CHAPTER 12

General Principles of Level Design

If you have ever found yourself admiring the environment of a game or enjoying the way the game's challenges keep you guessing, you are appreciating the work of that game's level designer. The level designer creates not only the space in which the game takes place—its furnishings and backgrounds—but also the player's moment-by-moment experience of the game, and much of its emotional context. Successful level designers draw on fundamental design principles that apply to any kind of game, such as ensuring the player always knows his short-term goals and the consequences of risks, as well as design principles specific to the type of game being designed. Level designers work closely with the game designer to make sure layouts are appropriate for the storyline and to achieve the atmosphere and pacing required to keep players engaged in the game world. Level design will not be a quick and easy process if you do it right. This chapter will identify 11 steps that the level designer takes, from initial handoff to user testing. The final section details problems to avoid in the level design process, including the key directive to never lose sight of your audience.

What Is Level Design?

Chapter 2, "Design Components and Processes," described level design as the process of constructing the experience that will be offered directly to the player, using components provided by the game designer. Note that the terms *game designer* and *level designer* are not interchangeable but refer to separate roles that, on larger development teams, are almost always played by different members of the team. In the rest of this book, the word *you* means *the reader as game designer*, but in this chapter only, *you* indicates *the reader as level designer*.

Level designers create the following essential parts of the player's experience:

- **The space in which the game takes place.** If the game includes a simulated space, as most do, then level design includes creating that space using a 2D or 3D modeling tool. While game designers determine what *kinds* of things will be in the game world, level designers determine precisely *what features* will be in each level of the game world and where these features will be. Level designers take the game designer's general plans for levels and make them specific and concrete.

- **The initial conditions of the level**, including the state of various changeable features (Is the drawbridge initially up or down?), the number of artificial opponents the player faces, the amounts of any resources that the player controls at the beginning of the level, and the location of resources that may be found in the landscape.
- **The set of challenges the player will face within the level.** Many games offer challenges in a linear sequence; if so, level designers determine what that sequence will be, construct a suitable space, and place the challenges within it. In other games, the challenges may be approached in a number of different possible sequences or any order at all; see the later sections "Layouts" and "Progression and Pacing" for further discussion.
- **The termination conditions of the level**, ordinarily characterized in terms of victory and/or loss. In many games, levels can only be won but not lost, and in a few, such as the default mode in *SimCity*, levels can only be lost and never won.
- **The interplay between the gameplay and the game's story,** if any. The writer of the story must work closely with the level designer to interweave gameplay and narrative events.
- **The aesthetics and mood of the level.** Whereas the game designer and art director specify the overall tone of a level and artists create the specific models and textures, level designers take the general specifications and decide how to implement those plans. If the plan says, "Level 13 will be a scary haunted house," the level designers decide what kind of a house and *how* to make it feel scary and haunted.

Level designers normally construct all these parts using tools created specifically for the purpose. Some games, including *Warcraft III* and *Half-Life 2*, actually ship their level design tools along with the game, so players can expand and customize the game world; if you own one of these games, you can practice level design by using those tools.

Level design could easily be the subject of an entire book. However, this chapter concentrates on introducing the general principles and the process of level design.

Key Design Principles

Two types of design principles will help you design a level: *universal level design principles* aimed at designing levels in any kind of game, and *genre-specific level design principles*, which focus on design issues specific to the different genres. This section addresses each of these in turn.

Universal Level Design Principles

Barbarossa: *The [Pirate's] Code is more what you'd call "guidelines" than actual rules.*

—*Pirates of the Caribbean: The Curse of the Black Pearl*

Level designers have for some time tried to define a set of principles to guide the level design process so that new games will avoid the errors of older ones. Considerable debate surrounds this issue, because not everyone agrees on which, if any, principle is truly universal. Examining the important principles constitutes a valuable exercise in any case, so we present a brief list here. Some principles apply as much to game design generally as they do specifically to level design, but because the level designer constructs the play environment and sets the challenges, she will be the one who puts these principles into practice.

- **Make the early levels of a game tutorial levels.** The entire section "Tutorial Levels" is devoted to this extremely important topic later in this chapter.
- **Vary the pacing of the level.** This is also critically important. The "Progression and Pacing" section addresses this later.
- **When the player surmounts a challenge that consumes his resources, provide more resources.** This may seem obvious, but you might be surprised at how many games fail to do it. This, too, is addressed in "Progression and Pacing."
- **Avoid conceptual non sequiturs.** Unless your level is either intentionally surreal or meant to be funny, you shouldn't build elements that make no sense, such as rooms accessible only via ventilation shafts. Even more important, don't put dangers or rewards in places in which no sane person could possibly expect to find them. See the section "Avoid Conceptual Non Sequiturs" later in the chapter.
- **Clearly inform the player of his short-term goals.** At any given time, the player is working to achieve a whole hierarchy of challenges, from the overall victory condition of the game down to the problem occupying his attention (How do I get across this chasm?) at the immediate moment. (Chapter 9, "Gameplay," discusses the hierarchy of challenges at greater length.) While you do not always have to tell the player exactly what he needs to do to win (he may have to discover the long-term goal through exploration or observation), you should never leave him wondering what to do next; the current or next short-term goal should be obvious.
- **Be clear about risks, rewards, and the consequences of decisions.** When facing a challenge, the player should always have some idea of the benefits of success and the price of failure or, if the player has to make a decision, the likely consequences associated with his options. Old video games used to implement a *learn by dying* approach, which gave players no means of knowing what elements of the game world were dangerous and what weren't, so the avatars died repeatedly as the players learned. Industry professionals now consider this extremely bad design practice. Although the player should not necessarily know every detail of what consequences his decisions will produce, he should be able to make a reasonable guess based on the context in which you present the decision. If you give him a doorknob, it should open the door. It may *also* release a giant killer robo-camel into the room, but it should open the door first.

- **Reward the player for skill, imagination, intelligence, and dedication.** These four qualities distinguish a good player, and good players deserve to be rewarded. You may create rewards in many forms: powerups and other resources, shortcuts through the level, secret levels, minigames, cut-scenes and other narrative material, or simple praise. Players like to be told when they've done a good job.

- **Reward in a large way, punish in a small way,** or to use an old adage, you catch more flies with honey than vinegar. The hope of success motivates players more than the fear of failure does. If a game repeatedly smacks them down hard, players will become discouraged and abandon the game with a feeling that they're being abused. Don't forget that the *duty to empathize* is one of the obligations of player-centric game design: Your primary objective is to give players an enjoyable experience. Build more rewards into your level than punishments.

- **The foreground takes precedence over the background.** Design the visual appearance of your level so that the player's attention is naturally drawn to his immediate surroundings. Don't make the background so complex that it distracts the player. Spend more of your machine's limited resources (polygons, memory, CPU time) on foreground objects than on background ones.

- **The purpose of an artificial opponent is to put up a good fight and then lose.** Design your level so that the player will get better and better at overcoming the challenges until he succeeds at all of them. In a multiplayer competitive game, the skill and luck of the players decide who wins, but in a single-player game, you always want the player to win eventually, and it's up to you to make sure that happens. An unbeatable level is a badly designed level.

- **Implement multiple difficulty settings if possible.** Make your game accessible to a wider audience by allowing them to switch the difficulty of your game to easy, normal, or hard settings. In games with an internal economy, you should be able to tweak the numbers to adjust the difficulty to accommodate the player's preference; Chapter 11, "Game Balancing," addresses this in more detail.

THE 400 PROJECT

In 2001, veteran game designers Hal Barwood and Noah Falstein began assembling a list of design rules for video games, hoping eventually to compile a list of as many as 400 of them. So far, Barwood and Falstein's *400 Project* has gathered more than 100 rules from fellow designers; these represent the combined wisdom of many people. Rather than outright commandments, these are tools to guide your own creative work, as a ruler guides a pencil. Some of the rules conflict with others, and it will be up to you to decide when one rule is more important than another. You should download the rules and take them to heart as you design your game and the levels within it. You can find *The 400 Project* at www.finitearts.com/400P/400project.htm.

Genre-Specific Level Design Principles

Some principles of level design apply only to games within specific genres. Since there isn't room to present a comprehensive list of principles specific to each genre, this section offers *one* highly important genre-specific principle for each genre covered in Part Two, "The Genres of Games," of this book. For more details on each genre, see the relevant chapter in Part Two.

