Inhaltsverzeichnis

Vorwort **3**

Wie gehe ich mit dem Dokument um? **3**

Feedback **3**

Phaser3 - Projekt **3**

Die Idee / Das Projekt . . . 3
Erste Javascript Grundlagen . . . 3
Ausgaben und Variablen . . . 4
Rechnungen . . . 5
Blöcke . . . 6
Fallunterscheidung . . . 7
Schleifen . . . 9
Funktionen . . . 9
Die Phaser Funktionen . . . 10
Unser erstes Phaser Beispiel . . . 10
Beginn mit unserem Projekt . . . 14

Weiteres **17**

Erweiterungen **18**

Ball schneller werden lassen in x-Richtung . . . 18
Ball zurücksetzen nachdem ein Spieler gewonnen hat . . . 18
Ball schneller werden lassen in y-Richtung . . . 19
Die Geschwindigkeit des Balles zurücksetzen . . . 20

Exkurs - HTML **20**

Aufbau eines Tags . . . 21
Webseiten-Struktur . . . 21
Head und Body . . . 21
Ein paar weitere Tags . . . 22
Kommentare . . . 23
Quellcode-Strukturieren . . . 23

Exkurs CSS **23**

Selektoren . . . 23
Erste Eigenschäften ändern . . . 24

Einheiten und Werte 25
Werte 25
Maßeinheiten 25
Abstände 26
Abstand nach innen 27
Wie mache ich jetzt weiter? 27

Exkurs - HTTP und HTTPS **28**
HTTP-Verbindungsaufbau 28
HTTPS-Verbindungsaufbau 28

Exkurs - Ports **28**

Vorwort

Willkommen in der Summerschool und dem Workshop. Ich hoffe ihr hattet viel Spaß und konntet eine Menge lernen.

Wie gehe ich mit dem Dokument um?

Das Dokument dient als Dokumentation für das Projekt. Es reicht normalerweise, wenn ihr euch die relevanten Sachen raussucht und euch nur die anguckt, aber natürlich könnt ihr euch auch das komplette Dokument angucken.

Feedback

Wenn ihr Feedback oder einen Fehler zu dem Dokument gefunden habt bitte unter meiner Email (dominikrobert@gmx.de) melden und mir bescheid geben.

Phaser3 - Projekt

Hier wird es alle Informationen zu dem Phaser3 - Projekt geben.

Die Idee / Das Projekt

Die Idee hinter dem Projekt ist es ein Pong-Clon zu entwickeln. Dabei handelt es sich um ein Spiel aus den 80ern, wo ein „Ball“ zwischen zwei Spielpaddeln immer hin und her wechselt und man versuchen muss den Ball mit dem Paddle zu treffen. ## Was ist Phaser? Phaser ist ein Spieleentwicklungs-Framework für Javascript mit dem man relativ leicht eigene Spiele für den Browser entwicklen kann aufgrund der leichten vordefinierten Funktionen.

> Ein Framework ist eine Sammlung von Funktionen und Techniken, für jemand anderen um mit einem bestimmten Thema die arbeit zu erleichtern (hier z.B. Spieleentwickeln).

Erste Javascript Grundlagen

Als erstes haben wir uns mit dem Editor beschäftigt, dazu gibt es nächste Woche nochmal etwas mehr zu, bis dahin geh ich auf den nächsten Punkt ein.

Ausgaben und Variablen

Eine Ausgabe kann in Javascript auf mehrere Weisen erfolgen. Wir werden allerdings die Methode mit `console.log` nutzen. Dabei wird `console.log()` geschrieben um die Funktion aufzurufen und in den runden Klammern folgt nun die eigentliche Ausgabe. Das kann erstmal alles sein.

```
1 console.log('Ich bin eine Ausgabe');
2 console.log('Ich bin auch eine Ausgabe');
3 console.log(42);
```

Bei den obigen Ausgaben fällt auf, dass die 42 nicht mit Anführungsstrichen geschrieben wurde, dass liegt daran, dass Javascript zwischen Zahlen und Texten unterscheidet. Texte werden dabei immer mit Anführungszeichen geschrieben ‘ oder ” und Zahlen nicht. Der unterschiedt liegt in der Mathematik, denn mit Zahlen kann gerechnet werden und mit Text eben nicht. So ist es möglich auf eine Zahl alle Operationen (+,-,*,/) anzuwenden und auf einem Text ausschließlich das +, welches dann zwei Texte aneinander packt.

Eine Variable hält immer einen Wert (bei uns z.B. eine Zahl oder einen Text) um mit diesen zu arbeiten. Eine Variable kann wie folgt angelegt werden

```
1 let meineVariable = 17;
2 let nochEineVariable = 'Mein Text';
```

> Erstellung zweier Variablen mit einer Nummer und einem Text.

Danach kann mit der Variable umgegangen werden als ob es ein Text oder eine Ziffer wäre:

```
1 let meineVariable = 17;
2 let nochEineVariable = 'Mein Text';
3
4 console.log(meineVariable);
5 console.log(nochEineVariable);
```

Eine andere Möglichkeit eine Variable zu definieren und mehrere Werte beieinander zu halten ist ein Objekt. Ein Objekt kann mehrere Werte beinhalten. So kann z.B. die Konfiguration für ein Spiel in einem Objekt festgehalten werden. Ein Objekt wird erstmal wie eine Variable behandelt und umfasst geschweifte Klammern `{}`. Innerhalb dieser Klammern können nun die Felder reingeschrieben werden die in dem Objekt zur Verfügung stehen. Dabei wird ein Feldname immer mit einem Doppelpunkt von dem Wert getrennt und mit einem Komma zum nächsten geleitet.

```
1 let meinObjekt = {
2   name: 'Hans Dieter',
3   alter: 17,
4   groesse: 200,
5   istSchueler: true
```

```
6  }
7
8  // der Zugriff erfolgt über die Variable mit einem Punkt und dem
       Feldnamen:
9  console.log(meinObjekt.name);
10 console.log(meinObjekt.alter);
11 console.log(meinObjekt.groesse);
12 console.log(meinObjekt.istSchueler);
```

