Build the next-gen multiplayer game web app right in your mobile or desktop browser

Pro Android Web Game Apps

Using HTML5, CSS3 and JavaScript

Juriy Bura

Apress®

Pro Android Web Game Apps

Using HTML5, CSS3, and JavaScript

Juriy Bura

Apress®

Pro Android Web Game Apps: Using HTML5, CSS3, and JavaScript

ISBN 978-1-4302-3819-5

ISBN 978-1-4302-3820-1 (eBook)

President and Publisher: Paul Manning
Lead Editor: Chris Nelson
Technical Reviewer: Charles Cruz
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Coordinating Editor: Brigid Duffy
Copy Editor: Kimberly Burton
Compositor: Bytheway Publishing Services
Indexer: SPi Global
Artist: SPi Global
Cover Art By: Sergey Lesiuk
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `www.springeronline.com`.

For information on translations, please e-mail `rights@apress.com`, or visit `www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `www.apress.com/bulk-sales`.

The source code for this book is available to readers at `www.apress.com`. You will need to answer questions pertaining to this book in order to successfully download the code.

To Elena and Alysa

Contents at a Glance

Contents

About the Authors

Juriy Bura is an independent consultant living between Kiev, Ukraine, and Zurich, Switzerland. His main area of expertise is games and real-time web applications for both desktop and mobile platforms. He is a co-owner of Deadline Solutions (`http://deadline-solutions.com/about.html`). Juriy is also a leader of the JavaScript User Group in Ukraine, a frequent conference speaker, and a passionate web developer seeking how to push the browser to its limits. Having more than seven years of experience in Java and JavaScript, he is sure that game development is the area with the most fun concentrated within a line of code.

Juriy spends his spare time with his family and playing board games in a small "geek" club. Juriy blogs at `http://juriy.com` and tweets as `@juriy`.

Paul Coates is a freelance copy editor and EFL teacher, based in Burton Upon Trent, UK, and Kyiv, Ukraine. He made sure that Juriy's ingredients had the right touch of English flavor for publication. When not teaching students of all ages how to speak his native tongue, Paul provides copy editing and proofreading for foreign works in English, as well as Russian and Ukrainian translations. Paul enjoys video gaming, cinema, and traveling.

Paul occasionally blogs at http://psykspopcornjungle.blogspot.com and occasionally tweets at @Psyklax.

Acknowledgments

This book would be impossible without the help and support from many great people. I'm most grateful to my wife, Elena, whose love gave me inspiration when I needed it most, and my parents, Alexander and Vera Bura.

I'm grateful to my good friends: Vadim Voituk for his brilliant ideas and technical expertise; Alexey Kondratov, who was the first to explain to me how a book is different from a 600-page blog; and Artyom Volkhonskiy. I'm so glad that you guys have always been there when I needed your support and opinion. Your help has improved this book tremendously.

A very special thanks to a brilliant artist—Sergey Lesiuk (`http://nitro-killer.deviantart.com`), who created the cover art and graphical assets for the isometric engine, and the guys at Marcus Studio (`www.marcusstudio.com.ua/`) for the fabulous animated knight character. With their help, we now have more free high-quality art to use in our game projects.

Thanks to the Apress team, who did a great job bringing this book to life: Steve Anglin, Brigid Duffy, Charlie Cruz, Kimberly Burton, Anna Ishchenko, Stephen Moles, Jonathan Gennick, Jean Blackburn, and many others working behind the scenes to publish this book.

Personal thanks to Chris Nelson for his review, advice, and for dealing with never-ending changes to the chapters' structure. With your help, this book shaped up. Working with you is a great pleasure.

Thanks to Paul Coates for his invaluable contribution to this book and his readiness to review all those "last minute edits" in the middle of the weekend.

And, of course, thank you, my little angel Alysa. No more "Daddy is working again tonight." I promise.

—Juriy Bura

I want to thank Juriy for a great job, Stephen and Chris at Apress for making our first book a pleasure to write, and most of all Sasha, without whom none of this would be possible.

—Paul Coates

Introduction

This book is about making web games with JavaScript for today's most promising mobile platform—Android. Game development is a challenging subject. Games aim to simulate life in some form or another, and the more realistic you want a simulation to be, the more knowledge and skill you have to apply to make it believable. Video games is the place where mathematics—which is quite typical in programming—meets kinematics, optics, acoustics, artificial intelligence, art, music, and storytelling. Where else can you find a mix like that?

