In 15 Tutorials zum Spiel

Kiddy & Co, 2019

Rudi Maisriml - scriptkiddy 15 Tutorials.odt

[Inhaltsverzeichnis]

[Das SkriptKiddy-Projekt]

Programmieren mit Kids. Geht das überhaupt?

Und wenn ja: wie?

Wir von **Kiddy & Co** haben vor einiger Zeit ein Projekt gestartet, in dem wir Jugendlichen zeigen wollten, **wie man programmieren lernt**. Dazu haben wir in unserem Internetcafe 2018 zwei Workshops für Kids angeboten, in denen sie Grundkenntnisse zum JavaScript-Programmieren und Spiel-Entwicklung erlernen konnten.

Gemeinsames Ziel war es, selbst ein kleines Online-Game zu entwickeln - wobei die Kids sehen konnten, worauf es beim Programmieren ankommt: **Phantasie, logisches Denken, Forschergeist, Fleiss, aber auch Mathematik und Englisch Grundkenntnisse**.

Die am Projekt beteiligten Jugendlichen erwiesen sich als sehr interessiert an der Materie; alle Beteiligten mussten aber auch erkennen, dass Lernfortschritte in diesem Bereich viel Geduld benötigen und oft nur kleine Schritte gemacht werden können.

Daher haben wir uns entschlossen, **dieses Tutorial nochmals zu überarbeiten**, mit dem Ziel es **einfacher** und besser nachvollziehbarer zu machen. Im Hinblick auf weitere Workshops in der Jugendarbeit sollen sie in gewissen Maße auch der Vorbereitung der Kids dienen. So können sich Interessierte schon vorab mit den Inhalten befassen.

Dateien im Internet: http://www.gorilla.at/scriptkiddy/

Programmieren macht Spaß!

Wenn du es geschafft hast, **die ersten Zeilen Code** zum Laufen zu bringen hast du bereits das Wichtigste gelernt!

Das Tolle am Programmieren ist, dass man keine besondere Voraussetzungen benötigt um damit zu beginnen. Wenn man eine gute Idee hat, kann man bereits loslegen. Die ersten selbstgeschriebenen Programme werden natürlich recht einfach sein, die Freude darüber, wenn etwas funktioniert, tut dem aber keinen Abbruch!

Programmieren ähnelt in vielem dem Spielen mit Lego-Bausteinen. Statt den Bausteinen verwendet man hier Funktionen und Variablen. Immer hat

man jedoch die Möglichkeit zur freien Gestaltung der eigenen Idee, die Stein um Stein entwickelt wird.

Was ist eigentlich programmieren?

Obwohl natürlich jeder weiß, was programmieren ungefähr ist, fällt die Antwort auf diese Frage trotzdem nicht so leicht. Wichtig ist es zu wissen, dass es **verschiedene "Programmiersprachen"** gibt - im Grunde sind sie sich aber sehr ähnlich. Die wesentlichen Befehle und Abläufe sind überall recht gleich.

Mit dem Programmieren wollte man zuerst **Rechen-Probleme lösen**, die ohne Computer vielmehr Aufwand gewesen wären. So waren die ersten Computerprogramme zumeist **mathematische Logik- und Rechenmaschinen**. Sie erleichterten den Menschen die Arbeit, weil sie kompliziertes Berechnungen in Blitzgeschwindigkeit ausführten.

Ein modernes **Computerspiel** ist eigentlich auch nur eine mathematische Rechenmaschine. Der Spieler merkt zwar nichts davon, das Spiel rechnet aber unentwegt alles Mögliche aus - z.B. die Entfernung des Gegners, die Kraft eines Stoßes - und berechnet dazu auch noch das entsprechende Bild.

Ein Autorennspiel beispielsweise ist eine komplizierte Abfolge von tausenden programmierten Rechnungen - und das 50 mal pro Sekunde. So wird verständlich, warum an der Programmierung eines bekannten Spiels z.B. 20 Leute jahrelang arbeiten können.

Kann man ein Spiel auch ganz alleine programmieren?

Ja! Vorausgesetzt das Spiel ist nicht zu aufwändig und man hat genug Zeit. **Ein kleines Online-Game** lässt sich mit einer guten Idee und den nötigen Programmier-Erfahrung recht einfach umsetzten. Trotzdem sollte man das

Spiele-Programmieren nicht unterschätzen. Es gilt nämlich als **Königsdisziplin** unter den Programmierern.

Das kleine Online-Game mit dem wir uns beschäftigten wird am Ende dieses SkriptKiddy-Projekt vorgestellt. Mit den einzelnen Lektionen eignen wir uns die Grundkenntnisse an, die es braucht um ein derartiges Programm selbst zu entwickeln: **Und zwar von Anfang an!**

Programmieren lernen mit JavaScript - Los gehts!

Für unseren Workshop haben wir uns für JavaScript entschieden. Es ist leicht zu erlernen und beinahe auf jedem Computer möglich.

Die folgenden Seiten erklären **Schritt für Schritt,** wie man erste einfache Programme schreibt. Dabei gehen wir davon aus, dass du schon gewisse Computer-Grundkenntnisse hast - z.B. weißt, wie man etwas speichert, kopiert oder Bilder bearbeitet.

Programmieren lernt man am besten, indem man **viele einfache Programme** schreibt. So bekommt man Schritt für Schritt alle Fertigkeiten, die man für ein größeres Projekt - z.B. ein kleines Spiel - benötigt.

Das ist auch der Aufbau unserer kleinen Tutorialreihe. Hier wird anhand kleiner Programme, die Grundlagen für späteres selbstständiges Programmieren gelegt.

1 | Los gehts (mit JavaScript, HTML und CSS) - für NOOBS

Unsere Programmiersprache ist hier JavaScript. Der Vorteil von JavaScript besteht darin, dass man damit Programme für das Internet - also zum Beispiel für eine Website - schreiben kann. Dabei muss man ganz einfach in eine HTML-Datei den entsprechenden JavaScript-Block hineinschreiben.

Wenn du bereits die Grundlagen von HTML-Codes kennst, überspringe dieses Tutorial einfach. Wir erzeugen nämlich hier nur eine einfache HTML-Seite.

Was ist eine HTML-Datei?

TIPP: Kopiere dir den Code einfach aus dem folgenden heraus. .

Internetseiten sind HTML-Dateien. Nimm einmal einen Texteditor (z.B. Notepad) und schreibe (oder kopiere) dort folgendes hinein:

CODE:

```
<!-- Du kannst es hier raus kopieren -->
<html>
<head>
</head>
<body>
<h1>Hallo Welt!</h1>
</body>
</html>
<!-- Ende  -->
```

code_bsp01.htm

TIPP: Datei speichern als HTML-Datei: du musst im Editor den Dateityp wechseln. Nun speicherst du diese als „meinedatei.htm“. So erhältst du eine Internetdatei statt einer txt-Datei.

Das speicherst du nun ab als **meinedatei.htm** ab. Nachdem du das gemacht hast, öffnest du die Datei mit deinem Browser - z.B. Google Chrome.

Gratulation: **Du hast eine Internetseite gemacht**. Aber wahrscheinlich wusstest du das ohnehin schon.

HTML dient uns hier vorerst nur dazu, eine Oberfläche zu bekommen, in die wir später unsere Javascript-Programme hineinschreiben. HTML ist sehr einfach und selbsterklärend; **für unsere Zwecke brauchen wir hier einmal nicht mehr zu wissen**.

TIPP: Im Internet findest du unzählige sehr gute Seiten zum Erlernen von HTML & Co.

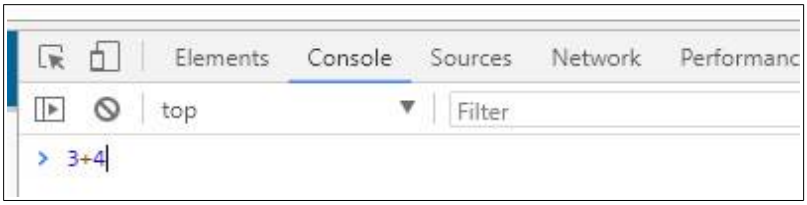
Wenn du bei HTML Hilfe brauchst suche doch mal im Internet unter HTML Lernen. Auf Seiten wie "https://html-einfach.de/" finden sich auch Einsteiger gut zurecht.

In diesem Tutorial erklären wir diese Sachen nur am Rande.

CSS ist ein zusätzlicher Teil von Internetseiten. Dieser bestimmt, wie die Seiten aussehen. Unser schlichtes **Hallo Welt** könnte damit farbig, unterstrichen oder blinkend verschönert werden. Hier gehts also nur um die Optik. **Beim Programmieren betrifft uns CSS vorerst überhaupt nicht**. Man muss nur wissen, dass es das gibt.. und dass wir später eine Zeile CSS beim Einbauen des CANVAS benötigen.

TIPP: Console! Die Console ist Teil deines Browser. Zum Programmieren ist sie extrem praktisch, weil sie das Programm bei der Arbeit zeigt und auch alle Werte ausgeben kann.

Mit der Tastenkombination [Strg]+ [Umschalt]+j kommt man in den meisten Browsern (z.B. **Google Chrome**) in eine Darstellung, die du noch wahrscheinlich noch nicht kennt. Hier hast du Zugriff auf die CONSOLE. Darin werden alle Fehlermeldungenund die Seitenstruktur angezeigt. Da kannst auch direkt dort mit JavaScript herumprogrammieren. Versuche es mal und schreibe dort hinein 3+4 und drücke dann [ENTER]. Als Ergebnis sollte nun 7 stehen.



Firefox: Tastenkombination [Strg] + [Umschalt] + K

TIPP: Maus!
Die RECHTE Maustaste ist nützlich **zum Betrachten von InternetCode**.

Rechte Maustaste: sicher weißt du, dass durch Drücken der rechten Maustaste über einem Seitenabschnitt, der gesamte Seitenquellcode - also die HTML-Codes und JavaScript - angezeigt wird. So kannst du dort einfach Code rauskopieren und in dein Notepad oder Webeditor einfügen.

2 | Hallo Welt! - mit JavaScript

Wenn du nun schon eine einfache HTML-Datei erzeugen kannst gehts nun los mit dem eigentlichen Programmieren:

Wir werden in diesem Beispiel JavaScript dazu verwenden **eine automatische Textzeile zu erzeugen**. Derartiges findet man übrigens in vielen Internetseiten. Wann immer dort maschinengenerierter Text erscheint, wurde er wahrscheinlich mit JavaScript programmiert.

Schaue dir unbedingt den Quelltext bzw. Frame-Quelltext des folgenden Fensters an. Du erkennst sofort, was hier gemacht wurde.

CODE:

```
<html>
<head>
</head>
<body>
<span id='meinTextFeld'></span>
<!-- HTML-Kommentar: Hier folgt der JavaScript-Block -->

<script>
document.getElementById('meinTextFeld').innerHTML = "Hallo Welt!";
//JS-Kommentar: Dieser Befehl nimmt 'meinTextFeld' und schreibt "Hallo
          Welt" hinein
</script>

<!-- Ende Script -->
</body>
</html>
```

`code_bsp02.htm`

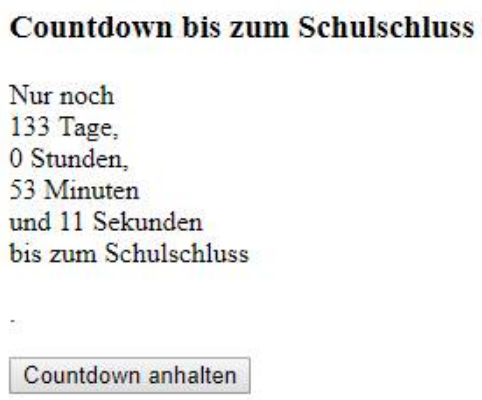
Das war nicht schwer - oder? Nun gehts weiter mit **etwas richtig Sinnvollem:** ein automatischer Counter bis zum Schulschluss.

3 | Counter bis zum Schulsschluss

Nun wollen wir eine richtig sinnvolle Anwendung programmieren!

Wie groß ist der Unterschied zwischen dem aktuellen Datum (Tage, Stunden, Minuten und Sekunden) und dem Schulschluss 12:00?