ACTION GAMES

Vary the pace. Action games put more stress on the player than any other genre does, so the universal principle *vary the pace* applies more strongly to action games than to other genres; that is why it is the most important genre-specific principle as well. Players must be able to rest, both physically and mentally, between bouts of high-speed action.

STRATEGY GAMES

Reward planning. Strategic thinking means planning—anticipating an opponent's moves and preparing a defense, as well as planning attacks and considering an opponent's possible defensive moves. Design levels that reward planning. Give players defensible locations to build in and advantageous positions to attack from, but let the players discover these places for themselves.

ROLE-PLAYING GAMES

Offer opportunities for character growth and player self-expression. Character growth is a major player goal in any RPG; some players consider it even more important than victory. Every level should provide opportunities to achieve character growth by whatever means the game rewards—combat, puzzle-solving, trade, and so on. RPGs also entertain by allowing players to express themselves; that is, to role play. Every level should include opportunities for the player to make decisions that reflect the player's persona in the game.

SPORTS GAMES

Verisimilitude is vital. Sports games, while not ordinarily broken into levels in the usual sense, consist of individual matches played in different stadiums or courses with different teams or athletes, so you can think of each match played as a level. Level designers design the stadiums and sometimes the teams and athletes. More than in any other genre, players of sports games value a close relationship between the video game and the real world. The simulation of match play must be completely convincing; try to model each team and each stadium as closely as possible to the real thing—which includes not only appearances but the performance characteristics of the athletes and the coaching strategies of the teams.

VEHICLE SIMULATIONS

Reward skillful maneuvering. All vehicle simulations offer steering a vehicle as the primary player activity and steering well, often in adverse circumstances, as the primary challenge. Construct levels that test the player's skill at maneuvering his vehicle and reward him for his prowess. Other challenges, such as shooting or exploring, should be secondary.

CONSTRUCTION AND MANAGEMENT SIMULATIONS

Offer an interesting variety of initial conditions and goals. Most construction and management simulations (CMS) start the player with an empty space and let her build whatever she likes within the constraints of the game's internal economy. In such games, you won't need to do much level design. However, a CMS can also offer the player an existing or partial construction and let her continue working from there, often with a goal to achieve within a certain time limit. These are normally called *scenarios* rather than *levels*, the difference being that a scenario, unlike a level, consists of a self-contained situation unrelated to any of the other scenarios. Typically the game allows players to try the scenarios in any order, and the gameplay (though not the goal) tends to be identical in each. Because you cannot alter the gameplay, scenario design becomes a matter of offering an interesting variety of initial conditions and goals. *SimCity 3000 Unlimited* comes with 13 scenarios, from reuniting East and West Berlin to preparing for a World Cup soccer match in Seoul.

ADVENTURE GAMES

Construct challenges that harmonize with their locations and the story. Adventure games offer much of their entertainment through exploration and puzzle-solving. Designers set different chapters of an adventure game in different locations or landscapes to add novelty and interest to the experience. (A chapter is the adventure game equivalent of a level.) Create challenges that harmonize with the current level and with the current events in the story. In a room full of machinery, the challenges should involve machines; on a farm, the challenges should involve farm animals or implements. This principle applies to some extent to any game, but because story is so important in adventure games, the principle is especially important for that genre.

ARTIFICIAL LIFE GAMES

Create many interaction opportunities for the creatures in their environment. Much of the enjoyment in playing an artificial life (A-life) game comes from watching the simulated creatures in the game and giving them things to do within their environment. The level designer for an A-life game, then, needs to create interaction opportunities. The game should also offer many opportunities for the player to interact with the creatures as well, but generally the game designer, not the level designer, specifies these.

PUZZLE GAMES

Give the player time to think. Puzzle-solving is problem-solving, and it knows no timetable. Few players enjoy being forced to solve puzzles under time pressure. (*Tetris*, a famous exception, at least lets the player pause the game.) You may not be able to offer the player multiple difficulty levels due to the complexity of balancing puzzle games—for further discussion of this issue, see Chapter 20, "Artificial Life and Puzzle Games"—which is another reason that time to think becomes important. Either create puzzles that give the player complete freedom to think things through before acting or allow him to pause the game and study the screen for a while.

Layouts

For games that involve travel, especially avatar-based games, the layout of the space significantly affects the player's perception of the experience. Over the years, a few common patterns have emerged, which this section introduces in simplified form. You should not hesitate to create any layout that your game needs.

Open Layouts

In an open layout, the player benefits from almost entirely unconstrained movement. An open layout corresponds to the outdoors, with an avatar in principle free to wander in any direction at any time. Even levels with open layouts, though, may include a few small regions that cannot be entered without difficulty or can be entered by only a single path (such as passing through a door into a building). War games make extensive use of open layouts, *Battlefield 1942* being a particularly successful example. Role-playing games offer open layouts while the player goes adventuring outdoors, but they typically switch to network or combination layouts (described later) when the party goes indoors or underground.

Linear Layouts

A linear layout requires the player to experience the game's spaces in a fixed sequence with no side corridors or branches. It does not mean that the spaces are actually arranged in a line (see **Figure 12.1**). A player following a linear path can move only to the next area or to the previous area and does not have to make any decisions about where to go next. A game in which all levels use linear layouts is often said to be *on rails* because, like a train on a track, the traveler goes wherever the predefined route takes her. Ordinarily, the player has no reason to go backward in a linear layout unless she forgot to pick up something that she needs. Linear layouts often require players to pass through one-way doors that actually prevent them from going back, so long as they have collected everything they need to go on. Be sure you don't lock a player out of a region that contains an item essential to her later progress—an elementary level design error.

Linear layouts naturally work well with linear stories; if your game features such a story, you might consider such a layout. See Chapter 7, "Storytelling and Narrative," for more on linear stories.

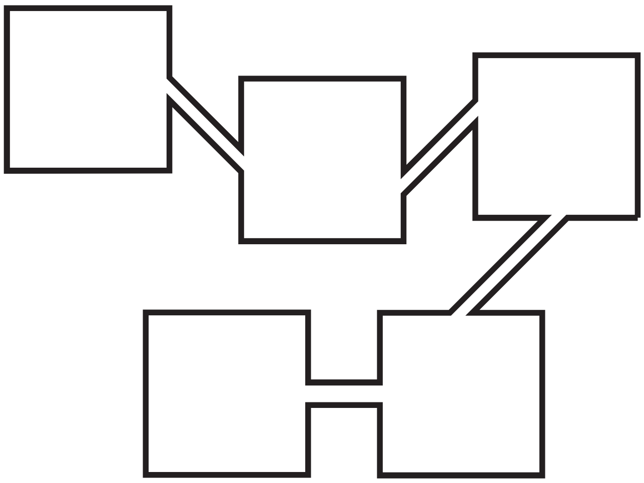

FIGURE 12.1 A level with a linear layout

Traditional for side-scrolling action games and rail-shooters, the linear layout is otherwise uncommon nowadays. Today's designers tend to favor the parallel layout.

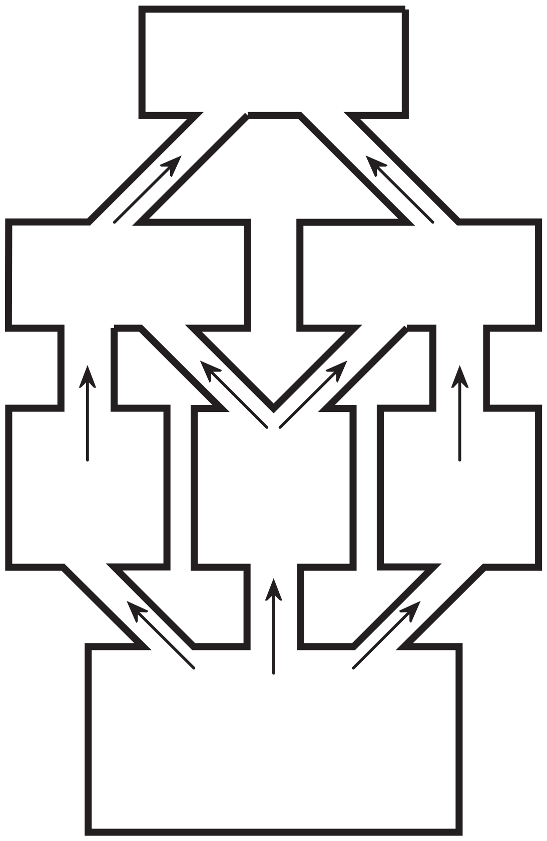

FIGURE 12.2 A level with a parallel layout

Parallel Layouts

A parallel layout—a modern variant of the linear layout—resembles a railroad switchyard with lots of parallel tracks and the means for the player to switch from one track to another at intervals. The player passes through the level from one end to another but may take a variety of paths to get there. See **Figure 12.2** for a much-simplified illustration.

