Kapitel 1
Spielen im Web

Inhalt

- Was ist HTML5 und woher kommt es?
- HTML5 im Spielkontext
- Funktionserkennung aktivieren
- Umgang mit betagten Browsern

Bevor wir uns mit den ersten Quelltexten beschäftigen, will ich erst einmal die von uns genutzten Technologien in den entsprechenden Kontext setzen. In diesem ersten Kapitel werde ich daher erklären, was HTML5 ist, und ein wenig von den geschichtlichen Hintergründen erzählen, die zur HTML5-Spezifikation geführt haben.

Einer der interessantesten Aspekte von HTML5 besteht darin, wie Spieleentwickler von den vielen, neuen Funktionen profitieren können. In diesem Kapitel werde ich einige dieser neuen Funktionen vorstellen und Ihnen ein paar kurze Beispiele für deren Einsatz liefern. Ich werde auf das `canvas`-Element und WebGL eingehen und den damit einhergehenden riesigen Verbesserungen bei der dynamischen Erzeugung von Grafiken. Zudem werde ich das `audio`-Element und die von der WebSockets-Spezifikation bereitgestellten zusätzlichen Multiplayer-Möglichkeiten ansprechen.

Unsere Begeisterung für neue Spielzeuge kennt häufig keine Grenzen, dabei sollten wir aber nicht vergessen, dass dort draußen im Web noch viele alte und veraltete Browser existieren, die dafür sorgen, dass viele Benutzer diese brandaktuellen Funktionen nicht nutzen können. In diesem Kapitel werde ich Ihnen daher einige nützliche Werkzeuge vorstellen, mit denen Sie feststellen können, welche Funktionen Sie ohne Probleme nutzen können. Aber ich werde Ihnen auch

zeigen, wie Sie diese Funktionstests dazu benutzen können, um bei Bedarf geeignete Fallback-Lösungen laden zu können.

Schließlich werde ich dann das Spiel kurz vorstellen, das ich im Verlauf dieses Buches benutzen werde, um Sie durch die Erstellung eines vollständigen HTML5-Spiels zu geleiten.

1.1 Die Geschichte von HTML5

HTML, die Sprache des Webs, hat seit ihrer Erfindung Anfang der 1990er-Jahre zahlreiche Revisionen durchlaufen. Als dann *XML* (*Extensible Markup Language*) um die Jahrtausendwende herum ein ganz heißes Thema war, wurden erhebliche Anstrengungen unternommen, um HTML in eine XML-konforme Sprache zu überführen. Durch die fehlende Akzeptanz, mangelhafte Browserunterstützung und Abwärtskompatibilität herrschte jedoch schließlich ein wildes Durcheinander im Web, ohne dass eine klare Richtung erkennbar gewesen wäre. Einige waren der Meinung, dass das für die Standards zuständige Gremium sein Gespür für die Realität im Web verloren habe.

Als das W3C schließlich das XHTML-Projekt fallenließ, hatte sich bereits eine unabhängige Gruppe mit dem Ziel gebildet, das Web für die heute verbreiteten Anwendungen geeigneter zu machen. Statt einfach auf der letzten Spezifikation aufzubauen, begann die *WHATWG* (*Web Hypertext Application Technology Working Group*), bestehende Entwicklungsmuster und die wildwüchsigen nicht-standardisierten Browserfunktionen zu dokumentieren. Schließlich vereinte das W3C seine Kräfte mit der WHATWG. Seitdem arbeiten die beiden Gruppen wieder zusammen daran, neue und aufregende Funktionen in die HTML5-Spezifikation einzubringen. Da diese Spezifikation eher der wirklichen Nutzung des Webs durch Webentwickler entspricht, erleichtert sie zudem den Umstieg auf HTML5. Anders als bisherige Revisionen versucht HTML5 keinen strengen Satz von Syntaxregeln mehr zu erzwingen. Zur Aktualisierung einer Seite reicht es häufig bereits aus, nur die Deklaration des Dokumenttyps zu ändern.

Was aber ist HTML5 dann? Ursprünglich bezog sich diese Bezeichnung einfach nur auf die jüngste Revision des HTML-Standards. Da er mittlerweile zu einem Schlagwort mutiert ist und auch viele Technologien beschreibt, die eigentlich gar kein Bestandteil der HTML5-Spezifikation sind, lässt sich der Begriff heute schlechter definieren. Selbst das W3C verhedderte sich zwischenzeitlich angesichts des alles

einnehmenden Wesens von HTML5. Kurzzeitig definierte es HTML5 unter Einbeziehung von *CSS* (*Cascading Style Sheets*) und *SVG* (*Scalable Vector Graphics*) und trug damit nur zur Verwirrung bei. Glücklicherweise distanzierte sich das W3C später von diesem Standpunkt und kehrte wieder zur strengeren Definition zurück, die sich nur auf die tatsächliche HTML5-Spezifikation beschränkt. In einem noch etwas kühneren Zug ließ die WHATWG einfach die Ziffer 5 fallen und nannte den Standard einfach nur noch *HTML*. Das entspricht tatsächlich viel eher der Wirklichkeit, als sich Spezifikationen wie HTML immer in der Entwicklung befinden und nie von irgendeinem Browser umfassend unterstützt werden. In diesem Buch werde ich über weite Strecken einfach nur von *HTML* sprechen. Sofern von *HTML5* die Rede ist, können Sie davon ausgehen, dass ich mich auf die eigentliche W3C-Spezifikation namens HTML5 beziehe.