Why JavaScript and HTML5? If you are holding this book in your hands, then you probably already have your answer to that question. If you are curious about *my* reasoning, it's because JavaScript is the most popular cross-platform client-side solution that developers have at their disposal. Every device that has Internet access also has a browser—from desktop computers and smartphones to tablets and set-top boxes. And without a doubt, every browser has JavaScript. An application built with a standard HTML5 stack will run on most devices. You want your game to be fast? You want it on desktops, mobiles, and tablets on Windows, iOS, Linux, and Android? You don't want to rewrite the code for a set of heterogeneous platforms in different programming languages? HTML5 comes to rescue!

The goal of this book is to give you a deep understanding of the algorithms and approaches that stand behind the most common types of games. I prefer this approach to that of streamlined how-to guides that often sacrifice important details in favor of immediate results. While the "how-to" approach might look like a quicker way to get to the goal, it usually leaves readers with knowledge gaps to fill on their own. Of course, this book has plenty of how-to examples in addition to thorough coverage of the underlying concepts.

That's why I couldn't avoid putting some math in the book. Yeah, there are few formulas on the pages. Real gamedev is impossible without fair amount of math. You don't need to have any special knowledge of mathematics beyond what you already know from school to master every subject in this book. If you are already proficient with math, you might find some explanations too obvious—feel free to skip them.

In this book, I deliberately avoided using any existing "Swiss Army knife"–style libraries like jQuery, prototype.js, or Underscore.js because I didn't want the examples to be hard-wired with any of them. While there are many great libraries, every developer has his own preferences. I find library-agnostic code to be the friendliest.

What This Book Is About

This book is about making games for the Android platform with HTML5 and JavaScript. It will guide you from an empty HTML page to a full-blown HTML5 game with animations, sound, endless worlds, and multiplayer support.

The following are among the many things you learn in this book:

- How to draw game elements with the Canvas element; how to use sprites and sprite sheets; and how to capture user input.

- How the exciting world of 3D programming works—including WebGL, one of the most promising APIs for web game development.

- How to create multiplayer games with the help of Node.js—the tool that brings the power of JavaScript to the server.

- How to establish real-time communication between users and let them play against each other in online matches. All of this is possible with JavaScript. You don't need to know any other server-side language to write efficient server-side code!

- How to make computer-controlled characters behave intelligently—have them find their way through the world and make decisions with the help of AI algorithms.

- How to add some neat sound effects.

- How to publish our masterpiece in the Android Market.

This book covers many gamedev algorithms and optimizations, most of which are not limited to JavaScript. Once you learn them, you will be able to quickly master game development on other platforms. Understanding how 3D rendering or pathfinding works will help you to build games for any platform, not just the web.

This book is about making games and writing the most exciting applications in the world—and having real fun while doing so.

What This Book Is Not About

This book is not about web programming in general. I will not cover what HTML is or how HTTP works. I assume that you already know how to write basic JavaScript and embed it into an HTML page. You don't need to be a web development guru, but at the very least, you need understand the language core. Operators, functions, objects, and variables should be familiar to you. If you don't feel comfortable with these concepts, you might want to start with Terry McNavage's *JavaScript for Absolute Beginners* (Apress, 2010).

This book is not about game design—creating levels, building character personalities, or designing economics for the online world. Everything related to the gameplay, story, plot, characters, and game mechanics is out of scope. While these topics are extremely interesting, there are special books devoted to them. One such book that I would recommend is *Game Design: Theory and Practice*, Second Edition, by Richard Rouse III (Jones & Bartlett, 2004).

Who Is This Book For?

This book is for programmers. It will guide you through the technical aspects of creating a game—rendering 2D and 3D graphics, user input, networking, sound, artificial intelligence, and publishing the game on the application market. Every concept explained here is illustrated with

code examples that you can run on your Android smartphone or tablet. I tried to make the book as practical as possible—working code is a very important way to provide a kick-start.

If you are a web developer and you want to learn how to make games for Android devices, this book is for you. You don't need experience with any specific JavaScript library—or even experience making sites for mobile platforms—to get the most out of this book. If you know how to make a personal web page from the scratch with some JavaScript in it, that's about enough to get started.

