Chapter 1
Gaming on the Web

In This Chapter

- Finding out what HTML5 is and where it came from
- Seeing HTML5 within the context of games
- Looking at important new features
- Enabling feature detection and dealing with legacy browsers

BEFORE I DIVE into code, I want to establish the context of the technology we use. In this first chapter, I discuss what HTML5 is as well as some of the history that led to the HTML5 specification.

One of the most interesting aspects of HTML5 is how game developers can profit from many of its new features. In this chapter, I introduce you to some of those features and give you a few quick examples of how to use them. I talk about the `canvas` element and WebGL and the huge improvement they make in creating dynamic graphics. I also cover the `audio` element and the added multiplayer possibilities created by the WebSocket specification.

Everybody likes new toys, but remember that in the real world, old and outdated browsers keep many users from taking advantage of these cutting-edge features. In this chapter, I discuss a few tools that can help you detect which features you can safely use as well as how you can use these feature tests to load appropriate fallback solutions when necessary.

Finally, I briefly introduce the puzzle game that I use throughout the rest of the book to take you through the creation of a complete HTML5 game.

Tracing the History of HTML5

HTML, the language of the web, has gone through numerous revisions since its invention in the early 1990s. When Extensible Markup Language (XML) was all the rage around the turn of the millennium, a lot of effort went into transforming HTML into an XML-compliant language. However, lack of adoption, browser support, and backward compatibility left the web in a mess with no clear direction and a standards body that some felt was out of touch with the realities of the web.

When the W3C finally abandoned the XHTML project, an independent group had already formed with the goal of making the web more suitable for the type of web applications you see today. Instead of just building upon the last specification, the Web Hypertext Application Technology Working Group (WHATWG) began documenting existing development patterns and non-standard browser features used in the wild. Eventually, the W3C joined forces with the WHATWG. The two groups now work together to bring new and exciting features to the HTML5 specification. Because this new specification more closely reflects how web developers already use the web, making the switch to HTML5 is easy, too. Unlike previous revisions, HTML5 doesn't enforce a strict set of syntax rules. Updating a page can often be as easy as changing the document type declaration.

But what is HTML5? Originally, it referred to the latest revision of the HTML standard. Nowadays, it's harder to define; the term has gone to buzzword hell and is now used to describe many technologies that aren't part of the HTML5 specification. Even the W3C got caught up in the all-inclusiveness of HTML5. For a brief period, they defined it as including, for example, Cascading Style Sheets (CSS) and Scalable Vector Graphics (SVG). This only added to the confusion. Fortunately, the W3C backed away from that stance and went back to the original, stricter definition that refers only to the actual HTML5 specification. In a somewhat bolder move, the WHATWG simply dropped the numeral 5, renaming it simply *HTML*. This actually brings it much closer to reality, in the sense that specifications such as HTML are always evolving and never completely supported by any browser. In this book, I just use the term *HTML* for the most part. You can assume that any mention of HTML5 refers to the actual W3C specification called HTML5.

Using HTML5 for Games

Many features from the HTML5 specification have applications in game development, but one of the first features to gain widespread popularity was the `canvas` element. The visual

nature of this element without a doubt helped it spread quickly when the first interactive animations and graphics effects started appearing. More advanced projects soon followed, giving the new standard a dose of good publicity and promising a future with a more dynamic and visually interesting web.

Canvas

Hobbyist game developers were also among the first to embrace HTML5, and for good reason. The `canvas` element provides web game developers with the ability to create dynamic graphics, giving them a welcome alternative to static images and animated GIFs.

Sure, people have created more or less ingenious (and/or crazy) solutions in lieu of better tools for creating dynamic graphics. Entire drawing libraries rely on nothing more than colored `div` elements—that may be clever, but that approach isn't sufficient for doing anything more than drawing a few simple shapes.

Uniform Resource Identifier (URI) schemes let you assign source files to `img` elements, for example, using a base64-encoded data string, either directly in the HTML or by setting the `src` or `href` property with JavaScript. One of the clever uses of this `data` URI scheme is to generate images on the fly and thus provide a dynamically animated image, which isn't a great solution for anything but small and simple images.

Wolf 5K, the winner of the 2002 The 5K contest, which challenged developers to create a website in just five kilobytes, used a somewhat similar technique. The game, a small 3D maze game, generated black and white images at runtime and fed them continuously to the image `src` property, relying on the fact that `img` elements can also take a JavaScript expression in place of an actual URL.

Graphics drawn on a `canvas` surface can't be declared with HTML markup; instead, they must be drawn with JavaScript using a simple Application Programming Interface (API). Listing 1-1 shows a basic example of how to draw a few simple shapes. Note that the full API provides much more functionality than the small portion shown in this example.