Diese Frage lassen wir uns mittels JavasScript berechnen. Du wirst vielleicht nicht alles auf Anhieb verstehen - das macht nichts. Denn in der nächsten Lektion werden wir uns wieder einigen Grundlagen widmen. Zuerst aber einmal unser kleines Programm. Schau dir den Code an, speichere ihn und verändere ihn. Vielleicht kannst du ja einen Countdown für deinen Geburtstag daraus basteln.



CODE:

```
<!-- HTML-Kommentar: Es wird nicht angezeigt. Es dient dem Programmie-
         rer zur besseren Übersicht-->
<html>
<head>
<title>SkriptKiddy-Tutorial</title>
</head>
<body>
<h3>Countdown bis zum Schulschluss</h3>
```

```
<!-- HTML-Kommentar: Wir definieren einen HTML-Bereich in den Java-
          Script das Ergebnis reinschreibt-->
<p id="countDownAnzeige"></p>
<!-- Kommentar: Ab jetzt beginnt das Javascript. -->
<!--  Beachte, dass man für Kommentare nun zwei "//" voranstellt-->

<script>
// Setze das Zieldatum ein
var zielDatum = new Date("Jun 29, 2020 12:00:00").getTime();
// Mache jede Sekunde eine neue Anzeige
var intervallID = setInterval(() => {
// Nimm das heutige Datum und Zeit
var jetzt = new Date().getTime();
// Suche die Distanz zwischen jetzt und dem Zieldatum
var zeitRaum = zielDatum - jetzt;
// Wenn der verbleibende Zeitraum grÃ¶ßer als null ist, zeige die ver-
          bleibende Zeit an.
if (zeitRaum > 0) {
// Zeitberechnungten für Tage, Stunden, Minuten und Sekunden
var tage = Math.floor(zeitRaum / (1000 * 60 * 60 * 24));
var stunden = Math.floor((zeitRaum % (1000 * 60 * 60 * 24)) / (1000 *
          60 * 60));
var minuten = Math.floor((zeitRaum % (1000 * 60 * 60)) / (1000 * 60));
var sekunden = Math.floor((zeitRaum % (1000 * 60)) / 1000);
// Zeige das Ergebnis im Element mit der id="countdownanzeige"
document.getElementById("countDownAnzeige").innerHTML = "Nur noch <br /
          >" + tage + " Tage, " + "<br />" + stunden + " Stunden, <br /
          >"
+ minuten + " Minuten <br /> und " + sekunden + " Sekunden <br />bis
          zum Schulschluss";
// Wenn der Countdown abgelaufen ist, gib eine entsprechende Meldung
          aus
} else {
clearInterval(intervallID);
document.getElementById("countDownAnzeige").innerHTML = "Ferien!";
}
}, 1000);
function anhalten() { clearInterval(intervallID); }
</script>


<!-- Kommentar: Ab jetzt wieder HTML-->

<button type="button" onclick="anhalten();">Countdown anhalten</button>
</body>

</html>
```

code_bsp03.htm

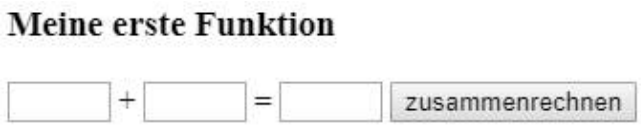
Nun gehts zurück zu einigen wichtigen Grundlagen. Eines der wichtigsten Sachen beim Programmieren sind **Variablen und Funktionen**. Weiter zum Abschnitt: Meine erste Funktion.

4 | Meine erste Funktion - wichtige Grundlagen

Variablen und Funktionen sind die wichtigsten Bausteine von Programmen. Den Ausdruck Variablen kennst du sicher aus dem Mathematikunterricht. Beim Programmieren gibt es etwas sehr ähnliches. Eine Variable kann eine Zahl sein, die bestimmte Werte annehmen kann. Etwa die Anzahl der Stunden bis zum Schulschluss.

Beim Programmieren geht es immer darum mit diesen Variablen bestimmte Operationen durchzuführen.

Funktionen sind ein Block von Anweisungen mit einem Namen. Diesen Namen kannst du natürlich selbst vergeben. Der Funktionsblock wird einmal definiert: Dann können die Javascript-Befehle der Funktion über den Namen mehrfach im Programm aufgerufen werden. Funktionen sind also notwendig für Programmaufgaben, die immer wieder benötigt werden. Im folgenden Beispiel siehst du eine ganz einfache Funktion, die 2 Zahlen zusammenzählt.



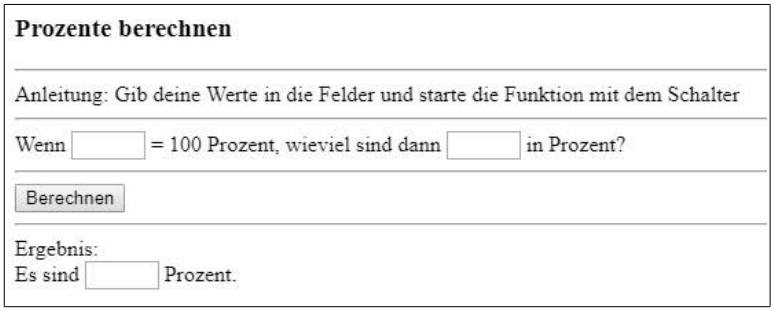
Schau dir einfach wieder den Quellcode dieses des Programmfensters an.

CODE:

```
<html>
<head>
</head>
<body>
<!-- eine Überschrift in HTML -->
<h3>Meine erste Funktion</h3>
<!-- der "Befehl" <script> zeigt an, dass nun der JavaScript-Teil kommt
        -->
<script>
/* Kommentare können auch so aussehen.
"Deklariere" eine Funktion mit dem Schlüsselwort "function", dem Namen
        (er sollte mit einem Kleinbuchstaben beginnen) und Klammern.
*/
function zusammenrechnen() {
var x = parseFloat(document.getElementById('Wert1').value);
var y = parseFloat(document.getElementById('Wert2').value);
var z = x + y;
document.getElementById('Summe').value = z;
}
</script>
<!-- der "TAG" </script> zeigt an, dass nun der JacaScript-Teil endet
        -->
<!-- mit "form" wird in HTML eine Eingabemaske gemacht -->
<form>
<input type="text" size="3" id="Wert1" /> +
<input type="text" size="3" id="Wert2" /> =
<input type="text" size="3" id="Summe" />
<!-- Wenn der Knopf gedrückt wird, rufe die Funktion "zusammenrechnen"
        auf -->
<button type="button" onclick="zusammenrechnen();">zusammenrechnen</
        button>
</form>
</body>
</html>
```

code_bsp04.htm

Ein etwas kompliziertere Anwendung siehst du im nächsten Beispiel. Dabei geht es um Prozentrechnen.



CODE:

```
<html>
<head>
</head>
<body>
<h3>Prozente berechnen</h3>
<script>
function prozent() {
 // Speichere zuerst die Eingabewerte in den Variablen a und b
var a = parseFloat(document.getElementById('Wert1').value);
var b = parseFloat(document.getElementById('Wert2').value);
//die eigentliche Rechnung
var c = (100 / a) * b;
c = (b / a) * 100;
document.getElementById('Ergebnis').value = c;
}
</script>
<hr /> Anleitung: Gib deine Werte in die Felder und starte die Funktion
         mit dem Schalter
<hr />
<form>
Wenn
<input type="text" size="3" id="Wert1" /> = 100 Prozent sind, wieviel
         sind dann
<input type="text" size="3" id="Wert2" /> in Prozent?
<br />
<hr />
<button type="button" onclick="prozent();">Berechnen</button>
<br />
```

```
<hr /> Ergebnis:
 <br /> Es sind
<input type="text" size="3" id="Ergebnis" /> Prozent.
</form>
</body>
</html>
```

code_bsp05.htm

Damit hätten wir die ersten vier Tutorials geschafft. Lies dir bitte noch die Zusammenfassung durch: Was wir jetzt bereits können sollten...

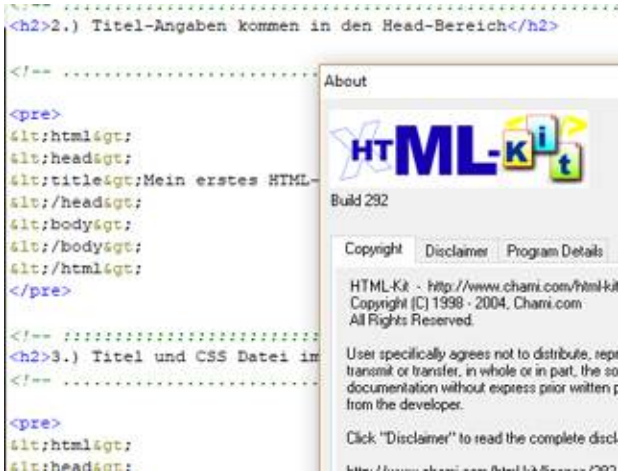
5 | Zusammenfassung der Basics

Was wir jetzt bereits können sollten...

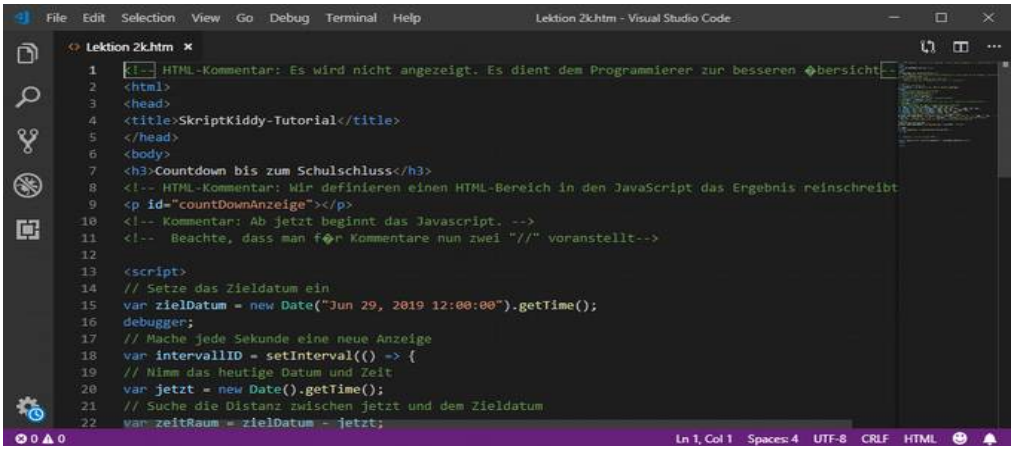
Die ersten vier Tutorials drehten sich darum, wie man ein JavaScript-Programm in einer HTML-Datei programmiert. Dies hört sich anfangs ziemlich kompliziert an. Es ist jedoch im Grunde ganz simpel. Einfache Programme zu schreiben, ist keine Hexerei. Man muss nur ein bisschen logisch denken können und eine gute Idee haben.

Wir haben erfahren, dass wir dazu eigentlich nur einen einfachen Editor (z.B. Notepad) benötigen. Besser wäre es allerdings ein spezielles Programm zu haben - also etwas zum Einfachen schreiben von Code.

Wir empfehlen hier das **Freeware HTML Kit 292**. Die Datei HK292Set-up.exe kannst du dir auf der kidsline herunterladen.



Etwas professioneller und sehr gut ist auch das Gratis Programm **Visual Studio Code.**



TIPP: Je mehr Möglichkeiten ein HTML-Editor bietet, desto unübersichtlicher ist er zumeist auch.

Das Programm Visual Studio Code ist teilweise schon recht schwierig. Die Einarbeitung lohnt sich aber.

Allerdings ist es auch schwieriger zu bedienen, weil Visual Studio Code viele zusätzliche Möglichkeiten hat. Dadurch verliert man aber oft auch Überblick.

Ob du nun einen einfachen Texteditor, ein einfaches Freewareprogramm oder teure Profisoftware verwendest: Am Vorgang des Programmierens ändert sich dabei nichts. Es wird dadurch nur etwas komfortabler.

Wichtig ist es zu wissen, wie man sich den Quellcode einer HTML-Datei ansieht. Das geht ja mit rechter Maustaste. In unserem konkreten Fall nimmt man die Option Frame-Quelltext.

Das Ansehen von fremdem Code ist eine sehr interessante Sache. Anfangs findet man natürlich nur das eine oder andere Code-Bausteinchen. Je besser man sich selber beim Programmieren auskennt, umso mehr wird einem bekannt vorkommen.

TIPP: schau dir öfter mal fremden Code an (rechte Maustaste).