If you are a game developer who created games for other platforms, and you want to leverage your experience to HTML5 and Android, this book is also for you. If this is the case, some sections might look familiar or even obvious to you. For example, if you have worked with OpenGL from within a Java application, you probably know what a shader is or how to map texture to polygons. Feel free to skip such sections and focus on practical aspects—JavaScript listings and examples that come with the book.

About the Art Files

This book comes with some great art created especially for it by Sergey Lesiuk (isometric tiles and buildings) and the guys at Marcus Studio (an animated knight character). You may use this art in your own projects—free or commercial—without tricky restrictions. The complete license text is distributed with the files.

Free and unrestricted art is very important in the early stages of development. It feels so much better to work on a game that looks like a game rather than a mess of stub graphics. The initiative to share commercial-looking sprites for free was inspired by Daniel Cook on his wonderful web site at `www.lostgarden.com`. I encourage you to join and share your gamedev assets for free—the developer community will be most grateful.

How This Book Is Structured

The book is divided into four parts that we jokingly call "worlds."

2D Worlds

This part of the book is devoted to 2D graphics and the Canvas element. It also gets you started in Chapter 1, "Preparing the Environment," by setting up required tools: the IDE, the web server, Java SDK, and the Android emulator. Once all of these are set, you are ready for action.

Chapter 2, "Graphics in the Browser: The Canvas Element," is where the magic starts. You will learn how to render shapes on HTML5 Canvas, how to use paths and curves, gradients and fills, transformations, and states of the 2D context.

In Chapter 3, "Creating the First Game," you create your first project—the Four Balls game. This small project uses elements you created in Chapter 2 and illustrates important, basic game development concepts, such as game state, mechanics, turn validation, and win/lose conditions.

Modern games are impossible without colorful animations. Chapter 4, "Animation and Sprites," guides you through the process of loading the images and drawing a running character frame by frame. You will also learn more advanced animation effects, such as interpolation, acceleration, deceleration, and easing functions.

Chapter 5, "Event Handling and User Input," will introduce you to the methods of working with input in your game. You'll learn how to capture browser events and build a high-level API for complex input models. We'll explore drag-and-drop and pixel-perfect picking with color masks.

At this point, you will be able to create your own simple games, as you will have all the "starter tools" under your belt. So it is time to move to the more advanced topics—rendering game worlds.

In Chapter 6, "Rendering Virtual Worlds," you'll learn how to render Really Big Worlds. We start with the simplest tile-map technique and gradually optimize it. You learn how to cache the fragments of the map, how to use the offscreen buffer, and render the world objects such as trees and rocks.

Chapter 7, "Making an Isometric Game," is the longest chapter of the book. It is devoted to isometric 2D game engines. The isometric view is the most popular way to represent the game world in strategy games, RPGs, tactics, and many other popular genres. You will learn about isometric projection, the shape of tiles, and the ways to render them. In addition to techniques described in Chapter 6, we'll introduce more rendering optimizations—the dirty rectangles algorithm and clustering of world objects. The result of this chapter is our second big project—an isometric engine ready to be used in the next strategy game or RPG.

3D Worlds

The "3D Worlds" part introduces 3D graphics—from the basic rendering concepts to WebGL.

In Chapter 8, "3D in a Browser," we learn what 3D is, how it works, and the math behind it.

Chapter 9, "Using WebGL," is devoted to WebGL—a very promising web standard that is making its way into the mobile world. You'll learn how to initialize WebGL, write shaders, work with geometry data, load textures, and work with 3D models.

Connecting Worlds

"Connecting Worlds" is all about communication and talking to the server. We start with learning Node.js and the Express framework in Chapter 10, "Going Server-Side." This chapter ranges from Node installation to a simple game server with proper templates, session handling, logging, error handling, and notifications.

In Chapter 11, "Talking to the Server," we move back to the client-side and learn how to connect to a server from a web page and exchange data with other players. We will look at different ways of communication, often called transports, and learn their pros and cons.

In Chapter 12, "Making Multiplayer Games," you make your third big project—the multiplayer version of Four Balls—with Node.js, Express, and Socket.IO.

