Fundamentals of Game Design

Ernest Adams

Andrew Rollings

Upper Saddle River, New Jersey 07458

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.

Copyright © 2007 by Pearson Education, Inc., Upper Saddle River, New Jersey, 07458.
Pearson Prentice Hall. All rights reserved. Printed in the United States of America. This publication is protected by Copyright and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permission(s), write to: Rights and Permissions Department.

Contents of this book are for general information only and should not be considered legal advice.

Pearson Prentice Hall™ is a trademark of Pearson Education, Inc.
Pearson® is a registered trademark of Pearson plc
Prentice Hall® is a registered trademark of Pearson Education, Inc.

Pearson Education LTD.
Pearson Education Singapore, Pte. Ltd
Pearson Education, Canada, Ltd
Pearson Education–Japan
Pearson Education Australia PTY, Limited
Pearson Education North Asia Ltd
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education Malaysia, Pte. Ltd

PEARSON
Prentice
Hall

10 9 8
ISBN 0-13-168747-6

Chapter 9

Gameplay

Chapter Objectives

After reading this chapter and completing the exercises, you will be able to do the following:

- Understand the basic principles that a designer should follow to make games fun.
- Explain how the hierarchy of challenges requires players to complete atomic challenges, sub-missions, and missions to accomplish the ultimate goal of winning the game.
- Define intrinsic skill and stress and discuss how these factors contribute to the difficulty of the game.
- List challenges commonly used in games, including physical coordination challenges, exploration challenges, conflict challenges, and economic challenges, among others.
- Define actions in the context of the game world and describe how actions are selected to allow a player to meet specific challenges or serve other functions in the game.
- Discuss the arguments in favor of and opposed to supplying a saving mechanism and explain the most widely used methods for saving a game.

Introduction

In Chapter 1, "Games and Video Games," we define ***gameplay*** as consisting of the challenges and actions that a game offers: challenges for the player to overcome and actions that let her overcome them. Games also include actions unrelated to gameplay, but the essence of gameplay remains the relationship between the challenges and the actions available to surmount them.

We begin this chapter with a discussion of how we make games fun, setting out some things to be aware of and principles to observe. We then introduce

some important ideas related to gameplay: the hierarchy of challenges and the concepts of skill, stress, and difficulty. The bulk of the chapter consists of a long list of types of challenges that video games offer, with some observations about how to present them, mistakes to avoid, and how to adjust their difficulty. We turn next to actions, listing a number of common types found in games. Finally, we address the questions of if, when, and how to save the game.

Making Games Fun

As we asserted in Chapter 1, the game designer's primary goal is to provide entertainment, and gameplay is the primary means by which games entertain; without gameplay, an activity may be fun, but it is not a game. Entertainment is a richer and more manifold idea than fun is. Nevertheless, most games concentrate on delivering fun rather than offering moving, thought-provoking, or enlightening entertainment. How, then, do we provide fun?

Execution Matters More Than Innovation

Genius is one per cent inspiration, ninety-nine per cent perspiration.

—Thomas Edison (attributed)

We adjust Edison's number slightly but keep the same idea. Ninety-five percent of what makes a game fun has nothing to do with imagination or creativity. The vast majority of things that make a game *not* fun—boring or frustrating or irritating or simply ugly and awkward—result from bad execution rather than a bad idea. The number one reason a game fails to be fun is that it contains elementary errors. A surprising amount of the job of making a game fun, therefore, simply consists of avoiding those things that reduce fun. Our breakdown of the different aspects of game development that contribute to fun follows:

- **50 percent** of making a game fun consists of avoiding errors. Bad programming, bad music and sound, bad art, bad user interfaces, and bad game design all ruin the player's fun. Basic competence will get you up to average.
- **35 percent** of making a game fun comes from tuning and polishing. Not bug fixing—that's in the first 50 percent—but attention to detail, getting everything perfect. Long and dedicated tuning sets a good game apart from a mediocre one.
- **10 percent** of making a game fun comes from imaginative variations on the game's premise, taking the elements of the game and constructing an enjoyable experience out of them. Of this, level designers contribute 9 percent and the game designer contributes 1 percent.

- **4 percent** of making a game fun results from true design innovation: the game's original idea and subsequent creative decisions that you make.
- **1 percent** of making a game fun emerges from an unpredictable, unanalyzable, unnamable quality—call it luck, magic, or stardust. You can't make it happen, so you might as well not worry about it. But when you can feel it's there, be careful about making changes to your design from that point on. Whatever it is, it's fragile.

So innovation by the game designer contributes no more than 5 percent to the fun of the game: 4 percent from the original idea and the 1 percent that you add to the level design process (assuming you don't design the levels yourself). That may make it sound as if there's not a lot of point in game design. But to build a game, someone must design it and design it well. Most game design decisions give little room for innovation, but they're still necessary. A brilliant architect may design a wonderful new building, but it still needs heat and light and plumbing, and in fact the majority of the work required to design that building goes into creating those mundane but essential details. The same is true of game design.

Finding the Fun Factor

There's no formula for making your game fun, nothing that anyone can set out as a reliable pattern and tell you that, if you just slot in a really cool monster here and a fabulously imaginative weapon there, the resulting game is guaranteed to be fun every time. What we can provide is a set of principles to keep in mind as you design and build your game; without them, you risk producing a game that's no fun.

- **Gameplay comes first.** Before all other considerations, create your game to give people fun things to do. A good many games aren't fun because the designers spent more time thinking about their graphics or their story than they did thinking about creating gameplay.
- **Get a feature right or leave it out.** It is far worse to ship a game with a broken feature than to ship a game with a missing feature. Shipping without a feature looks to players like a design decision; a debatable decision, possibly, but at least a deliberate choice. Shipping with a broken feature tells players for certain that your team is incompetent. Broken features destroy fun.
- **Design around the player.** We base everything in this book on player-centric game design, as we explained in detail in Chapter 2, "Design Components and Processes." You must examine *every* decision from the player's point of view. Games that lose sight of the player lose sight of the fun.

- **Know your target audience.** Different groups of players want different things. You don't necessarily have to aim for the mass market, and in fact it's much harder to make a game that appeals broadly than it is to make a game that appeals to a niche market you know well. But whatever group you choose, *know* what they want and what they think is fun, and then provide it.
- **Abstract or automate parts of the simulation that *aren't* fun.** If you model your game on the real world, leave out the parts that aren't fun. But remember your audience: To somebody who just wants a chance to drive a fast car, changing the tires isn't fun, but to a hardcore racing fan, changing the tires *is* fun and a critical part of the experience. If—and only if—you have the time and resources, you may include two modes. Otherwise, choose one market and optimize the fun for the members of that market.
- **Be true to your vision.** If you envision the perfect sailing simulation, don't add powerboat racing as well because you feel that adding features might attract a larger market. (Marketing people are notorious for asking game designers to do this.) Instead, adding powerboats will distract you from your original goal and cut in half the resources you were planning to use to perfect the sailing simulation. Both halves will be inferior to what the whole could have been. You will lose the fun, and without it you won't get the bigger market anyway.
- **Strive for harmony, elegance, and beauty.** A lack of aesthetic perfection doesn't take *all* the fun out of a game, but the absence of these qualities appreciably diminishes it. And a game that is already fun is even *more* fun if it's beautiful to look at, to listen to, and to play with.

The Hierarchy of Challenges

> *When you're up to your __ in alligators, it's hard to remember that your original objective was to drain the swamp.*
>
> —Unattributed

In all but the smallest games, the player faces several challenges at a time, organized in a ***hierarchy of challenges.*** Ultimately, he wants to complete the game. To accomplish that, he must complete the current mission. Completing the mission requires completing a *sub-mission,* of which the current mission probably has several. At the lowest level, he wants to deal with the challenge that immediately faces him: the enemies threatening him at the moment, perhaps, or the locked door for which he needs the right spell. These last we call ***atomic***

challenges (*atomic* in the sense of *indivisible*). Atomic challenges make up sub-missions; sub-missions make up missions; and missions make up the ultimate goal: completing the game.

Figure 9.1 illustrates this idea. It displays the hierarchy of challenges for a (very) small action-adventure game. The entire game consists of three missions or game levels; each level consists of three sub-missions, the last of which pits the player against a level boss. Each sub-mission consists of three atomic challenges, of which the final one (marked in boldface) completes the sub-mission. The gray boxes indicate the challenges the player faces simultaneously at one particular moment in the middle of level 2. At the atomic level, we find him trying to solve a puzzle; doing so will help him to—or allow him to proceed to—destroy the critical object, all of which contributes to succeeding in the Destroy Object sub-mission, which itself makes up a part of winning level 2. Ultimately, he hopes to win the entire game.

Game designer Ben Cousins proposed this notion of a hierarchy but in a slightly different form. For more information on Cousins' original scheme, see *In the Trenches: Cousins' Hierarchy* later in this chapter.

To design your game, you create this hierarchy and decide what challenges the player will face. During play, the player focuses most of her attention on the atomic challenges immediately facing her, but the other, higher-level challenges will always be in the back of her mind. Her awareness of the higher-level challenges creates anticipation that plays an important role both in entertaining her and in guiding her to victory. In the remainder of this section, we discuss how the hierarchy affects the player's experience and what that means for game design.

Informing the Player about Challenges

Video games normally tell the player directly about some challenges, which we call ***explicit challenges,*** and leave him to discover others on his own, which we call ***implicit challenges.*** In general, games give the player explicit instructions about the topmost and bottommost levels of the hierarchy but leave it to him to figure out how to approach the intermediate levels. The topmost level includes the victory condition for the entire game, and games tend to present their overall victory condition explicitly. They may also state an explicit victory condition for each level. Normally, the game's *tutorial levels* teach the player explicitly how to meet the atomic challenges. (For more on tutorial levels, see the discussion on that topic in Chapter 12, "General Principles of Level Design.") Unless you provide completely self-explanatory gameplay *and controls*, you should always include one or more tutorial levels in your game or an explanation of the controls and how to use them to meet atomic challenges.

