Moderne Webtechnologien I (WT 1)
Übung: JavaScript Teil 2

WiSe 23/24

Prof. Dr. C. Köhn
Daniela Böing

26. Oktober 2023

Website:
https://www.hochschule-bochum.de/fbe/fachgebiete/institut-fuer-informatik/labor-fuer-medienkommunikation-internet-und-robotik/

Hochschule Bochum
Bochum University
of Applied Sciences

MIRO LAB

Inhaltsverzeichnis

1 JavaScript Teil 2

1.1 Einstieg und Wiederholung

In dieser Übung wird es um vertiefende Inhalte zu JavaScript gehen, darunter besonders JavaScripts Objektorientierung und das Ereignishandling. Weiterhin wird die Arbeit mit dem DOM benötigt werden, weswegen hierzu eine kleine Wiederholung folgt.

Übung 1:

1. Wie kann ein `<div>`-Container mit der ID „`ergebnis`“ (am einfachsten) abgerufen werden?

2. Wie kann erreicht werden, dass dieses Element als Textinhalt „Hallo Welt!“ anzeigt?

3. Wie kann diesem Element ein Paragraph-Element-Knoten hinzugefügt werden (zwei Zeilen)?

4. Wie kann diesem Element die Klasse „results“ zugewiesen werden?

Im Folgenden soll das Projekt der HTML- und CSS-Übungen so erweitert werden, dass tatsächlich ein Spiel gespielt werden kann. Dazu werden wir das Framework *Phaser* (Framework zur Entwicklung von Spielen in JavaScript und HTML5) und dessen Beispiel-Game *Breakout* verwenden. (Gerne können Sie auch ein anderes Beispiel von hier verwenden, aber die Einbindung und Übungen beziehen sich auf das genannte: `https://phaser.io/examples/v3/category/games`).

Übung 2:

1. Laden Sie sich die Vorgaben, also die Lösung aus der CSS-Übung sowie die JS-Datei, aus Moodle herunter.
2. Binden Sie das Phaser-Framework in der *game.html*-Datei ein, indem Sie folgenden Script-Tag ergänzen:

```
<script src="https://cdn.jsdelivr.net/npm/phaser@3.15.1/dist/
    phaser-arcade-physics.min.js"></script>
```

3. Binden Sie nun das eigentliche Spiel, also die JS-Datei, in der *game.html*-Datei ein. Entfernen Sie außerdem das Canvas-Element.

Das Spiel sollte in seiner Grundfunktionalität nun spielbar sein, soll aber im Folgenden erweitert werden.

1.2 Funktionen

Der **allgemeine Aufbau** einer Funktion in JavaScript ist:

```
function funktionsName(parameter1, parameter2, ...){
  // Anweisungsblock
}
```

Beispiele dazu:

- `x = parseInt("19abc")` liefert die Zahl 19, aber nur, wenn die Zahl am Anfang steht.
- `x = parseFloat("17.80 EUR")` liefert die Kommazahl ohne das Euro-Zeichen.
- `x = eval("12+5")` interpretiert Strings, liefert also hier 17.

1.3 Objekte

1.3.1 Allgemeines

Neben der klassischen Funktionalität, wie der Ablaufplanung des Scripts durch Schleifen oder Bedingungen, beinhaltet JavaScript vordefinierte Objekte, auf welche man lesend und schreibend zugreifen kann. Eine Übersicht über die einzelnen Objekte und ihren Zugriff finden Sie auf den Seiten von SelfHTML oder den W3Schools. Interessant sind z.B. `Array`, `String`, `Date` oder `Math`.

An dieser Stelle soll nur der generelle Zugriff auf die Objekte und deren Verwendung anhand von Beispielen erläutert werden.

Beispiele für Objektorientierung unter JavaScript sind die `window` und `document`-Klassen. Der Thin Client „Browser“ ist der zentrale Einsatzort für JavaScript. Daher wird mit dem `window`-Objekt das Browserfenster angesprochen.

Viel interessanter ist das `document`-Objekt, denn dieses repräsentiert die angezeigte Webseite und damit alle Komponenten. Zu nennen ist dann noch das `screen`-Objekt, mit dem z.B. Informationen über den Bildschirm abgefragt werden können.

Seit ECMAScript 2015 gibt es verschiedene Möglichkeiten für das objektorientierte Arbeiten mit JavaScript: die prototypbasierte, die klassenbasierte und die Object-Literal-Herangehensweise (Letztere wird hier nicht näher behandelt, da in „Reinform“ nicht sehr verbreitet).