Improving Worlds

The final part is devoted to various small aspects of game development.

Chapter 13, "AI in Games," is about artificial intelligence—breathing life into computer-controlled opponents. You will learn basic approaches to pathfinding and decision making—a good start to making bots look intelligent.

Chapter 14, "JavaScript Game Engines," discusses game engines and introduces Crafty.js—a small yet quite powerful game engine written in JavaScript. Here's where you complete a fourth project—an Escaping Knight game.

Chapter 15, "Building Native Applications," explains what it takes to publish an HTML5 game as the native application to the Android Market. We will go through all steps of the process—packaging the game, signing it with the key, preparing it for market, and publishing—and then update the game to the next version.

Chapter 16, "Adding Sound," adds the final touch to the game—sound. In this chapter, you'll use SoundManager2 to load and play sounds in the Escaping Knight game. You will learn how to loop background MP3s, play "click sounds," and notify the user about game events.

Appendix

Appendix A, "Debugging Web Applications," explains how to debug JavaScript games. We try hard to write good code, but we're all human—mistakes are unavoidable. This appendix will give you a good understanding of how to find bugs and quickly eliminate them, saving more time for development.

Contacting the Author

If you have any questions, suggestions, comments, or ideas regarding this book or HTML5 game development in general, I'd be happy to receive your feedback via e-mail at `juriy.bura@gmail.com`, my web site at `http://juriy.com`, or on Twitter at `@juriy`.

Chapter 1

Getting Started

The goal of this chapter is to prepare a comfortable workspace for development. The environment and tools for mobile development are always a little more complicated than regular desktop projects. When it comes to Android *and* JavaScript in a single application, the right tools in the right place can make a huge difference. But a workspace is not only about tools. It is also very important to set up coding standards and best practices to follow during the development process. Coding conventions and basic architectural decisions are also discussed in this chapter.

Being a seasoned developer, you already have certain preferences in tools and coding approaches. For example, every web programmer has his favorite integrated development environment (IDE) and browser for basic testing. You probably also have your own vision on writing good and maintainable JavaScript code. If you are comfortable with your preferences, use them. At the very least, I encourage you to try the tools and techniques that are described in this chapter. You might find some of them more convenient.

This chapter is divided into two parts—tools and techniques—each describing its own important aspect of development. In this chapter, we will do the following:

- Tools:
  - Install Java Development Kit
  - Compare IDEs with good support of JavaScript
  - Install web server (nginx in the first part of the book)
  - Install Android SDK and configure the emulator

    - Create a basic web page and make sure that it loads in a desktop browser, a real device, and the emulator
- Techniques:
    - Review the JavaScript best coding practices
    - Implement a simple inheritance mechanism that will be used for OOP code throughout the book

Tools

In this section, we review and set up tools that are required to build JavaScript applications. JavaScript is a mature platform that is used to create complex state-of-the-art software. Naturally, there are a lot of software components that help to create, test, and debug rich JavaScript pages.

JavaScript is a dynamic language, unlike Java or C++. The major difference between static and dynamic languages is that a static language must define the data structures *before* runtime. In a static language, for example, a programmer who wants to create a class called `Van` has to explicitly describe all the properties and methods that it has: `color`, `maxSpeed`, `drive()`, and so forth. Every object of the `Van` type has the same strictly defined interface. No surprises here.

Dynamic languages like JavaScript allow adding, removing, or changing the structure of any class or object *at* runtime. So, tricks like the following are possible and valid:

```
var van = getTheRandomVan();
van.drive = racingCar.makeUTurn; // valid assignment of the new property
```

Just like that, you can take the method from the object of a different class and use it instead of the existing method. In this case, only one instance of `van` is affected, the rest of the objects stay intact!

As you can see, a lot of things can happen with the JavaScript data structures at runtime: the variables can change their types, and existing objects can be extended with the new methods using the local code or the code that was downloaded from the remote server via Ajax call.

The dynamic behavior gives extreme power to the language, but makes it way harder for tools like IDEs to predict the structure of the objects and their types. The dynamic nature of JavaScript prevents code analyzers from helping you in the same way they help with the “classic” static-typed languages.

