mitp

Hans-Georg Schumann

Spiele programmieren mit JavaScript FÜR KIDS

Hans-Georg Schumann

Spiele programmieren mit JavaScript für Kids

1. Auflage

mitp

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-95845-578-8
1. Auflage 2017

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

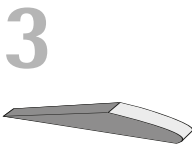
Lektorat: Katja Völpel
Sprachkorrektorat: Petra Heubach-Erdmann
Covergestaltung: Christian Kalkert
Satz: III-satz, Husby, www.drei-satz.de

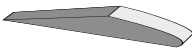
Für
Janne, Julia, Katrin und Daniel

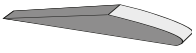
Inhalt

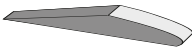
1

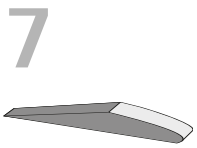
2

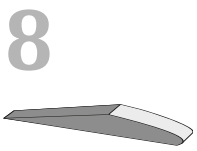


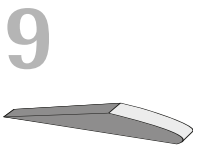


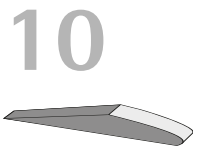


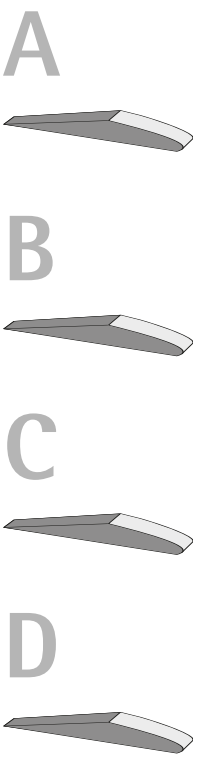












Vorwort

Eigene Fantasiewelten erschaffen, in denen man sich frei bewegen kann. Selbst gebauten Figuren begegnen. Abenteuer selbst erfinden, den Verlauf von Ereignissen selbst bestimmen: Wie wäre das?

Um ein Spiel selbst zu erstellen, muss man vom Programmieren anfangs eigentlich noch gar nichts verstehen. Denn zuallererst braucht man eine Idee und dann einen Plan.

Wovon soll das Spiel handeln? Welche Geschichte soll es erzählen? Personen, Orte und Ereignisse, all das führt zu einem Plan, der umfasst, was zu diesem Spiel gehören soll. Und erst wenn der Plan »steht«, kann die eigentliche Umsetzung in ein Programmprojekt beginnen. Dann allerdings sollte man schon möglichst gut programmieren können.

Wir wollen hier gar nicht so hoch hinaus: Ein professionelles Game wird heutzutage ja von einer ganzen Gruppe von Leuten erstellt, darunter Designer, Künstler, Techniker und nicht zuletzt natürlich Programmierer.

Trotzdem dauert die Arbeitszeit häufig mindestens Monate, wenn nicht Jahre. Die Beteiligten machen einen Vollzeitjob, es ist ihr Beruf. Hier hast du als Einzelgänger nur eine Chance, wenn deine Spielidee so hervorragend und einmalig ist, dass sie alles andere überstrahlt.

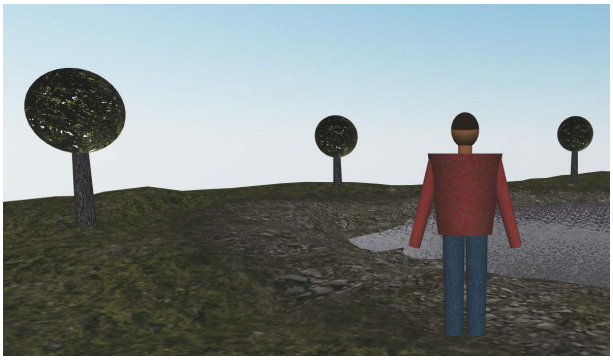
Bleiben wir also auf dem harten Boden der Tatsachen und planen wir nicht ein gigantisches Meisterwerk, sondern kümmern wir uns um solide Grundlagen. Wenn du die beherrschst, hast du durchaus Voraussetzungen, auch einmal an einem professionellen Spielprojekt mitzuwirken.

Welche Werkzeuge benötigen wir?

Um Spiele im 2D- und 3D-Bereich zu erstellen, brauchen wir als Herzstück eine sogenannte **Engine**. Sie muss mit physikalischen Gesetzen umgehen können, damit die Spielwelt mit ihren Figuren und Ereignissen möglichst echt wirkt. Und sie muss grafische Effekte beherrschen, damit das Ganze auch optisch etwas hermacht.