Listing 1-1 Drawing shapes with the canvas API

```
<canvas id="mycanvas"></canvas>
<script>
    var canvas = document.getElementById("mycanvas"),
        ctx = canvas.getContext("2d");

    canvas.width = canvas.height = 200;

    // draw two blue circles
```

continued

Listing 1-1 continued

```
    ctx.fillStyle = "blue";
    ctx.beginPath();
    ctx.arc(50, 50, 25, 0, Math.PI * 2, true);
    ctx.arc(150, 50, 25, 0, Math.PI * 2, true);
    ctx.fill();

    // draw a red triangle
    ctx.fillStyle = "red";
    ctx.beginPath();
    ctx.moveTo(100, 75);
    ctx.lineTo(75, 125);
    ctx.lineTo(125, 125);
    ctx.fill();

    // draw a green semi-circle
    ctx.strokeStyle = "green";
    ctx.beginPath();
    ctx.scale(1, 0.5);
    ctx.arc(100, 300, 75, Math.PI, 0, true);
    ctx.closePath();
    ctx.stroke();
</script>
```

The code produces the drawing shown in Figure 1-1.

FIGURE 1-1: This simple canvas drawing was created with JavaScript.

I revisit the canvas element in Chapter 6 and explore it in detail when I use it to create game graphics and special effects.

Audio

The new audio element is just as welcome to a web game developers' toolbox as the canvas element. Finally, you have native audio capabilities in the browser without resorting to plug-ins. Not too long ago, if a website had audio, some form of Flash was involved. Libraries like the SoundManager 2 project (www.schillmania.com/projects/soundmanager2) provide full JavaScript access to most of the audio features of Flash. But even if such a bridge allows your own code to stay on the JavaScript side, your users still need to install the plug-in. The HTML5 audio element solves this problem, making access to audio available in browsers out of the box using only plain old HTML and JavaScript.

The audio element still has a few unresolved issues, however. The major browser vendors all seem to agree on the importance of the element and have all adopted the specification, but so far they've failed to agree on which audio codecs should be supported. So, while the theory of the audio element is good, reality has left developers with no other option than to provide audio files in multiple formats to appease all the browser vendors.

The audio element can be defined in the mark-up or created dynamically with JavaScript. (The latter option is of more interest to you as an application and game developer.) Listing 1-2 shows a basic music player with multiple source files, native user interface (UI) controls, and a few keyboard hotkeys that use the JavaScript API.

Listing 1-2 A simple music player with HTML5 audio

```
<audio controls id="myaudio">
    <source src="Prelude In E Minor, Op. 28.ogg"/>
    <source src="Prelude In E Minor, Op. 28.mp3"/>
</audio>
<script>
    var audio = document.getElementById("myaudio");
    document.onkeydown = function(e) {
        if (e.keyCode == 83) {
            audio.pause(); // Key pressed was S
        } else if (e.keyCode == 80) {
            audio.play(); // Key pressed was P
        }
    };
</script>
```

TIP The W3C is currently working on expanding HTML5 with the Web Audio API, which enables advanced audio synthesizing and processing. Because this API is still experimental, I won't be using it for the game in this book, although I briefly examine the possibilities it presents in Chapter 10 when I dive into HTML5 audio.

WebSockets

Ajax and the `XMLHttpRequest` object at its heart brought new life to the web with the Web 2.0 explosion in the early 2000s. Despite the many great things it has enabled, however, it is still painfully limited. Being restricted to the HTTP protocol, the action is rather one-sided, as the client must actively ask the server for information. The web server has no way of telling the browser that something has changed unless the browser performs a new request. The typical solution has been to poll the server repeatedly, asking for updates, or alternatively to keep the request open until there is something to report. The umbrella term *Comet* (`http://en.wikipedia.org/wiki/Comet_(programming)`) is sometimes used to refer to these techniques. In many cases, that is good enough, but these solutions are rather simple and often lack the flexibility and performance necessary for multiplayer games.

Enter WebSockets. With WebSockets, you're a big step closer to the level of control necessary for efficient game development. Although it isn't a completely raw socket connection, a WebSocket connection does allow you to create and maintain a connection with two-way communication, making implementation of real-time multiplayer games much easier. As Listing 1-3 demonstrates, the interface for connecting to the server and exchanging messages is quite simple.

Listing 1-3 Interacting with the server with WebSockets

```
// Create a new WebSocket object
var socket = new WebSocket("ws://mygameserver.com:4200/");

// Send an initial message to the server
socket.onopen = function () {
    socket.send("Hello server!");
};

// Listen for any data sent to us by the server
socket.onmessage = function(msg) {
    alert("Server says: " + msg);
};
```

Of course, using WebSockets requires that you also implement a server application that's compatible with the WebSockets protocol and capable of responding to the messages you send to it. This doesn't have to be a complex task, however, as I show you in Chapter 13 when you build a simple chat application using WebSockets and Node.js.