What We'll Need

Since we are going to make games for the web, we need to set up a small web-like infrastructure that mimics a real environment. As a bare minimum, we need the following three components:

- An integrated development environment (IDE)
- A web server to serve static files: HTML pages, JavaScript files, images, and others
- A device emulator or a real device to test the product

This list is far from complete, of course, but it's a good start.

The goal of this section is to create a plain "Hello World" HTML page that can be viewed with an emulator, a real device, and a desktop browser. Once you see that this setup works, you can forget about the environment and focus on writing applications.

Almost every tool, except for the emulator, gives you some options. For example, there's no "best" IDE for JavaScript and there are around a dozen popular web servers that are good at serving static files. Once you get a basic setup going, feel free to experiment with individual components, and fine-tune them.

NOTE: When I write about software products, I often mention prices and versions. I think this information is useful. It is nice to know upfront how much you are expected to invest in a tool. This kind of information is, of course, subject to change and should be read as the "price at the time of writing." For the most up-to-date information, please refer to the respective companies' web sites.

NOTE: Everybody who writes code makes mistakes. No matter how experienced you are, if you are human, you will eventually introduce bugs in your program. Debugging is part of the process, just like development, and not the easiest part I must admit. Debugging tools are also very important to set and use. But this topic is outside the scope of this chapter. A more in-depth discussion about hunting bugs in mobile-oriented code is found in Appendix A.

Environment Variables

Most tools that we use in this book follow the same installation pattern:

- Install the tool or extract archive
- Set environment variable `TOOL_HOME`, for example, `JAVA_HOME` or `ANT_HOME`
- Add the folder with executables, usually `TOOL_HOME/bin` to the `PATH` so that you can call the tool straight from the console or terminal without typing the whole path

Setting and changing environment variables depends on your OS.

Windows 7

In Windows 7, right-click My Computer and select Properties from the drop-down menu. In the opened window, click Advanced System Settings ➤ Environment Variables. Under System Variables, click New. Enter a variable name (for example, `JAVA_HOME`), the variable value, and then click OK. Note that you have to reopen any opened console windows to make them "see" the changes.

Adding certain folders to `PATH` works the same way. Find the variable called `PATH` in the environment variable list and click Edit. Put the cursor at the end of the line, add a semicolon (;), and type the path to the folder. Usually when you add the new tool to the path, you refer the existing variable, as follows:

```
;%JAVA_HOME%\bin
```

To check that the variable is set correctly, open a new console window and execute:

```
> echo %JAVA_HOME%
```

Use the name of your variable instead of `JAVA_HOME`, of course. You should see the path immediately printed in the console window.

Mac OS X Lion

In Mac OS X Lion, first create the file called `.bash_profile` in your home folder. Open the terminal window and execute:

```
$touch ~/.bash_profile
$ open -e ~/.bash_profile
```

The file should now be opened in the text editor. Add the following line for every environment variable that you want to create:

```
export TOOL_HOME=/path/to/tool
```

For example:

```
export NODE_PATH=~/node
export ANDROID_HOME=~/android
```

The last line of this script should be the line that updates the `PATH` variable:

```
export PATH=$PATH:$ANDROID_HOME/tools:$NODE_PATH
```

`PATH` is a colon-separated list of paths where Mac OS looks for programs when you call them from the terminal without providing the exact location of the executable file. Save the `.bash_profile`, go back to the terminal window, and execute the following to reload the newly defined variables:

```
$ . .bash_profile
```

Now check that the variables are available. Type the following:

```
$ echo $VARIABLE_NAME
```

Use your own variable instead of `VARIABLE_NAME`, of course. You should immediately see the path defined for this variable. If you ever need to edit one of the variables, edit the file and change the value appropriately.

Paths

When you work with tools, you are often asked to enter different kinds of paths. In this book, I usually refer to them explicitly, like “IDE installation path” or “the project path” (meaning the path to your project folder). It is easy to understand what it means most of the time. There are cases, however, when I can’t use that kind of explanation; for example, the screenshots usually show a certain state of the program, and if I take the screenshot from my system, it shows my paths, of course. Code listings and config files sometimes refer to paths too; in this case, I also use the real paths that I use for development.