In unserem Fall ist das gesamte Tuitorial auf dieser Idee aufgebaut: Codeteile aus den Scriptfenstern in den eigenen Editor zu kopieren und dort für eigene Zwecke weiterzubearbeiten. So lernt man programmieren!

Wenn du das vierte Tutorial geschafft und gut verstanden hast, dann hast du einen riesigen Schritt gemacht: du kannst bereits selber grundsätzlich ein kleines Programm schreiben. Der Rest ist jetzt eigentlich gar nicht mehr schwierig.

Funktionen sind die Bausteine von Programmen und sie lieben Variablen. Das sollte nun klar sein.

6 | Jetzt kommt Bewegung ins Spiel

In den ersten Tutorials konntest du grundlegende Kenntnisse zum JavaScript programmieren erwerben. Es wird hier vorausgesetzt, dass du daher bereits ein bisschen Ahnung von der Sache hast.

Etwa solltest du wissen, wie man eine JavaScript Programm in eine HTML-Datei hineinschreibt. Jetzt beginnen wir mit einem neuen Abschnitt unserere Reihe. **Wie kann ich eigentlich etwas bewegen!**

In diesem Tutorial zeigen wir die Grundlagen, wie jegliche Bewegung, Animation und Action in Spielen gemacht werden. Man benötigt so etwas wie einen **Taktgeber, einen Motor oder Antrieb** . Wir nennen es im Beispiel einen Loop. Er wird immer und immer wieder durchlaufen und bildet somit unseren Antrieb.

Was das Programm macht ist wenig sensationell: es zählt einfach zu einer Zahl - wir haben sie zahl genannt - immer 1 dazu.

Das Kernstück des Codes bildet die Funktion **loop**. Sie ruft sich mittels requestAnimationFrame(**loop**); am Ende wiederum selbst auf. So ensteht eine Schleife, die vom Computer meistens 60 mal /Sekunde abgearbeitet wird. Eine derartige Konstruktion bildet die Grundlage unseres JavaScript-Spieles, welches am Ende entstehen soll.

Schaue dir unbedingt den Quelltext bzw. Frame-Quelltext des folgenden Fensters an. Du erkennst sofort, was hier gemacht wurde.

CODE:

```
<html>
<head>
</head>
<body>
<h1 id="Anzeige"></h1>
<script>
var zahl = 0;
function loop() {
zahl++;
document.getElementById("Anzeige").innerText = zahl;
```

```
requestAnimationFrame(loop);
}
loop();
</script>
</body>
</html>
```

code_bsp06.htm

7 | Ein Loop für unser Spiel

Im letzten Tutorial haben wir gezeigt wie prinzipiell ein Antrieb für ein Spiel gemacht wird. In der Praxis müssen wir diese grundlegende Idee noch etwas verbessern. Ein Problem sind etwa die unterschiedlichen Geschwindigkeiten auf verschiedenen Geräten. Außerdem hört der Loop auch nicht auf zu laufen: er läuft und läuft und läuft und lässt sich nicht abschalten.

Daher müssen wir hier noch etwas verbessern. Das folgende Script (den Quellcode findest du ja über dem unteren Kasten) mag dir zuerst etwas unverständlich erscheinen. Um es zu verstehen muss man schon Zeile für Zeile durchgehen.

CODE:

```
<html>
<head>
</head>
<body>
<h3>Einfacher Loop </h3>
<h4>klein, aber oho</h4>
z:
<span id="Anzeige1"></span> - fps:
<span id="Anzeige2"></span>
<p>
```

```
<!-- HTML-Formular mit Buttons -->
<form>
<button type="button" onclick="langsamer();">Langsamer</button>
<button type="button" onclick="pause();">Ein / Aus</button>
<button type="button" onclick="zNull();">z=0</button>
<button type="button" onclick="schneller();">Schneller</button>
</form>
<!-- HTML-Forular ENDE  -->
</p>

<!-- JavaScript -->
<script>
var requestID; //um den "loop" auch abzuschalten
var angehalten = false; //Hilfsvariable um Pause zu steuern
var zaehler = 0;
var fps = 18;
var jetzt;
var dann = Date.now();
var intervall = 1000 / fps;
var delta;

function loop() {
jetzt = Date.now();
delta = jetzt - dann;
if (delta > intervall) {
dann = jetzt - (delta % intervall);
if (!angehalten
) {
zaehler++;
anzeigen();
}
}
requestID = requestAnimationFrame(loop);
}
// Rufe den Loop zum ersten mal auf
loop();

function anzeigen() {
document.getElementById("Anzeige1").innerText = zaehler;
document.getElementById("Anzeige2").innerText = fps;
}

//Diese Funktionen werden von den Buttons aufgerufen
function schneller() {
fps++;
intervall = 1000 / fps;
}
function langsamer() {
fps--;
intervall = 1000 / fps;
}
```

```
function pause() {
angehalten = !angehalten;
if (!angehalten) {
loop();
}
}
function zNull() {
zaehler = 0;
}
</script>
</body>
</html>
```

code_bsp07.htm

8 | Bewegte Grafiken

Jetzt wirds spannend: wir wissen bereits, wie wir einen Loop als Taktgeber benutzen können. Nun werden wir zeigen, wie man eine Fläche erzeugt, auf der eine einfache Grafik dargestellt und bewegt wird. Diese Fläche bezeichnen wir als canvas. **Canvas bedeutet "Leinwand"**. Auf dieser Leinwand bauen wir später unser Spiel.

Wenn du dir den Code unten ansiehst, erkennst du zuerst den CSS-Teil (<style>). Hier wird dem Browser mitgeteilt, wie unser **canvas** aussehen wird.

Im BODY-Teil wird der canvas dann eingebaut.

im SCRIPT-Teil gibt es die **function anzeigen()**. hier wird der Inhalt des Canvas gezeichnet.

CODE:

```
<html>
<head>
<!-- ~~~~~~~~~~~Canvas Style ANFANG~~~~~~~~~~~~~~~~~~ -->
<style>
#canvas {
border: 0px solid black;
margin-top: 4px;
background: #4F4D4E;
box-shadow: 0px 0px 0px 0px #003333;
border-radius: 20px;
```

```
}
</style>
<!-- ~~~~~~~~~~~~~~~~Canvas Style ENDE~~~~~~~~~~~~~~~~~~~ -->
</head>
<body>
<canvas id="canvas" width="600" height="400"></canvas>
<!--~~~~~~~~~~~~~~~~~~JavaScript~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<script>
// Hole einen Zugang zum Canvas
var canvas = document.getElementById('canvas');
var context = canvas.getContext('2d');
var requestID; //um den "loop" auch abzuschalten
var Angehalten = false; //Hilfsvariable um Pause zu steuern
var zaehler = 0;
var fps = 18;
var jetzt;
var dann = Date.now();
var intervall = 1000 / fps;
var delta;
function loop() {
jetzt = Date.now();
delta = jetzt - dann;
if (delta > intervall) {
dann = jetzt  - (delta % intervall);
if (!Angehalten) {
zaehler++;
anzeigen();
}
}
requestID = requestAnimationFrame(loop);
}
loop();
// aktualisiere die Anzeige
function anzeigen() {
context.clearRect(0, 0, canvas.width, canvas.height);
context.fillStyle = '#FF4500';
context.font = "100px Verdana";
context.fillText("z= " + zaehler, 50, 200);
context.font = "20px Verdana";
context.fillText("fps= " + fps, 150, 300);
//Viereck zeichnen
context.fillRect(zaehler, 320, 20, 20);
}
//Diese Funktionen werden von den Buttons aufgerufen
function schneller() {
fps++;
intervall = 1000 / fps;
}
function langsamer() {
fps--;
intervall = 1000 / fps;
```

```
}
function pause() {
Angehalten = !Angehalten;
if (!Angehalten) { loop(); }
}
function zNull() {
zaehler = 0;
}
</script>
<p>
<!-- ~~~~~~~~HTML-Formular mit Buttons~~~~~~~~~~~~~~~~~~~~~~~ -->
<form>
<button type="button" onclick="langsamer();">Langsamer</button>
<button type="button" onclick="pause();">Ein / Aus</button>
<button type="button" onclick="zNull();">z=0</button>
<button type="button" onclick="schneller();">Schneller</button>
</form>
<!-- ~~~~~~~~HTML-Forular ENDE    ~~~~~~~~~~~~~~~~~~~~ -->
</p>
</body>
</html>
```

code_bsp10.htm

9 | Ein springender Ball

Im letzten Tutorial haben wir einen wichtigen JavaSript-Baustein namens CANVAS kennengelernt. Ein Canvas (dt.: Leinwand") ist eine Fläche zum Darstellen unseres Spiels. Dort hinein werden die Bilder und Grafiken des späteren Spiels geladen.

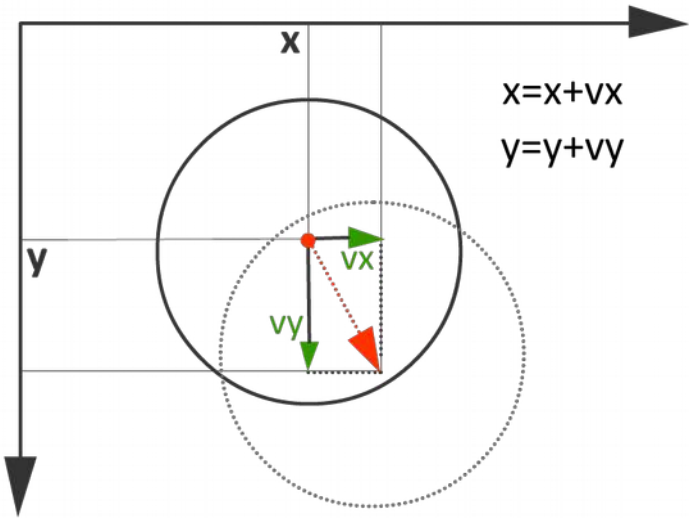
Außerdem haben wir gelernt, dass wir so etwas wie einen **Motor, Antrieb oder Taktgeber** für unser Spiel brauchen. Ohne diesen Antrieb entsteht keine Bewegung am Bildschirm. Die entsprechende Funktion haben wir **gameLoop** genannt. Leider ist sie relativ kompliziert, sie ist aber unerlässlich für ein Spiel.

```
function gameLoop() {
jetzt = Date.now();
delta = jetzt - dann;
if (delta > intervall) {
dann = jetzt - (delta % intervall);
if (!angehalten) {
context.clearRect(0, 0, canvas.width, canvas.height);
macheAnzeige();
berechneUndZeichneBall();
}
}
requestID = requestAnimationFrame(gameLoop);
}
```

Wenn ihr euch die Funktion anseht, findet ihr Variablen wie "jetzt" und "dann" und "intervall". Offensichtlich wird hier eine Zeit gemessen und nach einer bestimmten Zeit die Funktion erneut aufgerufen. Dieser Code bildet tatsächlich das wichtigste **Herzstück eines Spiels - leider ist er etwas schwierig**.

Sofern unser Spiel nicht angehalten wurde (if (!angehalten) wird hier 30 mal Sekunde der Ball neu berechnet und die Anzeige neu gemacht. Wenn ihr mittels derSchalter die Frames per Seconds ändert, merkt ihr, wie das Spiel immer flüssiger läuft, oder eben langsam und abgehackt.

