HTML5 Games Most Wanted

Build the Best HTML5 Games

EGOR KURYANOVICH, SHY SHALOM,
RUSSELL GOLDENBERG, MATHIAS PAUMGARTEN,
DAVID STRAUß, SEB LEE-DELISLE, GAËTAN RENAUDEAU,
JONAS WAGNER, JONATHAN BERGKNOFF,
ROB HAWKES, AND BRIAN DANCHILLA

friendsof ™
an Apress® company

HTML5 Games Most Wanted

Egor Kuryanovich
Shy Shalom
Russell Goldenberg
Mathias Paumgarten
David Strauß
Seb Lee-Delisle
Gaëtan Renaudeau
Jonas Wagner
Jonathan Bergknoff
Brian Danchilla
Rob Hawkes

friendsof ED™
DESIGNER TO DESIGNER™
an Apress® company

HTML5 GAMES MOST WANTED

ISBN 978-1-4302-3978-9

ISBN 978-1-4302-3979-6 (eBook)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 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`.

Any source code or other supplementary materials referenced by the author in this text are available to readers at `www.apress.com`. For detailed information about how to locate your book's source code, go to `www.apress.com/source-code/`.

Credits

President and Publisher:
Paul Manning

Lead Editor:
Ben Renow-Clarke

Technical Reviewer:
Andrew Zack

Editorial Board:
Steve Anglin, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh

Coordinating Editor:
Jessica Belanger

Copy Editors:
Kimberly Burton

Compositor:
Mary Sudul

Indexer:
SPi Global

Artist:
SPi Global

Cover Image Artist:
Corné van Dooren

Cover Designer:
Anna Ishchenko

Contents at a Glance

Contents

About the Authors

Egor Kuryanovich is a web developer from Belarus living in Minsk. He goes by the name of Sontan on the web, and his game, Far7, won the Best Technology category of Google's 2010 Game On competition.

Shy Shalom is an Israeli software engineer living in Tel Aviv. While his day job consists of writing enterprise applications in C++, his true passion is the aesthetics of 3D rendering. Shy has been active in the field of computer graphics throughout his career.

He has published several papers in the field of geometry processing while completing his graduate degree and has tutored courses on graphics and OpenGL at Tel Aviv University. Shy started working with WebGL at a very early stage in its conception and participated in the beta testing of its early implementations in Firefox and Google Chrome.

Some of Shy's projects in graphics, which are available online, include *CycleBlob*, a 3D lightcycle game that runs in the web browser; the *Happy Cube Solver*, an aid to solving 3D foam puzzles; and *KawaiiGL*, a small IDE for writing GLSL shaders and basic geometric modeling.

Russell Goldenberg is a candidate for a master's degree in interactive media at Emerson College in Boston. He is currently finishing up his final semester while teaching an undergraduate course in web design and development at Emerson. In 2009, he graduated from Union College with an interdepartmental degree in computer science and visual arts. If he's not coding, he's cooking or playing basketball.

His creative work primarily uses computer programming to create art. He investigates ideas in a variety of methods: games, data visualizations, interactive installations, and generative animation. Both form and topic tend to change. His process is to dive in and see what works and emerges along the way.

Mathias Paumgarten is a web developer that focuses on front-end and interaction development. Starting with a background in Flash development, he now focuses mainly on JavaScript. He found his passion for the web and code-based animation at a very young age and made his way from a small town in Austria all the way to New York City. He currently works for Firstborn Multimedia as a creative developer.

David Strauß is a creative programmer who discovered the magic of creating things at a very young age through his father's passion for working with wood. David mainly uses CoffeeScript and the Ruby on Rails framework to create beautiful web applications. One of his first projects, *Marble Run*, won Mozilla's Game On 2010 Challenge, beating teams like Fantasy Interactive and the Google Chrome Team.

Seb Lee-Delisle is a creative coder, speaker and teacher, working across platforms including JavaScript, Processing and openFrameworks. He works to bring people together with large scale installations like PixelPhones, interactive firework displays or glow-stick voting systems.

His work has pushed the boundaries of what is possible both on and off the web, and won two BAFTAs with Plug-in Media, the agency he co-founded in 2004.

A sought-after speaker, his recent Creative JavaScript / HTML5 workshop series sold out within hours. He co-hosts the Creative Coding Podcast, his blog can be found at seb.ly and he tweets @seb_ly.

