CHAPTER FIFTEEN

Players Play Games through an *Interface*

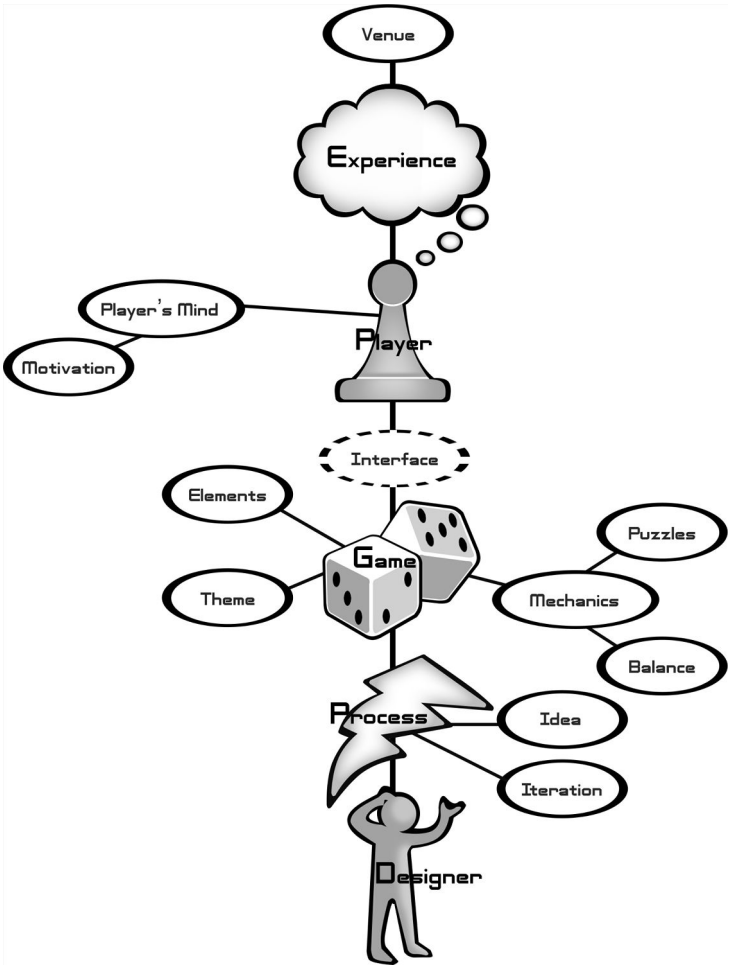


FIGURE 15.1

Between Yin and Yang

FIGURE
15.2

Remember in Chapter 10 when we talked about the strange relationship between player and game? Specifically, that the player puts their mind inside the game world, but that game world really only exists in the mind of the player? This magical situation, which is at the heart of all we care about, is made possible by the game interface, which is where player and game come together. Interface is the infinitely thin membrane that separates white/yang/player and black/yin/game. When the interface fails, the delicate flame of experience that rises from the player/game interaction is suddenly snuffed out. For this reason, it is crucial for us to understand how our game interface works and to make it as robust, as powerful, and as invisible as we can.

Before we proceed, though, we should consider the goal of a good interface. It isn't "to look nice" or "to be fluid," although those are nice qualities. The goal of an interface is to make players feel in control of their experience. This idea is important enough that we should keep a lens around for frequent examination of whether a player feels in control.

Lens #59: The Lens of Control

This lens has uses beyond just examining your interface, since meaningful control is essential for immersive interactivity. To use this lens, ask yourself these questions:

- When players use the interface, does it do what is expected? If not, why not?

- Intuitive interfaces give a feeling of control. Is your interface easy to master or hard to master?
- Do your players feel they have a strong influence over the outcome of the game? If not, how can you change that?
- Feeling powerful = feeling in control. Do your players feel powerful? Can you make them feel more powerful somehow?

Breaking It Down

Like many things we encounter in game design, interface is not simple or easily described. "Interface" can mean many things—a game controller, a display device, a system of manipulating a virtual character, the way the game communicates information to the player, and many other things. To avoid confusion and to understand it properly, we need to separate it out into component parts.

Let's work from the outside in. Initially, we know that we have a player and a game world.

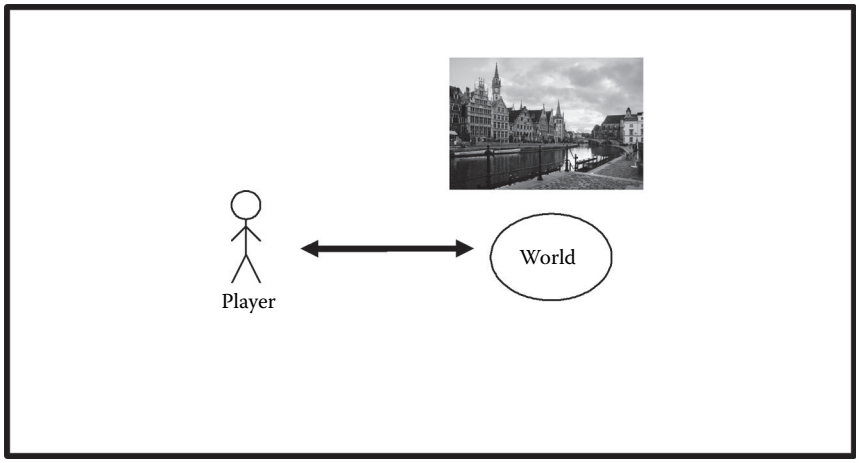
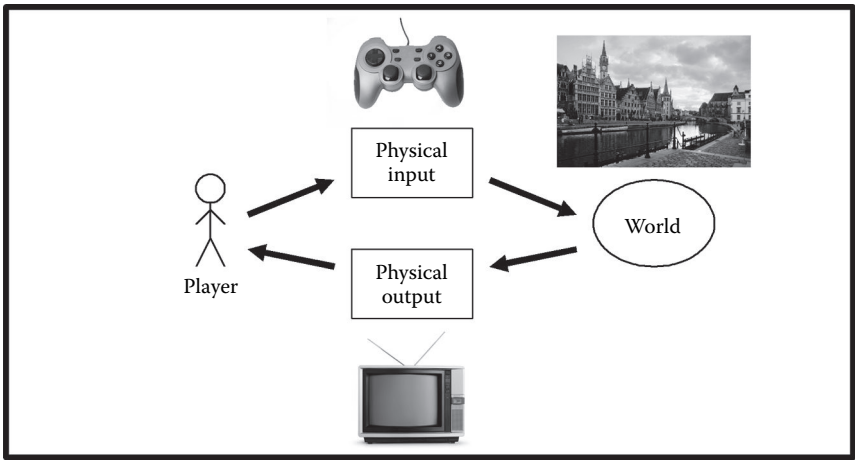


