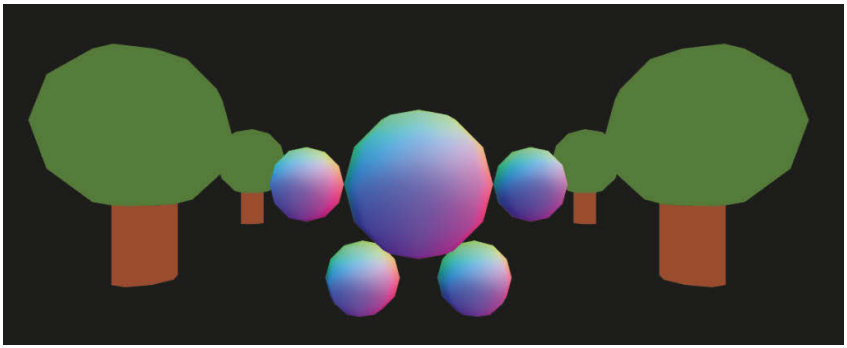
PROJEKT: AVATARE BEWEGEN

4

WENN DU DIESES KAPITEL GELESEN HAST, DANN

- kannst du Dinge mit deiner Tastatur steuern
- verstehst du, wie man Kameras dazu bringt, sich mit dem Avatar zu bewegen
- hast du erste Erfahrungen mit JavaScript-Ereignissen gemacht, die irre wichtig und auch ein bisschen seltsam sind

In Kapitel 3, *Projekt: Einen Avatar herstellen*, haben wir behandelt, wie man einen Spiele-Avatar baut. Ein Avatar, den wir nicht bewegen können, ist aber ziemlich langweilig. Deshalb lernst du in diesem Kapitel, wie du den Avatar in unterschiedliche Richtungen bewegst. Wir stellen ihm dazu noch einen kleinen Wald zur Verfügung, in dem er herumlaufen kann. Das Ganze wird dann ungefähr so aussehen:



◀ Abbildung 4-1
Ein Männlein steht im Walde ...

Ehrlich, wenn du mit diesem Kapitel fertig bist, wirst du dich fühlen, als hättest du Superkräfte. Und so ganz verkehrt ist das gar nicht – Programmieren *ist* eine Superkraft!

Leg los

Dieses Kapitel baut auf der Arbeit auf, die wir in dem *Projekt: Einen Avatar herstellen* erledigt haben. Falls du die Übungen in dem Kapitel noch nicht gemacht hast, solltest du dorthin zurückgehen und sie durchführen, bevor du weitermachst. Du müsstest dich speziell mit der `animate()`-Übung am Ende des Kapitels befassen.

Wir wollen auf dem Code aus dem letzten Kapitel aufbauen, möchten diesen aber nicht verlieren. Kopieren wir daher den Code aus Kapitel 3 in ein neues Projekt. Auf diese Weise ist unser alter Code noch da, falls wir ihn brauchen.

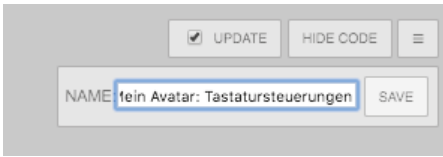
Sichere deine Arbeit

Funktionierender Code ist eine kostbare und wunderbare Sache. Ich programmiere schon seit 20 Jahren und schreibe trotzdem immer noch mehr kaputten als funktionierenden Code. Aber sobald ich es geschafft habe, dass endlich etwas funktioniert, sichere ich es zuallererst. Na ja, ehrlich gesagt, ist das Erste, was ich mache, ein kleiner Freudentanz – ein winzig kleiner, ich tanze nämlich nicht besonders gut –, weil das gefeiert werden muss. Und alles was man feiern kann, sollte man sich sichern.

Um eine Kopie dieses Projekts herzustellen, klickst du auf den Menübutton und wählst MAKE A COPY aus dem Menü:

Abbildung 4-2 ▶
Make a Copy auswählen

Nennen wir das Projekt `Mein Avatar: Tastatursteuerungen`. Gib das als Projektnamen ein.



◀ Abbildung 4-3
Das Projekt benennen

Klicke anschließend auf SAVE. Jetzt sind wir bereit, die Tastatursteuerungen hinzuzufügen.