Even though the parallel layout does not require players to pass through every available path, most players search them all anyway if the game lets them do so. One path may offer a greater risk and therefore a greater reward, while another path may give the player greater insights into the storyline. You can easily construct a parallel layout to reflect a foldback story structure. (See Chapter 7 for a discussion of story structures.)

You can also use parallel paths to provide shortcuts that let a player bypass particularly difficult challenges that lie on the more obvious path. If you do so, you may want to hide the entrance to the shortcut so only a particularly dedicated explorer will find it. When you create a hidden entrance, you must provide some clue, however subtle, that it is there. Otherwise, finding it becomes a trial-and-error challenge, a sign of bad design. The original *Wolfenstein 3D* contained hidden rooms accessible only through wall panels that looked exactly like the rest of the wall, which forced players to check every single wall panel in the entire level to see which might conceal a hidden room.

Ring Layouts

In a ring layout, the path returns to its starting point, although you may include shortcuts that cut off a portion of the journey (see **Figure 12.3**). Designers mainly use ring layouts for racing games, in which players pass through the same space a number of times, facing challenges from the environment and each other along the way. Shortcuts require less time but should be proportionately more difficult than the regular route; balancing this will be a big part of the level designer's job.

Rings do not necessarily look like circles. Oval tracks or twisting road-racing tracks qualify as rings.

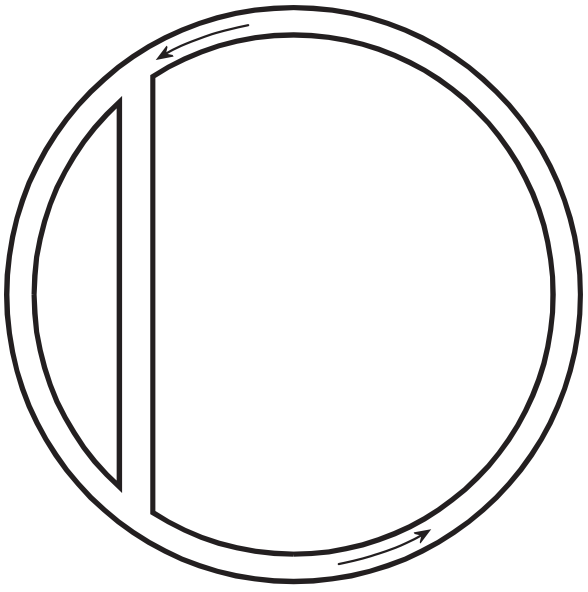

FIGURE 12.3
A level with a ring layout

Network Layouts

Spaces in a network layout connect to other spaces in a variety of ways. **Figure 12.4** shows a simple example. A large network poses a considerable exploration challenge; just learning the way around made up a significant part of the gameplay in old text adventure games. Modern graphical games that implement three-dimensional spaces usually present architecturally appropriate and logical networks (going downstairs from the ground floor of a building leads to the basement, for instance) but still offer plenty of opportunities to create enjoyable exploration challenges. See the section "Exploration Challenges" in Chapter 9 for further discussion.

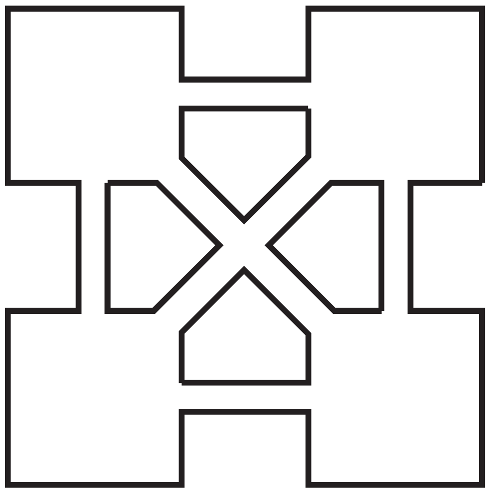
FIGURE 12.4
A level with a simple network layout

A network layout gives the player considerable freedom about what path to take, so you will find it difficult to tell a story that requires a particular sequence of events in a network layout. This doesn't mean that you can't tell stories, only that your stories have to tolerate the player experiencing events in any sequence. To enforce *some* sequence, use a combination layout, described in the later section "Combinations of Layouts."

In a network with a small number of major spaces, every space may be connected to every other space for maximum freedom of movement. This arrangement poses little exploration challenge to the player but makes an ideal fighting ground for deathmatch contests in games such as *Quake* because there are no choke points. Enemies may enter and exit in several directions, which prevents a player from guarding one particular location indefinitely.

Hub-and-Spoke Layouts

In the hub-and-spoke layout, the player begins in a central hub that ordinarily doesn't present significant challenges or dangers. As such, it serves as a place of comfort or safety, a base to which to return. To explore the rest of the world, the player follows a linear path out from the hub and then returns back to the hub on the same path (see **Figure 12.5**). The return journey either should be quick—because the player covers old ground during the return—or should offer new opportunities for gameplay and new rewards as the player comes back. Normally you would also put a major challenge and a major reward at the outer end of the spoke.

This layout gives the player some choice about where he goes, which many players appreciate. You need not offer the player access to all the spokes at the beginning of the level; to make sure that the player doesn't try the harder challenges too soon, you can lock off some areas until the player tries the easier challenges available in other spokes. Note that if you unlock the spokes only one at a time, you effectively change the hub-and-spoke layout into a linear layout.

The *Spyro the Dragon* games use a hub-and-spoke plan. The games include several hubs, called *homeworlds*, each of which is the center of its level. Various spokes lead off from the hubs to areas with different themes.

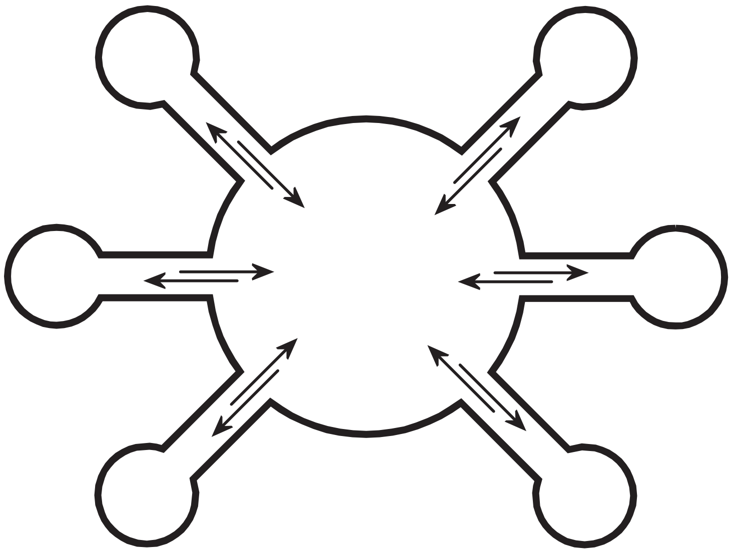

FIGURE 12.5
The hub-and-spoke layout

Combinations of Layouts

Many layouts combine aspects of each type of layout, providing, for instance, networked spaces to accomplish tasks within a larger linear framework. The layout in **Figure 12.6** corresponds to the story structure of many large RPGs, which tend to offer one major story arc and a large number of subplots or quests. Adventure games quite often use a combination structure too, letting players do considerable exploration in one area before moving on to another. Note the similarities with Figure 19.7 in Chapter 19, "Adventure Games."