FIGURE 15.3

On the simplest level, the interface is everything that is in between them. So, what is in there? There is some way that the player touches something to make changes in the world. This could be by manipulating pieces on a game board or by using a game controller or keyboard and mouse. Let's call this **physical input**. And similarly, there is some way the player can see what is going on in the game world. It could be by looking at a game board, or it could

be some kind of display screen with audio or other sensory output. Let's call this **physical output**. So we have the following:

FIGURE 15.4



This looks pretty simple and is the way most people naively think about game interface. But some important things are missing from this picture. While there are times when the physical input and output are directly connected to elements in the game world, there are other times that there is some amount of intermediate interface. When you play *Pac-Man* and there is a score display at the top of the screen, this is not really part of the game world—it is really part of the interface. The same goes for menus and buttons on mouse-based interfaces or when you hit an enemy for ten points of damage and a stylized "10" floats out of his body. When you play most 3D games, you do not see the entire world, but instead you see a view into the world from a virtual camera with a position in the virtual space of the game world. All these things are part of a conceptual layer that exists between the physical input/output and the game world. This layer is usually called **virtual interface** and has both input elements (such as a virtual menu where the player makes a selection) and output elements (such as a score display) (see Figure 15.5).

Sometimes, the virtual layer is so thin that it is almost nonexistent, but other times, it is very dense, full of virtual buttons, sliders, displays, and menus that help the player play the game, but aren't part of the game world. This virtual layer must be handled delicately, for, as designer Daniel Burwen notes, the less abstraction there is in an interface, the more emotional connection we feel to our content.

And that makes a pretty complete picture of the major interface elements involved in a game. But we've left out something crucial to the design of any game interface: **mapping**. On every arrow on the right side of the diagram, some special things are happening—it is not as if data are simply passed through—rather, these data go through a special transformation based on how the software is designed. Every one of the arrows on the game side represents a separate piece of computer code. How all this behaves together in composite defines the interface for your game.

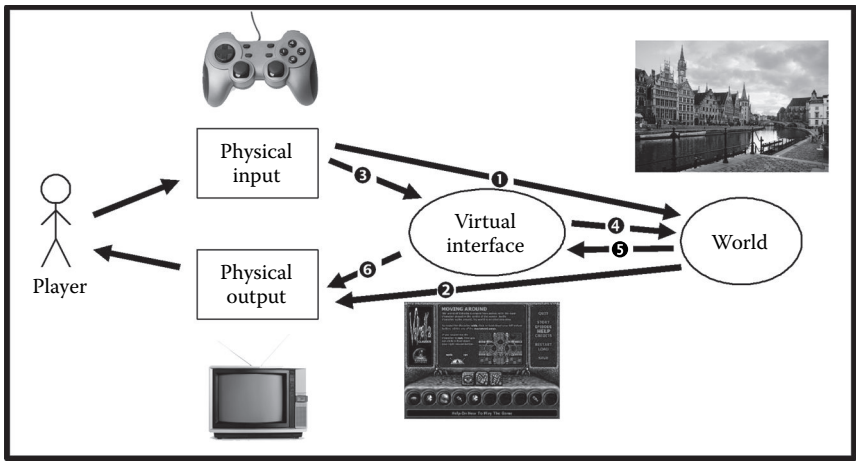


FIGURE 15.5

Some quick examples of the kinds of logic that can be contained in each of those six arrows are as follows:

1. **Physical Input → World**: If pushing a thumbstick makes my avatar run, the mapping tells how fast it will run and how quickly it will slow down if I let go. If I push the thumbstick harder, does my character run faster? Will my character accelerate over time? Will "double tapping" the thumbstick make my character dash?
2. **World → Physical Output**: If you cannot see the entire world at once, what parts of it can you see? How will it be shown?
3. **Physical Input → Virtual Interface**: In a mouse-based menu interface, what does clicking do? What does double-clicking do? Can I drag parts of the interface around?
4. **Virtual Interface → World**: When the player manipulates the virtual interface, what effect does this have on the world? If they select an item in the world and use a pop-up menu to take an action on it, does that action take effect immediately or after some delay?
5. **World → Virtual Interface**: How are changes in the world manifested in the virtual interface? When do scores and energy bars change? Do events in the world lead to special pop-up windows or menus or mode changes in the interface? When players enter a battle, will special battle menus appear?
6. **Virtual Interface → Physical Output**: What data are shown to the player, and where does it go on the screen? What colors will it be? What fonts? Will hit points pulse or make a sound when they are very low?