Gaëtan Renaudeau (aka gre) is a web enthusiast studying towards a master's degree in computer science in France. Since 2009, he has worked for Zenexity in Paris as a web developer, building web applications, web services, and mobile applications. He enjoys working both in front-end (JavaScript, HTML5, CSS3) and server-side (mainly Play framework) development.

For fun you'll find him making web experiments, applications, and libraries— such as the recently completed HTML5 game called *Same Game Gravity*, available on six different platforms.

Jonas Wagner is a software engineer from Zurich, Switzerland, working on one of the biggest Swiss web sites. When not at work, he likes to explore the boundaries of what's possible with web technology.

Jonathan Bergknoff is a hobbyist programmer from New Jersey. He graduated from Cornell University in 2006 with degrees in math and physics and is currently earning a PhD in theoretical physics at UCLA. His programming projects include the JavaScript tower defense game *Picnic Defender*, a *Minesweeper* clone for the Nintendo DS, an NES emulator, and the iPod manager *jPod*.

Brian Danchilla is a freelance PHP and Java developer and author. He is the coauthor of *Pro PHP Programming* (Apress, 2011), a technical reviewer of *Foundation HTML5 Animation with JavaScript* (friends of ED, 2011), and the author of the upcoming *Beginning WebGL* (Apress, 2012). Brian has worked with OpenGL for several years and has a passion for graphics, art, and mathematics. His other interests include playing guitar, listening to music, and enjoying the outdoors.

Rob Hawkes is a serial experimenter who specializes in JavaScript development. He works for Mozilla as a Technical Evangelist and takes a strong interest in the game-related events and projects that happen within Mozilla and the wider developer community.

About the Technical Reviewer

Andrew Zack is the CEO of ZTMC, Inc. (www.ztmc.com), which specializes in search engine optimization (SEO) and internet marketing strategies. His project background includes almost 20 years of site development and project management experience and over 15 years as an SEO and internet marketing expert.

Andrew has been very active in the publishing industry, having coauthored *Flash 5 Studio* (Apress, 2001) and serving as a technical reviewer on more than 10 books and industry publications.

Having started working on the internet close to its inception, Andrew continually focuses on the cutting edge and beyond, concentrating on new platforms and technology to stay at the forefront of the industry.

About the Cover Image Artist

Corné van Dooren designed the front cover image for this book. After taking a brief hiatus from friends of ED to create a new design for the Foundation series, Corné worked at combining technological and organic forms with the results now appearing on this and other book covers.

Corné spent his childhood drawing on everything at hand, and then he began exploring the infinite world of multimedia—His journey of discovery hasn't stopped since! His mantra has always been, "The only limit to multimedia is the imagination," a saying that keeps him constantly moving forward.

Corné works for many international clients, writes features for multimedia magazines, reviews and tests software, authors multimedia studies, and works on many other friends of ED books. You can see more of his work and contact him through his website at: `www.cornevandooren.com`.

Introduction

HTML5 is a “game changer,” allowing web browsers on such diverse hardware as smartphones, tablets, and personal computers to display the same games, interactive ads, and rich-media applications that were previously only possible as long as the end-user had downloaded the appropriate third-party plug-ins, most notably Flash. With the advent of HTML5, programmers are able to create cross-platform web applications.

The authors of this book are all real-world games programmers who have come together to share their HTML5 expertise, tips, and tricks with you.

In the first chapter of the book, Rob Hawkes, a Mozilla evangelist, discusses the state of open web gaming today. He explores the core technologies of HTML5 and JavaScript and the new APIs that have been introduced.

In Chapter 2, Russell Goldenberg will walk you through the design implementation and coding of his game *A to B*. This physics-based game involves a ball that must be manipulated by a series of modifiers, including walls and speed boosters, from point A to point B. It was written in Processing.js, a tool with which Goldenberg was very familiar, allowing him to quickly and easily create the game. For this project, Goldenberg was not looking to create a scalable, multi-author game, but instead was looking for a “quick and dirty” approach. The resulting game was a finalist in the Mozilla Game On 2010 competition.

In Chapter 3, Gaëtan Renaudeau shows us how to make multi-platform HTML games from scratch. He introduces us to the use of CSS, JavaScript, the canvas element, and DOM within HTML5 applications as demonstrated in a chess game that he has coded.

