USING JAVASCRIPT AND HTML5 TO DEVELOP GAMES

THE WEB GAME DEVELOPER'S COOKBOOK

Evan BURCHARD

Praise for *The Web Game Developer's Cookbook*

"*The Web Game Developer's Cookbook* is a fun hands-on introduction both to building games and to web technologies. Learning through making is an empowering, exciting first step."

—**Jonathan Beilin**
DIY.org

"It is not only a book about libraries: it teaches how web pages work, how games work, and how to put everything together. Study one, learn three: best deal ever."

—**Francesco "KesieV" Cottone**
Web Alchemist, and Technical Advisor at Vidiemme Consulting

"A wonderful overview of the HTML5 Game Development landscape, covering a wide range of tools and 10 different game genres."

—**Pascal Rettig**
Author of *Professional Mobile HTML5 Game Development*

"With a friendly and reassuring tone, Burchard breaks down some of the most well-known gaming genres into their basic ingredients. *The Web Game Developer's Cookbook* transforms a seemingly daunting task into an approachable crash course even for those who've never written a line of code before."

—**Jason Tocci, Ph.D.**
Writer, Designer, and Researcher

This page intentionally left blank

The Web Game Developer's Cookbook

This page intentionally left blank

The Web Game Developer's Cookbook

Using JavaScript and HTML5 to Develop Games

Evan Burchard

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data is on file.

ISBN-13: 9780321898388
ISBN-10: 0321898389

Text printed in the United States on recycled paper at RR Donnelley, Crawfordsville, IN.
First printing: March 2013

Editor-in-Chief
Mark Taub

Acquisitions Editor
Laura Lewin

Development Editor
Songlin Qiu

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Apostrophe Editing Services

Indexer
Cheryl Lenser

Proofreader
Sarah Kearns

Technical Reviewers
Jonathan Beilin
Rich Jones
Jason Tocci

Editorial Assistant
Olivia Basegio

Interior Designer
Kim Scott

Cover Designer
Chuti Prasertsith

Compositor
Nonie Ratcliff

For Jade

This page intentionally left blank

Contents

PREFACE

When I was little, I learned that fun comes in plastic cartridges from Japan, stamped with the "Official Nintendo Seal of Quality," and smelling of Styrofoam. Challenge, discovery, and companionship were all bundled together in a magical box that output entertainment when you put these littler boxes in it and pressed "POWER." Later, I uncovered something shocking: These games (and games like them) could be made by mortal humans, sometimes only one or a few of them, but the team sizes were growing. As I was watching, what had started with small teams of hackers was becoming the 50-billion-dollar video game industry we know today.

But now, even as large studios dominate the market, a renaissance is brewing of small, independent teams working on games. Distribution platforms supporting these efforts have sprung up by the dozens, but nowhere is this revolution more pronounced than in the humble-and-often-overlooked web browser arena. Backed by advancements in browser technology, hundreds of free game engines are available that enable even a solo game designer to create games that are memorable, personal, fun, and potentially lucrative. All you need is a browser, a text editor, and the kind of information in this book. It's a few more button presses than turning on a console, but it has never been easier to make this kind of fun for yourself and others.

"POWER"

ACKNOWLEDGMENTS

First of all, you. Seriously. You are reading a book that I wrote. That blows my mind. Thank you so much.

I want to thank the team at Pearson, and especially Laura, Olivia, and Songlin for giving me the chance and the guidance I needed to be able to write this book.

Thanks to my friends and reviewers, Jon, Rich, Jason, Greg, BBsan, Pascal, Tim, and Tony.

To my mom, for her wisdom of people. To my dad, for his appreciation of nuance. To Amy, for her patience and perspective. To Gretchen and Max, my first play testers and the most candid, hilarious people I know.

To everyone who made the games that I loved growing up. And to everyone in the ROM hacking community of the '90s who first showed me how to break those games into pieces.

To open source contributors. I wouldn't be able to have fun or work in nearly the same way if there weren't people so committed to making the world awesome. Thanks especially to the people who built the tools that I use in this book (see Appendix C, "Resources"). This book absolutely couldn't exist without their efforts. And a special thanks to Kesiev for his early work in synthesizing and presenting the promise of HTML5 gaming.

I want to thank Mr. Morris for simultaneously justifying rebellion and paying attention; Dr. Jamison for teaching me to value breadth and depth of understanding; and Dr. Hatasa for giving me a chance to see the world from a completely new perspective.

Thanks to all the choir and theater kids, punks, weirdos, nerds, hackers, engineers, entrepreneurs, researchers, designers, dreamers, and polyglots who have kept me sane, entertained, and in just the right amount of trouble over the years. And a special thanks to the one theater kid who has put up with me for so long.

Finally, thanks to everyone who believed in me and told me why, and to everyone who didn't and told me why.

ABOUT THE AUTHOR

Evan Burchard recognizes that he is not the first or last person driven to learn programming by an interest in creating games, and seeks to empower others to take full advantage of the modern, free, and game-friendly web. In addition to designing games with electricity, ice, fire, and the latest browser technologies, he enjoys extremely long walks (his current record is Massachusetts to Iowa).