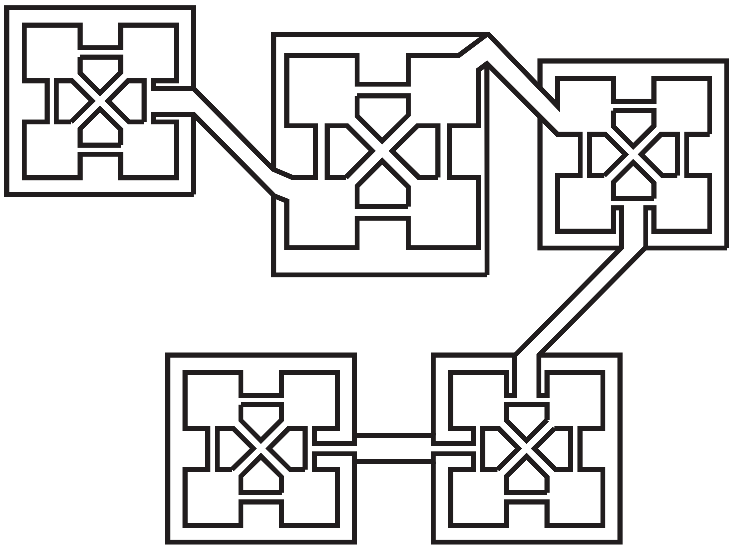

FIGURE 12.6
A combined linear and networked layout

Expanding on the Principles of Level Design

This section looks at a few particularly important issues in the list of universal design principles: atmosphere, pacing, and tutorial levels.

Atmosphere

The art director and lead game designer decide on the overall look of a game; the artists build the models; and the audio engineers create the sound effects. But it's up to the level designer to assemble all this material into a specific level in such a way that it's aesthetically coherent and creates the appropriate mood. A level designer does what in movies would be four or five jobs—set designer, lighting designer, special effects designer, Foley editor, and even cinematographer—because a level designer must look at the game world the way the player sees it, through the lens of the game's virtual camera.

As you work to establish the atmosphere of your game, you will use all of the following tools:

- **Lighting.** The placement and orientation of the lights in a level can create a sunny day, a moonlit night, or a dark alley. Soft morning light filtering in through a window creates a sense of warmth and well-being, whereas the odd glowing colored lights of a machine room evoke a sense of danger. The yellow of a sodium vapor street lamp or the harsh fluorescent lights of an office and any other lighting you choose must work with other aesthetic choices you make to set the mood of a level. What you choose *not* to light is just as important as what you choose *to* light.
- **Color palette.** Just as the color palette of the avatar's clothes reflects her character, the color palette of the level reflects its mood. The color palette of the level will emerge from a combination of the original colors of the objects you place in it (created by the artists under neutral lighting conditions) plus the lighting that you add. Notice how television commercials use color to telegraph an emotion, calm you down, get you excited, or keep you interested in watching. Do some research on color, and you will find many ways to create an effect in your level or elicit a particular response from the player.
- **Weather and atmospheric effects.** Fog, rain, snow, and wind all create distinct impressions. So many games take place in indoor spaces that we sometimes forget the importance of weather to our moods. Dark, tumbling skies presage a storm and make us instinctively react with "Find shelter!" even in a video game. Fog creates mystery, while strong winds suggest instability and disturbances to come.
- **Special visual effects.** When weapons recoil or screeching tires create smoke, when magic spells produce colored sparks or blood splashes across a wall, you're seeing visual effects. You can startle players, discomfit them, amuse them, or reward them, all with visual effects.

- **Music.** You won't write the music unless you're also a musician, but you may well choose the music of your level in conjunction with your game's audio director. The rhythm of the music helps to set the pace, and its timbre and key help to set the mood. Generally, but not always, music remains consistent throughout the level, part of its overall tone.
- **Ambient audio.** Like music, ambient audio contributes to the overall mood of a level. Notice how golf games use the sounds of birds singing and crickets chirping to suggest the peaceful outdoor tranquility of a golf course. The ambient audio can also vary with place and time, which tells the player something about where he is and helps him to orient himself. Great steam engines churning create a feeling of power and danger; owls hooting and foxes crying tell us it's nighttime; the hubbub of talk and regular cries of vendors put us in a market square.
- **Special audio effects.** Audio effects naturally do for the ears what visual effects do for the eyes, and in some respects, provide even more important information. From inside a car, you can't see the tires losing their grip on the road, but the squealing sound tells you you're on the edge of danger—you're pushing the vehicle to its limits.

Progression and Pacing

A large video game—one designed to be played for more than an hour, say—is almost always divided into a number of levels. If you want the player to experience those levels in a sequence, they should exhibit *progression* of some kind: changes from level to level that represent growth in some form, or narrative advancement, or both.

The *pacing* of a level refers to the frequency at which the player encounters individual challenges. A fast pace creates *stress*, offering challenges at a rapid rate while giving the player no opportunity to relax. (Chapter 9 defines stress and discusses the relationship between *stress* and *difficulty*.) A slow pace offers challenges at a slow rate and permits the player to take his time about addressing them.

This section of the chapter discusses both progression and pacing, and how to design them properly.

DESIGNING THE PROGRESSION

Games obviously need to change from level to level, but *how* should they change? Designer Mike Lopez has written a useful article on the subject in his "Gameplay Design Fundamentals" column for the *Gamasutra* webzine (Lopez, 2006). He identifies five game features that should exhibit progression throughout the game; these serve us as a starting point:

- **Mechanics.** Lopez uses this term to refer both to the core mechanics of the game and the actions available to the player. This book organizes these concepts differently, so we'll look at core mechanics here and actions later. Generally speaking, the core mechanics should become richer as the game goes along. In the early levels, especially the tutorial levels, the internal economy of the game should be easy for the player to learn. Later, the mechanics can become more intricate, as in games like the *Civilization* series. Many games also exhibit economic growth throughout the game, so the player is dealing with larger and larger quantities of resources—money, hit points, horsepower, or whatever the game deems to be of value.

- **Experience duration.** Except for the occasional atypical level (see the later section "Make Atypical Levels Optional"), it should take more and more time to play through each subsequent level. This rule is not absolute, but generally speaking, levels later in the game should be longer than those earlier in the game.

- **Ancillary rewards and environmental progression.** *Ancillary rewards* are unrelated to the gameplay: cut-scenes, trophies, unlockable content, and so on. (When the player gets to the end of *Silent Hill 3,* she earns the right to dress Heather, the avatar character, in new clothes and play the game again wearing them. This has no effect on the gameplay.) By *environmental progression* Lopez means enjoyable changes in the landscape of the game world, which makes sense when the game involves travel. Both of these provide novelty, one of the ways that video games entertain.

- **Practical gameplay rewards.** These are rewards that directly influence the player's future gameplay: new vehicles in driving games; new gear or skills in role-playing games; new moves or characters in fighting games; new technology in strategy games, and so on.

- **Difficulty.** Generally speaking, the perceived difficulty of the game should go up, remaining flat only for games for small children and some casual games. Chapter 11 dealt with this issue extensively.

In addition to Lopez's list of features, you may wish to consider a few more:

- **Actions available to the player.** Lopez lumped these together with mechanics, but they aren't quite the same. A game can possess core mechanics that don't change much from level to level, yet still offer players new moves or other activities to perform as the game goes along. This is particularly noticeable in platform games. It's always a good idea to introduce new actions through a series of tutorial levels so that players can become skilled with one before learning the next one.

- **Story progression.** As your player progresses through the game, he should also progress through the story, if it has one. Exactly how this happens depends on a number of design decisions you must make: whether the plot is linear or not, and what mechanism causes the plot to advance. Chapter 7 addresses these details.

- **Character growth.** Video game characters often become more powerful through practical gameplay rewards, and sometimes they become more visually interesting through ancillary rewards such as new clothing. But you can also make them grow in a literary sense: become more mature, well-rounded people. A character who doesn't grow, especially over the course of several games, eventually begins to seem like a cartoon character with no emotional depth.

It will be easiest to implement these features if you organize your game into a number of discrete levels, each of which contains its own environment, starting conditions, victory condition, and so on. However, levels are naturally rather artificial. If you want to offer a strongly storylike experience, you may prefer to avoid having breaks between one level and the next, and try to create an entirely seamless experience. *Half-Life* is a famous example.

DESIGNING THE PACING

The pace you choose for your level will depend to a considerable extent on the genre of the game you're creating; players expect a faster- or slower-paced game depending upon the genre. The fastest-paced games of all, the old 2D side-scrolling or top-scrolling shooters, required players to move the joystick and bang the fire button continuously just to survive. Multiplayer deathmatch shooters such as *Quake* and its kin represent the modern equivalent. (Stealth games such as the *Rainbow Six* series, which involve careful planning, often move at a slow pace except for a brief wild flurry when the enemy comes into view.) Adventure games use the slowest pace because much of the activity consists of interactive dialog (generally a story action rather than an action the player takes to surmount a challenge), exploration without much effort, and puzzle solving in which the players can take as long as they like. Play a variety of games and study their pacing. Chapter 13, "Action Games," discusses pacing extensively because it is so important in that genre.

