

# Spielentwicklung

Wie man ein einfaches Spiel mit HTML5 und Javascript entwickelt

Lena, Kristin, Charlotte | 4. August 2018



LEARN TO  
CODE

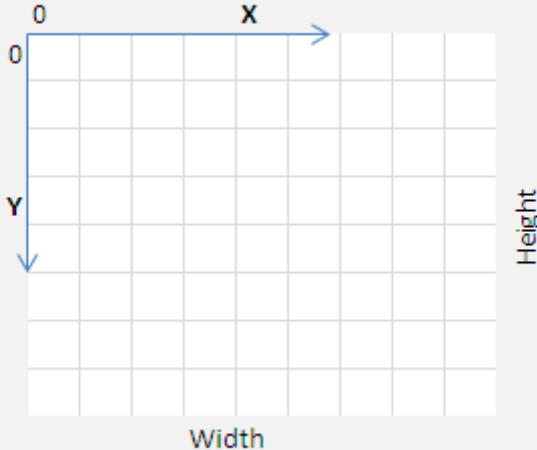
- HTML5 Element um Grafiken auf eine Webseite zu zeichnen
- Wird in HTML mit `<canvas>`-Tag eingefügt
- Ist nur ein Container
- Zeichnen erfolgt aus Javascript-code

# Canvas HTML Element einfügen

```
1 <div>
2 <canvas id="game" width="600" height="300"><
  /canvas>
3 </div>
```

```
1 <script>paintGame();</script>
```

# Das Koordinatensystem in Canvas



# Wie zeichnet man Vierecke

```
1 //um in JS auf das Canvas zuzugreifen
2 canvas = document.getElementById("game");
3 context = canvas.getContext("2d");
4
5 //gefülltes Viereck
6 context.fillStyle = color;
7 context.fillRect(x,y,breite,hoehe);
8
9 //ungefülltes mit Rand
10 context.strokeStyle = color;
11 context.strokeRect(x, y, breite, höhe);
```

- Fügt ein Canvas Element zum HTML-Dokument hinzu
- Ruft die Funktion Paint game am Ende des Bodys aus dem HTML auf
- Implementiert die Funktion paintGame()
- Fügt verschiedene Vierecke dem Canvas Element hinzu

- Komponenten, wie Vierecke bestehen eigentlich aus mehreren zusammenhängenden Informationen (Koordinaten, Breite, Höhe...)
- Also alle Informationen in einem Objekt speichern
- Objekt für gefüllte Viereck und für Text ist jeweils bereits in Javascript Datei vorhanden

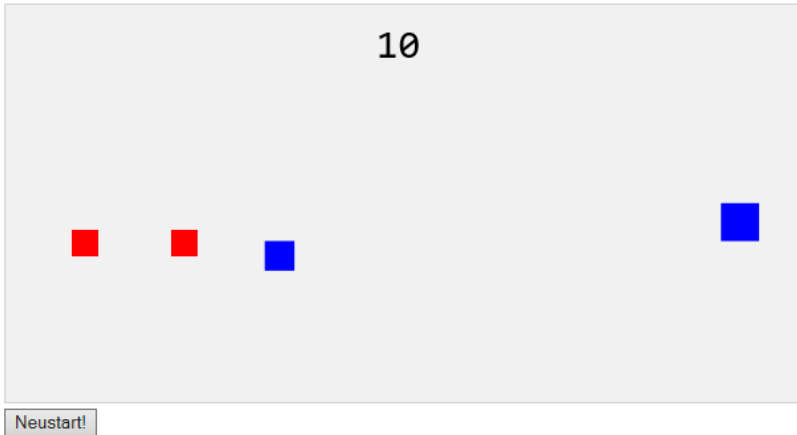
```
1 var component = new component(20, 20, "red", 50,  
    50);  
2 //width, height, color, x, y  
3 component.update();  
4 var textComponent = textComponent("30px", "  
    Consolas", "black", 280, 40, "Hallo");  
5 //testSize, font, color, x, y, text  
6 textComponent.update();
```

