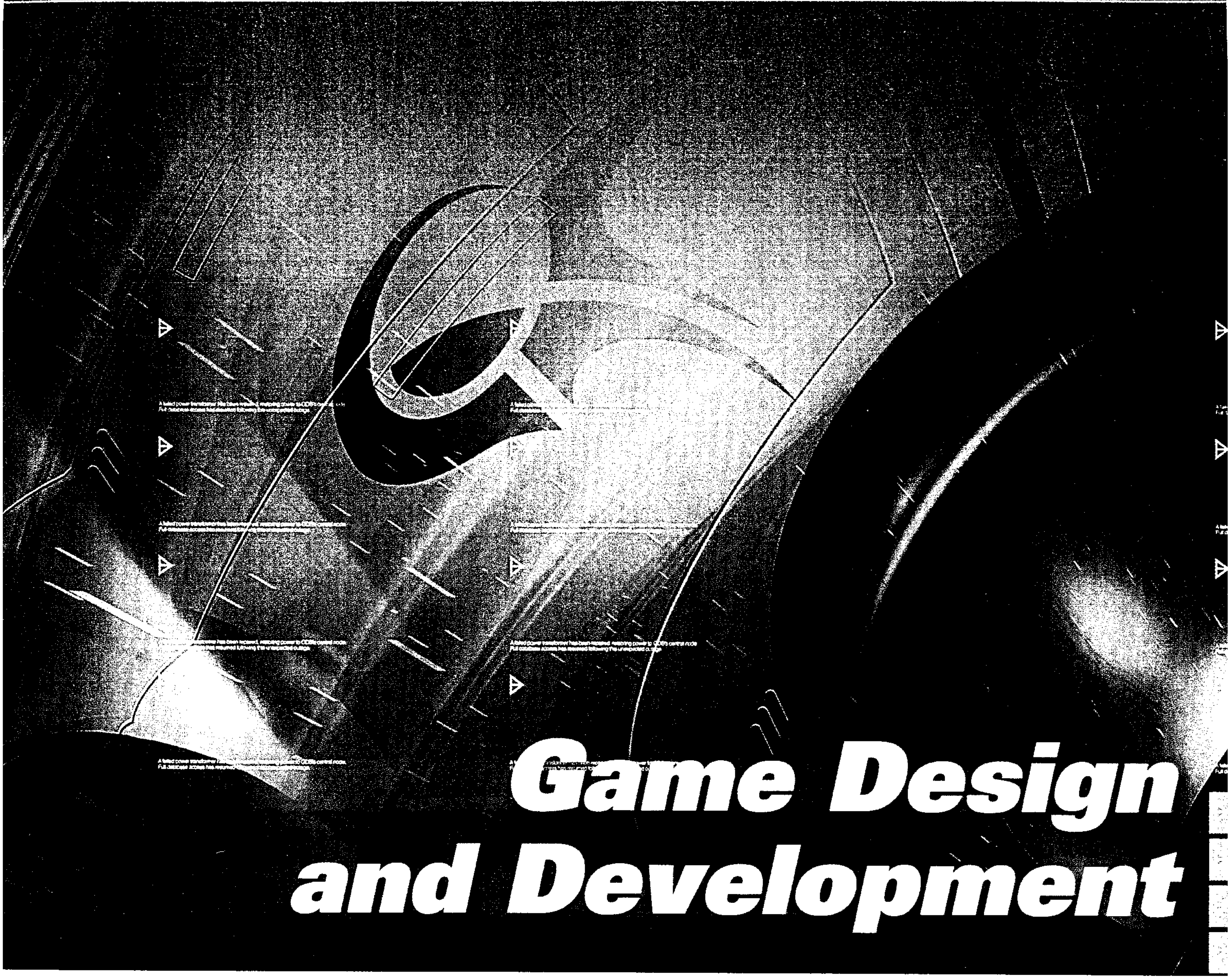
The Level Design Process



Game Design and Development

Fundamentals of Game Design

Ernest Adams • Andrew Rollings

Library of Congress Cataloging-in-Publication Data

Adams, Ernest.
Fundamentals of game design/Ernest Adams and Andrew Rollings.
p. cm.
ISBN 0-13-168747-6 (alk. paper)
1. Computer games—Design. 2. Computer games—Programming. 3. Video games—Design.
I. Rollings, Andrew, 1972- II. Title.
QA76.76.C672A322 2006
794.8'1536—dc22