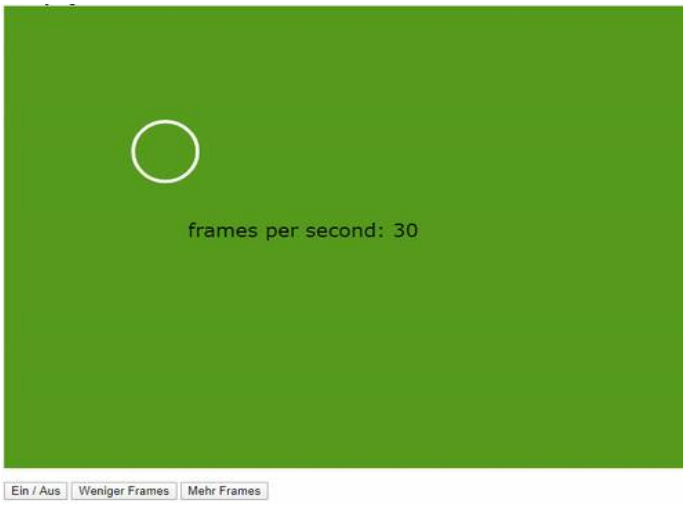
Wie ihr sehen könnt, ergibt das unser einfaches Programm mittlerweile schon ganz schön viel Code. Dadurch sollte man sich aber nicht abschrecken lassen. Denn hat man erst einmal diese Grundstruktur zusammen, ist der Rest nun wieder einfacher. **Und vor allem viel spannender, weil nun Action ins Spiel kommt.**



Unser Beispiel zeigt, wie ein Ball hüpft und an den Ecken abprallt. Wie man sehen kann, werden die **Richtung** und die **Geschwindigkeit** des Balls durch **Mathematik** festgelegt. Die Variablen x und x sind dabei die aktuellen **Koordinaten** des Balls. Die Variablen vx und vy sind die jeweiligen Richtungsvektoren der Bewegung. Im gameLoop wird nun einfach ständig dem x sein vx und dem y ein vy dazugezählt. Dadurch entsteht die Bewegung des Balls. Wie groß das vx und das vy jeweils sind wird vom Programm dabei ständig neu berechnet.

Das v in vx und vy steht dabei übrigens für "Velocity" (dt.: Geschwindigkeit).

Ihr solltet euch dieses Beispiel herunterkopieren und damit herumprobieren. Erhöht z.B. einmal den Wert der **Schwerkraft** oder **Elastizität** und seht, wie sich die physikalischen Verhältnisse verändern.



CODE:

```
<html>

<head>
    <!-- Canvas Style (hier oder im CSS) -->
    <style>
        #Spielflaeche {
            background: rgb(86, 155, 29);
        }
    </style>
</head>

<body>
    9 Ein springender Ball<br />
    <canvas id="Spielflaeche" width="800" height="540"></canvas>
    <script>
        var canvas = document.getElementById('Spielflaeche');
        var context = canvas.getContext('2d');
        var requestID; //um den "gameLoop" auch abzuschalten
        var angehalten = false; //Hilfsvariable um den Status "idle" zu
          erkennen
        var fps = 30;
        var jetzt;
        var dann = Date.now();
        var intervall = 1000 / fps;
        var delta;
        const schwerkraft = 0.2, elastizitaet = 0.95, energieverlust =
          0.9, r = 35;
```

```
var x, y, vx, vy;
x = canvas.width / 2;
y = -r;
vx = 3;
vy = 0;
context.font = "22px Verdana";
context.fillStyle = '#000000';

function gameLoop() {
    jetzt = Date.now();
    delta = jetzt - dann;
    if (delta > intervall) {
        dann = jetzt - (delta % intervall);
        if (!angehalten) {
            context.clearRect(0, 0, canvas.width, can-
 vas.height);
            macheAnzeige();
            berechneUndZeichneBall();
        }
    }
    requestID = requestAnimationFrame(gameLoop);
}

gameLoop();

function macheAnzeige() {
    context.fillText("frames per second: " + fps,
 canvas.width / 4, canvas.height / 2);
}
/* ball */
function berechneUndZeichneBall() {
    if (x + r >= canvas.width) {
        vx = -vx * elastizitaet;
        x = canvas.width - r;
    } else if (x - r <= 0) {
        vx = -vx * elastizitaet;
        x = r;
    }
    if (y + r >= canvas.height) {
        vy = -vy * elastizitaet;
        y = canvas.height - r;
        vx *= energieverlust;

    } else if (y - r <= 0) {
    }

    vy += schwerkraft;
    x += vx;
    y += vy;

    context.beginPath();
```

```
            context.strokeStyle = "#ffffff";
            context.lineWidth = 4;
            context.arc(x, y, r, 0, 2 * Math.PI, false);
            context.stroke();
        }
        //Diese Funktionen werden von den Buttons aufgerufen
        function schneller() {
            fps++;
            intervall = 1000 / fps;
        }
        function langsamer() {
            fps--;
            intervall = 1000 / fps;
        }
        function pause() {
            angehalten = !angehalten;
            if (!angehalten) { gameLoop(); }
        }
    </script>
    <p>
        <!-- HTML-Formular mit Buttons -->
        <form>
            <button type="button" onclick="pause();">Ein / Aus</button>
            <button type="button" onclick="langsamer();">Weniger
          Frames</button>
            <button type="button" onclick="schneller();">Mehr Frames</
          button>
        </form>
        <!--HTML-Forular ENDE  -->
    </p>
</body>

</html>
```

code_bsp11.htm

10 | Interaktion mit der Maus

In diesem Tutorial zeigen wir, wie man nun dem Canvas eine Maus-Aktion hinzufügt. Dadurch kann eine **Interaktion mit der Maus** stattfinden.

CODE:

```
<html>
<head>
    <style>
        #Spielflaeche {
            background: rgb(86, 155, 29);
        }
    </style>
</head>

<body>
    <!-- ~~~~~~~~~~Die HTML-Datei bekommt ein CANVAS~~~~~~~~~~~~~~~ -->
    <canvas id="Spielflaeche" width="800" height="540"></canvas>
    <!-- ~~~~~~~Jetzt gehts los mit dem javaScript
         Teil:~~~~~~~~~~~~~~~~~ -->
    <script>
        /* ~~~~~~~~~~~~~~~~~HAUPT TEIL~~~~~~~~~~~~~~~~~~~~~ */
        var canvas = document.getElementById('Spielflaeche');
        var context = canvas.getContext('2d');
        //Dem canvas wird ein "EVENT-LISTENER" für Mausabfragen zugeor-
          det. Damit kann eine Maussteuerung gebaut werden
        canvas.addEventListener('mousedown', mausDruck);
        canvas.addEventListener('mouseup', mausLos);
        //Die Variablen unseres Programms werden definiert:
        var schwerkraft = 0.1, elastizitaet = 0.95, energieverlust =
          0.9, r = 35;
        var x, y, vx, vy;
        x = canvas.width / 2;
```

```
        y = -r;
        vx = 3;
        vy = 0;
        //////////////////
        var requestID;
        //WICHTIG: um den "gameLoop" auch abzuschalten muss eine re-
          questID vorhanden sein.
        var angehalten = false;
        //Hilfsvariable um den Status "idle" zu erkennen
        var fps = 40;
        var now;
        var then = Date.now();
        var interval = 1000 / fps;
        var delta;

        ////////////////////////Die Mausfunktionen
        function mausDruck() { angehalten = true; }

        function mausLos(eventDaten) {
            console.log(eventDaten);
            x = eventDaten.clientX;// - document.getElementById('can-
          vas').offsetLeft;
            y = eventDaten.clientY;// - document.getElementById('can-
          vas').offsetTop;
            angehalten = false;
            vx = 0;
            vy = 0;
        }
        ////////////////////////////////////
        function gameLoop() {
            now = Date.now();
            delta = now - then;
            if (delta > interval) {
                then = now - (delta % interval);
                if (!angehalten) {
                    macheAnzeigen();
                    berechne_und_zeichneBALL();
                }
            }
            requestID = requestAnimationFrame(gameLoop);
        }
        ////////////////////////////////////////////
        gameLoop();
        ////////////////////////////////////////////


        function macheAnzeigen() {
            context.clearRect(0, 0, canvas.width, canvas.height);
            context.font = "24px Verdana";
            context.fillStyle = '#000000';
            context.fillText("fps: " + fps, 280, 350);
```

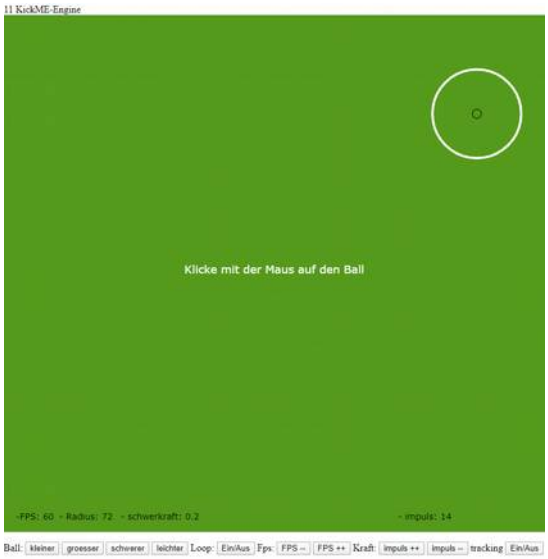
```
        context.textAlign = "center";
        context.font = "36px Verdana";
        context.strokeStyle = "#fcfcfc";
        context.lineWidth = 2;
        context.strokeText("Klicke HIER mit der Maus", canvas.width
      / 2, canvas.height / 2);
    }

    /*~~~~~~~~~~~~~~~~~~~~~~~~~~ball~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    function berechne_und_zeichneBALL() {
        if (x + r >= canvas.width) {
            vx = -vx * elastizitaet;
            x = canvas.width - r;
        }
        else if (x - r <= 0) {
            vx = -vx * elastizitaet;
            x = r;
        }
        if (y + r >= canvas.height) {
            vy = -vy * elastizitaet;
            y = canvas.height - r;
            vx *= energieverlust;
        } else if (y - r <= 0) {
        }
        vy += schwerkraft;
        x += vx;
        y += vy;
        context.beginPath();
        context.strokeStyle = "#ffffff";
        context.arc(x, y, r, 0, 2 * Math.PI, false);
        context.stroke();
    }
    //Diese Funktionen werden von den Buttons aufgerufen
    function schneller() {
        fps++;
        interval = 1000 / fps;
    }
    function langsamer() {
        fps--;
        interval = 1000 / fps;
    }
    function pause() {
        angehalten = !angehalten;
        //So toggelt man einen Boolean
        if (!angehalten) { gameLoop(); }
    }
    function yNull() {vy = 0; y = 0;}
    function schwerer() {
        schwerkraft += 0.1;
        elastizitaet *= 0.9
    }
```

```
        function leichter() {
            schwerkraft -= 0.1;
            elastizitaet = 0.95;
        }

        function kleiner() {
            r -= 3;
        }

        function groesser() {
            r += 3;
        }
    </script>
    <!-- ~~~~~~~~~~~~~~~~~~JavaScript-ENDE~~~~~~~~~~~~~~~~~~~~~~~~~~
         -->
    <p>
        <!-- ~~~~~~~~~HTML-Formular mit Buttons~~~~~~~~~~~~~~~~~~~ -->
        <form>
            Canvas:
            <button type="button" onclick="langsamer();">Fps -- </but-
          ton>
            <button type="button" onclick="pause();">Ein/Aus</button>
            <button type="button" onclick="yNull();">y=0</button>
            <button type="button" onclick="schneller();">Fps ++ </but-
          ton>
            <!-- ~~~~~~~~~~~~HTML-Forular ENDE
          ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
            Physik:
            <!-- ~~~~~~~~~~~~~~~HTML-Formular mit
          Buttons~~~~~~~~~~~~~~~~~~ -->
            <button type="button" onclick="schwerer();">schwerer </but-
          ton>
            <button type="button" onclick="leichter();">leichter </but-
          ton>
            <button type="button" onclick="kleiner();">kleiner</button>
            <button type="button" onclick="groesser();">grosser </but-
          ton>
        </form>
        <!-- ~~~~~HTML-Forular ENDE ~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    </p>
</body>
</html>
```

code_bsp12.htm

11 | KickME-Engine

In diesem Tutorial gehts bereits ganz schön zur Sache. Zwar sieht das Ganze relativ unspektakulär aus, der Code beinhaltet aber bereits die wesentlichen Mausaktionen (Drücken, Loslassen und Bewegen der Maus) , die später im Spiel benötigt werden.



Im JavaScript Teil unseres Beispiels werden zuerst dem Canvas-Baustein die Maus-Ereignisse zugordnet. Später werden dann die nötigen physikalischen Berechnungen durchgeführt.