1.3.2 Prototypen

Bei Prototypen werden Eigenschaften und Methoden in der Klasse festgelegt, die durch eine Funktion repräsentiert werden. Mittels Aufruf der Methode und dem Schlüsselwort `new` wird dann ein neues Objekt erzeugt. Ein Beispiel für eine Klasse `Farbe`:

```
1  <html><head><title>Test</title>
2  <script type="text/javascript">
3  function Farbe (R, G, B) {
4    this.R = R;
5    this.G = G;
6    this.B = B;
7    this.hex = "#";
8  }
9
10 function HintergrundWechseln () {
11   var Hintergrund = new Farbe("E0", "FF", "E0");
12   document.bgColor = Hintergrund.hex + Hintergrund.R +
13                      Hintergrund.G + Hintergrund.B;
14 }
15 </script>
16 </head>
17 <body bgcolor="#FFFFFF">
18   <h1>Das eigene Farb-Objekt mit JavaScript</h1>
19   <a href="javascript:HintergrundWechseln()">
20               Hintergrundfarbe wechseln</a>
21 </body>
22 </html>
```

Sie sehen hier, dass zunächst in der Funktion `Farbe()` eine Zuweisung der übergebenden RGB-Werte erfolgt. Im nächsten Schritt kann dann innerhalb von `HintergrundWechseln()` auf diese zugegriffen werden.

Auch mittels dieser Variante können Klassenmethoden verwendet werden, die sich jedoch wie Funktionen verhalten und so definiert werden. Dies kann beispielweise wie folgt aussehen:

```
function Farbe () { ... }

Farbe.prototype.getFarbe = function() {
  return this.hex + this.R + this.G + this.B;
}

function HintergrundWechseln () {
  var Hintergrund = new Farbe("E0", "FF", "E0");
  document.bgColor = Hintergrund.getFarbe();
}
```

Die Durchführung von Vererbung ist so nicht direkt möglich. Bei Interesse finden sich weitere Informationen z.B. hier: `https://developer.mozilla.org/de/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript`

Darüber hinaus können Sie in JavaScript auch kapseln:

```
function Abc() {
  this.a = 1;
  var b = 4;
  this.f = function() {
    alert(this.a + quadrat(b));
  }
  function quadrat(x) { return x*x; }
}
var obj = new Abc();
obj.f(); // Zugriff auf oeffentliche Methode
alert(obj.a); // Zugriff auf oeffentliche Eigenschaft
alert(obj.b); // 'undefined', da b privat
alert(obj.quadrat(4)); // nicht moeglich da Methode privat
```

1.3.3 Klassen

Neu eingeführt (mit Version 6) ist das Schlüsselwort `class` sowie der hiermit verbundene klassenbasierte Ansatz. Vor allem die Syntax bei Objekterstellung und Vererbung soll hierdurch verbessert werden, wobei jedoch der objektorientierte Ansatz von JavaScript selbst nicht verändert wurde. [1] Zusammen mit Klassen wurden auch Module eingeführt.

Mittels dieser Variante können auch wie gewohnt der Konstruktor aufgerufen und Vererbung durchgeführt werden. Ein kleines Beispiel:

```
class GeometrieObjekt {
```

[1] Quelle: `https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Klassen`

```
2     constructor (bezeichnung) {
3       this.bezeichnung = bezeichnung;
4     }
5     get bezeichnung(){
6       return this.bezeichnung;
7     }
8     set bezeichnung(bezeichnung){
9       this.bezeichnung = bezeichnung;
10    }
11  }
12  class Quadrat extends GeometrieObjekt {
13    constructor (laenge, breite){
14      super("Quadrat");
15      this.laenge = laenge;
16      this.breite = breite;
17    }
18    get flaeche(){
19      return this.berechneFlaeche();
20    }
21    berechneFlaeche(){
22      return this.laenge * this.breite;
23    }
24  }
25  var quadrat1 = new Quadrat(10, 10);
26  window.alert(quadrat1.flaeche);
```

1.4 Spiel erweitern

Erweitern Sie das Spiel nun um die Möglichkeit zum Erreichen von Punkten und dem Verlieren von Leben. Die folgenden Änderungen werden in der *breakout.js* vorgenommen.

Übung 3:

Ergänzen Sie die Klasse Breakout um die Variablen `livesLeft` und `textLives`. Fügen Sie außerdem eine globale Variable hinzu mit dem Namen `achieved_points`, die Sie mit 0 vorinitialisieren können.