Rechnungen

Um mit einer Variable oder einem Wert zu rechnen kann man einfach das Operationszeichen schreiben:

```
1  let meineVariable = 17 + 3;
2  let nochEineVariable = 'Mein Text';
3
4  meineVariable = 28 / 2 + 5;
5  nochEineVariable = 'Hallo' + ' Welt';
6  console.log(meineVariable);
7  console.log(nochEineVariable);
```

> Wichtig: Nur bei der ersten Verwendung einer Variablen muss das Wort let davor geschrieben werden, danach darf es nicht mehr. Wichtig: Die Werte der Variablen werden dabei immer überschrieben und sind somit nicht mehr nutzbar. Um mit einer Variablen weiterzurechnen kann folgendes gemacht werden:

```
1  let test = 17;
2  test = test + 5;
```

> Hier wird eine Variable erzeugt mit dem Namen test der der Wert 17 zugeordnet wird. In der nächsten Zeile wird ihr ein neuer Wert zugewiesen. Dafür wird wieder die Variable benutzt um den Wert von ihr mit 5 zu addieren und der Variablen erneut zuzuweisen. ### Kommentare Kommentare werden in Javascript ignoriert (nicht ausgeführt) und dienen der Übersicht für den Programmierer. Kommentare werden mit einem `//` eingeleitet um einen Kommentar in einer Zeile zu haben oder aber über mehrere Zeilen sieht es dann wie folgt aus

```
1  // Ich bin ein Einzeiliger-Kommentar
2
3  /*
4    Ich gehe
5    über
6    mehrere
7    Zeilen
```

```
8 */
```

Blöcke

Blöcke werden in Javascript mit den geschweiften Klammern beschrieben `{}`. Diese Blöcke bilden eine Einheit und sorgen in Verbindung mit Schlüsselwörtern dazu, dass alles ausgeführt wird was in dem Block steht => z.B. bei späteren Funktionen oder Fallunterscheidungen. Diese Blöcke haben auch die Besonderheit, dass Variablen immer nur in dem Block gültig sind, in dem Sie erstellt wurden und in den Blöcken, die innerhalb dieses Blockes erstellt wurden.

Als Beispiel:

```
{
  let meineVariable = 5;

  // Hier natürlich noch gültig
  console.log(meineVariable);

  {
    // hier auch, da es ein innerer Block ist
    console.log(meineVariable);
  }
}
// Fehler: meineVariable ist nicht bekannt
console.log(meineVariable);
```

Eine Ausnahme bildet hier das Schlüsselwort `var`. Wenn ihr damit eine Variable erstellt anstatt mit `let` ist diese Variable `global` gültig.

```
{
  let meineVariable1 = 5;
  // meineVariable2 ist jetzt in jedem Block gültig und steht global
     zur Verfügung (d.h. auch in jeder Funktion usw.)
  var meineVariable2 = 10;

  // Hier natürlich noch gültig
  console.log(meineVariable1);

  console.log(meineVariable2);
  {
    // hier auch, da es ein innerer Block ist
    console.log(meineVariable1);
  console.log(meineVariable2);
  }
}
// Fehler: meineVariable ist nicht bekannt
console.log(meineVariable1);
```

```
18 console.log(meineVariable2);
```

Fallunterscheidung

Eine Fallunterscheidung dient dazu zwischen zwei Zuständen zu unterscheiden und verschiedenen Code für die Zustände auszuführen. Als Beispiel möchte man z.B. zusätzlichen Code ausführen wenn jemand über 18 ist (z.B. Zugang zu einer Webseite). Auch in der realen Welt gibt es natürlich auch Fallunterscheidungen die wir intuitiv fällen. So gibt es z.B. die Möglichkeit sich wenn es kalt ist eine Jacke anzuziehen.

In der Programmierung ist es allerdings nicht ganz so einfach mit den Bedingungen. Hier muss jede Bedingung nach WAHR oder FALSCH beurteilt werden können. Diese Werte WAHR oder FALSCH werden auch als boolsche Werte bezeichnet. Für diese Aussagen (boolsche Bedingung => Eine Bedingung die nach einer boolschen Wert auswertbar ist) stehen uns folgende Operatoren zur Verfügung:

Operator	Wort	Beispiel
>	größer	5 > 10 5 ist kleiner als 10 (WAHR)
<	kleiner	10 < 5 10 ist kleiner als 5 (FALSCH)
>=	größer gleich	16 >= 17 16 ist größer oder gleich 17 (FALSCH)
<=	kleiner gleich	16 <= 16 16 ist kleiner oder gleich 16 (WAHR)
!=	nicht gleich	0 != 1 0 ist nicht gleich 1 (WAHR)
==	gleich	1 == 1 1 ist gleich 1 (WAHR)
===	typengleich	1 === „1“ Der Typ und der Wert von 1 ist gleich dem Typ und der Wert von 1 (FALSCH => da links eine Zahl und rechts ein Text)

Mithilfe dieser Operatoren lassen sich nun die Bedingungen schreiben. Um so eine Bedingung zu schreiben, müssen wir vorher das Schlüsselwort für eine Fallunterscheidung voran schreiben `if` um in

Klammern dahinter die Bedingung zu schreiben. Sollte nun die Bedingung wahr sein, wird alle was in den geschweiften Klammern dahinter kommt ausgeführt. Wenn man möchte kann man hinter der `if` noch ein `else` schreiben um einen block nur Auszuführen, wenn die Bedingung in der `if` Anweisung den Wert FALSCH liefert.