For close examination of these six types of connections, we introduce two new lenses.

Lens #60: The Lens of Physical Interface

Somehow, the player has a physical interaction with your game. Copying existing physical interfaces is an easy trap to fall into. Use this lens to be sure that your physical interface is well suited to your game by asking these questions:

- What does the player pick up and touch? Can this be made more pleasing?
- How does this map to the actions in the game world? Can the mapping be more direct?
- If you can't create a custom physical interface, what metaphor are you using when you map the inputs to the game world?
- How does the physical interface look under the Lens of the Toy?
- How does the player see, hear, and touch the world of the game? Is there a way to include a physical output device that will make the world become more real in the player's imagination?

The world of videogames occasionally goes through dry spells where designers feel it is not feasible to create custom physical interfaces. But the marketplace thrives on experimentation and novelty, and suddenly specially crafted physical interfaces, like the *Dance Dance Revolution* mat, the *Guitar Hero* guitar, and the Wiimote appear bringing new life to old gameplay by giving players a new way to interact with old game mechanics.

Lens #61: The Lens of Virtual Interface

Designing virtual interfaces can be very tricky. Done poorly, they become a wall between the player and the game world. Done well, they amplify the power and control a player has in the game world. Ask these questions to make sure that your virtual interface is enhancing player experience as much as possible:

- What information does a player need to receive that isn't obvious just by looking at the game world?
- When does the player need this information? All the time? Only occasionally? Only at the end of a level?
- How can this information be delivered to the player in a way that won't interfere with the player's interactions with the game world?

- Are there elements of the game world that are easier to interact with using a virtual interface (like a pop-up menu, for instance) than they are to interact with directly?
- What kind of virtual interface is best suited to my physical interface? Pop-up menus, for example, are a poor match for a gamepad controller.

Of course, these six kinds of mapping cannot be designed independently—they must all work in unison to create a great interface. But before we move on, we must consider two other important kinds of mapping, represented by the arrows that come and go from the player or, more specifically, from the player's imagination. This is when a player becomes so immersed in a game, he or she is no longer pushing buttons and watching a television (TV) screen, instead, he or she is running, jumping, and swinging a sword. And you can hear this in a player's language. A player generally won't say, "I controlled my avatar, so he ran to the castle, and then I pressed the red button to make him throw a grappling hook, then I started tapping the blue button to make my avatar climb up." No, a player describes the gameplay this way: "I ran up the hill, threw my grappling hook, and started climbing the castle wall." Players project themselves into games and on some level disregard that the interface is there at all, unless it suddenly becomes confusing. A person's ability to project consciousness into whatever they are controlling is almost alarming. But it is only possible if the interface becomes second nature to the player, and this gives us our next lens.

Lens #62: The Lens of Transparency

> *No matter how beautiful your interface is, it would be better if there were less of it.*
>
> —Edward Tufte

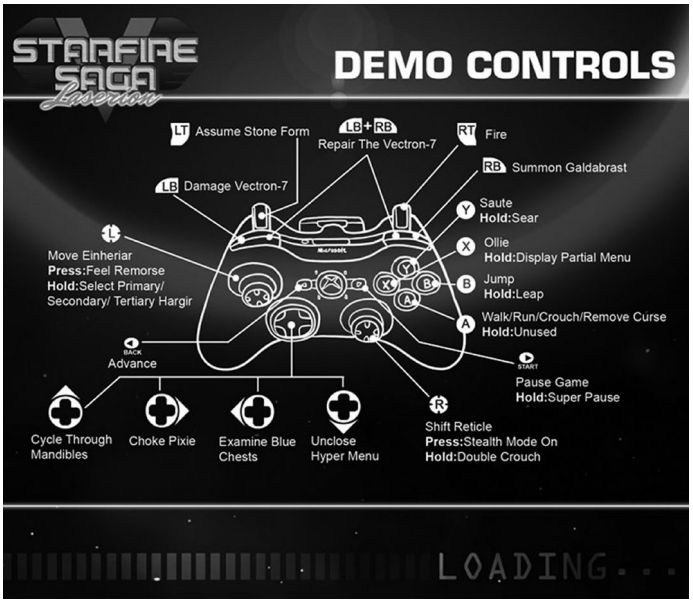
The ideal interface becomes invisible to the player letting the player's imagination be completely immersed in the game world. To ensure invisibility, ask yourself these questions:

- What are the player's desires? Does the interface let the players do what they want?
- Is the interface simple enough that with practice, players will be able to use it without thinking?
- Do new players find the interface intuitive? If not, can it be made more intuitive, somehow? Would allowing players to customize the controls help or hurt?

- Does the interface work well in all situations, or are there cases (near a corner, going very fast, etc.) when it behaves in ways that will confuse the player?
- Can players continue to use the interface well in stressful situations, or do they start fumbling with the controls or missing crucial information? If so, how can this be improved?
- Does anything confuse players about the interface? On which of the six interface arrows is it happening?
- Do players feel a sense of immersion when using the interface?

This interface, a parody from the web comic *Penny Arcade*, is probably not transparent.

FIGURE 15.6



(From www.penny-arcade.com. Used with permission.)

The Loop of Interaction

Information flows in a loop from player to game to player to game, round and round. It is almost like this flow pushes a waterwheel that generates experience when it spins. But it can't be just any information that flows around this loop. The information that is returned to the player by the game dramatically affects what the player will do next. This information is generally called **feedback**, and the quality of this feedback can exert a powerful influence on how much the player understands and enjoys what is happening in your game.