Mit Tastaturereignissen interaktive Systeme bauen

Wenn Dinge in Webbrowsern passieren, sind diese Dinge für JavaScript sogenannte *Ereignisse* (Events). Damit unser Code etwas macht, sobald ein Ereignis eintritt, schreiben wir Code, der auf verschiedene Arten von Ereignissen *lauscht*. Dies wird *Event-Listener* genannt, es sind also sozusagen Ereignisüberwacher. Events und Event-Listener kommen dir zu Anfang vielleicht ein bisschen seltsam vor, aber du wirst schnell dahinterkommen, sobald du den entsprechenden Code siehst. Schreiben wir uns welchen!

Fügen wir Folgendes ganz am Ende unseres Code hinzu, und zwar unter der `animate()`-Zeile aus Kapitel 3, *Projekt: Einen Avatar herstellen*.

```
document.addEventListener('keydown', sendKeyDown);
function sendKeyDown(event) {
  alert(event.code);
}
```

Dieser Code lauscht auf ein `keydown`-Ereignis. Ein solches Ereignis tritt ein, wenn eine Taste gedrückt wird. Wenn unser Code ein `keydown` *hört*, teilt er der `sendKeyDown()`-Funktion mit, dass sie diese Taste benutzen soll, um einen Befehl an den Avatar zu schicken.

Bevor wir dem Avatar sagen können, dass er sich bewegen soll, müssen wir herauskriegen, welche Taste gedrückt wurde. Deshalb untersuchen wir in unserer `sendKeyDown()`-Funktion den `event.code`-Wert.

Was ist das für ein `code`? Um das zu beantworten, probieren wir es doch einfach aus! Klicke auf den HIDE CODE-Button oben auf der Seite und drücke dann die Taste [A] auf deiner Tastatur. Du solltest einen solchen Benachrichtigungsdialog sehen.

Abbildung 4-4 ▶
A hat den Tastencode KeyA.

Cool! Wir haben die Taste A gedrückt, die JavaScript `KeyA` nennt. Was ist mit den Pfeiltasten?

Falls du es noch nicht gemacht hast, klicke in dem Dialog auf OK. Wiederhole das dann für die vier Pfeiltasten (Cursortasten) auf deiner Tastatur. Du wirst feststellen, dass der Computer für den Linkspfeil an `ArrowLeft` denkt. Für den Hochpfeil denkt er an `ArrowUp`. Und für den Rechtspfeil erkennt der Computer die Taste als `ArrowRight`. Für den Runterpfeil denkt der Computer sich `ArrowDown`.

Benutzen wir diese Tastencodes, um unseren Avatar zu bewegen!

Tastaturereignisse in Avatar-Bewegungen verwandeln

Blende deinen Code wieder ein und entferne dann die `alert(event.code)`-Zeile in `document.addEventListener()`. Ersetze sie durch Folgendes:

```
document.addEventListener('keydown', sendKeyDown);
function sendKeyDown(event) {
  var code = event.code;
  if (code == 'ArrowLeft') avatar.position.x = avatar.position.x - 5;
  if (code == 'ArrowRight') avatar.position.x = avatar.position.x + 5;
  if (code == 'ArrowUp') avatar.position.z = avatar.position.z - 5;
  if (code == 'ArrowDown') avatar.position.z = avatar.position.z + 5;
}
```

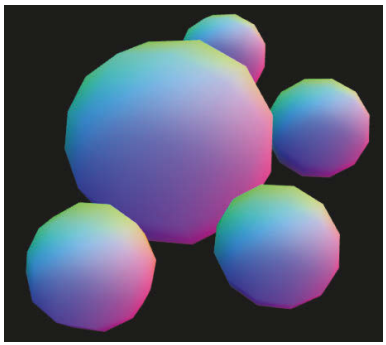
Wir werden in Kapitel 7, *Die Grundlagen von JavaScript näher untersucht*, über `if` (ob), `==` (ist es gleich?) und `=` (setze es gleich) sprechen. Der Code sollte aber auch so bereits verständlich sein. Wir prüfen, ob (`if`) der Tastencode von einer Pfeiltaste stammt. Handelt es sich bei dem Tastencode zum Beispiel um `ArrowLeft`, ändern wir die x-Position des Avatars, indem wir 5 subtrahieren.

