



## Inhaltsverzeichnis

Einleitung.....	3
Erklärung des Programms .....	5
Ablauf des Spieles.....	5
HTML.....	5
JAVASCRIPT .....	7
Variablen .....	7
So beginnt das Spiel .....	9
Der Fullscreen Modus .....	10
Positionierung der Mauern und der Gegner .....	11
Die Class .....	12
Start.....	14
Die Koordinaten .....	15
Das Update.....	16
Autonomie der Gegner .....	17
Der Schuss.....	18
Wie kann man Schüsse stoppen .....	19
Gegner halten auch Schüsse an .....	22
Der Spieler kann den Canvas nicht mehr verlassen.....	24
Freie Bewegung vom Spieler.....	25
Der Letzte große Schritt .....	26
Game Over .....	27
Fachwörter .....	28
Quellen .....	31
Bilder Nachweis .....	31

## Einleitung

Ich erstelle ein Spiel mit der Programmiersprache JavaScript und dokumentiere die Fortschritte, die ich im Laufe des Schuljahres (19-20) gemacht habe. Dieses Spiel ist über das Internet verfügbar und kann jederzeit auf einem Computer gespielt werden. (Hier der Link: [https://www.lem.codes/users/herfe564/hit\\_man/hit\\_man.html](https://www.lem.codes/users/herfe564/hit_man/hit_man.html))

Auf die Idee bin ich gekommen, da ich mich die Jahre davor schon ziemlich für Programmieren von Internetseiten interessiert habe. Aber das Programmieren von Spielen war vor einem Jahr noch viel zu komplex für mich, da ich nicht genug Mathematikkenntnisse hatte und auch weil ich nicht die genügende räumliche Wahrnehmung dazu hatte.

Um ein Spiel zu programmieren gehen Sie von nichts aus und Sie müssen alles einzeln definieren. Das Schwierigste daran ist, dass Sie als Programmierer das Spiel nicht als Spiel betrachten können sondern als Formen mit einer gewissen Höhe und Breite, die sich in einem Feld bewegen können.

Die Programmierung eines Spieles ist dieses Jahr mein TRAPE hauptsächlich, weil ich nächstes Jahr gerne eine 4e GIG (section ingénierie) im „Lycée des Arts et Métiers“ mache. Anschließend kann ich wählen ob ich eine 2e & 1e GIN (section informatique) oder die 2e & 1e GIG (section ingénierie) weiterführe. Ein weiterer Grund für diese Entscheidung ist, dass mir Physik und Mathematik sehr viel Spaß machen. Allerdings habe ich keine richtigen Pläne was ich darüber hinaus machen werde, sicher ist nur, dass ich irgendwas mit Physik, Informatik oder Mathematik studieren möchte.

In dieser Arbeit werden Sie öfters Kisten sehen können mit Wörtern die entweder **braun**, **rot**, **"blau"** oder noch **grün** sind. Diese Kisten enthalten gewisse Teile des Quellcode des Spieles, die Erklärung vom Quelltext befindet sich über oder unter der jeweiligen Kiste.

Wenn die Kiste leicht **blau grau** gefärbt ist, ist es ein HTML Dokument und die:

- **<braunen>** Wörter sind HTML Tags.
- **rote** Wörter sind Attribute. Die sind da, um eine zusätzliche Information in einem Tag hinzuzufügen. Z.B. wenn man dem Tag einen Wert zuordnen will, muss man dem Wert eine Eigenschaft geben, das ist ein Attribut.
- **"blaue"** Wörter sind Werte eines Attributs.
- **<!--Grüne-->** Wörter sind Kommentare, die den Quelltext nicht beeinflussen können.
- Schwarze Wörter sind normale Textstellen die, so wie sie im Quelltext stehen, auch auf der Internetseite angezeigt werden.
- **Hellgrüne** Wörter sind Eigenschaften von Tags die im Style definiert sind (siehe Fachwörter).

Wenn die Kiste leicht grau gefärbt ist, ist es ein JavaScript (kurz Js) Dokument und die:

- braunen Wörter sind Attribute von Funktionen.
- blauen Wörter sind definierte Funktionen von Js selbst.
- Grünen Wörter sind Werte.
- /\*Grüne\*/ Wörter sind Kommentare, die den Quelltext nicht beeinflussen können

Wenn die Kiste keine Farbe trägt kommt sie nicht im Quelltext vor und sie dient nur zu Demonstrationszwecken.

Wörter, die so unterstrichen sind, sind meist unverständliche Fachwörter die auf der Seite 29 erklärt werden.

Wenn Sie etwas nicht anhand der Fachwörter verstehen können, können Sie diese Internetseite besuchen, sie erklärt das Programmieren ziemlich gut anhand von vielen Beispielen, mir hat sie persönlich schon öfters geholfen.

<https://www.w3schools.com/default.asp>

# Erklärung des Programms

## Ablauf des Spieles

Wenn man sich auf der Internetseite des Spieles befindet, steht erstmals „PRESS SPACE TO START THE GAME“. Wenn man dann die Leertaste gedrückt hat, fängt das Spiel schon an. Es erscheint (spawnt) ein roter Spieler und ein paar Sekunden darauf erscheinen auch schon die Gegner, die Sie an ihrer dunkelgrünen Farbe erkennen. Diese bewegen sich auf den Spieler zu und fügen ihm Schaden zu bis dieser stirbt. Um das zu verhindern kann der Spieler die Gegner abschießen.

## HTML

Das hier (siehe Kasten unten) ist das [HTML](#) Dokument vom Spiel, es ist in zwei Hauptteile eingeteilt der `<head>` und der `<body>`. Alle zwei befinden sich im `<html>` [Tag](#). Dieser [Tag](#) ist da, um dem Browser Bescheid zu sagen, dass es sich um ein [HTML](#) Dokument handelt.

```
<!DOCTYPE html>
<html>
  <head>
    <title>title</title>
    <meta charset="utf-8">
    <link href="game.css" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=VT323&display=swap"
      rel="stylesheet">
  </head>
  <body>
    <div id="gamearea">
      <canvas
        id="game"
        height="700"
        width="1000"
        onmousemove="showCoords(event)"
        onmouseout="clearCoor()"
        onmousedown="mouseDown()"
        onmouseup="mouseUp()">
      </canvas>
    </div>

    <p id="demo_points"></p>
  </body>
  <script src="game.js"></script>
</html>
```

Im `<head>` befinden sich der Titel der Internetseite (Titel im Browser Tab), [Metadaten](#) und Links die zu anderen [Quellcodes](#) führen, die aber zu der Internetseite gehören. Z.B. können Sie

hier sehen, dass ich den [CSS/Style](#) der Internetseite verlinkt habe. Genauso habe ich auch eine [Metadatei](#) in den `<head>` gesetzt die dem Browser sagt, dass ich mit dem lateinischen Alphabet schreibe, sonst würde der Browser die verschiedenen Zeichen: ë; ü; ï; ö; é; è; ê und so weiter nicht erkennen und nachher falsch anzeigen. Dort im `<head>` habe ich aber auch einen Link eingesetzt, der eine Schriftart verlinkt, damit ich im Nachhinein eine andere Schriftart benutzen kann als die, die von [HTML](#) als Standard festgelegt wurde.

Im `<body>` wird hingegen der Inhalt der Internet Seite definiert. Den Inhalt kann man anschließend mit [CSS](#) optisch anders im Internet darstellen. In diesem Fall habe ich ein [Div](#), zwei Paragraphen (die im Nachhinein wichtig werden) und die Links zu den [JavaScript](#) Dokumenten in den `<body>` gesetzt.

Anschließend habe ich ein Canvas in den [Div](#) gesetzt. Der Canvas ist die Stelle im [Quelltext](#), wo sich im Nachhinein das Spielfeld befinden wird. Den [Div](#) in dem sich der Canvas befindet ist da, weil der Canvas nicht vom HTML als Objekt angesehen wird und ohne den [Div](#) würde der Canvas alles auf der Internetseite überschreiben.

Die verschiedenen Attribute vom Canvas die mit "onmouse" beginnen sind [JavaScript](#) Funktionen, die eine andere Funktion auslösen. Das klingt anfangs schwer ist es aber nicht. Z.B. wird die „onmousemove“ Funktion aufgerufen, wenn sich die Maus im Feld vom Canvas bewegt. Wenn das der Fall ist, ruft die „onmousemove“ Funktion, die „showCoords(event)“ Funktion auf. Die Aktionen der Funktion, die im Wert aufgerufen werden, sind im [JavaScript](#) definiert.