Web Storage

Cookies are the usual choice when web applications need to store data on the client. Their bad reputation as spyware-tracking devices aside, cookies have also given developers a much-needed place to store user settings and web servers a means of recognizing returning clients, which is a necessary feature for many web applications because of the stateless nature of the HTTP protocol.

Originally a part of HTML5 but later promoted to its own specification, Web Storage can be seen as an improvement on cookies and can, in many cases, directly replace cookies as a larger storage device for key-value type data. There is more to Web Storage than that, however. Whereas cookies are tied only to the domain, Web Storage has a local storage that's similar to cookies and a session storage that's tied to the active window and page, allowing multiple instances of the same application in different tabs or windows. Unlike cookies, Web Storage lives on only the client and isn't transmitted with each HTTP request, allowing for storage space measured in megabytes instead of kilobytes.

Having access to persistent storage capable of holding at least a few megabytes of data comes in handy when you want to store any sort of complex data. Web Storage can store only strings, but if you couple it with a JavaScript Object Notation (JSON) encoder/decoder, which is natively available in most browsers today, you can easily work around this limitation to hold structures that are more complex. In the game that you develop during the course of this book, you use local Web Storage to implement a Save Game feature as well as to store local high score data.

Listing 1-4 shows the simple and intuitive interface to the storage.

Listing 1-4 Saving local data with Web Storage

```
// save highscore data
localStorage.setItem("highscore", "152400");

// data can later be retrieved, even on other pages
var highscore = localStorage.getItem("highscore");

alert(highscore); // alerts 154200
```

WebGL

WebGL is OpenGL for the web. It's based on OpenGL ES 2.0. The most widely used graphics API is now available for web developers to create online 3D graphics content. Of course, this has major implications for the kind of web games that are now possible. As a testament to this significance, Google developers released a WebGL port of the legendary first-person shooter, "Quake II," on April 1, 2010, to general disbelief because of both the carefully chosen release date and the achievement itself.

REMEMBER Using WebGL requires you to be very aware of the platforms you plan to target. Neither Android nor iOS currently supports WebGL, limiting you to desktop browsers. Furthermore, Internet Explorer prior to IE 11 supports only the 2D canvas context. In this book, I show you how to create the game graphics using both WebGL and 2D canvas.

HTML5 and Flash

Ever since the arrival of the `canvas` element and the improved JavaScript engines, the Internet has seen discussions and good old flame wars over whether the new standards would replace Flash as the dominant delivery method for multimedia applications on the web. Flash has long been the favorite choice when it comes to online video, music, and game development. Although competing technologies such as Microsoft's Silverlight have tried to beat it, they've made only a small dent in the Flash market share. HTML5 and its related open technologies now finally look like a serious contender for that top spot.

Adobe, the company behind Flash, has also shifted its priorities toward HTML5 and recently released Adobe Edge (`http://html.adobe.com/edge`), a development environment very similar to Flash but based fully on HTML5, CSS3, and JavaScript. Add to this the fact that Adobe Flash Professional CS6 introduced the option to publish directly to HTML5, and it seems all but certain that the proprietary Flash format will be phased out in favor of HTML5 and the open web.

REMEMBER HTML5 isn't a drop-in replacement for Flash. You must know where the new standards fall short and when alternative solutions like Flash might be more appropriate. Flash is still very handy for ensuring backward compatibility with older browsers, which I talk about next.

Creating Backward Compatibility

As with most other new technologies, issues with backward compatibility inevitably show up when working with HTML5. HTML5 isn't one big, monolithic thing: Browsers support *features*, not entire specifications. No browsers today can claim 100 percent support for all of the HTML5 feature sets, and Internet Explorer, still the most widely used browser, has only recently caught up with the rest of the browsers with features such as WebGL.

However, even if the current crop of browsers fully supports HTML5 and the related standards, you still have to think about legacy browsers. With browsers like Internet Explorer 8 still seeing significant use today you can't safely assume that the users of your applications and games can take advantage of all the features of HTML5 for many years to come. I recommend using the CanIUse website (`http://caniuse.com/`), which keeps tabs on most features and their past, current, and future browser support. A similar site, Mobile HTML5 (`http://mobilehtml5.org/`) focuses on feature support in mobile browsers.

Using feature detection

No one says that the applications and games you build today must support all browsers ever released—doing so would only lead to misery and hair-pulling. You shouldn't just forget about those users, though. The least you can do is try to tell whether the user is able to play the game or use a certain feature and then handle whatever problems you detect. Browser sniffing—that is, detecting what browser the user is using by examining its user agent string—has almost gone out of style. Today, the concept of feature detection has taken its place. Testing for available properties, objects, and functions is a much saner strategy than relying on a string that users can change and assuming a set of supported features.

