathy, you'll write a better game, and you'll build it faster and more cheaply as well.

Feedback

The basic interaction between a player and a game is simple: The player does something. The game does something in response. This feedback is what distinguishes a game from every other form of entertainment. It's the *interactivity* that makes our games unique. Without it, the player would just be watching a movie on the screen.

Every input the player makes in the game should give him a discernible response. No input should go unanswered. This "answer" can take many forms. It can be visual feedback, aural feedback, or even tactile feedback (if the controller is so equipped). It can be positive feedback or negative feedback, but there must be *some* feedback.

Generally, this is easy when the player "gets" the game and is progressing nicely through it. It becomes more difficult when he's doing something "wrong." Nothing is more frustrating for a player than pressing a key, clicking the mouse, or pushing on the controller and having *nothing* happen. For every conceivable input, be sure to give the player some feedback about it.

If you can detect what the player is doing and know how to steer him in the right direction, do so. Give him a message about what he has done. If you simply don't understand the input, at least send a *BOOP* noise back to him. He'll quickly learn that this noise means, "I know you tried to do something, and I heard you, but I don't know what to do about that."

Grounding the Player

The player should always know where he is in the game and why he's doing what he's doing. At any given point, he should have a long-term goal, a medium-range goal, and an immediate goal. (This is true even of *software toys*, games that ostensibly have no goals, but in reality have a series of goals the player creates for himself.)

Computer games are huge, and it's easy for a player to feel lost. Also, usually games aren't played start-to-finish in one sitting. If a player has an overall map in his head, it encourages him to come back to the game again and again until he's done. Physical maps also help (see Figure 2.1).

In a strategy game, the long-term goal can be to conquer the world. In an action/adventure or RPG, it can be to defeat the ultimate bad guy. In a golf game, it can be to win an individual match.

Medium-range goals are good-sized steps toward the long-term goal. For the strategy game, perhaps it's establishing a home base. For the RPG, perhaps it's the completion of a simple quest. For the golf game, it can be the battle to win the first hole. Frequently, these medium-range goals turn out to be embodied in levels.

An immediate goal is the problem that's right in front of the player. In the strategy game, it can be figuring out which units to build to fend off an impending attack. In the RPG, it can be ordering a party before marching into the next battle. In the golf game, it can be figuring out which club will carry the ball over the water hazard without rolling it into the bunker on the far side of the green.

Throughout the game, as the player wrestles with the problem in front of him, he should always have some idea of how this single step fits into the longer path that will eventually lead to success.

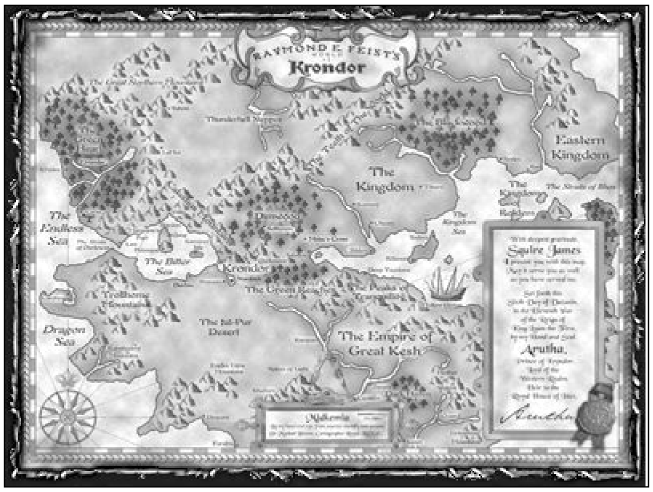


Figure 2.1 This map of Raymond Feist's *Krondor* helps keep the player oriented toward his goals. Used with permission of Sierra On-Line Inc.

The Moment-to-Moment Experience

At any instant while he's playing the game, the player has the option to turn it off and do something else. You can't let that happen. You have to hold his attention constantly and entertain him from moment to moment.

This is far more important than most designers realize. At every point in the game, the player should have something interesting to do. One of the worst things you can do to a player is to bore him.

Verbs

The positive side of creating a good moment-to-moment experience is giving the player a constant stream of interesting choices that have significant outcomes.

It's useful to think of the things the player can do as "verbs." In early shooters, the player had two main verbs: *move* and *shoot.* He also had some adverbs: move *slowly* (walk), move *quickly* (run), shoot *quickly* (machine gun), shoot *accurately* (sniper). As the genre expanded, it was primarily through the addition of more verbs, which allowed players to do new and interesting things: *climb, rappel, zip-wire, set explosives, unlock, move stealthily,* etc. In essence, every time you give the player an inventory item, you're giving him another new verb.

Each genre uses verbs. In an RTS, the player *builds, researches, surveys the terrain, gives orders,* etc. In an RPG, he *moves, talks, fights, buys, sells,* etc.