FIGURE
15.7

The importance of good feedback is easily overlooked. One example is the net on a basketball hoop. The net does not affect the gameplay at all—but it slows the ball as it descends from the hoop, so that all players can clearly see, and even hear, that it went in.

A less obvious example is the Swiffer (Figure 15.7), a simple device designed to be a better solution for cleaning floors than the traditional broom/dustpan combination. Some people who have attempted to redesign the broom and dustpan have merely modified the existing solution, making a pan that clips to the broom handle, making sturdier broom bristles, adding a lid to the dustpan, etc. It would appear that the designers of the Swiffer used Lens #14, *Problem Statement*, to invent a brand new solution. If we look at some of the problems with the broom/dustpan solution,

Problem #1: It's impossible to sweep all the dust into the dustpan.

Problem #2: When standing, the dustpan is hard to use. When crouching, the broom is hard to use.

Problem #3: The broom doesn't really get all the dust.

Problem #4: Your hands get kind of dirty when you try to sweep dust into the dustpan.

Problem #5: Transferring the dirt from the dustpan to the trash is perilous—it often spills or blows around.

We see that the Swiffer, with its disposable cloth, solves these problems fairly well:

Solution #1: No dustpan is needed.

Solution #2: There is no need to crouch when using the Swiffer.

Solution #3: The Swiffer cloth captures far more dust than a broom can.

Solution #4: Your hands stay clean.

Solution #5: The cloth is easily disposed of.

So, the Swiffer solves a lot of problems, which makes it very appealing. But it has an appeal beyond these practical things. It has a strong psychological appeal—frankly, it is fun to use. Why? Because the design addresses problems that most people wouldn't state as problems. Here is an example:

Problem #6: The user gets little feedback about how well they have cleaned the floor.

Unless a floor is really dirty, it is hard to see whether your sweeping is making any difference just by looking at the floor. You might say, "Who cares? All that matters is how well it cleans, right?" But this lack of feedback can make the entire task feel somewhat futile, which means that the user enjoys it less and will clean their floor less often. In other words, less feedback = dirtier floor. But the Swiffer solves this problem very well:

Solution #6: The dirt you have removed from the floor is clearly visible on the cleaning cloth when you are done.

This feedback shows the user quite clearly that what they have done makes a real difference in how clean the floor is. This triggers all kinds of pleasures—satisfaction of having done something useful, the pleasure of purification, and even the pleasure of having secret knowledge that others cannot see. And though this feedback doesn't come until the end of the task, the user comes to anticipate it and looks forward to seeing this concrete evidence of a job well done.

Lens #63: The Lens of Feedback

The feedback a player gets from the game is many things: judgment, reward, instruction, encouragement, and challenge. Use this lens to be sure your feedback loop is creating the experience you want by asking these questions at every moment in your game:

- What do players need to know at this moment?
- What do players want to know at this moment?
- What do you want players to feel at this moment? How can you give feedback that creates that feeling?
- What do the players want to feel at this moment? Is there an opportunity for them to create a situation where they will feel that?
- What is the player's goal at this moment? What feedback will help them toward that goal?

Using this lens takes some effort, since feedback in a game is continuous but needs to be different in different situations. It takes a lot of mental effort to use this lens in every moment of your game, but it is time well spent, because it will help guarantee that the game is clear, challenging, and rewarding.

FIGURE
15.8

Experiences without feedback are frustrating and confusing. At many crosswalks in the United States, pedestrians can push a button that will make the DON'T WALK sign change to a WALK sign so they can cross the street safely. But it can't change right away, since that would cause traffic accidents. So the poor pedestrian often has to wait up to a minute to see whether pressing the button had any effect. As a result, you see all kinds of strange button-pressing behavior: some people push the button and hold it for several seconds; others push it several times in a row, just to be safe. And the whole experience is accompanied by a sense of uncertainty—pedestrians can often be seen nervously studying the lights and DON'T WALK sign to see if it is going to change, because they might not have pushed the button correctly.

What a delight it was to visit the United Kingdom and find that in some areas the crosswalk buttons give immediate feedback in the form of an illuminated WAIT sign that comes on when the button has been pushed and turns off when the WALK period has ended (Figure 15.9)! The addition of some simple feedback turned an experience where a pedestrian feels frustrated into one where they can feel confident and in control.

Generally, it is a good rule of thumb that if your interface does not respond to player input within a tenth of a second, the player is going to feel like something is wrong with the interface. A typically problematic example of this often appears when you make a game with a "jump" button. If the animator working on the jump animation is new to videogames, he is very likely to put a "wind up" or "anticipation" on the jump animation, where the character crouches down, getting ready to jump, for probably one-quarter to one-half a second. This is sound animation practice, but because this breaks the tenth of a second rule (I push the jump button, but my character doesn't actually end up in the air until a half second later), it drives players crazy with frustration.

FIGURE
15.9

Helpful feedback.

Juiciness

But let's return to our sweeping example: a dirty cloth is not the only feedback that the Swiffer gives the user. Let's consider another problem with the broom and dustpan that most people would be unlikely to state.

Problem #7: Sweeping is boring.

Well, of course it is! It's sweeping! But what do we mean by boring? We need to break this down further. Specifically:

- Sweeping is repetitive (same motion over and over).
- It requires you to focus your attention on something with no surprises (if you don't monitor that little pile of dust, it goes everywhere).

How does the Swiffer meet this challenge?

Solution #7: Using the Swiffer is fun!