CODE:

```
<html>
<head>
    <!-- ~~~~~Canvas Style (hier oder im CSS)~~~ -->
    <style>
        #Spielflaeche {
            background: rgb(86, 155, 29);
        }
    </style>
</head>

<body>
    11 KickME-Engine<br />
    <!-- ~~~~~~Die HTML-Datei bekommt ein CANVAS~~~~~ -->
    <canvas id="Spielflaeche" width="880" height="840"></canvas>
```

```
<!-- ~~~~~~Jetzt gehts los mit dem javaScript Teil:~~~~~ -->
<script>
    /* ~~~~~HAUPT TEIL~~~ */
    var canvas = document.getElementById('Spielflaeche');
    var context = canvas.getContext('2d');
    //Dem canvas wird ein "EVENT-LISTENER" für Mausabfragen zugeor-
     det. Damit kann eine Maussteuerung gebaut werden
    canvas.addEventListener('mousedown', mausDruck);
    //Die Variablen unseres Programms werden definiert:
    var schwerkraft = 0.2, elastizitaet = 0.93, energieverlust =
     0.9, r = 72, impuls = 14;
    var x, y, vx, vy;
    x = canvas.width / 2;
    y = -r;
    vx = 3;
    vy = 0;
    ////////////////////
    var requestID;  //WICHTIG: um den "gameLoop" auch abzuschalten
     muss eine requestID vorhanden sein.
    var angehalten = false;
    var fps = 60;
    var jetzt;
    var dann = Date.now();
    var intervall = 1000 / fps;
    var delta;
    ////////////////zusätzliche Variabel um "Tracking" einzuschal-
     ten
    var track = false;
    ////////////////////
    ///////////Die Mausfunktionen:
    function mausDruck(eventDaten) {
        // die Variablen dx und dy geben den jeweiligen Abstand vom
     Mauszeiger zum Mittelpunkt an
        var dx, dy;
        dx = x - eventDaten.offsetX;
        dy = y - eventDaten.offsetY;
        //wenn der Ball angeklickt wurde
        if (Math.sqrt((dx * dx) + (dy * dy)) < r) {
            vx += impuls * dx / r
            vy += impuls * dy / r
        }
    }
    ////////////////////////////
    function gameLoop() {
        jetzt = Date.now();
        delta = jetzt - dann;
        if (delta > intervall) {
            dann = jetzt - (delta % intervall);
            if (!angehalten) {
                checkWand();
                berechneFlugbahn();
```

```
                if (!track) { loescheAlteGrafik(); }
                zeichneBall();
                zeichneBallMittelpunkt();
                macheTexte();
            }
        }
        requestID = requestAnimationFrame(gameLoop);
    }
    ////////////////////////////////////////////
    gameLoop();
    ///////////////////////
    function loescheAlteGrafik() {
        context.clearRect(0, 0, canvas.width, canvas.height);
    }
    /////////////
    function checkWand() {
        if (x + r >= canvas.width) {
            vx = -vx * elastizitaet;
            x = canvas.width - r;
        }
        else if (x - r <= 0) {
            vx = -vx * elastizitaet;
            x = r;
        }
        if (y + r >= canvas.height) {
            vy = -vy * elastizitaet;
            y = canvas.height - r;
            vx *= energieverlust;
        }
    }
    //////////
    function macheTexte() {
        context.fillStyle = '#ffffff';
        context.font = "18px Verdana";
        context.fillText("Klicke mit der Maus auf den Ball", can-
     vas.width/3, canvas.height/2);
        context.font = "14px Verdana";
        context.fillStyle = '#000000';
        context.fillText("-FPS: " + fps, 20, canvas.height - 20);
        context.fillText("- Radius: " + r, 90, canvas.height - 20);
        context.fillText("- schwerkraft: " + schwerkraft, 190, can-
     vas.height - 20);
        context.fillText("- impuls: " + impuls, 640, canvas.height
     - 20);
    }
    /* ~~~~~~~ball~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    function berechneFlugbahn() {
        vy += schwerkraft;
        x += vx;
        y += vy;
    }
```

```
            /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    function zeichneBall() {
        context.beginPath();
        context.strokeStyle = "#ffffff";
        context.lineWidth = 4;
        context.arc(x, y, r, 0, 2 * Math.PI, false);
        context.stroke();
    }
     /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    function zeichneBallMittelpunkt() {
        context.beginPath();
        context.strokeStyle = "#000000";
        context.lineWidth = 1;
        context.arc(x, y, r / 10, 0, 2 * Math.PI, false);
        context.stroke();
    }
     /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    //Diese Funktionen werden von den Buttons aufgerufen
    function schneller() { fps++; intervall = 1000 / fps; }
    function langsamer() { fps--; intervall = 1000 / fps; }
    function tracking() { track = !track; }
    function pause() { angehalten = !angehalten; if (!angehalten) {
     gameLoop(); } }
    function yNull() { vy = 0; y = 0; }
    function schwerer() { schwerkraft += 0.1; elastizitaet *=
     0.9; }
    function leichter() { schwerkraft -= 0.1; elastizitaet =
     0.95; }
    function kleiner() { r -= 3; }
    function groesser() { r += 3; }
    function stark() { impuls += 2; }
    function schwach() { impuls -= 2; }
</script>
<!-- ~~~JavaScript-ENDE~~: ab jetzt gehts wieder mit HTML weiter --
    >
<p>
    <!-- ~~~HTML-Formular mit Buttons~~ -->
    <form>
        Ball:
        <button type="button" onclick="kleiner();">kleiner</button>
        <button type="button" onclick="groesser();">groesser </but-
     ton>
        <button type="button" onclick="schwerer();">schwerer </but-
     ton>
        <button type="button" onclick="leichter();">leichter </but-
     ton>
        Loop:
        <button type="button" onclick="pause();">Ein/Aus</button>
        Fps:
        <button type="button" onclick="langsamer();">FPS -- </but-
     ton>
```

```
            <button type="button" onclick="schneller();">FPS ++ </but-
          ton>
            Kraft:
            <button type="button" onclick="stark();">impuls ++ </but-
          ton>
            <button type="button" onclick="schwach();">impuls -- </but-
          ton>
            tracking
            <button type="button" onclick="tracking();">Ein/Aus</but-
          ton>
        </form>
        <!-- ~~~HTML-Forular ENDE       ~~ -->
    </p>

</body>

</html>
```

`code_bsp13.htm`

Damit haben wir eine Datei mit der sich erstklassig weiterarbeiten lässt, um daraus ein kleines Spiel zu erzeugen.

Im Internet gibt es natürlich eine Fülle von ausgezeichneten Programmieranleitungen. Du brauchst nur etwas googeln nach "JavaScript programmieren". Wir empfehlen hier z.B.: Self HTML.

12 | Szenenwechsel

Navigation im Spiel: Ein Spiel besteht immer aus mehreren Szenen.

Nachdem wir wissen, wie ein gameLoop gebaut wird benötigen wir noch den passenden Rahmen - also Spielanfang und Spielende

Nachdem wir uns genau mit der Programmierung des Spielmotors (Game-Loop) befasst haben, müssen wir uns einmal dem größeren Rahmen des Spiels zuwenden. Ein Spiel besteht ja nicht nur aus dem eigentlichen Spiel, sondern im einfachsten Fall aus einem **Begrüßungsbild** (z.B. **Spielstart**) dem eigentlichen Spiel (m it dem **GameLoop**) und dem Spielende (**GameO-ver**).

Was wir also konkret benötigen ist ein Programmteil, der zwischen Szenen innerhalb des Spiels herumschalten kann. Diesen wollen wir nun entwickeln.

Um den Code kurz und übersichtlich zu halten, besteht unsere eigentliche Spielaktion lediglich aus einer Zahl, die hinuntergezählt wird. Der nachfolgende Code demonstriert das: die Funktion **countDown** zählt jedesmal wenn sie aufgerufen wird von einer bestimmten Spielzeit immer 1 ab [spielzeit = spielzeit -1;]. Wenn die Spielzeit dann aus ist [if (spielzeit < 0)] wird eine Funktion **gameEnde** aufgerufen. Dort wird dann der GameLoop abgeschalten.

CodeAuszug:

```
function countDown() {
            spielzeit = spielzeit -1;
            if (spielzeit < 0) {
                gameEnde();
            }
        }
```

Anfang:	GameLoop aus
Spiel:	**GameLoop ein**
Ende:	GameLoop aus

Nun wird klar warum wir vorher viel Aufwand mit dem Ein- und Abschalten des GameLoops betrieben haben.

Um zwischen den Szenen Anfang, Spiel und Ende zu wechseln definieren wir eine Variable "**szene**"; Deren Wert kann hier 1,2 oder 3 sein. Damit können wir nun steuern was passiert ,wenn ich eine Aktion setze, also wenn ich z.B. die Maus loslasse. Befinde ich mich in szene =1, wird beim "Maus-Event" der gameLoop gestartet, die Variable "angehalten" abgeschaltet und die szene auf 2 gesetzt. Damit weiß nun auch das Programm selbst, dass es sich im SpielModus befindet.

```
function mausLos(eventDaten) {
            if (szene == 1) { szene = 2; angehalten = false;
          gameLoop(); }
            if (szene == 2) { }
            if (szene == 3) { szene = 1; gameStart(); }
        }
```

CODE:

```
<html>

<head>
    <!-- ~~~~~Canvas Style (hier oder im CSS)~~~ -->
    <style>
        #Spielflaeche {
            background: rgb(86, 155, 29);
        }
    </style>
</head>

<body>
    12 Szenenwechsel<br />
```

```
<!-- ~~~~~~Die HTML-Datei bekommt ein CANVAS~~~~~ -->
<canvas id="Spielflaeche" width="830" height="560"></canvas>
<!-- ~~~~~~Jetzt gehts los mit dem javaScript Teil:~~~~~ -->
<script>
    var canvas = document.getElementById('Spielflaeche');
    var context = canvas.getContext('2d');
    //Dem canvas wird ein "EVENT-LISTENER" für Mausabfragen zugeor-
      det. Damit kann eine Maussteuerung gebaut werden
    canvas.addEventListener('mouseup', mausLos);
    var szene;
    var spielzeit;
    ///////
    var requestID;
    //WICHTIG: um den "gameLoop" auch abzuschalten muss eine re-
      questID vorhanden sein.
    var angehalten = false;
    angehalten = true;
    //Hilfsvariable um den Status "idle" zu erkennen
    var fps = 30;
    var jetzt;
    var dann = Date.now();
    var intervall = 1000 / fps;
    var delta;

    //.............................................................
      ..............................//
    function mausLos(eventDaten) {
        if (szene == 1) { szene = 2; angehalten = false;
     gameLoop(); }
        if (szene == 2) { }
        if (szene == 3) { szene = 1; gameStart(); }
    }
    //.............................................................
      ..............................//
    function gameStart() {
        szene = 1;
        spielzeit = 60;
        console.log("Start");
        loescheAlteGrafik();
        context.fillStyle = "#00aff0";
        context.fillRect(0, 0, canvas.width, canvas.height);
        angehalten = true;
        if (requestID) {
            cancelAnimationFrame(requestID);
            requestID = undefined;
        }
        context.textAlign = "center";
        context.font = "100px Verdana";
        context.fillStyle = "#292929";
        context.fillText("Anfang", canvas.width / 2,
     canvas.height / 2);
```

```
        context.strokeStyle = "#fcfcfc";
        context.lineWidth = 2;
        context.strokeText("Anfang", canvas.width / 2, can-
    vas.height / 2);
        context.font = "20px Verdana";
        context.fillText("Szene#" + szene, canvas.width / 2, can-
    vas.height - 200);
    }
    //..........................................................
      ..............................//
    function gameEnde() {
        console.log("Ende");
        szene = 3;
        loescheAlteGrafik();
        context.fillStyle = "#e52d27";
        context.fillRect(0, 0, canvas.width, canvas.height);
        angehalten = true;

        if (requestID) {
            cancelAnimationFrame(requestID);
            requestID = undefined;
        }
        context.textAlign = "center";
        context.font = "100px Verdana";
        context.fillStyle = "#292929";
        context.fillText("SpielEnde", canvas.width / 2, can-
    vas.height / 2);
        context.strokeStyle = "#fcfcfc";
        context.lineWidth = 2;
        context.strokeText("SpielEnde", canvas.width / 2, can-
    vas.height / 2);
        context.font = "20px Verdana";
        context.fillText("Szene#" + szene, canvas.width / 2, can-
    vas.height - 200);
    }
    //..........................................................
      ..............................//
    function countDown() {
        spielzeit -= 1;
        if (spielzeit < 0) {
            gameEnde();
        }
    }
    //..........................................................
      ..............................//
    function gameLoop() {
        countDown();
        jetzt = Date.now();
        delta = jetzt - dann;
        if (delta > intervall) {
            dann = jetzt - (delta % intervall);
```

```
                if (!angehalten) {
                    zeichneSpiel();
                }
            }
            requestID = requestAnimationFrame(gameLoop);
        }
        //.....................................................
          ..............................//
        function loescheAlteGrafik() {
            context.clearRect(0, 0, canvas.width, canvas.height);
        }
        //.....................................................
          ..............................//
        function zeichneSpiel() {
            loescheAlteGrafik();
            context.fillStyle = "#82b541";
            context.fillRect(0, 0, canvas.width, canvas.height);
            context.textAlign = "center";
            context.font = "100px Verdana";
            context.fillStyle = "#292929";
            context.fillText(spielzeit, canvas.width / 2, canvas.height
          / 2);
            context.strokeStyle = "#fcfcfc";
            context.lineWidth = 2;
            context.strokeText(spielzeit, canvas.width / 2, can-
          vas.height / 2);
            context.font = "20px Verdana";
            context.fillText("Szene#" + szene, canvas.width / 2, can-
          vas.height - 200);
        }
        //.....................................................
          ..............................//
        gameStart();
    </script>
</body>

</html>
```

code_bsp14.htm

13 | Canvas als Zeichenfläche


Unsere Leinwand: das Canvas .

