Chapter 14
The Design Document

We have said throughout this book that digital game development is an inherently collaborative medium. In the last two chapters we have looked at all the various types of people who make up that collaborative environment, as well as the stages of the production process. One of the most important parts of managing this process is communicating the overall vision of the game to each and every team member. If your team is very small, or if you are working alone, this might not be a problem. But most games are complex enough, and most teams are large enough, that the most effective way to ensure communication is to write down that vision as well as a detailed plan for executing it. This plan is called the design document, and the game designer is its primary author and caretaker. In recent years, many teams have begun using online tools such as wikis to create and manage their design documents in a collaborative environment. Wikis can include text, images, and other media and changes to the design that can be tracked by users.

Whether you use a wiki or standard word processing software, the goals of the design document are the same: to describe the overall concept of the game, target audience, gameplay, interfaces, controls, characters, levels, media assets, etc. In short, everything the team needs to know about the design of the game. The artists will use it to lay out interfaces that reflect the features that you and the team have designed, the programmers will use it to define the software modules for those features, the level designers will use it to understand how their level fits into the overall story arc, the producer will use it to generate an accurate budget and schedule, and the QA department will use it to develop a comprehensive test plan.

As team sizes, schedules, budgets, and the overall complexity of game designs have grown exponentially, the need for clear, comprehensive documentation has become unmistakable. Most game developers and publishers today would never think of going into production without a detailed design document, and updating this living document throughout production is a critical responsibility of the game designer.

Communication and the Design Document

A good design document is like sound blueprints for a building. Everyone on the team can refer to and add comments while they do their separate tasks and understand how their work fits into the game as a whole. The writing of the document facilitates collaboration and useful conversations between team members.

Without design documentation to direct their efforts, the individuals on a team might interpret what they know about the game in their own unique ways, working hard, but not necessarily toward the same ends. When it comes time to integrate that work, art might have been made to unusable specs, technology

might reflect out-of-date features, or the essence of the gameplay might have been lost in the level designs.

To create an effective design document, the game designer needs to work with every other member of the team to make sure that the areas of the document affecting their work are accurate and achievable. In this way, the writing of the document itself generates communication. By conferring on the details of the game via text, wireframes, concept art, flowcharts, etc., team members have to think through the entire game, from the highest-level vision concepts to the lowest-level art specifications, the file types, and the font sizes. Game developers tend to be visual people, so supplementing the document with lots of visuals is generally a good thing.

There is a tendency for design documents to become very large. This is especially true if you use a wiki to write your design document because the collaborative management of these online documents allows them to grow very easily. However, a good design document can (and should) be succinct. Effective design documents can communicate core information in 50 to 100 pages so that a busy executive or programmer can find the areas that affect them quickly and easily. If there are areas that need to be expanded on as production moves forward, one strategy is to create subdocuments that delve into these areas more deeply.

Always keep in mind that you are not writing the design document for the sake of writing it—your objective is communication, so do whatever it takes to accomplish that goal. Documentation is not a substitute for talking to your team. Just because you have written it down, do not assume that everyone has read and understood your vision. Writing the document provides a process for establishing communication and serves as a touchstone for the entire team in terms of creative and technical designs, but it is not a substitute for team meetings and in-person communication.

Contents of a Design Document

The game industry has no standard format for documenting designs. It would be nice if there were a set formula or style to follow, like the standards for screenplays or architectural blueprints, but this simply does not exist. Everyone does agree that a good design document needs to contain all the details required to create a game; however, what those details are will be affected by the specifics of the game itself.

In general, the contents of a design document can be broken up into the following areas:

- Overview and vision statement
- Audience, platform, and marketing
- Gameplay
- Characters (if applicable)
- Story (if applicable)
- World (if applicable)
- Media list

The design document can also include technical details, or these can be articulated in a separate document called a technical specification. The technical specification or the technical sections of the design document are generally prepared by the technical director or lead engineer.

Exercise 14.1: Researching Design Documents

To get a feeling for the various ways that designers approach the writing of design documents, go on the Web and do a search using Google for "game design documents." You will find dozens posted on the Internet. Pick two and read through them. What are their strengths and weaknesses? If you were a member of the design team, would you be able to execute the design as described? What questions do you have for the designers after reading the documents?

When you approach the writing of a design document, it is easy to get distracted by the scope of the document and forget the ultimate goal: to

communicate your game design to the production team, the publisher, the marketing team, and anyone else with a vested interest in the game. This is one reason why we advise you not to write your design document until you have built and playtested a working prototype of your idea. Having this type of concrete experience with your proposed gameplay can make all the difference in your ability to articulate that gameplay in the design document.