KEY POINT

Make the victory condition and the atomic challenges explicit. Be sure the player always has some overarching goal in mind toward which she works. Never leave her without a reason for continuing to play.

You should always tell the player about the victory condition or she won't know what she's trying to accomplish. You don't have to tell the complete truth, however. In storytelling games, you'll usually want to keep the outcome a surprise. Many stories start by telling the player one thing, but plot

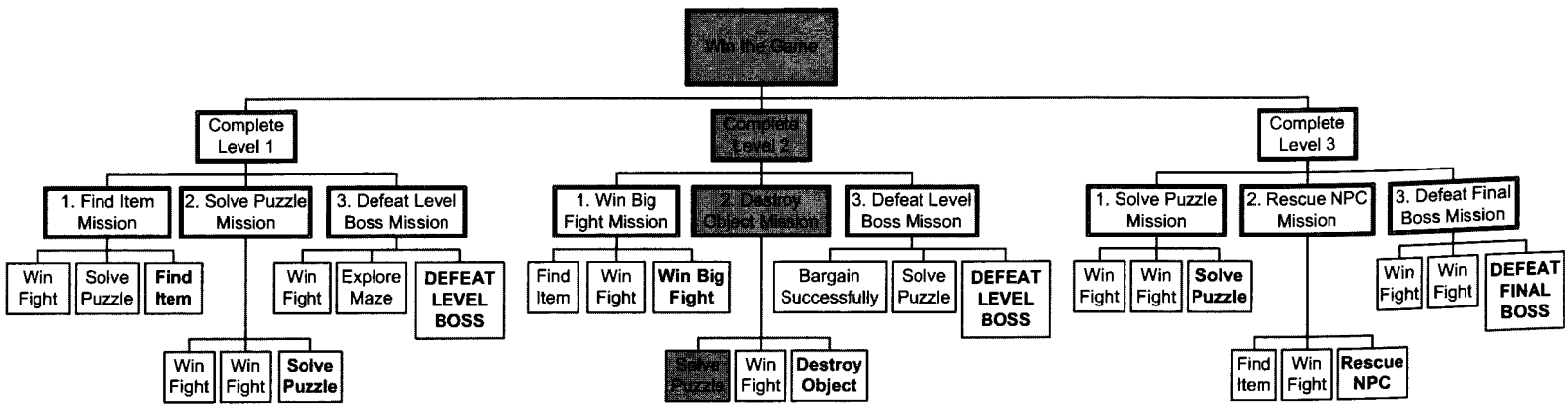

FIGURE 9.1 A hierarchy of challenges for a small action-adventure.

twists along the way deepen and complicate matters. She may change or meet a goal only to find it replaced by another, more important goal. Detective stories, in particular, are famous for this. (Don't do it more than three or four times in any one story, though, or the player will start to be irritated about being repeatedly lied to. Detectives are also famous for getting irritated about being lied to.) Be sure that the player gets whatever information she needs to *think* she clearly knows the victory condition so she's never left without any motivation.

The Intermediate Challenges

Most designs leave intermediate-level challenges implicit. If you give the player nothing to do except follow explicit instructions, it doesn't feel like a game; it feels like a test. Part of the player's fun lies in figuring out—whether through exploration, through events in the story, or by observing the game's internal economy—what he's supposed to do. Armed with the knowledge of both the victory condition at the top and the right way to meet the atomic challenges at the bottom, he has the tools to figure out the intermediate challenges—if you have constructed them coherently. (See *Avoid Conceptual Non Sequiturs* in Chapter 12, "General Principles of Level Design," for an example of how *not* to construct an intermediate-level challenge.)

For a good many games, overcoming the intermediate-level challenges requires only that the player meet all the lowest-level ones in sequence. That's how most action games work, and what Figure 9.1 illustrates. If the player beats all the enemies and gets past all the obstacles, he finishes the level. If he finishes all the levels, he wins the game.

In more complex games, the player may have a choice of ways to approach an intermediate-level challenge. Suppose the explicit top-level challenge—the victory condition—in a war game consists of defeating all the enemy units, and the atomic challenge consists of destroying one enemy unit. The simple and obvious strategy would apparently be to destroy all the enemy units one by one, but the player isn't likely to get that chance. Most war games include a production system for generating new units, so even if the player can kill off enemy units one by one, his opponent can probably produce new ones faster than he can destroy them. Disrupting the enemy's production system is often an effective strategy, while protecting his own production system ensures that he can eventually overwhelm the enemy with superior numbers. Neither the specification of the victory condition nor of the atomic challenges explicitly includes the intermediate-level challenge of *disrupt the enemy's production system,* and protecting his own production system doesn't destroy enemy units at all. The player must figure out what he should do by observation and deduction and, by planning and experimentation, find ways to accomplish his goals. The observation, deduction, planning, and experimentation all add to the fun.

You construct these implicit, intermediate challenges for the player to figure out. The conventions of the genre you've chosen will guide you, but keep in mind a couple of points:

- **The most interesting games offer multiple ways to win.** To give your player a richer experience, design your game so that he can win in a variety of ways—that is, so that meeting *different* intermediate challenges will still get him to victory. Different strategies may be better or worse, but if you only permit one right way to win, you don't reward the player's lateral thinking skills. The game becomes an exercise in reading the designer's mind.
- **Recognize and reward victory no matter how the player achieves it.** Players will think of things to try that you might not have anticipated; even if you've given them multiple ways to win, they may find another way entirely. If the player achieves the victory condition, even in a completely unexpected way, he deserves credit for it. Don't test to see if he got there in one of the ways that you intended; just test to see if he got there.

Figure 9.2 illustrates the idea of offering multiple ways to win a war game. The victory condition is to capture the flag. The hierarchy is organized as before, except that the dotted lines indicate a *choice of possible approaches* rather than a *sequence of required sub-missions* as in Figure 9.1. As before, the gray boxes indicate the challenges in the player's mind at one particular moment—in this case, assuming that he chose to use a stealth approach and sent units out to scout.

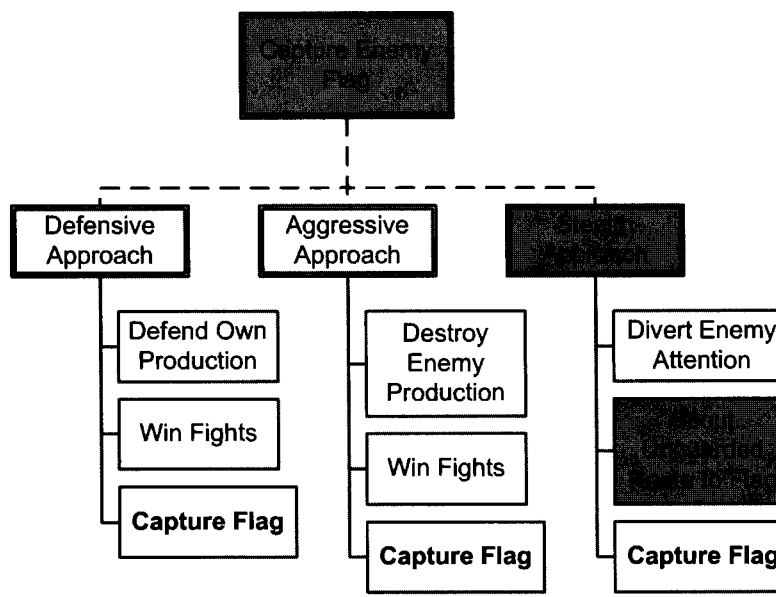

FIGURE 9.2 A hierarchy of challenges with multiple options.

Simultaneous Atomic Challenges

Although the player simultaneously faces several challenges in the hierarchy the farther up the hierarchy a challenge lies, the less of her attention that challenge demands on a moment-by-moment basis. But games can also present several atomic challenges at once. These divide the player's attention. If she can deal with them one at a time at her own pace (as in a turn-based game), then they're not really different from sequential challenges, but if she has to surmount them all in a limited amount of time, then adding simultaneous challenges makes the game more stressful. (We discuss stress in *Skill, Stress, and Absolute Difficulty* later in the chapter.)

An early and still common way of creating simultaneous atomic challenges, typical of side-scrolling shooter games, consisted of bombarding the player with enemies. Each represents a significant risk, and the player must defeat each one while fending off the others. A player who works quickly can generally defeat these added enemies one at a time while keeping the others at bay.

Other games present more complex and interrelated simultaneous challenges. In its default mode, *SimCity* imposes no victory condition; its highest-level challenge is to achieve economic growth so the player can expand his city. (Expanding the city itself isn't a challenge, just a series of choices available so long as the player brings in enough money to keep going.) The player won't attain economic growth unless he can provide a balanced supply of services to the city. The city needs police protection *and* power *and* hospitals *and* water and so on, all at the same time; each represents an atomic challenge, and all must be met simultaneously. The complex juggling of competing needs requires regular attention and frequent action. Furthermore, unlike fighting enemies, the player can never finish balancing the services; the juggling act never stops.

It's part of your job to design the hierarchy of challenges and decide how many of them the player will face at once: both vertically up the hierarchy and at the bottom of the hierarchy. The more simultaneous atomic challenges he faces under time pressure, the more stressful the game will be. The more different levels of challenge he has to think about at once—especially if he can't simply achieve the higher ones by addressing the lower ones in sequence—the more complex and mentally challenging the game will be.