INTRODUCTION

Games used to require specialized tools to produce. Now all you need is a browser and a text editor. Even outside of HTML5 games, the time and cost to make a game has dropped so dramatically that people can now build games in hours or days. The indie game developer scene is growing, as are game jams, the online and in-person get-togethers for rapid game making.

The typical time frame for a game jam is 48 hours, as codified by larger distributed events such as Global Game Jam and Ludum Dare. But game designers (by definition) like to invent their own rules, so some jams can be as short as 1 hour long. Besides the social and collaborative benefits, game creators forcing themselves to create a game quickly can make them faster the next time around, which are great skills for long-term or short-term projects.

"Building things quickly" isn't just for those indie developer punks. In the corporate world, it's called productivity. Finding and learning to use good tools is a much easier path to productivity than the ill-defined goal of "getting smarter." And it passes quite convincingly as intelligence, especially if your definition of "tool" encompasses things such as mathematics.

This book tracks down some of the best HTML5 game engines available, whittled down from an initial list of more than 100. These, and the other tools in this book, enable you to create games quickly in the browser. All of them are accessible in that nothing is required beyond loading their JavaScript into an HTML file and occasionally adding a few lines of code. Overall, the chosen few have great documentation and a thriving community around them. Some of the engines are bigger than others. They all expose a unique set of functionality for game making, and through learning to use a few, you can start to see what is common among them as well as what is different.

Each engine is paired with a complementary game genre in each chapter. The complexity of the genre informs the requirements for increasingly feature-rich engines as you progress through the book. By the end, you should feel comfortable learning a new game engine or even hacking together your own.

Each game can be created in a few hours. Will these be your favorite games of the given genre? It's highly unlikely. This book demonstrates how to break down genres of games into their basic

elements. This lays the foundation, puts up the frame, and installs the drywall. In some cases, the author has decorated sparsely. There might be a big hole in the roof and the author's favorite pictures are hanging on the wall. Don't hesitate to build a courtyard, install shag carpeting, or plant some ginkgo trees if you want. Take down my pictures. You'll see where to get all of the materials you need, but it's your house. Do whatever you want with it. These are your games as soon as you load them up.

When you finish this book, you should be able to easily think of a scene from your favorite game, break it into a list of features, and know how you would create a similar experience by using the toolset you use throughout this book. You might even have a sense of how difficult it is or how long it would take. If you are productive with these tools, and you have a good story to tell, you should be able to create something that someone loves in no time.

Audience for This Book

There are many paths that may have brought you here. If you have an interest in games, and are just learning to code, this book is for you. If you are a web developer or designer who is looking for exposure to tools, techniques, or templates for making games, or you want to go from beginner to intermediate level JavaScript coding, this book is for you. If you are a game designer or developer for flash, native mobile/desktop applications, or some other platform, investigating how to build things in an HTML5/JavaScript context, this book is for you also. If you have a tattoo of the HTML5 shield, regularly present about your open source contributions to game engines, and can jam out an HTML5 Mario 64 clone with a native iPhone port in a weekend, this book might not be what you're looking for.

Coding Style Conventions Used In This Book

To indicate that a line is new or has changes, bold text is used. When code is omitted from a listing, an ellipsis (...) is used in place of 1 or more lines of code. To explicitly call out removed or changed lines of code, a commented (begins with //), bolded line of code is used in its place. If an entire listing shows new code, the text will not be bold.

The continuation character (➥) indicates that code is continued from the previous line.

When code appears inside of the text, `it will look like this`.

> ### note
>
> When there is something that requires a bit more explanation, it is called out in a "note" that looks like this.

tip

When there is something that doesn't quite fit in the text, but is helpful to know, it appears as a "tip" that looks like this.

warning

WARNINGS LOOK LIKE THIS A "warning" is used when there is something that may not be obvious and could cause problems if you did not know about it.

How This Book Is Organized

This book is broken up into 11 chapters, with one game per each in Chapters 1 through 10, along with three appendixes (A, "JavaScript Basics," B, "Quality Control," and C, "Resources"). Chapter 1, "Quiz," assumes no knowledge of HTML, CSS, JavaScript, or a functional toolset. The rest of the chapters assume that Appendix A and Chapter 1 are well understood by you. From a code standpoint, none of the chapters rely on tools built in previous chapters. That said, the genres are ordered roughly by their complexity, so gaining experience in the simpler genres may be of benefit in creating the games in the later chapters. Chapter 11, "Leveling Up," serves as a guide to what you might want to do after completing this book. It is complemented by the list of resources in Appendix C, which also supports Chapters 1 through 10 by highlighting what tools are needed to create the games in this book.

Each game is broken up into "recipes," which, in addition to breaking up games into understandable chunks of code and text, are reflected in the source files provided at jsarcade.com. What this means is that every recipe contains a complementary folder within the code that can be downloaded on the companion site. If you get lost or want to warp ahead, you can start fresh with the code in a later recipe. Also, if you want to preview what the game will be like when you finish a chapter, you can warp straight to the "final" directory for the given game/chapter and see what it is you are making.

