7 Principles of Interface Design

In This Chapter

- Five Golden Rules
- Five Cautions
- Learning Curve
- Subjective Metrics of the Action Space
- Concept Models
- Immersive Menus
- Tutorials
- Conclusion

The creation of the game interface is a crucial step in the design process. A game interface defines all the ways in which the player can interact with the game and is arguably the single most important element of a design. A weak, disorganized or overly complex interface is a barrier to enjoyment for all players (especially the Casual gamers), and therefore every interface should strive to be as simple as is feasible to express the required game actions.

There are three basic components of the interface design for any game:

Front End: This is the term applied to all menus and screens that occur outside of gameplay. In essence, the front end takes the player from the title screen of the game to the point that gameplay begins, but many games have front end screens that are, in effect, part of the game itself (especially those in which the

game levels are not hermetic, and elements cross over from one section of gameplay to the next).

In-Game Menus: These are a set of menus and screens accessed in-game, often from a pause menu, but sometimes as part of the game space. The same principles that apply to front end design can, in general, be applied to the in-game menus, and for some PC games the distinction between front end and in-game menus can be minimal. However, the in-game menus form part of the game mechanisms (and therefore can present issues of suspension of disbelief), whereas the front end is usually distinctly separate from the game world.

Control Mechanism(s): The way in which the player controls their avatar or other game entities is dictated by a control mechanism. Many games have just one control mechanism, but it is not uncommon (especially in sports games) for a game to have multiple control mechanisms at use. /

In this chapter, we look at the general principles of interface design, of menu design (front end and in-game), and of control mechanism design. Having laid the groundwork, we discuss how a designer can take into account the needs of different demographics in interface design. Because the game interface effectively connects the player to the game, interface design is of great importance in the designer's goal of creating games that provide enjoyment to specific audiences.

You should consider many issues when working on an interface design, although three in particular are worth considering in detail:

Simplicity: The principle of *simplicity* is that an interface should be as simple as is possible to support the required degree of actions. The more tightly designed a game is, the easier this goal is to achieve, and the more extensive the design, the harder this goal becomes.

Expressibility: The principle of *expressibility* is concerned with the degree of options and choices of action that the player has in the game world. The more extensive a game design is, the greater expressibility is required in the interface. Therefore, an essential tension between simplicity and expressibility exists in all interfaces.

Learning Curve: The *learning curve* is a conceptual measure of how hard it is to learn to control or use a game. The classic formulation is to see learning curve as a chart with time along the x axis, and players' level of mastery along the y axis.

Throughout this chapter, the terms *button* and *key* can be considered synonymous: a button refers to an interface switch on a console controller or joystick,

while a key refers to a keyboard switch, but functionally these are simply binary switches that are used in the interface.

/ FIVE GOLDEN RULES

The following are a set of basic suggestions for constructing interfaces. Although we have characterized them as "Golden Rules," remember that these are just suggestions—they should not be taken as a dogmatic attempt to define eternal rules.

Rule 1: Be Consistent

Although it sounds self-evident, all too many games do not ensure that their controls maintain their functionality across all contexts. If functionality remains consistent, the player need learn the controls only once. If the functionality is different in many different contexts, the result is either an excessive learning curve or player irritation.

Most importantly, ensure that all menus are operated using essentially the same controls. Choose how the players accept a menu option and how they go back up a menu level, and stick to this convention rigidly. The player must be confident that a certain button will do what they expect it to do.

One powerful method for consistency is a concept model for the interface device—we will discuss this later.

Rule 2: Use the Simplest Interface Feasible for the Gameplay

Always strive for the maximum number of different actions in the smallest set of controls. Indeed, although it is tempting to make use of every button on a controller (or every key on a keyboard!), the core controls of a game should strive to use as *few* buttons as possible.

Imagine you are teaching someone to play the game verbally: you should be able to tell them the basics to get started in two or three sentences ("move with this, use this button to do this, and the other button to do that"). Other controls should be picked up while playing (or be purely cosmetic).

In striving for the balance point between simplicity and expressibility, simplicity should be given the edge—it's almost always better for an interface to be easy to learn rather than full of bells and whistles. Even a game developed for a Hardcore cluster (say, a game exhibiting complex Type 2 Manager play, targeting primarily H2 players) is competing against similar games of the same type; the player always wants to engage with the play elements of the game rather than interface, no matter how game literate they are.

Along the same lines, don't add actions that you don't need—and *never* add an action just because other games of the same type do. Many FPS games use a jump button, despite jumping being peripheral or even distracting to the core activity of many of these games. If your game doesn't need it, you should generally leave it out. However, it might be worth considering whether a common feature meets a particularly player need before cutting it—in the case of jumping in FPS games, players might feel artificially constrained if their avatar cannot reach a platform that could realistically be jumped onto.

Metrics for measuring the degree of simplicity of an interface are discussed later in this chapter.

Rule 3: Draw from the Familiar

Don't reinvent the wheel. If an interface style already in use that the audience knows about fits the situation, use that as a starting point. For example, the standard Windows-style WIMP (Windows-Icons-Mice-Pull-down menus) environment is fine for PC sim games. If you replace it with something else, it had better be easy to learn and offer significant advantages. The same rule applies to icon design. You can use many internationally recognizable symbols to improve the player's immediate comprehension of your interface.

Of course, if you are targeting the mass market, you have to be careful because what the Hardcore consider to be a widely known interface style might be entirely unfamiliar to any given Casual player.

One last caveat: as with so many issues of competence, the development team cannot assume that their audience is as skilled with a particular control scheme as they are. Issues of dimensionality of control, discussed later, cannot be ignored, no matter how familiar a control mechanism seems to the development team.