Chapter 4 is an in-depth walk-though of the use of the canvas element in HTML5. David Strauss and Mathias Paumgarten show us some of the techniques behind their massively popular *Marble Run* game.

We see how to create a three-dimensional iPhone/iPad game using HTML5 and CSS in Chapter 5. Seb Lee-Delisle of the BAFTA-winning Plug-in Media, will show you how to use CSS 3D transformations to move HTML elements in three-dimensional space. The example game in this chapter involves puffer fish that are to be exploded on touch.

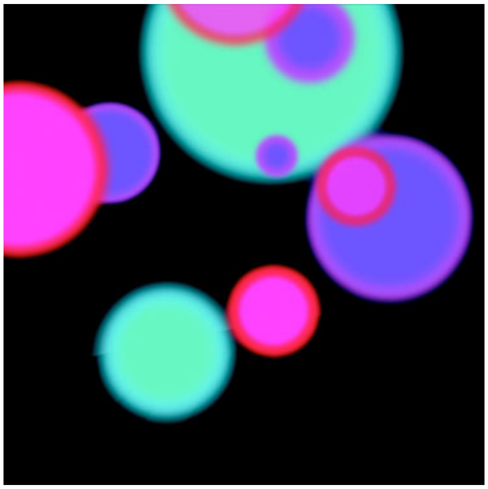
Chapter 6, by Jonas Wagner, focuses on particle systems to create effects such as fire, rain, and smoke. You’ll create a high-performance particle system that can deal with tens of thousands of particles, and find many code examples that you can use in your own projects.

Chapters 7 and 8 introduce the reader to WebGL. In Chapter 7, Brian Danchilla uses the example of a darts game to walk the reader through checking for WebGL support, understanding 3D coordinate systems, drawing basic shapes, animating 3D objects, and adding textures.

Chapter 8 advances from the previous chapter and shows how Shy Shalom used WebGL to create a three-dimensional, *TRON*-inspired, lightcycle game called *CycleBlob*.

in Chapter 9, learn how Jonathan Bergknoff used canvas, netcode, and WebSockets to create a real-time, multiplayer, bumper cars game. Bergknoff demonstrates the complexities of the game logic needed to enable his game to handle glancing collisions of several vehicles and their resulting trajectories in a multiplayer environment.

Finally, we round off the book with Chapter 10, where Egor Kuryanovich introduces us to the decision-making process when assessing which technologies to include in our HTML5 application. He outlines the benefits and disadvantages of the HTML5 canvas element, SVG, web sockets, server-sent events, and web fonts, preparing us for understanding the implications of the choices made by the other authors in subsequent chapters.



Chapter 1

The State of Open Web Games

In this chapter, I introduce the concept of open web game development using technologies such as HTML5 and JavaScript. I cover the various features that these technologies provide, including such gems as the Gamepad API, which allows you to break away from keyboard and mouse input. Towards the end of the chapter, I cover the current method of distributing and making money from your games, as well as highlighting the events that must happen for the web to become a viable platform for game development.

A brief introduction

My name is Rob Hawkes. I am a serial experimenter who specializes in JavaScript development. I work for Mozilla as a Technical Evangelist and take a strong interest in the game-related events and projects that happen within Mozilla and the wider developer community.

I'm a hobbyist game developer and I have worked on a variety of game-related projects using a variety of programming languages (Unity, ActionScript, PHP, Processing, and JavaScript) and technologies (augmented reality, mobile phones, desktop, and browsers).

Most recently I have been working on a multiplayer HTML5 game called Rawkets (`http://rawkets.com`) that acts as my test bed for experimenting with the various game-related technologies that I will be covering further on.

You can get in touch with me through my personal blog (`http://rawkes.com`) or through Twitter (`@robhawkes`). I'm always happy to help where I can.

Why should you care about open web games?

Open web games are by nature games created with open web technologies; as of today, these technologies are HTML5, CSS3, and JavaScript. In this chapter, I will be referring to these technologies under the umbrella term “open web games,” as I feel it sums up the importance of the technologies quite well while also being easier to write multiple times—rather than listing every single individual technology, which I think we’ll both appreciate in the long run.

The beauty of the technologies behind open web games is that they are the same ones that countless developers are already using to create web sites and web apps (arguably the same thing, but in this case it is worth defining them separately). These are technologies that have evolved since the dawn of the internet and have been proven as reliable and stable while other technologies rise and fall beside them.