1.2 HTML5 für Spiele nutzen

Für viele der Funktionen aus der HTML5-Spezifikation gibt es Anwendungen im Bereich der Spieleentwicklung, aber das `canvas`-Element stellte die ersten Funktionen bereit, die größere Beliebtheit erlangen konnten. Die visuelle Natur dieses Elements trug zweifellos stark zu seiner schnellen Verbreitung bei, nachdem die ersten interaktiven Animationen und Grafikeffekte erschienen waren. Schon bald folgten anspruchsvollere Projekte, die für den neuen Standard eine gute Werbung waren und für die Zukunft ein dynamischeres und visuell interessanteres Web versprachen.

1.2.1 Canvas

Hobby-Spieleentwickler waren aus gutem Grund mit die ersten, die HTML5 übernahmen. Mit dem `canvas`-Element können Web-Spieleentwickler dynamische Grafiken erstellen und es bietet Ihnen eine willkommene Alternative zu statischen Bildern und animierten GIFs. In Ermangelung besserer Werkzeuge wurden eine ganze Menge mehr oder weniger genialer (und/oder verrückter) Lösungen zum Erzeugen dynamischer Grafiken entwickelt. Komplette Grafikbibliotheken stützen sich auf nicht mehr als kolorierte `div`-Elemente. Das mag ja clever sein, ist aber nicht gerade effizient, wenn mehr als ein paar sehr einfache Formen gezeichnet werden sollen. Es existieren URI-Verfahren (*Uniform Resource Identifier*), bei denen Sie `img`-Elementen beispielsweise Quelldateien entweder direkt im HTML-Code oder durch Setzen der `src`- oder der `href`-Eigenschaft mit JavaScript zuordnen können, wozu

base64-codierte Datenstrings verwendet werden. Eine der cleveren Einsatzmöglichkeiten dieses `data: URI`-Verfahrens bestand in der Echtzeiterzeugung von Bildern, die somit dynamisch animiert wurden, sich aber nicht für kleine und einfache Bilder eignete. *Wolf 5K*, der Gewinner des Wettbewerbs *The 5K* des Jahres 2002, bei dem sich Entwickler der Herausforderung stellen müssen, eine Website mit maximal 5 Kilobyte Umfang zu erstellen, verwendete ein recht ähnliches Verfahren. In dem Spiel, bei dem es sich um ein kleines 3D-Labyrinth handelte, wurden schwarze und weiße Bilder in Echtzeit erzeugt und laufend in die `src`-Eigenschaft des Bildes übertragen. Dabei baute es auf den Umstand, dass `img`-Elemente auch einen JavaScript-Ausdruck anstelle einer wirklichen URL übernehmen können.

Die auf der `canvas`-Leinwand gezeichneten Grafiken lassen sich nicht über HTML-Auszeichnungen erzeugen, sondern müssen über ein einfaches *API* (*Application Programming Interface*) mit JavaScript erzeugt werden. Listing 1.1 zeigt ein einfaches Beispiel für das Zeichnen einiger einfacher Formen. Beachten Sie, dass das vollständige API sehr viel mehr Funktionen bietet, als im Beispiel dargestellt werden können.

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

  canvas.width = 200;
  canvas.height = 200;

  // zwei blaue Kreise zeichnen
  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();

  // ein rotes Dreieck zeichnen
  ctx.fillStyle = "red";
  ctx.beginPath();
  ctx.moveTo(100, 75);
  ctx.lineTo(75, 125);
  ctx.lineTo(125, 125);
  ctx.fill();

  // einen grünen Halbkreis zeichnen
  ctx.strokeStyle = "green";
  ctx.beginPath();
```