Als Beispiele:

```
// definiere eine Variable die mein Alter speichert
let meinAlter = 23;

if (meinAlter >= 18) {
  console.log('Du bist volljährig');
}
```

> Programm welches das Alter speichert und „Du bist volljährig“ auf der Konsole schreibt wenn man älter oder gleich 18 Jahre alt ist.

```
let meinAlter = 23;

if (meinAlter >= 18) {
  console.log('Du bist volljährig');
} else {
  console.log('Du bist noch minderjährig');
}
```

> Programm welches das Alter speichert und „Du bist volljährig“ auf der Konsole schreibt wenn man älter oder gleich 18 Jahre alt ist. Sollte man jedoch jünger als 18 Jahre alt sein wird „Du bist noch minderjährig“ auf die Konsole geschrieben

```
let meinAlter = 23;

if (meinAlter >= 18) {
  console.log('Du bist volljährig');
} else if (meinAlter >= 16) {
  console.log('noch nicht volljährig, aber nah dran');
} else {
  console.log('Du bist noch minderjährig');
}
```

> Dieses Programm gibt ein „Du bist volljährig“ wenn der Benutzer älter oder gleich 18 Jahre alt ist. Zudem wird noch ein anderer Text ausgebenen wenn das alter größer oder gleich 16 ist. Als alternative wenn nichts anderes zutrifft wird „Du bist noch minderjährig“ ausgegeben. In dem Beispiel wird das **`if else`**-Konstrukt verwendet, welches nur die erste Bedingung ausführt. Das ist hier besonders Sinnvoll, da ansonsten immer beide Texte ausgebeben werden würden (da jemand der über 18 Jahre ist auch immer über 16 Jahre alt ist) und wir mit dem **`if else`**-Konstrukt nun

> nach der ersten Bedingung den rest des Konstrukts überspringen können, sollte jemand über oder gleich 18 Jahre alt sein.

Schleifen

Schleifen dienen in der Programmierung dazu Aufgaben zu bewältigen die öfters getan werden müssen. So ist bei einem Spiel immer so eine Schleife dabei die z.B. unseren Bildschirm immer neu lädt um so z.B. Dinge bewegen zu können. So kann man mit dem Schlüsselwort `while` so eine Schleife einleiten und dann in runden Klammern dahinter sofort die Bedingung definieren.

```
1 while(x < 1000) {
2   // Wird solange ausgeführt wie die Bedingung (in den Klammern) wahr
        ist.
3 }
```

Also um ein kleines Beispiel zu machen, wollen wir nun die Zahlen von 1 bis 1000 ausgeben. Mit der alten weise müssten wir leider 1000x `console.log()` schreiben. Wir wollen das aber natürlich etwas vereinfachen und nehmen daher eine Schleife.

```
1  let x = 1; // wir wollen bei 1 anfangen
2
3  while(x <= 1000) { // solange x kleiner oder gleich ist als 1000 gehen
       wir in die Schleife
4    // wird solange ausgeführt wie die Variable x kleiner oder gleich
        1000 ist.
5    // Ausgabe der Variablen x
6    console.log(x);
7
8    // erhöhe x um eins um die Schleife auch irgendwann beenden zu können
        .
9    x = x + 1;
10 }
```

Funktionen

Mittels Funktionen können Codeteile ausgelagert werden und zentral an einer Stelle zur Verfügung gestellt werden. Das ist von Vorteil, wenn man Code in einem Programm oft verwendet und daher den Code nur an einer Stelle updaten möchte. Desweiteren ist es mit Funktionen einfacher funktionalitäten zu teilen (wie es z.B. Phaser3 macht bzw. alle anderen Frameworks). Um in Javascript eine Funktion zu erstellen, wird das Schlüsselwort `function` vor dem Namen der Funktion geschrieben. Danach kommen Klammern `()`. In diesen Klammern können nun Parameter definiert werden um bestimmte Variablen auch innerhalb der Funktion zur Verfügung zu haben. Die Funktion kann nach der definition aufgerufen werden, indem man den Namen mit den beiden Klammern schreibt.

```
1  // definition der Funktion
2  function meineFunktion() {
3    console.log('Ich bin eine Funktion');
4  }
5
6  // Aufruf der Funktion.
7  meineFunktion();
8
9  // definition der Funktion mit einem Parameter
10 function meineFunktion2(meinParameter) {
11   console.log(meinParameter);
12 }
13
14 // Aufruf der Funktion mit Parameter
15 // Hier wird in den Klammern der Wert eingetragen und kann in der
       Funktion benutzt werden
16 meineFunktion('hello');
```

Die Phaser Funktionen

Phaser stellt mehrere Funktionen zur Verfügung, die verschiedene Aufgaben haben:

- preload
  - Die Preload-Funktion wird ausgeführt beim starten von Phaser. Diese Funktion kann benutzt werden um z.B. Bilder und Sounds zu laden.
- create
  - Ist die 2. Funktion die Ausgeführt wird. Hier werden normalerweise die Bilder platziert und an die richtige Position mit der richtigen größe geschoben
- update
  - Die update-Funktion wird als letztes von unseren hier genannten Funktionen aufgerufen und wird immer so oft es geht ausgeführt.

Unser erstes Phaser Beispiel

Unser erstes Beispiel erläuter ich mal mit den Zeilenangaben und dem Aufbau des Codes in kleineren Schritten.