No matter what the genre, the more verbs you can give a player, the more you allow him to do. It's the doing that's at the heart of good gameplay and a positive moment-to-moment experience.

Hazards

The hazards that will destroy a good moment-to-moment experience are easy to design around, once you're aware of them. Here are several experience-killing pitfalls and how to avoid them.

Don't make the player perform a complex action twice. In an adventure game, after he has completed the steps to a puzzle, don't make him do it again. For example, after he has figured out the combination to a safe and opened it, don't make him reenter the combination. Just give him an Open Safe command to use.

In an action game, don't make him travel back and forth across the world for frivolous reasons. Today's games have big, beautiful environments, but no matter how pretty the pictures are, they wear thin after a while. If you have large environments, build in shortcuts to get from one end of the world to the other, and don't design the gameflow so that the player has to constantly backtrack or crisscross your world.

If you have rendered transitions, let the player bypass them by pressing the Esc key or a button on the controller.

The same principle applies to audio and dialogue trees. Don't make the player listen to every line of dialogue over and over to get a bit of information he forgot. Instead, let him abort each piece of dialogue as it begins, so he can get quickly to the line he wants to hear and then leave the dialogue altogether.

The same applies to cutscenes. No one wants to sit through the same cutscene over and over. God gave us the Esc key for a reason. Use it.

Restarting the game can also be tedious for the player. You might have created the most beautiful introductory movie known to man, but hey—he's seen it already. Let him bypass it.

Avoid text or dialogue dumps. Instead, dole out information in bits and pieces. Don't make the player sit through long, boring screeds.

In general, have the computer do set-up tasks the player might find boring. In an RPG, for example, allow the gamer to have the computer generate his character and party automatically. In a racing game, give him a default car that will perform acceptably without having to be tweaked. In a basketball game, don't make him select each player on each team before he can begin a game.

Make the game entertaining, moment to moment, by keeping it interesting. Give the player a lot to do—but also make sure that what he does is fun.

Immersion

Immersion is what happens when you make the moment-to-moment experience so compelling that the player is drawn completely into the game and the real world disappears. It isn't until he hears birds chirping that he realizes he's spent the whole night playing the game (again!). This can be as true of chess games as it is of action games.

John Gardner, in *The Art of Fiction*, writes that a good book creates "a continuous dream" that's bolstered by providing a constant stream of concrete detail. Immersion works the same way. You bathe the player in a constant stream of images that pull him into your world, and you avoid gaffes that jar him out of his reverie. If you break the dream, you lose the immersion.

These gaffes can be anything from typos to bad voice acting. Modern slang in a medieval world will destroy the player's suspension of disbelief in a heartbeat, as will graphical styles that change from scene to scene, or stupid AIs.

A successful game entices the player into the gameworld, and then never lets him go.

Writing

Good writing is invisible. Bad writing draws attention to itself and instantly destroys the player's sense of immersion.

Every game uses words somewhere. The player might see them as text on the screen or hear them as spoken dialogue. The writing can be confined to cutscenes between levels, or it can be an integral part of gameplay. Regardless, you can be sure that at some point, someone will be sitting down with pen in hand (keyboard on lap?) to put words into your game.

It turns out, though, that writing well is hard. People spend a lifetime learning how to do it. If you've never given writing much thought before—*don't write.*

This doesn't mean that you can't be a good game designer. It just means that if you've never studied writing, if you've never struggled to learn the difference between good writing and bad, you should bring in someone else to do it.

Design Within Limits

Designers often forget that building a game is actually a software development project. It has a cost and a schedule, and its ultimate success or failure hinges not just on good gameplay, but on whether you can deliver that gameplay on time, on budget, with technical features that work, and without crashing the player's machine.

The person who makes this happen is the tech lead, and you must work with him to make his job easier. Even if you're not a programmer, you should read books about the software development process and adapt your design to the tech specs and the budget.

As a designer, you must limit yourself to features that can be implemented on the target machine, so you don't find out at the end of the project that the game runs like a dog.

Removing Impediments

Another way to enhance the moment-to-moment experience is to remove technical impediments to the player's enjoyment, such as excessive disc swapping, long load times, game interruptions, limited saves, bugs, a poor interface, and so on.

You might think of these as technical problems, not design issues, but they're areas where programming and design meet. The practitioners of both areas have to work together to create an enjoyable game for the player.

Disc Swapping

If you design a multiple-disc game with a huge world and give the player complete access to the entire world at any time, it will result in annoying disc swaps. If he shuttles back and forth between two locations that have graphics and sound on different discs, he'll be faced with the onerous task of swapping discs every time he crosses the boundary.

One solution to that problem is to dump everything onto the player's hard drive, but this creates a huge footprint that will drive your sales and marketing people crazy. The system requirements will go up, thereby driving sales down.