You should also think of your design document as a living document. You will likely have to make a dozen passes before it is complete, and then you will need to constantly update it to reflect changes that are made during the development process. Because of this, it is important to organize your document modularly. If you organize your document carefully from the beginning, it will be easier to update and manage as it grows in size and complexity. Also, as we mentioned earlier, it will be easier for each group to find and read the sections that affect their work.

Using a wiki to create your design document will naturally enforce this idea of modularity. You will want to create separate pages for the various areas of your design and subpages for areas that require deeper descriptions, images, charts, or other materials.

The following outline is an example of how you might organize your design document. We have noted under each section the types of information it should contain. Keep in mind that our goal here is not to give you a standard format that will work for every game, but rather to provide you with ideas for the types of sections you might want to include. Your game and its design should dictate the format you use for your own document, not this outline.

1. **Design History**
   A design document is a continuously changing reference tool. Most of your teammates won't have time to read the whole document over and over again every time that a new version is released, so it is good to alert them to any significant modifications or updates that you have made. As you can see, each version will have its own section where you list the major changes made in that iteration. If you use a wiki, this section will be replaced by the editing history feature of the software. This makes it simple and effortless to track changes to the document and to backtrack changes if it becomes necessary.

   1.1 Version 1.0
   1.2 Version 2.0
      1.2.1 Version 2.1
      1.2.2 Version 2.2
   1.3 Version 3.0

2. **Vision statement**
   This is where you state your vision for the game. It is typically about 500 words long. Try to capture the essence of your game and convey this to the reader in as compelling and accurate a way as possible.

   2.1 Game logline
   In one sentence, describe your game.
   2.2 Gameplay synopsis
   Describe how your game plays and what the user experiences. Try to keep it concise—no more than a couple of pages. You might want to reference some or all of the following topics:

   - Uniqueness:
     What makes your game unique?
   - Mechanics:
     How does the game function? What is the core play mechanic?
   - Setting:
     What is the setting for your game: the Wild West, the moon, medieval times?
   - Look and feel:
     Give a summary of the look and feel of the game.

3. **Audience, Platform, and Marketing**
   3.1 Target audience
   Who will buy your game? Describe the demographic you are targeting, including age, gender, and geographic locations.
   3.2 Platform
   What platform or platforms will your game run on? Why did you choose these platforms?
   3.3 System requirements
   System requirements might limit your audience, especially on the PC, where the hardware varies widely. Describe what is required to play the game and why those choices were made.

3.4 Top performers
List other top-selling games in the same market. Provide sales figures, release dates, information on sequels and platforms, as well as brief descriptions of each title.

3.5 Feature comparison
Compare your game to the competition. Why would a consumer purchase your game over the others?

3.6 Sales expectations
Provide an estimate of sales over the first year broken down by quarter. How many units will be sold globally, as well as within key markets, like the United States, England, Japan, etc.?

4. Legal Analysis
Describe all legal and financial obligations regarding copyrights, trademarks, contracts, and licensing agreements.

5. Gameplay

5.1 Overview
This is where you describe the core gameplay. This should tie directly into your physical or software prototype. Use your prototype as the model, and give an overview of how it functions.

5.2 Gameplay description
Provide a detailed description of how the game functions.

5.3 Controls
Map out the game procedures and controls. Use visual aids if possible, like control tables and flowcharts, along with detailed descriptions.

5.3.1 Interfaces
Create wireframes, a type of functional visualization described on page 400, for every interface the artists will need to create. Each wireframe should include a description of how each interface feature functions. Make sure you detail out the various states for each interface.

5.3.2 Rules
If you have created a prototype, describing the rules of your game will be much easier. You will need to define all the game objects, concepts, their behaviors, and how they relate to one another in this section.

5.3.3 Scoring/winning conditions
Describe the scoring system and win conditions. These might be different for single player versus multiplayer or if you have several modes of competition.

5.4 Modes and other features
If your game has different modes of play, such as single and multiplayer modes, or other features that will affect the implementation of the gameplay, you will need to describe them here.

5.5 Levels
The designs for each level should be laid out here. The more detailed the better.

5.6 Flowchart
Create a flowchart showing all the areas and screens that will need to be created.

5.7 Editor
If your game will require the creation of a proprietary level editor, describe the necessary features of the editor and any details on its functionality.

5.7.1 Features

5.7.2 Details

6. Game Characters

6.1 Character design
This is where you describe any game characters and their attributes.

6.2 Types

6.2.1 PCs (player characters)

6.2.2 NPCs (nonplayer characters): If your game involves character types, you will need to treat each one as an object, defining its properties and functionality.

