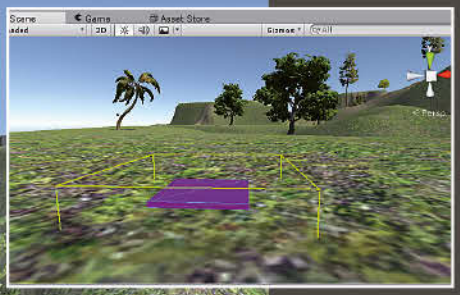
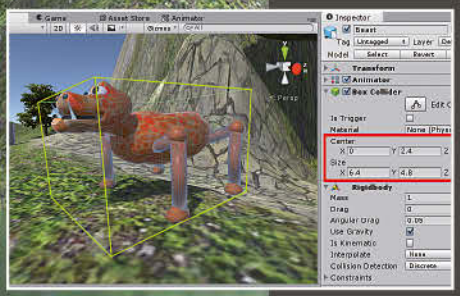
mitp

Hans-Georg Schumann





Unity-Workshop

Spiele-Welten mit Unity, MakeHuman und Blender erstellen

Inhaltsverzeichnis

Kapitel 1

Projekt »Atmo-Sphere«

Hier erstellen wir ein Basis-Projekt in Unity. Es lässt sich für viele andere Spiel-Projekte einsetzen. Anstatt bei jeder Idee ein völlig neues Projekt zu beginnen, können Sie Ihre Idee auch auf diesem Basis-Projekt aufbauen.

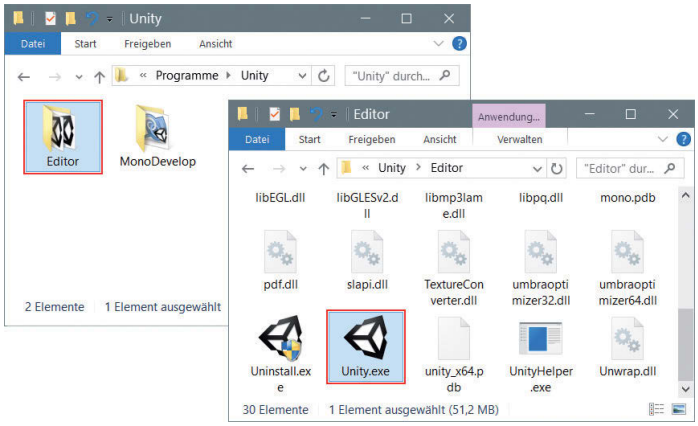
Was erwartet Sie?

Zuerst lernen Sie die wichtigsten Bestandteile der Unity-Arbeitsumgebung kennen. Sie erfahren, wie man Spiel-Objekte einfügt und bearbeitet, was eine Szene ist, wozu man (mindestens) ein Script braucht und wie man es erstellt. Außerdem lernen Sie, wie man eine Spielfigur mit den Tasten steuert.

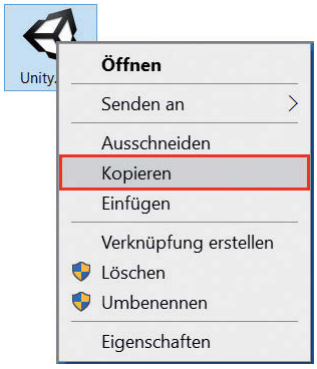
1.1 Einstieg in Unity

Natürlich sollte das Game-Entwicklungssystem Unity bereits installiert sein. Wie das geht, steht im Anhang. Weil Sie Unity nicht nur einmal, sondern wohl sehr oft starten (müssen), ist es sinnvoll, sich gleich ein Icon auf dem Desktop anzulegen. Die folgenden Schritte beschreiben, wie das geht. (Wenn Sie allein zurechtkommen, überspringen Sie einfach Schritt 1 bis 7.)

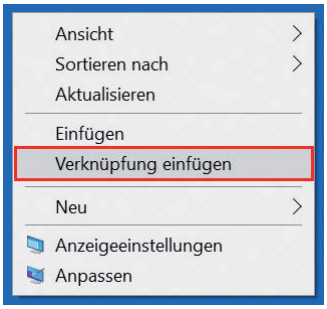
1. Öffnen Sie den Ordner, in dem Sie Unity bei der Installation untergebracht haben (bei mir ist das der Unterordner UNITY im Ordner PROGRAMME auf Laufwerk C:).