Rule 4: One Button, One Function

Each button or key should have a single defined function; the player should know when they hit a particular button roughly what it should do. The range of expressible actions can be expanded by having context-sensitive functions. A context-sensitive 'Action' button, whose functionality changes according to what the player's avatar is next to, means that only one action results in any given situation. (Ideally, an icon indicating the nature of that action should be displayed on screen so that the player does not need to guess its meaning.)

When this rule is not followed, controls are considered to be *overloaded*. In *Jet Set Radio* (Smilebit, 2000), the interface is beautifully designed except for the overloading of the left trigger, which is used for both camera control and spraying graffiti. This means that you cannot move the camera when you are close to a graffiti tag, which can frustrate many players.

These issues will be examined in more depth in relation to concept models later in the chapter.

Rule 5: Structure the Learning Curve

Casual players can get swamped if you attempt to teach them all the controls from the start, and therefore it is advisable to stagger the learning curve. Introduce the player gradually to both in-game functionality and interface controls, ideally within the main gameplay, but if that's not possible, make sure the player is encouraged to play the tutorial before they start play.

We discuss both learning curve and tutorials later in this chapter

FIVE CAUTIONS

To complement the five "Golden Rules," here are five cautions that represent game design "red flags" to be checked for.

Caution 1: Shortcuts Are for Advanced Users Only

In PC games, avoid requiring the keyboard for the main interface (with the possible exception of the cursor keys and the space bar). You should generally provide some way (no matter how contrived) to achieve an action from the mouse alone. That doesn't mean you shouldn't include keyboard shortcuts, because the advanced user will certainly want them, but few mass market players want to memorize a list of keys before they can play.

On consoles, consider providing advanced elements of the control mechanism that allow the player to achieve certain actions more quickly, such as *GoldenEye 007*'s (Rare, 1997) ability to trigger mines by hitting the A and B buttons simultaneously. Hardcore players find themselves wanting to do certain things more easily after they have played a lot and can be frustrated by the *absence* of shortcuts. (In addition, the discovery of such shortcuts deep into the play window of a game gives its audience a tremendous sense of involvement and expertise.)

Also bear in mind the fact that players have to remember shortcut controls (generally a problem that applies only to PC games). Keyboard shortcuts suffer often because the only sensible key for an action is the letter it begins with—and certain keys go quickly. A, S, D, and W are commonly used for movement keys, for example, although Casual players generally expect the cursor keys to perform this function.

Don't settle for a contrived solution, such as using the second letters, because this won't help the player at all (especially when the game is being translated into other languages). Try arranging them in sensible spatial clusters on the keyboard,

or renaming the game action so that it can begin with another letter (although again, this might present issues when the game is localized).

Ideally, allow the player to define or redefine their own shortcuts for all the main game actions, although in console games this can be tricky for various reasons, not least of which is the more rigorous QA protocols that are applied. For console games, a set of predefined control options is usually the best solution.

Caution 2: Icons for Speed, Text for Clarity

Icons are great for immediate recognition, provided the player knows what they mean. Casual players don't generally have the patience of the Hardcore, so it is wise to ensure that you provide a text description for all your icons (either as a tool tip in a PC game, or in a help line somewhere on screen, or if all else fails, as a page in the manual).

The front end for *SSX* (EA, 2000) is a good example of using both text and icons to produce a pleasing interface that is simple to use. Each menu consists of a horizontal row of icons, which, where possible, clearly relate to what they represent. Beneath this row is a pictorial representation or other display pertaining to the option currently selected—for example, when the row of icons is used for character selection, the attributes of the currently selected character appear in the center of the screen. Similarly, when selecting courses, a picture of the course is shown. Finally, at the bottom right of the central display area, the meaning of the currently selected icon is shown in text, so that the player never has to guess what the current icon means.

When icons are used in the game world, the player can generally be left to divine their meaning on their own, provided the risk of letting the player experiment is minimal (that is, no cost accrues to the player for experimenting). However, when the icons require controls to activate, some guidance is required. In *Grand Theft Auto* series games (DMA Designs/Rockstar North, 1997 onwards), for example, this occurs when the player is carrying a weapon, but standing over an icon of the same weapon type (for example, they are carrying a chainsaw and standing over a pool cue, which are both types of melee weapons in the game). In this situation, a non-interactive alert box in the top left of the screen tells the player which control allows them to switch weapons.

Similarly, if an icon is activated only when a control is pressed, the control can be displayed over the icon when the player is close enough, or the game can cycle between showing the icon and the control as happens in *Crimson Skies: High Road to Revenge* (FASA Studio, 2003).

Caution 3: Allow Skipping of Non-Interactive Sequences

You might want the player to see your expensively rendered cut scene, but they might not care—or they might have seen it a hundred times before, especially if its between a save point and a tough boss. Provide a method to skip cut scenes, but don't use a control the player might hit by accident.

Game designers encounter some issues with this, however, because some cut scenes contain essential story or game information that the player must see. Acceptable compromises include a menu option "movie viewer" that allows you to see any cut scene already seen, and only allowing the player to skip cut scenes they have already seen (requiring the game to save data on which scenes have been seen, the overheads for which are largely trivial). An example of both can be found in *Eternal Darkness: Sanity's Requiem* (Silicon Knights, 2002), which has three different versions of its framing story so that on repeat play the player sees a mix of new and old cut scene material. The game permits a cut scene to be skipped only if the player has never seen it before and also includes a movie viewer option to show all of the cut scenes already seen.

It is also important that skipping a cut scene doesn't happen by accident. If one of your core game controls skips a cut scene, players might cancel the cut scene just as it occurs! On consoles, using Start/Pause to skip a cut scene is becoming an accepted convention; on PC, ESC is the obvious candidate for skipping, situated as it is at a fair distance from the rest of the keys.