6.2.2.1 Monsters and enemies

6.2.2.2 Friends and allies

6.2.2.3 Neutral

6.2.2.4 Other types

6.2.2.5 Guidelines

6.2.2.6 Traits

6.2.2.7 Behavior

6.2.2.8 AI

7. **Story**
   - 7.1 Synopsis
     If your game includes a story, summarize it here. Keep it down to one or two paragraphs.
   - 7.2 Complete story
     This is your chance to outline the entire story. Do so in a way that mirrors the gameplay. Do not just tell your story, but structure it so that it unfolds as the game progresses.
   - 7.3 Backstory
     Describe any important elements of your story that do not tie directly into the gameplay. Much of this might not actually make it into the game, but it might be good to have it for reference.
   - 7.4 Narrative devices
     Describe the various ways in which you plan to reveal the story. What are the devices you plan to use to tell the story?
   - 7.5 Subplots
     Because games are not linear like books and movies, there might be numerous smaller stories interwoven into the main story. Describe each of these subplots and explain how they tie into the gameplay and the master plot.
     - 7.5.1 Subplot #1
     - 7.5.2 Subplot #2

8. **The Game World**
   If your game involves the creation of a world, you need to go into detail on all aspects of that world.
   - 8.1 Overview
   - 8.2 Key locations
   - 8.3 Travel
   - 8.4 Mapping
   - 8.5 Scale
   - 8.6 Physical objects
   - 8.7 Weather conditions
   - 8.8 Day and night
   - 8.9 Time
   - 8.10 Physics
   - 8.11 Society/culture

9. **Media List**
   List all of the media that will need to be produced. The specifics of your game will dictate what categories you need to include. Be detailed with this list, and create a file naming convention up front. This can avoid a lot of confusion later on.
   - 9.1 Interface assets
   - 9.2 Environments
   - 9.3 Characters
   - 9.4 Animation
   - 9.5 Music and sound effects

10. **Technical Spec**
    As mentioned, the technical spec is not always included in the design document. Often it is a separate document prepared in conjunction with the design document. This spec is prepared by the technical lead on the project.
    - 10.1 Technical analysis
      - 10.1.1 New technology
        Is there any new technology that you plan on developing for this game? If so, describe it in detail.
      - 10.1.2 Major software development tasks
        Do you need to do a lot of software development for the game to work? Or are you simply going to license someone else's engine or use a preexisting engine that you have created?
      - 10.1.3 Risks
        What are the risks inherent in your strategy?
      - 10.1.4 Alternatives
        Are there any alternatives that can lower the risks and the cost?
      - 10.1.5 Estimated resources required
        Describe the resources you would need to develop the new technology and software needed for the game.
    - 10.2 Development platform and tools
      Describe the development platform, as well as any software tools and hardware that are required to produce the game.
      - 10.2.1 Software
      - 10.2.2 Hardware
    - 10.3 Delivery
      How do you plan to deliver this game? On DVD, over the Internet, on wireless devices? What is required to accomplish this?

- 10.3.1 Required hardware and software
- 10.3.2 Required materials

10.4 Game engine
- 10.4.1 Technical Specs
  What are the specs of your game engine?
- 10.4.2 Design
  Describe the design of your game engine.
  - 10.4.2.1 Features
  - 10.4.2.2 Details
- 10.4.3 Collision detection
  If your game involves collision detection, how does it work?
  - 10.4.3.1 Features
  - 10.4.3.2 Details

10.5 Interface technical specs
This is where you describe how your interface is designed from a technical perspective. What tools do you plan to use, and how will it function?
- 10.5.1 Features
- 10.5.2 Details

10.6 Controls' technical specs
This is where you describe how your controls work from a technical perspective. Are you planning on supporting any unusual input devices that would require specialized programming?
- 10.6.1 Features
- 10.6.2 Details

10.7 Lighting models
Lighting can be a substantial part of a game. Describe how it works and the features that you require.
- 10.7.1 Modes
  - 10.7.1.1 Features
  - 10.7.1.2 Details
- 10.7.2 Models
- 10.7.3 Light sources

10.8 Rendering system
Rendering is a big part of games these days, and the more details you can provide, the better.
- 10.8.1 Technical specs
- 10.8.2 2D/3D rendering
- 10.8.3 Camera
  - 10.8.3.1 Operation
  - 10.8.3.2 Features
  - 10.8.3.3 Details

10.9 Internet/network spec
If your game requires the use of the Internet, LANs, or wireless networks, you should make the specs clear.