Easy to get started

There is also an almost non-existent barrier to entry to develop games or anything else with these technologies. First off, they are completely free to use in all senses of the word. Secondly, the tools needed to develop and host games with these technologies can also be found for free or extremely low cost. In short, very little investment needs to be made to begin developing games using open web technologies. This is a massive plus point for indie developers who might be used to more restrictive environments like Flash, where you are required to buy into the proprietary technology and related development tools.

Excellent documentation

On top of the low barrier to entry is the well-written and free documentation that exists to help developers learn about every minor detail of these technologies. Web sites like the Mozilla Developer Network (`https://developer.mozilla.org`) have provided services like this for many years, with numerous other web sites and personal blogs doing the same.

Large and friendly community

Complimenting the documentation effort is a thriving community of developers and designers who care for nothing more but to further the web and share their experience with others. This is a community that can be found nearly everywhere you look; from the HTML5 group on Facebook, to Twitter, to dedicated forums like SitePoint, all the way to real-time chatrooms on IRC. For example, there is a growing community of open web game developers hanging out in the `#bbg` channel on `irc.freenode.net`—and we would love for you to come along and take part.

Write once, use anywhere

On a more technical level, the beauty of developing games with open web technologies is that it’s very much a “write once, use anywhere” kind of approach. Now this isn’t entirely true, as there are always nuances and exceptions to the rule; however, what is true is that this approach is inherently cross-platform and the technology has been created to work on a variety of operating systems and browsers with little to no platform-specific code.

What you *can* be certain of is that if a platform supports HTML5 and the JavaScript APIs that your game requires, then it's likely that your game will function in the way that you expect. Obviously things like hardware performance will cause issues on a per-device level, but that is something you'll experience using any technology anyway.

Uncompiled and open

Something that many proprietary developers find uncomfortable about the move to open web technologies is that the code is completely uncompiled and open for users to view. If you right-click on any web site within your browser, you will be able to view the source code and assets with relative ease—and the same is the case for open web games.

This behavior is core to the strength and purpose of the open web and it is unlikely to change— however much developers from other platforms would like it to. It is unlikely that digital rights management (DRM) will make its way onto the web in a non-proprietary way, and the same can be said for the compilation of code and assets, so that others can't see them in a readable fashion.

In my eyes, this clash of cultures is one of the major sticking points for game developers coming from platforms like mobile, console, and desktop. Traditional game development in that sense has been built around the idea of protecting intellectual property and making code and assets as difficult to access as possible. Now, one could argue that such moves are fruitless (I've yet to see a method of DRM that *hasn't* been eventually cracked), but I get why they exist and the motives behind their use. Unfortunately, all browsers are unlikely to accommodate this way of thinking, so it's not a viable way to look at game development on the web.

Instead, I believe that it is important for the web to prove itself as a viable platform to these developers and show that open assets and code does not mean rampant theft and a loss of control. History tells us that this won't happen and I'm confident that the benefits of the web as a platform will far outweigh the (minor) issues. In other words, how many web sites have you seen get stolen, replicated, and then perform better than the original?

Everyone has control

Something that still amazes me to this day is how no single entity controls the technologies and platform that the web is built on. This idea is another foreign concept to developers coming from a proprietary background, as there isn't a single point of contact to reach for when you want something added or changed. Instead, the technologies behind the web are defined by a set of specifications that are each managed by either the World Wide Web Consortium (W3C) or the Web Hypertext Application Technology Working Group (WHATWG).

Both of these groups are made up of a variety of stakeholders ranging from browser manufacturers, to technology companies, to general web developers. Anyone can be a part of these groups and that is why everyone has control. If you want something added or changed, then all you need do is take part in the discussions and have your say. For example, if you'd like to be a part of the discussions surrounding open web gaming technologies, then you should get involved in the W3C Games Community Group (`www.w3.org/community/games/`).

Access to the world's biggest audience

If anything, this is perhaps one of the most important aspects of the web as a platform for games. By building for browser technologies, you have access to practically every web user out there—all 6.9 billion of them! OK, perhaps not *all* of them and admittedly not every person will have an up-to-date browser. But still, my point here is that there are an astonishing amount of people using the web, with more and more getting connected every day. Even if we just counted Firefox users, that's hundreds of millions of people (a lot). And Facebook users? That's well over 800 million people (more than a lot)!