Lass uns spielen!

Drücke den HIDE CODE-Button und versuche es selbst einmal. Benutze die Pfeiltasten, um den Avatar zu bewegen. Funktioniert das so, wie du es erwartet hast?

Merke dir: Falls etwas schiefgeht, musst du auf der JavaScript-Konsole nachschauen!

Wenn alles richtig funktioniert, solltest du in der Lage sein, deinen Avatar nach ganz weit hinten, nach ganz vorn, nach links oder rechts und sogar aus dem Bildschirm heraus zu bewegen.



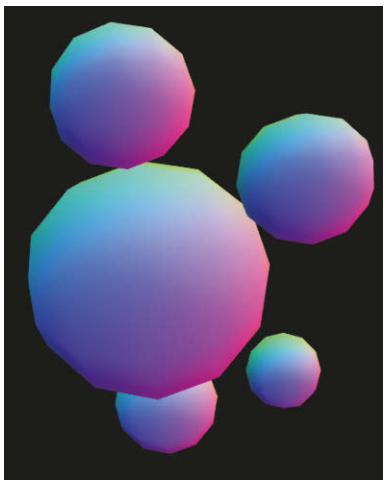
◀ Abbildung 4-5
Der Avatar läuft davon.

Als wir unserem Avatar im Abschnitt *Räder schlagen* eben genau diese Fähigkeit gaben, hast du gelernt, wie man dafür sorgt, dass sich die Hände und Füße des Avatars zusammen mit dem Körper bewegen. Da die Hände und Füße dem Avatar-Objekt hinzugefügt wurden und nicht der Szene, bewegen sie sich zusammen mit dem Avatar über den Bildschirm.

Schauen wir uns einmal an, was passiert, wenn eines der Beine nicht mit dem Avatar verbunden ist. Wir ändern in diesem Fall `leftFoot` so, dass dieser Fuß mit der Szene verbunden ist anstatt mit dem Avatar.

```
var leftFoot = new THREE.Mesh(foot, cover);
leftFoot.position.set(75, -125, 0);
➤ scene.add(leftFoot);
```

Siehst du? Der Fuß geht verloren.



◀ Abbildung 4-6
Hoppla, hier fehlt ihm ein Fuß!

Unterschätze die Macht dieses Konzepts nicht. Wir werden später damit einige wirklich verrückte Dinge veranstalten.

Doch jetzt solltest du den linken Fuß wieder am Avatar befestigen!

Herausforderung: Animation starten/stoppen

Erinnerst du dich an die `isCartwheeling`- und `isFlipping`-Werte aus Kapitel 3, *Projekt: Einen Avatar herstellen*? Fügen wir dem Tastatur-Event-Listener noch zwei weitere `if`-Anweisungen hinzu. Wenn C gedrückt ist, soll der Avatar entweder beginnen oder aufhören, Räder zu schlagen. Ist F gedrückt, soll das Herumdrehen beginnen oder enden.

Hinweis: Du kannst zwischen `true` und `false` wechseln, indem du ein Ausrufezeichen vor einen Wert setzt. So kannst du den gegenteiligen Wert des Ursprungswerts von `isCartwheeling` zuweisen, und zwar mit so etwas wie `isCartwheeling = !isCartwheeling`. Uns wird das später im Abschnitt *Boolesche Werte* wieder begegnen. Jetzt versuch erst einmal, `isCartwheeling` und `isFlipping` zu ändern, wenn die richtige Taste gedrückt ist.