2006023805

Vice President and Publisher: Natalie E. Anderson
Associate VP/Executive Acquisitions Editor, Print: Stephanie Wall
Executive Acquisitions Editor, Media: Richard Keaveny
Executive Acquisitions Editor: Chris Katsaropoulos
Product Development Manager: Eileen Bien Calabro
Editorial Supervisor: Brian Hoehl
Editorial Assistants: Rebecca Knauer, Kaitlin O'Shaughnessy
Executive Producer: Lisa Strite
Content Development Manager: Cathi Profitko
Senior Media Project Manager: Steve Gagliostro
Project Manager, Media: Alana Meyers
Director of Marketing: Margaret Waples
Senior Marketing Manager: Jason Sakos
Marketing Assistant: Ann Baranov
Senior Sales Associate: Joseph Pascale
Managing Editor: Lynda J. Castillo
Production Project Manager: Lynne Breitfeller
Manufacturing Buyer: Chip Poakeart
Production/Editorial Assistant: Sandra K. Bernales
Design Manager: Maria Lange
Art Director/Interior Design/Cover Design: Blair Brown
Cover Illustration/Photo: Northern Electrics
Composition: Integra
Project Management: BookMasters, Inc.
Cover Printer: RR Donnelley/Harrisonburg
Printer/Binder: RR Donnelley/Harrisonburg

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on appropriate page within text.

10 9 8 7 6 5 4 3 2 1
ISBN 0-13-168747-6

The Level Design Process

Having introduced the general *principles* of level design, we now 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, 2004). They described the 11-stage process by which their company builds levels, which we summarize here. In the following sections, we assume that the development teams consist of game designers, artists, programmers, and sound designers, as well as you: the level designer.

Throughout our 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 will determine 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. We don't have room in this book to teach you the skills needed 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, in this section we assume that you are creating levels for a game that uses a 3D graphics engine to display a three-dimensional game world. If you are making a 2D game, where we refer 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.

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

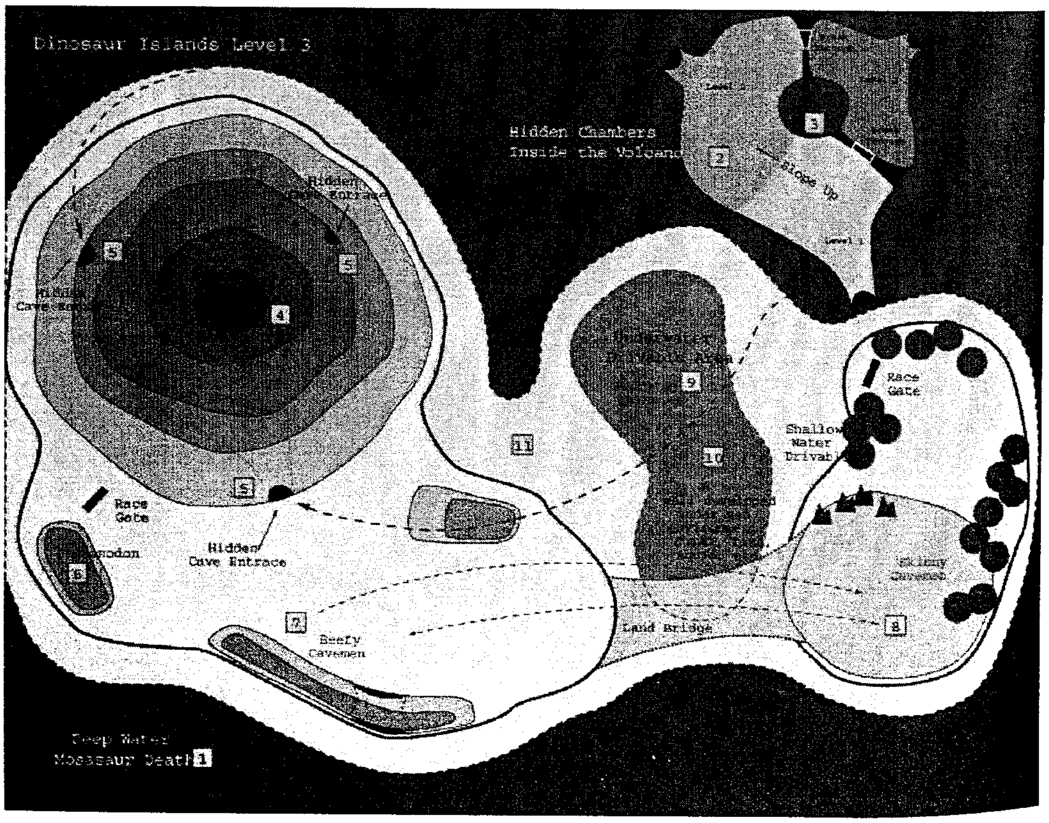


FIGURE 12.7 Rough level sketch for a driving game showing key features. Image courtesy Pseudo Interactive.