Unter dem [Div](#) befindet sich ein Paragraph (p) mit der ID `gamearea`, diese hat zwei Zwecke, einerseits können Sie den Paragraphen mit dem [Style / CSS](#) verbinden oder, andererseits können Sie anhand von der ID verschiedene [Tags](#) vom [JS](#) aus aufrufen und dem Text z.B. eine andere Farbe geben.

Darunter wird der `<body>` schon wieder geschlossen und noch bevor der `<html>` [Tag](#) sich schließt habe ich ganz zum Schluss den [JavaScript](#) verlinkt. Es ist besser ihn am Schluss zu verlinken um verschiedene Fehler beim Laden der Internetseite zu vermeiden.

Und schließlich habe ich den `</html>` [Tag](#) wieder geschlossen.

## JAVASCRIPT

Jetzt wird alles ein bisschen komplizierter, das hier ist jetzt nämlich das JavaScript Dokument. Dort habe ich das Spiel programmiert. Um es einfacher zu erklären werde ich das Spiel so erklären wie es auch der Reihe nach abläuft.

### Variablen

Um mit der Erklärung der Programmierung des Spieles zu beginnen müssen Sie als erstes verstehen was Variablen sind, denn mit ihnen fängt das Programmieren erst an.

Eine Variable ist eigentlich wie ein Ort, in dem etwas gespeichert werden kann. Es gibt vier verschiedene Arten von Informationen, die in einer Variablen gespeichert werden können.

- Die einfachste Art von Information ist einfach eine Zahl. Eine Variable kann aber auch Rechnungen ausführen. Z.B. kann man den Wert von zwei Variablen addiert. Von [JavaScript](#) selbst gibt es aber auch verschiedene Funktionen, die eine Zahl generieren können. Wie die `Math.floor(Math.random())` Funktion, sie wählt eine zufällige Zahl zwischen 0 und 1 mit der Null inbegriffen und der Eins nicht inbegriffen.

```
var Zahl = 10;  
var zufälligeZahl = Math.floor(Math.random());
```

- Dann kann aber eine Variable auch eine Eigenschaft besitzen. Sie kann entweder „false“ oder „true“ sein, das ist ein bisschen wie ein An-und-aus-Knopf.

```
var eigenschaft = false;
```

- Eine Variable kann auch eine Folge von Buchstaben und Zahlen besitzen. Solche Variablen besitzen einen bestimmten Namen: „String“. (Sie werden immer zwischen zwei Anführungszeichen geschrieben.) Man kann im Nachhinein einen String auch bearbeiten oder Informationen über ihn herausfinden. Z.B kann man mit der Funktion „.length“ die Länge eines Strings herausfinden, Sie müssen dann nur den Namen vom String vor den „.length“ hinschreiben.

```
var myString = "#!?!_abc_123";
```

- Als letztes gibt es noch sogenannte „Arrays“ das sind die kompliziertesten Variablen zu verstehen. In den Arrays kann man nämlich alle die schon erwähnten Fälle speichern, und zwar so viel wie man möchte. Dies kann nützlich sein, wenn Sie z.B. mit vielen Objekten in einem Spiel arbeiten und wenn alle diese Objekte den gleichen Wert besitzen sollen. Dann brauchen Sie nicht zehn Mal dieses Objekt zu definieren, sondern einmal und Sie müssen dann nur, anschließend den Vorgang ein paar Mal wiederholen anhand einer „For-Schleife“. Um etwas in einen Array rein zu setzen, muss man den Namen des Arrays hin schreiben und dann hinter den Namen „.push();“. Zwischen den Klammern können Sie dann das gewollte Objekt in den Array setzen. Das was Sie in den Array „pushen“ wird sich an der ersten Stelle befinden, und alles andere wird eine Stelle nach vorne gedrückt.

```
var myArray = ["10"; "false"; "#!?!_abc_123"];  
var myArray.push("ich_setzte_etwas_in_ein_Array");
```



## So beginnt das Spiel

Wenn Sie sich auf der Internetseite meines Spiels befinden, werden Sie aufgefordert die Leertaste zu drücken. Der Satz, der euch dazu auffordert, befindet sich in dem Canvas (siehe Seite 6). Den Canvas muss ich sowohl im HTML wie auch im [JS](#) definieren, darum habe ich die folgenden Variablen benutzt.

```
var g =  
document.getElementById('game')  
var ctx = g.getContext("2d");  
var canvasWidth = g.width;  
var canvasHeight = g.height;
```

Die erste Variable (`var g`) beinhaltet den [Style / CSS](#) vom [Tag](#) mit der ID „Game“. Einfacher erklärt, steht sie für den [Tag](#) `<canvas>`. Die zweite Variable (`var ctx`) macht dem [JS](#) noch einmal klar, dass der

`<canvas>` zwei ein dimensionales Dokument ist. Deswegen befindet sich auch das `g` vor dem „`getContext("2d");`“. Es besagt, dass die `var g` ein zweidimensionales Dokument ist. Jetzt weiter zu der `var canvasWidth`, sie ist die Breite des `<canvas>`, da `width` ein prädefiniertes Objekt ist von [JS](#) ist. Darunter befindet sich auch logischerweise die Höhe vom `<canvas>`.

Um den Satz „PRESS SPACE TO START THE GAME“ in den `<canvas>` zu schreiben habe ich eine Variable benutzt die anfangs die Eigenschaft `false` besitzt und damit eine `if(){};` Funktion aktiviert. Eine `if` Funktion wird so wie im vorigen Satze ausgeschrieben. Solche Funktionen werden erst durchgeführt, wenn eine bestimmte Kondition erfüllt wird. Diese Kondition wird zwischen den Klammern ausgeschrieben. In diesem Fall überprüft die `if` Funktion ob die „`srt`“ Variable `false` ist (deswegen die `==`) und nur dann erfüllt sie die Aufgabe die zwischen den zwei `{}` steht.

```
var srt = false;  
  
if (srt == false){  
    ctx.font = "50px VT323";  
    ctx.fillText("PRESS SPACE TO START THE GAME", canvasWidth/8,  
        canvasHeight/2);  
};
```

Da die Variable „`srt`“ von Anfang an `false` ist wird die Funktion sofort ausgeführt und schreibt den Satz „PRESS SPACE TO START THE GAME“. Um den Satz zu schreiben habe ich erstmals die Schriftart definiert ("`50px VT323`") diese habe ich ja schon im [HTML](#) verlinkt (siehe Seite 6) aber um das auszuführen benutzt man im [JS](#) „`ctx.font =`“ `ctx` steht ja für den `<canvas>` und `.font` steht für die Schriftart. Anschließend, haben Sie vielleicht ja schon bemerkt, dass vor dem Namen der Schriftart „`50px`“ steht, das steht für die Größe die der Text im Nachhinein haben soll. Was ihr da wissen müsst ist, dass „`50px`“ nicht für die Pixel des Computer stehen sondern eine eigene Einheit ( $1\text{px} \approx 0.26458333333333$ ), auch wenn man diese [`ˈpɪksəl`] ausspricht. Dann brauchte ich nur noch hinter „`ctx.fillText(`“ den Text hinzuschreiben, der im Nachhinein im `<canvas>` stehen soll und die X, Y Koordinate eingeben. Um dort die Mitte zu finden habe ich erstmals ein Achtel der Canvasbreite angegeben und bei der Höhe habe ich die Hälfte der Canvashöhe angegeben, so wird sich der Text immer in der Mitte des `<canvas>` befinden, auch wenn ich die Höhe des `<canvas>` im Nachhinein ändere.

## Der Fullscreen Modus

Anschließend habe ich eine Funktion eingebaut, die es ermöglicht in einen „Fullscreen Modus“ zu wechseln indem man die Taste F drückt.

```
window.addEventListener("keydown", function(e){  
  
    if( e.keyCode == 70 /*F*/ ) {  
        openFullscreen();  
    }  
  
});
```

Um zu erkennen, wann die F Taste gedrückt wird benutzt man eine gewisse Funktion (`window.addEventListener("keydown", function(e){});`) die von [JS](#) selbst schon definiert ist, und es den darin enthaltenen `if` Funktionen ermöglicht, egal wo der Mauszeiger sich auf der Seite befindet, darauf zu hören wenn eine Taste gedrückt wird.

Die `if` Funktionen, die sich darin befinden haben immer als Kondition „`e.keyCode ==`“ mit einer Zahl hinten dran. Das liegt daran, dass der kleine „`e`“ [JavaScript](#) noch einmal daran erinnert, dass es sich um die `"keydown"`, `function(e)` handelt. Deswegen auch das kleine „`e`“ zwischen den Klammern hinter `function(e)`.