With so many discrepancies in the various implementations and features that can be tricky to detect, adequate feature detection is no simple task. Fortunately, you don't usually need to reinvent the wheel because many clever tricks for detecting feature support have already been developed and aggregated in various libraries. One collection of these detectors is available in the Modernizr library (`www.modernizr.com`). Modernizr provides an easy-to-use method of testing whether a certain feature is available. You can detect everything from the `canvas` element and WebGL to web fonts and a whole slew of CSS features, allowing you to provide fallback solutions where features aren't supported and to degrade your application gracefully.

Filling the gaps with polyfills

Beginning in the early 2000s, a popular trend has been to favor so-called *progressive enhancement* when adding new features to websites. This strategy calls for websites to target the lowest common denominator in terms of supported features. Any technology that isn't supported across the board should be used only to add enhancements to the site, never critical functionality. This ensures that everyone can access and use the website. If the user has a modern browser, he simply has a better experience.

Progressive enhancement is a sound strategy in many cases, but sometimes you simply need to use a certain feature. If some browsers don't have native support for that feature, that hole must be plugged, even if it means using less than ideal or even hackish fallback solutions. These fallbacks are sometimes called *polyfills*, named after the spackling paste Polyfilla because their function is somewhat similar. They fill the cracks in the supported feature sets when you're running your code in actual browsers, bridging the gap between specifications and the reality of dealing with imperfect browsers. As an example, Internet Explorer had no support for `canvas` until IE9, but several polyfills exist that provide various amounts of `canvas` functionality for legacy browsers.

The ExplorerCanvas project from Google (`http://code.google.com/p/explorercanvas/`) was one of the earliest of these polyfills. It uses Vector Markup Language (VML), an old Microsoft developed XML-based language, to simulate a `canvas` element. Because it provides enough 2D drawing functionality, it's been used successfully in many projects. Some features are missing, however, because VML doesn't perfectly overlap the `canvas` specification and lacks support for patterns and clipping paths, for example.

Other polyfills use Flash or Silverlight to get even closer to the full `canvas` API, letting you use advanced features like image data access and compositing effects. With all these different options, picking the right fallback solutions is no easy task. Depending on the target platforms, sometimes even the polyfills need fallbacks.

Building a Game

Starting with Chapter 2 and throughout the rest of the book, I take you through the process of developing an HTML5 web game from scratch. I show you how to create a match-three gem-swapping game in the style of Bejeweled or Puzzle Quest, casual games that have been very popular on many platforms over the past decade. This type of game has tried-and-tested game mechanics, allowing you to focus your attention on the use of web technologies in the context of game development. Additionally, these games play well on desktop browsers and on mobile devices such as smart phones and tablets, all of which give you the opportunity to explore multiplatform web game development.

The game you'll develop takes advantage of several features from the HTML5 specification and also uses related technologies such as web fonts and CSS3 for building the UI. Although the game may not be revolutionary, it allows me to cover many of the newest advances in open web technology. Among other things, I use the `canvas` element to generate some of the game graphics, and I show you how to add sound effects using HTML5 audio. The finished game will be playable on a desktop browser, and I show you how to ensure that it plays just as well on mobile devices and even offline. I show you how to use Web Storage to save high-score data and to allow players to pick up where they left off.

The `canvas` element lets you create interesting dynamic graphics, but it isn't always suitable for creating user interfaces. You don't really need any new tools for that part, however, because traditional HTML and CSS give you all you need to build a great UI. With the latest additions to the CSS specification, you can add animations, transforms, and other features that bring life to the UI experience. In Chapters 6 and 7, I show you how to build the display module with the canvas element. Later on, in Chapter 11, I take you a bit further as I show you how to use WebGL to add 3D graphics to the game.

In Chapter 13, I show you how to create a simple chat application using WebSockets. For this purpose, I also show you how to develop a small server application using the Node.js framework (`http://nodejs.org`). WebSockets are supported in most modern browsers, one notable exception being the Android browser, which as of Android 4.3, still doesn't support this feature.

Summary

It's been a bumpy road but it finally looks like HTML and the web in general are on the right track again. The WHATWG brought in some fresh perspective on the standards process, and web developers are now beginning to enjoy the fruits of this undertaking in the plethora of new tools and an HTML standard that's more in line with how the web is used today. As always, using new features requires dealing with older browsers but many polyfills that you can use to ensure cross-browser compatibility are already available.

Many of the new additions are of special interest to game developers because real alternatives to Flash-based web games are now available. Canvas and WebGL bring dynamic- and hardware-accelerated graphics to the table; the `audio` element has finally enabled native sound; and with WebSockets, it's now possible to create multiplayer experiences that more closely match desktop games than was possible just a few years ago. Advances in other, related areas like CSS and the increasing support for web fonts let you create richer UI experiences using open, standardized tools.