10.10 System parameters
We won't go into detail on all the possible system parameters, but suffice to say that the design document should list them all and describe their functionality.
- 10.10.1 Max players
- 10.10.2 Servers
- 10.10.3 Customization
- 10.10.4 Connectivity
- 10.10.5 Web sites
- 10.10.6 Persistence
- 10.10.7 Saving games
- 10.10.8 Loading games

10.11 Other
This section is for any other technical specifications that should be included, such as help menus, manuals, setup and installation routines, etc.
- 10.11.1 Help
- 10.11.2 Manual
- 10.11.3 Setup

We want to emphasize that the previous outline is merely a list of suggested topics that might need to be addressed to communicate your design. Every game will have its own specific needs, and the organization of your design document should reflect these needs.

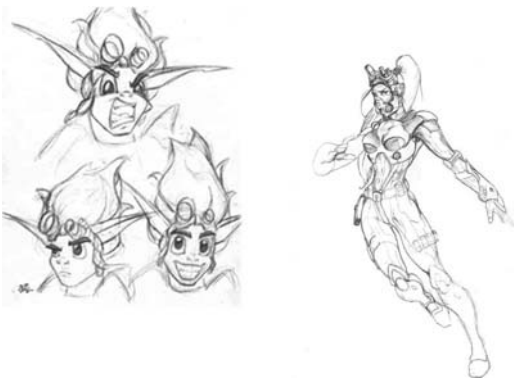
Under each of the sections in your design document, you need to answer all the questions that a team member might have. For example, the character designs section would include drawings and a description of each character in the game, while the levels section would include not only the intended gameplay for each level but explanations of any story elements that would be found in each level.

Writing Your Design Document

Before you sit down to write your design document, you should have spent a considerable amount of time thinking through the gameplay. The best way to do this, as we already discussed, is to build a physical or software prototype of your game and playtest it, improving and expanding your design until you have a solid foundation for your full game. Only after you have gone through several iterations of prototyping can you really be ready to write your design document.

Many designers will move to an outline and start pounding out the text of the design document at this point. We recommend flowcharting your entire game and building a set of wireframe interfaces for every screen in the game first. A wireframe is a rough sketch or diagram that shows all the features that will need to be included on an interface screen. By sketching out both the flow of the game and every required screen, you will be forced to think through the entire player experience for the game, finding inconsistencies and issues before any artwork or programming has been done.

Figure 14.2 is an example of a game flowchart created using typical software, like Microsoft Visio. This is for an online multiplayer version of the Wheel of Fortune game. As you can see, it illustrates how players move through the game and interact with it. Notice how the flowchart shows all possible paths through the game and all possible results, including how the player wins and loses and what happens if the player disconnects. You can use flowcharts to map out all sorts of processes within your game. The more detailed you are, the easier it will be to communicate your ideas to your teammates.



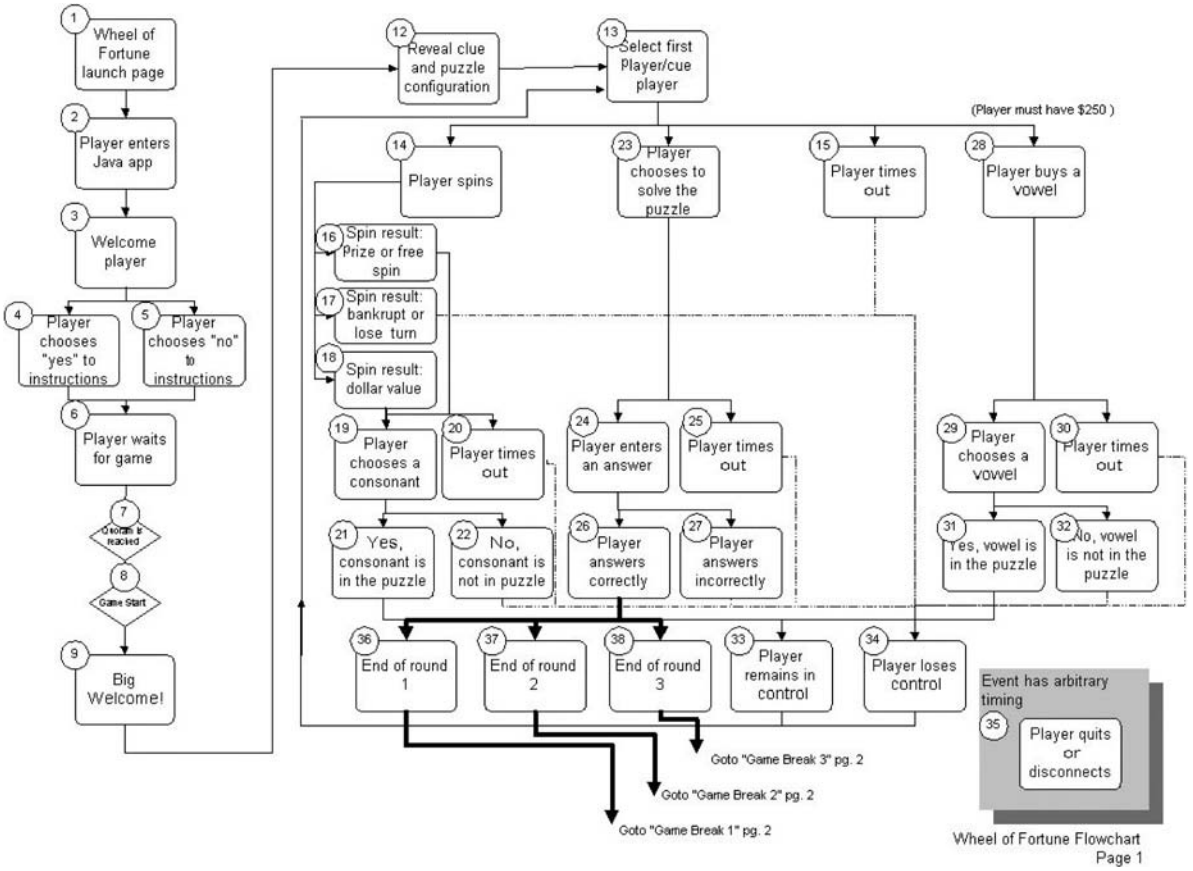
14.1 Character sketches from Jak & Daxter and Ghost