„`keyCode ==`“ steht für einen Code die alle Tasten besitzen, und mit dem Sie alle Tasten unterscheiden können. Um diesen Code herauszufinden gibt es verschiedene Internetseiten, mit denen ihr es herausfinden könnt. Also keine Angst da müssen Sie nichts auswendig lernen.

Um mir das Verstehen des Programmes zu vereinfachen habe ich aber hinter den „`keyCodes`“ einen Kommentar geschrieben mit dem jeweiligen Buchstaben (`/*F*/`).

Wenn jetzt der Buchstabe „F“ gedrückt wird, dann wird die Funktion „`openFullscreen()`“ ausgeführt und diese setzt das Öffnen des Fullscreen Modus um.

```
function openFullscreen() {  
    if (elem.requestFullscreen) {  
        elem.requestFullscreen();  
    } else if (elem.mozRequestFullScreen) { /* Firefox */  
        elem.mozRequestFullScreen();  
    } else if (elem.webkitRequestFullscreen) { /* Chrome, Safari & Opera */  
        elem.webkitRequestFullscreen();  
    } else if (elem.msRequestFullscreen) { /* IE/Edge */  
        elem.msRequestFullscreen();  
    }  
}
```

Um jetzt zu erklären was alle diese verschiedene `if` und `else if` Funktionen in der `openFullscreen()` Funktion zu tun haben ist ein bisschen kompliziert aber was ihr wissen müsst ist das die verschiedenen Funktionen für den Jeweiligen Browser gedacht sind. Das ist manchmal ein bisschen blöd, aber ihr müsst wissen das nicht alle JS Funktionen auf allen Browsern funktionieren, auch wenn [JS](#) eine Universelle [Programmiersprache](#) ist.

## Positionierung der Mauern und der Gegner

Anschließend werden die Positionen der Gegner und der Mauern, die der Spieler nicht durchdringen kann, ausgerechnet. Fangen wir mit den Positionen der Gegner an, diese werden in einer For-Schleife ausgerechnet. Diese wiederholt sich so oft wie die Variable wallnumber groß ist. Das habe ich so gemacht, da ich dann nur die Variable wallnumber ändern muss um die Anzahl der Mauern zu ändern.

```
for (var i = 0; i < wallnumber; i++){
    bPosX.push(Math.floor(Math.random() * canvasWidth));
    bPosY.push(Math.floor(Math.random() * canvasHeight));
    bSizeX.push(Math.floor(Math.random() * 90) + 30);
    bSizeY.push(Math.floor(Math.random() * 90) + 30);
    var c = Math.floor(Math.random() * 155) + 50;
    bcolor.push("rgb(" + c + ", " + c + ", " + c + ")");
    blocks.push( new Block(bPosX[i], bPosY[i], bSizeX[i], bSizeY[i]) );
};

for (var i = 0; i < enmienumber; i++){
    var posX = (Math.floor(Math.random() * canvasWidth - enSizeX));
    var posY = (Math.floor(Math.random() * canvasHeight - enSizeY));

    enemies.push(new Enmie(posX, posY, enSizeX, enSizeY))
};
```

In der For-Schleife wird in der ersten Zeile nach jeder Wiederholung der Schleife eine zufällige X Position in den Array bPosX „gepusht“. Das mache ich mit einer von JS festgelegten Funktion.

Die `Math.random()`; Funktion generiert eine zufällige Zahl zwischen 0 und 0.9. Um aber eine zufällige Zahl im Canvas zu bekommen, muss ich die Funktion `Math.floor()`; benutzen. Sie rundet die Zahl auf 0 ab und dann kann ich die `Math.random()`; Funktion mit der Größe des Canvas multiplizieren, so wird in meinem Fall eine zufällige Position zwischen 0 und 900 gewählt. Bei der Y Position ist es das gleiche nur, dass dort eine Zahl zwischen 0 und 550 gewählt wird.

Die Größen der Mauern werden auch in dieser For-Schleife festgelegt, und zwar multipliziere ich die Zufällige Komma zahl mit 90 und addiere nachher noch 30 dazu. So wird es schlussendlich eine zufällige Zahl zwischen 30 und 120 und die Mauern werden nicht zu groß und nicht zu klein.

Um die zufällige Farbe der Mauern auszurechnen habe ich ein Farbensystem von JS benutzt (rgb). Dieses rgb-System beinhaltet über 16 Millionen Farben und ist so aufgebaut, dass die Farben rot, grün, blau einen Wert von 0 bis 255 besitzen können. Wenn der Wert von allen Farben gleich ist wird es eine Farbe zwischen Weiß und Schwarz. Da es das ist das ich möchte, habe ich eine Variable c erstellt die einen zufälligen Wert besitzt zwischen 50 und 205, weil ich nicht möchte, dass meine Mauern ganz schwarz oder weiß sind.

Anschließend pushe ich die Variable `c` in den Array `bcolor` und mit ein bisschen Text arrangiere ich mich so, dass es wie ein String ist und es von JS als rgb Code erkannt wird. Für die Gegner habe ich mich genauso angelegt wie bei den Mauern, nur, dass ich dort nur die Positionen ausrechne, weil ja die Größen und Farben der Gegner immer gleich sind.

## Die Class

Was ihr vielleicht bemerkt habt ist, dass ich die letzte Zeile bei beiden [For-Schleifen](#) nicht erklärt habe. Dort wird nämlich eine `Class` in ein Array gepusht.

```
class Block{
  constructor(bPosX, bPosY, bSizeX, bSizeY){
    this.posX = bPosX;
    this.posY = bPosY;
    this.sizeX = bSizeX;
    this.sizeY = bSizeY;
  }
  draw(color){
    ctx.fillStyle = color;
    ctx.fillRect( this.posX, this.posY, this.sizeX,this.sizeY );
  }
};
class Enmie{
  constructor(enPosX, enPosY, enSizeX, enSizeY){
    this.posX = enPosX;
    this.posY = enPosY;
    this.sizeX = enSizeX;
    this.sizeY = enSizeY;
    this.deleted = false;
  }
  draw(color){
    ctx.fillStyle = color;
    ctx.fillRect( this.posX, this.posY, this.sizeX,this.sizeY );
  }
};
class Man{
  constructor(mPosX, mPosY, mSizeX, mSizeY){
    this.posX = mPosX;
    this.posY = mPosY;
    this.sizeX = mSizeX;
    this.sizeY = mSizeY;
  }
  draw(color){
    ctx.fillStyle = color;
    ctx.fillRect( this.posX, this.posY, this.sizeX,this.sizeY );
  }
};
```

```
class Bullet{
  constructor(buPosX, buPosY, buSizeX, buSizeY, buspeedX, buspeedY){
    this.posX = buPosX;
    this.posY = buPosY;
    this.sizeX = buSizeX;
    this.sizeY = buSizeY;
    this.buspeedX = buspeedX;
    this.buspeedY = buspeedY;
  }
  draw(color){
    ctx.fillStyle = color;
    ctx.fillRect( this.posX, this.posY, this.sizeX, this.sizeY );
  }
};
```

Wenn man eine **Class** in JavaScript kreiert, erschafft man ein Objekt dem man Attribute zu eignen kann. Diesen Attributen kann man anschließend einen Wert geben, aber das alles passiert in einer noch anderen Funktion, und zwar im **constructor**. Um es kurz zu fassen, warum ich **Classen** benutze, müsst ihr einfach vorerst wissen, dass man anhand einer Class ein Objekt in eine Variable setzen kann und es so einfacher ist damit umzugehen im Nachhinein.

Wie ihr sicher sehen könnt, stehen hinter dem **constructor** zwischen Klammern verschiedene Variablen, die die Größen und Positionen des jeweiligen Objektes definieren. Diese stehen dort, weil ich sie weiterhin in der Funktion benutzen werde. Nun stehen in der Funktion einmal verschiedene Wörter mit **this**. davor und anschließend eine Variable dahinter.

Diese **this**. Wörter sind nichts anderes als Variablen, die nur in der **Class** gültig sind. Um ihnen aber den gewollten Wert (die gewollten Größen und Positionen) zu geben muss ich die Variablen oben in den Klammern mit denen im **constructor** gleichsetzten. Das ist so wichtig, da ich nur mit den **this**. Variablen in der **Class** arbeiten kann. So kommen wir auch schon zum nächsten Punkt, und zwar der Funktion unter dem **constructor**.

Diese draw Funktion ist da, um die Gegner zu malen, in den Klammern der Funktion steht color. Das ist wiederum eine Variable der man im Nachhinein, wenn man den Befehl gibt das Objekt zu malen, einen Wert/Farbe geben kann. Das passiert in der Funktion, da sage ich dem JavaScript in der ersten Zeile nämlich, dass er dem nachfolgenden Objekt (in der nächsten Zeile) die jeweilige Farbe geben soll. Wie ganz am Anfang, benutze ich hier auch wieder das ctx. mit der fillRect Funktion, diese ermöglicht es ein gefülltes Rechteck zu malen. Hinter dem fillRect stehen auch schon in Klammern die **this**.-Variablen die die jeweiligen Koordinaten und Größen darstellen.