The better solution is to design *choke points* in the game. These are points beyond which the player can't go back, other than by restoring a saved game. This allows the programmers to organize the game's data on successive discs, put fewer core assets on the hard drive, and eliminate the need for the player to swap discs continually.

By doing this, you give up a theoretical design advantage (total freedom for the player to go anywhere at any time) in favor of a practical gameplay advantage (gameplay uninterrupted by annoying disc swaps).

Load Times

Another potential impediment is long load times. This, too, is something you can address in your design. If you suspect that load times will be an issue (your tech lead should be able to give you a good feel for this), perhaps you should alter your design to allow smaller levels. Or you can designate points along the way where you pause the game for just a second or two for a quick load. This is a technique that *Half-Life* has used with much success.

Game Interruptions

In every game there are breaks in the action. Perhaps the player has come to the end of a level. Perhaps his character has died or otherwise hit a failure condition. Regardless of what causes the break, try to keep him involved. If he fails, don't kick him all the way back to the opening screen. Instead, as quickly as you can, cycle him back to a point just before the failure and let him try again. Always give him the sense that just one more try will bring success. Make it hard for him to give up. If he masters a level, tease him right away with the challenge of the next one. Always have another goal waiting just around the corner.

Saving the Game

It's astonishing that games are still being designed that allow no saves (or only one save at a time), or that let you save only at infrequent junctures, such as between levels.

This is horrible. It condemns people to replaying sections of the game they've already completed, which is a huge disincentive for them to pick up the game again when they've been interrupted.

In a PC game, you simply *must* allow the player to save his game whenever he wants, wherever he wants, and as many times as he wants. You should also let him name the save files whatever he wants. In a console game, where the size of the memory card is restricted, you should still try to give him as many save slots as you can, allow him to name them, and let him save as often as he wants.

Autosave, undo, and autorestore-on-death are all nice features, but they don't make up for preventing the player from saving the game when he wants to. This is a design requirement you must communicate to your tech lead at the very start of the project, so he can design the game's architecture around it. Mentioning it to him halfway through development might be too late.

Housekeeping

There are a few activities that the player should be able to perform at virtually any point in the game. If you handle these activities gracefully, no one will notice, but players will subconsciously appreciate it. If you handle them poorly, everyone will notice and they'll complain.

The player should be able to *pause* at any point. The phone rings, he has to go to the bathroom, the boss walks by. . . . There are any number of reasons why he might need to suspend your game world temporarily in favor of the real world.

It should be easy for the player to *quit.* (Mechanically easy, that is—psychologically, you want to make it as hard as possible for him to leave the game.) It's very frustrating to finish a session of gaming and not be able to clear the game off the screen.

The player should be able to *save/load* whenever he wants (see the preceding section).

The player should have easy access to the *options screen* so that he can customize the game controls and settings.

Help should be available to the player at all times. The initial help screen should be easy for the player to bring up, and the subsequent screens should answer as many of the player's questions as possible. You should tell the player how to save and load the game, how to customize the game by going to the Options menu, where to find additional information, and so on.

Bugs

Nothing knocks a player out of a game like a bug. Many designers think that bugs are the exclusive domain of programmers. Not so. There are many ways you can help keep the game bug-free.

Be clear in your design documents. If you're unclear and the programmers do things the wrong way, they'll have to go back and do it again. This reduces the amount of time they have to address other problems, and vestiges of the incorrect way are certain to remain and will be hard to stamp out. The more you can get it right the first time, the more they can too.

Be flexible in creating your design. Consult with your tech lead and listen to his advice. If he says of a particular feature, "It will be hard to code and buggy as hell," *believe* him. Perhaps there's some other way to accomplish what you want. Stay involved throughout the whole development cycle. You can't create a design document and walk away.

A bug doesn't have to be a crash. It can be anything that deviates from what you intended: a weapon that's too powerful, a line of dialogue that's spoken in jest but that you meant to be taken seriously, or an inappropriate lighting scheme that creates the wrong mood for a room. The earlier you catch these problems in the development cycle, the easier they are to fix. The later you discover them, the more likely they are to remain in the game.

Keep a level head. As the game comes down to its final days in development, everyone's life revolves around managing the bug list. Deciding what gets fixed and what doesn't is the joint responsibility of the test lead, the tech lead, the producer, and the designer.

When everyone is working 20-hour days in a superheated atmosphere, it's easy for problems to be blown out of proportion, arguments to erupt, and friendships to shatter. At this stage, you must distinguish between problems that are a matter of taste and problems that will actually hurt the game. Yield on the former. Stand firm on the latter. Remember that yours is not the only voice in the room, and try very hard to check your emotions at the door.