Cousins' Hierarchy

Ben Cousins, in an article for *Develop* magazine, suggested thinking of gameplay as a hierarchy (Cousins, 2004). We have adopted his idea for this book but modified it somewhat and used different terminology. Cousins referred, for example, to *atoms of interaction* rather than *atomic challenges*.

CONTINUED ON NEXT PAGE

Cousins studied the game *Super Mario Sunshine* by making a video recording of the screen while he played, then examining the results in a video editor, which enabled him to identify the atoms of interaction in the game. By thinking about what he was trying to accomplish at each moment as he played, he found that he could organize the gameplay into a five-level hierarchy with "Complete the whole game" as the topmost interaction, "Complete the current game level" as the second level of the hierarchy, and so on down to individual atoms of interaction at the bottom level.

Cousins studied an action game; action games typically require players to use specific low-level actions to meet low-level challenges (to get across the chasm, jump). In other genres, however, there isn't a one-to-one mapping between challenges and actions even at the atomic level. Some challenges may be overcome by several different kinds of actions; overcoming others requires complex sequences of actions. Accordingly, we have not included actions in our hierarchy, making ours a hierarchy of challenges only.

We recommend Cousins' technique of analyzing the way that games organize their gameplay by examining them second-by-second in a video editor.

Skill, Stress, and Absolute Difficulty

Most of our discussion about gameplay difficulty we reserve for Chapter 11, "Game Balancing," but we introduce some important distinctions here because the terms involved will come up in our discussion of different types of gameplay later in this chapter. While these are not industry standard terms, we find them useful.

In this chapter, we are concerned with controlling the absolute difficulty of the challenges that you will present to the player. Two different factors determine the absolute difficulty of a challenge. We define these as *intrinsic skill required* and *stress*. In Chapter 11, we will address additional factors that affect the player's perceptions about how easy or hard the game is.

Intrinsic Skill

We define the ***intrinsic skill required*** by a challenge as the level of skill needed to surmount the challenge *if you give the player an unlimited amount of time in*

which to do it. The intrinsic skill required for a challenge may be computed from the conditions of the challenge but leaving out any element of time pressure. How you measure the skill level of a challenge varies with the type of challenge and can involve physical tasks, mental tasks, or both. Consider three examples:

- An archer aiming at a target requires a certain level of skill to hit the target. It takes more skill to hit the target if you move the target farther away or make it smaller. The archer gets an unlimited amount of time to aim. Even if he takes more time, it does not change the skill required.
- Sudoku puzzles printed in the newspaper often include a rating that indicates whether they are easy or hard to solve. The player may take as long as he wants to solve the puzzle, so the rating reflects an intrinsic quality of the puzzle—how many clues the player gets—rather than the effect of a time limit.
- A trivia game requires the player to know certain factual knowledge. Some questions are about familiar facts and some are about obscure facts. The skill level required by a question doesn't change if you give the player more time to answer.

Some challenges *must* include time pressure by the way they are defined—a test of the player's reaction time, for example. A test of pure reaction time (hit the button when the light comes on) requires no intrinsic skill at all. The carnival game *Whack-a-Mole* is a real-world example.

Stress

If a challenge includes time pressure, a new factor comes into play, *stress*. ***Stress*** measures how a player perceives the effect of time pressure on his ability to meet a challenge requiring a given level of intrinsic skill. The shorter the time limit, the more stressful the situation. Succeeding in a stressful game requires both quick reflexes and a quick mind. The challenges of *Tetris* do not require a great deal of intrinsic skill—if the player had plenty of time to think about the task, it would be easy to keep the blocks from piling up—but we class *Tetris* as stressful because the player must complete the task under time pressure. Golf, on the other hand, demands skill without being stressful—at least, in the sense in which we use the term. It would be considerably more stressful if the rules imposed more time pressure.

Games often create physical stress on the player's body. Time pressure requires players to use their eyes and hands more quickly; it makes them stiffen

their muscles, and it raises their heart rates and adrenaline levels. Many people love this sensation, but you should modulate the pacing of your game to give them time to rest. We discuss this at length in *Vary the Pacing* in Chapter 12, "General Principles of Level Design."

Absolute Difficulty

KEY POINT

The *absolute difficulty* of a challenge consists of a combination of the *intrinsic skill required* to meet the challenge without time pressure and the *stress* added by time pressure.

Absolute difficulty refers to intrinsic skill required and stressfulness put together. When a game offers multiple difficulty levels, the easy mode both demands less skill and exerts less stress than the hard mode. Some players like a challenge that demands a lot of skill but can't tolerate much stress. If they know they have plenty of time to prepare for a challenge, they're perfectly happy for the challenge to require great skill. Others thrive on stress but don't have much skill. Simple, high-speed games like *Tetris* and *Collapse!* suit them best. See Figure 9.3 for a graph showing the relationship of intrinsic skill and stress in various games or tasks. The higher the task ranks on both scales, the greater its difficulty.

When you're deciding how difficult you want your game to be, think about both skill levels and stress, and keep your target audience in mind. Teenagers and young adults handle stress better than either children or older adults because teenagers and young adults have the best vision and motor skills. When you allow the player to set a difficulty level for the game, try to preserve an inverse relationship between skill level and stress at that particular level of difficulty. If a challenge requires more skill, give the player longer to perform it, and vice versa.

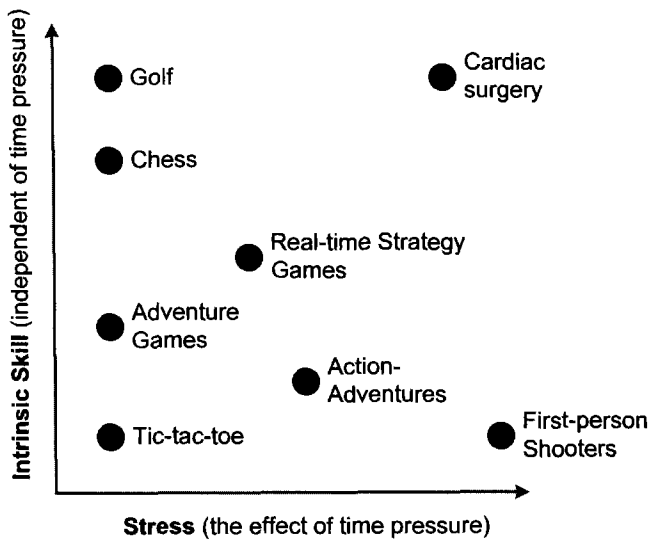

FIGURE 9.3 Intrinsic skill required versus stress in different tasks.

Commonly Used Challenges

This section presents commonly used challenges you should understand. Because games can set a top-of-the-hierarchy challenge of just about anything—take on the role of Jason and win the golden fleece, use your zombie army to drive the werewolves out of your ancestral home, capture a giant squid from the ocean deeps, win a rodeo competition—we can't discuss all the possible top-of-the-hierarchy goals. This discussion focuses on lower-level challenges, many of which could be classified as atomic challenges.

This list may help you to think about the kinds of traditional challenges you'd like to include in your game, but you're free to design any type of challenge you can imagine if you can make it workable. Players always appreciate innovative challenges; a beautiful setting or interesting overall concept only gets you so far. Gameplay is in the challenges.

Throughout this section, we will include observations from time to time about ways to make a particular type of challenge easier or harder. Most of these comments concentrate on making them easier because novice game developers frequently make their games too hard without realizing it.

Physical Coordination Challenges

Physical coordination challenges test a player's physical abilities, most commonly hand-eye coordination. One of the earliest coin-op video games, *Pong,* required only this one skill to win. Physical coordination challenges remain a basic component of arcade gaming and a significant part of most video games to this day. They fall into several subcategories, which we discuss in the following sections.

The absolute difficulty of a physical coordination challenge most frequently relates to the amount of time pressure the player is under; to make such a challenge easier, simply give the player more time. We will address exceptions, where they occur, with each subcategory.

Speed and Reaction Time Speed challenges test the player's ability to make rapid inputs on the controls, and reaction time challenges test his ability to react quickly to events. Both of these usually appear in combination with other types of challenges, most often other coordination challenges. You can expect to find speed and reaction time challenges in platform games, shooters, and fast puzzle games such as *Tetris*. From the frenetic button-banging of Konami's 1983 arcade game *Track & Field* (two buttons controlled an athlete's legs; the player pressed them alternately to make the athlete run) to the modern-day frenzy of *Quake 4*, speed and reaction-time challenges perennially please those who have the reflexes for them.

Accuracy and Precision Steering and shooting comprise the majority of tests of accuracy or precision, though you can devise many more. Steering includes navigating characters as well as vehicles. Usually found in action and action-adventure games, sports games, and vehicle simulations, accuracy and precision challenges increasingly feature in role-playing games, such as *Fable,* which include a combat element.

Accuracy challenges need not take place within time limits; in a sport such as archery, athletes may take as long as they want to line up a shot but still face a considerable challenge. To make an accuracy challenge easier or harder, adjust the degree to which the physics engine in the game forgives errors in the inputs. For example, the player of an archery game ordinarily needs to position the joystick or mouse within a particular range of values to hit the bull's-eye; you can make the game easier by widening that range.

Intuitive Understanding of Physics Vehicle simulations require more than just an ability to steer a vehicle; they also require an intuitive understanding of the physics of the game world. Players must learn, usually through experience, a car's braking distance, acceleration rate, at what rate it may take a turn without sliding off the road, and so on. Learning and internalizing these features of the game world and the vehicle constitute another challenge. Games such as pool and darts also require the player to develop an intuitive understanding of physics. We include these under physical coordination challenges because the player tends to develop a visceral rather than an intellectual understanding of these aspects of the game world (darts players don't need to know calculus), which finds its expression in successful physical coordination.