Zuerst, da es sich dabei um ein Spiel handelt welches im Browser dargestellt wird und der Browser üblicherweise mit HTML Dateien arbeitet definieren wir das Grundgerüst für eine HTML Datei:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4   </head>
5   <body>
6   </body>
7 </html>
```

In Zeile 1 definieren wir also, dass es sich dabei um eine HTML-Datei handelt. Nun wird alles was zu der HTMl-Datei gehört in den `<html>` Tag geschrieben (für weitere Erläuterungen => siehe Exkurs HTML). Nun können wir mittels eines CDN das Phaser-Framework mit den Funktionen die wir brauchen hinzufügen.

> Ein CDN oder auch Content Delivery Network stellt Inhalte über das Netzwerk (Internet) zur Verfügung um mal schnell etwas zu testen oder Inhalte zentral verwalten zu können. In unserem Fall können wir es verwenden um mit Phaser schnell anfangen zu können ohne vorher die Phaser-Dateien downloaden zu müssen.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <script src="https://cdn.jsdelivr.net/npm/phaser@3.55.2/dist/phaser
          -arcade-physics.min.js"></script>
5   </head>
6   <body>
7   </body>
8 </html>
```

Nun können wir in den Body-Tag unser Javascript für Phaser schreiben. Da HTML eigentlich kein Javascript unterstützt, müssen wir den Code dafür innerhalb eines Script-Tags schreiben:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <script src="https://cdn.jsdelivr.net/npm/phaser@3.55.2/dist/phaser
          -arcade-physics.min.js"></script>
5   </head>
6   <body>
7     <script>
8     </script>
9   </body>
10 </html>
```

> Ab jetzt werde ich auch ausschließlich den Code aus dem Script-Tag schreiben und nur zum Schluss noch einmal den vollständigen Code zeigen.

Zuerst können wir uns ein Konfigurationsobjekt erstellen, in dem festgelegt wird die groß unser Spiel

z.B. ist (die Angabe erfolgt in der Einheit Pixel). Hier könnten wir auch direkt eine Hintergrundfarbe für das Spiel festlegen.

```
var config = {
  width: 800,
  height: 600,
  backgroundColor: '#4488aa',
}
```

Als nächstes muss in dieses Objekt auch die Funktionen die wir von Phaser benutzen wollen mit rein. Dafür definieren wir das Attribut `scene` innerhalb unserer Konfiguration und schreiben die Funktionen rein die wir benutzen möchten mit der Funktion wie Sie bei uns heißt.

```
var config = {
  width: 800,
  height: 600,
  backgroundColor: '#4488aa',
  scene: {
    preload: preload,
    create: create
  }
}
```

Wir können jetzt auch sofort diese Funktionen defnieren.

```
var config = {
  width: 800,
  height: 600,
  backgroundColor: '#4488aa',
  scene: {
    preload: preload,
    create: create
  }
};

function preload() {
}

function create () {
}
```

> Hier haben wir die Funktions-Namen nicht geändert. Dazu gibt es meistens auch keinen Grund und der Standardname kann einfach beibehalten werden.

Als nächstes müssen wir unser Spiel erstellen. Das geht ganz einfach mittels

```
var game = new Phaser.Game(config);
```

Damit legen wir eine neue Variable an, die alle Informationen und Daten für das Spiel enthält. Dazu

benutzen wir eine Funktion die von Phaser bereitgestellt wird `Game` und übergeben dieser Funktion unser Konfigurationsobjekt als Parameter.

```
var config = {
  width: 800,
  height: 600,
  backgroundColor: '#4488aa',
  scene: {
    preload: preload,
    create: create
  }
}

var game = new Phaser.Game(config);

function preload() {
}

function create () {
}
```

> Unser Spiel würd nun schon funktionieren und die Funktionen ausführen. Das können wir auch testen, indem wir in diese Funktionen ein `console.log`schreiben.

Wir fangen nun an, ein Bild hinzuzufügen.

```
var config = {
  width: 800,
  height: 600,
  backgroundColor: '#4488aa',
  scene: {
    preload: preload,
    create: create
  }
}

var game = new Phaser.Game(config);

function preload() {
  this.load.image('sky', 'assets/bild1.png');
}

function create () {
  this.add.image(500, 200, 'sky');
}
```

> Mittels **`this`** kann auf Werte und Funktionen eines Objekts zugegriffen werden. Dieses Objekt stellt die Funktion `image` zur Verfügung an die wir mittels eines weiteren Objekts `load` kommen.

> Damit können wir ein Bild laden und ihn einen Namen geben (erster Parameter ist der Name und der zweite ist der Pfad zum Bild).

> In der Create Funktion wird das Bild nun dargestellt mittels dem Objekt `add` und der Funktion `image`. Da geben wir die Position auf der X-Achse und der Y-Achse an um zum Schluss den Namen des Bildes angeben zu können. Die Position 0/0 ist in Phaser in der linken oberen Ecke.

Unser kompletter Code sieht also nun wie folgt aus:

```
<!DOCTYPE html>
<html>

  <head>
    <script src="https://cdn.jsdelivr.net/npm/phaser@3.55.2/dist/phaser
        -arcade-physics.min.js"></script>
  </head>

  <body>
    <script>
      var config = {
        width: 800,
        height: 600,
        backgroundColor: '#4488aa',
        scene: {
          preload: preload,
          create: create
        }
      };

      var game = new Phaser.Game(config);

      function preload() {
        this.load.image('sky', 'assets/bild1.png');
      }

      function create () {
        this.add.image(500, 200, 'sky');
      }
    </script>
  </body>
</html>
```

Beginn mit unserem Projekt

Für unser Projekt können wir wie oben anfangen, können nur die Preload Funktion weglassen, da wir keinerlei Bilder benötigen.

Wir überlegen uns welche Variablen wir brauchen und können diese erstmal Oben global anlegen um diese in allen Phaser-Funktionen verwenden zu können.

```
var paddleLeft; // Linkes Paddle
var paddleRight; // Rechtes Paddle
var ball; // Ball
var ballX = 1; // Geschwindigkeit in X
var ballY = 1; // Geschwindigkeit in Y
```

Nun können wir die Create Funktion erstellen:

```
function create() {

}
```

> der nun nachfolgende Code wird wenn nichts dabei steht, in die Create-Funktion geschrieben.

Innerhalb dieser Funktion können wir nun den Ball und die beiden Paddles erstellen und platzieren.