How you distribute your game to all those people is another problem entirely, and one that I will touch on briefly further on in this chapter.

What is the current state of open web games?

The past few years have seen a massive improvement in browser platforms and the adoption of technologies required to create open web games. This is coupled with the recent increase in the quantity of open web games that are being distributed on app stores and social networks, a number that is increasing every day. Also, large game studios are beginning to take interest and the general quality of these games are improving at a noticeable rate.

However, I think what has been most key in the recent improvements in open web gaming is the unease surrounding the future of Flash on mobile and the web. What we have now with HTML5 and JavaScript is a platform that can no longer be simply cast aside as unviable—open web games are definitely here to stay.

Game-related browser technologies already exist

What I still find most fascinating with this area of the web are the technologies that already exist and that are arriving soon; things like the Gamepad API, Mouse Lock API, and Full Screen API, among many others. These simple technologies are the ones that will help demolish the idea that games on the web are small boxes played embedded within another web site. Instead, with the ability to connect gamepad controllers and allow HTML elements to run full screen, open web games will become much more immersive experiences, much like on consoles and the desktop.

The following are just a few examples of the technologies that are in browsers today or on their way very soon. I encourage you to look into them all in more detail to discover how simple they are to get started with. It's also worth mentioning that browser support for these technologies changes at a rapid pace. I would check out the web site When Can I Use… for up-to-date information (`http://caniuse.com`).

2D graphics with HTML5 canvas and SVG

Visual output is one of the core components of most games, so the ability to produce and manage 2D graphics within a browser is very important. This is where both HTML5 canvas and scalable vector graphics (SVG) come in.

HTML5 canvas (often referred to as simply "canvas") is a JavaScript API and corresponding HTML element that allows for bitmap graphics to be created and edited within the browser. The plus points of canvas are that it's speedy and that can produce pin-point pixel graphics without relative ease. The negative aspects to canvas are that performance varies across platforms and animation functionality isn't built in.

On the other hand, SVG is another 2D solution that uses the document object model (DOM) to produce and manage vector graphics. The plus points of SVG are that it's accessible (in that the graphics are described with DOM elements), has animation ability built in, and that the vector approach means that graphics can be scaled easily to accommodate various devices and screen sizes. The negative aspects of SVG are that it isn't as popular as canvas and that it doesn't cope as well with pixel-perfect precision.

3D graphics with WebGL

If you're looking for 3D graphics for your game, then the WebGL JavaScript API is exactly what you need. It's based on OpenGL ES 2.0 and provides all the functionality required to produce some pretty spectacular effects.

The plus points of WebGL are that it's hardware accelerated (fast) and allows for some pretty complex visual effects. The negative aspects are that it is complicated to learn and isn't supported by Internet Explorer (IE) yet. The factor of it being complicated can by mitigated by using frameworks such as three.js (`https://github.com/mrdoob/three.js/`).

Better animation performance with requestAnimationFrame

Most animation within open web games is created by repeatedly changing what's on the screen with what's known as a loop, and if you do this fast enough, the updating graphics appear to move smoothly.

Until now the easiest way to do this has been with the JavaScript `setTimeout` or `setInterval` methods. However, the problem with this is that they run constantly and can cause all sorts of performance issues. They also don't stop running when a game is left open in an inactive tab or when the browser is minimized, which isn't ideal.

To solve this, the `requestAnimationFrame` JavaScript method has been introduced. The purpose of this method is to give control of the animation loop to the browser so that it can be performed in the most optimal way possible. This often increases performance and prevents those nasty situations where a 30-millisecond loop is running continuously in an inactive tab or hidden browser window. With the new method, the animation loop is drastically slowed down or even stopped, which can have a positive effect on things like battery life on mobile devices.

Music and sound with HTML5 audio and the audio data APIs

Another fundamental aspect to most games is audio, something that until recently would have been difficult to implement and would likely have used Flash. The HTML5 audio element has removed this need and provides a pluginless method of playing audio within the browser.

The limitation of the HTML5 audio element is that its purpose is really to play single audio files, like background music within a game. It isn't suitable for sound effects, particularly if they are fast paced and if there are many of them to play at once.

To solve this, the Audio Data API (Mozilla) and Web Audio API (Chrome) have been introduced to allow for much more fine-grained audio functionality. With these data APIs, you can create sounds from JavaScript, you can edit audio on the fly, you can play more than one channel of audio at a time, and you can retrieve data about the audio in real-time as it plays.