CLASSIC ARCADE PACING

In arcade games, especially old ones such as *Space Invaders*, the pace at which the player faces challenges becomes faster and faster as each level progresses. If the player succeeds in beating the level—destroying all the invading aliens—he gets a few seconds of rest before the next level begins. The next level offers identical challenges, but it starts at a faster pace than the previous level started and ends at a faster pace than the previous level ended. The pace of *Space Invaders* increases both within each level and from level to level until it overwhelms the player and he loses the game. He cannot win; he can only hope to get a high score.

This classic arcade pacing explains how arcade games used to make their money. This scenario is now considered a bit old-fashioned and inappropriate for console and PC games because they don't need to make the player lose to force him to put more money in the machine. However, with the continuing popularity of retro gaming, classic arcade pacing remains common in simple web-based games such as *Collapse!*

VARY THE PACING

As a general principle, the pacing of a level in any game, especially a game with physical challenges, should alternate between fast and slow periods, just as the tempo of movements in a symphony or the levels of excitement in an action movie vary. Players need moments to rest, both physically and mentally, and on the whole, the faster the pace of the level, the more important rest becomes. A particularly stressful challenge should be followed by a brief period with no challenges at all and then by easier challenges that gradually ramp up to more stressful ones again. This also gives the player a chance to savor the pleasant emotions that accompany success.

Varying the pace not only gives the player a rest from physical challenges, it also produces a more balanced game. If overcoming a challenge requires spending a resource (ammunition or health or the like), then the more the player spends on a given challenge, the weaker and more vulnerable he is afterward. In his weakened state, he should not face another demanding challenge immediately. You should also make fresh supplies available to him immediately after he surmounts a challenge that costs him a lot of resources, as Chapter 11 explained. In shooter games, these traditionally take the form of boxes of ammunition and medical kits for restoring health, stored in an area immediately beyond a large group of enemies. In role-playing games, enemies drop valuable resources when killed, thus helping to replenish the player's supply.

PACING IN *THE LORD OF THE RINGS*

For a wonderful example of varied pacing from literature, read *The Lord of the Rings*. Almost every major adventure or threat the Fellowship experiences in the first two volumes is followed by a period of rest and refreshment to heal wounds and in particular to replenish food supplies. The hobbits flee the Black Riders and take refuge with Farmer Maggot. They are caught in the Old Forest and rescued by Tom Bombadil. After the attack at Weathertop, they find shelter in Rivendell. After losing Gandalf in the Mines of Moria, they find succor in Lórien, and so on. This change of pace not only creates emotional variety for the reader, allowing her to enjoy the beauty and warmth of the heroes' places of shelter after the terrors of their journey, but it also makes the story more credible. No one can carry six months' worth of food on his back, so the supplies had to come from somewhere.

The Da Vinci Code, notwithstanding its financial success, is less credible in this regard. Involved in almost nonstop action from start to finish, the heroes never seem to need any sleep.

You can vary the pacing in a variety of ways: by creating an area free of challenges in which the player can simply explore; by creating an area that contains only low-stress challenges; or by making the player's avatar temporarily invulnerable or particularly strong as a reward for successfully overcoming a demanding challenge. You can also deliver a bit of the story through narrative: Watching a cut-scene, for example, gives the player a moment to relax.

You will find it easiest to vary the pacing in games that involve avatar travel through a linear space, because you can control the sequence in which the player confronts challenges. Games that give the player freedom to explore at will give you less control. In genres that use multipresent interaction models rather than avatar- or party-based ones, you may have little control at all. For example, in a real-time strategy game, the pacing depends to a large degree on the player's own style of play. Those who attack aggressively experience a faster pace than those who slowly build up huge armies before attacking.

OVERALL PACING

Although the pacing of a level should vary from time to time (depending on the genre), the overall pacing of the level should either remain steady or become more rapid as the player nears the end. A longstanding tradition in action games, and many other genres as well, calls for the inclusion of a *boss* to defeat at the end of the level: a particularly difficult challenge. Victory, and the end of the level, reward the player for defeating the boss, and this sometimes includes a cache of resources or treasure as well. Bosses, although something of a cliché, fit neatly into games with a Hero's Journey story structure. Chapter 13 discusses bosses in greater detail.

Levels should not, in general, get easier and easier as they go along. If the player does well, *positive feedback* may come into play to make the game easier, and you will need to design the level, or the core mechanics, to reduce that effect. Chapter 11 discusses positive feedback at length, including various means of limiting it.

Tutorial Levels

Years ago, video games shipped with large manuals that explained how to play the games. Designers had no other way to teach the player because the distribution media (cartridges and floppy disks) couldn't hold enough data to spare any room for tutorial levels. Nowadays, however, all games should be designed so that the player can start playing immediately. Games still use manuals, mostly in electronic form, but for detailed reference information rather than instructions.

TIP If your game is complex enough to need a manual, be sure to make the manual available for download from a web site or page dedicated to the game. Players lose manuals.

Instead of instructions, games offer *tutorial levels*—early levels that teach the player how to play. Every commercial game except the simplest ones should include one or more tutorial levels. Although tutorial levels require more time and effort to build than a manual does to write, tutorial levels have the tremendous advantage that they let the player learn in a hands-on fashion. Players learn physical activities, such as how the control devices function in the game, far more quickly if they can try the actions for themselves.

TIP If you discover that players are often uncertain about how to play, establish a Frequently Asked Questions (FAQ) web page for your game. Ideally, however, you will design the game so well that there won't be any frequently asked questions!

A tutorial level is not simply an easy level or a short level. A tutorial level should be a scripted or partially scripted experience that explains the game's user interface, key challenges, and actions to the player. Use voiceover narration, text superimposed on the screen, or a special mentor character to explain things to the player.

As you design one or more tutorial levels for your game, consider these key principles:

- Introduce the game's features in an orderly sequence, starting with the most general and most often used features and proceeding to the more specialized and rarely used ones. Your tutorial should introduce each individual action that the game permits, but it need not discuss combinations of actions and what effects they may have. The players can work that out for themselves.
- Don't make all the game's features available at once. It will only confuse the player if he happens to select, by accident, a maneuver that you haven't yet introduced, which produces an effect on the screen that the player doesn't understand. Disable features until the tutorial introduces them.
- If the interface is complex, as interfaces tend to be in many war games and construction and management simulations, introduce the information over two or three tutorial levels.
- Highlight user interface elements that appear on the screen with an arrow or a colored glow whenever your explanatory text or helpful guide character refers to them. Don't just say where these items appear on the screen and make the player look for them.
- Let the player go back and try things again as often as he wants, without any penalty for failure. All the costs of making a mistake that you might put into the ordinary game world should be switched off in the tutorial levels.

DESIGN RULE Make Tutorial Levels Optional

Make the tutorial levels optional. Experienced players may not need them and will be irritated by being forced to go through them. (*America's Army* violated this rule, largely because of the game's function as a representation of the U.S. Army. The developers wanted to make the point that not just anybody would be allowed into the army, so the tutorial levels symbolized Basic Training in the real army. *America's Army* is not a pure entertainment product, however.)

The Level Design Process

Now that you have learned the general *principles* of level design, let's turn to the *process*. Level design takes place during the elaboration stage of game design and, like the overall game design, is an iterative process. At points during the procedure, the level designers should show the work-in-progress to other members of the team for analysis and commentary. Early input from artists, programmers, and other designers prevents you from wasting time on overly complex levels, asking for features the programmers cannot implement, or making demands for artwork that the artists don't have time to meet.

At the 2004 Computer Game Technology Conference in Toronto, Canada, level designers Rick Knowles and Joseph Ganetakos of Pseudo Interactive presented an excellent lecture simply entitled "Level Design" (Knowles and Ganetakos, 2004). They described the 11-stage process by which their company builds levels, which is summarized here. The following sections assume that the development teams consist of game designers, artists, programmers, and sound designers, as well as you: the level designer.

Throughout the discussion of this process, you will notice a strong emphasis on the relationship between the level designer and the art team, and less emphasis on the relationship between the level designer and the audio or programming teams. The reason for this is that level designers build prototype artwork that the art team then uses as a blueprint from which to build final artwork that will actually go into the game. This requires that the level designers hand off their prototype to the art team and receive the final artwork back from the art team at particular stages in the process. The relationship with the programmers and the audio team is less sharply defined. Level designers request special features from these groups, and the project manager determines when and how that work gets done, but generally it doesn't involve handing off material to the audio or programming teams and receiving material back from them in the same way. Your relationship with the programmers and audio people is just as important as your relationship with the artists, but your interactions with them may be less formally scheduled.