Hast du es selbst geschafft? Keine Panik, falls es nicht geklappt hat – diese Art von JavaScript wirkt manchmal beim ersten Versuch etwas seltsam. Hier ist die `animate()`-Funktion, die das Räderschlagen und Drehen erledigt:

```
var isCartwheeling = false;
var isFlipping = false;
function animate() {
  requestAnimationFrame(animate);
  if (isCartwheeling) {
    avatar.rotation.z = avatar.rotation.z + 0.05;
  }
  if (isFlipping) {
    avatar.rotation.x = avatar.rotation.x + 0.05;
  }
  renderer.render(scene, camera);
}
animate();
```

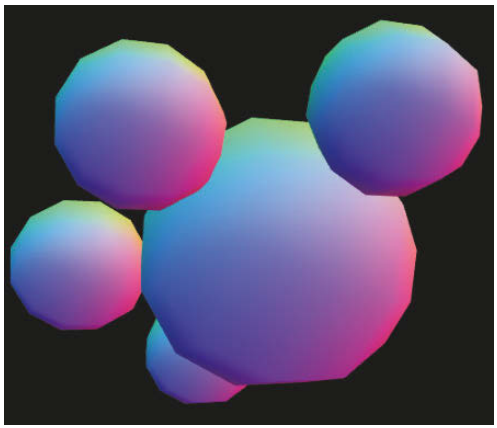
Und dies ist der vollständige Tastaturcode zum Bewegen, Drehen und Radschlagen unseres Avatars:

```
document.addEventListener('keydown', sendKeyDown);
function sendKeyDown(event) {
  var code = event.code;
  if (code == 'ArrowLeft') avatar.position.x = avatar.position.x - 5;
```

```
  if (code == 'ArrowRight') avatar.position.x = avatar.position.x + 5;
  if (code == 'ArrowUp') avatar.position.z = avatar.position.z - 5;
  if (code == 'ArrowDown') avatar.position.z = avatar.position.z + 5;

  if (code == 'KeyC') isCartwheeling = !isCartwheeling;
  if (code == 'KeyF') isFlipping = !isFlipping;
}
```

Wenn du es richtig gemacht hast, solltest du in der Lage sein, den Avatar Überschläge in verschiedene Richtungen machen zu lassen, während er sich aus dem Bildschirm herausbewegt.



◀ Abbildung 4-7
Der Avatar kugelt aus dem Bildschirm heraus.

Es ist eigentlich ziemlich verrückt, dass der Avatar den Bildschirm verlassen kann. Wir werden das in Kürze beheben. Zuerst aber wollen wir unserem Avatar einige Bäume spendieren, zwischen denen er herumspazieren kann.

Mit Funktionen einen Wald bauen

Für unseren Wald brauchen wir eine Menge Bäume. Wir könnten sie einzeln bauen, werden das aber nicht tun. Stattdessen wollen wir hinter den Avatar-Körperteilen das folgende JavaScript einfügen:

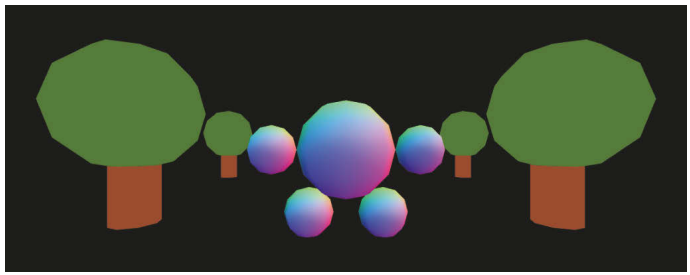
```
function makeTreeAt(x, z) {
  var trunk = new THREE.Mesh(
    new THREE.CylinderGeometry(50, 50, 200),
    new THREE.MeshBasicMaterial({color: 'sienna'})
  );
  var top = new THREE.Mesh(
    new THREE.SphereGeometry(150),
    new THREE.MeshBasicMaterial({color: 'forestgreen'})
  );
```

```
  top.position.y = 175;
  trunk.add(top);

  trunk.position.set(x, -75, z);
  scene.add(trunk);
}
makeTreeAt( 500, 0);
makeTreeAt(-500, 0);
makeTreeAt( 750, -1000);
makeTreeAt(-750, -1000);
```

Wenn du all diesen Code korrekt eingetippt hast, siehst du deinen Avatar vor einem Wald, der aus vier Bäumen besteht.

Abbildung 4-8 ▶
Da steht ein Avatar im Wald.



Das ist ziemlich cool, aber wie haben wir das gemacht?