You can help the player develop an intuitive understanding of the game's physics. First, make sure the physics remains consistent. The physics engine must be reliable and produce predictable results. Programmers handle the physics engine, but you should also keep this in mind. Second, the simpler the physical model of the world, the easier it will be for the player to develop that intuitive understanding. Sports games often simplify their physics to help the player. For example, many sports games don't implement the physical property of inertia for an athlete running under the player's control because the player wants to be able to turn his player instantaneously and will find it harder to get used to the game if he cannot. Flight simulators, too, model the physics of flight with greater and lesser degrees of accuracy depending on how easy the designer wants to make it for the player to understand how the airplane behaves.

Timing and Rhythm Side-scrolling action games rely heavily on timing challenges, in which the player learns to dodge swinging blades and attack predictable enemies. Rhythm challenges, tests of the player's ability to press the right button at the right time, feature in dance games such as *Dance Dance Revolution* and other music-based games such as *Donkey Konga* and *Guitar Hero.*

The popularity of rhythm-based games resulted in a significant aftermarket in specialty input devices such as dance mats and electronic conga drums.

Combination Moves Many fighting games require complex sequences of joystick moves and controller-button presses that, once mastered, allow the player's avatar to perform some especially powerful feat. (See *Fighting Games* in Chapter 13, "Action Games.") Executing a combo move requires speed, timing, and a good memory, too: The player has to remember the button sequence and produce it perfectly at just the right time. You can make combination moves easier by shortening them, requiring fewer presses.

Logic and Mathematical Challenges

Logical and mathematical reasoning has been part of gameplay since the dawn of human history. Logic provides the basis for strategic thinking in any turn-based game of perfect information and many other games in which the player can make precise deductions from reliable data. In this section, we confine ourselves to logic *puzzles,* and we deal with strategic thinking in *Strategy,* later in the chapter.

Mathematics underlies all games in which chance plays a role or the player does *not* have reliable data and so must reason from probabilities. Such games present explicit mathematical challenges to the player: If he doesn't compute the odds when playing poker, or at least know the odds and reason correctly given what he knows, he's much more likely to lose.

In the broadest sense, any game that includes numeric relationships offers a mathematical challenge, because the player must learn how those relationships work. Much of the time, games present mathematical challenges implicitly, couching numeric relationships in other terms: physics, strategy, or economics. (For an example from strategy, see our discussion of Lanchester's laws in Chapter 14, "Strategy Games.") We deal with implicit mathematical challenges in other sections of this chapter.

Formal Logic Puzzles A puzzle is a mental challenge with at least one specific solution. *Formal logic* means classic deductive logic in which the definition of the puzzle contains, or explains, everything the player needs to know to solve the puzzle. A formal logic puzzle can be solved by reasoning power alone. It shouldn't require any outside knowledge. Many other types of puzzles require logic too, but they also expect the player to supply some additional information.

A logic puzzle typically presents the player with a collection of objects related in ways that are consistent but not directly obvious. To solve the puzzle, the player must put the objects into a specified configuration. The player manipulates the objects and receives feedback about their relationships, which he eventually comes to understand by observation and deduction. Rubik's Cube,

a classic logic puzzle with a simple mechanism, consists of so many cubes which move in ways so intricately interrelated that it is quite difficult to solve.

Adventure games often present logic puzzles as combination locks or other machinery that the player must learn to manipulate because those devices make sense in the fantasy world that the game presents. Other puzzle-based games don't try to be realistic but concentrate on offering an interesting variety of challenges.

To adjust the difficulty of a logic challenge, raise or lower the number of objects to be manipulated and the number of possible ways in which the player can manipulate them. A Rubik's Cube with four faces per side (a $2 \times 2 \times 2$ cube) instead of nine ($3 \times 3 \times 3$) would be far easier to solve.

Players normally get all the time they need to solve puzzles. Because different people bring differing amounts of brainpower to the task, requiring players to solve a puzzle within a time limit might make the game impossible for some. Exceptions to this rule can sometimes succeed; *ChuChu Rocket* offers both a time-limited multiplayer mode and an untimed mode.

A few games do not make the correct solution clear at the outset of the puzzle. The player not only has to understand how the puzzle works but also guess at the solution she must try to achieve. We consider this bad game design: It forces the player to solve the puzzle by trial and error alone because there's no way to tell when she's on the right track. In order to open the stone sarcophagus at the end of *Infidel,* the player had to find the one correct combination of objects out of 24 possible combinations. The game gave no hints about which combination opened the box; the player simply had to try them all.

Avoid Trial-and-Error Solutions

Solving most logic puzzles requires a certain amount of experimentation, but the player must be able to make deductions from his experiments. Do not make puzzles that can *only* be solved by trial and error.

Mathematical Challenges Games don't usually test the player's mathematical abilities explicitly but often do require the player to reason about probabilities. Many games include an element of chance or require the player to make educated guesses about situations of which he has only an imperfect knowledge. Consider *Microsoft Hearts* (see Figure 9.4) as an example of a game giving imperfect information. Initially, the player does not know what cards the other players hold, but a skilled player can work out to a reasonable degree of certainty what those cards must be by using the information revealed as cards are passed and played during the game.

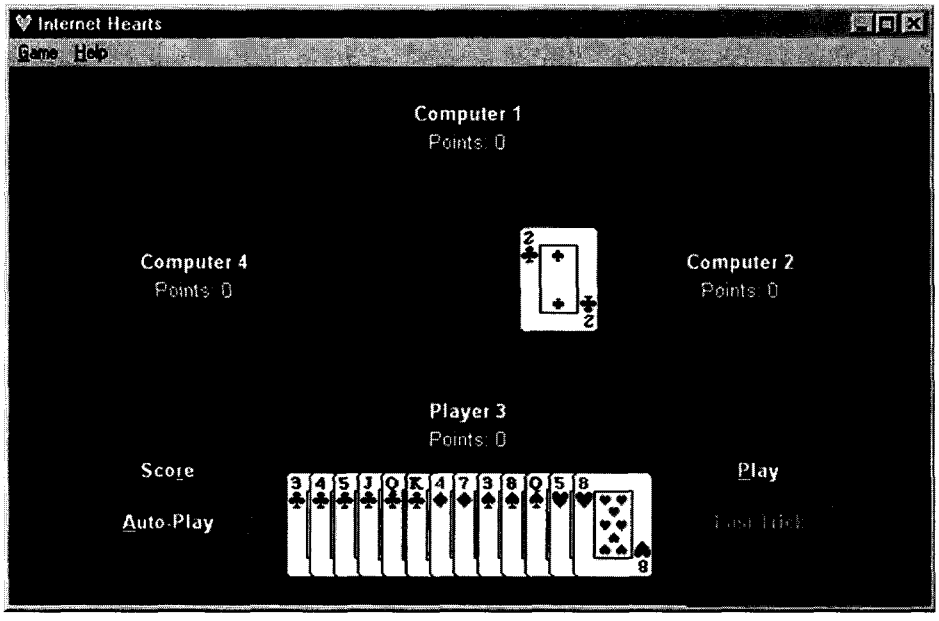

FIGURE 9.4 *Microsoft Hearts.*

Races and Time Pressure

In a *race,* the player attempts to accomplish something before someone else does, whether that involves a physical race through space or a race to create a structure, to accumulate something, or to do practically anything as long as the game can distinguish which player finishes first. Normally we think of races as peaceful, involving competition without conflict, but races can be combined with fighting and many other kinds of challenges.

Time pressure discourages careful strategic thought and instead encourages direct, brute-force solutions. With only 15 seconds to get through a host of enemies and disarm a bomb, the player won't stop to pick off enemy units one by one with sniping shots; he's going to mow them down and charge through the gap, even if that means taking a lot of damage.

Time pressure increases the stress on a player and changes the feeling of the gameplay considerably, sometimes for better and sometimes for worse. In something like a car racing game, time pressure is an essential part of the experience, but we urge caution in adding time pressure to challenges that aren't ordinarily based on time constraints. You will deter some players entirely and make the challenge more difficult in any case. To keep the absolute difficulty level constant, whenever you increase the time pressure on a player, you should also reduce the amount of intrinsic skill required.

Factual Knowledge Challenges

Direct tests of the player's knowledge of factual information generally occur only in trivia and quiz games. In any other type of game, you must either present all the factual knowledge required to win the game somewhere *in* the game (or the manual) or make it clear in the game's marketing materials that players will need some factual knowledge. It's not fair to require the player to come up with some obscure fact from the real world in order to make progress through the game; doing so also detracts from full immersion.

Note the difference between factual knowledge challenges and conceptual reasoning challenges (discussed in *Conceptual Reasoning and Lateral Thinking Puzzles,* later in the chapter). If an adventure game features a puzzle the player can't solve without knowing that helium balloons rise or that metal objects conduct electricity, we would characterize such a puzzle as a conceptual reasoning challenge, not a factual knowledge challenge.

Trivial Pursuit, the popular board game that tests players' factual knowledge in several different domains, also runs as a video game on a variety of platforms.

Make It Clear When Factual Knowledge Is Required

If your game requires factual knowledge from outside the game world to win, you must make this clear to the player in advance.

Memory Challenges

Memory challenges test the player's ability to recall things that she has seen or heard in the game. Adventure games and role-playing games often make use of memory challenges. Players can defeat memory challenges by taking notes, so you may want to impose a limit on the length of time you give them to memorize material they must recall. To make a memory challenge easier, give them longer to memorize it and ask that they recall it soon after memorizing it rather than much later.

Memory challenges often form one component of exploration challenges. In Raven's *Star Trek Voyager: Elite Force,* for example, the player must remember the layout of complex tunnels onboard the Borg cube.