If you find yourself getting lost a lot, and you have a good understanding of the material in Chapter 1 and Appendix A, Appendix B is there to provide more context around how to prevent getting stuck, and what to do about it when you are.

How To Use This Book

To make full use of the text, you need to download the source code files for each chapter. This includes JavaScript, HTML, CSS, images, and any additional files needed for each recipe. They are linked to at jsarcade.com. Code is organized first by chapter title. Inside of each chapter's directory is a full copy of the code you need to make the game run, with three different types of directories. "initial/" marks the minimum amount of code you need to have a game running. "after_recipe<x>/" directories specify "checkpoints" after each recipe (most headings in each chapter) so that in case you get lost along the way somehow, you can be confused for only a page or two. The "final/" directory specifies the finished game after you complete a chapter. While inside any of the chapters' recipe directories, you can see an index.html file. If you double-click it or otherwise open it in a browser by some other means, you can see the game as it exists after following the recipe that is indicated by the directory name. Demos of all the final versions of each game are available at jsarcade.com, so you can preview a game and choose which one you want to implement next.

> ## note
>
> The source files for all the games, game engines, and other required software are available to download at jsarcade.com and the Publisher's website at informit.com/title/9780321898388

You can skip around, but keep in mind that the games get more complex as the book progresses. If you have trouble understanding anything, make use of the checkpoint (after_recipe<x>) code, and pay special attention to Chapter 1 and Appendix A. If you have trouble understanding why something is going wrong, read through Appendix B.

You may notice that after finishing a chapter, you still feel like the game is missing something. It could be an explosion, a great storyline, or a boss battle. You can find suggestions at the end of each chapter of things that you could add to them—whether you have different ideas or like the suggestions provided, go for it. These become your games as soon as you get the code running on your computer. They are templates, and meant to be hacked, extended, and personalized. I will applaud and definitely not sue you for beating me at my own game making.

CHAPTER 1

QUIZ

This type of game has simple rules. There is a right answer. You either know or you guess. From trivia nights at restaurants to the SAT, it would be an understatement to call this genre of game "pervasive" in today's world. TV game shows often find their way into an interactive format. Even when there are more elements to a game than simple questions and answers, every piece of software operates on some kind of underlying logic. When the king asks if you want to fight the dragon, and you have to say "yes," that's a fairly small, easy quiz. It would be a stretch to say that falling in a hole in a platformer, or losing all of your hit points in an RPG, is the same experience as missing a question on a quiz, but programming the rules and consequences of each is similar in all game genres.

Recipe: Making the Questions

To accommodate various levels of web development experience among readers, this chapter is intended to be as clear as possible for people just starting out. The following chapters become more complex, but this chapter's material is meant to ensure an appropriate level of understanding early. We all started somewhere, and for some it might be here. If the information presented in this chapter seems extremely simple to you, you can skim it or skip it entirely. Things will become more complicated and difficult later.

There are three primary goals in developing this chapter's game. First, we want to establish a baseline understanding of HTML, CSS, and JavaScript. The knowledge of JavaScript required for this book is the deepest of the three. If you're unsure about JavaScript basics along the way, check out Appendix A, "JavaScript Basics," for a reference. Second, we use a lot of external libraries throughout the book, so we want to ensure at this point that you have no trouble bringing them in. Third, we want to establish a comfortable, repeatable pattern for creating, editing, saving, and opening files that are fundamental to using this book.

If you don't have a text editor, you will need one now. You can use any number of tools to create and edit the JavaScript, HTML, and CSS files in this book. If you don't know what text editor is right for you, see Appendix C, "Resources," to get an idea of your options.

To start, open your text editor, and add the code in Listing 1.1 to the quiz/initial/index.html file. If you haven't downloaded the files for this book yet, see the Introduction for details on getting the code.