Das Ganze auseinandernehmen

Die meiste Arbeit passiert in der Funktion `makeTreeAt()`. Wie wir in Kapitel 5, *Funktionen: immer und immer wieder benutzen*, sehen werden, bietet eine JavaScript-Funktion eine Möglichkeit, denselben Code immer wieder zu wiederholen. In diesem Fall baut unser Code wiederholt einen Baumstamm und eine Baumkrone. Wir hätten die Funktion auch irgendwie anders nennen können, aber wir möchten, dass der Name uns genau sagt, was sie macht – sie erzeugt einen Baum an den Koordinaten `x` (links/rechts) und `z` (vorn/hinten).

Das Innenleben der `makeTreeAt()`-Funktion dürfte uns schon ein bisschen bekannt vorkommen.

```
  function makeTreeAt(x, z) {
❶   var trunk = new THREE.Mesh(
      new THREE.CylinderGeometry(50, 50, 200),
      new THREE.MeshBasicMaterial({color: 'sienna'})
    );

❷   var top = new THREE.Mesh(
      new THREE.SphereGeometry(150),
```

```
      new THREE.MeshBasicMaterial({color: 'forestgreen'})
    );
❸   top.position.y = 175;
❹   trunk.add(top);

❺   trunk.position.set(x, -75, z);
❻   scene.add(trunk);
  }
```

❶ Stell aus einem Zylinder einen Baumstamm her.

❷ Stell aus einer Kugel eine Baumkrone her.

❸ Verschiebe die Baumkrone nach oben (denk dran, y ist hoch und runter) auf den Stamm.

❹ Füge die Baumkrone dem Baumstamm hinzu.

❺ Setze die Position des Stamms auf die x- und z-Werte, mit denen die Funktion aufgerufen wurde – zum Beispiel `makeTreeAt(500,0)`. Der y-Wert –75 verschiebt den Stamm so weit nach unten, dass er wie ein Baumstamm aussieht.

❻ Füge den Stamm mit seiner Baumkrone der Szene hinzu.

Du musst dir unbedingt merken, dass wir die Baumkrone zum Stamm hinzufügen müssen und nicht zur Szene. Würden wir Baumkrone und Stamm zu der Szene hinzufügen, müssten wir daran denken, beide zu bewegen. Wird die Krone zum Stamm hinzugefügt, wird beim Verschieben des Stamms auch die Baumkrone bewegt.

Eine Funktion ist hier nicht unbedingt nötig. Du bist inzwischen so gut beim Herstellen von Formen, dass du vermutlich vier Zylinder mit vier darauf sitzenden Kugeln irgendwo in der Szene anlegen könntest. Aber faule Programmierer (das sind die besten!) tippen am liebsten nicht mehr als nötig. Deswegen zeigen wir einer Funktion einmal, wie sie einen Baum bauen kann, und benutzen dann diese Funktion immer wieder. Und wenn wir 20 Bäume hinzufügen wollen, benutzen wir eben noch 20 Mal diese Funktion.

Neu ist hier auch die Farbe. Wir wählten diese Farben aus der (englischsprachigen) Wikipedia-Liste der Farbnamen.[1] Der Baumstamm hat die Farbe Sienna. Du kannst natürlich deine eigenen Farben ausprobieren. Die meisten Farbnamen funktionieren. Denk lediglich daran, alles kleinzuschreiben und keine Leerzeichen zu verwenden (`'forestgreen'` statt `'forest green'`).

Wenn wir die Funktion haben, ist es kein Problem, sie zu benutzen. Wir stellen bei einem x-Wert von 500 und einem z-Wert von 0 einen Baum her,

1 *https://en.wikipedia.org/wiki/Web_colors*

indem wir die Funktion mit den entsprechenden Werten in der Klammer aufrufen, also etwa so: `makeTreeAt(500, 0)`.

```
makeTreeAt( 500, 0);
makeTreeAt(-500, 0);
makeTreeAt( 750, -1000);
makeTreeAt(-750, -1000);
```

Jetzt haben wir den Wald und wollen einmal schauen, was wir machen können, wenn unser Avatar immer aus dem Bildschirm herauswandert.

Die Kamera mit dem Avatar bewegen

Am einfachsten kannst du den Avatar auf dem Bildschirm behalten, wenn du die Kamera stets dorthin bewegst, wo auch der Avatar hingeht. Wenn die Kamera immer auf den Avatar gerichtet ist, kann er nicht vom Bildschirm verschwinden!