A Note on Duties and Terminology

The nature of a level designer's job varies considerably depending on both the genre of the game and the technology that implements it. A few years ago, level designers were not expected to possess either art or programming skills. As the size and complexity of games has increased, so has the size and complexity of the level designer's job. In modern 3D games, level designers often use 3D modeling tools to construct temporary—and sometimes even final—artwork to go into a game. (The term *model* refers to a three-dimensional geometric structure that depicts a single thing, such as a human, vehicle, tree, or the underlying landscape of a level.) Also, games now often include *scripting engines* that allow level designers to write small programs, or *scripts*, that control some aspects of the behavior of the level during play. Scripting engines normally implement scripting languages less powerful than the programming language used by the programmers, but the scripting language will be sufficient for defining the behavior of automated traps, doors, and other special events that may occur in the level. There isn't room in this book to teach you the skills you need to use such tools, but you can find many resources for learning to use them on the Internet and at colleges and universities.

For simplicity's sake, this section assumes that you are creating levels for a game that uses a 3D graphics engine to display a 3D game world. If you are making a 2D game, where it refers to models, think in terms of their 2D equivalents: *sprites* (2D art and animation) for movable objects and the *background* (a 2D painting, often made up of interchangeable rectangular tiles) for the landscape.

Design to Level Design Handoff

In the first stage, the game designers will tell you in a general way what they want for the level: its setting, mood, key gameplay activities, and events. You should then generate a list of features you want to appear in the level:

- Events that can be triggered by player action
- Props (objects that will be present in the level)
- Nonplayer characters (NPCs)

At this point you also create a rough overview map of the level, showing how the landscape varies and what props and NPCs will be in which areas. See **Figure 12.7** for an example from an unproduced driving game by Pseudo Interactive, set on islands inhabited by dinosaurs.

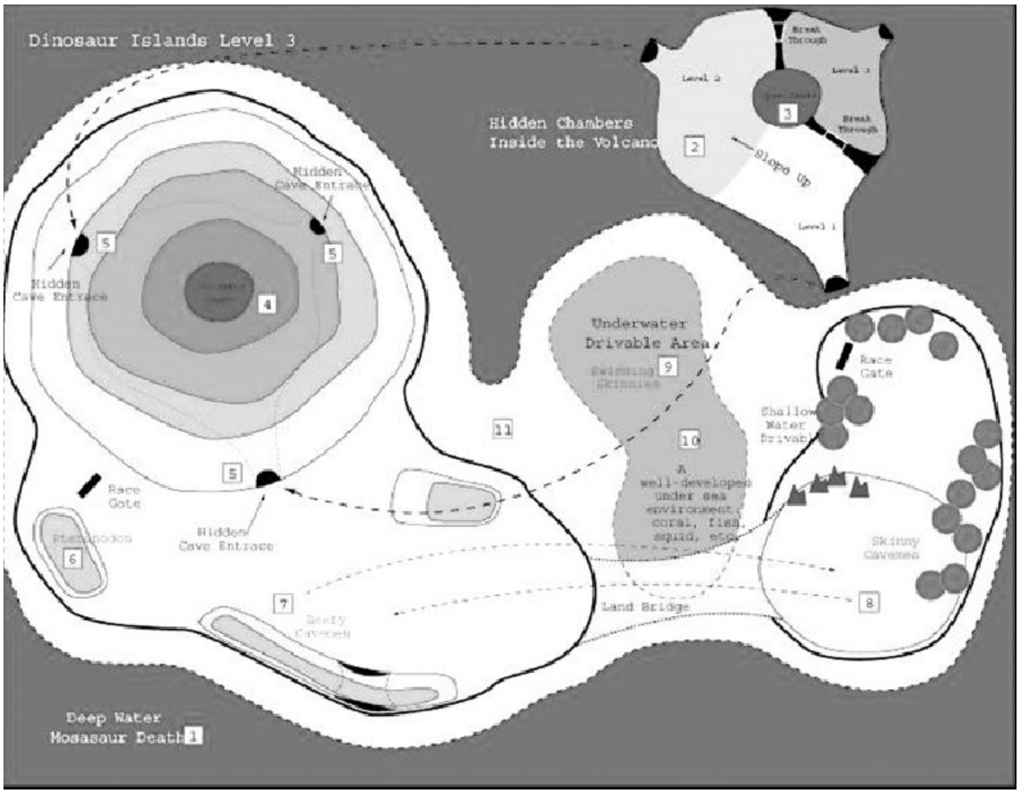

FIGURE 12.7 Rough level sketch for a driving game showing key features. Image courtesy Pseudo Interactive.

Planning Phase

Armed with the list and sketch created in the first stage, you now start to plan the level in detail. Use pencil and paper to work out the sequence of events: both what you expect the player(s) to do and how the game will respond. Begin to document your decisions in the following key areas: gameplay, art, performance, and code requirements.

GAMEPLAY

As you plan the gameplay for your level, you will need to consider all the following issues:

- **Layout** (discussed extensively in the "Layouts" section earlier). Where can the player-controlled characters (avatar, party, or units) go and where can they not go? What paths can they use to get there? Many parts of your level may be cosmetic: The player can see them but cannot reach them.
- **Areas devoted to major challenges.** Which areas carry strategic importance? Which will offer the biggest challenges? If the game involves combat, where would you like it to occur?
- **Pacing.** How will the intensity of action vary throughout the level? Where will the key events and the rest periods occur?
- **Termination conditions.** How does the player win or lose the level?
- **Resource placements.** Are depots of weapons, health points, powerups, or any other resources hidden in the environment? Where? What resources, and how much?
- **Player start and end points.** Do the player-controlled characters begin the level at one or more specific locations? Where? Do the characters end at one or more locations? Where?
- **NPC positions and spawn points.** If NPCs—whether enemies, friends, or neutrals—appear in the level, where are they initially positioned? Can they suddenly appear in the level at a specific location or *spawn point* during play? Where?
- **Elevations.** How much vertical movement does the level permit, and how does that affect play? Higher elevations naturally allow the player to see farther in first- and third-person perspectives; will this cause problems or constitute a positive feature of your level?
- **Secret areas**. Do you plan to incorporate hidden areas or secret shortcuts? Where will they be, and what clues will be available to suggest they might be present?
- **Special event issues.** What special events, unique to this level, can occur? Where will they occur? What will set them off? How do the special events reflect the setting and tone of the level?

- **Landmarks.** How does the player find her way around? How can she tell where she is? Establishing major landmarks will help her out.
- **Destruction.** Can any part of the level be destroyed or its landscape radically altered? Where does this happen and what causes it? How does it affect the gameplay? Does it have the potential to introduce anomalies, such as enemies who wander off the edge of the world and never return?
- **Storytelling.** How does the sequence of events the player experiences integrate with the game's story? Which events are dramatically meaningful and which are not? Where and when do you want cut-scenes or other narrative events to occur?
- **Save points and checkpoints.** Does the level include save points or checkpoints? Where? In games in which the player fails frequently and has to reload, positioning the save points is a critically important part of balancing the game.

ART

In the art planning phase, you determine the *scope* of your level and decide how much artwork it will need. Scope refers to the magnitude and complexity of the level, both in terms of the number of objects and characters that it contains and the special events that it includes. You can make a serious error by choosing too large a scope, because if you overload your art staff, you may never get the level finished at all. See "Get the Scope Right" near the end of this chapter.

You already have your sketch and a general idea of what the environment will be like, whether on the sea floor, in outer space, or inside an anthill. First decide on the scale of the level: How big will this level be in the game world's units of measure? This will help you to determine just how many other features the level needs. In almost every genre, if you've balanced the challenges correctly, the size of the level is directly proportional to the length of time that it takes the player to play through that level, so the scale you choose will, in a rough way, determine how much gameplay you can offer.

Next, start thinking about the kinds of objects that should be present in the level. Do research at the library or on the Internet for visual reference material to give you inspiration. Count the number of unique types of props that the level will require and plan in a general way where to put them. Certain generic items such as streetlights (or the infamous crates in first-person shooters) can simply be duplicated, but natural objects such as trees and boulders should come in several types, and the art team will need to know this. Try to avoid including too many identical objects in a level; it destroys realism.