Unfortunately, the audio data APIs solutions aren't yet housed within a single specification, and as such, you need to accommodate both slightly different Mozilla and Chrome proposals. My hope is that in the near future common ground will be found for a single audio data API to be born from.

Real-time multiplayer gameplay with WebSockets

If you're thinking of creating a multiplayer game, then before now you would either have put up with the latency involved in constant AJAX requests, or you would have moved to Flash. Neither option is ideal. What's cool is that since 2011, this is no longer the case, WebSockets have now arrived in all the major browsers (yes, including IE10) to allow for real-time *bi-directional* communication between the browser and a server.

But why is bi-directional real-time communication important for games? Well, this means that you can now literally *stream* data to and from a player's browser in real time. One obvious benefit to this is that it saves bandwidth by not requiring constant AJAX requests to check for new data. Instead, the WebSocket connection is left open and data is instantly pushed to the player or server as soon as it is needed. This is perfect for fast-paced games that require an update every few milliseconds. On top of this, the bi-directional nature of WebSockets means that data can be instantly sent both from the server to the player and from the player to the server *at the same time*.

Store data locally with IndexedDB, Local Storage, and other APIs

Many games require data to be stored on the player's machine so that it can be retrieved at a later date—things like save-game data or cached graphical assets. Until recently, the only way to do this has been to store data on a web server and put up with the latency, or to use things like cookies and only store very small pieces of data.

Fortunately, there are now a variety of solutions that solve various aspects of this problem. The most common are IndexedDB, Local Storage, as well as the various File and FileSystem APIs. The first two allow large quantities of data to be stored in a structural way within a player's browser, with IndexedDB even allowing files to be stored. The File and FileSystem APIs allow a game to access the player's OS file system using JavaScript, letting you save and retrieve files much larger than would be permitted in any other solution.

Play games offline with the application cache

Creating games on the web is all well and good, but what about if you want to play that game offline? Or, what if the player's internet connection drops out half way through an epic gaming session? Most open web games today would, at worst, simply stop working as soon as an internet connection failed, and, at best, they would stop sending data to your server and saving player data. When your player refreshes the page that the game is on, they'll just see a blank page and all their hard work achieved while offline will have been lost. That probably won't make your players very happy, and unhappy players are not ideal.

There are a few solutions available today that can help solve these issues. The first is the application cache, which allows you to use a cache manifest to declare particular assets (like HTML, CSS, images, and JavaScript) that you would like the browser to cache for offline use. Not only that, the browser uses the cached versions of the files when online to speed up the loading process.

Another technique that you can use is to store a player's game data locally and periodically sync it with the game server. Normally you wouldn't be able to store enough data in the browser to achieve this (like with cookies), but with Local Storage and IndexedDB you can now store many megabytes of data in a structured way.

You can also add functionality to your game so that it is alerted when a player's internet connection goes offline. The `navigator.onLine` property allows you to use JavaScript to see if your player is currently online or not. You can also use the offline and online events to trigger automatic behavior in your game when a change in connection occurs, like stopping all WebSockets communication and caching player data locally until the connection is back.

Immersive gameplay with the FullScreen API

Something that prevents current games on the web from feeling immersive is that they look like they're just tiny boxes embedded into another web site. Why do they feel like that? Well, because they *are* just tiny boxes embedded into other web sites. The odd five-minute puzzle game during your lunch hour might feel OK in a tiny box surrounded by browser UI and other distractions, but a first-person shooter or driving game certainly wouldn't.

Fortunately, the FullScreen API has arrived to solve this problem. It allows you to make any DOM element fill the player's entire screen, something normally only considered for videos. Using this in a game can make the difference between five minutes of relative fun and hours of immersive delight.

Tame the mouse with the Mouse Lock API

An issue related to input in games is that of misbehaving cursors, where the mouse is used to rotate a player around a fixed point (like moving the viewpoint in a 3D first-person shooter) or rotating the player in a top-down 2D game.

In both of these situations, the mouse cursor is visible at all times, which is generally annoying and ruins the experience. However, the most debilitating problem is that all movement stops when the mouse cursor leaves the browser window. This same behavior occurs in full-screen mode when the mouse cursor hits the edge of the screen. It's a horribly simple problem for a player that completely ruins the experience.