The paths are different for Windows users and Mac users, but since the software that we use is mostly cross-platform, any path format can be used. Most tools accept the forwardslash(/) in Windows paths for distinguishing between the path separator and escape characters. If you try to use code like `var path = "c:\nginx"`, for example, it will not work correctly. The `\n` will be treated as the escape sequence and transformed to the “new line” character. This issue is not JavaScript specific; most programming languages use the backslash(\ ) to denote that the character standing after it will be processed in a

special way. You must use either "c:\\nginx" or "c:/nginx" to specify the valid path, and both strings will work fine. I recommend using the second approach (it is easier to read at least). The cases when you need to specify the absolute paths in JavaScript are quite rare and usually relate to server-side development.

Personally, I like to use the c:\apps folder for the development-related tools like IDEs, development kits, emulators, and everything else that can be executed. I keep my projects in c:\apps\projects. You are free to use your own conventions, of course. Just keep in mind that when you see a path like c:\apps\projects\myproject in this book, you have to use your own value instead.

The other good reason to use the real paths in listings is that you know the expected format straightaway. There are many ways to specify the location of a file or folder in a file system: absolute, relative, or in the form of URI (with file:// protocol). The relative paths may be calculated either from the current working folder or from the folder where the currently executing file is located, and so forth. It is often good to see a real example rather than a placeholder like %YOUR_PROJECT_PATH%.

Java Development Kit

Java Development Kit, or simply JDK, is an essential part of Android development, even if you don't plan to write a single line of Java code. The Android emulator requires Java to run and some JavaScript IDEs require it too. JDK is a set of tools used to compile and run programs written in Java. We will not use JDK directly in this book, but several components that we will need require it.

For windows users, JDK can be downloaded from the official site at www.oracle.com/technetwork/java/javase/downloads/index.html (click Java on this page), select the version according to your OS, and install it. Once this is done, you will have to set the environment variable called JAVA_HOME to let other programs know where they can find Java. Also, add JAVA_HOME/bin to PATH.

For Mac users, open the terminal and type:

```
$ java -version
```

If you see the version number, then JDK is already installed. Otherwise, you will be prompted to install the best available package. Type the same command once again after the installation to make sure that JDK is ready. The installation path on Mac OS X 10.7.x is:

```
/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home
```

It might be slightly different, depending on the particular Java version. Create the new variable called `JAVA_HOME` and point to this folder.

That's it. You have just installed Java and you're ready for more exciting things.

Integrated Development Environment

Sometimes JavaScript projects are as simple as a couple of scripts that can hide fields from an HTML form or load some content with Ajax. In this case, you can get by with a text editor—it loads faster than an IDE, has a simple interface, and saves a lot of memory (IDEs are really memory-hungry these days). For tiny projects all you need is a syntax highlighter to make your code look pretty and to save you from trivial typos.

For anything bigger than that, an IDE is essential. You will need a set of advanced features that a typical text editor doesn't have: good code analysis, inspections, checking for potential errors, autocompletion, refactoring tools, integration with version control systems, bug trackers, and many others.

As I mentioned already, there's no perfect IDE in the market. Some are good for JavaScript while others are not. They differ in price, system requirements, supported platforms, and featuresets. When choosing an IDE, it is most important that you feel comfortable with it. The first steps with a new IDE might seem hard, but if you feel like you're struggling with each line of code even after a couple of weeks, you should try other products. The increase in productivity will most likely make up for the lack of a feature or two.

If you've worked with JavaScript before, you might have picked a favorite IDE already. If not, then the following are a couple of options:

- **IntelliJ Idea** (`www.jetbrains.com/idea/`) has good support for the whole web stack: HTML, CSS, JavaScript, and server-side languages like PHP and Java. If your project is open-sourced, IntelliJ Idea is free—otherwise you will have to pay around $200 for it. IntelliJ has several lightweight IDEs derived from Idea. WebStorm is the one for HTML and JavaScript, and it is only $69.
- **Aptana Studio** (`http://aptana.com`) is based on the glorious and powerful Eclipse project (`www.eclipse.org`). It is extremely feature-rich, and has a plug-in for virtually anything from exploring databases and building enterprise reports to reminding you that your tea is ready. Aptana is free and open source.

The choice between the two usually comes down to one's own preference. There is an army of Eclipse fans and a similar army of IntelliJ fans, which tend to start a holy war each time one side releases a new version. If you're in doubt, try both and choose the IDE you like best. Next, I give you a brief look at these IDEs and demonstrate how they work by making a Hello World project in each of them.