Pattern Recognition Challenges

Pattern recognition challenges test the player's ability to spot visible or audible patterns or patterns of change and behavior. One of the most common instances of a pattern recognition challenge crops up in action games when a large number of identical enemies, each of which behaves in a predictable way, confronts

the player. The player can try alternative strategies until he finds one effective against that enemy, then use that strategy to vanquish any number of enemies that attack using the same pattern. To make things harder and more interesting, the boss enemy at the end of a level usually has a different and more complex pattern of behavior from the smaller enemies that preceded it.

Visual clues often figure in pattern recognition challenges. In *Doom,* secret doorways could be found by searching for areas of wall that looked slightly different from the norm. *Dungeon Master* required the player to use pattern recognition skills to decipher the complex systems of runes governing spells and spell casting.

To make pattern recognition challenges easier, make the patterns shorter, simpler, and more obvious. To make them harder, make the patterns longer, more intricate, and more subtle.

Exploration Challenges

Exploration is often its own reward. Players enjoy moving into new areas and seeing new things, but exploration cannot be free of challenge or it becomes merely sightseeing. Design obstacles that make the players earn their freedom to explore.

Spatial Awareness Challenges The most basic form of exploration challenge simply requires the player to learn her way around an unfamiliar and complicated space. In the old text adventure games, this was particularly difficult because the games lacked visual cues, but even modern 3D environments can be made so tangled that they're hard to navigate. Unfamiliar architecture also challenges the player; navigation is easier when things look the way she expects them to look. *Descent* required the player to move around a warren of similar corridors inside an asteroid and complicated matters further by setting the environment in zero gravity, so the player perceived no obvious *up* or *down* to help orientation.

To make spatial awareness challenges easier, give the player a map that always shows his location precisely within the game world. If you want to give the player a map but make it slightly more difficult, give him a map of the game world that doesn't include his location, so he has to work it out from landmarks.

Locked Doors We use *locked door* as a generic term for any obstacle that prevents the player from proceeding through the game until he learns the trick for disabling it. A sheet of ice covering a cave entrance that melts if you build a fire constitutes a locked door for game design purposes. Assuming, for discussion only, you want an actual door, you can require that the player find a key elsewhere and bring it back, find and manipulate a hidden control that opens the door, solve a puzzle built into the door, discover a magic word that opens the door, defeat the doorkeeper in a test of skill, or perform some other task—just make sure you offer an interesting and fresh challenge.

Avoid using an unmarked switch far from the door. *Doom* featured these, and they weren't much fun. Arriving at a locked door and seeing no means of opening it or any clue, the player had to search the entire world pressing unmarked switches at random, returning to see whether one of the switches had opened the door. Worse yet, in a few cases the switch *did* open the door, but only for a little while. If the player didn't get back to the door in time, he found it locked again and assumed that switch must not have been the right one.

Traps A *trap* is a device that harms the player's avatar when triggered—possibly killing her or causing damage—and, in any case, discouraging her from going that way or using that move again. Similar to a locked door with higher stakes, a trap poses an actual threat. Traps can take a variety of forms:

- Some fire once and then are harmless.
- Others fire and require a certain rearming time before they can fire again.
- Still others respond to particular conditions but not to others, like a metal detector at an airport, and the player must learn what triggers the trap and how to avoid triggering it.

A player may simply withstand some traps that don't do too much damage; he may disarm or circumvent other traps. A trap the player can find only by falling into it is really just the designer's way of slowing the player down; if you make these, don't make many of them because the player can only find them by trial and error and they become frustrating after a while. For players, the real fun comes in outwitting traps: finding and disabling them without getting caught. This gives players a pleasurable feeling of having outfoxed you.

Mazes and Illogical Spaces A maze is an area in which every place looks alike, or mostly alike; to get out, the player must discover how the rooms or passages relate to each other, usually by wandering around. Good designers implement mazes as logic or pattern-recognition puzzles, in which the player can deduce the organization of the mazes from clues found in the rooms. Poor mazes offer no clues and make the player find the way out by trial and error. Mazes are now considered rather old-fashioned and difficult to justify in the context of a story, but they can still be fun to solve if you make them truly clever and attractive.

In illogical spaces, areas do not relate to each other in a way that the player might reasonably expect. In text adventures, a player often found that going north from area A took you to area B, but going south from area B did not take you back to area A. Illogical spaces require the player to keep a map, because he can't rely on his common sense to learn his way around. Now also considered outdated, and more difficult to implement with today's 3D engines, illogical space challenges still crop up from time to time. If you use them, do so sparingly,

and only if you can explain their presence: "Beware! There is a rip in the fabric of space-time!"

Teleporters Teleporters superseded illogical spaces in the game designer's toolkit. A *teleporter* is any mechanism that suddenly transports the player from where she is to someplace else, often without warning if the designer created no visual representation for the teleporter device. Several hidden teleporters in an area can make exploration difficult. Teleporters can further complicate matters by not always working the same way, teleporting the player to one place the first time they are used but to some other place the second time, and so on. You can also use one-way teleporters if you want to leave the player with no way to get back.

To make the exploration challenge created by teleporters easier, make the teleporter predictable and reversible, so the player can return at will to where she came from. (A good many games include teleporters not as a challenge but as a visible and optional feature to let the player jump across large distances that she has already explored.)

Conflict

We define a ***conflict challenge*** as one requiring the direct opposition of forces, some of which are under player control. If one player must beat the others by opposing or impeding them directly, the challenge qualifies as conflict, even without combat or violence. Checkers has no bloodshed but still presents conflict challenges. Classic activities to overcome conflict challenges include taking away another player's resources and impeding another player's ability to act.

FYI Conflict Challenges Versus Conflict of Interest

Formal ***game theory*** is a field of mathematics that studies situations that contain a ***conflict of interest.*** By that definition, any game in which players are rivals for victory contains conflict. However, in pole-vaulting, darts, and many other games, that opposition applies only at the top of the challenge hierarchy. At lower levels, the players do not (and sometimes are forbidden to) impede each other directly. Even in *Monopoly,* the rules provide no means by which players may choose to target each other for hostile action. Such games may contain a conflict of interest but no conflict challenges. The players must achieve their top-level goal not through the direct opposition of forces but through vaulting over the higher bar, throwing darts more accurately, or whatever other atomic challenges the game specifies.

The asymmetric board game Fox and Geese (introduced in Chapter 1, "Games and Video Games") gives the two players different conflict challenges. The fox tries to eat the geese by jumping over them on the board (taking away the other player's resources). The geese try to trap the fox while moving in configurations that prevent the fox from jumping over them (preventing the enemy from acting).

Create interesting conflict challenges by varying such factors as:

- The scale of the action (from individuals to whole armies)
- The speed at which the conflict takes place (from turn-based, allowing the players all the time they want, to frenetic activity as in action games)
- The complexity of the victory conditions (from simple survival to complex missions with goals and subgoals)

Many action games focus on the immediate, exciting, visceral excitement of personal conflict. The player generally controls an avatar that battles directly against one or more opponents, often at high speeds.

Conflict challenges can be broken down into strategy, tactics, logistics, and other components we discuss here in turn.

Strategy *Strategy* means planning, including taking advantage of your situation and resources, anticipating your opponent's moves, and knowing and minimizing your weaknesses. A strategic challenge requires the player to carefully consider the game (a process called situational analysis) and devise a plan of action. In a turn-based game of *perfect information* (one that contains no element of chance or hidden information), players may use *pure strategy* to choose their moves by analyzing possible future states of the game. Chess is a classic game of perfect information. (Note that in formal game theory, pure strategy has a special meaning, but we use the term *pure strategy* in an informal sense to distinguish it from applied strategy.)

Succeeding in a game of pure strategy requires a talent for systematic reasoning that relatively few people possess in a high degree. Computer game developers usually aim to attract a broad audience, so few of them offer these kinds of challenges. Instead, they hide information from the player and include elements of chance, making situational analysis to some extent a matter of guesswork and of weighing probabilities rather than a matter of logic. Such games call for applied strategy. Real-time strategy (RTS) games normally require applied strategy and offer economic and exploration challenges as well, making RTS games accessible to players with less skill at logic and providing other ways to win besides strategy alone.

Tactics *Tactics* involve putting a plan into execution, accomplishing the goals that strategy calls for. Tactics also require responding to unexpected

events or conditions: new information or bad luck. Even chess involves tactics—responding to opponent's moves requires tactical skill.

You can design a purely tactical game with no strategy. A small-squad combat game in which the soldiers continually move into unknown territory contains no opportunities for strategy—a player can't plan if she doesn't know where she's going or what she's up against—but contains many opportunities for tactics, such as keeping soldiers covered, taking advantage of their particular skills, and so on.

Logistics The business of supporting troops in the field and bringing fresh troops to the front lines is called logistics. Most war games don't bother with logistical challenges such as transporting food and fuel to where the troops can use them; players tend to find combat entertaining but find logistics a boring distraction from the combat.

Modern RTS games routinely include one important logistical challenge: weapons production. Unlike war board games, in which the player often starts with a fixed number of troops, RTSs require the player to produce weapons and to research new types of weapons using a limited amount of raw material. The production facilities themselves must be constructed and then defended. Adding this new logistical challenge to what was formerly a purely combat-oriented genre changed the face of war gaming. (See Chapter 14, "Strategy Games.")

In role-playing games, the limited size of the characters' inventories presents another logistical challenge, requiring players to decide what to carry and what to leave behind. Equipping and balancing a party of heterogeneous characters with all that they need to face a dangerous adventure occupies a significant amount of the player's time.

Survival and Reduction of Enemy Forces The fundamental challenge in any game based on conflict is survival. The player must preserve the effective playing time—the lives—of his units, or he cannot achieve the victory condition. In a few games, survival itself constitutes the victory condition regardless of other achievements, but in most, survival is necessary, but not sufficient, to win.