Caution 4: Provide Options and Save Options

Options allow the player to tailor the interface to their own needs. Despite the name, they are not optional to the game design and are a vital part of the interface, especially when designing for a Hardcore audience. Try to allow the player to customize everything that doesn't affect the core game play—you can never be certain what might annoy a player in your interface design.

If the controls are customizable, remember to transfer secondary actions. *Black & White* (Lionhead, 2001), for example, allows you to redefine the control for the move operation (normally on left mouse button), but if this is done, the player loses the double left-click function that allows you to jump to a particular location directly.

Also, make sure you save all the options. The player doesn't want to reconfigure every time they start playing the game. Ideally, maintain a player profile for anyone who might be playing the game, and save their options separately.

Caution 5: Document It!

Even though most players don't read the manual, they will turn to it if they have a problem or want to find out if such-and-such a thing is possible. Good documentation saves your players much frustration. A Hardcore player might be willing to go online to get the answer to an interface question from a FAQ, but everyone else is more likely to turn to the manual.

LEARNING CURVE

How hard a game is to learn determines to a considerable degree who ends up playing it. Complex games appeal only to Hardcore players, except in rare cases (flight simulators, for example, have a niche audience who tolerate complexity for the sake of realism). To appeal to the mass market, simplicity is king. The learning curve is a way of thinking about the impact of interface complexity on the player.

To a certain degree, the learning curve is a product of both simplicity (simpler interfaces have shallower learning curves) and expressibility (the more expressive the game, the steeper the learning curve), but one other key factor exists: how familiar the techniques used are. Using techniques that players already have some familiarity with always eases the learning curve—but beware. While you can count on a Hardcore gamer to know all the current interface styles (including complex forms such as twin stick/mouse-keyboard first-person shooter interface), you cannot count upon the mass market to have the same skills.

Examples, such as those depicted in Figure 7.1, can depict some basic points.

The three learning curves shown in Figure 7.1 represent three different types of situations with regards to learning an interface. The solid line represents the "optimum" learning curve—players achieve a steady level of mastery over the controls as the game progresses, with the interface elements being introduced gradually. The dashed line represents a more typical situation: the player has a lot to learn at the beginning, resulting in a very steep learning curve—the player is effectively thrown in at the deep end. Despite this being a common state of affairs, it is not a good sign for a commercial game. Games with steep learning curves like these rarely penetrate beyond the Hardcore.

The dotted line represents the effect of prior player knowledge on learning a game, specifically when the interface is in a form the player is already familiar with. Initially, the player already knows much of the control scheme of the game, so learning doesn't really take place fully until later, and less overall learning must take place. Most of the learning is the player adapting to the specifics of the game itself.

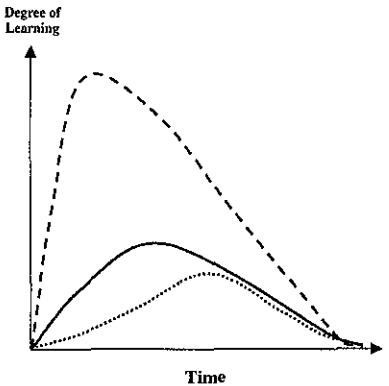

FIGURE 7.1 Three different learning curves for hypothetical interfaces. The solid line is the archetypal "optimal" learning curve, the dashed line shows the learning curve for a complex interface, and the dotted line represents the effect of prior knowledge in attenuating the learning process.

Note that knowledge of different interfaces is not evenly distributed, and therefore profiles like the dotted line *cannot* be planned for except in rare cases. One such exception is a game for PC under a Windows operating system. Because the player already knows the basis of operating a WIMP interface, you can use this in your game with some confidence of prior player knowledge.

Figure 7.1 showed learning curve with respect to achieving mastery, but you need to understand another aspect of learning curve: the point in the game that mastery is achieved. Figure 7.2 demonstrates this point most clearly.

The solid line in Figure 7.2 shows a good learning curve for a game—the player steadily learns controls in the early part of the game, but before the halfway point the emphasis changes from learning to enjoyment of the game. The dashed line shows a somewhat frustrating style of learning curve: the player is taught a set of tough controls, but just as they get to grips with those, they are expected to learn something knew. Learning progresses throughout the game, so that the player has managed to master everything only by the very end. Although this might potentially work with a Hardcore audience, it is an awkward approach. Learning what to do is not what most players want to be doing: they want to be actually doing it.

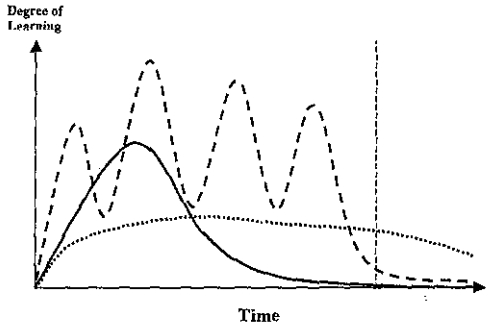

FIGURE 7.2 Learning curves with respect to mastering an interface. The solid line shows a well-paced learning curve, the dashed line a game that is constantly providing the player more and more things to master, and the dotted line a game that is easy to learn but with hidden complexity.

The dotted curve represents a "quick to learn, lifetime to master" interface. The player gets to grips with the controls quickly and easily, but still has much to learn. The player continues to uncover features of the interface organically as the game progresses. This was the approach taken with *Ghost Master* (Sick Puppies, 2003)—letting the player learn and master all the controls they needed relatively quickly, but imbedding complexity in the nature of the actions that the ghosts could carry out. Most players had not learned even a fraction of the capabilities of the ghosts' powers by the end of the game. In principle, this would have stood the game in good stead for future add-on packs, leaving much left to play with.