Listing 1.1 index.html Showing the html Structure

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Quiz</title>
    <link rel="stylesheet" type="text/css" href="main.css">
  </head>
  <body>
    <h1>Quiz</h1>
    <div id="quiz">
    </div>
  </body>
</html>
```

note

HTML stands for HyperText Markup Language. Way back when, *links* were also called *hyperlinks*, and there were a few other *hyper*-related things that amounted to being able to jump from document to document. HyperText is kind of like normal text with these hyperlinks in it. Markup is auxiliary text around the HyperText to give a bit more context. So HTML is a collection of syntax guidelines for combining different types of text to produce linkable pages that end up having an .html extension.

An HTML tag is text that appears in <angle brackets like these>, and an HTML *element* is anything that appears between a <beginning tag> and a </closing tag>, including the beginning and closing tags. Notice the "/" used in the closing tag.

You start by declaring the `DOCTYPE`. This lets the browser know that the document it is parsing and presenting is HTML. There are other file formats that browsers can open, from XML documents to audio files and images, so this makes it clear that you want this file processed as a normal web page. You might be wondering about the consequences of not doing this. The truth is that it depends on what browser you are using, and the effects can be anywhere from subtle to terrible. Effectively, they are unknown, which is a good enough reason not to forget this preamble in your document.

Next, you have an `<html>` tag. This wraps the document with a global container, and its typical contents are one `<head>` tag and one `<body>` tag, just as you have here. You might notice that all three of these tags having ending tags with a slash (for example, `</body>`). This is how you specify what inner elements the element is wrapping.

What goes inside of the `<head>` tag is, generally speaking, information that is important for the browser to know about, but not directly related to what the user sees within the main part of the browser window. The `<meta>` tag serves many purposes. Here it is describing what the text encoding should be for the document. If you don't have this, characters that are outside of a fairly limited (although common) set will be treated in unpredictable ways. Most day-to-day characters for English speakers will render correctly, but if you work with international characters, you could have a problem. Additionally, you see a warning in the JavaScript console (Firebug or Chrome developer tools) saying that you should have this. That said, this tag has been omitted in many cases throughout this book so that you can focus on what is original in each chapter.

The `<title>` tag describes what text displays in the uppermost part of the browser (depending on the browser, this could be the header bar, the tab, or both) and is also often used by applications providing shortcuts and bookmarks to give an "at a glance" idea of the page.

Next is a `<link>` tag that specifies its `rel` attribute as a `stylesheet`, the `type` attribute as a css file, and the `href` to describe the path of where the file is stored on your computer. This path (just the filename) indicates that it is sitting next to index.html in the same directory. You see this tag frequently to link an external style sheet (CSS file), and except for the path, it will stay the same most of the time. One additional thing to note about the `<link>` tag is that it, like the `<meta>` tag, does not have a closing tag (like `</link>`). For tags that are not acting as containers, these are not always necessary.

In the `<body>` tag, you have two nested elements. The first is a header (`<h1>`) tag that has default styling to make the text inside appear larger. The `<div>` tag is a major organizational element for containing chunks of marked-up information. In this case, it has an attribute, `id`, which, along with the tag name and the attribute `class`, forms the three most common ways to specify styles with CSS (for example, new colors and text sizes) and behavior with JavaScript (for example, when this is clicked, turn the page upside-down).

You don't yet have anything in your `<div>` element, but before you start adding anything new, check to see that you're on track. Save your file as index.html, and open a browser. Then, either type the file path for the file into the URL bar, drag it from the desktop into the URL bar, or double-click it.

After you get the file to open in a browser, you should see something like Figure 1.1. Notice that "Quiz" appears as the name of the tab because of the `<title>` tag you specified earlier.

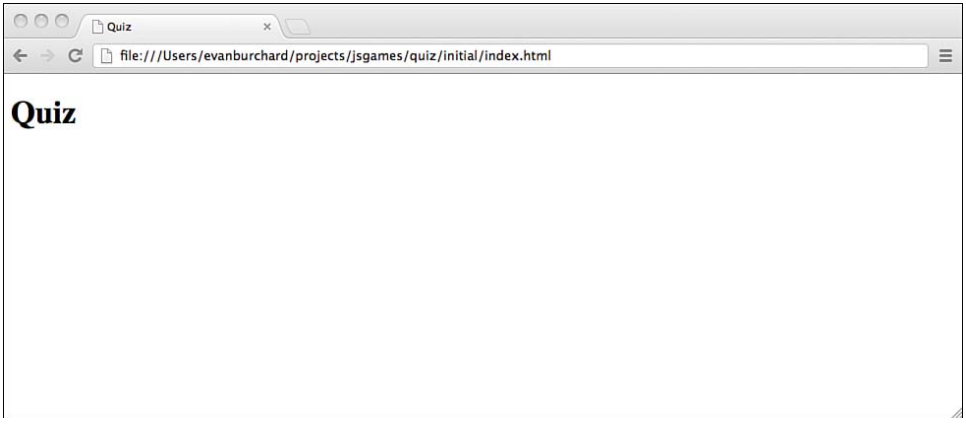

Figure 1.1 Opening the html file in Chrome

If you don't have Chrome or Firefox, you should download both now. They highlight different issues that arise when developing HTML5 games, and you will use both. They are not treated as completely interchangeable throughout this book.

Now head into your tag, and add some questions to your index.html file with the bold code in Listing 1.2. This sample is rather long but regular. If you want to save yourself some typing, this file can be copied from quiz/after_recipe1/index.html.

Listing 1.2 The Questions of the Quiz

```
...
<div id="quiz">
  <div id="question1">
    <div class="question">Which is not a main file type that we
➥use to make websites?</div>
    <input type="radio" name="question1" value="a"/>
    <label>.html</label>
    <input type="radio" name="question1" value="b"/>
    <label>.exe</label>
    <input type="radio" name="question1" value="c"/>
    <label>.js</label>
    <input type="radio" name="question1" value="d"/>
    <label>.css</label>
  </div>
  <br />
  <div id="question2">
    <div class="question">A JavaScript object is wrapped by what
➥characters?</div>
    <input type="radio" name="question2" value="a"/>
    <label>[]</label>
    <input type="radio" name="question2" value="b"/>
    <label>;;</label>
    <input type="radio" name="question2" value="c"/>
    <label>{}</label>
    <input type="radio" name="question2" value="d"/>
    <label>()</label>
  </div>
  <br />
  <div id="question3">
    <div class="question">Moles are which of the following?</div>
    <input type="radio" name="question3" value="a"/>
    <label>Omniverous</label>
    <input type="radio" name="question3" value="b"/>
    <label>Adorable</label>
    <input type="radio" name="question3" value="c"/>
    <label>Whackable</label>
    <input type="radio" name="question3" value="d"/>
    <label>All of the above</label>
  </div>
  <br />
  <div id="question4">
```