```
  ctx.scale(1, 0.5);
  ctx.arc(100, 300, 75, Math.PI, 0, true);
  ctx.closePath();
  ctx.stroke();
</script>
```

Listing 1.1 Formen mit dem Canvas-API zeichnen

Dieser Quelltext erzeugt die in Abbildung 1.1 dargestellte einfache Zeichnung.

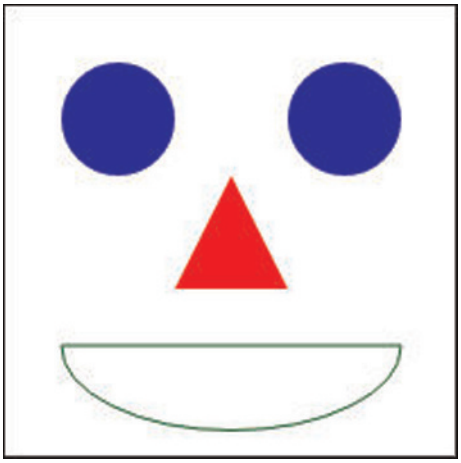

Abbildung 1.1 Eine einfache Canvas-Zeichnung

In Kapitel 6 werde ich bei der Erstellung von Spielgrafiken und Spezialeffekten auf das `canvas`-Element zurückkommen und es ausführlicher vorstellen.

1.2.2 Audio

Das neue `audio`-Element ist für den Werkzeugkasten der Web-Spieleentwickler eine ähnlich willkommene Ergänzung wie das `canvas`-Element. Endlich bieten Browser native Audiounterstützung, ohne dass dabei auf Plug-ins zurückgegriffen werden muss. Vor ein paar Jahren konnten Sie sich nahezu sicher sein, dass Flash auf irgendeine Weise beteiligt war, wenn auf einer Website Audio verwendet wurde. Bibliotheken wie das SoundManager-2-Projekt (*www.schillmania.com/projects/soundmanager2/*) bieten umfassenden JavaScript-Zugriff auf die meisten Flash-Audiofunktionen. Aber selbst wenn sich Ihr eigener Code bei der Nutzung derartiger Brückenfunktionen auf JavaScript beschränken kann, muss das Plug-in doch weiterhin beim Nutzer installiert sein. Das HTML5-Element `audio` löst dieses

Problem und ermöglicht den Zugriff auf Audio direkt aus dem Browser heraus unter ausschließlicher Nutzung von gutem alten HTML- und JavaScript-Code.

Allerdings gilt es auch beim `audio`-Element noch einige Probleme zu überwinden. Zwar sind alle wichtigen Browseranbieter gleichermaßen von der Bedeutung des Elements überzeugt und halten sich dabei an die Spezifikation, sie konnten sich aber bisher nicht darauf einigen, welche Codecs unterstützt werden sollen. Theoretisch ist beim `audio`-Element zwar alles in Ordnung, praktisch bleibt Entwicklern, wenn sie alle Browser bedienen wollen, aber letztlich keine andere Möglichkeit, als Audiodateien in mehreren Formaten bereitzustellen.

Das `audio`-Element kann sowohl über Tags im HTML-Quelltext, als auch dynamisch mit JavaScript erstellt werden. (Die letztere Option ist für uns als App- und Spieleentwickler interessanter.) Listing 1.2 enthält einen einfachen Audioplayer mit mehreren Quelldateien, den Steuerelementen der nativen Bedienoberfläche (UI – User Interface) und einigen Tastaturkürzeln, die das JavaScript-API nutzen.

```
<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(); // S-Taste (Start) gedrückt
    } else if (e.keyCode == 80) {
      audio.play(); // P-Taste (Pause) gedrückt
    }
  }
</script>
```

Listing 1.2 Einfache Musikwiedergabe mit HTML5-Audio

Hinweis

Sowohl im Mozilla-, als auch im WebKit-Lager wird an APIs gearbeitet, die schließlich das dynamische Erzeugen von Soundeffekten und Filtern für Audiodaten ermöglichen werden. Da sich diese APIs aber noch im Frühstadium ihrer Entwicklung befinden, werde ich sie nicht für die in diesem Buch dargestellten Spiele verwenden. Allerdings werde ich die von ihnen gebotenen Möglichkeiten in Kapitel 10 untersuchen, in dem ich mich näher mit HTML5-Audio befassen werde.

1.2.3 WebSockets

Ajax mit seinem zentralen `XMLHttpRequest`-Objekt und den daraus resultierenden Web-2.0-Entwicklungen haben das Web Anfang der 2000er-Jahre neu belebt. Ungeachtet der vielen neuen Möglichkeiten bleiben die Beschränkungen jedoch weiterhin offenkundig. Durch die Beschränkung auf das HTTP-Protokoll bleiben die Aktivitäten recht einseitig, da der Client aktiv Daten beim Server abfragen muss. Der Webserver kann dem Browser erst mitteilen, dass sich etwas geändert hat, wenn dieser eine neue Anfrage stellt. Bei der typischen Lösung wird der Server daher wiederholt abgefragt, um eine Aktualisierung anzufordern, oder er der Vorgang wird so lange nicht abgeschlossen, bis es etwas Neues zu berichten gibt. Für all diese Verfahren wird zuweilen der Sammelbegriff *Comet-Programmierung* verwendet. (Einen englischsprachigen Wikipedia-Artikel dazu finden Sie unter *en.wikipedia.org/wiki/Comet_(programming)*.) Vielfach reichen diese Lösungen zwar aus, sie sind aber recht einfach und für Multiplayer-Games oft zu wenig flexibel und leistungsfähig.

Und hier kommen *WebSockets* ins Spiel. Mit WebSockets kommen wir dem für eine effiziente Spielentwicklung erforderlichen Umfang an Steuerungsmöglichkeiten einen riesigen Schritt näher. Auch wenn es sich dabei nicht um eine umfassende Raw-Socket-Verbindung handelt (damit ließen sich eigene TCP- und UDP-Header erstellen), ermöglichen uns WebSocket-Verbindungen doch die Erstellung und Verwaltung von Verbindungen mit einer Zweiwegekommunikation, was uns speziell die Implementierung von Echtzeit-Multiplayer-Spielen erheblich erleichtert. In Listing 1.3 können Sie sehen, dass die Schnittstelle für die Verbindung mit dem Server und den Austausch von Botschaften recht einfach ist.