```
// fügt die Rechtecke auf dem Bildschirm hinzu
paddleLeft = this.add.rectangle(50, 350, 25, 100, 0xffffff)
paddleRight = this.add.rectangle(950, 350, 25, 100, 0xffffff)
ball = this.add.rectangle(500, 300, 25, 25, 0xffffff)
```

> Erstellung dreier Rechtecke mittels der add.rectangle Funktion. Diese bekommt als erstes die Position für X, dann Y und anschließend die größe für X und Y mit. Der letzte Parameter setzt die Farbe.

Als nächstes müssen wir für unsere Objekte (Rechtecke) die Physik auch aktivieren und können gleichzeitig darunter auch die Kollision für die Welt mitgeben. Die Welt ist quasi das Fenster und die Kollision damit würde uns hindern, die Paddles außerhalb des Fensters zu schieben.

```
// Erstellt für jeden vorhandenen Körper die Physik
this.physics.add.existing(paddleLeft, false);
this.physics.add.existing(paddleRight, false);
this.physics.add.existing(ball, false);

// Legt fest, dass die einzelnen Rechtecke nicht außerhalb des
   Bildschirms fliegen dürfen
paddleLeft.body.setCollideWorldBounds(true);
paddleRight.body.setCollideWorldBounds(true);
ball.body.setCollideWorldBounds(true);

// aktiviert die Funktion um die Berührung des Bildschirms zu überprü
   fen und entsprechend zu reagieren
ball.body.onWorldBounds = true;
```

> Aktiviert die Physik und die Kollision mit der Welt

Nun können wir festlegen, was genau passieren soll, wenn die Objekte miteinander Kollidieren:

```
// fügt einen Collider (prüft die Kollision zwischen zwei Objekten) für den Rechten Schläger und dem Ball hinzu
this.physics.add.collider(
  paddleRight,
  ball,
  function (paddle, ball) {
    ballX *= -1;
  }
)

// fügt einen Collider (prüft die Kollision zwischen zwei Objekten) für den Linken Schläger und dem Ball hinzu
this.physics.add.collider(
  paddleLeft,
  ball,
  function (paddle, ball) {
    ballX *= -1;
  }
)

// fügt einen Collider (prüft die Kollision zwischen zwei Objekten) für den Ball und dem oberen und unteren Bildschirmende hinzu
this.physics.world.on('worldbounds', function (body, up, down, left, right) {
  if (up || down) { // kann auch einfach nur up || down sein, da es ja schon true ist. Das || ist das oder Zeichen
    ballY *= -1; // *= ist gleichbedeutend mit ballY = ballY * -1. Das *= ist also ein Zugriff auf die Variable selbst mit dem Wert der dahinter steht

  }
})
```

> Was passiert mit bei einer Kollision? Dafür können wir eine neue Kollision hinzufügen für den Linken und den Rechten Paddle mit dem Ball und zum Schluss für die Welt. Dabei ändern wir aber lediglich immer die Koordinaten für unsere globalen Variablen.

Wir wären nun mit der Create-Funktion durch und widmen uns nun der Update Funktion.

```
function update() {

}
```

> der nun nachfolgende Code wird wenn nichts dabei steht, in die update-Funktion geschrieben.

Als erstes wollen wir uns die Tastatureingaben holen, dafür stellt Phaser uns eine Funktion bereit in der wir sowas Abfragen können und uns die Tasten geben lassen welche wir benötigen.

```
let cursors = this.input.keyboard.createCursorKeys();
let wasd = this.input.keyboard.addKeys('W,S,A,D');
```

> Stellt uns Objekte zur Verfügung welche wir Anschließend wieder benutzen können um die Eingaben auf der Tastatur zu erfassen

Nun können wir die Eingaben abfangen:

```
if (cursors.down.isDown) {
  paddleRight.y += 10;
} else if (cursors.up.isDown) {
  paddleRight.y -= 10;
}

if (wasd.S.isDown) {
  paddleLeft.y += 10;
} else if (wasd.W.isDown) {
  paddleLeft.y -= 10;
}
```

> Bewege die Paddles bei der Richtigen Tastatureingabe. Das Linke Paddle wird mit WASD und das Rechte mit den Pfeiltasten gesteuer

Nun fehlt nur noch die Bewegung des Balls. Diese können wir ebenfalls mit den globalen Variablen steuern und bewegen den Ball einfach mit der jeweiligen Geschwindigkeit pro Zug weiter.

```
ball.x += ballX;
ball.y += ballY;
```

Weiteres

- Wenn ihr weiter machen möchtet empfehle ich euch den Talentkolleg Kurs für Informatik wo immer spannende Themen rund um die Programmierung und der Informatik abgehalten werden.
- Die Phaser Bibliothek mit den Beispielen:
  - https://phaser.io/
  - https://phaser.io/learn

Außerdem gibt es den Kompletten Code + diese Dokumentation auf Github zum nachlesen. https://github.com/Talentkolleg-Herne/summerschool-2021-informatik-phaser

Erweiterungen

Hier fasse ich einmal die Erweiterungen zusammen, die wir nachträglich eingebaut haben.

Ball schneller werden lassen in x-Richtung

Um den Ball schneller werden lassen zu können bei jeder Berührung müssen wir drauf achten aus welcher Richtung wir kommen und können dann jenachdem welchen Wert ballX hat die Variable eröhen (wir müssen aufpassen, da der Ball von Rechts kommt der Wert Minus ist).

Damit sehen unsere beiden Kollider nun wie folgt aus:

```
1  // fügt einen Collider (prüft die Kollision zwischen zwei Objekten) für
       den Rechten Schläger und dem Ball hinzu
2    this.physics.add.collider(
3      paddleRight,
4      ball,
5      function (paddle, ball) {
6        ballX += 1; // erhöhe ballX um eins, da wir hier von links kommen
             und daher der Wert minus ist.
7        ballX *= -1;
8      }
9    )
10
11   // fügt einen Collider (prüft die Kollision zwischen zwei Objekten) f
       ür den Linken Schläger und dem Ball hinzu
12   this.physics.add.collider(
13     paddleLeft,
14     ball,
15     function (paddle, ball) {
16       ballX -= 1; // vermindere ballX um eins, da wir hier von rechts
           kommen und daher der Wert minus ist.
17       ballX *= -1;
18     }
19   )
```

Ball zurücksetzen nachdem ein Spieler gewonnen hat

Um den Ball zurück setzen zu können, muss bei einer Rechten- oder Linken-Rand-Kollision der x und y Wert des Balles wieder auf den Ursprung gesetzt werden.