The good news is that the Mouse Lock API has been created to solve this problem, and it just landed in experimental builds of Firefox Nightly and will soon land in Chrome (it is likely that the support will be in public builds of these browsers by the time you read this). Its sole purpose is to tame the mouse by hiding the cursor and locking it in place so that it doesn't hit the edges of the screen. This means that instead of relying on x and y coordinate values for mouse position in related to the top-left corner of the browser, you instead rely on x and y distance values from the position that the mouse was locked to.

Console-like experience with the Gamepad API

Another input-related improvement that is coming to the web is that of the GamepadAPI. No longer are the keyboard and mouse the only options available for your players to engage with your game. The GamepadAPI now allows for all sorts of gamepads to be accessed via JavaScript. This even includes some of the console controllers like those on the Xbox 360 and PlayStation 3 (with third-party drivers)!

Like the Mouse Lock API, the GamepadAPI has just landed in experimental builds of Firefox Nightly and Chrome—and it's nice and simple to use (again, it is likely that the support will be in public builds of these browsers by the time you read this). Coupled with the Full Screen API, gamepad support can really change the experience of your game from that of a game within a web site to that of a desktop game or console.

Identify players with services like BrowserID

Just like how iOS has services like OpenFeint and the Apple Game Center, games on the web need open and reliable methods of identifying players. BrowserID is one of Mozilla's solutions to this problem, which allows players to log into your game using their existing e-mail address and without needing a password.

Identifying players in this way is just the first step in providing all sorts of functionality with your game, like friends lists, leader boards, chat, and multiplayer.

Create native OS applications with environments like WebRT

One of the more profound initiatives within Mozilla is the integration of a web run-time (WebRT). It allows players to install your game "natively" on their chosen operating system (Windows, Mac, and Android right now), with a launch icon just like standard OS applications.

WebRT also runs your game using an app-centric user agent (in contrast to browser-centric user agents like Firefox) and runs your game using a separate user profile and OS process to your player's normal Firefox that they use for browsing.

The ability of WebRT to break away into another process and remove all the browser UI makes the experience for gaming that much sweeter. There's something about having an icon in the dock on a Mac that launches your game in its own "native" window with no mention or feel that this is a browser.

As a developer, this is slightly magical. It allows you to break free from a game being a glorified web site and instead turning it into an application, an experience in its own right. Mark my words: this will be a turning point in the transition from five-minute puzzle games on the web to professional-grade games that have a console-like experience.

With much more on the way

These technologies are really just scratching the surface when it comes to creating games on the web using open technologies. Mozilla and other companies are working hard to bring you these APIs and services to help make the web a better place for games.

There are plenty of good open web games out there

Although open web game development is still fairly new, there are already many great examples of games out there today. I'd like to briefly highlight just a few of them.

Bejeweled

Towards the end of 2011, PopCap released a HTML5 version of their massively popular *Bejeweled* game (see Figure 1-1). It uses WebGL to provide accelerated graphics, falling back to HTML5 canvas if WebGL is not supported. You can play this game by visiting `http://bejeweled.popcap.com/html5`.

Angry Birds

Arguably one of the most popular games around right now is *Angry Birds*, and earlier this year Rovio brought out an HTML5 version (see Figure 1-2). It uses WebGL for accelerated graphics. You can play this game by visiting `http://chrome.angrybirds.com`.

Robots Are People Too

Unique gameplay always stands out amongst the plethora of clones and ports from existing and popular games. *Robots Are People Too* (see Figure 1-3) requires two players to cooperate to survive, helped by the innovative split-screen mechanic within the game. It uses HTML5 canvas for the graphics. You can play this game by visiting `http://raptjs.com`.

Runfield

As part of the Firefox 4 release earlier in 2011, *Runfield* (see Figure 1-4) was created to show off some of the capabilities of the browser. It used HTML5 canvas for the graphics and HTML5 audio for the sound. You can play this game by visiting `https://developer.mozilla.org/en-US/demos/detail/runfield`.

TF2 WebGL demo

Arguably not a *real* game is the TF2 demo (see Figure 1-5) created by Brandon Jones. It's a tech demo that shows how Valve's Source maps can be rendered with high performance using WebGL graphics. You can find out more by visiting `http://blog.tojicode.com/2011/10/source-engine-levels-in-webgl-video.html`.