The converse of the survival challenge is the challenge to reduce enemy forces. To design such a challenge, you must create rules that determine how a unit may be removed from the game. Chess and checkers provide examples of such rules: *capture by replacement* and *capture by jumping,* respectively. War games implement simulated combat using complex mathematical models and track the health of each unit, which may be reduced by repeated attacks until reduction to zero destroys the unit.

Defending Vulnerable Items or Units The player may be called upon to defend other units or items, especially items that cannot defend themselves. In chess, all units protect the king. To meet such a challenge requires that the player know not only the capabilities and vulnerabilities of her units but also

those of the entity she must protect. She must be prepared to sacrifice some units to protect the vital one.

Stealth The ability to move undetected, an extremely valuable capacity in almost any kind of conflict, especially if the player takes the side of the underdog, can form a challenge in its own right. Games occasionally pose challenges in which the victory condition cannot be achieved through combat but must be achieved through stealth. *Thief: The Dark Project* was designed entirely around this premise. It required players to achieve their missions by stealth as much as possible and to avoid discovery or combat if they could.

Stealth poses a considerable problem in the design of artificial opponents for war games. In a game with no stealth, the AI-driven opponent has access to the complete state of the game world; to include stealth, you must restrict the opponent's knowledge, limit its attention, leave it ignorant of whole regions of the game world. You decide what the AI opponent does and doesn't know and define what steps it takes, if any, to gain further information.

Economic Challenges

An *economy* is a system in which resources move either physically from place to place or conceptually from owner to owner. This doesn't necessarily mean money; any quantifiable substance that can be created, moved, stored, earned, exchanged, or destroyed can form the basis of an economy. Most games contain an economy of some sort. Even a first-person shooter boasts a simple economy in which the player obtains ammunition by finding it or taking it from dead opponents and consumes it by firing his weapons. Health points are also part of the economy, being consumed by hits and restored by medical kits. You can make the game easier or harder by adjusting the amount of ammunition and number of medical kits, so that a player running short of fire power or health must carefully manage his resources.

The behavior of resources, as defined by the core mechanics of the game, creates economic challenges. Some games, such as *SimCity,* consist almost entirely of economic challenges. Such games tend to have flattened challenge hierarchies in which the atomic challenges appear similar to the overall goal of the game. Other games, such as first-person shooters, combine economic challenges with others such as conflict and exploration. We address the internal economies of games at length in Chapter 10, "Core Mechanics."

Accumulating Resources Many games challenge the player to accumulate something: wealth, points, or anything else deemed valuable. Acquisition of this kind underlies *Monopoly* and many other games in which the top-level challenge is to accumulate more money, plutonium, or widgets than other players. The game challenges the player to understand the mechanisms of wealth creation and to use them to his own advantage. In the case of *Monopoly,* the player

learns to mortgage low-rent properties and use the cash to purchase high-rent ones because high-rent properties produce more in the long run.

Achieving Balance Requiring your players to achieve balance in an economy gives them a more interesting challenge than simply accumulating points, especially if you give them many different kinds of resources to manage. Players in *The Settlers* games juggle quantities of raw materials and goods that obey complex rules of interaction: Wheat goes to the mill to become flour, which goes to the bakery to become bread; bread feeds miners who dig coal and iron ore, which goes to the smelter to become iron bars, which then go to the blacksmith to become weapons; and so on. Produce too little of a vital item and the whole economy grinds to a halt; produce too much and it piles up, taking up space and wasting time and resources that could be better used elsewhere.

Caring for Living Things A peculiar sort of economic challenge involves looking after a person or creature, or a small number of them, as in *The Sims* and *Creatures*. Unlike a large-scale simulation such as *Caesar*, in which the player must build and manage an entire town by meeting all of the economic and other challenges such a problem presents, these smaller-scale simulations focus on individuals. The game challenges the player to meet the needs of each individual and possibly improve its development. Because the game measures and tracks these needs via numeric terms, these qualify as economic challenges.

Conceptual Reasoning and Lateral Thinking Puzzles

We group *conceptual reasoning puzzles* and *lateral thinking puzzles* together because they both require *extrinsic knowledge,* that is, knowledge from outside the domain of the challenge itself. This sets them apart from formal logic puzzles in which all the knowledge required to solve the puzzle must be contained within its definition. Lateral thinking puzzles and conceptual reasoning puzzles may still require the use of logical thinking, however.

Conceptual Reasoning *Conceptual reasoning puzzles* require the player to use his reasoning power and knowledge of the puzzle's subject matter to arrive at a solution to a problem. In one round of the online multiplayer game *Strike a Match,* a number of words or phrases appear, and as they do, the player must find conceptually related pairs: if *Kong* appears, should the player watch out for a match with *King* (a movie) or *Hong* (a place) or *Donkey* (a video game)?

Another sort of conceptual challenge occurs in mystery or detective games, in which the player must examine the evidence and deduce which of a number of suspects committed the crime and how. In the game *Law and Order,* based on the television series of the same name, players follow clues, ignore red herrings, and arrive at a theory of the crime, assembling the relevant evidence to demonstrate proof. In order to succeed, however, the players must have some

familiarity with police forensic techniques as well as an understanding of human motivations for committing crimes. These details are extrinsic knowledge, not spelled out as part of the definition of the puzzle.

You may find designing conceptual reasoning challenges a lot of fun because they offer a lot of scope to the designer, but you'll work harder when creating these than putting together simpler trials such as physical or exploration challenges.

Lateral Thinking Lateral thinking puzzles are related to conceptual reasoning puzzles, but they add a twist: The terms of the puzzle make it clear to the player that what seems to be the obvious or most probable solution is incorrect (or the necessary elements to achieve the obvious solution are unavailable). The player must think of alternatives instead. A classic test of lateral thinking—and one used to demonstrate that chimpanzees possess this faculty—requires the subject to get an item down from a high place without using a ladder. Deprived of the obvious solution, he must find some other approach, such as putting a chair on top of a table, climbing up on the table, and then up on the chair. Because chairs do not ordinarily belong on tables, and neither chairs nor tables are intended for climbing, the test requires the subject to transcend her everyday understanding of the functions of objects.

Lateral thinking puzzles often require the player to use extrinsic knowledge gained in real life, but to use it in unexpected ways. In *Escape from Monkey Island,* the player has to put a deflated inner tube onto a strange-looking cactus to make a giant slingshot (or catapult), which requires knowing that inner tubes are stretchy. Adventure games frequently include lateral thinking puzzles. You must be careful not to make the solution too obscure or to rely on information that goes beyond common knowledge; you can expect the average adult player to know that wood floats, but you cannot expect the player to know that cork comes from the bark of certain species of Mediterranean oak tree (that challenge belongs in a trivia game). Provide hints or clues to help a player who gets stuck. In general, the more realistic the game, the more it may rely on extrinsic knowledge because players know that they can count on their real-world experience being meaningful in the game world. In a highly abstract or highly surreal game, the player won't expect common-sense experience to be of any use. Such games may still include lateral thinking puzzles, but the knowledge required to solve them must be available within the game.

Actions

As we explained in Chapter 8, "Creating the User Experience," the user interface links the input devices in the real world to actions that take place in the game world. *Actions*, as we use the term, refers to events in the game world

directly caused by the user interface's interpreting a player input. If the player presses a button on a game controller and the user interface maps that button to striking a cue ball in a game of pool, striking the cue ball constitutes an action. If the cue ball knocks another ball into a pocket, that is an event, but not an action; the movement of the other ball is a *consequence* of the player's action.

FYI No Hierarchy of Actions

We explained challenges in terms of a hierarchy because that hierarchy remains in the player's mind throughout the game, a collection of goals that she works to achieve. You might think, then, that there should be an equivalent hierarchy of actions—that if the game presents the high-level challenge "try to defeat the boss monster," there should be a high-level action called "defeat the boss monster."

We don't describe actions in terms of a hierarchy because a hierarchy of actions doesn't benefit either you or the player. Making up an artificial high-level action (defeat the boss monster) to go with a high-level challenge isn't terribly useful. If you tell the player, "To defeat the boss monster, perform the defeat the boss monster action," he hasn't learned anything. There's no such button on the controller, so what good does it do him?

Instead, we define actions in low-level terms, as events resulting directly from the player's use of the control devices. In fact, a game's tutorial levels often teach players how to defeat monsters not in terms of game actions but in terms of real-world button-presses. Tutorials say, "Attack monsters using your punch, kick, and throw shuriken buttons." It's up to the player to figure out how to combine these to defeat the boss monster.

Actions for Gameplay

Most of the actions that a player takes in a game will be intended to meet the challenges that she faces. We cannot possibly provide a list of all the kinds of gameplay-related actions that players can perform in game worlds; they vary from the simple and concrete, such as *fire weapon,* to those as complex and abstract as *send covert operatives to arouse antigovernment sentiment in a hostile nation*—which the player could do in *Balance of Power* by choosing a single item on a menu. We encourage you to study other games in the genre you have chosen to see what actions they support.

The interaction model that you have chosen for a particular gameplay mode determines a lot about the kinds of actions available in that mode. If you use an avatar-based interaction model, then the actions available to the player will, for the most part, consist of influencing the game world through the avatar.

In games using a multipresent model, the player acts indirectly by issuing commands to units, which themselves act within the game world (as in real-time strategy games) or acts directly on features of the world itself (as in construction and management simulations and god games).

Don't expect a one-to-one mapping between actions and challenges; many games include a large number of types of challenges but only a small number of actions, leaving the player to figure out how to use the actions in various combinations to surmount each challenge. Puzzles frequently do this. Faced with a scrambled Rubik's Cube, the player can take only one action: She can rotate one face of the cube 90 degrees. The solution to the puzzle consists entirely of making similar rotations.