IntelliJ Idea

Download the installer from the official site (`www.jetbrains.com/idea/download/index.html`) and launch it. Follow the regular installation process (there are no odd questions here; IntelliJ Idea only wants to know an installation folder location).

After the installation has completed and you launch IntelliJ Idea for the first time, you will need to choose which plug-ins to enable. If you plan to use IntelliJ as your IDE, it is better to review the lists and select only the plug-ins that you will really use. A smaller number of enabled plug-ins improves the startup time. Otherwise, just leave all the checkmarks with default values. Finally, you see the Welcome screen shown in Figure 1-1; it has several rows of buttons. Click Create New Project to see the New Project window, and choose Create Project from Scratch.

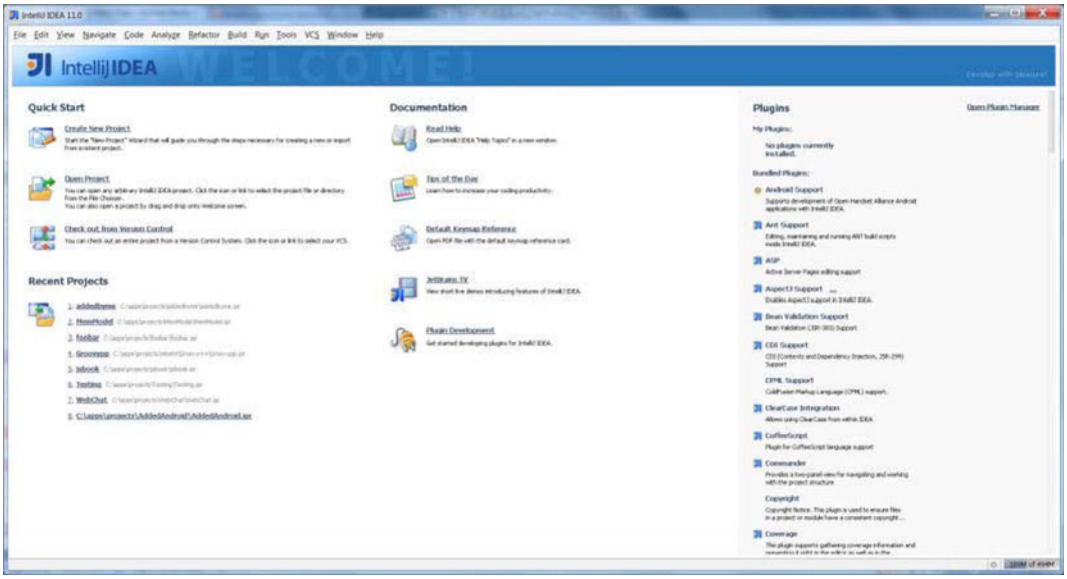

Figure 1-1. *IntelliJ Idea welcome screen*

IntelliJ Idea treats a "project" as a set of one or more modules. For example, if you write a chat application, the "chat application" as a whole is the project. The modules of this project could be Server, Android Client, Desktop Client, and so forth. The idea behind the modules is to separate the different components of the projects since they might have different dependencies or build steps, or they may use different programming languages. The project doesn't have to use many modules, of course. For a simple application, one module is enough.

Each module has a type: Java, J2ME, Android, Grails, and the most important type of module for this book—a Web Module. Select it from the list on the left, as shown in Figure 1-2. If you decide to use IntelliJ Idea as your main IDE, use these steps for every new project that you make.