```
    <div class="question">In Japanese "か" is prounounced...</div>
    <input type="radio" name="question4" value="a"/>
    <label>ka</label>
    <input type="radio" name="question4" value="b"/>
    <label>ko</label>
    <input type="radio" name="question4" value="c"/>
    <label>ke</label>
    <input type="radio" name="question4" value="d"/>
    <label>ki</label>
  </div>
  <br />
  <div id="question5">
    <div class="question">The gravitational constant on earth is
➥approximately...</div>
    <input type="radio" name="question5" value="a"/>
    <label>10m/s^2</label>
    <input type="radio" name="question5" value="b"/>
    <label>.809m/s^2</label>
    <input type="radio" name="question5" value="c"/>
    <label>9.81m/s^2</label>
    <input type="radio" name="question5" value="d"/>
    <label>84.4m/s^2</label>
  </div>
  <br />
  <div id="question6">
    <div class="question">45 (in base 10) is what in binary
➥(base 2)?</div>
    <input type="radio" name="question6" value="a"/>
    <label>101101</label>
    <input type="radio" name="question6" value="b"/>
    <label>110011</label>
    <input type="radio" name="question6" value="c"/>
    <label>011101</label>
    <input type="radio" name="question6" value="d"/>
    <label>101011</label>
  </div>
  <br />
  <div id="question7">
    <div class="question">4 << 2 = ...</div>
    <input type="radio" name="question7" value="a"/>
    <label>16</label>
    <input type="radio" name="question7" value="b"/>
    <label>4</label>
    <input type="radio" name="question7" value="c"/>
    <label>2</label>
    <input type="radio" name="question7" value="d"/>
    <label>8</label>
```

```
  </div>
  <br />
  <div id="question8">
    <div class="question">Given the lengths of two sides of a
➥right triangle (one with a 90 degree angle), how would you find
➥the hypotenuse?</div>
    <input type="radio" name="question8" value="a"/>
    <label>Pi*Radius^2</label>
    <input type="radio" name="question8" value="b"/>
    <label>Pythagorean Theorem</label>
    <input type="radio" name="question8" value="c"/>
    <label>Calculator?</label>
    <input type="radio" name="question8" value="d"/>
    <label>Sin(side1 + side2)</label>
  </div>
  <br />
  <div id="question9">
    <div class="question">True or False: All games must run at at
➥least 60 frames per second to be any good.</div>
    <input type="radio" name="question9" value="a"/>
    <label>True</label>
    <input type="radio" name="question9" value="b"/>
    <label>False</label>
  </div>
  <br />
  <div id="question10">
    <div class="question">Using a server can help you to...</div>
    <input type="radio" name="question10" value="a"/>
    <label>hide your code.</label>
    <input type="radio" name="question10" value="b"/>
    <label>have a performant game.</label>
    <input type="radio" name="question10" value="c"/>
    <label>create shared experiences for players.</label>
    <input type="radio" name="question10" value="d"/>
    <label>all of the above.</label>
  </div>
</div>
...
```

Each question in this quiz has the same structure but with different indicators of the question number and different question and choice text. Now pretend that you are concerned only with the first question. In this case, you start with a `<div>` tag with the `id` of `question1`. This is a unique identifier that you can use later. This `<div>` tag wraps the entire question and answer block. Next, another `<div>` tag contains the text of the question. This has a `class` of

`question`. Recall that `class`, like tag name (for example, `<div>`) or `id`, is a way to reference this element later. The major difference between `class` and `id` is that an `id` is unique but classes can be shared.

Next, you have an `<input>` tag with three attributes. The `type="radio"` means that it is a radio button. If you don't know what that looks like, see Figure 1.2. The second attribute, `name`, must be unique among all the choices that ask the same question. The `value` stores what is typically passed during html form submissions in a similar way that text entered into a text field gets passed along. You won't be submitting any forms, but you will be using JavaScript later to check the answers on this page via these values. So far, you have seen tags that don't require an ending tag and ones that do. This input tag is a third type that ends with a `/>`, indicating that it is acting as its own closing tag.

The `<label>` tags are used for text that sits outside of an input element. Their main function is to give focus to their complementary input when clicked. You don't implement that here, but if you want that functionality, add a unique `id` to each answer, such as `id="question-10-"answer-b"` and give the corresponding label a `for` attribute like `<label for="question-10-answer-b">`.

Between each question is a break tag (`<br />`) that ends with a slash and acts as its own ending tag. The break tag puts vertical space between things. How much space is not consistent from browser to browser, so in cases in which the layout matters (most but not here), this space should be added with CSS styling instead.

If all went according to plan, if you save and open this file in your browser, you should now see something like Figure 1.2.

Recipe: Hiding and Showing Your Quiz

Games frequently have a notion of unlockables. Some examples are unlockable characters, unlockable side quests, and unlockable levels. Here, you have unlockable questions. It may seem like you're headed backward, but it is only because your content is so simple and perceivable. You would not expect to play every level of a Mario game at once, right? If your quiz were 100 questions instead of 10, you might feel the same way about showing all the questions at once.

So how should you lock down your content? You have many options, including putting questions on separate pages, but for the sake of starting out simple, just add a CSS file that can prevent the content from showing. Next, you need a new file called main.css with the code in Listing 1.3 to sit in the same directory as index.html.

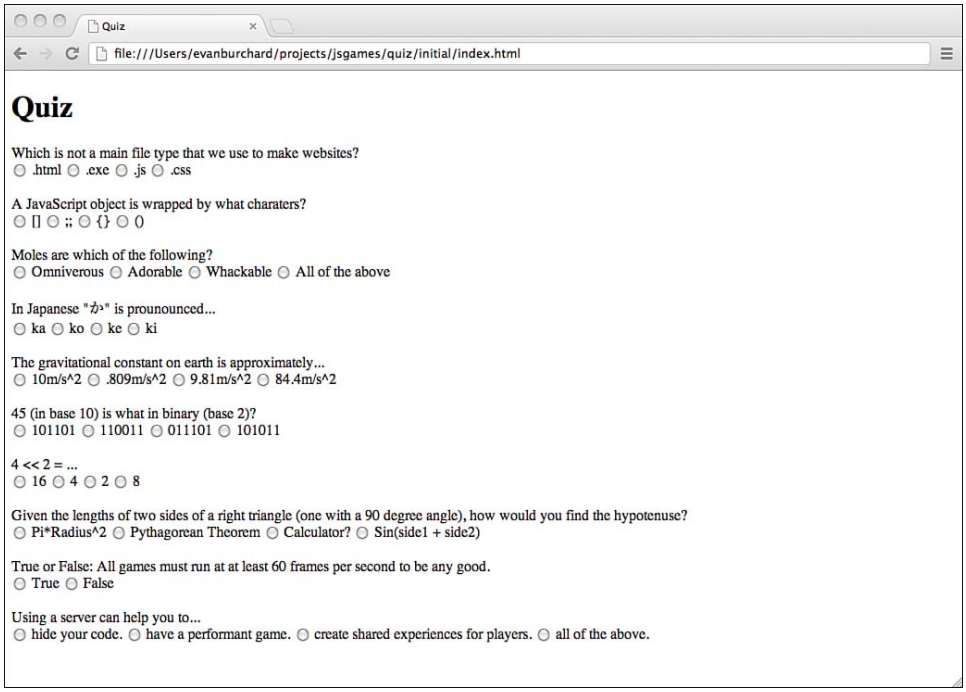

Figure 1.2 The questions and answers for the quiz

Listing 1.3 main.css to Hide the Content