```
// Ein neues WebSocket-Objekt erzeugen
var socket = new WebSocket("ws://echo.websocket.org/");

// Eine Anfangsbotschaft an den Server senden
socket.onopen = function () {
  socket.send("Hallo Server!");
};

// Auf Daten warten, die uns der Server sendet
socket.onmessage = function(msg) {
  alert("Der Server meldet: " + msg.data);
}
```

Listing 1.3 Kommunikation mit einem Server über WebSockets

Um WebSockets nutzen zu können, müssen wir natürlich auch eine Serveranwendung implementieren, die mit dem WebSockets-Protokoll kompatibel ist und auf

die ihr zugesandten Mitteilungen reagiert. Wie ich Ihnen in Kapitel 13 zeigen werde, in dem ich unter Nutzung von *Node.js* auf der Serverseite Multiplayer-Funktionalität implementieren werde, muss es sich dabei jedoch nicht zwangsläufig um eine komplexe Aufgabe handeln.

In Listing 1.3 wird ein Echo-Server aufgerufen, der die ihm übermittelten Daten einfach nur zurückgibt. Abbildung 1.2 zeigt das Fenster, in dem Firefox 14 die Antwort des Servers ausgibt.

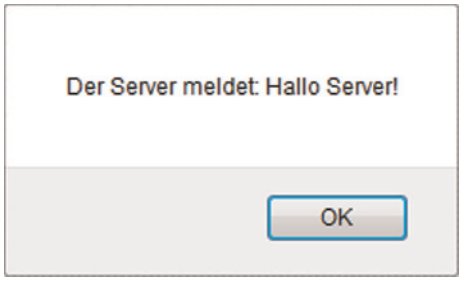

Abbildung 1.2 Die Ausgabe des einfachen WebSocket-Beispiels in Firefox

1.2.4 Web Storage

Wenn Webanwendungen Daten beim Client speichern müssen, werden dazu üblicherweise Cookies genutzt. Einmal abgesehen von ihrem schlechten Ruf als Spyware, die die Bewegung von Nutzern aufzeichnen, haben Cookies den Entwicklern den dringend benötigten Speicherplatz bereitgestellt, um Benutzereinstellungen speichern zu können. Zudem bieten sie Webservern Möglichkeiten, zurückkehrende Clients zu erkennen, wobei es sich angesichts der zustandslosen Natur des HTTP-Protokolls für viele Webanwendungen um eine notwendige Funktion handelt.

Web Storage war ursprünglich Bestandteil von HTML5, wurde später aber in eine eigene Spezifikation ausgelagert. Es lässt sich als eine Verbesserung des Cookie-Ansatzes betrachten und kann Cookies in vielen Fällen direkt ersetzen und mehr Speicherplatz für Daten bereitstellen. Die Daten werden dabei in Key-Value, also Schlüssel-Wert-Paaren gespeichert. Web Storage bietet aber noch weitere Vorteile. Während Cookies nur mit der Domäne verknüpft sind, bietet Web Storage neben dem lokalen Speicher, der Cookies ähnlich ist, auch noch Sitzungsspeicher (Session Storage), der mit dem aktiven Fenster und der aktiven Seite verknüpft ist, wodurch mehrere Instanzen derselben Anwendung auf verschiedenen Registerkarten oder in mehreren Fenstern unterstützt werden. Im Unterschied zu Cookies werden die

Daten mit Web Storage nur auf dem Client gespeichert und nicht mit jedem HTTP-Request übertragen. Dadurch lassen sich Megabytes und nicht nur Kilobytes an Speicherplatz verwenden.

Die Möglichkeit, auf nicht-flüchtigen Speicher zurückgreifen zu können, der immerhin einige Megabyte Daten aufnehmen kann, wird spätestens dann praktisch nützlich, wenn Sie irgendwelche komplexen Daten speichern wollen oder müssen. Mit Web Storage lassen sich zwar nur Zeichenketten speichern, aber wenn Sie diese Möglichkeit mit einem der von den meisten modernen Browsern nativ bereitgestellten *JSON-Encoder/Decoder* (*JavaScript Object Notation*) verknüpfen, können Sie diese Beschränkung leicht umgehen und auch komplexere Strukturen auf diesem Weg speichern. In dem Spiel, das ich im Verlauf dieses Buches entwickeln werde, nutze ich Web Storage lokal zur Implementierung einer Funktion zum Speichern des Spiels und zum Ablegen der Highscore-Daten.

Listing 1.4 zeigt die äußerst einfache und intuitive Schnittstelle für die Nutzung von Web Storage.