For *Ghost Master*, this was a mixed blessing. That the controls were relatively easy to master was a success, but the hidden extent of the ghosts powers proved problematic, because it invited criticism of the game based on ignorance (because even at the game's completion, the players still had much they had not come to understand). Typically, the idea that powers affected the world as a system, rather than as individual actions, was difficult for some players to appreciate (especially where game assumptions preceded experimentation). Perhaps more effort could have been expended to allow the player direct access to some of the knowledge locked up in the "organic world" that the game used, although it is equally likely that Hard-

core players who thrive on micro-management would have struggled to adapt to the more holistic system used in *Ghost Master* without specific assistance.

The bottom line is that numerous learning curves can be made to work, but in general, learning should be seen as something to keep manageable. Sharp peaks on a learning curve diagram show periods where many players are struggling to achieve anything; this sort of frustration is to be avoided wherever possible.

SUBJECTIVE METRICS OF THE ACTION SPACE

The *action space* of a game is a projection of all the possible activities that can be carried out within the game world. The more different actions possible (and the more different operational modes the game uses), the larger the action space. One way of representing an action space for control mechanisms is to imagine all the controls the player has as the leftmost column of a table, and all the different contexts within which the controls might be used as the topmost row of the table.

For menus, rather than controls, the action space can be considered as a flowchart (or similar representation) that shows all the possible states that the menu system can be in. This applies equally to the front end and any other menu screens at use in the game.

The total action space is therefore the sum of all control mechanism action spaces and all the menu action spaces.

As a general rule, we want to keep the action space as small as possible, to keep the learning curve of the game as low as possible, but we want a sufficient degree of expressibility within this space that the player can do anything that is relevant to the game.

To assess interface design, it is useful to apply metrics that "measure" certain properties of the action space defined for the game. For control mechanisms, a good metric is dimensionality of control, while for menus, action depth is a useful metric.

Dimensionality of Control

The concept of *dimensionality of control* is a measure of the degree of complexity inherent in a control mechanism. It is a numerical figure representing the number of dimensions inherent to the interface mechanic presented to the player. Although it is derived in an essentially subjective fashion, any formulation can be used as the measure, because games are measured relative to one another.

The main component of the dimensionality of control is the number of degrees of freedom of movement inherent to the control mechanism. For example:

Tetris (Atari, 1988): This has essentially one degree of freedom of movement—the blocks move only left and right.

The Legend of Zelda: A Link to the Past (Nintendo, 1987): This game is presented in top-down 2D projection and has essentially two degrees of freedom of movement—the player can move left, right, up, and down.

GoldenEye 007 (Rare, 1997): In its default configuration, this game has two degrees of freedom of movement—the stick changes the direction faced and moves in that direction. (Strafing and tilting the view up and down are also provided as an additional component, but in this control mode are entirely optional.) In its other configurations it has four degrees of freedom of movement—the player can adjust their view in two dimensions, move in and out of that plane, and strafe left and right.

Half-Life (Valve, 1998): This game and other PC first-person shooters are based around four degrees of freedom—two degrees of view and two degrees of movement. (Although it might seem strange to have more than three degrees, remember that these control mechanisms simulate both the facing of the avatar and independent planar movement.)

In addition to the degrees of freedom to move, dimensionality of control must take into account any other dimensions to control:

- Different ways of moving along a line of some kind (such as strafing, throttle-based velocity, or time control) constitute an additional full dimension of control.
- Any *enfolded* control aspects, such as discrete rotation, accelerator-velocity, or jumping, can be considered an additional half a dimension of control.
- Atomic actions that are non-spatial can be considered to add dimensions equal to one quarter of the number of actions.

Deciding what constitutes an "enfolded" aspect is in general a judgment call; however, a simple test is as follows: if you plot a chart of this property, does it fit on one axis and consist of a limited set of values? So, for example, discrete rotation is an angle that is another dimension, but is enfolded because it is non-spatial and limited to set values. An argument can be made that velocity control should constitute a full dimension of control; however, the intuitive familiarity of an accelerator system (cars, etc.) and the fact that the range of velocities is fixed and limited in most games make this reasonable as a half dimension. However, a throttle system—where the speed and the setting of the control are different—should be considered a full dimension because of the added complexity.

Jumping, similarly, can be shown on a single axis (varying height), but the profile is fixed and therefore can be considered a half dimension. Note, however, that jumping generates a half dimension from one control, while most other enfolded elements use two controls, making this a special case.

Continuing with our previous examples:

Tetris: The player can rotate blocks left and right, so we consider this an addition half a dimension (discrete rotation).

The Legend of Zelda: A Link to the Past: In this game the player has a number of different actions in the game world, but all from two buttons. These two atomic actions add another half dimension to the controls.

GoldenEye 007: The controls gain an additional half dimension from selecting weapons, and another half from atomic actions. (In the basic configuration, you also have an additional dimension from strafing and from vertical adjustment of view, although players using this configuration tend not to use these components of the interface.)

Half-Life: This game adds to its basic controls various added functions including swimming with unique separate controls (+1), jumping (+0.5), ducking (+0.5), weapon changing (+0.5), four additional essential atomic actions (+1), and a large number of optional additional controls that can be ignored.

This makes these four games, by this measure of dimensionality of control:

Tetris: 1.5 dimensional.

The Legend of Zelda: A Link to the Past: 2.5 dimensional.

GoldenEye 007: 3 dimensional in the basic configuration (two degrees of movement, plus weapon select, shoot, and action), and 5 dimensional in the advanced configuration.

