Einführung in Canvas

Da ich in letzter Zeit über einige Projekte geschrieben habe, die auf Canvas beruhen, möchte ich über dessen Möglichkeiten und Funktionen in Form eines kleinen Tutorials berichten. In meinen Ausführungen orientiere ich mich an der Umsetzung im Mozilla Firefox und benutze als Grundlage die englischsprachige Dokumentation von Mozilla

Was ist Canvas, wer unterstützt es und woher kommt es?

Canvas ist ein grafisches Element, das mit HTML5 eingeführt werden wird. Mit Canvas ist es möglich Grafiken, Graphen und einfache Animationen mit Hilfe von Skript-Sprachen (vorrangig Javascript) dynamisch zu erzeugen. (Das Thema Animationen lasse ich in diesem Artikel außen vor.) Der aktuelle Spezifikationvorschlag ist auf der Seite http://www.whatwg.org einzusehen.

Das Canvas-Element wurde ursprünglich von Apple im OS X Dashboard zum skriptbasierten Erstellen von Minianwendungen (sogenannten Widgets) ins Leben gerufen und wurde somit vom hauseigenen Browser Safari unterstützt. Der Support seitens Mozilla begann mit der Gecko-Engine in Version 1.8, die im Firefox 1.5 verwendet wurde. Opera kommt ab Versionsnummer 9 mit Canvas zu recht.

Einzig der Internet Explorer (in allen Varianten) bietet keine Unterstützung des Elementes. Bisher ist dies auch nicht geplant, wie ich bereits in meinem Beitrag über die IE8-Beta erwähnte. Abhilfe schafft das ExplorerCanvas-Skript, das dem Internet Explorer die fehlende Unterstützung nachliefert.

Einbau in den Quellcode

Wie jedes andere HTML-Element wird das Canvas-Element im Quellcode eingebettet. Hierbei ist zu beachten, dass man die Ausmaße der anzuzeigenden Grafik über *width* und *height* angibt.

```
<canvas id="canvas1" width="400" height="100"></canvas>
```

Der eigentliche Canvas-Code ist in Javascript geschrieben und befindet sich im entsprechenden HTML-Tag. Es empfiehlt sich den Canvas-Code in eine Funktion zu packen, die dann nach dem Laden der Seite aufgerufen wird. Dies erreicht man durch den *onload*-Befehl, den man entweder direkt in HTML auf das body-Element anwendet oder per Javascript über Event-Methoden anhängt.

in HTML:

```
<body onload="draw();">
```

in Javascript:

```
if (window.addEventListener)
{
  addEventListener("load", draw, false);
}
else
{
  attachEvent("onload", draw);
}
```

Im Javascript-Bereich wird die von mir *draw* genannte Funktion definiert.

Canvas-Methoden

Ich habe eine Demonstrationsseite erstellt, in der alle hier vorgestellten Methoden angewandt werden. Hinweis: Scheinbar nicht alle Beispiele sind im IE mit ExplorerCanvas nachvollziehbar.

Eine draw-Funktion sieht beispielsweise wie folgt aus:

```
function draw()
{
  var canvas = document.getElementById('tutorial');

  if (canvas.getContext)
  {
    canvas = canvas.getContext('2d');
    canvas.fillStyle = "rgb(200,0,0)";
    canvas.fillRect(10, 10, 55, 55);
  }
}
```

Die Abfrage nach der Funktion *getContext* dient dazu, festzustellen ob der Browser Canvas unterstützt und es somit zu keinen JS-Fehlern bei fehlendem Support kommt. Als erstes weist man einer Variable den Pfad zum Canvas-Element zu, in das die Grafik hineingezeichnet werden soll, und wendet dann darauf die Methode *getContext('2d')* an, mit der das Zeichnen eingeleitet wird.

Rechtecke und Gestaltungsmöglichkeiten

Wie man dem oberen Beispiel entnehmen kann, werden einfarbige, gefüllte Rechtecke mit der Funktion *fillRect* gezeichnet.

Parameter: fillRect(x-Koordinate, y-Koordinate, Breite, Höhe)

Mittels *fillStyle* kann man die Farbe des darzustellenden Rechtecks festlegen. Erlaubt sind die gewohnten Angaben mit Farbnamen, im rgb(a)-Format und Hexadezimalsystem.

Es ist ebenso möglich ein Rechteck nur mit Umrissen zu zeichnen. Dies wird über die Funktionen *strokeRect* und *strokeStyle* realisiert, die genauso benutzt werden können, wie die Methoden für ausgefüllte Rechtecke.

Linien und freie Formen

Das Erstellen von Linien geschieht mit der Methode *lineTo*, die als Parameter den Endpunkt der zu zeichnenden Gerade enthält.