Games offer many challenges but limited numbers of actions for two reasons. First, if you give the player a large number of actions to choose from, you must also provide a large user interface, which can be confusing to the player and increase the difficulty of learning the game. (If you implement a context-sensitive interface that chooses the correct action for the user based on context, you don't give the player the freedom to try interesting combinations.) Second, a large number of actions usually requires a large number of animations to display them all. This makes the game expensive to develop.

Many great games implement only a small number of actions but still let the player use them to overcome a wide variety of challenges. If you are imaginative enough, the challenges will be so interesting that the player will never notice.

Defining Your Actions

To define the actions that you'll implement, begin by thinking about the player's role in the game. At the concept stage of your project, you asked, "What is the player going to do?" and should have written down some general answers to that question. Now it's time to go into detail for each gameplay mode. Begin with the primary mode. If you wrote, "In the primary gameplay mode, the player will drive a car," think about exactly what actions driving entails. Pressing the accelerator, turning the steering wheel, and braking, of course, but what else? Shifting the gears? Turning on the lights? Using the handbrake? Some of the actions you decide on may have to do with challenges; others will simply be another part of the role.

Next, look at the challenges you designed for the primary gameplay mode. Begin with the atomic challenges you plan to offer; for each atomic challenge, write down how you expect the player to overcome it. Your answers will probably consist of individual actions or small combinations of actions. Ben Cousins has argued that game designers should spend most of their effort defining and refining the way that actions overcome atomic challenges because the player spends most of his time performing those actions (Cousins, 2004); this is excellent advice.

After you define the actions that will meet the atomic challenges, consider the intermediate and higher-level challenges in the gameplay mode. Can they

all be met with the actions you've defined, or will they require additional ones? Add those to the list.

Finally, consider actions unrelated to gameplay that you may want to make available to the player. You may already have some that come with the player's role, but you may want to include others for other reasons. See the list in the next section for some ideas.

Once you have been through this process for the primary gameplay mode, do it all again for each of the other modes. When you believe you have comprehensive lists of all the actions that you want to include in each mode, you're ready to start defining the user interfaces for the different modes: assigning actions to control mechanisms. (See Chapter 8, "Creating the User Experience.")

Actions That Serve Other Functions

Games include many actions that allow the player to interact with the game world but not engage in gameplay. Games also offer actions that aren't specifically play activities but give the player control over various aspects of the game. The following list describes a number of types of non–challenge-related actions.

- **Unstructured play.** You will almost certainly want to include some fun-to-perform actions that don't address any challenge. Players often move their avatars around the game world for the sheer fun of movement or to see a new area even if it offers no challenges, referred to as sightseeing. You may want to include actions just because they're part of the role. In most driving games, honking the horn accomplishes nothing, but if you couldn't honk the horn, the game would feel incomplete.
- **Actions for creation and self-expression.** See Chapter 5, "Creative and Expressive Play," for a discussion of actions allowing players to create and customize things, including avatars. Much of the activity in construction and management simulations consists of creative play rather than gameplay, although the player's actions are often constrained by limitations imposed by the game's internal economy.
- **Actions for socialization.** Players in multiplayer games, especially online games, need ways to talk to each other, to form groups, to compare scores, and to take part in other community activities. See Chapter 21, "Online Games" available on the Companion Website.

- **Actions to participate in the story.** Participating in interactive dialog, interacting with NPCs, or making decisions that affect the plot all constitute actions that allow the player to participate in a story even if those actions don't directly address a challenge. The more of them you offer, the more your player feels she is taking part in a story.

- **Actions to control the game software.** The player takes many actions to control the game software, such as adjusting the virtual camera, pausing and saving the game, choosing a difficulty level, and setting the audio volume. Some such actions may affect the game's challenges (setting the difficulty level certainly does), but the player doesn't take them specifically to *address* a challenge.

Saving the Game

Saving a game takes a snapshot of a game world and all its particulars at a given instant and stores them away so that the player can later load the same data, return to that instant, and play the game from that point. Saving and restoring a game is technologically easy, and it's essential for testing and debugging, so it's often slapped in as a feature without much thought about its effect on gameplay. As designers, though, it's our job to think about anything that affects gameplay or the player's experience of the game.

Saving a game stores not only the player's location in the game but also any customizations she might have made along the way. In *Michelle Kwan's Figure Skating Championship,* for example, the player could customize the body type, skin tone, hair color and style, and costume of the skater. The player could even load in a picture of her own face. The more freedom you give the player to customize the game or the avatar, the more data must be saved. Until recently, this limited the richness of games for console machines, but now console machines routinely come with enough storage to save a lot of customization data.

Reasons for Saving a Game

Reasons for saving a player's game or allowing him to save it include:

- **Allowing the player to leave the game and return to it later.** This is the most important reason for saving the game. In a large game, it's an essential feature. It's not realistic and not fair to the player to expect him to dedicate the computer or console machine to a 40-hour game from start to finish with no break.
- **Letting the player recover from disastrous mistakes.** In practice, this usually means the death of the avatar. Arcade games, which offer no save-game feature, traditionally give the player a number of lives and chances to earn more along the way. Until recently, console action games have tended to follow the same scheme. Richer games, such as role-playing or adventure games, usually give the player only one life but allow him to reload a saved game if his avatar dies or he realizes that he cannot possibly win the game.

- **Encouraging the player to explore alternate strategies.** In turn-based strategic games, saving the game allows the player to learn the game by trying alternative approaches. If one doesn't seem to work, he can go back to the point at which he committed himself and try another approach.

Consequences for Immersion and Storytelling

Saving a game is not always beneficial to the player's experience. The act of saving a game takes place outside the game world and, as a consequence, saving harms the player's immersion. If a game tries to create the illusion that the player inhabits a fantasy world, the act of saving destroys the illusion. One of the most significant characteristics of real life is that you cannot return to the past to correct your errors; the moment you allow a player to repeat the past, you acknowledge the unreality of the game world.

The essence of a story is dramatic tension, and dramatic tension requires that something be at stake. Reloading a game with a branching storyline affects the player's experience of the story because if he can alter the future by returning to the past and making a different decision, nothing really hangs in the balance. Real-world decisions bring permanent consequences; some can be modified in the future, but the original decision itself cannot be unmade. But when a player follows first one branch of a branching storyline and then goes back in time and follows another branch, he experiences the story in an unnatural way. The consequences of his actions lose their meaning, and his sense of dramatic tension is either reduced or destroyed completely. What is a benefit to strategic games—the chance to try alternate strategies—presents problems for storytelling.

Nevertheless, we feel that the arguments for saving outweigh these disadvantages. If the player destroys his immersion by repeatedly reloading the game, that is his choice and not the fault of the game designer or the story. As we pointed out in Chapter 7, "Storytelling and Narrative," a weakness of branching storylines is that they require the player to play the game again if he wants to see plot lines that he missed on his first play-through. Allowing the player to save and reload makes that easier for him. He may always choose not to reload if he doesn't want to.

Ways of Saving a Game

Over the years, designers have devised a variety of different ways of saving a game, each with its own pros and cons for immersion and gameplay.

Passwords If your game runs on a device with no storage at all (a rarity nowadays), you can't save the game in the middle of a level, but you can let the player restart the last level attempted. Each time the player completes a level, give her a unique password that unlocks the next level. At start-up, ask if she

wants to enter a password, and if she does so correctly, load the level unlocked by that password. She can go directly to that level without having to replay all the earlier ones. This method also allows her to go back and replay any completed level if she wants to.

Save to a File or Save Slot The player may interrupt play and save the current state of the game either to a file on the hard drive or, more usually, to one of a series of named "save slots" managed by the game program. When the player wants to begin the saved game, he tells the program to load it from the directory of files or slots. This allows the player to keep several different copies, saved at different points, and to name them so that he can remember which one is which.

Unfortunately, while this is the most common way of saving, it's also the method most harmful to the game's immersiveness. The user interface for managing the files or save slots necessarily looks like an operating system's file-management tool, not like a part of the fantasy world that the game depicts. You can harmonize this procedure better with appropriate graphics, but saving almost always takes the player out of the game world. Some games salvage the immersion to some degree by calling the file system the player's *journal* and making it look as if the saved games are kept in a book.

Quick-Save Fast-moving games in which the player's avatar stays in more or less constant danger (such as first-person shooters) frequently offer a quick-save feature. The player presses a single button to save the game instantly at any time, without ever leaving the game world. The screen displays the words *Quick saved* for a moment, but otherwise the player's immersion in the world remains unperturbed. The player can reload the game just as swiftly by pressing a quick-load button. The game returns immediately to its state at the last quick-save, without going out of the game world to a file-management screen.

Disadvantages of quick-save arise because saving so quickly usually means the player doesn't want to take the time, and isn't offered the chance, to designate a file or slot. Most such games offer only one slot, although some let players designate a numbered slot by entering a digit after they press the quick-save button. Players remember which slot is which when quick-loading. Quick-save sacrifices flexibility to retain immersion and speed.

Automatic Save and Checkpoints A few games automatically save the state of the game when the player exits, so players can leave and return at any time without explicitly saving. This harms the player's immersion least of all, but if the player has recently experienced a disaster, he has no way to recover from it. More often, games save whenever the player passes a checkpoint, which may or may not be visible to him. Checkpoint saving is less disruptive

than quick-saving because the player never has to do anything. The player *can* go back and undo a disaster, provided that the disaster happened after the most recent checkpoint. But it means that the player can't choose to save whenever he wants or choose to restart at some earlier point. If the checkpoints occur infrequently, he might lose a great deal of progress in the event of a disaster. Although it's better for immersion from the player-centric standpoint, automatic checkpoint saving is inferior to quick-saving. With quick-save, the player always has the option *not* to save if for some reason he enjoys the risk of having to go back a long way. With automatic checkpoints, he has no choice.