Half-Life: 7.5 dimensional (in the PC configuration).

The recent explosion of controls included in games has meant that the trend in dimensionality in control has increased with each new controller—as game designers insist on using all the controls available to them. The NES controller could support only 2.5 dimensional play; the PlayStation controller offers up to 4 dimensions from its two analog sticks, 1–2 dimensions from the D-pad, 1–2 dimensions from the face buttons, another 1–2 dimensions from the shoulder buttons, and a final 0.5 to 1 dimension from the stick buttons for a staggering maximum of 11 dimensions of control (although typically a game using all buttons will be only 7.5 dimensional, because very few actions like jumping can be considered to add 0.5 dimensions from a single control).

A summary of this particular system of evaluating dimensionality of control can be found in Table 7.1.

TABLE 7.1 Dimensionality of Control

Element	Type	Dimensionality
Control	1 dimensional (left-right)	+1
	2 dimensional (left-right, up-down)	+2
	3 dimensional (left-right, up-down in-out)	+3
	3 dimensional (2D view that extends off the screen, for example, space shooter controls)	+3
	4 dimensional (2D view, 2D move, for example, first-person shooter twin sticks/mouse keyboard)	+4
Additional Movement	Strafing mechanism (2 buttons)	+1
	Throttle-type acceleration (2 buttons)	+1
	Height control (2 buttons)	+1
	Leaning (2 buttons)	+1
Enfolded Dimensions	Jumping	+0.5
	Accelerator-type acceleration (2 buttons)	+0.5
	Discrete rotation (1 or 2 buttons)	+0.5
Atomic Actions	Any other action (1 button)	+0.5 per 2

Of course, not all actions in the game world are essential. To count control of the radio stations or beeping the horn as part of the dimensionality of control in *GTA* is unfair, because they are aesthetic elements, not gameplay elements. This suggests it might be worth thinking in terms of core dimensionality of control and full dimensionality of control for some games. *GTA*'s core controls (for the driving element) are 1.5 dimensional (steer left and right, plus half a dimension for velocity), with another 5 dimensions in optional components (2 dimensions of optional camera control, 0.5 dimensions from look left or right, and 1.5 dimensions from atomic actions).

Different demographics can handle different degrees of dimensionality of control; as a rough guide:

Casual players: In general these players can cope with between 3 and 5 dimensions of control.

C1 players: Players fitting this cluster can be expected to deal with 7 dimensions of control, or more if they are built upon an already learned mechanism (for example, FPS control has become so ubiquitous, they can handle 9 or 10 dimensions if built on this interface configuration). Many, but by no means all, C2 players also fit this profile.

Hardcore players: These players can (in general) handle 9 dimensions of control without any difficulty and can master more if sufficiently motivated, with potentially no upper limit. H3 and H4 players prefer fewer dimensions, however, and 5–7 is recommended for these clusters.

Games targeting the mass market have significant benefit from introducing controls gradually, because the player masters a low dimensionality of control interface mechanism before having it expanded. Although this might annoy H1 (and possibly H2) players (who apparently like to be put in at the deep end), it is worth bearing in mind for other target clusters.

Thinking in terms of dimensionality of control, it becomes clear that the player shouldn't be the only force in charge of the camera if you want to hit a mass market audience. If the game controls the camera, but the player can adjust it, you are just adding an additional 2 dimensions of optional control, but if the player must control the camera, you are adding an additional 2 dimensions of required control. This probably means the game can no longer appeal to C3 or C4 players.

The reason so many companies produce third-person games using a twin stick control mechanism with no automatic camera control can be attributed to the fact that the majority of game developers routinely play PC first-person shooters (which usually have at least 5.5 dimensions of control, and generally much more) and therefore have an exaggerated sense of what the normal player can handle. This trend might ultimately be doomed for the mass market (but might survive for niche markets, such as pure Hardcore games) because any such game is going to have a minimum of 5 dimensions—and that assumes that only four face buttons are used for controls.

Credit must be given to Nintendo for keeping dimensionality of control low in their games, which are always targeting the maximum possible market (especially in terms of age range). Although it is doubtful they couch these design decisions in terms of dimensionality of control, it is clear they have recognized the limited degree of control the mass market can master, relative to the Hardcore. It is notable

that Nintendo's only FPS series, *Metroid Prime* (Nintendo, 2002 onwards), does not use a twin stick control mechanism, although the full control mechanism for this game does offer the same high degree of dimensionality control of other FPS once mastered. A parallel can be drawn with *GoldenEye* (which was also on a Nintendo platform) with its low-dimension default control mechanism that expanded into full FPS style controls.

Action Depth

What dimensionality of control is for a control scheme, *action depth* is for a menu system. Simply put, the depth of a particular action can be defined as the number of subactions required to execute that action. For example, opening a pause menu by pressing Start would be at depth 1, quitting a game from the pause menu is at depth of 2 (open the pause menu, then select quit).

In general, every action should be at the lowest depth achievable, and all common actions should be within a depth of 3 or less. Minimizing action depth makes the interface easy to learn and fast to navigate and minimizes player frustration.

The principles of action depth should be applied to the front end and to the in-game menus. For example, you do not want the action depth to continue playing a game to be more than 1, except in rare cases. Similarly, it is desirable to allow the player to start playing a game from the title screen with very little menu navigation, and most games default to having the new game option as the default. However, it should be noted that if an extant save game exists, it is often better for "continue" to be presented as the default option on the title screen; the player needs to start a new game only once in most cases, but continues an existing game frequently. New game can be offered as the default option only when no extant save game exists.