Parameter: lineTo(x-Koordinate, y-Koordinate)

Um den Startpunkt der Geraden zu verlegen, nutzt man die Funktion *moveTo*, die ebenfalls einen x- und y-Wert als Parameter erwartet. Durch *moveTo* kann man sich frei im (virtuellen) "Raum" bewegen.

Parameter: moveTo(x-Koordinate, y-Koordinate)

Um zusammengesetzte Linien zu zeichnen verwendet man die Path-Funktion, die mit *beginPath* eingeleitet wird. Die danach folgenden Anweisungen wie *lineTo* geben die Eckpunkte des Pfades an.

Mit den anschließend notierten Methoden *fill* oder *stroke* entscheidet man, ob die neu geschaffene Form ausgefüllt oder nur umrandet dargestellt werden soll. Es existiert auch eine *closePath*-Method zum Festlegen des Pfadendes, die bei Verwendung von *fill* getrost weggelassen werden kann. Hierbei werden der erste und der letzte Punkt automatisch miteinander verbunden und es ergibt sich somit eine selbstdefinierte "Form".

Hier ein Codeschnipsel zur Verdeutlichung:

```
canvas.fillStyle = "rgba(0,150,0,0.7)";
canvas.beginPath();
canvas.moveTo(50, 75);
canvas.lineTo(100, 125);
canvas.lineTo(100, 25);
canvas.fill();
```

Kreise, Bögen und Kurven

Doch Canvas ist nicht nur auf Geraden und rechteckige Figuren beschränkt. Mit dem Befehl *arc* zeichnet man Kreise – wie bei den Rechtecken entweder umrahmt oder ausgefüllt.

Parameter: arc(x-Koordinate Kreismittelpunkt, y-Koordinate Kreismittelpunkt, Radius, Startwinkel, Endwinkel, Uhrzeigersinn [true / false]);

Bei Kreisen gibt es zwei Dinge zu beachten. Zum einen sind der Start- und Endwinkel in der Einheit Bogenmaß anzugeben, also nicht im Gradmaß. Der 360°-Vollwinkel ist dementsprechend 2*Math.PI. Zum anderen muss man festlegen, ob man die Winkelangaben mit oder entgegen dem Uhrzeigersinn betrachtet.

Ein gefüllter Vollkreis wird beispielsweise mit folgendem Code erzeugt:

```
canvas.fillStyle = "rgb(0,100,0)";
canvas.beginPath();
canvas.arc(200, 75, 30, 0, 2*Math.PI, true);
canvas.fill();
```

Das Darstellen von komplexeren Kurven geschieht mit den beiden Methoden *bezierCurveTo* und *quadraticCurveTo*.

Parameter: quadraticCurveTo(x-Koordinate Kontrollpunkt, y-Koordinate Kontrollpunkt, x-Koordinate Endpunkt, x-Koordinate Endpunkt)

Parameter: bezierCurveTo(x-Koordinate Kontrollpunkt 1, y-Koordinate Kontrollpunkt 1, x-Koordinate Kontrollpunkt 2, y-Koordinate Kontrollpunkt 2, x-Koordinate Endpunkt, x-Koordinate Endpunkt)

Während bei der quadratischen Bezierkurve nur ein Kontrollpunkt zur Verlaufsbestimmung benutzt wird, werden bei der *bezierCurveTo*-Funktion 2 Kontrollpunkte benutzt. Jedoch ist das freie Zeichnen von Bezierkurven, also ohne visuellen Vorschau, äußert zeitaufwendig und eine Herausforderung.

Hier ein Beispiel dazu:

```
canvas.fillStyle = "rgb(245,0,0)";
canvas.beginPath();
canvas.moveTo(75, 40);
canvas.bezierCurveTo(75, 37, 70, 25, 50, 25);
canvas.bezierCurveTo(20, 25, 20, 62.5, 20, 62.5);
canvas.bezierCurveTo(20, 80, 40, 102, 75, 120);
canvas.bezierCurveTo(110, 102, 130, 80, 130, 62.5);
canvas.bezierCurveTo(130, 62.5, 130, 25, 100, 25);
canvas.bezierCurveTo(85, 25, 75, 37, 75, 40);
canvas.fill();

canvas.beginPath();
canvas.moveTo(200, 25);
canvas.quadraticCurveTo(150, 25, 150, 62.5);
canvas.quadraticCurveTo(150, 100, 175, 100);
canvas.quadraticCurveTo(175, 120, 155, 125);
canvas.quadraticCurveTo(185, 120, 190, 100);
canvas.quadraticCurveTo(250, 100, 250, 62.5);
canvas.quadraticCurveTo(250, 25, 200, 25);
canvas.stroke();
```

Weitere gestalterische Möglichkeiten

Desweiteren bietet Canvas grundlegende Funktionen zum Manipulieren der Linien und Linienübergänge.