- Ersetzt eure zuvor gezeichneten Vierecke durch Objekte
- Fügt Text hinzu
- speichert alle Objekte in einer globalen Liste

```
1 var component = new component(20, 20, "red", 50,  
    50);  
2 //width, height, color, x, y  
3 component.update();  
4 var textComponent = textComponent("30px", "  
    Consolas", "black", 280, 40, "Hallo");  
5 //testSize, font, color, x, y, text  
6 textComponent.update();
```



# Die Idee - Was wir hier Programmieren wollen



# Was ist eine Gameloop?

- Um das Feld zu aktualisieren wird das Spielfeld nach wenigen Millisekunden neu gezeichnet
- Mit `startIntervall` kann eine Methode in einem bestimmten Zyklus immer wieder aufgerufen werden
- Zum beenden `stopIntervall()` aufrufen
- Im Zyklus wird immer die Funktion `updateGameArea()` aufgerufen  
⇒ alles was gezeichnet werden muss, soll da rein

# start, stop, neustart, clear

- Start besteht aus 2 Schritten: Werte zurück setzen und Inverall starten
- Stop muss nur stopInverall ausführen
- Neustart besteht auch aus 2 Schritten: stop und start
- Clear: leert das Canvas und wird zum Starten aufgerufen, aber auch bei jedem neuzeichnen des Spielfelds

# start, stop, neustart, clear

- Wieso habt ihr zuvor die Objekte in einer Liste gespeichert?
- Wie kann man den Spielablauf mit einer gameloop als Ablauf zeichnen?

- Spieler ist rotes Viereck mit Höhe und Breite 20px und Startpunkt bei  $x=y=50px$
- Wird als Objekt global gespeichert (redGamePiece)
- Wird beim Start angelegt und in der Loop-Funktion dann geupdatet

- Durch Aufruf der Funktion `initKeyHandling()` kann der Spieler hoch und runter bewegt werden und eine Schussfunktion aufgerufen werden

- Es wird auf zwei Events vom Browser reagiert: Taste runterdrücken und Taste loslassen
- Das Event speichert als keyCode die Nummer der betroffenen Taste
- Relevant für uns: 38 (up), 40 (down) und 32 (Leertaste)

- Bewegung wird gestartet in dem die Komponente des Spielers eine speedY gesetzt bekommt (hier +/-3)
- Bewegung wird beendet in dem die Geschwindigkeit wieder auf 0 gesetzt wird
- Beim nächsten Update der Komponente wird diese um speedY bewegt und neu gezeichnet
- Wird die Leertaste runtergedrückt wird die function shoot() aufgerufen



- Eine Komponente wird erzeugt. Wo?
- Diese wird mit den anderen Geschossen in der Liste `runningGamePieces` gespeichert
- Das Geschoss benötigt eine Geschwindigkeit in X-Richtung (positiv oder negativ?) (hier 5)
- In der Loopfunktion müssen diese so lang sie sichtbar sind immer geupdatet werden
- Wie?

- In der Funktion `updateGameArea()` (also der Loopfunktion)
- Liste der Geschosse durchgehen in einer Liste (`runningGamePieces`)
- Wenn `piece.x` über den Rand überspringen
- Ansonsten: `update()` aufrufen und in einer Liste der noch gültigen Geschosse speichern
- Am Ende Liste der `runningGamePieces` ersetzen

```
1 for(var i = 0; i<runningGamePieces.length; i++) {
2     var piece = runningGamePieces[i];
3     if(!piece) {
4         continue;
5     }
6     if(piece.x > canvasXSize) {
7         continue;
8     }
9     piece.update();
10    newRunningGamePieces.push(piece);
11 }
12 runningGamePieces = newRunningGamePieces;
```