This may well be the single biggest selling point of the Swiffer. In TV advertisements for the Swiffer, they show people joyously dancing through houses cleaning

floors, and some ads featured people picking up the Swiffer out of sheer curiosity and then cleaning the floors while playing with the Swiffer like a child plays with a toy. And the Swiffer does very well under Lens #17, *The Toy*—it is fun to play with… but why? It's just a cloth on a stick, right? Yes, in one sense, but the base of the Swiffer, where the cloth goes, is attached to the stick with a special sort of hinge, so that when you rotate your wrist, even slightly, the base that holds the cloth rotates dramatically. A little motion from my wrist makes the cleaning mechanism move easily, fluidly, and powerfully—getting into exactly the position you want it to be in with a minimum of effort. Using it feels kind of like running a magic race car around the floor of your house. The motion that the cleaning base shows is **second-order motion**, that is, motion that is derived from the action of the player. When a system shows a lot of second-order motion that a player can easily control and that gives the player a lot of power and rewards, we say that it is a **juicy** system—like a ripe peach, just a little bit of interaction with it gives you a continuous flow of delicious reward. Juiciness is often overlooked as an important quality in a game. To avoid overlooking it, use this lens.

Lens #64: The Lens of Juiciness

To call an interface "juicy" might seem kind of silly—although it is very common to hear an interface with very little feedback described as "dry." Juicy interfaces are fun the moment you pick them up. To maximize juiciness, ask yourself these questions:

- Is my interface giving the player continuous feedback for their actions? If not, why not?
- Is second-order motion created by the actions of the player? Is this motion powerful and interesting?
- Juicy systems reward the player many ways at once. When I give the player a reward, how many ways am I simultaneously rewarding them? Can I find more ways?

We discussed in Chapter 4 how the difference between work and play is one of attitude. I chose this nongame example of the Swiffer as an illustration because the feedback it gives is so powerful that it changes work into play. And it is important for your interface to be fun, if possible—since your game is meant to be fun and you run the risk of creating inner contradictions and a self-defeating experience if you put a dry, painful interface as the player's gateway to your supposedly fun experience. Remember, fun is pleasure with surprises, so if your interface is going to be fun, it should give both.

Primality

One kind of interface that tends to be associated with juicy fun is the touch interface found on phones and tablets. Touch interfaces have done a lot to change the world of gaming in a very short time. Young children, in particular, seem to take to touch interfaces with surprising ease. But why? The obvious answer is "because they are intuitive." But that's really a pretty vaporous answer, since the definition of "intuitive" is "easy to understand." So the question becomes "why is it that touch interfaces are so easy to understand?" And the answer is this: they are primal.

Until the advent of touch computing, every computer interface took the form of tool use. I would interact with some physical object (keyboard, mouse, button panel, punch card, etc.), and some remote (not near my finger) response would take place. Gradually, like with all tools, we learn how they work and become accustomed to them. But tool use is not primal, by which I mean, prehuman. Humans started using tools about three million years ago, which is a pretty good run. But still animals have been touching things, intuitively, for much longer: probably something like 300 or 400 million years. And our brains, of course, are evolved from those brains. When you think about the three-layer structure of the human brain, it becomes clearer—the lowest-level "reptilian" section of the brain is able to process touch, but tool use probably requires help from the neocortex, the highest level of the brain.

When you think of it that way, it becomes obvious why touch is more intuitive than using a mouse or game controller. But then, of course, it raises a broader question—what parts of my game are primal, and what parts require higher brain function? It seems certain that the more you can engage and involve the primal parts of the brain, the more intuitive and powerful your gameplay will feel, which helps to explain why so many games contain elements such as the following:

- Gather fruit-like items.
- Fight a threatening enemy.
- Find your way through an unfamiliar environment.
- Overcome obstacles to get to a mate (how scientists say "rescue the princess").

To really know what parts of the brain are involved in a given activity, you need to be a brain scientist doing MRI research. But to make an educated guess about whether your interface and game activity has low-level primality, just think about whether it is something that animals can do. If they can, there's a good chance you are tapping into the power of primality.

Lens #65: The Lens of Primality

Some actions and interfaces are so intuitive that animals were doing them hundreds of millions of years ago. To capture the power of primality, ask yourself these questions:

- What parts of my game are so primal an animal could play? Why?
- What parts of my game could be more primal?

Channels of Information

One important goal of any interface is to communicate information. Determining the best way for your game to communicate necessary information to the player requires some thoughtful design, since games can often contain a great deal of information and often much of it is needed at the same time. To figure out the best way to present the information in your game, try following these steps. Referring back to our interface data flow diagram from the beginning of the chapter, we are mostly talking about arrows 5 (World → Virtual Interface) and 6 (Virtual Interface → Physical Output).

Step 1: List and Prioritize Information

A game has to present a lot of information, but it is not all equally important. Let's say we were designing the interface for a game similar to the classic NES game, *Legend of Zelda*. We might begin by listing all of the information the player needs to see. A simple unprioritized list might look like

1. Number of rubies
2. Number of keys
3. Health
4. Immediate surroundings
5. Distant surroundings
6. Other inventory
7. Current weapon
8. Current treasure
9. Number of bombs

Now, we might sort these by importance:

Need to know every moment:

4. Immediate surroundings

Need to glance at from time to time while playing:

1. Number of rubies
2. Number of keys
3. Health
5. Distant surroundings
7. Current weapon
8. Current treasure
9. Number of bombs

Need to know only occasionally:

6. Other inventory

Step 2: List Channels

A channel of information is just a way of communicating a stream of data. Exactly what the channels are varies from game to game—and there is a lot of flexibility in how you choose them. Some possible channels of information might be

- The top center of the screen
- The bottom right of the screen
- My avatar
- Game sound effects
- Game music
- The border of the game screen
- The chest of the approaching enemy
- The word balloon over a character's head