Mit dem Canvas als Leinwand haben wir die Möglichkeit JavaScript auch für grafische Anwendungen zu nutzen. Dass das Einbauen eines solchen Canvas nicht ganz einfach ist, habt ihr sicher schon mitbekommen. Allerdings findet ihr den nötigen Code hier in diesem Tutorial. Im folgenden zeigen wir eine kleine Anwendung in der ein Canvas definiert wird und dann eine Funtion aufgrufen wird wo alles Mögliche auf das Canvas gezeichnet wird.

13.1 | Einfache Grafiken und Texte

Das direkte Zeichen auf dem Canvas - wie es hier gezeigt wird - ist eine von 2 Möglichkeiten mit Bildern zu arbeiten. Ein zweite wäre es Bilder ins Canvas hineinzuladen. Dies zeigen wir aber erst später. Zuerst einmal eine Demonstration verschiedener Zeichenmöglichkeiten auf dem Canvas.

```
<html>
<head>
</head>
<body>
    <canvas id="canvas" width="600" height="400"></canvas>
    <script>
        var canvas = document.getElementById('canvas');
        var context = canvas.getContext('2d');

        function aufCanvasZeichnen() {
            //Hintergrundfläche:
            context.fillStyle = '#ff3300';
            context.fillRect(0, 0, canvas.width, canvas.height);
            //Rahmen:
            context.strokeStyle = "#CCCCCC";
            context.lineWidth = 10;
            context.strokeRect(20, 20, canvas.width - 40, canvas.height
        - 40);
            //Halbkreis:
            context.beginPath();
            context.strokeStyle = "#FFFFFF";
            context.lineWidth = 6;
            context.arc(canvas.width / 2, canvas.height / 2 + 40, 100,
        0, 1 * Math.PI);
            context.stroke();
            //Schriftzug:
```

```
            context.font = "90px Verdana";
            context.fillStyle = '#000000';
            context.fillText("Hallo Welt!", 50, 200);
            //Linie:
            context.beginPath();
            context.moveTo(0, canvas.height / 2);
            context.lineTo(canvas.width, canvas.height / 2);
            context.strokeStyle = "#FFFFFF";
            context.lineWidth = 2;
            context.stroke();
        }
        aufCanvasZeichnen();
    </script>
</body>
</html>
```

code_bsp08.htm

Sicherlich hast du bemerkt, wie im Code mit den Farben umgegangen wird. Sie werden durch die HEX-Farbcodes festgelegt (zB.: #FF3300, #FFFFFF, . . .)

```
 vgl.:
context.fillStyle = '#ff3300';
```

13.2 | Bilder und Sprites im Canvas

Das Zeichnen, wie es gerade gezeigt wurde, hat viele Vorteile:

- Damit lassen sich Texte ins Canvas schreiben
- Man benötigt keine Extrabilder, so lässt sich alles schnell ausprobieren
- Das Programm lädt schnell, es muss keine datenintensiven Bilder laden
- Man kann damit vieles bereits gut umsetzen, etwa Diagramme oder ähnliches

Die Nachteile liegen natürlich auch auf der Hand: s**ie sind zu einfach um schöne Grafiken darstellen zu können**.

Wie bekommt man also nun ein besseres Bild oder Foto ins Canvas?

Folgende drei Code-Zeilen sind prinzipiell nötig: Eine Variable namens foto definieren, dieser den Dateipfad und die Datei zuweisen und schließlich in das Canvas hineinladen.

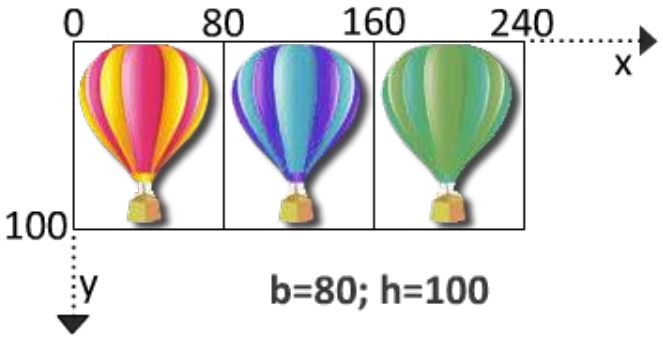
```
var foto = new Image();          //eine Bild-Variable namens Ffto
foto.src = "himmel.jpg";           //src gibt Quelle an
context.drawImage(foto, 0, 0); //Foto auf Koordinanten 0,0 geben
```

Das spezielle am folgenden Script ist die Art, wie das Ballonbild behandelt wird. Wir nutzen dabei die Möglichkeit ein Bild nur teilweise darzustellen.

Mit diesem Trick, den Spielentwickler gerne benutzen, kann so aus einer Bildsequenz ein einzelnes Teilbild gezielt angesteuert werden. Bei unserem Ballon-Bild wollen wir daher nur einen Ballon auswählen. Das machen wir mit folgendem Code:

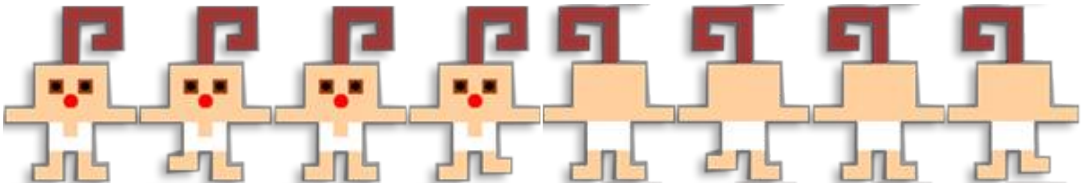
```
context.drawImage(ballon, 0, 0, 80, 100, 400, 120, 80, 100);
```

Wir teilen dem Context im Canvas mit, die Bild-Variable "ballon" darzustellen. Und zwar von der Bilderkoordinate 0, 0 um 80 nach rechts und 100 nach unten. Dadurch **entsteht ein Bildausschnitt**, der nun im zweiten Teil des Befehls seine eigentliche **Position im Canvas** (400,120) bekommt und mit 80 breit und 100 hoch dargestellt wird.



Somit kennen wir nun die wesentlichen Grafikmöglichkeiten eines Canvas:

- Die programmeigenen Grafiken um effizient Vierecke, Kreise, Striche und Text darzustellen.
- Die Möglichkeit ein Foto an eine Variable zu hängen
- Bilderstreifen als Sprites für Bewegungsabläufe einer Spielfigur zu nutzen.



Zur besseren Veranschaulichung sollte das folgende Beispiel betrachtet werden. Man lädt zwei externe Bilder himmel.jpg und ballone.png. Wenn das Ballonbild geladen ist [ballon.onload = function ()] wird die Funktion "aufCanvasZeichnen" abgearbeitet. Die Reihenfolge der Anweisungen ist dabei natürlich entscheidend, da die Dinge schichtweise aufeinandergelegt werden.

Im Code sind die grundlegenden Ansätze vereint. Außerdem gibt es noch den kleinen Gag mit der Zufallszahl der beim Programmstart frech eine 0,1 oder 2 zieht und somit den jeweiligen Ballon auswählt.

```
<html>
```

```
   <head>
</head>

<body>
    <canvas id="canvas" width="600" height="400"></canvas>
    <script>
        var canvas = document.getElementById('canvas');
        var context = canvas.getContext('2d');

        var bild = new Image();
        bild.src = "himmel.jpg";

        var ballon = new Image();
        ballon.src = "ballone.png";

        var zufallszahl = Math.floor((Math.random() * 3)); //0,1,2

        function aufCanvasZeichnen() {
            context.drawImage(bild, 0, 0);

            context.drawImage(ballon, 80 * zufallszahl, 0, 80, 100,
          400, 120, 80, 100);
            context.font = "26px Verdana";
            context.fillStyle = '#FFFFFF';
            context.fillText("Ballon " + (zufallszahl + 1), 400, 250);

            context.font = "14px Verdana";
            context.fillStyle = '#000000';
            context.fillText("Drücke [F5] für einen neuen Ballon", 40,
          350);

        }
        ballon.onload = function () {
            aufCanvasZeichnen();
        }
    </script>
</body>

</html>
```

code_bsp09.htm


Ballon 1
Drücke [F5] für einen neuen Ballon

14 | Zusammenfassung

Nun sind wir an einem Punkt angelangt, wo fast alles gelernt wurde, um damit zu beginnen, sein eigenes kleines Spiel zu bauen. Der Code des Szenenwechsel-Beispiels und der Code des springenden Balls ergeben zusammen die Grundlage für unser konkretes Beispiel hier.

Anstatt des Beispiels mit dem Ball könnte man hier aber ab jetzt auch andere Spielideen umsetzen. Etwa ein kleines Plattform-Jump-and Run-Spiel, einen Shooter oder ein einfaches Autorennspiel.

Der Phantasie und Kreativität sind ja hier keine Grenzen gesetzt, und genau das macht ja ein gutes Online-Spiel aus. Eine Idee muss witzig sein, dann machen auch kleine Spiele Spaß.

Im letzten Tutorial zeigen wir, wie das funktioniert. Und wir stellen das Spiel fertig. Konkret: wir nehmen unseren Szenewechsel-Code und verbinden ihn mit der KickMe-Engine, wir laden selbstgezeichnete Bilder ins Canvas, geben Sound dazu und hoffen, dass dann alles funktioniert.

Zuvor möchten wir aber noch einmal die wichtigsten Prinzipien, Befehle und Anweisungen beim Programmieren zusammenfassen.

1. Java Script Code läuft innerhalb einer HTML-Datei:

HTML-Dateien lassen sich im einfachsten Falle sogar mit einem normalen Text-Editor schreiben. Üblicherweise verwendet man jedoch einen komfortableren HTML-Editor.

Einfache HTML-Struktur

```
<html>
<head>
</head>
<body>
Hallo Welt!
</body>
</html>
```

Mit der <script>-Anweisung wird nun dem HTML-Browser mitgeteilt, dass dies nun JavaScript Code ist, der entsprechend zu verarbeiten ist. Der Text "Hallo Welt!" wird in unserem Beispiel nun also betreits mittels Script in unser vorher definiertes HTML-Textfeld "meinTextFeld" geschrieben.

Einfache HTML-Struktur mit JavaScript

```
<html>
<head>
</head>
<body>
<span id='meinTextFeld'></span>

<script>
document.getElementById('meinTextFeld').innerHTML = "Hallo Welt!";
</script>

</body>
</html>
```

Wenn man das Zusammenspiel zwischen HTML und JavaScript verstanden hat, hat man bereits ein ganze Menge an kreativen Möglichkeiten zur Verfügung.

2. Programme arbeiten mit Variablen:

Klammersetzung beachten:
IF-Strukturen können beliebig kombiniert und verschachtelt werden.

Achte jedoch auf die richtige Klammersetzung.

Programme sind im Grunde Rechenmaschinen. Wie in der Mathematik gehts dann auch natürlich um Variablen. Im einfachsten Falle ist eine derartige Variable eine Zahl - (es kann aber auch ein Wort, ein Bild oder ein Liste sein). In unserem Beispielen mit dem hüpfenden Ball sind die wichtigsten Variablen die Lage auf der X-Achse (x), auf der Y-Achse (y), die Geschwindigkeit vx und vy.

```
<script>
var x = 300;
var y = 400;
var vx = 0;
var vy = 4;
//Ball wird versetzt:
x=x+vx;
y=y+vy;
</script>
```

Wichtig ist dabei, dass eine Variable bevor sie verwendet wird zuerst einmal deklariert werden muss. Dies geschieht am Programm-Anfang mit dem Befehl "var".