Und das brauchst du:

- die **Babylon Engine**, ein vollwertiges System, das vielfältige Möglichkeiten zum Erstellen von auch professionellen Spielen bietet.



- einen Editor, der dich beim Programmieren unterstützt. Mehr als Notepad, den Windows bereits »eingebaut« hat, bietet **Notepad++**, das wir in diesem Buch einsetzen.

Programmiert wird in der Sprache **JavaScript**, in der auch die komplette Babylon Game Engine erstellt wurde.

- Damit die Programme funktionieren, benötigen wir also einen **Browser**, wie du ihn für das Surfen im Internet benutzt: Ob Google Chrome, Microsoft Edge oder Mozilla Firefox, jeder Browser versteht JavaScript. (Näheres über einige Unterschiede erfährst du im Anhang.)

Wir verwenden hier die derzeit aktuelle Babylon-Version 2.5. Die bekommst du zum Herunterladen auf *http://www.babylonjs.com*. Dort erfährst du auch, ob es eine neuere Version gibt. Alle Programme im Buch funktionieren mit BABYLON.JS 2.5. Bei deutlich höheren Versionsnummern müssen vielleicht einige Programme neu angepasst werden.

Einen komfortablen Editor zum Bearbeiten deiner Programme findest du auf *https://notepad-plus-plus.org*.

Und was bietet dieses Buch?

Vorwiegend geht es um die Programmiersprache JavaScript und natürlich um Spiele. Du erfährst hier unter anderem

- einiges über JavaScript
- etwas über Tasten- und Maussteuerung
- wie man 2D-Spiele mit Sprites erstellt
- wie man Material und Texturen einsetzt
- wie man eine Landschaft zum Spielfeld macht
- wie man einen Androiden selber baut
- etwas über den Umgang mit Kollisionen
- wie man Objekte animiert

Im **Anhang** gibt es dann noch zusätzliche Informationen, unter anderem über Anpassungsprobleme und den Umgang mit Fehlern.

Einleitung

Wie arbeite ich mit diesem Buch?

Um dir den Weg vom ersten Projekt bis zu einem 3D-Game einfacher zu machen, gibt es einige zusätzliche Symbole, die ich dir hier gern erklären möchte:

Arbeitsschritte

≫ Wenn du dieses Zeichen siehst, heißt das: Es gibt etwas zu tun. Damit kommen wir beim Entwickeln Schritt für Schritt einem neuen Ziel immer näher.

→ Grundsätzlich lernt man besser, wenn man etwas selber programmiert. Aber nicht immer hat man große Lust dazu. Weil es alle Projekte im Buch auch zum Download gibt, findest du am Ende des einen oder anderen Abschnitts auch den jeweiligen Dateinamen (zum Beispiel → MYGAME1). Wenn du also das Projekt nicht selbst erstellen willst, kannst du es stattdessen auch aus dem Internet laden – zu finden unter *www.mitp.de/577*

Fragen und Aufgaben

Am Ende eines Kapitels gibt es jeweils eine Reihe von Fragen und Aufgaben. Diese Übungen sind nicht immer ganz einfach, aber sie helfen dir, deine Spiele noch besser zu entwickeln. Lösungen zu den Aufgaben findest du ebenfalls auf der **mitp**-Homepage. Du kannst sie dir alle im Editor von Windows oder auch in deinem Textverarbeitungsprogramm anschauen. Oder du lässt sie dir ausdrucken und hast sie dann schwarz auf weiß, um sie neben deinen PC zu legen.

Notfälle

Vielleicht hast du irgendetwas falsch gemacht oder etwas vergessen. Oder es wird gerade knifflig. Dann fragst du dich, was du nun tun sollst. Bei diesem Symbol findest du eine Lösungsmöglichkeit. Auch ganz hinten im Anhang D findest du ein paar Hinweise zur Pannenhilfe.

Wichtige Stellen im Buch

Hin und wieder findest du ein solch dickes Ausrufezeichen im Buch. Dann ist das eine Stelle, an der etwas besonders Wichtiges steht.

Wenn es um eine ausführlichere Erläuterung geht, tritt Buffi in Erscheinung und schnuppert in seiner Kiste mit Tipps & Tricks.

Was brauchst du für dieses Buch?

Du findest die Babylon Engine als komplette Entwicklungsumgebung zum Download auf dieser Homepage:

http://www.babylonjs.com

Babylon ist kostenlos, du kannst dir dort sogar dein System zusammenstellen, indem du es mit verschiedenen Zusätzen erweiterst.