It can be a good idea to list out the possible channels that you think you might use. In *Legend of Zelda*, the main channels of information the designers settled on were

- Main display area
- Dashboard of information at the top of the screen

Also, they decided there would be a "mode change" the player could activate by hitting the "select" button (we'll discuss mode changes later in this chapter), which has different channels of information:

- Auxiliary display area
- Dashboard of information at the bottom of the screen

Step 3: Map Information to Channels

Now, the difficult task comes of mapping the types of information to the different channels. This is usually done partly by instinct, partly by experience, and mostly by trial and error—drawing lots of little sketches, thinking about them, and then redrawing them, until you think you have something worth trying out. In Zelda, the mapping is as follows:

Main display area:

4. Immediate surroundings

Dashboard of information at the top of the screen:

1. Number of rubies
2. Number of keys
3. Health
5. Distant surroundings
7. Current weapon
8. Current treasure
9. Number of bombs

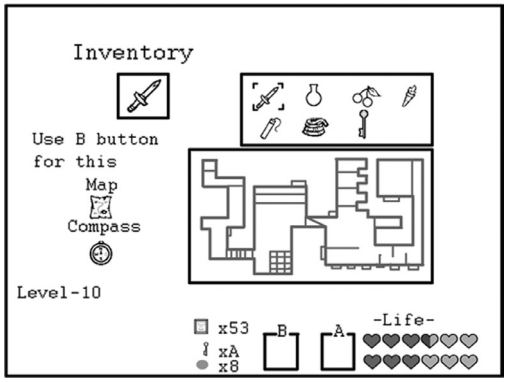
Auxiliary display area:

6. Other inventory

Taking a look at the main screen (Figure 15.11) and subscreen (Figure 15.10), you can see other interesting choices that were made:

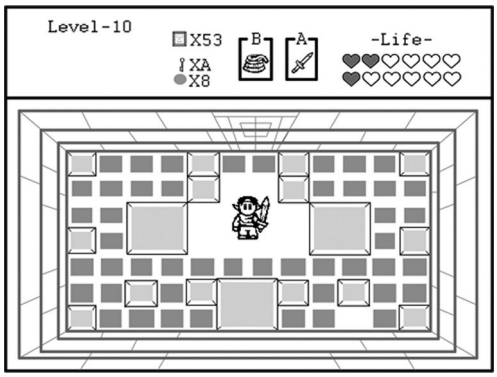
Note that the dashboard information is so important to gameplay that it needs to be shown all the time on both the main screen and subscreen. And the contents of that dashboard really involve seven different channels of information. Notice how they split them up—"life" was deemed so important that it got nearly one-third of the interface. Rubies, keys, and bombs, though their functions are different, each have to communicate a two-digit number, so they are all grouped together. The weapon and treasure you are holding are so important that they have little boxes around them. The "A" and "B" are reminders to the player about which buttons to hit in order to use these items.

FIGURE
15.10



NES Zelda subscreen.

FIGURE
15.11



NES Zelda main screen.

Also note on the inventory screen how extra space was used to give the player some instruction on how to use it.

You can see that even though this is a relatively simple interface compared to more modern games, there were many decisions the designer made about how to lay it out, and these decisions made a significant impact on the game experience.

Step 4: Review Use of Dimensions

A channel of information in a game can have several dimensions. For example, if you have decided to map "damage to an enemy" to "numbers that fly out of that enemy," you have several dimensions you can work with on that channel. Some of these might be

- The number you display
- The color of the number

- The size of the numerals
- The font of the numerals

Now you have to decide which of these dimensions, if any, you want to use. Surely you will use the first one, the number. But will the color mean anything? Perhaps you will use the other dimensions as **reinforcers** of the information—numbers under 50 will be white and small, numbers from 50 to 99 will be yellow and medium sized, but numbers 100 and over will be red and very large and in a special font to emphasize the amount of damage.

And while using multiple dimensions on a channel to reinforce a piece of information is a way to make the information very clear (and also kind of juicy), you could also take a different approach and decide to put different pieces of information on the different dimensions. For example, you might decide to color the numbers to indicate friend (white) or foe (red). Then you might make the size of the numbers indicate how close the character is to defeat—small numerals might mean the character has a lot of hit points left, while large numerals might mean they are about to die. This kind of technique can be very efficient and elegant. By using a single number, you have communicated three different pieces of information. The risk is that you have to educate the player on what these different dimensions on one channel of information represent, which might be difficult for some players to understand or remember. Good use of channels and dimensions is what makes for an elegant, well laid out interface, so we keep a special lens around for this kind of examination.

Lens #66: The Lens of Channels and Dimensions

Choosing how to map game information to channels and dimensions is the heart of designing your game interface. Use this lens to make sure you do it thoughtfully and well. Ask yourself these questions:

- What data need to travel to and from the player?
- Which data are most important?
- What channels do I have available to transmit these data?
- Which channels are most appropriate for which data? Why?
- Which dimensions are available on the different channels?
- How should I use those dimensions?

Modes

What is an interface mode? Simply put, it is a change in one of the mapping arrows (1–6) in our interface diagram. For example, if pressing the B button changes the

functionality of the gamepad so that instead of making your avatar run around, it makes your avatar aim a water hose, which is a mode change, the mapping on arrow #1 (Physical Interface → World) has just changed. Mode changes can happen as a result of mapping changes on any of the six arrows.

Modes are a great way to add variety to your game, but you must be very careful, since you run a risk of confusing the player if they don't realize that a mode change has occurred. Here are a few tips to avoid getting in trouble with interface modes.

