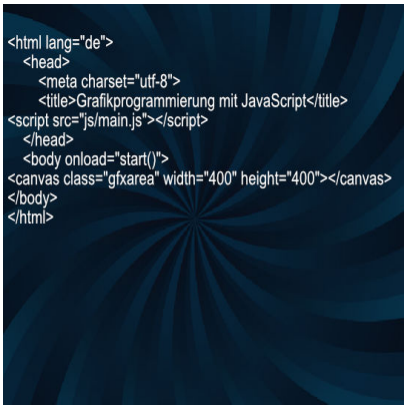


Webentwicklung Grundkurs Teil 6

Description

Nachdem wir im letzten Teil ein wenig in die Tiefen der JavaScript Programmierung eintauchten, gehen wir heute an die Oberfläche. Statt Daten von A über B nach C zu schaufeln, werfen wir einen Blick auf die Grafikprogrammierung.

Canvas kann was

Die meisten Jobs in der Programmierung befassen sich damit, Daten zu verarbeiten, zu speichern oder zu kopieren. Auf Dauer ist das in etwa so spannend wie Kühen beim Grasen zuzuschauen. Auch deshalb befassen sich viele Programmierer in ihrer Freizeit mit unterhaltsameren Aufgaben. Grafikprogrammierung, Spieleprogrammierung und dergleichen sind sexy.

Im letzten Teil haben wir als Container für die Inhalte ** ** benutzt. Für die Grafikausgabe nimmt man hingegen **<canvas>**. Canvas bedeutet „Leinwand" und beschreibt schon ganz gut, worum es geht. Anhänger von Flash werden gleich an die „Bühne" erinnert. Hier werden Grafiken und Animationen dargestellt. Im heutigen Teil werden wir zwei einfache Grafiken zeichnen und einen Schriftzug im Kreis rotieren lassen. Das Beispiel kann hier angesehen werden.

Zugegeben, es verleitet einen noch nicht zum Eisprung, aber als Ausgangspunkt für eigene Grafikergüsse ist es nahezu ideal. Wie auch im letzten Tutorial brauchen wir eine **index.html** und eine **main.js**.

index.html

```
<!DOCTYPE html>
<html lang="de">
    <head>
        <meta charset="utf-8">
        <title>Grafikprogrammierung mit JavaScript</title>
  <script src="js/main.js"></script>
    </head>
```

```
    <body onload="start()">
  <canvas class="gfxarea" width="400" height="400"></canvas>
 </body>
</html>
```

Der Aufbau der HTML ist mal wieder unspektakulär. Nach dem Titel laden wir die JS-Datei ein. Damit es nicht gleich zum Konflikt kommt, gibt es im <body> den Zusatz *onload=”start()”*. Das bedeutet, dass die Funktion *start* in der JS-Datei aufgerufen wird, sobald der Inhalt geladen wurde. Das ist wichtig, weil das JavaScript auf die Eigenschaften vom canvas zugreift und es eine Fehlermeldung (in der Konsole) gibt, wenn dies noch nicht geladen wurde.

Das canvas hat drei Eigenschaften. Neben Höhe und Breite haben wir noch die Klasse **gfxarea** definiert. Dies brauchen wir, um darauf zugreifen zu können.

main.js

Wie so oft geht es hier um das Prinzip. Rotation und andere Dinge kann man auch anders lösen. Wichtig ist, dass wir den Fokus darauf legen, was wichtig ist. Zunächst die ganze Datei:

```
"use strict";
var deg = 0;
var canvas, width, height, context;

function start()
{
 canvas = document.querySelector('.gfxarea');
 width = canvas.width = window.innerWidth;
 height = canvas.height = window.innerHeight;
 context = canvas.getContext('2d');

 loop();
}

function loop()
{
 // Bildschirm löschen
 context.clearRect(0, 0, canvas.width, canvas.height);

 // Hintergrund füllen
 context.fillStyle = 'rgb(12,12,12)';
 context.fillRect(0, 0, width, height);

 // BGMM Rechteck
 context.fillStyle = 'rgba(151, 227, 82, 1)';
 context.fillRect(64, 64, 128, 128);

 // Weißer Rahmen
 context.strokeStyle = 'rgba(255, 255, 255, 1)';
 context.lineWidth = 3;
 context.strokeRect(60, 60, 136, 136);

 if (deg < 360)
```

```
 {
  deg += 0.05;
 } else {
  deg = 0;
 }

 // Schriftzug
 context.save();
 context.translate(128, 128);
 context.rotate(deg);
 context.font  = '18px verdana';
 context.fillStyle = 'rgba(255, 0, 0, 1)';
 context.textAlign = „center";
 context.beginPath();
 context.fillText('Bytegame.de', 0, 0);
 context.fill();
 context.restore();

 window.requestAnimationFrame(loop);
}
```

Erneut haben wir einen kleinen Kopfteil. Anschließend folgen zwei Funktionen.

```
"use strict";
var deg = 0;
var canvas, width, height, context;
```

Hier definieren wir zunächst die globalen Variablen. Die Variable **deg** brauchen wir für die Drehung. Wir definieren es bei 0°. Die anderen Variablen brauchen wir für den Start.

```
function start()
{
 canvas = document.querySelector('.gfxarea');
 width = canvas.width = window.innerWidth;
 height = canvas.height = window.innerHeight;
 context = canvas.getContext('2d');

 loop();
}
```

Das ist noch sehr einfach. Wir lesen die Eigenschaften unserer **gfxarea** aus. canvas.getContext(‘2d’) holt uns einen 2d-Kontext her. Das brauchen wir, da wir nicht im canvas arbeiten, sondern in einem Kontext davon. Mit loop() rufen wir die zweite Funktion auf.