Die Beispielprojekte in diesem Buch findest du ebenso wie die Lösungen zu den Aufgaben auf der Homepage des Verlages in der gerade aktuellen Version:

http://www.mitp.de/577

Alle Projekt-Ordner enthalten nur die Beispiel-Programme, **nicht** aber die Babylon Engine – aus Copyright-Gründen. Du musst also selbst die Babylon-Datei herunterladen und in den jeweiligen Projekt-Ordner kopieren.

Betriebssystem

Die meisten Computer arbeiten heute mit dem Betriebssystem Windows. Davon brauchst du eine der Versionen ab 7. Mit Babylon lassen sich übrigens nicht nur Spiele für Windows entwickeln, sondern auch für andere Systeme.

Speichermedien

Auf jeden Fall benötigst du etwas wie einen USB-Stick oder eine SD-Card, auch wenn du deine Programme auf die Festplatte speichern willst. Auf einem externen Speicher sind deine Arbeiten auf jeden Fall zusätzlich sicher aufgehoben.

Gegebenenfalls bitte deine Eltern oder Lehrer um Hilfe: Sie sollten den Anhang A lesen. Dann können sie dir bei den ersten Schritten besser helfen.

1

Das erste Projekt

Du möchtest natürlich gleich dein erstes Spiel erstellen. So schnell geht das leider nicht, aber mit dem Programmieren können wir sofort loslegen. Du brauchst dazu anfangs nur den Browser, mit dem du auch im Internet surfst. Dort benutzen wir eine »Spiel-Maschine« namens Babylon, mit der lassen sich jede Menge Spiele erstellen. Aber wir fangen erst mal mit etwas Einfachem an.

In diesem Kapitel lernst du

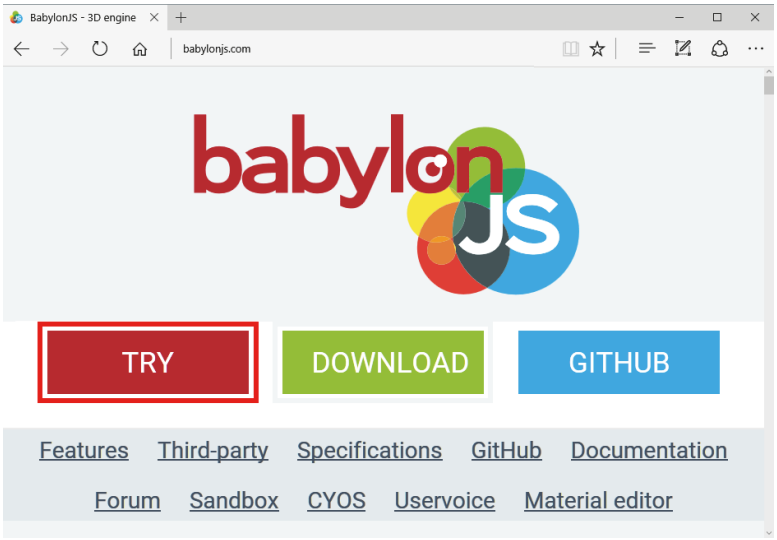
- wie man in Babylon einsteigt
- wie man Objekte in einem Projekt einsetzt
- etwas über 2D und 3D
- wie man die Position von Objekten ändert
- was Funktionen und Vektoren sind

In Babylon einsteigen

Starte einen Browser deiner Wahl, das kann zum Beispiel Microsoft Edge, Google Chrome oder Mozilla Firefox sein. Dort gib in der Adresszeile diese Adresse ein:

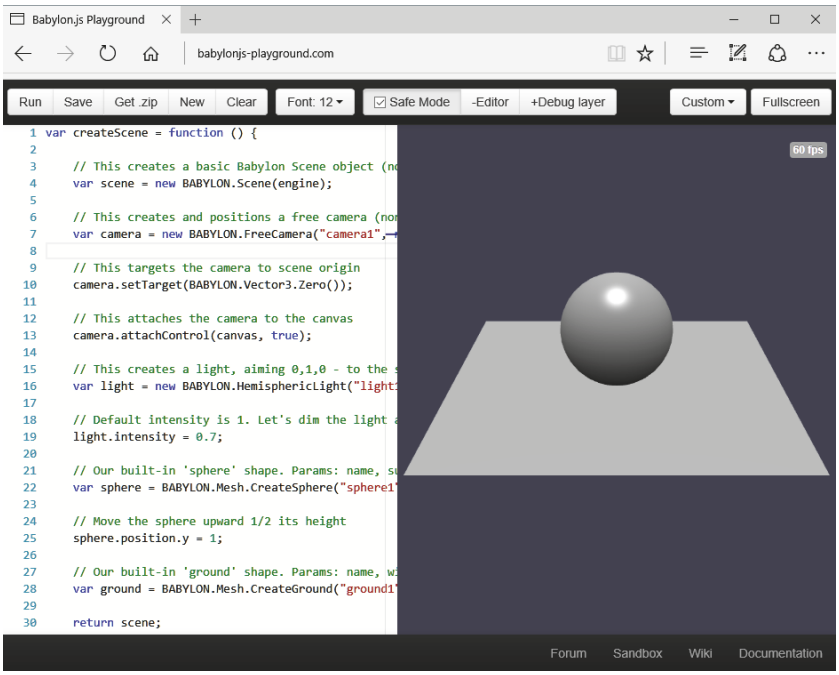
http://www.babylonjs.com/

Nach einer Weile landest du auf der Homepage von **Babylon JS**. Dort findest du eins der Werkzeuge, die wir für dieses Buch brauchen: ein komplettes System zum Erstellen von Spielen.



Probieren wir das gleich aus.

≫ Klicke auf die Schaltfläche TRY, um in den sogenannten **Playground** umzuschalten.



Auf der rechten Seite siehst du eine Fläche und darüber eine Kugel. In diesem Bereich findet auch später dein Spiel statt.

Man nennt das Ganze auch **Spielwelt**, denn natürlich lässt sich dieser Bereich später auf den ganzen Bildschirm vergrößern. Wie bei einem echten Spiel kann man dann auch nach rechts oder links weiterwandern. Das alles gehört zur Spielwelt (die mitunter sogar riesig sein kann).

Die Fläche unter der Kugel ist so etwas wie das **Spielfeld**. Und die Kugel selbst ist dann die **Spielfigur**. Und damit kennst du nun das, was mindestens zu einem Spiel gehört: eine Spielfigur und ein Spielfeld.

Auf der linken Seite deines Browsers befindet sich furchtbar viel Text. Alles, was da steht, ist in der Programmiersprache **JavaScript** geschrieben. Und das ist auch die Sprache, in der du hier vorwiegend programmieren wirst. (Und jetzt kannst du dir wohl denken, was das **»JS«** hinter »Babylon« bedeutet, oder?)

Es bleibt dir nicht erspart, sich näher mit dem »ganzen Kram« zu befassen, der da steht. Ein besserer Ausdruck dafür ist **Quelltext**. Wir gehen das Ganze gleich langsam Zeile für Zeile durch. Genau genommen ist es gar nicht so viel, wie es zuerst aussieht.

Ich habe alle Zeilen, die mit einem doppelten Schrägstrich beginnen, einmal entfernt, weil sie für das Funktionieren des Programms nicht nötig sind. Es handelt sich um sogenannte **Kommentare**, in denen (auf Englisch) erläutert wird, was an der jeweiligen Stelle passiert (oder passieren soll). Solche Zeilen werden von Babylon einfach übersprungen.

Was übrig bleibt, sieht so aus:

```
var createScene = function () {
    var scene = new BABYLON.Scene(engine);
    var camera = new BABYLON.FreeCamera
    ("camera1", new BABYLON.Vector3(0,5,-10), scene);
    camera.setTarget(BABYLON.Vector3.Zero());
    camera.attachControl(canvas, true);
    var light = new BABYLON.HemisphericLight
    ("light1", new BABYLON.Vector3(0,1,0), scene);
    light.intensity = 0.7;
    var sphere = BABYLON.Mesh.CreateSphere
    ("sphere1", 16, 2, scene);
```

```
    sphere.position.y = 1;
    var ground = BABYLON.Mesh.CreateGround
    ("ground1", 6, 6, 2, scene);
    return scene;
};
```

In Wirklichkeit hat der Quelltext noch weniger Zeilen, ich musste nur einige sehr lange Exemplare auf zwei Buch-Zeilen verteilen.

Dieser Quelltext ist eine Ansammlung von **Anweisungen**, die der Computer ausführen soll. Jede davon wird mit einem Semikolon (;) abgeschlossen. Alle Anweisungen zusammen bewirken das, was du rechts neben dem Text zu sehen bekommst.