Interface Design

Creating a good-looking yet functional interface is one of the most underrated tasks of game design—but it's vital to get it right. You must decide what the game looks like on the screen, how information is passed along to the player, and how the player uses the controller or keyboard/mouse to input commands.

Influential game designer Brian Moriarty tells us a game interface should be "desperately simple." (And anyone who has met Brian will recognize the passion he puts behind the word "desperately.") Designer Noah Falstein refines this idea by quoting Albert Einstein: "Make things as simple as possible, but no simpler." Noah's point is to keep paring away elements of the interface until it's as simple as possible, but don't go so far in the name of simplicity that you remove something the player needs to play the game easily.

Vital information must always be easy to find (see Figure 2.2). The player should be able to understand what's going on at a glance. For some games, this means creating a *HUD (heads-up display)* that overlays information on the action screen. For others, it might be best to display status information and control buttons in a wrapper around a smaller active area. For still others, the information doesn't have to be visible at all, as long as the player can bring it up quickly.

The controls must be clear. The actions that the player takes most often should be physically easy for him to perform using the controller or keyboard/mouse. You must hone these inputs to a minimum number of nonawkward clicks, keypresses, or button pushes.

In his excellent book, *The Design of Everyday Things*, Donald A. Norman insists that the physical appearance of an object must tell us how it works. In a sentence that's directly applicable to game interfaces, he writes, "Design must convey the essence of a device's operation; the way it works, the possible actions that can be taken, and, through feedback, just what it is doing at any particular moment." He also goes on to emphasize the importance of constraints: "The surest way to make something easy to use . . . is make it impossible to do otherwise—to constrain the choices."

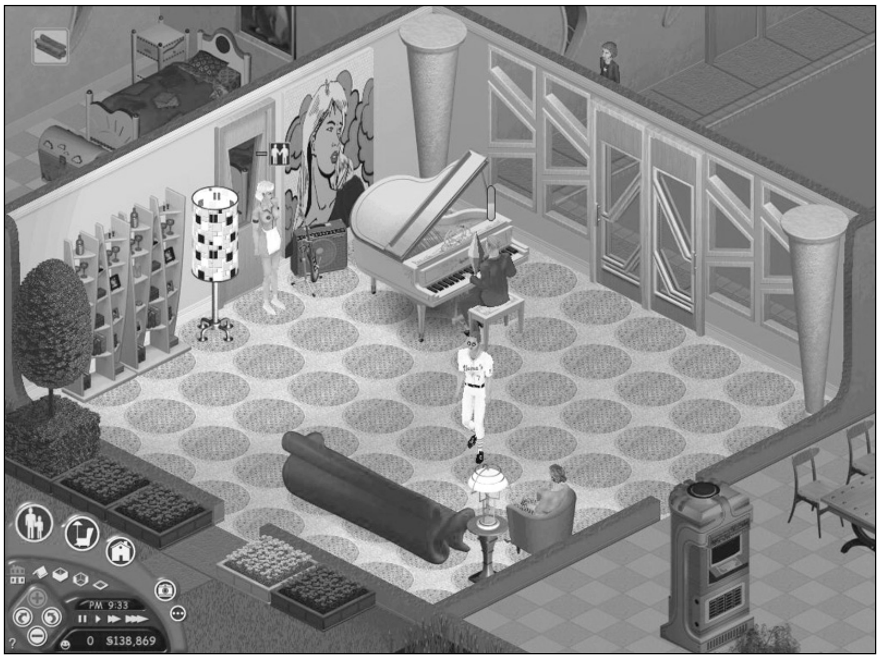


Figure 2.2 The interface of *The Sims* is clear and easy to use.
© 2000 Electronic Arts Inc. All rights reserved.

You cannot rely on your instincts to get this right. You have to try out the interface, first with team members and later with testers. What's intuitive to you can be awkward to someone else.

Pay attention to the conventions of your genre, and use them to your advantage. Don't try to reinvent the wheel. If there's an established way to play the kind of game you're designing, and you like it, and it works, *don't change it*!

Elegance and ease of use are more important than increased functionality. Achieving this compromise is never easy, however. Frequently, the team will argue about it for months. If including a nonvital feature comes at the cost of messing up the interface, you're better off without it.

Prototype the interface early, and keep noodling with it. Usually, there are several interfaces within a game. Look at all of them. Get people to test them early, and listen to their feedback. Most importantly, *play your own game.* If the start-up menu annoys you, it will annoy others. If the options menu is clunky, you'll be the first to know about it. If saving and loading is awkward and drives you nuts, you should fix it before a customer ever sees it.

You need to come back to this over and over again throughout the development process. The game must be easy to play. The player should not have to fight the interface. The whole point is to let the player do things quickly and simply. If the interface looks good and its theme is well integrated into the game, you get plus points. If making a "cool" interface confuses the player or makes it harder for him to play the game, it's not worth it.