After you create your flowcharts, you will need to sketch out the main interfaces for the game in wireframe format. Figure 14.3 shows an interface wireframe from Wheel of Fortune, an early concept sketch for the interface, and the final interface as released. If you look closely, you will see that several changes were made during production, notably in the way that chat is handled in the game. Wireframes are not the end of the design process; they are the beginning. They give the game designer, artists, programmers, and producers a visual reference point to discuss the game in its early stages. They can also be usability tested at this point. Changes can be made to the design at this time, when they cost nothing to make, rather than months down the road when they would have much higher repercussions.

Exercise 14.2: Flowchart and Wireframes

Either on paper or using a software tool like Microsoft Visio or Flow Charting PDQ, create a full flowchart for your original game design. Next create a complete set of wireframes for every interface state in the game. Then annotate your wireframes with callouts describing every feature as described in the next paragraph.

Now that you have created a prototype, a flowchart, and a full set of wireframes, you will have a very good idea of what you need to communicate in your design document. You will also have a set of visual aids for explaining the features of your game. Most people can absorb information more clearly from visual displays like your wireframes than from long paragraphs of text explaining the features of your game. Because of this, your wireframes are not only a good tool to help you think through your game, but they are an excellent reference point for



14.2 Flowchart for multiplayer Wheel of Fortune

your readers. As you outline your document, use the flowchart and wireframes to explain the areas and features of the game. You might want to create callouts on your wireframes explaining how various features will work. These callouts can be supported by bullet points that expand on the visual diagrams.