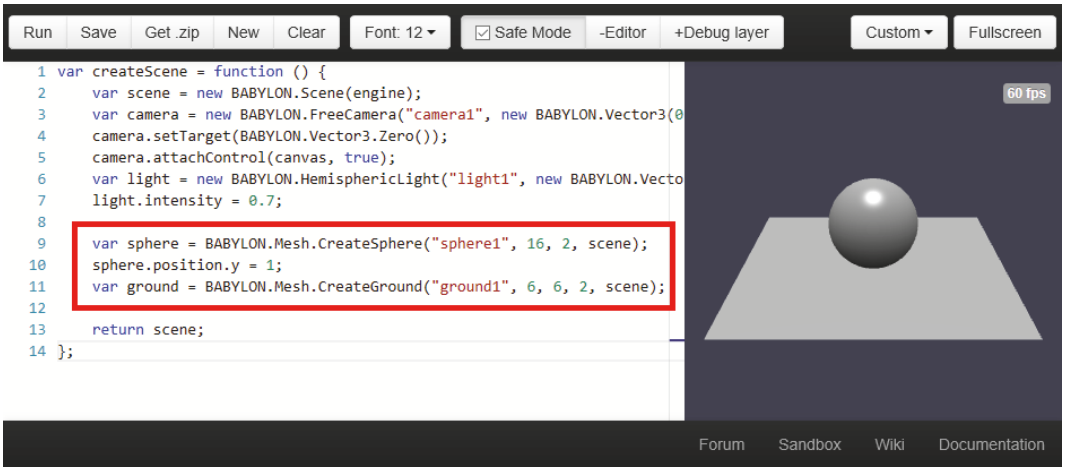
Da taucht ständig dieses Wörtchen var auf. Das ist eine Abkürzung für **Variable**. Direkt dahinter steht ein Name und dann folgt ein sogenanntes **Zuweisungszeichen** (=), das wie ein Gleichheitszeichen aussieht.

Jedes Mal, wenn var erscheint, heißt das: Es gibt etwas Neues, das Programm oder Spiel wird um ein neues Element erweitert. Dabei ist die Variable ein Platzhalter oder eine Art Behälter, dessen Inhalt veränderbar ist.

Und die **Zuweisung** bedeutet, dass in den Behälter etwas »eingefüllt« wird. Hinter dem Zuweisungszeichen kann alles Mögliche stehen. Und wie du oben siehst, werden dem Spiel mithilfe von var Elemente **hinzugefügt** (wie Kamera, Licht, Kugel und Untergrund).

Zu beachten ist, dass bei einigen var-Zuweisungen ein zusätzliches new nötig ist. Ich komme weiter unten darauf zurück.

Schauen wir jetzt mal noch genauer hin. Dabei beginnen wir nicht ganz oben. Das wäre noch zu kompliziert, dazu kommen wir also erst später. Beginnen wir mit den Zeilen, in denen wir selbst schon mal etwas »machen« können.



Ein Objekt positionieren

In den beiden Anweisungen, die ich meine, geht es zuerst um ein Objekt namens `sphere`, zu Deutsch: Kugel. Diese Zeilenfolge sorgt dafür, dass nebenan im Fenster eine solche Kugel zu sehen ist:

```
var sphere = BABYLON.Mesh.CreateSphere
("sphere1", 16, 2, scene);
sphere.position.y = 1;
```

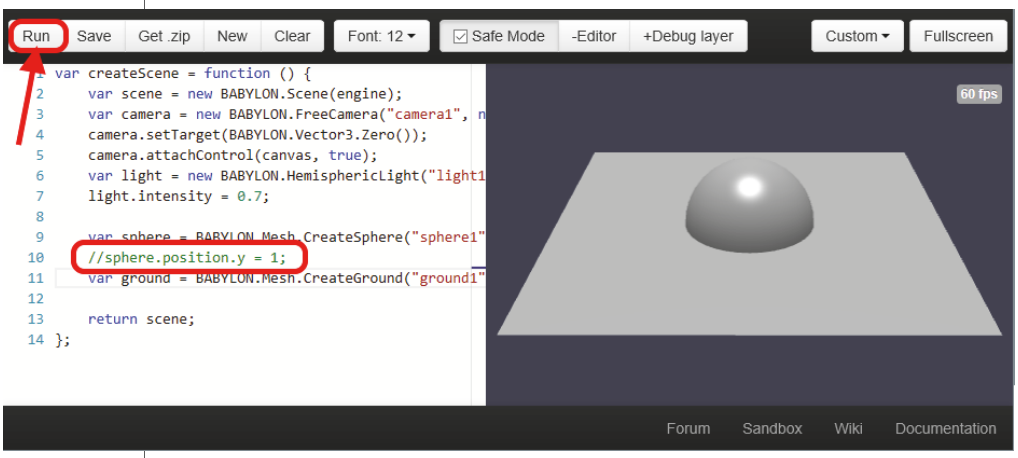
Mit `CreateSphere()` wird eine Kugel erzeugt. Und mit `position` wird diese Kugel auf der Unterlage positioniert und nicht »dazwischen«.