Create a list of textures that the level will probably need. In an office, you may need tiles for the floor coverings, wood or metal for the desks, fabric for the chairs, and so on. Some offices may be streamlined, with severe geometric shapes, whereas others may be ornate, featuring a Louis XIV desk and antique chairs.

Decide on the visual appearance of any special effects that the artists will have to implement. It may take a while for the artists to come up with the visuals for a never-before-seen eruption of semisentient magma at zero gravity, so you need to plan ahead.

PERFORMANCE

You normally think of performance as the programmers' problem, but it's up to the level designer not to build a world that bogs down the machine. You will need to sit down with the programmers and set some boundaries. How complex can the geometry be? How far into the distance will the graphics engine be able to render objects? How many autonomously moving units or creatures can the game support at one time? Know your machine's limitations as you plan your level.

CODE

Finally, as part of the planning process, identify specific requests that you intend to make of the programmers for features unique to this level. These may take the form of special events (sometimes called *gags*) that require coding, unique NPCs who appear only in this level but need their own behavior model and artificial intelligence, or special development tools you may require in order to build and test the level effectively. The more of these special coding problems you identify during planning and can discuss with the programmers in advance, the more likely that implementation will go smoothly.

Working through these steps results in an initial plan for the level. Don't expect the numbers and details in this plan to exactly match what you end up with in the finished level, but working out in advance as much as you can will ensure a smoother design process. Charging in without a plan and making it up as you go along creates more problems in the long run.

Prototyping

In this stage, you will build a prototype of the level. Much of this work will consist of using a 3D modeling tool to construct temporary models of the landscape and objects that can appear within it. The models you create will not end up in the game but will serve as blueprints from which the art team will create the final artwork.

The prototyping phase requires that at least part of the game engine be running so that you can load the model into it and test it. Your prototype should include such features as:

- The basic geometry (physical shape) of the game world created in a 3D modeling tool. If it's a 2D world, the prototype should show the layout of the 2D landscape.
- Temporary textures to place on the geometry to give it a surface. These will eventually be replaced by final textures created by the artists.

- Temporary models of props (trees, furniture, buildings, and so on) and NPCs that will appear in the level, so you can put them where they belong in the landscape.
- Paths planned for AI-driven NPCs—where they travel within the level.
- A lighting design for the level.
- The locations of trigger points for key events. Placing these triggers and documenting what sets them off is referred to as *rigging*.

In some cases, you may be able to use final audio effects in your prototype; that is, the sound effects that will actually end up in the game. If those are not yet available from the audio team, use temporary sound effects and note that they will need to be replaced later.

Level Review

At this point, you have a working prototype of the level; if the programmers have the game engine running, you should be able to play your level in a rudimentary way. Hold a *level review*, inviting members of the design, art, programming, audio, and testing teams to get their feedback. Each should examine your prototype for potential problems that may come up in his own field when he is working on the real thing. The issues that the level review should address include these:

- **Scale.** Is the level the right size? Will it take too much or too little time to play through?
- **Pacing.** Does the flow of events feel right?
- **Placement of objects and triggers.** Are things where they need to be to make the level play smoothly and produce the experience you want?
- **Performance issues.** Is the level too complicated for the machine's processor to handle? The programmers should be able to flag any potential problems.
- **Other code issues.** Does the level call for software that represents a problem for the programmers? For example, a unique NPC that appears only in this level still needs its own AI; will this be an issue?
- **Aesthetics.** Is the level attractive and enjoyable to inhabit? Because the prototype uses temporary geometry and textures, a certain amount of imagination will be called for here.

Level Refinement and Lock-Down

After the level review, take the feedback you've received and refine the prototype, correcting any problems and implementing any new decisions made in the course of the review. This can require any amount of work from tuning a few numbers to scrapping the entire design and starting over from scratch. When you think you've got it right, hold another review and make another refinement pass. Continue this process until everyone agrees (or the person in charge agrees) that the level is ready to go into full production.

At this point, lock the level design. Once a level is designated as *locked*, no additions or changes may be made except if grave problems are discovered. This corresponds to the lock-down that occurs in overall game design at the end of the concept stage. If you don't treat the level as locked, you could go on tuning and tweaking it forever, stretching out the development time and running up the budget.

Level Design to Art Handoff

With the level locked, it's time to hand off your prototype and all your design work to the artists who will use it as a blueprint to build the geometry, animations, and textures that will end up in the real game. The artists will need all your files, as well as a detailed list that explains each file. Your job includes making this list; you cannot simply give them a directory dump and leave them to figure it out. If they don't already know about your design from the level reviews, you should sit down with the artists and give them a thorough briefing not only on how everything looks in the level, but where everything should be and how everything works. From this information, the art director will create a task list to construct all the content the level requires: models, textures, animations, special visual effects, and so on.

If your prototype has been relying on placeholder audio, at this point you will also need to provide details to the audio team about what the level will need in the way of final audio. Notify the programmers about any special code that is required for the level at this point so they can have it ready for the content integration stage. (*Content* refers to the non-software part of the game: artwork, audio, movies, and text.)

First Art and Rigging Pass

The project now enters the first art and rigging pass, during which the art team builds the real artwork and rigging. You may be working on other levels at the same time, but you should also stay in close touch with the art team because they will undoubtedly have questions. It may also be your responsibility to incorporate the content they create into the software, to make sure that it all works.

Art to Level Design Handoff and Review

When the art team finishes the final artwork, the artists hand all their work back to you, and you should conduct another review. This will highlight any problems or errors with the artwork that need correcting.

Content Integration

At this point, you will assemble all the assets into the completed (but not yet tested) level—artwork, new code required by the level, audio, and any remaining tweaks to the lighting. You'll also adjust any remaining issues with the rigging, by repositioning characters, effects, and triggers as necessary.

Bug Fixing

Test the level at this point, looking for bugs in the code and mistakes in the content. This will be another iterative process, working back and forth between the art, audio, and code teams and yourself. After finishing your own testing, you hand the level off to the quality assurance (QA) department for formal testing.

User Testing and Tuning

In the last stage, QA will create a test plan for the level and begin formal testing, known as *alpha testing*. Their testing will ordinarily be more thorough and strict than the testing you've done; it will also find things that you missed because of your overfamiliarity with the material. As in your own testing, they'll work in an iterative process with the various teams involved, including reporting the bugs in the rigging and gameplay mechanics that you need to fix. When QA considers the level to be thoroughly tested, they may make it available for *beta testing* (testing by end-users).

Pitfalls of Level Design

This chapter ends with a discussion of some important mistakes to avoid—classic errors of level design that, unfortunately, some designers continue to make.

Get the Scope Right

The single most common error made by inexperienced level designers is to try to build something too big. (They almost never try to build something too small.) Everyone would love to make an epic such as a *Final Fantasy* game, but such games require huge production teams, giant budgets, and multiyear development cycles. And even among experienced professionals, epic projects often run late and go over budget.

You must design within the resources of your team, your budget, and the time you have available. Scope, you should remember, refers not only to the size and complexity of the landscape but to the number of props, NPCs, and special events in the level. In order not to undertake an unrealistically large level, you must make lists of these things during the planning stage before you actually start constructing the prototype. The process of making these lists may surprise you by showing you just how much work goes into making even a relatively small level.

Before you choose a scope for your level, determine how much time and staff you have available, taking into account any vacations and holidays that may be coming up. Then assume that half of your team will be out sick for a week at some point during the development process—it's entirely possible. *Now* think again about the scope. How many models can your team build in a day? How quickly can you detect an error, correct it, and test it again? Choose a level size that you and your team can manage. If you make a level too small, it's not easy to enlarge it, but at least you won't have the art team killing themselves to create all the content. If you make a level too big and find that there isn't time to complete everything, you'll have to either deliver a sparse, unfinished level or scramble to cut things out, which will almost certainly harm your level's balance and pacing.

Avoid Conceptual Non Sequiturs

At the beginning of the first level of *James Bond: Tomorrow Never Dies*, the player, in the persona of James Bond, sneaks into an enemy military outpost armed only with a pistol and faces numerous Russian guards; how many, he doesn't know. If he blows up some of the oil drums scattered somewhat randomly outside the outpost, he will find medical kits hidden inside, which he can use later to restore his health when wounded.