Making the player go through all of the previous menus before returning to play should be avoided, and wherever possible the most common options should be immediately available at the end of a section of gameplay. *Wave Race 64* (Nintendo, 1996) allows the player to change the course or their chosen watercraft from a menu at the end of each race, minimizing the action depth of the most common options by offering them on a menu displayed after each race.

Mandatory replays should also be considered carefully, because you are committing the player to an extra action just to continue playing. Perhaps the optimal approach for replays in terms of action depth is demonstrated by *SSX*, which plays the replay automatically behind the results screen and offers the player a menu on this screen from which they can choose to restart the level, view the replay (without the results overlay), see the records, or quit.

Action depth can also be applied to control mechanisms. Consider *Turok 2*'s (Iguana, 1998) radial weapon select, which is always around depth 1, versus its con-

temporary rival *GoldenEye*'s linear sequence of weapons, where depth increases as you acquire more weapons. The latter style has become the norm in console FPS games because of a perceived need to select a weapon and move at the same time—a product of the extreme Hardcore attitude of many FPS players. The *Turok* wheel is a superior choice for a more mass market audience: holding a button brings up a radial menu; the player then selects their option, and releases the menu button. Although only eight different options are allowable in this system, it offers much shallower action depth, and it is notable that the system was revived for *Ratchet & Clank* (Insomniac, 2002), which presumably was hoping for a mass market audience.

Any number of systems can be defined for measuring action depth, and your choices of how to do this can affect how you think about the interface design. If you consider selecting to be half an action and pressing a button or key to be another half action, it is apparent that using a single key to close a window as a shortcut generally halves the depth of exiting the menus when you have finished. Try to use a system that reflects the factors most important to the interface in question. If you add depth for loading times, for example, you are more likely to spot the need to include a restart option on the game over screen, so the player doesn't have to reload to play the level again.

The point of thinking in terms of action depth is to spot where the player might end up spending too much time wrestling with the interface and then simplify.

CONCEPT MODELS

A *concept model* is a framework that can be taught to the player that assigns a single role to each control. For example, a very simple concept model for a simple game might be as follows:

- Stick → Move
- A Button → Action
- B Button → Attack

Now it might be that the A button carries out different actions in different contexts, but if the player thinks of it as the "Action" button, they learn simply that pressing this button will carry out most actions. Similarly, "Attack" might mean different things in different situations, but by learning it as "Attack" the player always knows that the Attack button performs an offensive move.

In general, a concept model emerges from looking at all the actions that need to occur in the game world and then grouping them according to functionality.

Designing Concept Models for Multiple Platforms

Increasingly, games are appearing cross-platform (or multi-format)—that is, they appear on multiple different consoles and possibly on PC as well. In this situation, it is vital that the concept model can be applied to all the different platforms the game will appear on.

As an example of what happens when this point is overlooked, consider *Turok: Evolution* (Acclaim, 2002). The concept model for this game was worked out for the PS2, and the functionality then converted for the other formats (GameCube and Xbox).

The clearest sign that multiple formats were not considered occurs with the GameCube version. Next Weapon and Previous Weapon are control concepts that belong on related buttons. On the PS2, the four face buttons are part of a set of four, and any two adjacent buttons can be paired for these functions. But on the GameCube, the A and B keys are paired, and the X and Y buttons (which are grey and kidney shaped) are conceptually a different element of the controller. However, *Turok: Evolution* on the GameCube assigns Previous Weapon to B and Next Weapon to Y. This is counterintuitive; it would have been better to assign these two functions to X and Y because they are part of the controller that imply related (and lower priority) actions.

The same problem occurs in the flight controls in *Turok: Evolution*. Bank Left is assigned to B, and Bank Right is assigned to X, with weapons on the shoulder buttons. Speed Up and Slow Down are assigned to A and Y. It makes considerably more sense that Bank Left and Bank Right be assigned to shoulder buttons so that the Left shoulder executes Bank Left and the Right shoulder button executes Bank Right. Speed Up and Slow Down can then be grouped on the grey X and Y buttons, with A and B firing weapons. Curiously, the game options include an alternate configuration that does take into account these issues; why it is not the game's default is a mystery.

One could argue that the motivation for the contrived flight control configuration was to keep the concept model for A (fire) the same. But as the only control that maps between configurations, this seems misguided. It is likely that the game design favored the PS2 version because this was the platform that greatest sales would be expected for (on account of a larger installed base), but given the role of the Hardcore as evangelists and the tendency for Hardcore players to buy games on the most technically advanced platform they own, this was probably a mistake.

Whenever a concept model needs to map across multiple formats, consider the nature of the control devices on all formats and identify the common elements and features that can be used. The control schema must be compatible with the control devices with the fewest number of controls if all the target platforms are going to have adequate control methods.

IMMERSIVE MENUS

Whereas a front end menu is almost always a simple list of options, menus appearing inside the game world need not resemble the classic list of text. Indeed, a growing trend is towards imbedding menu functionality within the game world in a form that seamlessly integrates with that world. We can consider such an approach an *immersive menu*.

An early example can be found in *The Legend of Zelda: The Ocarina of Time* (Nintendo, 1998) in which the shops displayed all the items for sale on shelves that can be seen by the player as they move around the world. When talking to a shop keeper, the menu functionality manifests by allowing the player to move around the shelves to select what to buy, as if their avatar were scanning the shelves with their eyes. This is a more immersive approach than the classic overlaid text menu that had predominated before 3D games began to become the norm. Clearly this kind of approach is rooted in mimicry, and as such, it can be assumed that attempting this kind of interface is a decision to incorporate elements that appeal to players who enjoy Type 3 Wanderer and Type 4 Participant play.