One day, voice recognition and speech synthesis may revolutionize the way we interact with our games. But until then, the whole point of the interface is to let the player do what he wants without having to think about it. After a while, his fingers should move unconsciously on the controller or keyboard/mouse so that he's thinking only about *doing* it, rather than *how* to do it.

The Start-Up Screen

When the game boots, you have no way of knowing anything about the person sitting at the controls. Is he a complete novice for whom this is the first videogame ever? Is he an expert in the genre who can't wait to get to the tough stuff? Is he the gamer's roommate, who's only on the first level while his roommate is halfway through? Is he the proud but battered gamer who's firing it up for the zillionth time, hoping to finally defeat that last level?

You must design a start-up sequence that will accommodate all these users.

Your start-up screen should give the player the option of

- Going right into the game for the first time ("new game")
- Loading a saved game
- Going to the tutorial or practice area
- Opening the options menu to tweak features
- Replaying the opening movie (just in case he bypassed it unintentionally in a frenzy of button-pushing while trying to make the game load faster)

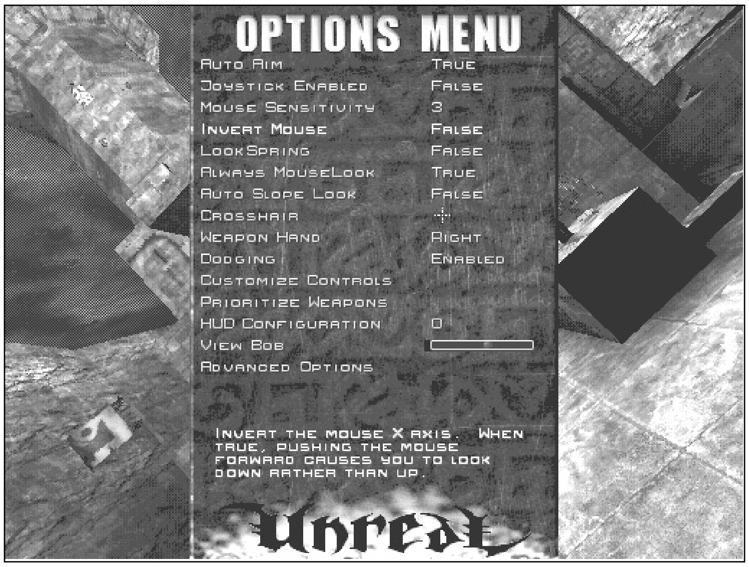
If you have an opening movie, let the player bypass it with a keystroke or button-click.

Customizable Controls

Give the player as much control over the interface as possible. Make everything as adjustable as you can. This includes game controls, monitor settings, volume . . . everything. Give him the best default settings you can arrive at, but then let him change whatever he wants.

Different things are important to different players. One player might want to optimize for speed instead of graphics, because he's an action addict. Another might prefer a higher resolution, even though it slows down the game, because he likes to look at the pictures. A third might want to remap the commands to different buttons or keys, because that's what he's used to. Whenever possible, let the player customize the game to his liking.

On the options screen, explain what each option does (see Figure 2.3). Don't assume that the player knows what gamma intensity or mouse inversion is. Explain each feature or setting, and tell him how changing it will affect the game.



Auto Aim	True
Joystick Enabled	False
Mouse Sensitivity	3
Invert Mouse	False
LookSpring	False
Always MouseLook	True
Auto Slope Look	False
Crosshair	+
Weapon Hand	Right
Dodging	Enabled
Customize Controls	
Prioritize Weapons	
HUD Configuration	0
View Bob	
Advanced Options	

Invert the mouse X axis. When true, pushing the mouse forward causes you to look down rather than up.

Figure 2.3 The highlighted option on this screen from *Unreal* is explained at the bottom of the screen so the player knows what it does. Used with permission of Epic Games Inc.

Cheat Codes

Include as many cheat codes as you can, while acknowledging that they break the play-balancing rules.

Entire third-grade classes are playing *Age of Empires*. Do they play by the rules? No, but those nine-year-olds are enjoying the game anyway. Not only that, but every one of them made his parents buy him a copy, and many of them will continue buying games in the future. This is a good thing for game designers everywhere.

In other words, let the player decide what's fun for himself. If he wants to get the biggest monster weapon there is and go around flattening everyone else with no challenge at all, *let him*. In an action game, include god mode and the cheats to get all the weapons or walk through walls. In an adventure game, give him the cheats to get around puzzles.

These cheats need to be tested, however. The more hierarchical the game (especially something like an adventure game), the more the game designer depends on the player having followed a certain path to get to where he is. If you let him jump there directly via a cheat, make sure that the cheat simultaneously sets all the game parameters as if he arrived there legitimately, especially in terms of objects in his inventory and flags set in the environment.