Hiding medical kits inside oil drums belongs to a class of design errors, usually made at the level design stage, called *conceptual non sequiturs*—game features that make no sense. No sane person would think of looking in an oil drum to see if a medical kit might be hidden within. Furthermore, any thinking player would reason that if he's trying to sneak into an enemy military installation armed only with a pistol, causing a loud explosion right outside is not a good idea; several dozen people will come running to see what made the noise. He would further assume that any medical kit that *was* inside an oil drum when it blew up wouldn't be good for much afterward. Consequently, a reasonable player wouldn't blow up the oil drum and wouldn't get the benefit of the medical kit. In other words, the game punishes players for using their brains. It's simply poor design.

James Bond: Tomorrow Never Dies made the mistake of copying a 20-year-old cartoon-game mechanic—resources hidden in odd places—into a realistic game. A realistic game assumes that players can count on certain similarities between the real world and the game world (oil drums store oil, not medical kits; explosions destroy things rather than reveal things). No flight simulator bothers to explain gravity, for the same reason. The player of a realistic game expects the assumptions he makes in the real world to be valid in the game world. By violating these expectations with a conceptual non sequitur, *James Bond: Tomorrow Never Dies* became considerably harder for all but an experienced gamer who already knew the conventions of cartoon-style video games.

In short, avoid conceptual non sequiturs in realistic games. They discourage new players and make your game unnecessarily hard without making it more fun. Remember the principle that level designers should reward players for using their intelligence, not punish them for it.

Make Atypical Levels Optional

Level designers naturally like to vary the content of their levels, and it is good design practice to make creative use of the game's features or to set your levels in different environments to provide the novelty that players like.

Still, you should *not* create wildly atypical levels and force the player to play them in order to get through the game. Level designers sometimes create a level filled with only one kind of challenge—an action game level consisting of nothing but platform jumps, say, with no enemies to fight or treasure to find. Others like to take away some of the actions that a player uses routinely on other levels and force her to make do with a limited subset of actions for the duration. Some have created levels that borrow from a different genre entirely: a real-time strategy game level in which both sides control exactly one unit, thus turning the level into a strange sort of action game.

There are two reasons not to make these kinds of levels obligatory. First, it breaks the player's suspension of disbelief to be suddenly confronted with a situation that would never occur according to the rules of the game world as the player has already learned them. Second, it may actually make the game unwinnable for some players. If you create a level filled with only one kind of challenge, then a player who happens to be terrible at that kind of challenge—but who reasonably expected to make it through the game by being good at other kinds of challenges—might not be able to finish the game at all, stymied by one atypical level. And there may be many players who don't find that challenge as exciting as you do, who will find an entire level of it boring.

You shouldn't avoid making atypical levels at all; they can be a lot of fun. But make them optional—hidden levels the player can unlock through excellent play or side missions for extra points.

Don't Show the Player Everything at Once

As they say in theater, "Always leave them wanting more." This advice applies to the overall progression of the game, so both game designers and level designers need to be aware of it. If your players have faced every challenge, seen every environment, and used every action that you have to offer—all in a single level—then the rest of the game will be old hat for them. You have nothing further to offer but variations on a set of play mechanics and game worlds that they already know everything about. Let your game grow from level to level. Introduce new features gradually. Just as it all starts to seem a bit familiar, bring in a twist: a new vehicle, a new action, a new location, a new enemy, or a sharp change in the plot of the story.

Never Lose Sight of Your Audience

Level design, more than any other part of the game design and development process, brings with it the risk of building a game that your audience won't enjoy. *You* assemble all the components that the others provide, and when the player starts up the game, she finds herself in *your* environment. The game designers may decide on the types of challenges the game contains, but you decide when the player will face them, in what sequence, and in what combinations. Consequently, you, more than anyone else on the team, must apply the player-centric approach to every design decision you make. Go inside the mind of your player and try to imagine what it will be like to see it all for the first time.

Always remember that you are not the player. Your own personal circumstances have *nothing to do with the game*. You may be a 22-year-old male, but your player may well be a 10-year-old girl or a 50-year-old man. Understand the game's target audience and what that audience wants from the game; then make sure you give it to them—at all times!

TWINKIE DENIAL CONDITIONS

Since 1997, I have written a regular column called "The Designer's Notebook" for the *Gamasutra* developers' webzine. In the course of writing the column I have amassed a collection of design errors—mistakes to avoid—which I document in an annual column titled "Bad Game Designer, No Twinkie!" Many of these errors were suggested by other game designers or by angry gamers. The errors have come to be known as "Twinkie Denial Conditions." (A Twinkie is a snack cake often sold in vending machines; game developers frequently resort to vending machines for sustenance when they are working so late that all the pizza-delivery places are closed.) Some of the errors listed in this chapter are Twinkie Denial Conditions, but there are many more. You can find a complete list of Twinkie Denial Conditions with links to the articles in which they appeared at www.designersnotebook.com/Design_Resources/No_Twinkie_Database.

Summary

In this chapter, you explored level design, a key stage in the development of any video game. The level designer is responsible for actually presenting the game experience to the player by designing elements such as the space in which the game takes place, deciding what challenges a player will face at each level of the game, creating the atmosphere of the game world, and planning the pacing of events for each level. Level design is governed by universal principles as well as principles specific to the game's genre. In a strategy game, for example, the level design should reward planning; in a vehicle simulation, the level designer creates levels that test a player's skill at maneuvering her vehicle. An important aspect of level design is the actual layout of the level. Different stories require different layouts, but every layout should be designed to enhance the playing experience.

The level design process requires interaction among the game's design team, including artists, programmers, and the audio team. Attention to detail and a methodical approach to the steps of level design can help to prevent the kind of level design pitfalls that will make your game infamous rather than famous.

Design Practice EXERCISES

1. Pick one of the layouts described in the chapter and, using pencil and paper, create a sketch of a level layout for a hypothetical first-person shooter. (Your instructor will tell you the required number of rooms or locations.) Mark in the layout all of the necessary objects for your level. Mark the starting positions of all the enemies and where the trigger will be or what action will trigger them. If you include such things as traps or doors, mark where they are and what triggers change their state. Mark where supplies such as medical kits and ammunition will be placed. Be sure to consider the path your player will take, remembering that open spaces are good for outdoor exploration and that parallel, linear, network, or combination layouts are good for indoor spaces. Now make one list of all the different kinds of objects that you think you will need and another list of all of your textures. (Do not forget floors, walls, furniture, decorations, weapons, and resources such as ammunition and medical kits.)

2. Choose a game genre that involves avatar travel through the game world and create the background details (not the layout or placement of objects) of a typical level in that genre. In four or five pages, describe what your level looks like and what kinds of things happen in your level. Keep character backgrounds and backstory, if there is one, to minimal details. Instead, focus on the atmosphere, the look and the sounds, the actions the player will take, the events the player will experience, and the motivation(s) that keep the player engaged. Be sure to document what features will set the mood and pace.

3. In four or five pages, explain a tutorial level of an existing game that you have played. How does the player learn the character's moves and capabilities? Remember universal principles and keeping the player interested enough to actually want to play the game. Do the player's skills build on each other, or are they all separate actions? Does the player get to customize his avatar, and if so, how? What, if anything, did the game leave for the player to discover on his own?

4. Choose two levels from two different games, one that has a great level design and one that is, in your opinion, lacking. Take two or three pages for each level and describe the design features that made the great level great and the design pitfalls that detracted from the gameplay or undermined the story of the other level.

5. Go to: www.finitearts.com/400P/400project.htm. Read up on Hal Barwood and Noah Falstein's *400 Project*, the concept of design rules, and the reasons why they are worth breaking or keeping. Choose four or five design principles/rules listed and write a page each on why you think they should be used or broken. Can you come up with any design rules that are not listed? If so, explain why you feel they should be considered for the *400 Project*.

Design Practice QUESTIONS

1. Where is it? What is the time and place?

2. What are the initial conditions of the level? What resources does the player start with? Are there additional resources in the landscape, and if so, which ones, how much, and where?

3. What is the layout of the level? What freedom of movement does the player have within it? In what sequence will he experience challenges, and to what extent can he change that sequence?

4. How will you keep the player informed of his short-term goals? How does he know what to do next?

5. What challenges will the player face there? What actions can the player take?

6. What rewards and punishments are built into the level? How does the player win or lose the level?

7. How do you plan to control and vary the pacing?

8. What events in the level contribute to the story, if any? What narrative events might happen within the level?

9. What is the mood of the level? What is its aesthetic style? What will contribute to the player's experience of these things? Consider music, art, architecture, landscape, weather, ambient sounds and sound effects, and lighting.