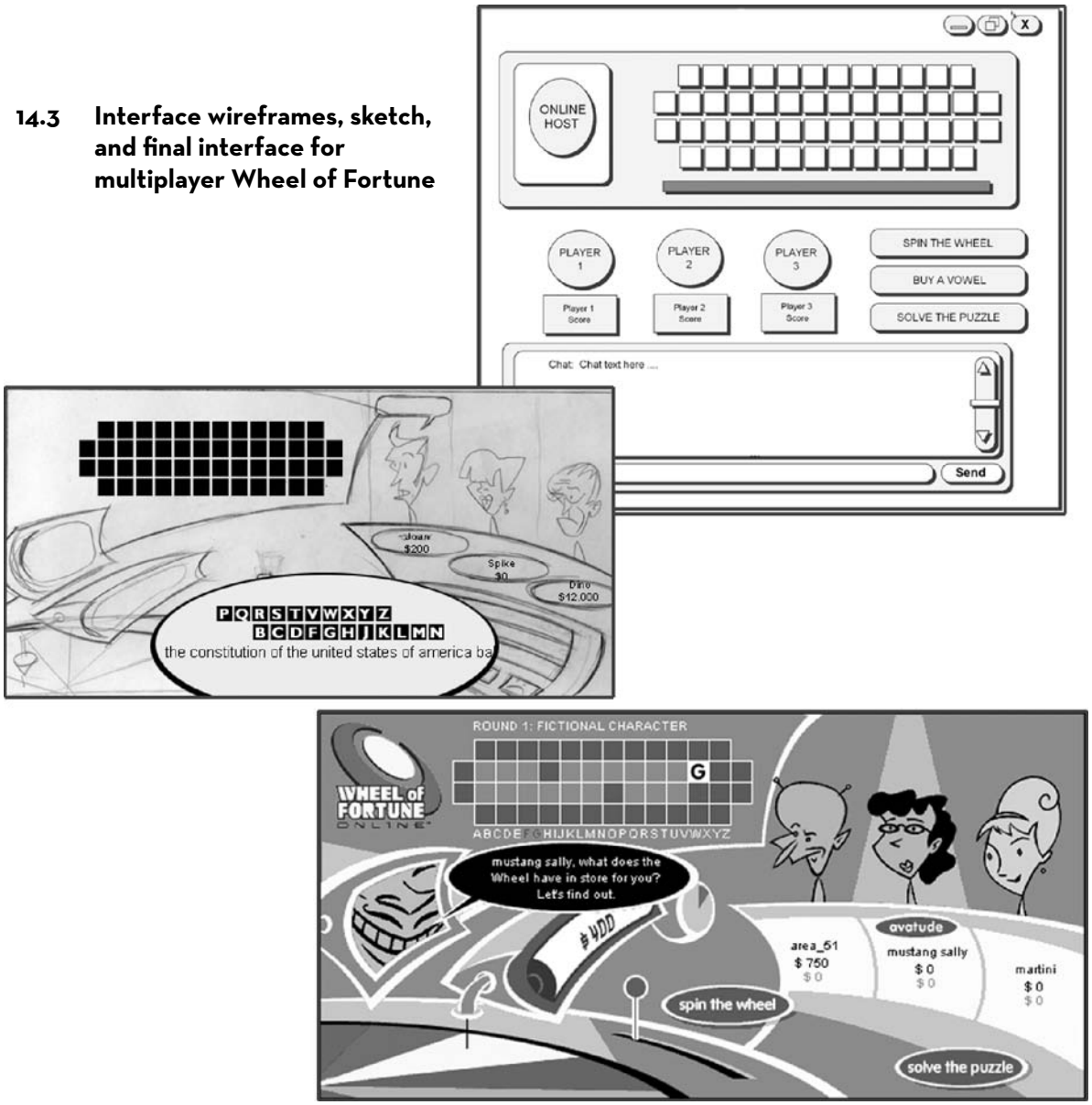
Ideally, by working through your concept from prototype to flowchart and wireframes to documentation, you will find that the document is actually quite simple to write. Instead of being faced with the mammoth task of thinking through the game while you write the document, you will have broken the tasks down into smaller stages that allow the game itself to dictate how the design document should be written.

As with the rest of your design experience, the document should be an iterative process. Do not try to complete it in a single pass. Let it grow over time. Fill out sections as they become clear, then go back to other sections and refine them.

Exercise 14.3: Table of Contents

Outline the table of contents for your original design document. Consider every aspect of your prototype, flowchart, and wireframes when you are deciding how to describe your game. Draw from the example documents you downloaded from the Web and the generic template found in this chapter.

14.3 Interface wireframes, sketch, and final interface for multiplayer Wheel of Fortune



You might realize while writing a later section that your thoughts from an earlier section need to be revised. Because each aspect of a game system is interrelated to the others, you will be continually going back and forth between the sections, modifying and updating them. Here is an example as to how the process might go for the level design section:

- *First pass:* Make outline of all levels and give them names.
- *Second pass:* Write one-paragraph descriptions of what takes place on each level.
- *Third pass:* Design maps for each level.
- *Fourth pass:* Populate maps with content.

Indie Game Jam: An Outlet for Innovation and Experimental Game Design

by Justin Hall

Austin Grossman is fleeing through a city, fervently searching through tens of thousands of shuffling, pixellated citizens for the wide sombrero of the man who wants to kill him. Austin's movements with the mouse are jerky, and he's holding his breath. Suddenly he hears a loud reverberating ping off to his left. Austin whirls to face the sound. The crowd parts, and before Austin can react, the man in the sombrero shoots him down.

Thatcher Ulrich watches over Austin's shoulder, gently nibbling his finger. Austin and his colleague, Doug Church, are playing Thatcher's game, Dueling Machine. In Dueling Machine, one player hunts another using sonar pings to find or avoid their enemy amidst tens of thousands of innocent bystanders. Few games use sound in such an integral way; this idea came from Marc LeBlanc. At the time, Marc worked for Visual Concepts, a video game development studio owned by Sega. Thatcher worked at Oddworld Inhabitants, a video game development studio owned in part by Microsoft. Given the competitive nature of the modern game industry, it doesn't make much sense that these two would be designing a video game together.