Um die Hände und Füße zusammen mit dem Avatar bewegen zu können, haben wir diese nicht der Szene, sondern dem Körper des Avatars hinzugefügt. Und genau das müssen wir auch mit der Kamera machen.

Suchen wir zuerst die Zeile, in der `scene.add(camera)` steht, und löschen wir sie. Unter der Zeile, in der der Avatar der Szene hinzugefügt wird, und über der `makeTreeAt()`-Funktion fügen wir dem Avatar dann die Kamera hinzu:

```
var leftFoot = new THREE.Mesh(foot, cover);
leftFoot.position.set(75, -125, 0);
avatar.add(leftFoot);

➤ avatar.add(camera);
```

Nachdem du den Code ausgeblendet hast, siehst du, dass die Kamera direkt vor dem Avatar bleibt, auch wenn du ihn bewegst.

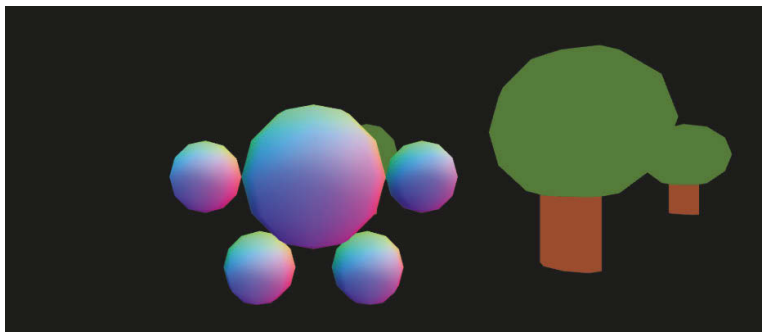
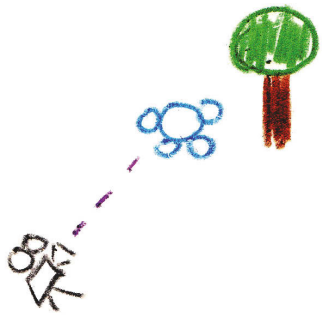


Abbildung 4-9 ▶
Kamera und Avatar bewegen sich gemeinsam.

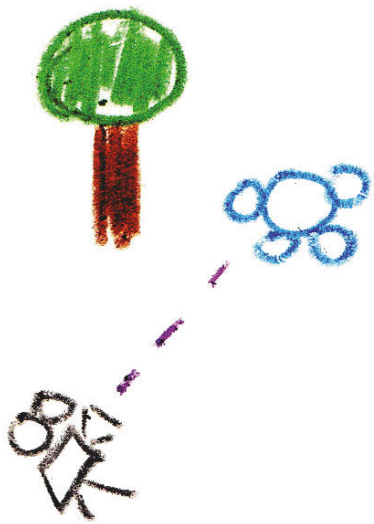
Die Kamera startet 500 Einheiten vor dem Avatar. Der Code ganz oben setzt ihre Position auf `camera.position.z = 500` oder 500 Einheiten davor. Vorher war sie 500 Einheiten vor der Mitte der Szene. Und dort blieb sie, selbst wenn sich der Avatar bewegt hat. Nachdem wir sie nun dem Avatar hinzugefügt haben, hält sie stets denselben Abstand zum Avatar ein.

Vielleicht hilft es, wenn du dir vorstellst, dass die Kamera mit einem Selfiestick an dem Avatar befestigt ist.



◀ Abbildung 4-10
Bewegt sich der Avatar ...

Wo immer der Avatar hingeht, geht auch die Kamera hin.

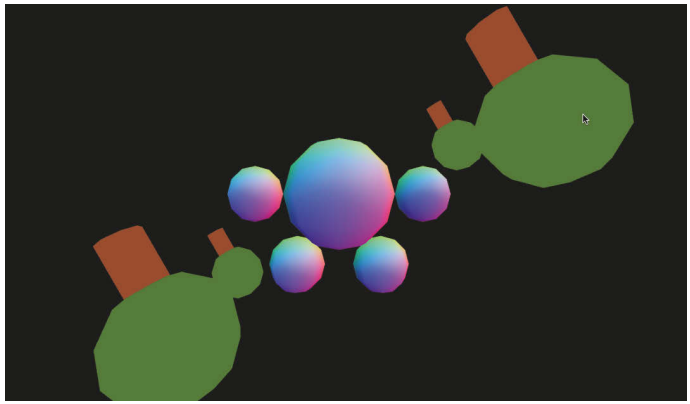


◀ Abbildung 4-11
... dann bewegt sich auch die Kamera.