```
#quiz{
  display:none;
}
body{
  margin-left:50px;
}
```

`#quiz` refers to everything inside a container element such as a `div` with the `id` of `quiz`. The `display:none` here is what hides everything inside of the `div` with the `id` of `quiz`. If you were selecting an element with the `id` of `another-quiz`, your style selector would be `#another-quiz`. If you were selecting the elements with a `class` of quiz, you would use a dot instead of a # symbol for your selector, so it would be `.quiz`.

Tag-based selectors have nothing in front of their names, so to select the `body` tag, you don't need a dot or a # symbol. Adding `margin-left:50px;` here bumps the page to the right a little bit. Notice the format of both of these styling blocks. You have a selector, an open brace,

styling information, and then a closing brace. In the styling format, you have the label for the css attribute name on the left, followed by a colon, followed by the value of the css attribute you want to affect, and followed by a semicolon to terminate the line.

If you're just starting out with web development, this syntax can be tricky, especially when paired with html tags, along with their ids, classes, and other attributes being introduced simultaneously. The good news is that you have just covered basics of CSS and HTML. You will be exposed to new attributes and selectors, but the fundamentals are right here. In the short term, if you make a mistake, it's probably something simple like swapping a dot for a # symbol, not terminating with a semicolon or curly brace, or some other form of typo. Truthfully, those are the most common mistakes that professionals make, too. If something isn't working, slow down and carefully read what you have written.

At this point, if you save the files and open index.html in a browser, it will look like you are back at Figure 1.1, but with the indenting from the styling you added to `body`.

Recipe: Getting Your Questions Back

With all your questions gone, you need a way to get them back. You can do this in a roundabout way by adding packages used in each of the later chapters. For each package you add, you'll add a question back in.

Before you ensure that packages can be loaded, first you need to make sure that you can load any JavaScript whatsoever. At almost the bottom of your index.html file, add the bold line in Listing 1.4.

Listing 1.4 Loading Your First External JavaScript File

```
...
    <script src="game.js"></script>
  </body>
</html>
```

This loads the game.js JavaScript file. Next, you need to create that file. Create a file called game.js in the same directory as main.css and index.html and add the code in Listing 1.5.

Listing 1.5 The game.js File