Das alles passiert mit den vier verschiedenen Objekten in dem Canvas (**Block**, **Enmie**, **Man**, **Bullet**). Die **Block** und **Enmie** Class werden wie vorher erklärt in einer [For-Schleife](#) in die Variable enmie gepusht - zu den anderen werde ich später noch im update() kommen.

## Start

Kommen wir jetzt wieder zu der „srt“ Variable, bis jetzt habe ich ja nur erklärt, wie ich einen Satz in den Canvas schreiben kann und wie die Fullscreen Funktion funktioniert.

```
if( e.keyCode == 32 /*SPACE*/) {  
    srt = true;  
    start();  
}
```

Wie vorher befindet sich die `if` Funktion in der `"keydown"` Funktion und wenn diese durch die Leertaste aktiviert wird, wechselt die Variable „srt“ von der `false` Eigenschaft auf die `true` Eigenschaft. Dabei wird aber auch die `„start();“` Funktion ausgeführt.

```
function start(){  
    if(srt == true){  
        ctx.clearRect( 0, 0, g.width, g.height );  
        setTimeout(function(){ wave = true; }, 3000);  
        requestAnimationFrame(update);  
    };  
}
```

Die `„start()“` Funktion beinhaltet eine `if` Funktion, die erst ausgeführt wird, wenn die „srt“ Variable `true` ist, was sie ja in dem Moment schon ist, da sie ja schon in der `"keydown"` Funktion auf `true` gesetzt wurde. Die `if` Funktion entfernt erstmals den vorigen Satz den ich in den `<canvas>` geschrieben habe, indem ich diese `„ctx.clearRect( 0, 0, g.width, g.height );“` Funktion durchgeführt habe.

Sie ist wie viele eine von [JS](#) definierte Funktion und entfernt nicht nur, wie vorhergesagt den Satz, sondern sie löscht alles was sich in den da hinteren Koordinaten befindet. Um auch sicher zu gehen, dass es alles nur im `<canvas>` löscht sollten Sie, wie ich, vor dem `clearRect()`; andeuten, dass es sich auch um das `<canvas>` handelt, deswegen steht auch das `ctx` davor.

Nach dem Löschen des Inhalts vom `<canvas>` habe ich eine Funktion benutzt, die nach 3 Sekunden eine Variable auf `true` setzt, die im Nachhinein die Funktion aktiviert die die Gegner erscheinen lässt.

Anschließend wird das Spiel auch schon gestartet mit der `requestAnimationFrame(update);` Funktion, sie ist der Ort wo alles während des Spieles berechnet wird. Der `requestAnimationFrame();` ist da, um dem [Browser](#) zu sagen, dass er die Funktion zwischen den Klammern mit der größtmöglichen Taktrate wiederholen soll, wie der Browser es ermöglicht.

## Die Koordinaten

Bevor ich euch jetzt erkläre was in der update Funktion passiert komme ich erstmals zu den kleineren Sachen, die im Spiel passieren, so wie die Funktion die es ermöglicht die Koordinaten unten rechts anzuzeigen, wenn der Mauszeiger sich im Canvas befindet.

```
onmousemove="showCoords(event)"
onmouseout="clearCoor()"
```

Die zwei Attribute im `<canvas>` lösen verschiedenen Funktionen aus die, die Koordinaten anzeigen. Das erste Attribut `onmousemove=""` aktiviert die Funktion `showCoords(event)` wenn sich der Mauszeiger im `<canvas>` befindet. Das zweite Attribut `onmouseout=""` aktiviert jedoch die `clearCoor()` Funktion wenn der Mauszeiger sich außerhalb des `<canvas>` befindet.

Im [JS](#) sieht das so aus.

```
var cPosX = 0;
var cPosY = 0;
var rect = 0;

function showCoords(event) {
    rect = event.target.getBoundingClientRect();
    cPosX = event.clientX - rect.left; //x position within the element.
    cPosY = event.clientY - rect.top; //y position within the element.

    updateCoordinateDisplay();
}

var coor = 0;

function updateCoordinateDisplay() {
    coor = "X/" + Math.round(cPosX) + ", Y/" + Math.round(cPosY);
}
```

Um die Koordinaten anzuzeigen habe ich erstmals im JavaScript drei Variablen festgelegt. Die erste `var cPosX` wird im Nachhinein die X Position des Spielers speichern, die zweite macht genau dasselbe nur mit der Y Position. Die dritte wird den X und Y Abstand zwischen dem Canvas und der Internetseite berechnen, da dieser je nach Größe des Bildschirmes ändert.

Um jetzt diesen Variablen aber auch einen Wert zu geben benutze ich die `showCoords(event)` Funktion, die im [HTML](#) aufgerufen wird, wenn der Mauszeiger sich im Canvas bewegt. Vielleicht habt ihr es schon bemerkt, aber hinter dem Namen der Funktion befindet sich in den Klammern das Word „event“. Das ist da, um dem [JS](#) zu befehlen, dass er alle möglichen Daten vom Canvas speichert, wie z.B. die Mausposition im Canvas, die Bewegung des Mauszeigers u.s.w.

In der Funktion setzte ich als erstes fest, dass die Variable `rect` auch die X und Y Abstände beinhaltet. Dafür benutze ich auch schon das „event“ und noch andere Daten, um schließlich nur die Daten zu haben die ich nachher brauche. Um die Maus X und Y Positionen im Canvas zu finden benutze ich auch das „event“, da es mit Hilfe des „events“ ja, die Mauspositionen

auf der Internetseite speichert. Anschließend habe ich nur noch die rect X und Y Positionen mit den X und Y Mauspositionen subtrahiert. So bekomme ich immer die richtige Mauspositionen im Canvas.

Um die Koordinaten dann aber auch richtig anzuzeigen aktiviert die `showCoords(event)` Funktion noch eine andere Funktion die die Koordinate schön in den String „`coor`“ einfügt. Dabei habe ich aber die X und Y Koordinaten vom Mauszeiger mit der `Math.round()` multipliziert, um zu verhindern, dass eine Kommazahl angezeigt wird.

```
ctx.font = "30px VT323";  
ctx.fillStyle = "black";  
ctx.fillText(coor,0,canvasHeight-10);
```

Angezeigt wird das Ganze in der `update()` Funktion und zwar ganz zum Schluss, damit die Koordinaten alles überschreiben und sich nicht z.B. hinter dem Spieler zu verstecken.

```
function clearCoor() {  
    coor = "X/-, Y/-"  
}
```

Die `clearCoor()` Funktion wird wie vorher gesagt im [HTML](#) aktiviert und löscht alle Daten aus dem `coor` String wenn sich der Mauszeiger außerhalb des Canvas befindet.

## Das Update

Kommen wir jetzt zu dem `update()`, dort wo alles während dem Spiel berechnet wird. Also die `update()` Funktion wird von der `start()` Funktion aufgerufen und befiehlt, dass die `update()` Funktion sich mit der größtmöglichen Taktrate wiederholt. So kommen wir auch schon zu der ersten Funktion im `update()`. Wie vorher schon erklärt löscht die `ctx.clearRect( 0, 0, g.width, g.height);` Funktion alles was sich im Canvas befindet, dies benötigt man ganz am Anfang einer Update Funktion um nach jedem [Frame](#) alles zu löschen was sich im vorigen [Frame](#) befand. Wenn man das nicht tun würde, würden alle Objekte, die sich im Canvas bewegen, eine Spur hinterlassen.

```
function update(){  
    ctx.clearRect( 0, 0, g.width, g.height );  
    ...  
};
```