Ziemlich cool, oder? Na ja, dieses Vorgehen bringt ein Problem mit sich. Was passiert, wenn der Avatar beginnt, Räder zu schlagen oder sich umzudrehen? Probiere es einmal aus (du weißt schon, mit den Tasten C und F)!

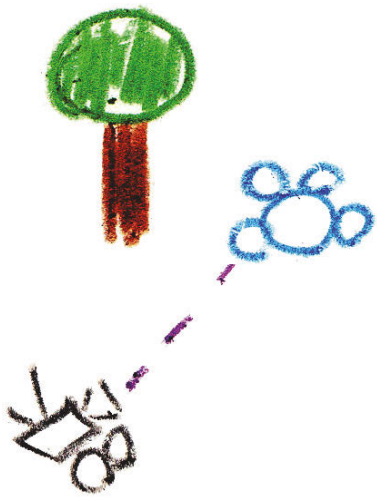
Der Avatar scheint still stehen zu bleiben, alles andere beginnt hingegen, sich zu drehen.

Abbildung 4-12 ▶
Alles dreht sich!



Das liegt daran, dass die Kamera an dem unsichtbaren Selfiestick hängt, der mit dem Avatar verbunden ist. Wenn sich der Avatar dreht, dreht sich die Kamera ebenfalls.

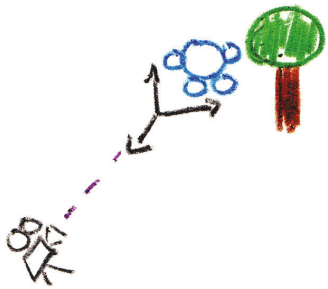
Abbildung 4-13 ▶
Die Kamera dreht sich mit dem Avatar.



Das ist nicht direkt das, was wir wollten. Statt auf den Avatar wollten wir die Kamera auf die *Position* des Avatars fixieren.

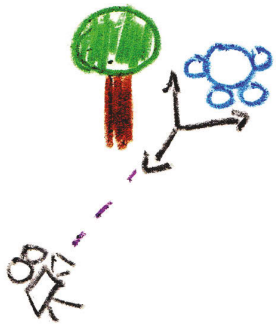
In der 3D-Programmierung gibt es keine einfache Methode, um etwas nur auf die Position einer anderen Sache zu fixieren. Aber noch ist nicht alles verloren.

Wir fügen dem Spiel eine Avatar-Positionsmarke hinzu.



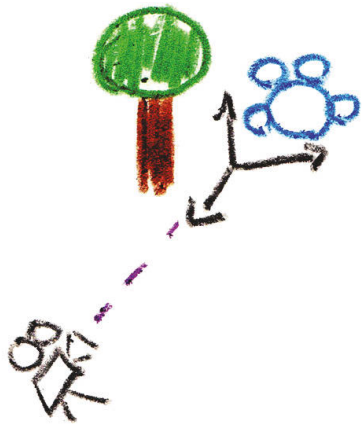
◀ Abbildung 4-14
Kamera und Avatar sind auf eine Marke fixiert.

Wenn wir die Kamera und den Avatar auf diese Marke fixieren, bewegen sich beim Bewegen der Marke sowohl Avatar als auch Kamera.



◀ Abbildung 4-15
Kamera und Avatar bewegen sich immer noch gemeinsam ...

Was aber noch wichtiger ist: Wenn der Avatar ein Rad schlägt, bewegt sich die Kamera nicht. Der Avatar überschlägt sich, die Marke hingegen dreht sich nicht. Da die Marke sich nicht dreht, dreht sich auch die Kamera nicht.



◀ Abbildung 4-16
... doch wenn sich der Avatar überschlägt, macht die Kamera nicht mit.