```
alert('Hello world');
console.log('Hello world');
```

This code prints information to two places. If you open the index.html file in your browser, the first place, the alert box, is obvious. The second line with `console.log` prints to the JavaScript console, an indispensable tool for development. If you need more information about getting a JavaScript console up and running, see Appendix B, "Quality Control."

With that out of the way, next let's take on jQuery. To get this, going to jquery.com is probably the fastest route. How you get it from its website into your file system is up to you. The author simply hit the biggest, shiniest button, which led to a page that showed only the text of the code. He then copied and pasted that into a new file that he created and called jquery.js. Then he saved that file.

There are buttons on the site that just give you a file to download in a more traditional way as well. How you do it is up to you, but make sure that you get it into the right directory on your file system (the same one where index.html, main.css, and game.js are located).

After you have the file in the proper directory, add the bolded line in Listing 1.6 to the bottom of the index.html file. Make sure that your filename matches what is listed in this file.

Listing 1.6 Adding jQuery to the index.html File

```
...
    <script src="jquery.js"></script>
    <script src="game.js"></script>
  </body>
</html>
```

If your file is called something other than `jquery.js`, you must change this to load the file properly.

Before we go any further, we need to adjust our CSS file a bit. We were too aggressive before. Instead of hiding the entire quiz, let's be more specific and hide every question individually with the code in Listing 1.7.

Listing 1.7 Hiding the Questions, Not the Quiz

```
body{
  margin-left:50px;
}
#question1, #question2, #question3, #question4, #question5,
#question6, #question7, #question8, #question9, #question10{
  display:none;
}
```

You deleted the `#quiz` id selector from before and replaced it with a comma-separated list of id selectors to affect. Instead of doing it this way, you could have declared a common class for all these elements and used a dot selector. However, using a list of selectors in this way is good to be aware of.

Now that you changed how your hiding works with the CSS acting as the bad guy, use jQuery as the good guy to help you unlock a question. To do this, change your game.js code to that which is found in Listing 1.8. This should replace the code previously written in this file.

Listing 1.8 Showing Your First Question if jQuery Is Loaded

```
if(jQuery){
  $("#question1").show();
};
```

In the first line, you check to see if jQuery is loaded. If it is, you execute the second line. On the second line, you use jQuery's `$` function, passing in the CSS selector `#question1`, surrounded by quotes and parentheses. Then, you execute jQuery's `show` function to change `display:none` to `display:block` for the first question.

If you save the files and load index.html in your browser, you see that your first question is back in action.

Recipe: The Shopping List

In this recipe, you have nine more files that you need to import. You might be wondering why you would hide questions only to show them again if you are able to load a file. Downloading files and including them in other files might seem like a repetitive or unnecessary exercise to many of you, but understanding how to access and leverage other peoples' code is crucial. Few projects are built from scratch, and learning how to build games by "standing on the shoulders of giants" is well worth your time if you don't have this skill yet. In addition, this section gives a preview of what kinds of files you will use in later chapters.

That said, if you are comfortable with integrating JavaScript libraries into a system and have a good understanding of version control, much of what follows will be a review. Feel free to skim or skip this recipe.

GROCERIES

Here are the files that you need, and what they are used for in the book:

1. **jquery.js:** You already have this one. It's used by a few different chapters to easily select and manipulate page elements.
2. **impress.js:** In Chapter 3, "Party," you use this presentation tool (like PowerPoint, but in JavaScript) as a game engine for managing the "pages" of your interactive fiction game.
3. **atom.js:** Weighing in at an uncompressed 203 lines of coffeescript, this is certainly one of the smallest game engines there is. You use this to build the party game.
4. **easel.js:** This provides a nicer interface to the canvas API that we use when exploring how to draw elements for the puzzle game.
5. **melon.js:** This is the engine that you use for Chapter 5, "Platformer."
6. **yabble.js:** When you build the fighting game, this is the library that you use to load the "game.js" game engine (not to be confused with your game.js file in this chapter and others).
7. **jquery.gamequery.js:** This is a jQuery plug-in that is also a game engine. You use it to make the side-scrolling shooter.
8. **jaws.js:** This is an all-around solid game engine that you use (with old-fashioned trigonometry) to build the first person shooter.
9. **enchant.js:** Hailing from Japan, this game engine has a ton of features and great support for mobile. You use this for the RPG in Chapter 9, "RPG."
10. **crafty.js:** This is a fully featured extremely well-supported game engine that you use to build the RTS. (If the author had to pick just one game engine to take to a deserted island, it might be this one.)

So, now that you have the main library that you use in this chapter, jQuery, taken care of, let's go shopping for the rest. If you were feeling particularly adventurous, you could try grabbing them all from their project pages listed in Appendix C, "Resources." Alternatively, they are available in the after_recipe4 section of this chapter's files. Just make sure to put the files in the same directory as your index.html file.

note

If you took a look at Appendix C, you may have noticed that these files are also hosted on github. For getting files from github, you have three options. The first is that you can download the entire project as a zip file, unzip it, and then use the files you need.

The second is that you make your way through the project, click the relevant file, create a new empty file on your computer, and copy and paste the code as it appears on github into your local file. This might seem convoluted, but it can be a fast process.

The third option is a little more complicated but could provide you with a better process while you work, now and in the future. The third option is to install git, use it to download (clone) the project, and head to the chapter's directory to get the files for this chapter. You can work out of this directory or just copy the files you need from it.

git is a "version control system" that enables you to keep track of the changes you make to your files. github is a website where people (many programmers working with many languages) who use this tool can find other projects and manage their own. It is free to use with public projects. The author highly recommends taking this path. The best instructions for installing git are at help.github.com/articles/set-up-git.

After getting all the required files, one way or another, you should end up with a file system containing files so that it looks like Figure 1.3.

With that in place, you can start requiring these JavaScript files by adding the bolded lines in Listing 1.9 to the bottom of index.html.

Listing 1.9 Requiring Your JavaScript Files in index.html