## Autonomie der Gegner

Anschließend habe ich eine `if` Funktion eingefügt die sich aktiviert wenn die `wave` Variable die `true` Eigenschaft besitzt (diese passiert ja nach drei Sekunden in der `start()` Funktion). Wenn das dann der Fall ist, wird eine For-Schleife (`for`) aktiviert die die X und Y Speeds der Gegner ausrechnet. Dies wird im `update()` berechnet, weil die Gegner euch verfolgen sollen und nicht nur in eine Richtung sich bewegen, das wäre der Fall wenn die Speed nur einmal berechnet würden.

```
function update(){
  ...
  if (wave == true) {
    for (var i = 0; i < enmienumber; i++){

      enDiffX[i] = mPosX - enemies[i].posX; //a²
      enDiffY[i] = mPosY - enemies[i].posY; //b²

      enDist[i] = Math.sqrt(Math.pow(enDiffX, 2) + Math.pow(enDiffY, 2));

      enspeedX[i] = enDiffX[i] / enDist[i]; //bullet speed immer gleich X
      enspeedY[i] = enDiffY[i] / enDist[i]; //bullet speed immer gleich Y

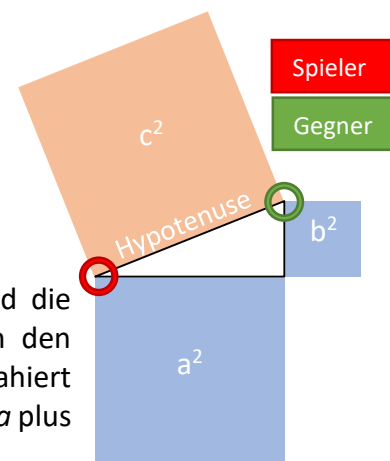
      enspeedX[i] = 3*enspeedX[i];
      enspeedY[i] = 3*enspeedY[i];

      enemies[i].posX = enemies[i].posX + enspeedX[i];
      enemies[i].posY = enemies[i].posY + enspeedY[i];

      if (enemies[i].deleted == false){
        enemies[i].draw("green");
      };
    };
  };
  ...
};
```

Dafür habe ich eine bestimmte Formel benutzt, die rechnet die Distanz zwischen den Gegnern und dem Spieler aus, indem  $a^2 + b^2 = c^2$  ergibt. Das hat Pythagoras herausgefunden.

Diese Formel ist in meinem Fall schwer wieder zu erkennen da, ich  $a$  und  $b$  in meinem Programm anders benannt habe. So ist bei mir  $a$  die Variable `enDiffX` und  $b$  die `enDiffY`. Die `enDiffX` und die `enDiffY` habe ich ausgerechnet indem ich die Positionen von den Gegnern (`enemies.posX/Y`) von denen vom Spieler (`mPosX/Y`) subtrahiert habe. Anschließend habe ich die Wurzel von dem Quadrat von  $a$  plus das Quadrat  $b$  genommen, um  $c$  herauszufinden.



Diese Länge  $c$  habe ich dann in die `enDist` eingefügt um anschließend die `speedX` und `Y` des Gegners auszurechnen. Da ich bis jetzt nur die Hypotenuse kenne und nicht die Geschwindigkeit, die die Gegner besitzen müssen, um den Spieler zu verfolgen, muss ich jeweils die `enDiffX` und die `enDiffY`, durch die `enDist` teilen.

Da die Geschwindigkeit jetzt noch nicht schnell genug ist, habe ich sie mit drei multipliziert (`enspeedX[i] = 3*enspeedX[i]; enspeedY[i] = 3*enspeedY[i];`). Dann muss aber noch festgelegt werden, dass es sich bei den `enspeedX` und `enspeedY` auch um Geschwindigkeiten handelt. Dafür habe ich in der nächsten Zeile die Position mit dem Speed vom jeweiligen Gegner addiert. So ändert die `update()` Funktion nach jedem Frame die Positionen der Gegner.

Nachdem diese Geschwindigkeit ausgerechnet wird, werden die Gegner im `update()` gemalt. Das passiert in einer `if` Funktion, die aktiviert wird wenn die Variable `this.deleted` die `false` Eigenschaft besitzt. Das ist eine Eigenschaft der Class `Enmie`, sie ist grundsätzlich `false` aber im Nachhinein, wenn die Gegner abgeschossen werden, wechselt diese Eigenschaft und dann werden die Gegner nicht mehr gemalt und für den Spieler sind sie verschwunden. Aber später mehr dazu.

## Der Schuss

Jetzt komme ich zu der nächsten Funktion, und zwar zu der die, die Schüsse ausrechnet. Diese befindet sich nicht in der `update()` Funktion, sondern außerhalb, da die Schüsse immer in die gleiche Richtung schießen sollen und nicht während dem Schuss die Richtung ändern.

```
function mouseDown() {
    shootBullet();
    //setInterval(shoot = false, 1000);
}

-----
function shootBullet() {
    buPosX = mPosX + mSizeX/2 - buSizeX/2;
    buPosY = mPosY + mSizeY/2 - buSizeY/2;;

    buDiffX = cPosX - buPosX;
    buDiffY = cPosY - buPosY;

    buDist = Math.sqrt( Math.pow( buDiffX, 2 ) + Math.pow( buDiffY, 2 ) );

    buspeedX = buDiffX / buDist;
    buspeedY = buDiffY / buDist;

    buspeedX = 6*buspeedX
    buspeedY = 6*buspeedY

    bullet.push( new Bullet(buPosX, buPosY, buSizeX, buSizeY, buspeedX,
    buspeedY) );
};
```

Die Funktion `shootBullet()` wird aktiviert durch die Funktion `mouseDown()` und diese wird wiederum von einem event im [HTML](#) aktiviert, wenn Sie einen rechts-klick oder links-klick ausführen. Wenn das der Fall ist und die `shootBullet()` Funktion ausgeführt wird, macht sie das gleiche wie die `if` Funktion um die Gegner auszurechnen. Nur werden dieses Mal andere Variablen eingesetzt und andere Werte, aber das Prinzip bleibt das Gleiche. Und am wichtigsten müssen Sie wissen, dass jedes Mal, wenn Sie auf die Maus drücken diese Funktion sich wiederholt und ein neues Objekt in den Array gepusht wird. Eine andere wichtige Sache die Sie wissen müssen ist, dass die X/Y Speeds der Bullets in der `Bullet` Class eine Variable besitzen, so kann ich die momentane Speed ausrechnen die eine Bullet haben soll, und diese im `update()` wieder benutzen, so dass die Bullet sich immer in die gleiche Richtung bewegt.

### Wie kann man Schüsse stoppen

Ein anderer Teil der Funktion um die „Bullets“ auszurechnen befindet sich aber im `update()`.

```
function update(){
  ...
  for (var i= 0; i < bullet.length; i++){

    var deletethisbullet = false;

    bullet[i].posX += bullet[i].buspeedX
    bullet[i].posY += bullet[i].buspeedY

    bullet[i].draw("#a34400");

    //walls
    if ((bullet[i].posX > canvasWidth) ||
        (bullet[i].posX < 0) ||
        (bullet[i].posY > canvasHeight) ||
        (bullet[i].posY < 0)){
      deletethisbullet = true;
    };

    for(var j = 0; j < wallnumber; j++){
      //top_blocks
      if(bullet[i].posX + bullet[i].sizeX > blocks[j].posX &&
        bullet[i].posX < blocks[j].posX + blocks[j].sizeX &&
        bullet[i].posY + bullet[i].sizeY > blocks[j].posY &&
        bullet[i].posY + bullet[i].sizeY < blocks[j].posY + 5.1){
        deletethisbullet = true;
      };
      //bottom_blocks
      if( bullet[i].posX + bullet[i].sizeX > blocks[j].posX &&
        bullet[i].posX < blocks[j].posX + blocks[j].sizeX &&
        bullet[i].posY < blocks[j].posY + blocks[j].sizeY &&
        bullet[i].posY > blocks[j].posY + blocks[j].sizeY - 5.1 ) {
        deletethisbullet = true;
      };
    };
  };
}
```

```
//left_blocks
if(bullet[i].posY + bullet[i].sizeY > blocks[j].posY &&
bullet[i].posY < blocks[j].posY + blocks[j].sizeY &&
bullet[i].posX + bullet[i].sizeX > blocks[j].posX &&
bullet[i].posX + bullet[i].sizeX < blocks[j].posX + 5.1){
    deletethisbullet = true;
};

//right_blocks
if( bullet[i].posY + bullet[i].sizeY > blocks[j].posY &&
bullet[i].posY < blocks[j].posY + blocks[j].sizeY &&
bullet[i].posX < blocks[j].posX + blocks[j].sizeX &&
bullet[i].posX > blocks[j].posX + blocks[j].sizeX - 5.1 ) {
    deletethisbullet = true;
};

***
};

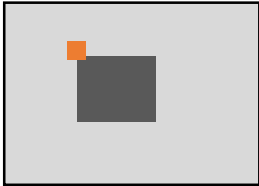
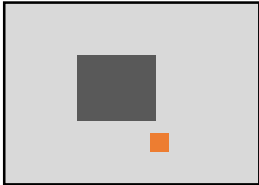
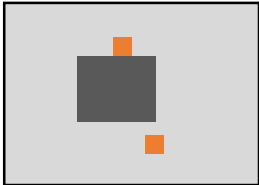
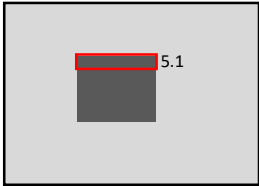
...
};
```

Das ist wieder eine For-Schleife die sich so oft wiederholt wie die Variable `bullet` (dort werden die Bullets mit allen ihren Informationen gespeichert) lang ist. Als erstes habe in dieser Funktion eine Variable festgelegt namens `deletethisbullet`, diese besitzt am Anfang die Eigenschaft `false`. Wenn diese Variable dann aber im Nachhinein die Eigenschaft `true` besitzt, dann wird die jeweilige Bullet gelöscht, aber später mehr dazu. Anschließend wie bei allen Objekten die sich bewegen wird die Speed X und Y noch einmal ausgerechnet, so dass die Bullets sich auch bewegen. Nachdem ich das getan habe, muss ich die Kugeln aber noch malen so wie ich es in der `Class` festgelegt habe.