```
// Highscore-Daten speichern
localStorage.setItem("highscore", "152400");

// Daten können später auch von anderen Seiten aus
// wieder genutzt werden
var highscore = localStorage.getItem("highscore");
alert(highscore); // meldet 152400
```

Listing 1.4 Daten mit Web Storage lokal speichern

1.2.5 WebGL

Bei WebGL handelt es sich um OpenGL, nur für das Web. Das meistgenutzte Grafik-API steht auch nun Webentwicklern für die Online-Erzeugung von 3D-Grafiken bereit. Natürlich hat dies erhebliche Auswirkungen auf die Art der nun möglichen Webspiele. Wie als Beweis für die Tragweite dieser Aussage haben Google-Entwickler am 1. April 2010 einen WebGL-Port des legendären First-Person-Shooters Quake II präsentiert. Damit begegneten sie angesichts des sorgfältig gewählten Termins und der Leistung selbst allgemeinem Zweifel. (Die Webseite zum *Quake II GWT Port* finden Sie unter *http://code.google.com/p/quake2-gwt-port/*).

Hinweis

Die Erstellung eines vollständigen 3D-Spiels ist eine recht komplexe Aufgabe, die den Rahmen dieses Buches sprengen würde. Bei der Nutzung von WebGL müssen Entwickler zudem genau auf die zu unterstützenden Plattformen achten. So unterstützt der Internet Explorer beispielsweise so gut wie gar keine der neuen Möglichkeiten, und dieses Problem lässt sich auch mit anderweitigen Fallback-Lösungen nicht ohne Weiteres überwinden. Da WebGL bisher auch von Mobilplattformen wie iOS oder Android noch nicht unterstützt wird (was sich allerdings schon bald ändern dürfte), wird es in diesem Buch kaum verwendet. Für Nutzer mit Browsern, die WebGL unterstützen, werde ich jedoch bei der Erstellung eines interessanten Startbildschirms für das Spiel ein wenig auf WebGL zurückgreifen.

1.2.6 HTML5 ist (k)ein Flash-Killer

Seit es das `canvas`-Element und die verbesserten JavaScript-Engines gibt, finden im Internet mehr oder weniger sachlich geführte Diskussionen darüber statt, ob die neuen Standards *Flash* als vorherrschende Lösung für Multimedia-Apps im Web ablösen können. Flash war lange erste Wahl, wenn es um die Online-Präsentation von Audio, Video und Spielen ging. Und auch Konkurrenztechnologien wie Microsoft Silverlight konnten den Siegeszug und die Marktanteile von Flash nur unwesentlich beeinträchtigen. Mit HTML5 und seinen verwandten offenen Technologien scheint es nun aber einen ernsthaften Herausforderer für die Spitzenposition zu geben.

Und fast sieht es so aus, als ob auch Adobe die Leistungsfähigkeit von HTML5 erkannt hätte, da hier aktuell der Start von *Adobe Edge* (*http://labs.adobe.com/technologies/edge/*) vorbereitet wird. Dabei handelt es sich um eine Entwicklungsumgebung, die zwar Flash stark ähnelt, aber komplett auf HTML5, CSS3 und JavaScript basiert. Adobe ist wohl der Meinung, dass HTML5 und Flash erst einmal koexistieren können, aber vielleicht handelt es sich ja auch um ein erstes Anzeichen dafür, dass man Flash auf lange Sicht zugunsten von HTML5 und dem offenen Web auslaufen lassen will.

Hinweis

Damit will ich nicht gesagt haben, dass Flash direkt durch HTML5 ersetzbar wäre. Sie müssen die Schwächen der neuen Standards kennen und wissen, wann Alternativlösungen wie Flash möglicherweise besser geeignet sind. Ein Bereich, in dem Flash weiterhin äußerst praktisch ist, ist die Abwärtskompatibilität mit älteren Browsern, auf die ich nun näher eingehen werde.

1.3 So sorgen Sie für Abwärtskompatibilität