If you can't preserve the design integrity of the game while allowing the player to cheat, let him do it anyway, but make sure that he knows he's breaking the rules. You need to let customer support know about this, however. They're sure to get calls from confused gamers who've used the cheat codes and can't figure out why the game isn't behaving the way it should. (Even though you warned them!)

Tutorial or Practice Mode

Some players like to jump right into a game. Others need a chance to get their feet wet in a nonthreatening atmosphere.

A tutorial gives the player hands-on experience without endangering him. A good example of this is Lara Croft's house in the original *Tomb Raider*, where the player is slowly introduced to the running, jumping, and climbing skills he'll need later in the game. If he fails at any one stage, he can simply try again. Even if he succeeds, he can go back and do it again to become more comfortable or to learn the limits of the move.

The *Tomb Raider* tutorial is especially good because, in addition to teaching those skills, it also introduces the player to the character and the world. Thus, he's hooked from the start.

You cannot assume that the player will actually play the tutorial, however. (You also can't assume that he's read the manual. Many people just slam the game disc in and start it right up.) This presents difficulties if he must learn a certain skill in order to advance, but he hasn't gone to the right place to learn it. Probably the best way around this is to have some other character show him this skill, or tell him that there's something he clearly hasn't learned and that he should go back to basic training (or whatever it's called in the game) in order to pick it up.

Structure and Progression

"A game should be easy to learn, but difficult to master."

It's the cliché you hear most often about game design.

It's true.

It's true for arcade games, where the entire design philosophy is built on getting players hooked quickly and then escalating the challenges just enough to continue sucking quarters out of their pockets. It's true for board games. It's true for console games, where you have young kids who pick up the controller and drop it in a second if they can't figure out how to play the game. It's true for PC games, where busy people want to get into the game experience as quickly as possible without a steep learning curve.

Easy to learn, difficult to master. Anyone can sit down at *Quake* and start shooting things. As he gains more experience, he realizes that if he stands in one place, he'll get killed, so he learns to start moving while shooting. Then he learns to circle-strafe. Then to shoot while running backwards. Then to figure out which weapons are better up close or far away. Then he learns to rocket jump. As he progresses, he learns the characteristics of each weapon. He learns to "lead" his opponent. Anyone can pick up *Quake* and start having a good time within minutes, yet the longer he spends mastering the game, the more enjoyable it becomes.

In an adventure game, you should make the first puzzles easy. In an action game, make the first opponents fall over when the player even looks at them. In a fighting game, give the player some easy attacks that are effective right away. In a racing game, make the controls easy enough that the player can get out onto the track and start moving around. Save the esoteric adjustments for later, or at least don't *require* the novice to customize his car before he's ready to do so.

"Let the game begin" should be your motto. The first few minutes of a game are like the first moments of a movie. They're supposed to grab the audience. If you don't get a player involved in your game within the first 15 minutes, you've probably lost him forever.

Later, after the player has figured out the basic gameplay mechanics, it's time to raise the ante. If the intermediate levels are too easy, people will lose interest in the game almost as quickly as they do if the first levels are too hard.

As the player advances through the game, slowly introduce the intricacies you've built into it. If he must acquire a special skill to defeat the boss monster late in the game, give him some lesser creatures to practice on in the intermediate levels. If a puzzle calls for an intuitive leap, scatter examples of that kind of leap elsewhere before the player encounters that puzzle.

The final levels should be the hardest of all. You must find that delicate balance between the challenging and the impossible. A game that's too hard is no fun. A game that's too easy is no fun either (except to the very young). People don't want to play a game they have no chance of winning, nor do they want to play a game that's so easy there's no challenge to it. The trick is to design something in-between. Something that frustrates the player just enough that he enjoys it.

This sense of gradually acquiring mastery over a game is a pleasure that cannot be had in traditional media. (One doesn't get better at watching TV, for example.) The feeling is more like learning a sport—increased skill brings increased pleasure. What you must do is design a game where the better the player plays, the more he wants to play it.

Throughout the process, you must listen to your testers. Remember that you're one of the most skilled people who will play your game. You know its ins and outs, its strengths and weaknesses, its guts. A level or puzzle you think is ridiculously easy can prove impossible to others. By cycling fresh testers onto the game, you get feedback that's impossible to obtain from more jaded testers who've played the game as much as or more than you. You need both kinds of testing to make the game successful.

Taking Care of the Player

You're not the player's adversary. Your job is to help him enjoy the game you've created. It's easy to lose sight of this, especially when so many of your tasks involve challenging the player and finding that delicate balance between frustration and pleasure. Remember, though, that you're not in competition with him. Although *his* goal may be to beat the game, *your* goal is not to beat *him.*

One of the biggest mistakes young designers make is trying to prove that they're smarter than the player. There's no point to this, and it would be an unfair fight anyway, because you hold all the cards. Anyone can design a puzzle based on obscure knowledge or create a path that's almost impossible to traverse. The skill comes in creating problems that are just challenging enough, and in confining the player's frustration to the problem at hand, rather than making the playing of the game itself a challenge.