Schauen wir uns zunächst die letzte Zeile der Funktion an:

```
window.requestAnimationFrame(loop);
```

Hier wird die Funktion erneut aufgerufen. Da wir eine Animation haben, brauchen wir die erste Zeile der Funktion:

```
context.clearRect(0, 0, canvas.width, canvas.height);
```

Damit löschen wir den ganzen Kontext. Würden wir das nicht tun, würden wir jeden Animationsschritt sehen. Das sieht leider furchtbar aus.

```
context.fillStyle = 'rgb(12,12,12)';
context.fillRect(0, 0, width, height);
```

Zunächst definieren wir eine Farbe. Anschließend zeichnen wir ein Rechteck. Die RGB-Farbe 12, 12, 12 ergibt ein ziemlich dunkles Grau.

Wer sich mit GML auskennt, fühlt sich gleich an die Befehle im Draw-Event erinnert. ?

```
context.fillStyle = 'rgba(151, 227, 82, 1)';
context.fillRect(64, 64, 128, 128);
```

Mit den identischen Befehlen, aber mit anderen Werten, zeichnen wir ein grünes Rechteck.

```
context.strokeStyle = 'rgba(255, 255, 255, 1)';
context.lineWidth = 3;
context.strokeRect(60, 60, 136, 136);
```

Mit diesen drei Zeilen definieren wir den weißen Rahmen. Bitte beachten, dass wir hier statt *fill stroke* verwenden!

Bis hierhin haben wir zwei Grafiken gezeichnet. Nun kommt die eigentliche Animation.

```
if (deg < 360)
{
 deg += 0.05;
} else {
 deg = 0;
}
```

Damit erzeugen wir den Winkel. Mit jedem Durchlauf wird 0,05 hinzu addiert. Bei 360 wird der Wert wieder auf 0 gesetzt. Das ist nicht die beste Lösung, aber sie funktioniert und wir können uns auf das wesentlich wichtigere Konzentrieren:

```
// Schriftzug
context.save();
context.translate(128, 128);
context.rotate(deg);
context.font  = '18px verdana';
context.fillStyle = 'rgba(255, 0, 0, 1)';
context.textAlign = „center";
context.beginPath();
context.fillText('Bytegame.de', 0, 0);
context.fill();
context.restore();
```

Erklärung

context.save() speichert die aktuellen Einstellungen der Zeichenwerkzeuge wie fill und stroke. Mit der

letzten Zeile, **context.restore()**, holen wir uns die Einstellungen zurück.

context.translate(128, 128) ordnet die Position (128, 128) auf der Zeichenfläche neu zu. Da soll später die Schrift zentriert dargestellt werden.

context.rotate(deg) definiert, wie der Schriftzug gedreht werden soll. Die Variable *deg* haben wir weiter oben definiert.

Mit **context.font** definieren wir Schriftgröße und Schriftart.

context.fillStyle füllt die Schrift mit der gewählten Farbe. Im Beispiel verwenden wir Rot.

context.textAlign stellt die Ausrichtung der Schrift ein. Wir zentrieren den Text.

context.beginPath() erzeugt einen Pfad. Beginnend davon könnten wir nun auch andere Dinge, etwa Linien, zeichnen.

context.fillText('Bytegame.de', 0, 0) schreibt den Text an die vorgegebene Stelle (0, 0). Die Koordinaten beziehen sich auf **context.translate**.

context.fill() füllt die aktuelle Zeichnung (Pfad).

Das war es schon. Mit dem erlernten können wir verschiedene Formen zeichnen und über den Bildschirm bewegen.

Ausblick

In der nächsten Folge bleiben wir bei JavaScript schauen uns an, was es eigentlich mit dem JQuery auf sich hat, von dem man so viel hört.

Überblick Webdev-Serie

Webentwicklung Grundkurs Teil 1 – Einstieg
Webentwicklung Grundkurs Teil 2 – Aufbau von Webseiten
Webentwicklung Grundkurs Teil 3 – Datei- und Ordnerstrukturen
Webentwicklung Grundkurs Teil 4 – Einstieg in JavaScript
Webentwicklung Grundkurs Teil 5 – Datenverarbeitung und Formulare mit JavaScript

Überblick Interviews

Interview mit Magnus Reiß – Webgamers
Interview mit Wolfgang Scheidle – Tischtennis Manager
Interview mit Warg – Drifting Souls II

Date Created
20. Dezember 2019
Author
sven