Wie bei den meisten anderen neuen Technologien treten auch bei der Arbeit mit HTML5 unvermeidlich Probleme mit der Abwärtskompatibilität auf. Bei HTML5 handelt es sich nicht um eine einzige monolithische Lösung, da Browser *Funktionen* und keine gesamten Spezifikationen unterstützen. Heute kann kein Browser von sich behaupten, den gesamten HTML5-Funktionssatz wirklich vollständig zu unterstützen. Zudem hat Microsoft im Internet Explorer, bei dem es sich in vielen Ländern und insbesondere der USA immer noch um den meistgenutzten Browser handelt, erst in der Version 9 damit begonnen, HTML5 zu unterstützen. Einige Funktionen wie WebGL sind hier überhaupt noch nicht implementiert und Microsoft hat bisher diesbezüglich auch noch keinerlei Interesse gezeigt. Aber selbst, wenn alle heute stärker verbreiteten Browser HTML5 und alle verwandten Standards vollständig unterstützen würden, dürften Sie die weiterhin benutzten älteren Browser immer noch nicht einfach vergessen. Angesichts von Browsern wie dem Internet Explorer 6, der auch ein gutes Jahrzehnt nach seiner Veröffentlichung 2001 immer noch recht weit verbreitet genutzt wird, können Sie noch über etliche Jahren hinweg nicht einfach davon ausgehen, dass Benutzer Ihrer Apps und Spiele auch alle Funktionen von HTML5 nutzen können.

1.3.1 Unterstützte Funktionen ermitteln

Niemand hat gesagt, dass heute programmierte Anwendungen und Spiele alle jemals erhältlichen Browser unterstützen müssen. Das würde uns nur ins Verderben und zu einer Menge Kopfschmerzen führen. Wir sollten diese Anwender aber auch nicht völlig vergessen. Wenigstens sollten wir versuchen, dem Benutzer mitzuteilen, ob er das Spiel oder bestimmte Funktionen nutzen kann und uns dann um das jeweils erkannte Problem kümmern. *Browser Sniffing* werden die Techniken genannt, mit denen die jeweils verwendeten Browser erkannt werden sollen. Dazu wird heute kaum noch der vom Browser im HTTP-Header übertragene *User Agent String* ausgewertet. Heute wird dazu vielmehr auf die *Feature Detection*, also die Ermittlung der vom genutzten Browser unterstützten Funktionen zurückgegriffen. Tests, mit denen die vorhandenen Eigenschaften, Objekte und Funktionen ermittelt werden, sind eine viel gesündere Strategie als die, sich einfach auf eine Zeichenkette zu verlassen, die zudem noch vom Nutzer geändert werden kann, und dabei einfach von einem bestimmten Satz unterstützter Funktionen auszugehen.

1.3.2 Die Modernizr-Bibliothek nutzen

Angesichts der vielen Unterschiede in den verschiedenen Implementierungen und den nur schwierig zu erkennenden Funktionen, handelt es sich bei der Funktionserkennung um keine leichte Aufgabe. Glücklicherweise müssen Sie das Rad üblicherweise nicht immer wieder neu erfinden, da eine Menge cleverer Tricks zur Erkennung der unterstützten Funktionen bereits entwickelt und in diversen Bibliotheken vereint wurden. Eine derartige Sammlung solcher Erkennungsfunktionen erhalten Sie mit der Modernizr-Bibliothek (*www.modernizr.com/*). Modernizr bietet Ihnen leicht nutzbare Methoden, mit denen Sie testen können, ob bestimmte Funktionen verfügbar sind. Sie können alles – vom `canvas`-Element und WebGL bis hin zur Web-Zeichensatzunterstützung und eine Unmenge an CSS-Funktionen – prüfen und auf diesem Weg Ausweichlösungen für nicht unterstützte Funktionen bereitstellen und die Leistungsfähigkeit Ihrer Apps nach und nach herabsetzen.

Die Modernizr-Bibliothek lässt sich höchst einfach nutzen. Sie müssen einfach nur eine kleine JavaScript-Datei einbinden, um prüfen zu können, welche Funktionen unterstützt werden. Wie Listing 1.5 zeigt, müssen Sie dazu nur die Eigenschaften des Modernizr-Objekts auswerten.