Thatcher and Marc are both working on Dueling Machine as part of the Indie Game Jam. Tired of calcification in commercial video games, a small group of game designers decided to jump-start some experimental game designs. In March 2002, they built a system for simultaneously displaying up to 100,000 moving characters on the screen at one time. Then they invited a dozen designer–programmer friends from a half a dozen game companies to join them for collective quick and dirty garage-style programming in a barn on a marina in Oakland, California.

The roots of electronic entertainment lie in these sorts of collaborations, in garages and basements and dank laboratories—places where people gathered over rudimentary machines to make virtual worlds. Thirty years ago, the people responsible for electronic entertainment were not yet game professionals, they were simply dedicated hobbyists. They tinkered with computers and code to make small simulations, establishing rules and parameters that their friends would break.

These tinkerers gave birth to a $20 billion industry. Electronic gaming has expanded rapidly; now 30 years later, most games are made by massive teams of specialized developers working for years with a marketable product in mind. Still, there are some folks who hope to encourage a generative creative spirit, to inject some creativity and vitality into a medium that's increasingly conservative.

This Indie Game Jam was organized in part by programmer Chris Hecker, who sees a games industry that is "too risk adverse." Hecker is eager to see an independent subculture for video games; a film festival or garage band ethos feeding new ideas to commercial developers.

When the Indie Game Jam started, Hecker worked out of the top floor of a red barn in a cluster of Victorian buildings surrounded by bland office parks and industrial sites. This cluttered, compact office served as a retreat for a revolving cast of independent programmers—laptops working on a futon propped up with old CPU cases between stacks of old graphics cards, books on math theory, and antique PC games.

Hecker opened up his Oakland office to serve as the site of the first few Indie Game Jams. Over a bowl of dry noodles at a nearby Vietnamese restaurant, Hecker speaks passionately about the evolution of gameplay: "Games of the future will be interactive not only at the second-to-second level, which we focus on now,

Indie Game Jam participants in action

Top left: Chris Hecker and Doug Church working on their game FireFighter. Top right: Chris Corollo and Brian Sharp of Ion Storm working on their game Wrath. Bottom: 2002 Indie Game Jammers, left to right: Austin Grossman, Robin Walker, Art Min, Brian Jacobson, Chris Hecker, Sean Barrett, Zack Simpson, Ken Demarest, Jonathan Blow, Doug Church, Brian Sharp, and Chris Corollo. Photos by Justin Hall.

but also at the minute-to-minute and hour-to-hour levels. This means that not only can you walk left or right interactively, but your decisions impact the overall flow of the game." Deeper interactions require richer simulations; Hecker gets excited about physics, the system of game rules determining interactions between physical objects in games. Do you want to stack those crates over there against the door to keep the bad guy from following you into this room to eat you alive? As Hecker points out, most games today won't allow that kind of strategy. Crates are for climbing on or smashing up only.

In March 2003, the second Indie Game Jam replaced 100,000 sprites with one main actor—the human body. The usually crowded Indie Game Jam headquarters was packed with projectors and webcams taped to the tops of halogen lamps and hanging off other hazardous supports. Building on Zack Simpson's Shadow Garden engine, 14 programmers had one weekend to design games that used the shadow of a human body cast on a wall as an interface for electronic entertainment. Casey Muratori and Michael Sweet created Owl Simulator, where a player's outstretched arms controlled their flight path. In Atman Binstock's Squisy Marshmallow Maze, two players struggle to move their colored block through a maze, using their shadow to block the progress of the other player. Squisy Marshmallow Maze games typically devolved into wrestling matches, a kind of physicality mostly unseen in video games until the advent of the Nintendo Wii, years later.

Programmer/designer Doug Church explains in an e-mail, "The game industry has always been about smoke and mirrors, and making the correct tradeoffs. So even little baby steps and experiments which seem silly now may lead to some real and unexpected successes later." Now that game development schedules have extended out beyond 18 months, it can be difficult to work through a few different game design concepts. The format of the Indie Game Jam weekend codefest is a good uninhibitor; as Doug explains, "Given a lack of research money and time, Game Jam-like 'no-time-to-think-just-type-and-see-what-happens' sort of events may help provide some of the ideas which eventually help us make progress."