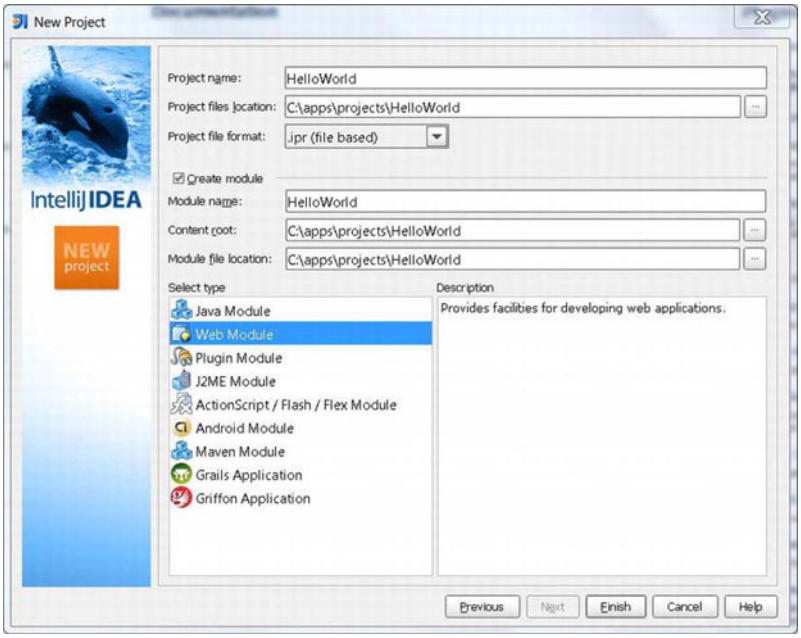

Figure 1-2. *Creating a new project*

Enter the project name and the location you wish to use for project files, and then click Finish. Your project is created and you are presented with a blank workspace, as shown in Figure 1-3.

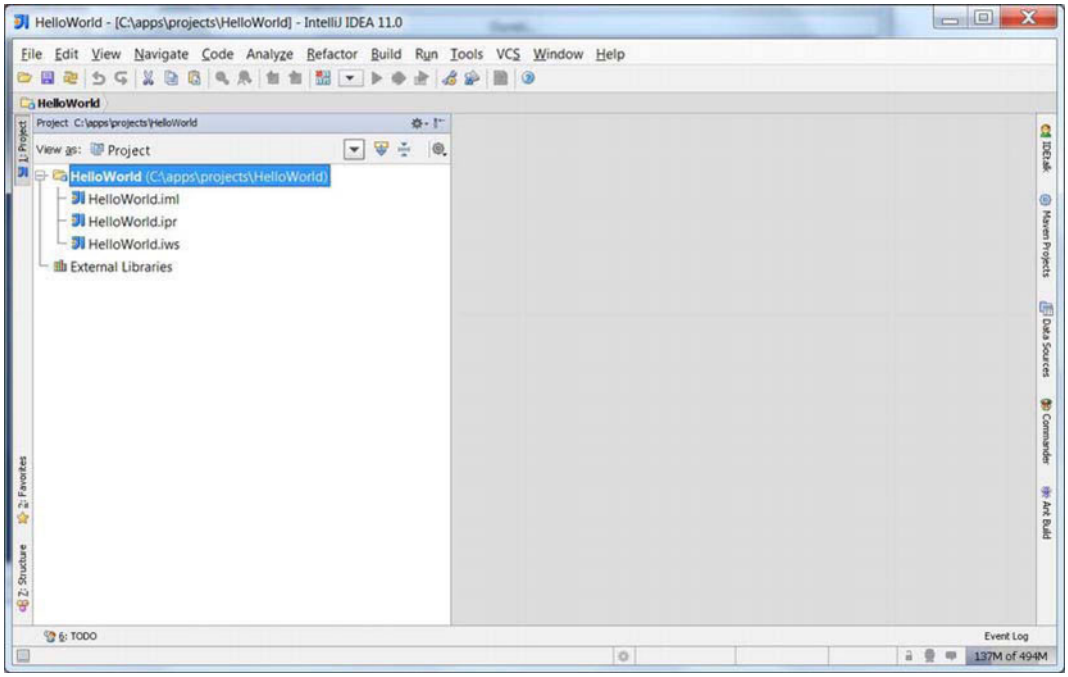

Figure 1-3. *The look of the blank new project*

Now you can write the Hello World page. In our simple example, we have only one module—HelloWorld. Right-click the folder icon with this name in IDE, and then select New File. Enter `index.html` in the dialog and press Enter. Idea creates a new empty file and you can start typing right away. Enter the code from Listing 1-1.

Listing 1-1. *Basic HTML5 Page*

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Hello World</title>
</head>
<body>
        It Works!
</body>
</html>
```

Open the newly created file in your favorite desktop browser and make sure that it renders the page. We still cannot open this file with a mobile device or in an emulator since both of them need the file to be accessible via HTTP. Neither a