```
if (Modernizr.localstorage){
  // Lokaler Speicher wird unterstützt: weitermachen
} else {
  // Lokaler Speicher wird nicht unterstützt:
  // Ausweichlösungen verwenden
}
```

Listing 1.5 Unterstützte Funktionen mit Modernizr erkennen

Modernizr erweitert das `html`-Element zudem um CSS-Klassen, die angeben, welche Funktionen unterstützt werden und welche nicht. Da alle anderen Elemente Nachfahren des `html`-Elements sind, können Sie so Ihre Auszeichnungen entsprechend den unterstützten Funktionen anpassen. Wenn Sie beispielsweise `rgba()`-Farben nutzen wollen, um einen teilweise transparenten Hintergrund zu erstellen, und auf einen undurchsichtigen Hintergrund ausweichen wollen, wenn dieses Farbsystem nicht unterstützt wird, können Sie zu diesem Zweck einfach Formate wie in Listing 1.6 erstellen.

```
/* rgba()-Farben werden unterstützt */
.rgba .some_element {
  background-color : rgba(255,255,255,0.75);
}
```

```
/* rgba()-Farben werden nicht unterstützt */
.no-rgba .some_element {
  background-color : rgb(220,220,220);
}
```

Listing 1.6 CSS-Formate auf der Grundlage unterstützter Funktionen

Hinweis

Modernizr kann Ihnen nur mitteilen, ob etwas unterstützt wird. Bei fehlenden Funktionen bleibt es dem Entwickler überlassen, geeignete Maßnahmen zu ergreifen. Im folgenden Abschnitt werde ich einige der Optionen für den Umgang mit fehlenden Funktionen untersuchen.

1.3.3 Die Lücken mit Polyfills stopfen

Seit Anfang der 2000er-Jahre favorisiert ein beliebter Trend beim Hinzufügen neuer Funktionen zu Webseiten die sogenannte *progressive Verbesserung* (*progressive enhancement*). Diese Strategie fordert, dass sich Webseiten hinsichtlich der unterstützten Funktionen am kleinsten gemeinsamen Nenner orientieren sollen. Alle nicht allgemein unterstützten Technologien sollten nur zur Verschönerung der Website und nicht für kritische Funktionen verwendet werden. Dadurch wird gewährleistet, dass alle Nutzer auf die Website zugreifen und diese nutzen können. Wenn moderne Browser benutzt werden, hinterlässt die Seite damit einfach nur einen besseren Eindruck.

Die Strategie der progressiven Verbesserung ist zwar häufig vernünftig, zuweilen sind Sie aber einfach auf bestimmte Funktionen angewiesen. Wenn einige Browser diese Funktion nicht nativ unterstützen, muss diese Lücke selbst dann gestopft werden, wenn dazu alles andere als optimale oder sogar notdürftige Ausweichlösungen verwendet werden müssen. Diese Ausweichlösungen werden zuweilen *Polyfills* genannt, wobei diese Bezeichnung auf den Markennamen *Polyfilla* für Füll- und Reparaturspachtel zurückgeht und somit deren eigentliche Funktion irgendwie treffend beschreibt. Sie füllen die Lücken im Satz der unterstützten Funktionen bei der Ausführung Ihres Codes im Browser und überbrücken die Lücke zwischen den Spezifikationen und der harten Realität beim Umgang mit keineswegs perfekten Browsern.

Vor der Version 9 wurde `canvas` im Internet Explorer beispielsweise überhaupt nicht unterstützt. Es gab aber einige Polyfills, die auch in eigentlich veralteten Browsern noch eine gewisse `canvas`-Funktionalität bereitstellen konnten. Eines der ersten war das ExplorerCanvas-Projekt von Google (*http://code.google.com/p/*

explorercanvas/). Es benutzt VML (Vector Markup Language), eine alte, von Microsoft entwickelte XML-basierte Sprache, um das `canvas`-Element zu simulieren. Es bietet so viel Funktionalität für 2D-Zeichnungen, dass es erfolgreich in vielen Projekten genutzt werden konnte. Einige Funktionen werden jedoch nicht unterstützt, weil sich mit VML die `canvas`-Spezifikation nicht vollständig nachbilden lässt und es beispielsweise Muster und Beschnittpfade nicht unterstützt. Andere Ausweichlösungen greifen auf Flash oder Silverlight zurück, kommen dem vollständigen `canvas`-API noch ein wenig näher und unterstützen anspruchsvollere Funktionen wie den Zugriff auf Bilddaten und kombinierte Effekte. Angesichts all dieser unterschiedlichen Optionen ist die Auswahl der richtigen Ausweichlösung selbst keine einfache Aufgabe. Je nach Zielplattform werden selbst für die Polyfills zuweilen wieder Ausweichlösungen benötigt.

Um sich diese Aufgabe zu erleichtern, können Sie die in Modernizr integrierte Funktion zum Laden von Scripts verwenden. Diese basieren auf einer kleine Bibliothek namens *yepnope.js* (*http://yepnopejs.com/*), die dynamische Ladetechniken für Scripts mit Funktionstests wie bei Modernizr verbindet. Mit dieser Kombination können Sie Scripts automatisch auf der Grundlage unterstützter Funktionen bedingt laden. Listing 1.7 verdeutlicht, wie Sie verschiedene Scripts in Abhängigkeit von der Unterstützung einer Funktion laden können.