Dann kommt noch eine kleine `if` Funktion, die aktiviert wird, wenn entweder die X oder Y Positionen der Bullets sich außerhalb des Canvas befinden. Dafür habe ich die zwei „||“ benutzt die so viel heißen wie oder. Wenn die Bullets sich dann auch außerhalb des Canvas befinden, dann befiehlt die `if` Funktion, dass die Variable `deletethisbullet` die Eigenschaft `false` besitzen soll und durch eine Funktion, zu der ich später kommen werde, wird die Bullet gelöscht.

Als nächstes befindet sich noch eine andere For-Schleife in der schon vorher erwähnten For-Schleife. Diese wird so oft wiederholt wie die Variable `wallnumber` lang ist. In der Schleife befinden sich vier `if` Funktionen, die jeweils für eine Seite der vier Seiten von allen Mauern stehen. Wie ihr sicher schon bemerkt habt, habe ich über die Funktionen in einem Kommentar geschrieben, um welche der vier Seiten es sich handelt. Diese Funktionen verhindern, dass die Bullets durch die Mauern fliegen. Ich werde nur eine der vier `if` Funktionen erklären müssen, da es sich ja immer um das Gleiche handelt.

Nehmen wir das Beispiel der oberen Seite ([//top\\_blocks](#)) dort habe ich:

- als erste Kondition festgelegt, dass die X Position der Kugel addiert mit der breite der Kugel **grösser sein muss als** die obere Seite des Blocks (alle), wo die Kugel gerade abprallt/verschwinden soll. 
- als zweite Kondition festgelegt, dass die Bullet X Position **kleiner sein muss** als die Block X Position addiert mit der Breite des jeweiligen Blocks. Um aber zu verhindern, dass die Kugel unter oder über dem Block verschwindet habe ich eine dritte Kondition hinzugefügt. 
- als dritte Kondition festgelegt, dass die Y Position der Bullet addiert mit der Höhe der Höhe **grösser sein muss** als die Block Position Y. Das Problem bei der Sache ist aber, dass Die Kugel unter dem Block auch verschwinden würde. Deswegen habe ich noch eine Kondition dazu gefügt. 
- als vierte Kondition festgelegt, dass die Y Position der Kugel plus die Y Größe der Kugel **kleiner sein muss** als die Block Position Y addiert mit einer Zahl in diesem Fall 5,1. Nachdem ich diese Kondition festgelegt habe kann die Kugel nur noch innerhalb des roten Feldes verschwinden. Diese 5,1 habe ich ausgewählt da der [JS](#) eine Zeitspanne benötigt um bemerken, dass die Kugel die obere Linie überschritten hat. Um zu wissen, wie groß dieser abstand sein muss, muss man es einfach nur ausprobieren solange es geht. 

Wenn die vier Konditionen erfüllt sind, dann wird die Eigenschaft der Variable `deletethisbullet` auf `true` umgesetzt. Mit den drei anderen Seiten bin ich genauso vorgegangen, wie ich es euch jetzt erklärt habe. Jetzt komme ich zu der Funktion, die die jeweilige Bullet löscht, die entweder den Canvas verlässt oder eine der Seiten eines Blocks berührt.

```
if(deletethisbullet == true){
    bullet.splice(i, 1);
};
```

Die Funktion, die die Bullet löscht, befindet sich auch in der Schleife, wo auch die Kugeln gemalt werden und ist eine `if` Funktion, die aktiviert wird, wenn die Variable `deletethisbullet` die Eigenschaft `true` besitzt. Wenn das der Fall ist, wird die jeweilige Kugel durch die `splice()` Funktion gelöscht. Sie ist eine Funktion wie `push()` die einen Array verändert. Um die Funktion `splice()` richtig zu benutzen, muss man in den Klammern zwei Zahlen angeben, getrennt von einem Komma. Die erste Zahl gibt an, an der wievielten Stelle im Array ein Objekt gelöscht werden soll und die zweite, wie viele Objekte gelöscht werden sollen. In meinem Fall muss ich als erste Zahl den `i` von der Schleife eingeben, weil das ja die Bullet ist, die gerade bearbeitet wird, und als zweite Zahl eine `1`, weil ich ja nur ein Objekt aus dem Array entfernen möchte.

### Gegner halten auch Schüsse an

Aber vor der `if` Funktion, die die Kugeln entfernt, gibt es noch eine andere [For-Schleife](#) (dort wo sich die drei `***` befinden), die das gleiche ausrechnet mit den vier Seiten, nur, dass es sich diesmal um die Gegner handelt. Die Kugeln sollen genauso, wie die Gegner verschwinden, wenn sie aufeinanderprallen.

```
function update(){
...
***
    for(var j = 0; j < enmienumber; j++){

        var deletethisenmie = false;

        //top_enmie
        if(bullet[i].posX + bullet[i].sizeX > enmies[j].posX &&
        bullet[i].posX < enmies[j].posX + enSizeX &&
        bullet[i].posY + bullet[i].sizeY > enmies[j].posY &&
        bullet[i].posY + bullet[i].sizeY < enmies[j].posY + 5.1){
            deletethisbullet = true;
            deletethisenmie = true;
        };
        //bottom_enmie
        if(bullet[i].posX + bullet[i].sizeX > enmies[j].posX &&
        bullet[i].posX < enmies[j].posX + enSizeX &&
        bullet[i].posY < enmies[j].posY + enSizeY &&
        bullet[i].posY > enmies[j].posY + enSizeY - 5.1 ) {
            deletethisbullet = true;
            deletethisenmie = true;
        };
        //left_enmie
        if(bullet[i].posY + bullet[i].sizeY > enmies[j].posY &&
        bullet[i].posY < enmies[j].posY + enSizeY &&
        bullet[i].posX + bullet[i].sizeX > enmies[j].posX &&
        bullet[i].posX + bullet[i].sizeX < enmies[j].posX + 5.1){
            deletethisbullet = true;
            deletethisenmie = true;
        };
        //right_enmie
        if(bullet[i].posY + bullet[i].sizeY > enmies[j].posY &&
        bullet[i].posY < enmies[j].posY + enSizeY &&
        bullet[i].posX < enmies[j].posX + enSizeX &&
        bullet[i].posX > enmies[j].posX + enSizeX - 5.1 ) {
            deletethisbullet = true;
            deletethisenmie = true;
        };
    };
};
```

```

        if (deletethisenmie == true){
            enemies[j].deleted = true;
        };

    };
    ***
    ...
};

```

Das erste in der Schleife ist eine Variable namens `deletethisenmie`, die Variable besitzt von Anfang an die Eigenschaft `false`. Das verändert sich aber, wenn eine der vier `if` Funktionen aktiviert wird. Wenn das der Fall ist, dann erfüllt sich ganz unten in der Schleife eine `if` Funktion, die dann die `this.deleted` Variable des jeweiligen Gegners von `false` auf `true` setzt.

Wenn das der Fall ist, dann werden wie vorher erwähnt (siehe [Class](#)), die Gegner nicht mehr gemalt und der Gegner erscheint nicht mehr im Spiel. Um dem Spiel aber ein Ende zu geben, habe ich anschließend mit dem gleichen Prinzip wie die zwei anderen [For-Schleifen](#) eine Funktion erstellt die das Spiel beendet sobald ein Gegner den Spieler berührt.

```

function update(){
    ...
    for(var j = 0; j < enmienumber; j++){

        //top_man
        if(mPosX + mSizeX > enemies[j].posX &&
        mPosX < enemies[j].posX + enSizeX &&
        mPosY + mSizeY > enemies[j].posY &&
        mPosY + mSizeY < enemies[j].posY + 5.1){
            gameOver = true;
        };

        //bottom_man
        if(mPosX + mSizeX > enemies[j].posX &&
        mPosX < enemies[j].posX + enSizeX &&
        mPosY < enemies[j].posY + enSizeY &&
        mPosY > enemies[j].posY + enSizeY - 5.1 ) {
            gameOver = true;
        };

        //left_man
        if(enemies[j].deleted == false &&
        mPosY + mSizeY > enemies[j].posY &&
        mPosY < enemies[j].posY + enSizeY &&
        mPosX + mSizeX > enemies[j].posX &&
        mPosX + mSizeX < enemies[j].posX + 5.1){
            gameOver = true;
        };
    };
}

```

```

        //right_man
        if(mPosY + mSizeY > enemies[j].posY &&
           mPosY < enemies[j].posY + enSizeY &&
           mPosX < enemies[j].posX + enSizeX &&
           mPosX > enemies[j].posX + enSizeX - 5.1 ) {
            gameOver = true;
        };
    };
    ...
};

```

Dafür aktivieren aber diesmal die `if` Funktionen in der Schleife die `gameOver` Variable. Zu der werde ich ganz zum Schluss noch mal kommen.