≫ Damit du verstehst, was ich damit meine, ändere die betreffende Zeile einmal so um:

```
// sphere.position.y = 1;
```

Durch die zwei vorgeschalteten Schrägstriche (`//`) wird das Ganze zu einem Kommentar. Es ist damit keine Anweisung mehr und wird nach dem Programmstart übersprungen.

≫ Klicke nun oben links auf RUN, um das Programm neu zu starten.

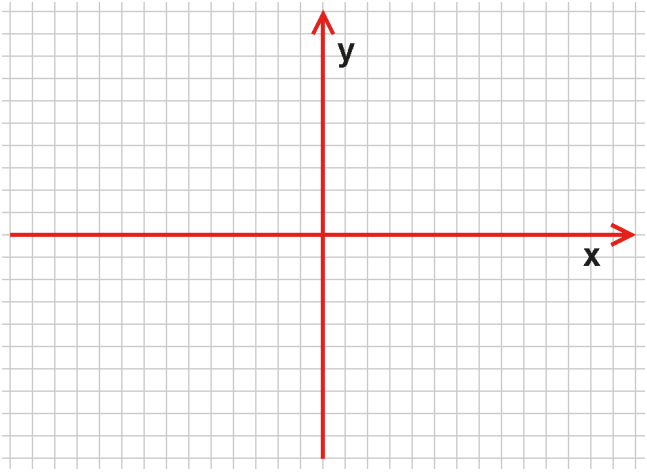


Und schon sackt die Kugel in den Untergrund. Mit dem Zusatz `position` können wir also die Position eines Objekts ändern.

Standardmäßig wird ein neues Objekt erst mal in die **Mitte** einer Spielwelt gesetzt. Das entspricht dann diesem Positionswert:

```
sphere.position.y = 0;
```

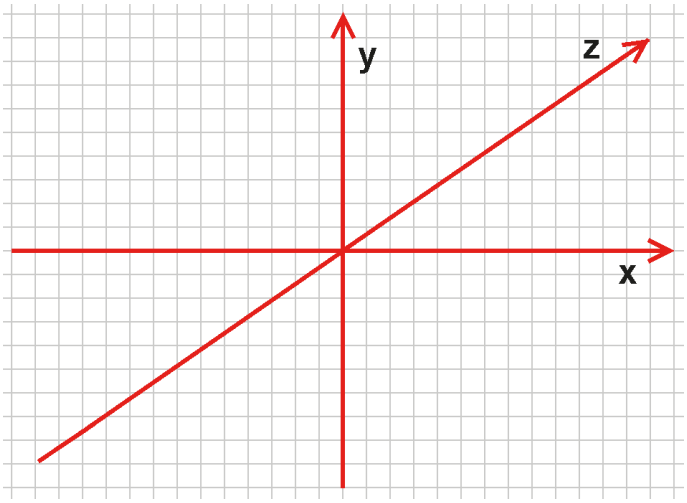
Wenn man also für `position` gar nichts angibt, dann wird der Wert auf 0 gesetzt. Aber schauen wir mal genauer hin. Da steht ja auch noch ein `y`. Vielleicht weißt du aus der Schule, was ein **Koordinatensystem** ist. Da gibt es eine x-Achse (= Horizontale oder Waagerechte) und eine y-Achse (= Vertikale oder Senkrechte).



Entlang der x-Achse geht es also nach links oder rechts, entlang der y-Achse nach oben oder unten. Genau in der Mitte, wo sich die Achsen kreuzen, sind die Koordinatenwerte (0 | 0). Das nennt man 2D.

In Babylon gibt es noch eine dritte Ache, die man z-Achse nennt. An der entlang geht es nach vorn oder nach hinten. Damit haben wir 3D.

Schaut man von vorn auf das Koordinatensystem, dann kann man diese Achse nicht sehen. Um alle Achsen dennoch in 2D sichtbar zu machen, greift man zu einem optischen Trick: Die z-Achse wird als Diagonale dargestellt.