Mode Tip #1: Use as Few Modes as Possible

The fewer the modes, the less chance a player is going to get confused. Having multiple interface modes isn't a bad thing, but you should add modes cautiously, for each one is something the new the player is going to have to learn and understand.

Mode Tip #2: Avoid Overlapping Modes

Just as we have channels of information from the game to the player, there are similar channels of information from the player to the game. Each button or thumbstick is a channel of information; for example, let's say you have a game that lets you change between walking mode (the thumbstick navigates) and throwing mode (the thumbstick aims). Later, you decide to add a driving mode as well (the thumbstick steers a car). What happens if the player changes into throwing mode while they are driving? You could try to allow this, potentially putting you into two modes at once (driving and throwing). And while this might work, it also might be a disaster if the thumbstick is simultaneously steering a car and controlling an aiming interface. It might be wiser to move the aiming, in all cases, to a second thumbstick, if your physical interface has one. By making your modes distinct and nonoverlapping, you keep yourself out of trouble. If you find you need to have overlapping modes, make sure they use different channels of information on the interface. For example, the thumbstick could have two navigating modes (flying or walking), and the button has two shooting modes (shoot fireball or lightning bolt). These modes are on completely different dimensions, so they can overlap safely—I can switch between shooting fireballs and shooting lightning bolts while either walking or flying with no confusing effects.

Mode Tip #3: Make Different Modes Look as Different as Possible

In other words, look at your modes with Lens #63, *Feedback*, and Lens #62, *Transparency*. If a player doesn't know what mode they are in, they are going to be confused and frustrated. The old Unix text editing system, *vi* (pronounced as "V. I."), was a symphony of confusing modes. Most people would expect that a text editor,

when it started up, would be in a mode that would allow you to enter text. But not so for *vi*. It was actually in a mode where each letter of the keyboard either would issue a command, like "delete line," or would put the editor into a new mode. But hitting these keys would give no feedback about what mode you were in. If you actually wanted to enter text, you had to type a letter "i," and then you would be in text insert mode, which looked exactly like command entry mode. It was impossible to figure out on your own, and even seasoned *vi* users would occasionally get confused about what mode they were in.

Some great ways to make your modes look different in video games are listed next.

- **Change something large and visible on the screen**: In *Halo* and most first-person shooters, when you change weapons, it is very visible. As a side note, the amount of ammo you have left is given through an interesting channel—it is right on the back of your gun.
- **Change the action your avatar is taking**: In the classic arcade game *Jungle King*, you go from a vine swinging mode to a swimming mode. Because your avatar is doing something so obviously different, it is clear that the mode has changed (also his hair changes color—that might be overkill).
- **Change the on screen data**: In *Final Fantasy* games and most RPGs, when you enter combat mode, many combat statistics and menus suddenly come up, and it is obvious there has been a mode change.
- **Change the camera perspective**: This is often overlooked as an indicator of a mode change, but it can be very effective.

Lens #67: The Lens of Modes

An interface of any complexity is going to require modes. To make sure your modes make the player feel powerful and in control and do not confuse or overwhelm, ask yourself these questions:

- What modes do I need in my game? Why?
- Can any modes be collapsed or combined?
- Are any of the modes overlapping? If so, can I put them on different input channels?
- When the game changes modes, how does the player know that? Can the game communicate the mode change in more than one way?

Other Interface Tips

Okay—we've covered interface data flow, feedback, channels, dimensions, and modes. That's a good start. But whole books have been written about the topic of interface design, and we have so many other interesting things to discuss; we must move on! But before we do, here are some general tips for making good game interfaces.

Interface Tip #1: Steal

More politely, we would call this the "top-down approach" to interface design. If you are designing an interface for a known game genre, say an action/platform game, you can begin with the interface of a known success in this area and then change it around to suit the things that are unique about your game. This can save you a lot of design time and has the benefit of being a familiar interface to your users. Of course, if your game has nothing new to offer, this will make it feel like a clone—but it is often surprising how one little change leads to another, which leads to another, and before you know it, your clone interface has morphed into something quite different.

Interface Tip #2: Customize

Also called the bottom-up approach, it is the opposite of stealing. With this approach, you design your interface from scratch, by listing out information, channels, and dimensions like we explained earlier. This is a great way to get an interface that looks unique and is custom to your particular game. If your gameplay is novel, you may find this is the only path available to you. But even if your gameplay is nothing new, you may be surprised when you try to build it from the bottom up—you may find that you invent a whole new way to play your game, because everyone else has just been copying what is successful and you took the time to actually examine the problem and tried to do a better job.

Interface Tip #3: Design around Your Physical Interface

The world of videogame development features platforms with radically different interfaces: touch interfaces, motion interfaces, mouse and keyboard, gamepads, and even virtual reality helmets. It is tempting to make games that can work equally well on all these platforms, so that you can sell them to as many people as possible. But the truth is that trying to design a game independent of any particular interface is usually a path to a dull game. Think about *Angry Birds*—its megasuccess was

partly due to the fact that it used the touch interface so well. Remember the Lens of the Toy? If the core interaction of your game is a unique type of play that takes advantage of what is unique to that physical interface, it can get enough attention to make giving up those other platforms more than worthwhile.

Interface Tip #4: Theme Your Interface

Often it is a different artist who designs the interface artwork than the one who designs the game world. In Chapter 6, "Theme," we talked about the importance of theming everything, and interface is no exception. Go over every inch of your interface with Lens #11, *Unification*, and see if you can find a way to tie it all together with the rest of the experience.