```
// fügt einen Collider (prüft die Kollision zwischen zwei Objekten) für den Ball und dem oberen und unteren Bildschirmende hinzu
this.physics.world.on('worldbounds', function (body, up, down, left, right) {
  if (up === true || down === true) { // kann auch einfach nur up || down sein, da es ja schon true ist. Das || ist das oder Zeichen
    ballY *= -1; // *= ist gleichbedeutend mit ballY = ballY * -1. Das *= ist also ein Zugriff auf die Variable selbst mit dem Wert der dahinter steht
  }
  // prüft
  if (left || right) {
    ball.x = 500;
    ball.y = 300;
  }
})
```

Ball schneller werden lassen in y-Richtung

Möchte man den Ball auch in die y-Richtung schneller werden lassen, benötigt man hierfür eine neue Variable. Man hat hier nämlich das Problem, dass es wie bei der X-Richtung einmal positiv und einmal negativ sein kann, man allerdings im Unterschied zu X nicht weiß ob man eine negative Zahl hat oder eine positive. Daher kann man das vorher abfragen und anschließend richtig erhöhen.

```
// fügt einen Collider (prüft die Kollision zwischen zwei Objekten) für den Rechten Schläger und dem Ball hinzu
  this.physics.add.collider(
    paddleRight,
    ball,
    function (paddle, ball) {
      ballX += 1;
      if (ballY > 0) {
        ballY += 1;
      } else {
        ballY -= 1;
      }
      ballX *= -1;
    }
  )

  // fügt einen Collider (prüft die Kollision zwischen zwei Objekten) für den Linken Schläger und dem Ball hinzu
  this.physics.add.collider(
    paddleLeft,
    ball,
    function (paddle, ball) {
      ballX -= 1;
```

```
22        if (ballY > 0) {
23          ballY += 1;
24        } else {
25          ballY -= 1;
26        }
27        ballX *= -1;
28      }
29    )
```

Die Geschwindigkeit des Balles zurücksetzen

Da man die Geschwindigkeit des Balles immer erhöht und dieser Wert auch gespeichert bleibt, muss bei einem Punkt der Wert wieder zurück gesetzt werden.

```
1   // fügt einen Collider (prüft die Kollision zwischen zwei Objekten) fü
       r den Ball und dem oberen und unteren Bildschirmende hinzu
2    this.physics.world.on('worldbounds', function (body, up, down, left,
        right) {
3      if (up === true || down === true) { // kann auch einfach nur up ||
          down sein, da es ja schon true ist. Das || ist das oder Zeichen
4        ballY *= -1; // *= ist gleichbedeutend mit ballY = ballY * -1.
            Das *= ist also ein Zugriff auf die Variable selbst mit dem
            Wert der dahinter steht
5      }
6      if (left || right) {
7        ball.x = 500;
8        ball.y = 300;
9
10       ballX = 1;
11       ballY = 1;
12     }
13   })
```

Exkurs - HTML

HTML (Hypertext Markup Language) wird benutzt um eine Webseite zu beschreiben. Das heißt mittels HTML wird der Aufbau einer Webseite beschrieben und nicht deren eigentlichen Design. Das Design wird anschließend in CSS gemacht. In der HTML gibt es sogenannte Tags, die für verschiedene Strukturen stehen und von dem Englischen Begriff abgeleitet werden. Ein Tag wird mit den folgenden Klammern eingeleitet <> und meistens wie folt beendet </>. Der Tagname steht zwischen den Klammern bzw. zwischen dem Schrägstrich und der schließenden Klammer. Hier sind ein paar Beispiele:

- `<html></html>`
- `<p></p>`

- <head></head>
- <body></body>

Man kann diese Tags auch als Container oder Behälter betrachten, da diese auch Daten beinhalten können. So wird ein Paragraphen-Tag p am Anfang eines Paragraphen geschrieben um diesen auch mit dem schließenden Tag zu beenden.

```
<p>Ich bin ein neuer Paragraph</p>
<p>Paragraphen können auch
mehrzeilig sein</p>
```

Aufbau eines Tags

```
                            +--- HTML-Attribut
                            |   +--- Wert in Hochkommas
                            |   |
<p>ein Text mit einem <a href="…">Link</a></p>
 |                    |                |  |
 |                    |                |  +--- schließendes
 |                    |                |       HTML-Tag mit
 |                    |                |       Schrägstrich
 |                    |                |
 |                    |                +--- schließendes Tag
 |                    +--- öffnendes Tag
 +--- öffnendes HTML-Tag
```

Webseiten-Struktur

Als ein Beispiel wird hier ein sehr simple Webseite herhalten.

```
<html>
  <head>
    <title>Meine Webseite</title>
  </head>
  <body>
    Lorem ipsum
  </body>
</html>
```

Head und Body

Eine HTML-Seite besteht immer aus einem <html>-Bereich, indem der eigentliche Seitenaufbau beschrieben wird. Darin wird nun ein <head> und einen <body>-Bereich definiert. Innerhalb des <head

> kommen nun Metadaten wie z.B. der Author, das Zeichenformat und dort kann man den Titel der Seite beschreiben. In den <body> Bereich kommt nun alles, was für den Benutzer sichtbar wird rein.

Ein paar weitere Tags

Um ein paar weitere Tags und einen ersten einfachen Aufbau zu bekommen schreiben wir hier folgende Zeilen, die ich anschließend erläutern werde. Dabei schreibe ich hier nur den <body>:

```
<body>
   <h1>Lorem ipsum </h1>
   <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. </p>
   <p>Mauris ante neque, vehicula quis, convallis sit amet,
      vestibulum ornare, arcu. </p>
   <p>Donec at lorem et elit congue dictum. Cras sapien ligula,
      rutrum quis, posuere id, faucibus id, risus. Maecenas sed est
      volutpat arcu adipiscing tempus. Sed dictum mauris euismod
      mauris.</p>
</body>
```

- <h1>
  - Eine h1 ist eine Überschrift1 (engl. Headline1). Die Überschriften gehen von 1 (die größte) bis 6 (die kleinste). Nach einer Überschrift folgt auch immer ein Zeilenumbruch.
- <p>
  - Ein p ist ein Paragraph. Ein Paragraph enthält meistens Text und anschließend ein Zeilenumbruch.
- <ul, ol>
  - erzeugen ungeordnete bzw. geordnete (nummerierte) Listen. Ungeordnete Listen werden mit einem führenden Listenpunkt dargestellt, geordnete Listen mit einem Index. In beiden Listen wird jedes Listenelement in li-Tags eingeschlossen.

```
<ol>
  <li>HTML strukturiert den Inhalt </li>
</ol>
```

-
  - In einem DIV können andere Elemente hineingefügt werden. Das div fungiert nur als eine Art Container.

Kommentare

Kommentare werden in HTML wie folgt beschrieben:

```
<!-- Dies ist ein Kommentar -->
```

> Kommentare dienen dazu den Quelltext besser zu Strukturen und für den Leser annehmlicher zu gestalten.

Quellcode-Strukturieren

Es ist üblich den Quellcode mit bestimmten Einrückungen besser lesbar zu machen. Dazu gehört z.B. dass verschachtelte Tags meistens in einer neuen Zeile (außer z.B. Links oder andere kurze Tags) und mit einem vorangegangenen Tab-Zeichen eingerückt werden.

Beispiel hier mit einer Tabelle:

```
<table>
   <tr>
      <td>Zeile 1 Spalte 1</td>
      <td>Zeile 1 Spalte 2</td>
   </tr>
   <tr>
      <td>Zeile 2 Spalte 1</td>
      <td>Zeile 2 Spalte 2</td>
   </tr>
</table>
```

> Wie man sieht sidn die `<tr>`und die `<td>` jeweils in einer neuen Zeile und weiter eingerückt als z.B. das `<table>`

Exkurs CSS

Mittels CSS (Cascade Style Sheets) wird die HTMl-Seite ansehnlicher gemacht und letzlich wird hier das Design festgelegt. Dafür kann man bestimmte HTML-Tags oder aber alle Tags ansprechen und mit den CSS-Eigenschaften ein bestimmtes styling zuweisen.

Selektoren

Mithilfe dieser Selektoren werden die Tags ausgewählt, wo man gerne das aussehen verändern möchte. Ein Selektor kann einfach der gewählte Tag sein:

```
div {}
```

> dieser Selektor wählt alle vorhandenen divs aus und würde die Regeln in den geschweiften Klammern `{}` auf diese divs anwenden.

Zwei weitere Selektoren sind der Klassen-Selector und der ID-Selector. Diese Selektoren müssen allerdings vorher im HTML versehen werden:

```
<body>
  <div id="inhalt">
    <p class="meinAbsatz">Mein erster Absatz</p>
  </div>
</body>
```

> Unser HTML mit einer ID und einer class (Klasse)

Um nun die ID anzusprechen benutzen wir den Namen der id mit einer Raute # voran und für die Klasse ein `.`.

```
#inhalt {

}

.meinAbsatz {

}
```

Erste Eigenschäften ändern

Um nun mal auf die verschiedenen Eigenschaften einzugehen würde ich gerne dieses Beispiel zeigen:

```
p {
  color: red;
  width: 500px;
  border: 1px solid black;
}
```

Diese Selektor addressiert erstmal alle Paragraphen in unserem HTML. Anschließend wird die Textfarbe mittels `color` auf rot gesetzt, die breite des PAragraphen wird auf 500px begrenzt und zum Schluss erhält er noch ein Rahmen mit 1px dicke in Schwarz mittels `border: 1px solid black`.

Einheiten und Werte

Werte

In CSS ist erstmal jede numerische Zahl gestattet (jede Rationale-Zahl ist in CSS ein geeigneter Wert). Dabei wird nur in Ganzzahl und Fließkommazahl unterschieden, wobei eine Fließkommazahl mit einem Punkt getrennt wird. Eine 0 wird vor einem Komma ist immer optional. Hier sind ein paar Beispiele für richtige Werte:

- 1
- -5
- .5
- 192
- -50
- 49.5
- 10.09

Maßeinheiten

Ein Maß (Maßeinheit) wird definiert, indem direkt nach dem Wert ohne ein Leerzeichen die Maßeinheit geschrieben wird. Hierbei gibt es viele unterschiedliche Arten von Maßeinheiten. Wir wollen einmal die zwei häufigsten hier erwähnen:

Feste Maßeinheiten Hier sind Maßeinheiten, die sich nicht an die Bildschirmgröße anpassen.

Einheit	CSS	Beschreibung	Größe in px
Zoll	in	Ein Zoll ist 2,54cm lang	96 Pixel
Pica	pc	1/6 eines Zolls	16 Pixel
Punkt	pt	1/72 eines Zolls	1,33 Pixel
Zentimeter	cm	Hunderste Teil eines Meters	37,8 Pixel
Millimeter	mm	Tausendste Teil eines Meters	3,78 Pixel

Einheit	CSS	Beschreibung	Größe in px
Pixel	px	Sichtbare Bereich eines Pixel bei 96 DPI und einer Armlänge Abstand	0,75 Punkt

Diese Maßeinheiten sind besonders für den Druck auf Papier geeignet, da dort das Format eines Papieres (z.B. A4) immer gleich ist und man nicht das Problem hat, dass es sich um ein kleines Mobiles Display handelt.

Relationale Maßeinheiten Als relativ werden Längenmaße bezeichnet, deren Umrechnungsfaktor zu einem absoluten Maß variabel ist. So kann ein bestimmtes Maß je nach Definition (auch innerhalb eines Dokuments) verschiedene Größen annehmen.

Einheit	CSS	Beschreibung
em	em	Bezieht sich auf die Schriftgröße
Wurzel-em	rem	Bezieht sich auf das Wurzelelement der Schrift
Prozent	%	Ist die Prozentangabe zu dem Elternelement
Viewport-Höhe	vh	Ist die Höhe des Anzeigegerätes
Viewport-Breite	vw	Ist die Breite des Anzeigegerätes

Abstände

Bei den Abständen gibt es zwei Möglichkeiten innerhalb von CSS. ### Abstand nach außen Mit Abstand nach außen ist der Abstand zwischen zwei Elementen (wie z.B. zweier Buttons) gemeint. Das heißt damit kann man den Abstand, der zwischen einem Element ist ändern. Die CSS-Eigenschaft hierfür ist: `margin` Das margin bietet die Möglichkeit den Abstand mittels margin-{Richtung} beliebig für jede Position festzusetzen. So kann z.B.