Und so kann man die Richtungen für die Positionierung von Objekten wie zum Beispiel einer Kugel in einer Tabelle zusammenfassen:

Achse	X	Y	Z
Richtung	links-rechts	oben-unten	vorn-hinten

2D oder 3D? Was genau bedeutet das und was ist der Unterschied? Ein Bild auf einem Blatt Papier ist zweidimensional (abgekürzt: 2D), es hat zwei Dimensionen, eine Länge und eine Breite. Dein Computer oder dein Smartphone sind dreidimensional (abgekürzt: 3D), denn dort gibt es zusätzlich noch eine Höhe oder Dicke.

Bei Spielen gibt es das eigentlich nicht, denn die Darstellung auf dem Monitor oder Display ist immer 2D. Die dritte Dimension wird künstlich erzeugt, es sieht dann für die Augen so aus, als wäre das Ganze 3D. Mit Babylon können die Spiele 2D oder 3D sein.

Probieren wir doch gleich mal aus, wie sich die Lage der Kugel verändern lässt.

≫ Entferne zuerst die Kommentarzeichen (//). Dann ändere die vorhandene Zeile so um:

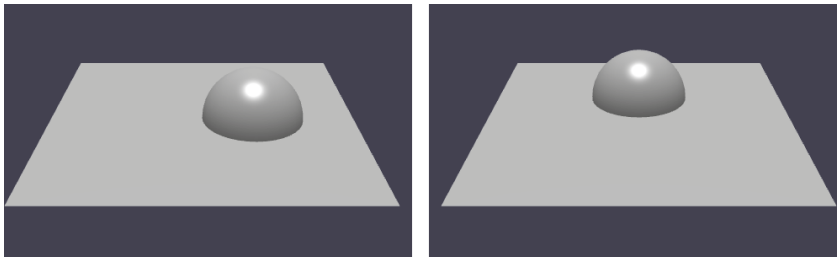
```
sphere.position.x = 1;
```

Und dann so:

```
sphere.position.z = 1;
```

≫ Vergiss nicht, jedes Mal auf RUN zu klicken.

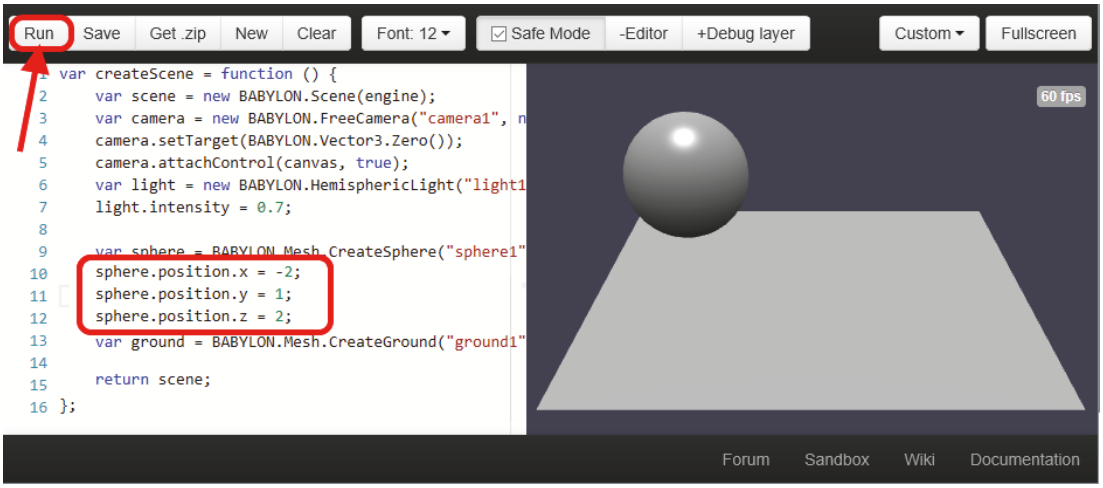
Im einen Fall verschiebt sich die (eingesackte) Kugel nach rechts, im anderen nach hinten:



≫ Nun mache aus der vorhandenen Zeile die folgenden drei:

```
sphere.position.x = -2;
sphere.position.y = 1;
sphere.position.z = 2;
```

≫ Dann klicke auf RUN.



Und du kannst sehen, dass die Kugel nun hinten links in der Ecke liegt, und nicht in, sondern auf der Spielfläche.

Wenn du etwas falsch eingetippt oder versehentlich etwas gelöscht hast, kannst du das in der Regel mit Strg+Z wieder rückgängig machen.