### Der Spieler kann den Canvas nicht mehr verlassen

Nach dem berechnet wird, ob der Spieler die Gegner berührt, wird die Bewegung vom Spieler im Canvas beschränkt. Das heißt einfacher gesagt, dass die Speed vom Spieler auf null gesetzt wird wenn er den Rand vom Canvas berührt.

```

function update(){
    ...
    if( ( mPosX + mSizeX > g.width && mspeedX > 0 ) ||
        ( mPosX < 0 && mspeedX < 0 ) ){
        mspeedX = 0;
    };

    if( (mPosY + mSizeY > g.height && mspeedY > 0 ) ||
        (mPosY < 0 && mspeedY < 0 ) ){
        mspeedY = 0;
    };

    mPosX += mspeedX
    mPosY += mspeedY

    var man = new Man(mPosX, mPosY, mSizeX, mSizeY);

    man.setColor("#e04d43");

    ...
};

```

Wie Sie sehen können, habe ich eine `if` Funktion benutzt, um den Spieler zu stoppen, wenn er eine der Seiten des Canvas berührt. In der Funktion habe ich als erste Kondition festgelegt, dass die Position vom Spieler addiert mit seiner Größe, grösser sein muss als der Canvas. Um zu verhindern, dass der Spieler nicht mehr weg kommt vom Rand, muss die Speed ein Wert besitzen (grösser als 0). So bin ich mit den vier Seiten des Canvas vorgegangen.



Anschließend habe ich wieder, wie bei allen Objekten, die sich bewegen, die Speed schlussendlich mit der Position vom Spieler addiert. Was ich bis jetzt nur in Array gemacht habe, habe ich jetzt nur einmal in eine Variable eingesetzt. Und zwar habe ich die Class nur einmal in eine Variable einfügen müssen, da es je nur einen Spieler gibt. Als letztes habe ich, dann auch schon den Spieler gemalt.

### Freie Bewegung vom Spieler

Bevor ich ihnen der Letzten Schritt erkläre komme ich nochmal kurz zu einer Funktion außerhalb des update(). Und zwar der Funktion, die es ermöglicht, dass der Spieler sich auch bewegen kann.

```
window.addEventListener("keydown", function(e){

    if( e.keyCode == 87 /*W*/ ) {
        mspeedY = -5;
    }

    if( e.keyCode == 83 /*S*/ ) {
        mspeedY = 5;
    }

    if( e.keyCode == 65 /*A*/ ) {
        mspeedX = -5;
    }

    if( e.keyCode == 68 /*D*/ ) {
        mspeedX = 5;
    }

});

window.addEventListener( "keyup", function(e){

    if( e.keyCode == 87 /*W*/ ) {
        mspeedY = 0;
    }

    if( e.keyCode == 83 /*S*/){
        mspeedY = 0;
    }

    if( e.keyCode == 65 /*A*/ ) {
        mspeedX = 0;
    }

    if( e.keyCode == 68 /*D*/ ) {
        mspeedX = 0;
    }

});
```

Ich habe die Funktion schon einmal am Anfang erwähnt, wo ich Ihnen erklärt habe wie Sie in den Fullscreen Modus gelangen. Aber hier lösen die `if` Funktionen der `keydown` Funktion keine anderen Funktionen aus, sondern verändern die Speed vom Spieler. Z.B., wenn Sie sich nach oben bewegen möchten und die Taste „W“ gedrückt halten ist die Y Speed des Spielers `-5`. Um aber die Speed wieder auf 0 zu bringen, wenn man die Taste loslässt, so dass der Spieler stehen bleibt, benutze ich die `keyup` Funktion. Sie ist das Gegenteil der `keydown` Funktion und löst die `if` Funktionen erst dann aus, wenn Sie die Taste loslassen.

### Der letzte große Schritt

Jetzt kommt der letzte große Schritt, damit der Spieler nicht durch die Wände des Canvas dringen kann, dafür habe ich wieder eine For-Schleife mit vier Konditionen benutzt.

```
function update(){
...
  for (var i= 0; i < wallnumber; i++){

    blocks[i].draw(bcolor[i]);

    //top
    if( mPosX + mSizeX > blocks[i].posX &&
        mPosX < blocks[i].posX + blocks[i].sizeX &&
        mPosY + mSizeY > blocks[i].posY &&
        mPosY + mSizeY < blocks[i].posY + 5.1) {
      mspeedY = 0;
      mPosY = blocks[i].posY - mSizeY;
    };

    //bottom
    if( mPosX + mSizeX > blocks[i].posX &&
        mPosX < blocks[i].posX + blocks[i].sizeX &&
        mPosY < blocks[i].posY + blocks[i].sizeY &&
        mPosY > blocks[i].posY + blocks[i].sizeY - 5.1 ) {
      mspeedY = 0;
      mPosY = blocks[i].posY + blocks[i].sizeY;
    };

    //left
    if( mPosY + mSizeY > blocks[i].posY &&
        mPosY < blocks[i].posY + blocks[i].sizeY &&
        mPosX + mSizeX > blocks[i].posX &&
        mPosX + mSizeX < blocks[i].posX + 5.1) {
      mspeedX = 0;
      mPosX = blocks[i].posX - mSizeX;
    };
  }
}
```

```

        //right
        if( mPosY + mSizeY > blocks[i].posY &&
            mPosY < blocks[i].posY + blocks[i].sizeY &&
            mPosX < blocks[i].posX + blocks[i].sizeX &&
            mPosX > blocks[i].posX + blocks[i].sizeX - 5.1 ) {

            mspeedX = 0;
            mPosX = blocks[i].posX + blocks[i].sizeX;
        };

    };
    ...
};

```

In der ersten Zeile der Schleife male ich die Gegner mit den zufälligen Zahlen, die ich am Anfang generiert habe. Anschließend habe ich dieses Mal zwei Reaktionen festgelegt, die erfüllt werden, wenn die vier Konditionen erfüllt sind. Die erste Reaktion macht, dass der Spieler sich nicht mehr in die jeweilige Richtung bewegt. Um zu verhindern, dass der Spieler sich nicht mehr zurück bewegen kann habe ich die Reaktionszeit von [JavaScript](#) benutzt. Denn der Spieler dringt durch die Reaktionszeit immer ein wenig in die Mauer ein, so Teleportiert die Zweite Reaktion den Spieler auf die Genaue Linie der jeweiligen Mauer. Das können Sie auch im Spiel feststellen, wenn Sie die Taste gedrückt halten und sich auf eine Mauer zu bewegen, dann wackelt der Spieler hin und her.

### Game Over

Jetzt kommen wir zum Schluss des Spieles, die Letzte Funktion des update() und damit auch die letzte im JavaScript. Wie schonmal vorher erwähnt ist das Spiel hinüber, wenn die Variable gameOver die Eigenschaft true besitzt.

```

function update(){
    ...
    if (gameOver == false){
        requestAnimationFrame( update );
    }
    else {
        ctx.clearRect( 0, 0, g.width, g.height );
        ctx.font = "100px VT323";
        ctx.fillText("GAME OVER", canvasWidth/8, canvasHeight/2);
    };
    ...
};

```

Im Normalfall besitzt sie ja die false Eigenschaft und wie Sie sehen können wird das update() so oft wiederholen, so lange gameOver false ist. Wenn das aber Mal nicht der Fall sein sollte wird die else Funktion aktiviert und alles im Canvas wird erstmal gelöscht und anschließend habe ich eine Funktion durchgeführt die GAME OVER in den Canvas schreibt. Und dabei ist das Spiel vorbei.