Another example of moving away from menu selection and towards immersive menus (or functionally equivalent situations) can be found with the *Tokyo Xtreme Racer* (*TXR*) series (Genki, 1999 onwards). Racing games prior to this generally allowed the player to select a race from a conventional menu front end. *TXR* turns this assumption on its head by letting the player drive around a world (based on the Tokyo highway system) within which many other cars are on the roads. Specifically marked cars represent a race—to trigger it, the player has merely to drive up behind them and flash their headlights on and off. This triggers a brief sequence as the two cars acknowledge each other and pull alongside, and then the race begins. The mechanic elegantly discards the need for a front end menu to select races and increases the degree of immersion experienced by the player.

Similarly, the *GTA* series have used their game worlds as a surrogate menu system. Missions are not selected from a menu, but are represented by spotlights in the world at large that the player drives to and stops within to trigger the mission. Another example within this series of games can be seen with the shop and wardrobe functionality in *Grand Theft Auto: San Andreas* (Rockstar North, 2004). Here, the menu system for dressing up exists both in text and in graphical display: the avatar walks into the wardrobe and then emerges wearing the selected item. Sadly, this system was marred slightly by the long load times associated with trying on clothes, which reduced the elegance of what was otherwise a sound approach to making a menu system more immersive.

We have already noted how these kinds of approaches favor mimicry over agon, and it is interesting to note that *Need for Speed Underground 2* (Black Box,

2004) provides a choice of game modes. One offers the immersive world approach of *GTA* whereby tasks are selected by driving around the world to certain points. The other strips away this layer of mimicry and replaces it with a conventional menu system. Reviewers have been divided on the two approaches, and this likely reflects a split between the agon-motivated Type 1 and 2 styles of play, and the mimicry-motivated Type 3 and 4 styles.

TUTORIALS

Once an interface design has been constructed, some thought must be given as to how the player learns the game. To date, no unified tutorial design philosophy has emerged, and the many different approaches have advantages and disadvantages to each. Much of the decision as to which way to go should depend upon the nature of the game and the nature of the target audience. In general, Casual players are patient when being told rules, while Hardcore players (specifically H1 and H2) greatly prefer to experiment for themselves.

Control Flashcard

The simplest form of tutorial is a *control flashcard*, where the game controls are shown on one screen. Game demos often use a control flashcard at the start of the demo to brief the player of the controls. The flashcard can also be shown on the loading screen, as in *Crimson Skies: High Road to Revenge*.

Incredibly cheap and simple, it is not recommended as a replacement for a tutorial, not least of which because it is often a somewhat indigestible way of delivering information—too much is shown to the player at once and players don't know what is of primary importance and what is an advanced control. Multiple flashcards each showing one element of the controls, as seen in *Beyond Good and Evil* (Ubisoft, 2003), improve this situation somewhat, but might leave players wondering what the other buttons do.

However, it is well worth including a control flashcard as a reminder for the player somewhere in a game, as happens with *Tak and the Power of Juju* (Avalanche Software, 2003) and other games that have a control flashcard available from the pause menu. It allows the player to check the controls without having to look in a paper manual.

Training Movie

This is another simple form of tutorial where the player is shown a short film that briefs them of what to do. In general terms, it is suited only to situations where the

game is very simple to pick up and hence a short briefing suffices, because in other situations the player should be afforded the chance to experiment with the controls they are being taught.

A good example of the training movie can be found in *Kirby Air Ride* (HAL Laboratory, 2003). Each of the game's three modes has a short training movie that is between 60 and 90 seconds long. The most important information is given right at the beginning and the least important at the end, so that the player generally has the gist of the game without having to watch the entire movie. The tutorial is played the first time a mode is selected and the first time a mode is selected after not having been accessed for a considerable length of time and can also be selected by the player manually using the "How to Play" option on the front end menu for that mode. One of the reasons this approach works well with this game is that the control scheme is very simple—just the stick and a single button—and the implementation of the training movie format in this game is practically optimal.

Less effective is the training movie in *Tak and the Power of Juju*, which briefs the player about the controls in the two snowboarding levels. The movie is several minutes long and occurs in the middle of play, thus breaking the player out of the game in a somewhat unwelcome fashion. The movie is necessary only because the snowboarding sections are essentially a completely different game.

Training movies are of most use on games where the availability of storage space is very high, especially console games that often have more data storage on their media than is needed. On portable devices and mobile phones, their applicability is more limited. They are also a great cost-saving move, because a training movie is almost invariably cheaper to construct (in terms of man hours) than a playable tutorial. This is presumably the reason that *Tak* uses a movie to introduce the secondary gameplay of the snowboarding levels, but a detailed playable tutorial for the main gameplay.

Linear Exercises

In this style of tutorial, information is given to the player as a series of exercises, each of which teaches one element of the interface. This has the huge advantage of being comprehensive and straightforward, but might lead to Hardcore players' (or any player who picks it up quickly) becoming frustrated with the slow pace. Players who display a preference for Sensing (which can be assumed to be the entire Casual audience) prefer this style of tutorial.

To allow for some freedom for Hardcore players to proceed at their pace, it is often worth including the capacity for players to skip exercises. This can be doubly beneficial, because it generally implies that earlier exercises can be returned to as well—thus allowing the player some degree of control over their tutelage.

Goal-Oriented Tutorials

This is a variant on linear exercises in which the exercises run in series, each with a goal. Completing each goal is required for progress, so the player must (in principle) have learned what they need if the goals are all completed. The downside of this approach is that the player might fluke the goal in a way the designers never intended and not learn the required mechanic. Plus, the issue of how the use of the control in question is to be conveyed to the player still exists. Nevertheless, this technique does not alienate any particular cluster of players.

Common techniques for teaching the player within a goal-oriented tutorial are context-sensitive commentary and help signs (both of which are discussed later in the chapter). Alert boxes that pop up near the start of each segment are a functionally equivalent, if somewhat clumsy, alternative.