A few games offer optional checkpoint saving, in which the player may choose to save or not every time he reaches a checkpoint. This gives him a little more control but still doesn't allow him to save at will, which we regard as preferable.

To Save or Not to Save

A few designers don't allow players to save their games within certain regions of the game or even to save at all. If the player can save and reload where he wants and without limit, he can solve puzzles or overcome other obstacles by trial and error rather than by skill, or he can use the saving system to avoid undesirable random events; if something bad happens by chance, the player can reload the game repeatedly until the undesirable event doesn't occur. This reduces the game's difficulty, and some designers argue against allowing players to save on that basis.

We disagree. Preventing the player from saving in order to make the game more difficult is lazy game design, adding difficulty without adding fun. If you really want to make the game harder, devise harder challenges. Forcing the player to replay an entire level because he made a mistake near the end wastes his time and condemns him to frustration and boredom.

You may not like it if a player repeatedly reloads a game to avoid a random event or to solve some problem by trial and error rather than skill, but the player doesn't play (or buy) the game to make you feel good. He might need to save the game for perfectly legitimate reasons. The notion that saving makes a game too easy assumes that the player is your opponent, a violation of the player-centric principle. Most games now recognize that players want—and sometimes need—to cheat by offering cheat codes anyway.

It's the player's machine; it's not fair to penalize him just because he has to go to the bathroom or because it's now his little brother's turn to play. Choose which mechanism works best for your game, weighing the advantages and disadvantages of each, but do let the player save the game, and preferably, whenever and wherever he wants to. It does no harm to your game to give the player the freedom to choose when he wants to save—or whether he wants to

save at all. We strongly believe in players' fundamental right to be able to stop playing without losing what they have accomplished.

Allow the Player to Save and Reload the Game

Unless your game is extremely short or your device has no data storage, allow the player to save and reload the game. His right to exit the game without losing the benefit of his achievements supersedes all other considerations.

Summary

Gameplay is the heart of a game's entertainment, the reason players buy and play games. We began the chapter with some principles to keep in mind in order to make gameplay fun. Next we examined the *hierarchy of challenges,* the fact that a player experiences several challenges at once, and defined the concept of atomic challenges. We noted the difference between the *intrinsic skill* required by a challenge and the *stress* that time pressure puts on a player and how these two elements combine to create *difficulty.*

Gameplay itself took up most of the chapter, with definitions and discussions of the many types of challenges that video games employ and various ways of adjusting their difficulty level. From challenges, we turned to the actions that you can offer the player, which include actions not directly related to gameplay. Finally we discussed the pros and cons of different ways of saving the game, an important feature for any game more than a few minutes long.

Armed with this information and with a little research, you should be able to analyze the gameplay of most of the video games currently for sale and to design others using similar kinds of challenges and actions.

Test Your Skills

MULTIPLE CHOICE QUESTIONS

1. The main reason a game fails to be fun is that it
   A. has too many levels.
   B. contains stereotyped characters.
   C. contains elementary errors.
   D. has challenges that are too easy.

2. Which of the following is *not* a principle that contributes to a fun game?
   A. Get a feature right or leave it out.
   B. Be true to your vision.
   C. Know your target audience.
   D. Design around the marketing campaign.

3. A player who needs a tool right now to cut down the thorn trees preventing him from getting into the castle is facing a(n)
   A. atomic challenge.
   B. intermediate challenge.
   C. logical challenge.
   D. economic challenge.

4. Games generally give the player explicit instructions about
   A. intermediate challenges.
   B. the victory condition for the entire game.
   C. simultaneous challenges.
   D. pattern recognition challenges.

5. A player needs intrinsic skill to
   A. press a button repeatedly to make a gun shoot.
   B. direct an avatar in exploring a game world.
   C. solve a mathematical code in a puzzle.
   D. communicate with nonplayer characters.

6. In an archery game, which victory condition includes both factors that contribute to absolute difficulty?
   A. The player who places the most arrows in the center ring in two minutes wins.
   B. The player who places the most arrows in the center ring wins.
   C. The player who hits the target the most times wins.
   D. The value of arrows that miss the target are subtracted from those that hit the center ring.

7. Examples of physical coordination challenges include
   A. putting objects in a specific order and understanding probabilities.
   B. recalling information and detecting traps.
   C. accumulating resources and understanding strategy.
   D. shooting rapidly and learning a vehicle's acceleration rate through experience.

8. When you design a challenge that will involve increasing time pressure, what should you do to keep the same level of absolute difficulty?
   A. Increase the intrinsic skill required at the same rate.
   B. Reduce the intrinsic skill required as the time pressure increases.
   C. Add simultaneous challenges as time pressure increases.
   D. Give players a way to reduce time pressure.
9. An example of a pattern recognition challenge might be
   A. snares set in wooded areas that can catch the player unawares.
   B. learning that a golden orb can always defeat a bloodbeast.
   C. testing all the doors to see if they lead to secret passages.
   D. learning to manage resources so that just enough manufactured items are produced.
10. The logistical challenge routinely included in real-time strategy games is
   A. weapons production.
   B. accumulating construction materials.
   C. solving logistical puzzles.
   D. racing to complete a factory.
11. If you require a player to figure out a way to get across a narrow, deep chasm in a forest when her only tool is an ax, you are asking her to demonstrate
   A. logistical thinking.
   B. strategic planning.
   C. lateral thinking.
   D. tactical planning.
12. Why do games offer many challenges but limited actions?
   A. Challenges are easier to create than actions.
   B. Actions are less important than challenges in gameplay terms.
   C. A player-centric approach requires the designer to concentrate on challenges.
   D. Offering many actions would require a large interface and many animations.
13. Which of the following is an action that does *not* contribute to gameplay?
   A. Steering a car.
   B. Picking up a weapon.
   C. Turning off sound effects.
   D. Opening a door.

14. One of the best reasons to allow a player to save a game is to
    A. recover from a disaster.
    B. eliminate the possibility that he will cheat.
    C. prevent him from using trial and error to solve challenges.
    D. avoid having to deal with the avatar death issue.

15. To avoid harming a player's immersion when saving,
    A. give the player a button to click that will take her to a Save dialog box.
    B. save the game automatically when the player exits or passes a checkpoint.
    C. have the player press a key combination and then type a password to save the file.
    D. allow the player to save only during cut-scenes.

EXERCISES

1. Write the rules for a simple, single-player, PC-based puzzle game like *Bejeweled* but make up your own mechanics for earning points. Document all the challenges and actions of the game. You must create at least ten different *kinds* of atomic challenges. Indicate what action the player should use to surmount each challenge and what reward the player gets for doing so. You must also create and document at least four actions that are not intended to meet challenges but serve some other purpose. You do not have to design a user interface in detail but may find it helpful to make and submit a quick sketch of the screen and the layout of the controls.

2. Choose an action or action-adventure game you are familiar with (or your instructor will assign one). Document the challenge hierarchy of the first level in the game that is *not* a tutorial level, diagramming it out as in Figure 9.1. (If the level includes more than fifty sequential atomic challenges, you may stop after fifty, but be sure to include any level bosses or major challenges that occur at the end of the level.) If you have the necessary software, play partway through the level, take a screen shot, and indicate on your diagram what challenges you were facing at that moment, similar to the gray boxes in Figure 9.1. If you faced simultaneous challenges, indicate that also. Submit the screen shot along with your diagram.

3. Think of a game you are familiar with that permits the player to achieve victory by different strategies, similar to Figure 9.2. Write a short essay documenting each approach and how the hierarchy of challenges (including the intermediate challenges) differs in each one. If one strategy seems more likely to achieve victory than another, say so and indicate why. Your instructor will give you the scope of the assignment.

4. Choose a single- or multiplayer role-playing game that you are familiar with (or your instructor will assign one). Identify all the actions it affords. (You may find the manual helpful.) Divide the actions into those intended to meet challenges, those that participate in the story, those that facilitate socializing with other players (if any), housekeeping operations such as inventory management, and those that control the software itself. If another category suggests itself, document it. Also note any actions that fall into more than one category and indicate why. The size of the game that you or your instructor selects will determine the scope of the assignment.
5. Choose ten different types of challenges from among the ones listed in *Commonly Used Challenges* in this chapter. For each type, devise one example challenge and *two* example actions that overcome it (this may rule out some types). Describe the challenge and the two actions in a paragraph, ten paragraphs in all.

DESIGN QUESTIONS

1. What types of challenges do you want to include in your game? Do you want to challenge the player's physical abilities, his mental abilities, or both?
2. Game genres are defined in part by the nature of the challenges they offer. What does your choice of genre imply for the gameplay? Do you intend to include any cross-genre elements, challenges that are not normally found in your chosen genre?
3. What is your game's hierarchy of challenges? How many levels do you expect it to have? What challenges are typical of each level?
4. What are your game's atomic challenges? Do you plan to make the player face more than one atomic challenge at a time? Are they all independent, like battling enemies one at a time, or are they interrelated, like balancing an economy? If they are interrelated, how?
5. Does the player have a choice of approaches to victory? Can he decide on one strategy over another? Can he ignore some challenges, face others, and still achieve a higher-level goal? Or must he simply face all the game's challenges in sequence?
6. Does the game include implicit challenges (those that emerge from the design), as well as explicit challenges (those that you specify)?
7. Do you intend to offer settable difficulty levels for your game? What levels of intrinsic skill and stress will each challenge require?

8. What actions will you implement to meet your challenges? Can the player surmount a large number of challenges with a small number of actions? What is the mapping of actions to challenges?
9. What other actions will you implement for other purposes? What are those purposes—unstructured play, creativity and self-expression, socialization, story participation, or controlling the game software?
10. What save mechanism do you plan to implement?