```
<script src="jquery.js"></script>
<script src="impress.js"></script>
<!-- atom needs this to run -->
<canvas></canvas>
<script src="atom.js"></script>
<script src="easel.js"></script>
<script src="melon.js"></script>
<script src="yabble.js"></script>
<script src="jquery.gamequery.js"></script>
<script src="jaws.js"></script>
<script src="enchant.js"></script>
```

```
    <script src="crafty.js"></script>
    <script src="game.js"></script>
  </body>
</html>
```

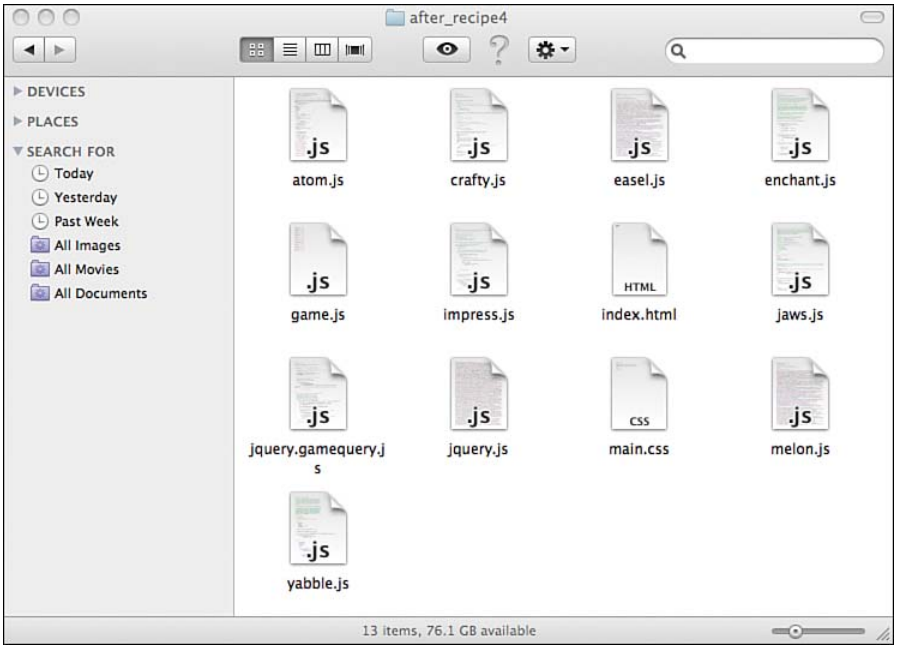

Figure 1.3 The directory with all your JavaScript files in it

Make sure that the names of the files that you are including match the names of your files as referenced in index.html. The process of requiring JavaScript files from html usually follows this simple pattern of using `<script>` tags. One oddity within this section is the `<canvas>` element that you added above atom.js, along with the `<!-- -->` line above that. You add this `<canvas>` element because without it, atom.js cannot work. In most game engines, you must indicate that you want to start the engine by calling an initializing function or referencing a particular `<canvas>` element to work with. atom.js starts hunting for a canvas tag as soon as you require it. Rather than fight it (edit atom.js), just give it what it wants. The `<!-- -->` line is an HTML comment. The idea here is that you can make a note to yourself or others, but the comments are ignored by the browser. Keep in mind that these are still potentially user-facing, and anyone who "views the source" of the HTML page can see them. If you don't know what that means, see Appendix B.

Next, in the game.js file, get the rest of your questions back. You can accomplish this by adding the bold lines in Listing 1.10 to game.js.

Listing 1.10 Getting the Rest of the Questions Back

```
if(jQuery){
  $("#question1").show();
};
if(impress){
  $("#question2").show();
};
if(atom){
  $("#question3").show();
};
if(createjs){
  $("#question4").show();
};
if(me){
  $("#question5").show();
};
if(require){
  $("#question6").show();
};
if($().playground){
  $("#question7").show();
};
if(jaws){
  $("#question8").show();
};
if(enchant){
  $("#question9").show();
};
if(Crafty){
  $("#question10").show();
};
```

Here you can see that, like jQuery, the most common, immediately perceivable effect when importing a JavaScript file is that some object becomes available. One notable exception is the check for `playground` to show question number 7. gameQuery is a jQuery extension, and as such, it provides utilities on top of jQuery. For this reason, it does not come with a core object of its very own, so it looks for the `playground` function to be defined on the jQuery's `$()` instead.