A good designer tries to help players get through the game, take care of them along the way, and protect them from time-wasting traps and pitfalls that take the fun right out of the whole thing.

Dead Man Walking

Don't put your player in a position where he can't win and doesn't know it. This is a familiar problem in adventure game design, but the phenomenon is now creeping into action games.

Let's suppose that in the fourth level of a game, the player needs some special goggles to spot the laser beams crisscrossing a narrow hallway. The designer put the goggles behind a crate on level 2, but the player didn't see them because it was dark back there. He went merrily on his way through level 3 and halfway through level 4, unaware that he missed something. Now he runs into this laser beam problem. He does everything he can to get around it. He becomes frustrated. He finally buys the strategy guide and learns for the first time that a) There's equipment he's missed, b) He's going to have to go back and get it, and, worst of all, c) He's been a "dead man walking" since the middle of level 2, and the last several hours of gameplay have been wasted.

What's the mindset of the player? He might decide that it's not worth it to go back and play through the game again. Even if he does continue, from there on out, he's going to play with one eye on the strategy guide, never trusting the designer to do the right thing again.

Instead, you want the player to trust you, to believe at any given moment that if he does the right thing, he can somehow win.

Protect Newbies

When the game begins, take it easy on the player. Ease him in until he acquires some confidence. Nothing is worse than to be a newbie in an online game and have some experienced player come along and kill you. Nothing is worse in a first-person shooter than to have the first set of opponents kill you over and over, while you struggle just to move around and can't figure out why you're always staring at the ceiling. Nothing is worse in an adventure game than a first puzzle that's so hard that you're made to feel like an idiot.

All these problems have design solutions. Devise a punishment for experienced players who kill newbies, or cordon off an area of the game where new players can fumble around safely. Make the first opponents easy. Make the first puzzle even easier.

Play It Again, Sam

Have you ever played a game where you come to a long, tricky sequence of moves that you have to get just right? And every time you get one move wrong, you die and have to go back and do the whole thing again? Often, entire levels have to be played this way. No matter how close the player is to completion, one false step sends him back to start over.

A special section of Hell should be reserved for game designers who do this. These same designers probably think that Sisyphus clapped his hands with glee every time his rock rolled back down to the bottom of the hill.

It needs to be stated once and for all, unequivocally, with no room for doubt:

This is not fun.

As the player repeats the sequence, pushing one step deeper into it each time, he comes to resent it, as well he should. He's demonstrated his ability to complete the early steps—why should he be condemned to repeat them? Why should he spend his time doing the same thing over and over again? What's new and exciting about that? Where's the entertainment?

Nothing makes a player want to fling down the controller and put his fist through the screen like dying for the hundredth time near the end of one of these sequences and having to go back and do the whole damn thing *again*! Nobody gains anything by this torture.

This problem has many solutions. The simplest is to allow the player to save the game at any point. That way he never has to return to the very start, but can pick up from just before he was killed.

Another solution is to code in checkpoints along the way. If the player dies, quickly cycle him back to the last checkpoint so that he can have another go at it.

The best solution is to avoid designing one of these sequences in the first place.

Give the Player the Information He Needs

All the knowledge a player needs in order to play a game should be included within that game. You cannot expect him to rely on strategy guides, Web sites, or word of mouth to pick up critical information. Whenever possible, the information should be on the disc itself, rather than in the manual. However, some games (especially simulations) are so complex that a beefy manual cannot be avoided.

Many games have *undocumented features*, special moves or tricks that aren't mentioned anywhere in the manual. These frills can be fun, but you must make sure they aren't essential to the completion of the game, because not everyone will discover them.

It's also a tricky problem for a designer to figure out what "everyone knows." It used to be that gamers were computer hobbyists with a scientific or mechanical bent, and you could rely on them to have a certain body of common knowledge. You could make jokes based on pop-culture references and devise puzzles using the order of colors in the spectrum. Now the person playing your game could be a teenager in Italy or a grandmother in Sweden. You can't be sure what these people know. So if your gameplay relies on specialized knowledge, you must make that knowledge available to the player.

Reduce Player Paranoia

Players spend much of a game worrying that they're doing the wrong thing. You need to reassure them when they're doing okay. Give them small, incremental rewards as they make progress toward their goals. Gently steer them in the right direction, and let them know when they're straying from the path.