```
margin-top: 5px;
margin-left: 5px;
margin-right: 5px;
margin-bottom: 5px;
```

> Das Element erhält zu jedem anderen Element 5px Abstand

Einzeln vergeben werden oder aber zusammenfassend geschrieben werden:

```
margin: 5px 5px 5px 5px;
```

> Das Element erhält zu jedem anderen Element 5px Abstand. Die Aufteilung ist hier wie folgt: top, rigth, bottom, left Hier können auch nur drei oder zwei Werte angegeben werden. Bei dreien wäre links und rechts das gleiche und bei zweien würde zuerst top|bottom und danach left|right definiert werden.

Abstand nach innen

Mit Abstand nach innen ist der Abstand vom Inhalt zum Rand des Elementes gemeint. So kann man z.B. einen Button deutlich größer erscheinen lassen, wenn man das Padding von den Seiten erhöht (der Abstand von der Schrift des Buttons zum Rand hin wird vergrößert). Die CSS-Eigenschaft hierfür ist: `padding` Das padding bietet die Möglichkeit den Abstand mittels padding-{Richtung} beliebig für jede Position festzusetzen. So kann z.B.

```
padding-top: 5px;
padding-left: 5px;
padding-right: 5px;
padding-bottom: 5px;
```

> Das Element erhält in jede Richtung 5px mehr zum Rand

Einzeln vergeben werden oder aber zusammenfassend geschrieben werden:

```
margin: 5px 5px 5px 5px;
```

> Das Element erhält in jede Richtung 5px mehr zum Rand. Die Aufteilung ist hier wie folgt: top, rigth, bottom, left Hier können auch nur drei oder zwei Werte angegeben werden. Bei dreien wäre links und rechts das gleiche und bei zweien würde zuerst top|bottom und danach left|right definiert werden.

Wie mache ich jetzt weiter?

Da es hier nahezu unmöglich ist alle CSS-Regeln und die verschiedenen Selektoren zu erläutern würde ich gerne auf die CSS-Referenz von Mozilla hinweisen. Dort sind quasi alle Selektoren und Eigenschaften aufgeführt. Um wirklich weiter zu machen und besser zu werden, eignet sich allerdings ein kleines Projekt wie z.B. den eignenen Lebenslauf mal in eine HTML-Seite zu beschreiben.

https://developer.mozilla.org/de/docs/Web/CSS/CSS_Referenz

Exkurs - HTTP und HTTPS

HTTP-Verbindungsaufbau

Vorgang	Beschreibung
Verbindungsaufbau	Der Client schickt eine Anfrage zum Server auf Port 80
Dokument Anfordern	Der Client fordert über das Kommando GET das gewünschte Dokument vom Server an
Übertragung	Der Server fängt an das Dokument zum Client zu übertragen
Verbindungsabbau	Der Verbindungsabbau erfolgt nach der Übertragung vom Server.

HTTPS-Verbindungsaufbau

Vorgang	Beschreibung
Client Anfrage	Der Client schickt eine Anfrage an den Server mit Verschlüsselungsoptionen.
Server Anfrage	Der Server schickt eine Antwort mit seinem Öffentlichen-Schlüssel.
Server überprüfung	Der Client überprüft nun den Schlüssel des Servers. Ist dieser ungültig wird die Verbindung abgebrochen ist sie gültig wird fortgefahren. Dafür generiert der Client einen Sitzungsschlüssel nur für diese Verbindung und verschlüsselt ihn mit dem Öffentlichen-Schlüssel des Servers
Client Annahme	Der Server kann die Anfrage mit dem Sitzungsschlüssel nun mit seinem privaten-Schlüssel entschlüsseln.
Verbindung aufgebaut	Die Verbindung ist nun aufgebaut und es können nun alle HTTP-Anfragen verschlüsselt übertragen werden, bis die Übertragung abgebrochen wird.

Exkurs - Ports

In einem System gibt es mehrere Dienste. Möchte man diese Dienste auch nach außen hin benutzbar machen muss man ihn eine eindeutige Nummer zuweisen. Das hat den Sinn, damit man mehrere Dienste adressieren kann und nicht jedes System nur ein Dienst unterstützen kann.

Bei den Portnummern gibt es unterschiedliche Bereiche.

Name	Bereich	Erläuterung
System Ports / Well-Known-Ports	0 - 1023	Sind standardisierte Ports für Dienste. Hier findet man zum Beispiel Ping (7) HTTP (80) und HTTPS (443)
Registered Ports	1024 - 49151	Sind für registrierte Dienste die nicht in den Standard aufgenommen werden. Hier finden sich diverse Programme von Unternehmen wie Adobe, Microsoft, Google aber auch kleinere Unternehmen.
Dynamic Ports	49152 - 65535	Dynamische Ports werden vom Betriebssystem an einen Client adressiert. So kann zum Beispiel der Webbrowser für eine Anfrage aus dem Bereich einen Port verwenden.