Der Vollständigkeit halber muss gesagt werden, dass es hier auch eine alternative Schreibweise gibt. Die Variablen unseres Beispiels könnten auch so geschrieben werden.

```
<script>
var x, y, vx,vy;
x=300;
y=400;
vx = 0;
vy = 4;
//Ball wird versetzt:
x=x+vx;
y=y+vy;
</script>
```

Klammersetzung beachten:
IF-Strukturen können beliebig kombiniert und verschachtelt werden.

Achte jedoch auf die richtige Klammersetzung.

3. Funktionen sind selbstprogrammierte Abläufe:

Wenn das Programm bestimmte Operationen immer weder durchführen muss, ist es günstig diese zu einer Funktion zusammen zu schreiben. Die Addition der Ball-Koordinaten mit ihrer Geschwindigkeit (vx,vy) ist hier in der Funktion zusammenRechnen ausgelagert. Wichtig ist dabei, dass eine Funktion erst aufgerufen werden kann, nachdem diese zuerst einmal definiert wurde. Das heißt: die Reihenfolge der Befehle und Codeteile ist immer ein besonders wichtiger Aspekt in der Programmierung. Wenn einmal ein Programm nicht funktioniert, sollte man daher immer zuerst die Reihenfolge der Befehle untersuchen!

```
<script>
var x, y, vx,vy;
x=300;
y=400;
vx = 0;
vy = 4;

function zusammenRechnen() {
x=x+vx;
y=y+vy;
}
//Ball wird versetzt:
zusammenRechnen();
</script>
```

4. Wenn-Dann-Abfragen:

Eine extrem wichtige Möglichkeit bieten if-Abfragen. Die Schreibweise zeigt das folgende Beispiel. Dieser Code würde etwa bewirken, dass die Geschwindigkeit vx nicht größer als 5 werden kann.

```
if (vx > 5) {vx=5;}
```

Klammersetzung beachten:
IF-Strukturen können beliebig kombiniert und verschachtelt werden.

Achte jedoch auf die richtige Klammersetzung.

Das Thema rund um die If-Abfragen ist sehr umfangreich, weil sich damit etliche Möglichkeiten bieten. So können diese Teile ineinander verschachtelt werden, zusamengehängt oder voneinander abhängig gemacht werden.

Unser nächstes Beispiel veranschaulicht eine derartige komplexe Abfrage. Wenn vx größer als 5 aber kleiner als 10 ist und wenn dann auch noch das vy genau 4 ist rufe die Funktion "tuWas" auf.

```
function tuWas() {console.log ("hallo");}

if (vx > 5 && vx < 10) {
  if (vy == 4) {tuWas ();}
}
```

5. Start the Engine

Ein Spiel braucht einen Taktgeber um Bewegung zu erzeugen. Damit werden im loop die Spielbewegungen berechnet. Dies ist nochmals das Beispiel aus Tutorial 6.

```
<html>
<head>
</head>
<body>
<h1 id="Anzeige"></h1>
<script>
var zahl = 0;
function loop() {
zahl++;
document.getElementById("Anzeige").innerText = zahl;
requestAnimationFrame(loop);
}
loop();
</script>
</body>
</html>
```

code_bsp06.htm

Allerdings verwenden wir hier eine etwas verfeinerte Version - vgl. Tutorial 7 - wo wir die Geschwindigkeit der Durchläufe genau festlegen können und den Antrieb ein- und abschalten können. So erhalten wir die volle Kontrolle über unseren Taktgeber. Beispielsweise läuft das Spiel dann auf allen Geräten gleich schnell und kann pausiert werden.

Der richtige Umgang mit dem **requestAnimationFrame**-Befehl ist schwierig. Das Gute am Programmieren ist aber, dass man nicht immer alles gleich verstehen muss. Immerhin gibt es ja auch die Kopierfunktion des Computers. So lässt sich auch fremder Code einbauen - wenn man weiß wo und wie - den man nicht 100% verstanden hat.

6. Und andere kryptische Codezeilen:

Wenn du an dieser Stelle angelangt bist, hast die Grundbegriffe des Programmierens verstanden und solltest mit dem bisher gelernten bereits ein ganze Menge anstellen können.

Allerdings sind dir sicher auch unzählige Codezeilen aufgefallen (falls du dir auch andere Codes angesehen hast, wie am Anfang des Tutorials empfohlen) , mit denen du gar nichts anfangen kannst.

Wofür steht etwa:

```
.
..
zahl++; //man könnte auch zahl = zahl + 1 schreiben
oder
spielzeit -= 1; //man könnte auch spielzeit = spielzeit - 1 schreiben
oder
jetzt = Date.now(); //Date.now () greift auf die Systemzeit zu

if (delta > intervall) {
dann = jetzt  - (delta % intervall);} // das %-Zeichen steht für Modulo
if (!Angehalten) {...} //ein "!" heißt NICHT! if (Anghalten==false)
..
.
```

Die Liste ließe sich noch lange weiterführen.

Dieses Tutorial ist aber nur eine Einführung und vieles kann in diesem Rahmen nicht erklärt werden. Es muss auch nicht, weil es für die Engtwicklung unseres Spiels nicht wichtig ist. Immerhin handelt es nur von einem hüpfendem Ball der mit der Maus angeklickt werden kann. Dieser verändert seine Lage auf dem Koordinatensystems des Canvas entsprechend seiner berechneten Flugbahn.

Das Problem ist mathematisch relativ einfach zu lösen - deshalb haben wir es ja auch ausgewählt.

Beim Selbst-Programmieren erlernen sollte man junbedingt immer ein kleines Handbuch parat haben oder öfter mal im Internet nachsehen.

Wichtige Themen, die hier nicht behandelt wurden sind

- for-Schleifen
- Listen
- Tastastureingaben
- um nur einige zu nennen

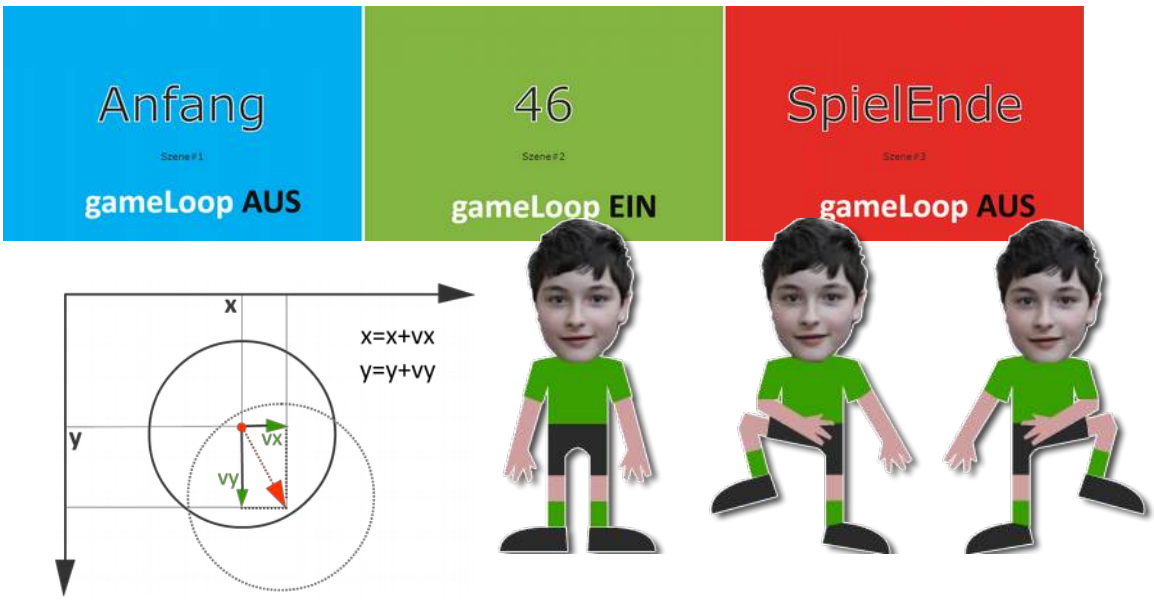
`vgl.: code_bsp15.htm`

Jetzt geht es also zur letzten Station dieses Tutorials: Ein komplettes Spiel entsteht. Dabei sind die Kenntnisse aus den vorhergehenden Tutorials gefordert. Besonders wichtig sind die Tutorials KickMe-Engine und Szenenwechsel. Das folgende Spiel setzt sich nämlich aus diesen beiden Code-Teilen zusammen.

15 | KickME - Minigame

In diesem letzten Tutorial werden wir nun die Programmierung des kompletten Spiels beenden. Dabei wird nochmals Schritt für Schritt das Wesentliche erklärt. Damit hast du auch eine gute Zusammenfassung und eine Kontrolle, ob du bisher alles richtig verstanden hast.

Besonders wichtig sind hierzu die Tutorials **KickMe-Engine** und **Szenenwechsel**. Das Spiel setzt sich nämlich hauptsächlich aus diesen beiden Code-Teilen zusammen. Dazu kommen noch Kenntnisse im Umgang mit **Bildern und Grafiken**. (vgl. **Canvas als Zeichenfläche für Fortgeschrittene**)



Zusätzlich lernen wir bei der Fertigstellung auch noch schnell etwas über Sounds. Die Einbindung funktioniert ähnlich wie bei Bildern.

```
var bummSound = new Audio('bumm.mp3');
spieleBumm = function () {
            bummSound.loop = false; //kein Loop, wird nur 1x gespielt
            bummSound.play();
}

spieleBumm(); //Funktionsaufruf
```

Unser GameLoop ist eine Folge von Funktionsaufrufen die 30 mal pro Sekunde (fps) durchgeführt werden. Die Namen der Funktionen sind hier selbsterklärend: Der Ball checkt ob er an der Wand angestoßen ist, dann berechnet er seine Flugbahn, die das alte Bild wird gelöscht und der Hintergrund, die Spielfigur und die ZeitAnzeige dargestellt. Zuletzt wird noch der Ball gezeichnet und die PunkteAnzeige gemacht.

```
        function gameLoop() {
            countDown();
            jetzt = Date.now();
            delta = jetzt - dann;
            if (delta > intervall) {
                dann = jetzt - (delta % intervall);
                if (!angehalten) {
                    checkWand();
                    berechneFlugbahn();
                    loescheAlteGrafik();
                    zeichneHintergrund();
                    macheSpielFigurSprite();
                    zeitAnzeige();
                    zeichneBall();
                    machePunkteAnzeige();
                }
            }
            requestID = requestAnimationFrame(gameLoop);
        }
```

Als Beispiel für eine verwendete Funktion sei hier die Punktezählung kurz erklärt:

```
function machePunkte() {count += 1;
if (count > countMAX) { countMAX = count;}
}
```

Sie wird jedesmal aufgerufen, wenn man den Ball mit der Maus klickt. Somit wird dann zu count +1 addiert. Wenn count größer wird als der bisher größte Wert von count (=countMAX), wird auch countMAX erhöht.

Fällt der Ball in unserem Spiel zu Boden, wird count übrigens wieder 0 gesetzt und das Spiel beginnt von vorne. Die Variable countMAX bleibt jedoch bei dem bisher höchsten Wert.

Der Code des fertigen Spiels enthält viele derartige Funktionen. Die Arbeit mit Funktionen erhöht die Übersicht im Code und erleichtert das Programmieren. Daher haben wir bewusst viele kleinere Funktionen geschrieben.