Offer Levels of Difficulty

Another way of taking care of the player is to include different levels of difficulty in your game. These usually come in three flavors—novice, intermediate, and expert. In an action game, for example, the novice level has fewer opponents, who die more easily and might not have the smartest AI. At the intermediate level, you can provide less ammunition for the weapons you give the player, or make the player's health packs less powerful. At the expert level, there are many opponents with good AI, sparse ammunition, and perhaps no armor or health kits.

Similar gradations can be made in other genres. Driving games can vary the performance of the other cars, adventure games can scatter more clues to the puzzles, and sports games can demand greater or lesser adherence to the rules. In every case, you're allowing the player to choose the degree of challenge that will provide the most enjoyment for him.

How to Design

With all these principles in mind, how do you actually go about designing your game?

Create an Integrated Whole

Game design doesn't have to be as hard as people make it out to be. Once you have an original inspiration (the high concept), a lot of the design process is mere logic. If you've settled on the central nugget around which the game revolves, a lot of design comes down to iteratively answering this question: "For this interesting thing to be true, what *else* has to be true?"

As you answer this question over and over, your world slowly grows into an integrated whole. "If we're doing a farming simulation, we're going to need tools. How does the player use these tools? What do they look like on the screen? How does the player select them? How does he manipulate them? What does he see after he's used them?" And so on.

Economy of Design

Good design in any field is distinguished by simplicity (see Figure 2.4). A good designer includes only those things that are necessary to create the effect he desires. Anything else is superfluous and detracts from the overall goal.

In game development, economy of design also helps your schedule and budget. If you know exactly what you want to build, you won't waste time and money creating material that ends up on the cutting room floor.

The high concept is also useful in this regard. While a project is in development, features will pop up that the team wants to implement. You can't do *everything*, so how do you decide what to put in and what to leave out? One very good benchmark is to assess the proposed feature against the high concept. If it doesn't help you achieve the game's basic goal, leave it out.

The best games aren't big and sprawling; they're tight and focused. They don't distract the player with irrelevancies.

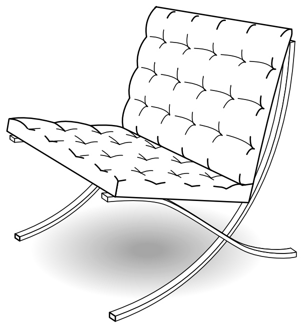


Figure 2.4 The Barcelona Chair, designed by Mies van der Rohe (originator of the phrase "Less is more").

Where Do You Get Your Ideas?

This is the question most frequently asked of writers and game designers. The answer is that if you're designing something of interest to *you*, the ideas will come naturally. This was mentioned in the preceding chapter, but it bears repeating. The game should be about something you're interested in, and it should be in a genre you're familiar with.

However, even though one person might set the game design in motion, there must still be a balance between his ideas and those of the rest of the team. Every game needs a "vision guy"—the person who holds the central idea in his head and evaluates all proposals and suggestions against that idea. In some cases, it's not even the game designer who fills this role, but the project leader or the producer.

One of the major tenets of this book is that no one person can come up with all the creativity necessary to make a game successful. Game design is a collaborative art, and you need contributions from all the disciplines, including story, art, programming, gameplay, sound, music—even sales and marketing. Everyone involved in the production of the game has a claim on the design, and the design process must be flexible enough to include each person's contributions.

Some endorse the *cabal* approach. This method sets up highly focused teams, each of which addresses one specific area. Each group usually includes one member from each of the areas of production (programmer, artist, level designer, tester, and so on). They have a series of meetings and are empowered to make decisions on behalf of the entire team.

A less formal approach to group design is brainstorming. In *The Art of Innovation*, Tom Kelley writes about the success of IDEO, a Silicon Valley product development firm. "The best way to get a good idea," Kelley writes, "is to get *lots* of ideas." A brainstorming session gathers team members in an open discussion that generally follows established rules. No ideas are labeled good or bad; they're merely recorded. No decisions are made during the meeting; the group tries to get a flow going, and they record the session without interfering with it. For example, they might photograph the whiteboard as they go along.

Keep the size of your brainstorming group small, preferably fewer than seven people. Larger groups tend to ramble and are less productive. Small groups stay more focused, kicking around many variations of one idea before moving on to the next.

When a brainstorming session is over, the designer is free to mull over the ideas, accepting some and rejecting others. This system takes advantage of the creativity of the entire team, but relies on the vision guy to keep the focus.

Finally, don't talk about your ideas too soon. A small flame can easily be extinguished by a puff of wind. If it grows into a fire, the stronger the wind, the more fiercely it burns. When an idea is born, it can easily be extinguished by a single puff of derision. After it's established, though, conversational buffeting only makes it stronger, requiring it to adapt, change, and grow. When you get an idea, nurture it along before exposing it to the winds of discussion. When the spark becomes a fire, bring it out for others to see, and then you'll discover whether it's strong enough to survive.