It is also possible for goal-oriented tutorials to be structured in a parallel fashion, for example, if the player is given a list of goals to be achieved. This can work for any audience group provided the number of elements being taught at any given time is kept low, although it is likely to be preferred by Hardcore players (whose game literacy often gives them a head start in learning how to play a game) and especially H1 players who have a desire to advance rapidly and might become impatient if constrained. When multiple goals are provided simultaneously like this, the player should always be provided access to a list of their goals (often from a pause menu).

Help Signs

A common technique in Nintendo games is to have all the help written on signposts in the game world that the player can read or ignore at their leisure. This has the huge advantage that experienced players can ignore the boards, but has several disadvantages. First, if the player doesn't find a particular board, they do not learn a particular element of the interface. Second, the player might pick up most of the interface easily, but still need advanced tuition—this leads to players' having to read every board, even though most of them are telling them things they already know.

The method works relatively well for Casual players, and Hardcore players (especially H1) are likely to simply ignore it and make their own way. However, placement of the signs is absolutely critical. Ideally, if the player *must* learn a particular mechanic, a barrier should be placed in front of the player at a mandatory point with the help signs nearby to provide instruction, much as in a goal-oriented tutorial approach (and, as already mentioned, the two methods readily combine).

Imbedded Manual

An *imbedded manual* is simply a situation where the text of the game manual (or a secondary manual created specifically for imbedding in the game) is provided that

allows the player to read about game functions in the game world. This can be considered in many ways a more detailed version of Help signs and a less immersive solution.

Games targeting Type 2 Manager players (especially the more complicated PC sim or strategy games) benefit greatly from the inclusion of an imbedded manual, and H2 players in particular have the patience to explore a manual at their leisure. H1 players, by contrast, might feel obligated to read all the manual material presented to them, even if this frustrates them, and Casual players generally look at manuals only when they are stuck.

In games targeting a more mass market audience, the imbedded manual can be softened by breaking up the text into smaller segments. The *Resident Evil* series (Capcom, 1996 onwards) uses a system whereby the player is provided short notes that introduce gameplay elements gradually as the game progresses. In this approach, it is similar to the Help sign approach, except that the player generally has access to all the notes they have collected from their pause menu, thus they gradually build up a complete imbedded manual.

Context-Sensitive Commentary

In this approach, the player is told about interface elements only when they first use them. The usual source of the commentary is generally a character in the game world, if only to avoid undermining the game's narrative abstractions. The need to have a character as the voice of such commentary is inevitable if all speech is to be recorded, although avoidable if the game presents it as text (usually only seen in games with large amounts of script, such as RPGs, or games made on much smaller budgets).

This approach can work well—except when the nature of controls to be taught means that many things can be encountered simultaneously. In this situation, it becomes difficult to know what to tell the player, and worse, the player might encounter huge "info dumps" of tutorial instructions all at once. Early versions of the *Ghost Master* tutorial suffered from this problem to a near-terminal degree, but fortunately adequate blind testing smoothed out many of the problems before the final release.

As already noted, this approach works best when combined with the goal-oriented approach; the context-sensitive instructions can be delivered when the player approaches the part of the game world in which that knowledge is needed to overcome the goal.

Checklist with Prompts

This is based around the idea of ticking off an item in a list when the player has learned a particular action. The list in question need not be seen by the player if the

criterion for ticking off an item is sufficiently inclusive. In a sense, it is a version of the parallel goal-oriented approach, in which the "checklist" is expressed in the game world as goals. The advantage of this approach is that the player can be prompted with context-sensitive dialogue whenever they need a particular action. When the player has definitively demonstrated that they can do a particular action, it is "ticked off," and the player is no longer prompted.

The challenge with this approach is ensuring that the checklist condition is a sufficient test that the action was learned. For example, this approach was used in the aforementioned *Ghost Master* tutorial and worked fine with many players. However, in several cases the player had apparently learned how to change the view (thus ticking off the instructions relating to camera control), but had in reality merely pressed the wrong control.

Staggered Complexity

This is a technique used in the *GTA* games, as well as many games developed by Nintendo. In this approach, not all the functionality of the interface is "switched on" at the start of the game, and controls are added as they are needed. This has simplicity on its side, but care must be taken in just how the player is told about the controls: in *GTA*, text is displayed in an alert box in the top-left corner of the screen. If the player is involved in something that requires their full concentration, these instructions can be missed, leaving the player permanently in the dark. (The pause screen menus allow some of this text to be reviewed, but Casual players are unlikely to think to look.)

Care must also be taken that players who replay the game are not annoyed by the actions blocked to them at the start. Controls should be introduced relatively quickly wherever possible (unless the game elements they refer to are not encountered until later and then they can be deferred).

CONCLUSION

The importance of good interface design cannot be overstressed. Any game design can be ruined by a thoughtless, overcomplicated, or inconsistent control scheme, and a badly structured front end can hinder a game's chances at market by alienating curious players before they reach the meat of the product. From a point of view of first impressions, these elements are vital in convincing the player that they are playing a carefully created product that is worth their time.

It is vital that a game interface is designed with that product's target audience in mind. Different players have markedly different needs from a control scheme. For example, Hardcore players (especially H1) actively require character actions to

deliver a sense of competence and might be frustrated when the game offers them insufficient options for controlling their character. Conversely, Casual players require simplicity (delivered via a low dimensionality of control), and those who favor Type 3 and 4 play generally want an interface that favors mimicry.

The degree of complexity in the control interface might determine the extent of a game's possible audience, *regardless of other game elements*. As such, interface design requires significant thought at the earliest possible stage of development.