## Fachwörter

**CSS (Cascading Style Sheets)** ist wie **HTML** keine **Programmiersprache**. Anhand von CSS kann man Bilder einfügen, Größen definieren, dem Text Farben zuordnen und noch vieles mehr. Im **Quelltext** kommt das CSS im Style oder halt in einem verlinkten Dokument vor. Man kann aber auch die Eigenschaften im **Tag** selbst definieren.

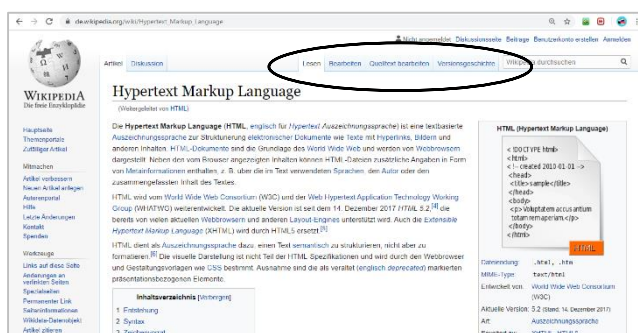
Wenn der Text eine gewisse Farbe haben soll, gibt es zwei Möglichkeiten: entweder man definiert die Farbe im **Tag** selbst. Das sieht wie hier in dem ersten Beispiel aus. Wenn dieser **Tag** aber mehr als eine Eigenschaft besitzt, werden öfters die Eigenschaften im Style definiert. Der Style ist ein **Tag** der sich im **<head>** befindet. Aber um den **Tag** mit dem Style zu verbinden muss den **Tag** eine „id“ oder eine „class“ besitzen. Der Unterschied zwischen den zwei ist, dass eine „id“ nur einem **Tag** zugeordnet werden kann und eine

„class“ mehrere **Tage** zugeordnet werden kann. Wenn man eine „id“ im Style definieren möchte muss man ihr wie im zweiten Beispiel einen „#“ davorsetzen und die Eigenschaften vom **Tag** wie im zweiten Beispiel zwischen zwei Klammern setzen. Wenn ein **Tag** jedoch eine „class“ besitzt setzt man einen Punkt vor die class im Style.

```
<body>
  <p style="color: red">
    p steht für Paragraph
  </p>
</body>
```

```
<style>
  #paragraf {
    color: yellow;
  }
  .rot {
    Background-color: red;
  }
</style>
```

```
<p id="paragraf">
  „p“ steht für Paragraph
</p>
<p class="rot">
  background-color definiert wie der
  Text markiert ist
</p>
```



Ein **Div** ist ein **Tag** in den Sie alles schreiben können was Sie möchten. Er erscheint anfangs nicht auf der Internetseite aber im Nachhinein können Sie ihm weitere Größen, Farben usw. zuordnen. Dieser **Tag** ist hilfreich, um z.B. eine Leiste in die Internetseite einzufügen die dem Kunden hilft um genaue Suchergebnisse zu erlangen.

Eine **For-Schleife** ist eine Art von Funktionen in **JavaScript** und ist da um einen Teil von einem Programm gewünschte Male zu wiederholen. Um solch eine Schleife zu programmieren muss man verschiedene Schritte ausführen.

Um anzufangen müsst Ihr in die Klammern eine Variable setzen mit einem Wert von null (diese gibt es nur in der Funktion selbst), anschließend müsst ihr der Funktion sagen, dass sie einmal aufhören soll. Das erreichen Sie wenn Sie bestimmen, dass die vorhin festgelegte Variable kleiner sein soll, als eine gewisse Zahl. Diese Zahl legt fest wie oft die Schleife sich wiederholt.

Und am Schluss müssen Sie nur noch die Variable hinschreiben, die Sie am Anfang festgelegt haben, mit zwei „+“ damit sich die Variable bei jeder Wiederholung der Schleife um eine Zahl vergrößert. Das ist wichtig damit die Variable auch die gewünschte Zahl erreicht und dann aufhört

```
for (var i= 0; i < 7; i++){  
  
};
```

Anschließend kommen die Arrays ins Spiel, die For-Schleifen werden fast immer mit ihnen benutzt. Da ein Array viele Objekte beinhalten kann, und Sie vielleicht mit allen diesen Objekten etwas durchführen möchten, aber nicht wollt, dass Sie für alle einzelne Objekte die Funktion die Sie durchführen möchten neu niederschreiben müsst, dann könnt ihr die Variable i benutzen die Sie vorher in der For-Schleife definiert haben. Das würde dann so aussehen.

```
var myarray = ["1", "2", "3"];  
  
for (var i= 0; i < myarray.length; i++){  
    myarray[i] = myarray[i] + 2;  
};
```

Hier wird die For-Schleife so oft durchgeführt wie die Variable myarray lang ist (myarray.length) und bei jeder der Wiederholungen der For-Schleife addiert die Schleife dem Objekt vom Array wo die Schleife sich gerade befindet (myarray[i]).

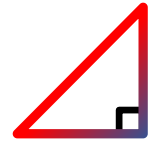
Ein **Frame** ist ein Bild von vielen die in einer Sekunde ablaufen. Um euch es bildlicher vorzustellen: ein Spiel hat meistens eine gewisse Anzahl an Fps (meist 60) dies ist eine Abkürzung „für Frames per second“ (Bilder pro Sekunde). Wenn die Fps über 60 liegen, dann läuft euer Spiel fleissend. Das liegt daran, dass jede Sekunde 60 Bilder vor euch ablaufen.

**HTML (Hyper Text Markup Language)** die in sogenannten **Tags** ausgeführt „<--->“ wird, ist keine **Programmiersprache** wie die meisten Leute fälschlicherweise meinen. Das liegt daran, dass man mit HTML nur vordefinierte Befehle ausführen kann, z.B kann HTML nicht 1+1 rechnen. Sie müssen sich vorstellen, dass eine Internetseite die nur mit HTML geschrieben wurde, sowie auf dem Bild rechts aussehen würde. Um Bilder, Tabellen,



Hintergrundfarben, ..., einfacher gesagt, um die Internetseite lebendiger zu gestalten benutzt man CSS zusätzlich zum HTML.

Hypotenuse ist die Seite eines rechtwinkligen Dreiecks die nicht mit dem rechten Winkel verbunden ist, wie ihr hier sehen könnt.



JavaScript (kurz JS) wurde in den 90er-Jahren wirklich bekannt, da Netscape (damals ein bekannter Browser) in der 2.0 Version den ersten JavaScript-Interpreter eingebaut hatte. Das ermöglichte das erste Mal in der Geschichte den Internetseiten Animationen einzubauen. Heute ist JS die meist benutzte Programmiersprache für Animationen im Internet. Auch wenn sie nicht für online-Spiele gedacht ist, ist es recht einfach mit ihr nicht allzu komplexe Spiele zu programmieren.

Eine JavaScript-Class ist eine Folie, der man verschiedene Attribute zuweisen kann.

Metadaten liefern Suchmaschinen Informationen über den Inhalt einer Website. Diese Metadaten befinden sich im Quellcode im „head“ und geben dem Browser Informationen über die Internetseite. So kann z.B. eine Metadatei die sogenannte *SERPs* beeinflussen.

SERPs (search engine results page) ist die Weise auf, die der Browser die verschiedene Internetseiten anzeigt, nachdem man ein Suchwort(-er) eingegeben hat.

Eine Programmiersprache ist eine virtuelle Sprache, die ein Computer versteht und mit der man alles animieren kann was man möchte, und die mathematischen Formeln umsetzen kann.

Pythagoras war ein sehr bekannter griechischer Philosoph (\* um 570 v. Chr. auf Samos; † nach 510 v. Chr. in Metapont in der Basilicata (<https://www.wikipedia.org/>)) aber diese Anerkennung bekam er wie er die Weltbekannte Formel  $a^2 + b^2 = c^2$  aufstellte.

Tags beinhaltet HTML Befehle. Verschiedene dieser Tags muss man öffnen und schließen, andere muss man nur öffnen. Die Tags werden mit zwei spitzen Klammern „<>“ geöffnet und auch wieder mit zwei spitzen Klammern geschlossen nur, dass man beim Schließen eines Tags auch ein Slash „</>“ benutzt wird. Der Tag, um einen Zeilenumbruch zu erzeugen, muss man z.B. nur öffnen „<br>“

Der Quelltext/Quellcode ist das Programm der Internetseite, in diesem Fall das Programm des Spieles.

## Quellen

<http://webkrauts.de/artikel/2015/eine-kurze-geschichte-von-javascript>

<https://www.wikipedia.org/>

## Bilder Nachweis

<https://www.html-seminar.de/bilder/analyse-laden-website-002-html-css.jpg> -> Seite \_ (wiki fachwörter div)

<https://www.html-seminar.de/bilder/analyse-laden-website-003-nur-html.jpg> -> Seite (HTML def)