Gegner sollen randomisiert erscheinen. Parameter die Randomisiert sein soll:

- Größe
- Geschwindigkeit
- Höhe

- Math enthält vordefinierte mathematischen Funktionen wie `Math.abs()`, `Math.floor()`, `Math.sin()`
- Mit `Math.random()` bekommt man eine zufällige Kommazahl zwischen 0 und 1 (0 inklusive, 1 nicht)
- Wie kann man mit `Math.random()` eine zufällige Zahl zwischen einem Minimum und einem Maximum machen?

```
1 function random(min, max) {  
2     return Math.random()*(max-min+1)+min;  
3 }
```

- 1. Zufällige Geschwindigkeit generieren (zwischen 0.8 und 4)
- 2. Zufällige Größe der Gegner erzeugen (z.B. zwischen 10 und 40 pixel)
- 3. Zufälligen Spawn Punkt, also die y-Koordinate, erzeugen (zwischen 0 und unterem Ende - Gegnergröße)
- 4. Gegner als neue Komponente erzeugen
- 5. Geschwindigkeit in X Richtung setzen(als Negativen Wert)
- 6. Komponente der Liste an Gegner hinzufügen

# Gegner in der Loop hinzufügen

- Basis: Liste `opponentGamePiece` mit allen Komponenten
- Ziel: alle Gegner Bewegen und neuzeichnen
- Wo & Wann: in der Loop-Funktion, also in `updateGameArea()`
- Idee: Alle Gegner durchgehen mit einer Schleife und jeweils `update()` aufrufen
- Mehr zu beachten?



# Gegner in der Loop hinzufügen

- Idee: Alle Gegner durchgehen mit einer Schleife und jeweils update() aufrufen
- Zu beachten:
- Linker Rand
- Kollisionen (was soll dann passieren?)

# Gegner in der Loop hinzufügen

```
1  for(var i = 0; i<opponentGamePieces.length; i++)  
    {  
2      var opponentPiece = opponentGamePieces[i];  
3      //überprüfe ob der Angreifer am linken Rand ist  
4      if(opponentPiece.x <= 0) {  
5          stopInterval();  
6          break;  
7      }  
8      //TODO: Kollisionen  
9      //Wenn nicht getroffen wurde:  
10     opponentPiece.update();  
11     newOpponentGamePieces.push(opponentPiece);  
12 }  
13 opponentGamePieces = newOpponentGamePieces;
```

- Implementiert als Funktion eines Objekt
- Aufruf über `obj.carshWith(otherObj)`
- Wie funktioniert das?

Erweiterung des Codes, bei dem die Gegner in der Loop hinzugefügt werden. Zwei verschiedene Kollisionen überprüfen:

- Gegner mit Spieler
- Geschoss mit Gegner

In der Schleife die die Gegner aktualisiert

```
1  if (opponentPiece.crashWith(redGamePiece)) {  
2      stopInterval();  
3      break;  
4  }
```

In der Schleife die die Gegner aktualisiert

- Idee: bestimme in der Schleife ob der Gegner mit einem Geschoss kollidiert ist
- Dazu: Pro Gegner werden jeweils alle Geschosse durchgegangen
- Bei Crash, speichern, Geschoss entfernen, score erhöhen, Gegner tauchen schneller auf und fortsetzen
- Bei Crash den Gegner nicht in die neue Gegnerliste hinzufügen

In der Schleife die die Gegner aktualisiert

```
1  var crashed = false;
2  for(var j = 0; j<runningGamePieces.length; j++) {
3      var piece = runningGamePieces[j];
4      if(piece && opponentPiece.crashWith(piece)) {
5          runningGamePieces[j] = null;
6          crashed = true;
7          score += 10;
8          minOpponentFrequency -=2;
9          continue;
10     }
11 }
12 //nur wenn nicht gecrashed wird der Gegner wieder
    in die Liste aufgenommen
```