Die Sache mit Create

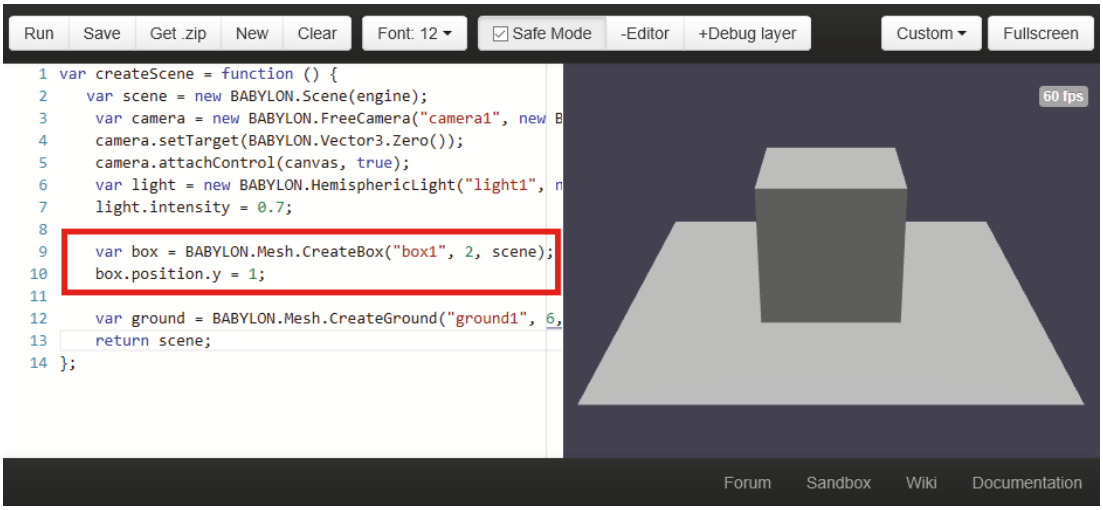
Du weißt also jetzt, wie man eine Kugel als Spielelement erzeugt und in Position bringt. (Spielfigur ist hier noch etwas übertrieben, solange sich das »Ding« nicht bewegen kann. Aber warte ab, das kommt noch.)

Was ist mit anderen Objekten? Da muss es doch noch mehr geben.

≫ Probiere doch mal das Folgende aus: Entferne die Zeilen für die Kugel und ersetze sie durch diese:

```
var box = BABYLON.Mesh.CreateBox("box1", 2, scene);
box.position.y = 1;
```

Damit bekommst du nun einen Würfel.



Weitere Objekte wirst du im Laufe dieses Buches kennenlernen. Nun kümmern wir uns um das Drumherum. Das ist auch wichtig dafür, dass

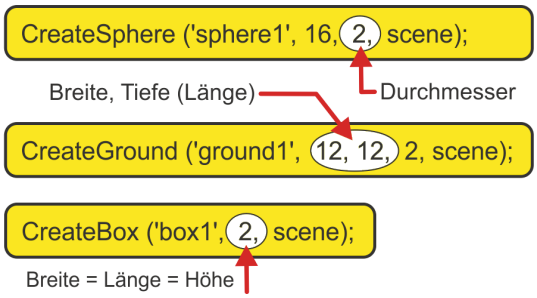
wir eine Kugel oder einen Würfel auf einem Spielfeld zu sehen bekommen. Das Spielfeld, auch Untergrund genannt, wird übrigens so erzeugt:

```
var ground = BABYLON.Mesh.CreateGround
 ("ground1", 6, 6, 2, scene);
```

Das erledigt die Funktion `CreateGround()`. Womit wir nun schon mal diese drei Funktionen kennen:

`CreateSphere()`	Eine Kugel erzeugen
`CreateGround()`	Eine Fläche (als Untergrund) erzeugen
`CreateBox()`	Einen Quader oder Würfel erzeugen

Alle werden mit »Create« eingeleitet, dem englischen Wort für »Erzeugen«. Schon ganz zu Anfang kannst du bestimmen, wie groß das jeweilige Objekt sein soll. Dafür sind sogenannte Parameter da, die in Klammern hinter dem `Create`-Namen stehen.



Eine Zwischenbemerkung zu den **Anführungszeichen** ("), die hier immer wieder auftauchen: Bis jetzt sind dir wohl immer die doppelten begegnet. Wenn du dir Quelltexte im Internet ansiehst, wirst du aber auch oft auf die einfachen Anführungszeichen (') stoßen. In JavaScript sind beide **gleichwertig** erlaubt.

Falls du Probleme hast, die Anführungszeichen auf der Tastatur zu finden, versuch's mal mit ⇧+2 und ⇧+#.

≫ Probiere jetzt diese Anweisungen für das Spielfeld gleich einmal aus:

```
var ground = BABYLON.Mesh.CreateGround
 ("ground1", 15, 15, 2, scene);
```