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?
- **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 nonplayer characters—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; 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 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 working on the real thing. The issues that the level review should address include:

- **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. If any special code is required that has not yet been written, the programmers must be warned at this point so they can have it ready for the content integration stage. (*Content* refers to the nonsoftware 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 them because they will undoubtedly have questions. It may also be your responsibility to incorporate the content they create into 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, the quality assurance department 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

We end this chapter 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 afterwards. 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 about 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!

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.

Test Your Skills

MULTIPLE CHOICE QUESTIONS

1. The level designer's job includes
   A. choosing the look of the game.
   B. determining how the virtual camera will behave.
   C. deciding how to turn general specifications into specific plans.
   D. none of the above.

2. Which of these is a universal level design principle?
   A. Clearly inform the player of his short-term goals.
   B. Offer opportunities for character growth and player self-expression.
   C. Verisimilitude is vital.
   D. Create many interaction opportunities for creatures in their environments.

3. Which of these is a genre-specific level design principle?
   A. Make early levels of games tutorial levels.
   B. Be clear about risks, rewards, and the consequences of decisions.
   C. Give the player time to think.
   D. Implement multiple difficulty levels.

4. Which of these is a good design choice for a level layout?
   A. Using a parallel layout with hidden entrances that have no distinguishing marks or clues.
   B. Using an open layout for outdoor areas in role-playing games to allow the player to explore freely, then switching to a network layout when indoors.
   C. Using a ring layout with shortcuts that have identical challenges to the ring path itself.
   D. Using a hub-and-spoke layout whenever possible.

5. The network layout
   A. requires the story to tolerate the player's experiencing events in any sequence.
   B. is most often used with games that are races.
   C. is sometimes called a *homeworld*.
   D. creates a game that is said to be *on rails*.

6. Level designers use pencil and paper to work out the sequence of events to take place because
   A. most studios do not want to spend the money on expensive 2D rendering solutions.
   B. roughing things out on paper is fast and the design can be easily changed.
   C. designers have traditionally used pencil and paper.
   D. this initial artwork is often used to market the game to publishers.

7. When it comes to pacing, a good rule of thumb is
   A. vary the pacing, alternating between fast and slow periods.
   B. slow the level down; they tend to be too fast for most players to enjoy.
   C. all levels should start slow and get progressively faster like a ball rolling downhill.
   D. levels in action games should start fast and stay fast.

8. The music in a game is important because it
   A. helps the player orient himself within the game world.
   B. echoes the atmospheric effects.
   C. helps set the pace of the gameplay.
   D. responds to player actions.

9. Tutorial levels should be
   A. the same as normal levels but much easier.
   B. a required part of the game's sequence of levels.
   C. a scripted experience that teaches the player.
   D. all of the above.

10. One of the first things a level designer does once she has been given the handoff from the game designer is to
    A. start modeling everything the game designer has told her to model.
    B. generate a list of desired features to appear in the level, including events, objects, and NPCs.
    C. create thumbnails of textures she may need for the level.
    D. have a meeting with the programmers to discuss special coding requirements she will need.

11. As you plan the gameplay of your level, you will want to consider
    A. the layout and how it affects the gameplay.
    B. how the player can win or lose the level.
    C. landmarks, so that the player can find his way around.
    D. all of the above.

12. Once the level has gone through several reviews and refinements and is considered ready to go into full production, it is said to be "locked down," meaning no new additions may be added, because
    A. most art teams have difficulty following directions.
    B. most game designers feel the need to have complete control over the project.
    C. "locked down" is a military term used for first-person shooters.
    D. otherwise the level could be tweaked forever, stretching the development time and running up the budget.

13. Getting the scope of a level correct is important because
    A. the scope has a direct impact on the marketing budget.
    B. if the scope is too big you may not have the team or resources to complete the level on time or on budget.
    C. players don't like to play epic style games.
    D. it is hard to find good artists.

14. Conceptual non sequiturs are
    A. game features within a level design that do not make sense, even in the understood fantasy of a game world.
    B. bad coding that causes a game to crash.
    C. design features that enhance the gameplay by incorporating a sequence of new concepts.
    D. a product of poor dialog in role-playing games.

15. Showing the player everything at once is a pitfall to avoid because
    A. players hate to see everything at once.
    B. it puts too much of a burden on the art team.
    C. when the game begins to get a little too familiar you cannot introduce anything new to keep the player engaged.
    D. all of the above.

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 open spaces are good for outdoor exploration and 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's 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 QUESTIONS

In designing a typical level in a game, consider the following 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.