Interface Tip #5: Sound Maps to Touch

Usually, when we think of using sound in a game, we think of creating a soundscape to give a sense of place (tweeting birds in a meadow), or to make actions seem more realistic (hearing glass break when you see it break), or to give the player feedback about their progress in the game (a musical glissando when you pick up a treasure). But there is an often overlooked aspect to sound that has a direct bearing on interface: the human mind easily maps sound to touch. This is important, since when we manipulate things in the real world, touch is a central component of feedback we get about manipulation. In a virtual interface, we get little, if any, information through our touch sense. But you can simulate touch by playing appropriate sounds. First, you have to think about what you would like your interface to feel like if it were real, and then you have to decide what sounds will best create that feeling. If you do this successfully, people will marvel at what a pleasure your interface is to use, but they will have difficulty expressing exactly why. I have high hopes that future interfaces will find ways to more successfully involve tactile feedback, but until they do, sound is your best bet.

Interface Tip #6: Balance Options and Simplicity with Layers

When designing an interface, you will be confronted by two conflicting desires: the desire to give the player as many options as possible and the desire to make your interface as simple as possible. As with so many things in game design, the key to success is striking a balance. And one good way to achieve this balance is by creating layers of interface through modes and submodes. If you have done a good job of prioritizing your interface, you will have a good head start toward figuring out how to do this. A typical videogame example of this is hiding inventory and configuration menus under an infrequently used button, such as "start."

FIGURE
15.12

The *ToyTopia* control panel. A "down" message has just been sent to Winnie the Pooh. (Courtesy of Disney Enterprises, Inc.)

Interface Tip #7: Use Metaphors

A great shortcut to giving a player understanding of how your interface works is by making it resemble something the player has seen before. For example, in designing the game *ToyTopia*, my team had a very unusual constraint. In this game, the player issues keyboard commands (go up, go right, etc.) to a small team of windup toys. Since it was a synchronous multiplayer game, the plan was to keep things in sync by introducing a delay between when a player issued a command and when a toy would receive it. This way, we could keep games in sync on different players' machines because the local artificial delay would be the same length as the unavoidable network delay of a signal traveling from one computer to another. Unfortunately (and not surprisingly), players found this to be confusing—they are used to a button push taking action immediately—not taking a half second before something happens. The team was frustrated to the point of considering abandoning the whole scheme, but then someone had the idea that if we showed a visible radio signal traveling from the virtual button to the toy and accompanied it by a "radio transmission" sound effect, it might help players understand the mechanism better. And it worked! With the new system, the radio transmission metaphor clearly explained the delay in action and also gave the players some immediate feedback about what was happening. And under Lens #11, *Unification*, this change helped reinforce the theme, which was about radio-controlled toys.

Interface Tip #8: If It Looks Different, It Should Act Different

Game developers frequently fall into the trap of going against this tip in the name of visual variety. For example, they might be making a game where you fight flying saucers. To add some visual spice, someone gets the idea to have the saucers be

different colors: some are red, some purple, and some green. Players who see these are surely going to think that they are functionally different—that they perhaps go different speeds or have different point values. If they are not and they just have different paint jobs, players are sure to be disappointed and confused.

Similarly, designers often make the opposite mistake, creating two things that look the same but behave differently. For example, you might create an "X" button that when pressed, closes part of the interface. Later, in need of a button that allows players to delete items from the game, "X" might seem like a logical choice to represent delete. But if the same "X" sometimes means "delete" and sometimes means "close window," it has a good chance of confusing and frustrating players.

Interface Tip #9: Test, Test, Test!

No one gets an interface right the first time. New games require new interfaces, and you cannot take it for granted that your new interface is going to be clear, power giving, and fun unless you have people try it out. Test it as early as possible and as often as possible. Build prototypes of your interface well before you have a complete playable game. Make paper and cardboard prototypes of any button or menu systems that you have, and get people to act out playing the game and using the interface so that you can see where they are having trouble. Most important, by working with players this way, like an anthropologist, you will start to get better ideas about their intentions from moment to moment, which will inform all of your interface decisions.

Interface Tip #10: Break the Rules to Help Your Player

Since many games are variations on existing themes, there is a lot of copying of interface designs from game to game. So much so that certain rules of thumb tend to show up for each genre of game. These can be useful, but it is easy to follow them slavishly without thinking about whether they are really a good idea for the players of your game. One example involves PC games using a mouse. The left mouse button is considered the main button, and some games choose to use the right mouse button for other functionality. So, a rule of thumb is that the right mouse button should generally not do anything, unless you are in a special mode where it has a purpose. However, this rule is often taken too far—and in simple games, such as children's games, where the right mouse button isn't used at all, most designers tend to leave it completely disabled, so that all gameplay happens through the left mouse button. But when children use a mouse, they frequently click the wrong mouse button because their hands are small. Smart designers break this rule of thumb and make the left and right mouse buttons both map to the same action, so that either button can be pressed successfully. Really, why wouldn't you do this for every game that only needs one mouse button?

The game interface is indeed the gateway to the experience. Let us pass now through that gateway and look more closely at the experience itself.

Other Reading to Consider

***The Design of Everyday Things* by Donald Norman**. This straightforward, down-to-earth book is full of thoughtful examples of good and bad design of real-world objects and systems. Its wisdom translates surprisingly well to the realm of game design.

***Game Feel* by Steve Swink**. This unique book focuses on game interface design at the millisecond level, carefully dissecting what it is that makes games feel great. This is required reading.

***The Visual Display of Quantitative Information* by Edward Tufte**. Considered the bible (or at least the Old Testament) of graphical interface design, this and Tufte's other three books can provide deep insights even when you just leaf through them.