In der 3D-Programmierung ist diese Marke nur ein Markierungspunkt. Sie sollte unsichtbar sein. Deshalb benutzen wir auch keine Meshes oder geometrischen Körper dafür. Stattdessen verwenden wir `Object3D`. Fügen wir den folgenden Code vor dem Code zur Herstellung des Avatars ein, also direkt hinter `START CODING ON THE NEXT LINE`:

```
var marker = new THREE.Object3D();
scene.add(marker);
```

Jetzt ändern wir den Avatar, sodass er zu `marker` hinzugefügt wird anstatt zu der Szene:

```
var avatar = new THREE.Mesh(body, cover);
marker.add(avatar);
```

Wir müssen außerdem ändern, wie die Kamera hinzugefügt wird. Statt zu dem Avatar fügen wir die Kamera der Marke hinzu.

```
marker.add(camera);
```

Zuletzt müssen wir noch den Tastatur-Event-Listener ändern. Jetzt ändern wir nicht mehr die Position des Avatars, sondern die Position der Marke.

```
document.addEventListener('keydown', sendKeyDown);
function sendKeyDown(event) {
  var code = event.code;
  if (code == 'ArrowLeft') marker.position.x = marker.position.x - 5;
  if (code == 'ArrowRight') marker.position.x = marker.position.x + 5;
  if (code == 'ArrowUp') marker.position.z = marker.position.z - 5;
  if (code == 'ArrowDown') marker.position.z = marker.position.z + 5;

  if (code == 'KeyC') isCartwheeling = !isCartwheeling;
  if (code == 'KeyF') isFlipping = !isFlipping;
}
```

Nun können wir die Position des Avatars mit der Tastatur verschieben, doch wenn wir ihn sich drehen oder ein Rad schlagen lassen, bleibt die Kamera aufrecht stehen.

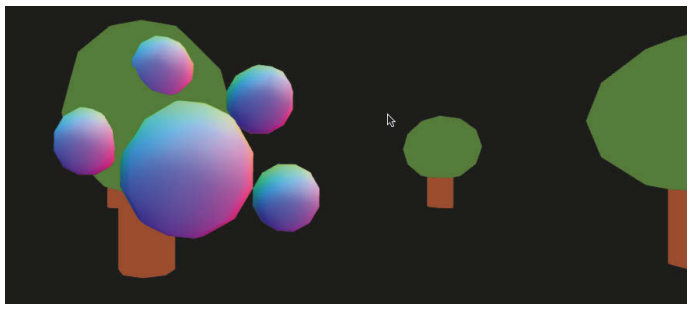


Abbildung 4-17 ▶
Der Avatar schlägt ein Rad, und wir schauen zu.

Der Code bisher

Wenn du deinen Code noch einmal überprüfen möchtest, vergleiche ihn mit dem Code im Abschnitt *Code: Avatare bewegen* auf Seite 294.

Wie es weitergeht

Wir haben in diesem Kapitel eine sehr wichtige Fertigkeit behandelt. Dinge wie Tastatur-Events sind in JavaScript außerordentlich wichtig. Wir werden Events später noch einmal begegnen. Und während sich unsere Kenntnisse in der Spieleprogrammierung weiterentwickeln, werden wir immer wieder Objekte miteinander gruppieren. Das Gruppieren vereinfacht das gemeinsame Bewegen sowie das Verdrehen, Wenden, Wachsen und Schrumpfen von Dingen.

Falls du jetzt gleich mit unserem Avatar weiterarbeiten willst, blättere vor zu Kapitel 6, *Projekt: Hände und Füsse bewegen*. Vergiss aber nicht, später wieder zum jetzt folgenden Kapitel zurückzukehren, in dem wir uns genauer mit JavaScript-Funktionen befassen. Wir haben Funktionen bereits für den Wald, zum Animieren und zum Lauschen auf Events eingesetzt. Man kann aber noch viel mehr über Funktionen lernen. Und sollte dich das immer noch nicht neugierig auf Funktionen gemacht haben, dann vielleicht dieses hier: Wir stellen 100 Planeten her und Flugsteuerungen, um sie umherfliegen zu lassen.

Ehrlich, das ist ziemlich cool.