Die Linienstärke wird über *lineWidth* festgelegt. Mit *lineCap* bestimmt man das Aussehen der Linienenden. Dazu stehen die folgenden Varianten zur Auswahl: butt (stumpf), round (abgerundet) und square (Überhang). Sowohl das Überhang als auch der Halbkreis haben eine Höhe respektive einen Radius entsprechend dem halben Wert der Linienbreite. butt entspricht dem Standardwert.

```
canvas.lineWidth = 15;
canvas.lineCap = "round";
canvas.beginPath();
canvas.moveTo(110, 20);
canvas.lineTo(110, 130);
canvas.stroke();
```

Mit *lineJoin* bestimmt man das Aussehen der Ecken zweier verbundener Linien. Hierbei stehen die Attribute round (abgerundet), bevel (abgeschrägt) und miter (eckig) zur Verfügung. Der default-Wert ist round.

```
canvas.lineJoin = "bevel";
canvas.beginPath();
canvas.moveTo(200, 60);
canvas.lineTo(225, 85);
canvas.lineTo(250, 60);
canvas.lineTo(275, 85);
canvas.lineTo(300, 60);
canvas.stroke();
```

Farbverläufe

Canvas ermöglicht Formen sowohl mit linearen als auch mit radialen Farbverläufen auszufüllen. Dazu wird einer Variable ein Verlauf zugewiesen. Die dazugehörigen Methodenaufrufe lauten *createLinearGradient* und *createRadialGradient*. Jeweils wird ein Startpunkt beziehungsweise -kreis und ein Endpunkt beziehungsweise -kreis als Parameter angeben, die den Bereich angeben, in dem der Farbverlauf sich erstreckt.

Parameter: createLinearGradient(x-Koordinate Startpunkt, y-Koordinate Startpunkt, x-Koordinate Endpunkt, y-Koordinate Endpunkt)

Parameter: createRadialGradient(x-Koordinate Mittelpunkt Startkreis, y-Koordinate Mittelpunkt Startkreis, Radius Startkreis, x-Koordinate Mittelpunkt Endkreis, y-Koordinate Endpunkt Startkreis, Radius Endkreis)

Anschließend werden die einzelnen farblichen Zwischenstufen mittels *addColorStop* definiert. Der Funktion *fillStyle* wird im Anschluß die Variable, die den Verlaufbeinhaltet, zugewiesen.

Parameter: addColorStop(Position [Werte zwischen 0-1 sind erlaubt], Farbdefinition);

Hier ein Beispiel eines Farbverlaufsskriptes:

```
var lingrad = canvas.createLinearGradient(0, 0, 0, 150);
lingrad.addColorStop(0, "#0099cc");
lingrad.addColorStop(0.5, "#fff");
lingrad.addColorStop(0.5, "#99cc00");
lingrad.addColorStop(1, "#0099ff");

canvas.fillStyle = lingrad;
canvas.fillRect(10, 10, 130 ,130);
```

Muster

Neben den bisher genannten Füllmöglichkeiten gibt es noch eine weitere Art die Fläche einer Form auszufüllen. Die Rede ist von Mustern, die aus gewöhnlich Bilddateien generiert werden. Dazu erstellt man zuerst ein neues *Image*-Objekt, weist diesem den Pfad zur zu verwendenden Grafikquelle zu und definiert eine onload-Funktion, die nach erfolgreichem Laden des Bildes mittels *createPattern* ein neues Muster erstellt.

Parameter: createPattern(Image-Objekt, Art der Wiederholungen)

Neben der Wiederholungsart “repeat” gibt es – ähnlich der CSS-Eigenschaft background-repeat – weitere Varianten um beispielsweise Wiederholungen nur in x- oder y-Richtung zu erzeugen. Jedoch sind diese noch nicht implementiert.

Der folgende Javascript-Code soll das Vorgehen aufzeigen:

```
var img = new Image();
img.src="wallpaper.png";
img.onload = function()
             {
               var pattern = canvas.createPattern( img, "repeat" );

               canvas.fillStyle = pattern;
               canvas.fillRect( 0, 0, 400, 150 );
             }
```

Ausblick

Mozilla wird mit der Version 3.0 des Firefox-Browser, der auf der Gecko-Enging 1.9 basiert, in Sachen Gestaltungsmöglichkeiten von Canvas einen Schritt weiter gehen und das Rendern von Text erlauben. Dazu werden die folgenden mozilla-spezifischen Methoden eingeführt:

mozDrawText()	zeichnet Text
mozMeasureText()	gibt die Breite des zu zeichnenden Textes zurück
mozPathText()	definiert einen Textpfad
mozTextAlongPath()	zeichnet Text entlang eines vorher definierten Pfades