Komletter Code des Spiels

```
<html>
<!-- ~HTML-
         Kommentar:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
         ~~~ -->

<head>
    <meta charset="utf-8" /> <!-- ~~~~~ermöglicht Umlaute~~~ -->
    <!-- ~~~~~Canvas Style (hier oder im CSS)~~~ -->
    <style>
        #Spielflaeche {
            background: #4F4D4E;
            border-radius: 20px;
        }
    </style>
</head>

<body>
    <!-- ~~~~~~Die HTML-Datei bekommt ein CANVAS~~~~~ -->
    <canvas id="Spielflaeche" width="830" height="560"></canvas>
    <!-- ~~~~~~Jetzt gehts los mit dem JavaScript Teil:~~~~~ -->
    <script>
```

```
/* JavaScript-
  Kommentar:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ~~~~~~~~~~~~~ */
// JavaScript-
  Kommentar:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ~~~~~~~~~~~~~
var fps = 30;
var spielminuten = 2;
var spielzeit = spielminuten * 60 * fps;
var grundlinie = 490; //dort steht unsere Spielfigur
var xMAUS, yMAUS; //zum Abfragen der M;aquskoordinaten

var boden = 40; //Boden für Ball ist 40 von unten entfernt
var szene; //steuert Spiel-Szene Start-Spiel-Ende
var countMAX = 0; //Die höchste Trefferserie
var count; //aktuelle Anzahl der Treffer. Wenn Ball den Boden
  berührt wird dies wieder auf 0 gesetzt

//Die Variablen unseres Balls werden deklariert:
var x, y, vx, vy, schwerkraft, elastizitaet, energieverlust, r,
  impuls;

//Zeitgeber-Variablen
var dann = Date.now();
var intervall = 1000 / fps;
var jetzt, delta;
// JavaScript-
  Kommentar:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ~~~~~~~~~~~~~
var requestID; //nötig bei gameLoopSteuerung
var angehalten = true; //nötig bei gameLoopSteuerung
// JavaScript-
  Kommentar:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ~~~~~~~~~~~~~
//Korrekturwerte, verbessern die Spielbarkeit,HardCoding!:
var vxMAX = 4; // verhindert ein zu starkes Abrallen zur Seite
var vyMAX = 15; // verhindert eine zu hohe Fallgeschwindigkeit
var mausKorrektur = 24; //vergrößert Kreis-Fläche zum Klicken,
  acht so Spiel leichter
// JavaScript-
  Kommentar:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ~~~~~~~~~~~~~
//
//
//-----------------------------------------------------------
  --
//Definieren von "BildVariablen" mittels "new Image ()" und La-
  den von Bildern
var startBild = new Image(); //BildVariable wird definiert
startBild.src = "start.jpg"; //Pfad zum Bild
var spielBild = new Image();
```

```
spielBild.src = "game.jpg";
var endeBild = new Image();
endeBild.src = "gameover.jpg";
//////////////////////////
var fussball = new Image();
fussball.src = "ball.png";


// JavaScript-
 Kommentar:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 ~~~~~~~~~~~~~
//-------------------------------------------------------------
 --
//------------------------Die animierte Spielfigur laden mit
 "Bildstreifentechnik"-----
/////////////////Die Spielfigur
var spielFigur = new Image();
spielFigur.src = "sprite.png";
//--------Sprite-
 Variablen-----------------------------------------------------
 --
var spriteBreite = 160;
var spriteHoehe = 230;
var pose = 0;
//-------------------------------------------------------------
 ------------------
//-------------------------------------------------------------
 --
//Sounds werden geladen:
var audio1 = new Audio('bumm.mp3');
var audio2 = new Audio('doing.mp3');
var audio3 = new Audio('boing.mp3');

//Das Canvas wird  deklariert:
/*
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~HA
 UPT
 TEIL~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 ~~~~~~~ */
var canvas = document.getElementById('Spielflaeche');
var context = canvas.getContext('2d');
//Dem canvas wird ein "EVENT-LISTENER" für Mausabfragen zugeor-
 det. Damit kann eine Maussteuerung gebaut werden
canvas.addEventListener('mousedown', mausDruck); //Mausklick
 auf dem Canvas startet unsere Funktion "mausDruck"
canvas.addEventListener('mouseup', mausLos);
canvas.addEventListener('mousemove', mausBewegung);
/*
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 ~~~~~~ */
function mausBewegung(eventDaten) {
```

```
        xMAUS = eventDaten.clientX;
        yMAUS = eventDaten.clientY;
    }

    ////////////////////////////Die Mausfunktionen auf dem Canvas
    function mausDruck(eventDaten) {
        // die Variablen dx und dy geben den jeweiligen Abstand vom
     Mauszeiger zum Mittelpunkt an
        let dx, dy;
        dx = x - eventDaten.offsetX;
        dy = y - eventDaten.offsetY;
        //wenn der Ball angeklickt wurde
        if (Math.sqrt((dx * dx) + (dy * dy)) < r + mausKorrektur) {
            vx += impuls * dx / r;
            vy += impuls * dy / r;
            // ACHTUNG Profi-Schreibweise
            if (Math.abs(vx) > vxMAX) { vx = Math.sign(vx) * vxMAX;
     }//Elegante Schreibweise Math.sign ermittelt Vorzeichen -1, 0,
     +1
            if (Math.abs(vy) > vyMAX) { vy = Math.sign(vy) * vyMAX;
     }//Elegante Schreibweise Math.sign ermittelt Vorzeichen -1, 0,
     +1

            machePunkte();
            PlaySound3();
        }
        if (xMAUS < x) { pose = 2; }
        if (xMAUS >= x) { pose = 1; }
    }
    ////////
    ///////
    function mausLos(eventDaten) {
        if (szene == 1) { szene = 2; angehalten = false; can-
     vas.style.cursor = "pointer"; gameLoop(); }
        if (szene == 3) { if (xMAUS > canvas.width / 2 - 140 &&
     xMAUS < canvas.width / 2 + 140 && yMAUS < 500 && yMAUS >= 460)
     { szene = 1; gameStart(); } }
        if (szene == 2) { pose = 0; if (xMAUS > canvas.width - 110
     && yMAUS < 140 && yMAUS >= 100) { gameEnde(); } }
    }
    //////////////

    /////SZENE1 GameStart
    ////////////////////////////////////////////////////////////////
     ////////////////////////////////////
    function gameStart() {
        szene = 1;
        canvas.style.cursor = "default";
        spielzeit = spielminuten * 60 * fps;
        startWerteBall();
        angehalten = true;
```

```
        context.drawImage(startBild, 0, 0);
        if (requestID) {
            cancelAnimationFrame(requestID);
            requestID = undefined;
        }
        context.font = "40px Verdana";
        context.fillStyle = '#FFFFFF';
        context.font = "30px Verdana";
        context.fillText("Versuche den Ball so lange wie möglich in
     der", 60, canvas.height - 105);
        context.fillText("Luft zu halten, ohne dass er den Boden
     berührt", 60, canvas.height - 65);
        context.fillStyle = "#55A30A";
        let btnB = 200;
        let btnH = 50;
        context.fillRect(canvas.width / 2 - btnB / 2, canvas.height
     / 1.5 - btnH / 2, btnB, btnH);
        context.font = "40px Verdana";
        context.fillStyle = '#FFFFFF';
        context.fillText("PLAY", canvas.width / 2 - 44, can-
     vas.height / 1.5 + 16);
    }
    ////////////////////////////////////////////////////////////////
     ////////////////////////////////////

    /////SZENE2 GameLoop
    ////////////////////////////////////////////////////////////////
     ////////////////////////////////////
    function gameLoop() {
        countDown();
        jetzt = Date.now();
        delta = jetzt - dann;
        if (delta > intervall) {
            dann = jetzt - (delta % intervall);
            if (!angehalten) {
                checkWand();
                berechneFlugbahn();
                loescheCanvas();
                zeichneHintergrund();
                macheSpielFigurSprite();
                zeitAnzeige();
                zeichneBall();
                machePunkteAnzeige();
            }
        }
        requestID = requestAnimationFrame(gameLoop);
    }
    ////////////////////////////////////////////////////////////////
     ////////////////////////////////////
    //////////
    ////////////////SZENE3 GameEnde
```

```
////////////////////////////////////////////////////////////////////
 ////////////////////////////////////
function gameEnde() {
    canvas.style.cursor = "default";
    szene = 3;
    loescheCanvas();
    angehalten = true;
    context.drawImage(endeBild, 0, 0);
    if (requestID) {
        cancelAnimationFrame(requestID);
        requestID = undefined;
    }
    context.font = "62px Verdana";
    context.fillStyle = '#FFFFFF';
    context.fillText("Gratulation!", 220, canvas.height - 240);
    context.font = "34px Verdana";
    context.fillText("Du hast den Ball " + countMAX + " mal in
 Folge gekickt!", 90, canvas.height - 140);
    context.font = "20px Verdana";
    context.fillStyle = '#000000';
    context.fillText("Weiter mit Mausklick", canvas.width / 2 -
 120, canvas.height - 75);
}
////////////////////////////////////////////////////////////////////
 ////////////////////////////////////
//Funktionen:
//////////////////////
function loescheCanvas() {
    context.clearRect(0, 0, canvas.width, canvas.height);
}
/////////////
function zeichneHintergrund() { context.drawImage(spielBild, 0,
 0); }

/* ~~~~~~~ball~~~~~~ */
function startWerteBall() {
    schwerkraft = 0.5;
    elastizitaet = 0.6;
    energieverlust = 0.9; r = 40;
    impuls = 38;
    x = canvas.width / 2;
    y = -r;
    vx = 0;
    vy = 0;
    count = 0;
}
function checkWand() {
    if (x + r >= canvas.width) {
        vx = -vx * elastizitaet;
        x = canvas.width - r;
        PlaySound2();
```

```
        }
        else if (x - r <= 0) {
            vx = -vx * elastizitaet;
            x = r;
            PlaySound2();
        }
        if (y + r >= canvas.height - boden) {
            vy = -vy * elastizitaet;
            y = canvas.height - boden - r;
            vx *= energieverlust;
            count = 0;
            if (Math.abs(vy) > 1) { PlaySound1(); }
        }
    }
    ///
    function berechneFlugbahn() {
        vy += schwerkraft;
        x += vx;
        y += vy;
    }
    function zeichneBall() {
        let R = 40;
        context.drawImage(fussball, x - R, y - R);
        context.beginPath();
        context.strokeStyle = "#CCCCCC";
        context.lineWidth = 5;
        context.arc(x, y, r, 0, 2 * Math.PI, false);
        context.stroke();
    }
    ////////////////////////////////////////////////

    function macheSpielFigurSprite() {
        context.drawImage(spielFigur, pose * spriteBreite, 0, spri-
      teBreite, spriteHoehe, xMAUS - (spriteBreite / 2), grundlinie
      - spriteHoehe, spriteBreite, spriteHoehe);
    }
    /*SpriteTechnik: TEILE EINES BILDES DARSTELLEN:
    mittels   context.drawImage ( image, sx, sy, sw, sh, dx, dy,
      dw, dh );
     vgl.: https://www.mediaevent.de/javascript/canvas-pixel.html
     Erklärung:
     sx, sy:        Linke obere Ecke des Bildausschnitts
     sw, sh:        Breite und Höhe des Ausschnitts aus dem Origi-
      nalbild
     dx, dy:        Linke obere Ecke des Bildausschnitts im Canvas
     dw, dh:        Breite und Höhe des Ausschnitts im Canvas
     */
    ////////////////////////////////////////////////////ZEIT:
    ////
    ////////////
    function countDown() {
```

```
            spielzeit -= 1;
            if (spielzeit < 0) {
                gameEnde();
            }
        }
        ////////////////
        function zeitAnzeige() {
            context.fillStyle = '#6A6D6A';
            context.fillRect(50, canvas.height - 35, Math.round(spiel-
          zeit / fps) * 6, 20); //hardcoding
            context.fillStyle = '#FFFFFF';
            context.font = "15px Verdana";
            context.fillText("Verbleibende Zeit: " + Math.round(spiel-
          zeit / fps) + " Sekunden", 60, canvas.height - 19);
        }
        ///
        function machePunkte() { count += 1; if (count > countMAX)
          { countMAX = count; } }
        /////
        function machePunkteAnzeige() {
            context.font = "18px Verdana";
            context.fillText("Kicks: " + count, 60, 30);
            context.font = "25px Verdana";
            context.fillText("Dein Rekord: " + countMAX, canvas.width -
          220, 35);
        }
        ////////////////////////SOUND
        PlaySound1 = function () {
            audio1.loop = false;
            audio1.play();
        }
        PlaySound2 = function () {
            audio2.loop = false;
            audio2.play();
        }
        PlaySound3 = function () {
            audio3.loop = false;
            audio3.play();
        }
        ///////

        //Wichtig Ladekontrolle:

        startBild.onload = function () {
            gameStart();
        }
    </script>
</body>
</html>
```

code_bsp16.htm

Natürlich findest du alle Dateien auch im Internet.

Am einfachsten http://www.gorilla.at/scriptkiddy/