IGJ co-organizer Sean Barrett is a refugee from the commercial game industry. Since leaving Looking Glass Studios in 2000, Barrett works independently to make software that goes beyond "people doing simple games and high-end graphics." Barrett has been integrating personality and conflicting priorities in a hovercraft game he's programming: "Your squadmates have personal agendas (derived from their allegiances to various political and religious groups) and become satisfied or unsatisfied with you based on whether you cooperate with those agendas in game." Choosing which objects to blow up might not seem like much of a step toward richer interaction, but any in-game motivation other than scoring and linear progress is remarkable. As Barrett points out about game making, "We do violent conflict great, but we don't do any other kind of conflict, especially interpersonal relationship sorts of conflict, at all."

In 2005, the fourth Indie Game Jam used an engine with human models from The Sims to make games about human interaction. By then, Indie Game Jam contributors included people drawn from other disciplines beyond programming: art, sound design, game theory, and education. And now Indie Game Jams were occurring elsewhere. In Lithuania in 2002, a group of young programmer/designers got together in a room, made games on a short deadline, and released them free online, using the first Indie Game Jam engine. Soon Toronto, Dallas, Boston, Ohio, and Nordic Game Jam groups launched, listing themselves on the Indie Game Jam page on Wikipedia.

Within years, schools, companies, and developer groups around the world were spending occasional weekends to hack at electronic play: professional and amateur programmers alike unleashed by a weekend of fast prototyping in a shared, supportive atmosphere. As commercial game industry budgets grow to accommodate ambitious graphics and rigid licenses, these kinds of development experiments give people a chance to poke at new ideas. Some rather mainstream developers are finding happiness in smaller, informal collaborations, some of which even evolve into commercial titles sold as PC or console downloads.

A weekend-long Indie Game Jam is usually not enough time to develop deep, engrossing games. Most of the participating programmers must leave happy fun prototyping land for their day jobs, and participating designers might not have the technical skills, tools, or time to accommodate big visions. For people who spend their days working on large, slow-moving entertainment software, an Indie Game Jam gives them a compressed freedom where personal inclination and a sense of playful experimentation direct game design more than market concerns.

On the last day of the first Indie Game Jam, most of the games had been finished and many of the programmers had headed home. Sean Barrett stood over Chris Hecker's shoulder as Hecker's dirty glasses reflected tens of thousands of tiny figures being mowed down by a single warrior on-screen. As Hecker handily killed with mouse and keyboard, the camera panned out slowly until his character was shown to be surrounded by an impossible sea of enemies. There was clearly no way he could survive. The artful futility of Barrett's Very Serious RoboDOOM put a wide grin on Hecker's face.

Each Indie Game Jam begins to explain how games can meet broader social needs: beyond athlete, gangster, and space marine power fantasies. Beyond our current vocabulary of "first person shooter" and "real time strategy." Past "attack" and "jump," where "manipulate" and "convince" are possible game verbs. Where learning to make your own rough draft games is part of being an active player.

More information about the Indie Game Jam can be found online at www.indiegamejam.com and on Wikipedia at en.wikipedia.org/wiki/Indie_Game_Jam.

About the Author

Justin Hall participates in digital culture and electronic entertainment. Present at the birth of the popular Web, Hall invested his mortal soul in the exchange of personal information online. Later consumed by machine stimulation, Hall set out to study video games to better understand his lifelong computer babysitters. He finished his MFA in interactive media at the University of Southern California in 2007. From there Hall went on to work with the team at GameLayers Corporation as CEO as they fashioned ongoing electronic play out of daily life through "Passively Multiplayer Online Games."

Exercise 14.4: Fleshing out Your Design Document

Flesh out your design document using your table of contents, flowchart, wireframes, and the documentation you created while designing your prototype, such as your concept document, rules, etc. Work with your team members to complete each section as described earlier.

Writing the design document will help you clarify the details of your design. When the document is written, it is used to manage team members from both the publisher and the developer. The core concept as detailed by the design document will be approved by the publisher before the team moves into production. Then the document will evolve as the project progresses.

Conclusion

In this chapter, you have learned how to take your original game concept from the prototype to a full design document or design wiki. You have created detailed flowcharts and wireframes for every area of your game. You have worked with your team, if you have one, to flesh out both the technical and creative tasks that will need to be accomplished to make your game a reality.

Writing and updating your design document is a monumental and sometimes tedious responsibility. The design document can be a useful tool or a millstone around the designer's neck. Always remember that the purpose of your design document is communication and articulation. A designer huddled in a cubicle writing in isolation for weeks on end will produce a document that is far less valuable than a designer who engages the team, includes them in the process, and works with them to build out each section.

By working with the team, a designer will not only wind up with a better design document but will also help focus the team on the project at hand. This is how living design documents are created, and when you have a living document in which everyone is an acting coauthor, it becomes a force in and of itself, which serves to unite the team and give them a common platform from which to understand the game as it evolves.