Übung 4:

Bearbeiten Sie die `create`-Funktion:

1. Setzen Sie `livesLeft` auf einen festen Startwert, der aussagt, wie viele Leben einem Spieler zur Verfügung stehen (z.B. 5).
2. `textLives` soll diese verbleibenden Leben darstellen. Verwenden Sie dazu die Methoden

```
this.textLives = this.add.text(positionX, positionY, 'Vorgabetext
    ', { font: '32px Courier', fill: '#DFDFDF' })
```

und

```
this.textLives.setText(['Lives Left: ' + this.livesLeft]);
```

Danach sollten die verbleibenden Leben angezeigt, aber noch nicht verändert werden.

Übung 5:

Bearbeiten Sie die `update`-Funktion, die für jeden Frame einmal ausgeführt wird:

1. Falls die gegebene Bedingung (also `this.ball.y > 600`) erfüllt ist, ziehen Sie ein Leben ab.
2. In dem Fall müssen Sie ebenfalls den Text im Textfeld aktualisieren.
3. Sollte in dem Fall kein Leben mehr übrig sein, setzen Sie einen Text wie „Game Over“ und deaktivieren Sie den Ball mittels `this.ball.disableBody(true, true);`.
4. Falls die vorgegebene Bedingung nicht zutrifft, behalten Sie das `this.resetBall()` bei.

Nun sollten die Leben korrekt runtergezählt und das Spiel beendet werden, wenn alle Leben verbraucht sind.

Nun wenden wir uns dem Highscore zu.

Übung 6:

Bearbeiten Sie die *hitBrick*-Methode. Diese wird bei der Berührung des Balls mit einem der Steine ausgelöst. Die möglichen Werte (Eigenschaft `brick.frame.name`) sind `purple1`, `silver1`, `yellow1`, `green1`, `red1` und `blue1`.

Überlegen Sie sich ein Punktesystem, z.B. gibt `purple1` die wenigsten (einen) Punkt und `blue1` die meisten (sechs) Punkte. Prüfen Sie dann, welcher Brick getroffen wurde und summieren Sie die entsprechende Punktzahl auf die globale `achieved_points` auf.

Übung 7:

Nun muss der Highscore noch ausgegeben werden. Rufen Sie dazu das Element mit der ID `points` ab und setzen den Wert auf `achieved_points`.

Nun sollten Sie das Spiel um eine Lebensanzeige und das Hochzählen des Highscores erweitert haben. Die tatsächliche Speicherung wird später über PHP passieren.

1.5 Ereignisse und Ereignishandling

Mit JavaScript ist es auch möglich, durch ausgelöste Ereignisse Skripte auszuführen. Grundsätzlich wird dabei folgendermaßen vorgegangen: Es wird ein entsprechendes Element (HTML-Tag) ausgewählt und einem bestimmten Event dieses Elements wird eine JavaScript-Methode zugewiesen, die beim Eintreten des Ereignisses ausgeführt wird.

Es gibt verschiedene Ereignisse, die im Browser auftreten können. Die Ereignisse `load` beim Laden eines Fensters und `click` sind vermutlich die gebräuchlichsten. Eine Liste von möglichen Events kann z.B. hier eingesehen werden: `http://www.w3schools.com/jsref/dom_obj_event.asp`. Es gibt verschiedene Möglichkeiten, JS auf ein Ereignis reagieren zu lassen, wobei im Folgenden nur auf den DOM-Ereignislistener eingegangen wird.

Der DOM-Ereignislistener ist die modernste Variante, mit Ereignissen umzugehen, die auch die Möglichkeit bietet, mehrere Funktionen mit einem Ereignis zu verknüpfen. Hierbei wird der Methode `addEventListener` das Ereignis, auf das reagiert werden soll, der Funktionsname der auszuführenden Funktion (ohne runde Klammern, da Funktionsübergabe) und optional ein Parameter für den Ereignisfluss übergeben (worauf hier nicht genauer eingegangen wird).

```
1 <a href="#" id="link0">Ein Link</a>
2 <script type="text/javascript">
3   var el = document.getElementById("link0");
4   el.addEventListener('mouseover', eventMethod); //
        Ereignislistener
```

```
5   function eventMethod(){
6     alert("Die Maus ist ueber dem Link gewesen.");
7   }
8 </script>
```

Übung 8:

Erweitern Sie die Game-Website, sodass beim Klick auf den Button die Highscore-Zahl als `alert`-Box angezeigt wird.