```
Modernizr.load([{
  test : Modernizr.localstorage,
  yep : "localstorage-data.js",
  nope : "cookiebased-hack-data.js"
}]);
```

Listing 1.7 Bedingtes Laden von Scripts mit Modernizr

Die Funktion `Modernizr.load()` übernimmt eine Liste von Objekten, die jeweils einen Funktionstest beschreiben und angeben, welche Scripts geladen werden sollen. Das bei der Eigenschaft `yep` angegebene Script wird geladen, wenn die Funktion unterstützt wird, das bei `nope` angegebene Script wird geladen, wenn die Funktion nicht unterstützt wird. *yepnope.js* unterstützt viele andere beeindruckende Funktionen, mit denen Sie die Ladephase vereinfachen und JavaScript- und CSS-Dateien auf der Basis unterstützter Funktionen, Modelle und Versionen des Browsers oder noch ganz anderer Kriterien laden können.

1.4 Die Entwicklung eines Spiels

Beginnend mit Kapitel 2 werde ich Sie in diesem Buch durch den kompletten Prozess der Entwicklung eines HTML5-Webgames begleiten. Ich werde Ihnen zeigen, wie Sie ein Spiel im »3-Gewinnt-Stil« mit Edelsteinen nach dem Vorbild von *Bejeweled* oder *Puzzle Quest* erstellen können. Derartige Spiele zum Zeitvertreib waren während des letzten Jahrzehnts auf vielen Plattformen sehr beliebt. Zudem existieren für diese Art von Spielen bewährte Verfahren, weshalb wir unsere Aufmerksamkeit auf die Nutzung von Webtechnologien im Kontext der Spieleentwicklung konzentrieren können. Zudem lassen sich derartige Games gut im Desktop-Browser und auf Mobilgeräten wie Smartphones und Tablets spielen, was uns wiederum Gelegenheit gibt, uns eingehender mit der Web-Spieleentwicklung für mehrere Plattformen zu befassen.

Das Spiel nutzt nicht nur Funktionen der HTML5-Spezifikation, sondern zur Gestaltung der Bedienoberfläche auch einige verwandte Technologien wie Webfonts und CSS3. Auch wenn das Spiel selbst vielleicht nicht gerade revolutionär sein mag, kann ich an seinem Beispiel doch viele der jüngsten Fortschritte aus dem Bereich der offenen Webtechnologien darstellen. Unter anderem werde ich das `canvas`-Element zum Erzeugen der Spielgrafiken verwenden und Ihnen zeigen, wie Sie mit dem HTML5-Element `audio` Soundeffekte hinzufügen können. Das fertige Spiel lässt sich in den gängigen Browsern spielen. Ich werde Ihnen aber auch zeigen, wie Sie dafür sorgen können, dass es sich ebenso gut auf Mobilgeräten und selbst im Offline-Betrieb nutzen lässt. Ich werde darstellen, wie Sie Web Storage dazu benutzen können, um Highscore-Daten zu speichern und das Spiel nach dem Speichern wieder an derselben Stelle fortsetzen zu können.

Mit dem `canvas`-Element lassen sich zwar interessante Grafiken dynamisch erstellen, es eignet sich aber nicht besonders für die Erstellung von Bedienoberflächen. Für diese Aufgabe benötigen Sie aber auch keine wirklich neuen Hilfsmittel, denn diese bieten uns bereits das traditionelle HTML in Verbindung mit CSS. Mit den jüngsten Erweiterungen der CSS-Spezifikation können Sie Animationen, Transformationen und andere Funktionen hinzufügen, die der Bedienoberfläche zusätzlich Leben einhauchen. In den Kapiteln 6 und 7 werde ich Ihnen zeigen, wie Sie das Anzeigemodul mit dem `canvas`-Element erstellen können. Später werde ich im Kapitel 11 noch darstellen, wie Sie WebGL nutzen können, um das Spiel um 3D-Grafiken zu erweitern.

In Kapitel 13 werde ich Ihnen zeigen, wie Sie eine einfache Chat-Anwendung mithilfe von WebSockets erstellen können. Zu diesem Zweck werde ich auch

darstellen, wie Sie eine kleine Serveranwendung mit dem Framework *Node.js* (*http://nodejs.org/*) erstellen können. Sie sollten dabei aber beachten, dass WebSockets bisher nur von wenigen Browsern unterstützt werden, und dass es noch eine Weile dauern kann, bevor sie wirklich für einen verbreiteten Einsatz reif sind.

1.5 Zusammenfassung

Es war ein holpriger Weg, aber im Augenblick sieht es doch so aus, als ob sich HTML und das Web allgemein wieder in der richtigen Spur befinden würden. Das WHATWG hat für eine neue Perspektive im Standardisierungsprozess gesorgt, und Webentwickler beginnen nun, die Früchte dieser Anstrengungen in Form einer Vielzahl neuer Werkzeuge und eines HTML-Standards zu genießen, der eher der heutigen Nutzung des Internets entspricht. Wie immer hat man bei der Nutzung der neuen Funktionen zwar mit den älteren Browsern zu kämpfen, aber es stehen bereits viele Polyfills bereit, die Sie nutzen können, um für Browser-übergreifende Kompatibilität zu sorgen. Helferlein wie Modernizr können Ihnen die Last dieser Aufgabe noch weiter erleichtern.

Viele der neuen Erweiterungen sind deshalb insbesondere für Spieleentwickler interessant, weil nun echte Alternativen zu Flash-basierten Webspielen existieren. Canvas und WebGL sorgen für Dynamik und Hardware-beschleunigte Grafik, das `audio`-Element bringt endlich die native Audiounterstützung und mit WebSockets lassen sich nun Multiplayer-Erlebnisse realisieren, die deutlich näher zu Desktop-Spielen aufschließen, wie sie erst vor einigen Jahren möglich wurden. Mit den Fortschritten in anderen verwandten Bereichen wie CSS und der zunehmenden Unterstützung von Webfonts können wir unter Einsatz offener, standardisierter Werkzeuge nutzerfreundlichere Bedienoberflächen erstellen.