2. Wechseln Sie in den Unterordner EDITOR.
3. Suchen Sie das Symbol mit dem Namen UNITY.EXE (sieht ähnlich aus wie ein schwarz-weißer Würfel), klicken Sie mit der rechten Maustaste darauf.



4. Wählen Sie im Kontextmenü den Eintrag KOPIEREN.
5. Klicken Sie nun auf eine freie Stelle auf dem Desktop, ebenfalls mit der rechten Maustaste.



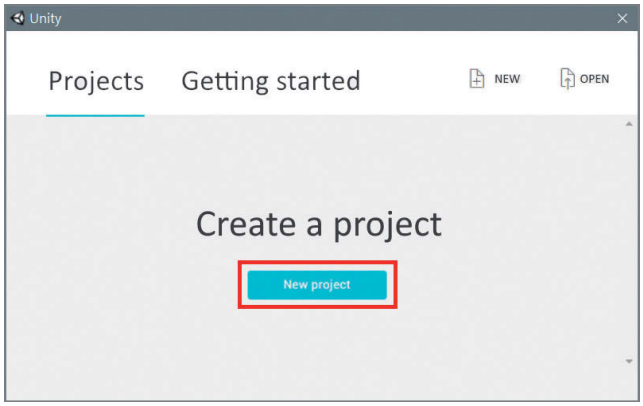
6. Wählen Sie im Kontextmenü den Eintrag VERKNÜPFUNG EINFÜGEN.
7. Geben Sie dem neuen Symbol den Namen UNITY.

Von nun an starten Sie Unity, indem Sie auf das neue Symbol doppelklicken.

Je nach Computer kann es eine Weile dauern, bis Unity geladen ist. Einige Zeit später finden Sie sich in einem neuen Fenster wieder.

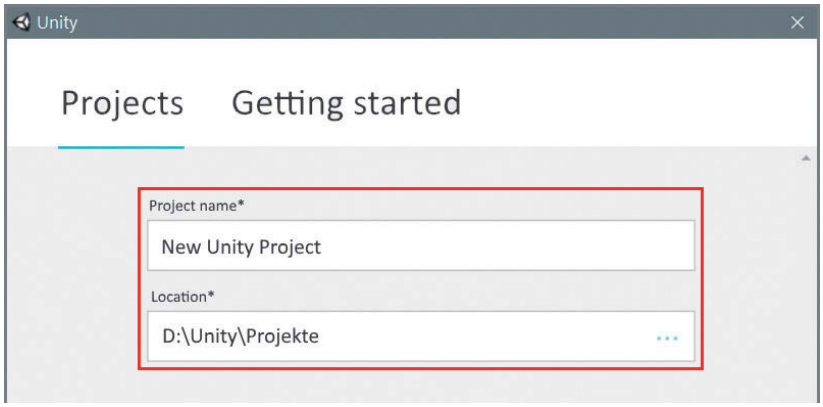
Schon angemeldet?

Wird Unity zum allerersten Mal nach der Installation gestartet, benötigen Sie in der Regel noch einen Account. Folgen Sie dazu einfach den Anweisungen und denken Sie sich ein Kennwort aus. Beim nächsten Start von Unity landen Sie direkt im Start-Dialog.



Angeboten werden oben rechts zwei Möglichkeiten: OPEN – ein Projekt zu öffnen (wenn vorhanden). Oder NEW – ein neues zu erstellen. Das geht hier auch über eine Schaltfläche in der Mitte unter dem Text CREATE A PROJECT.

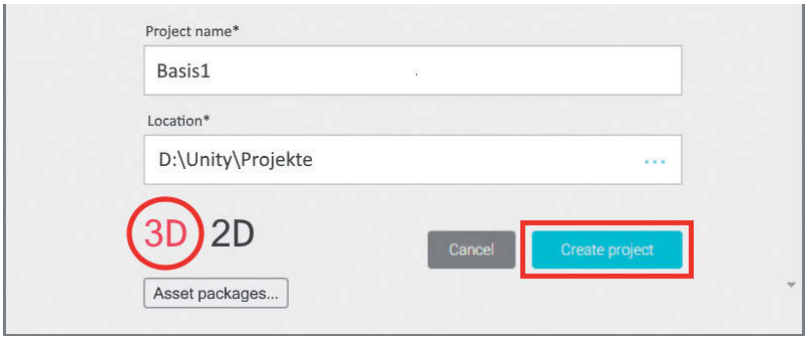
1. Klicken Sie auf NEW PROJECT.



2. Geben Sie im Feld unter PROJECT NAME den Namen des neuen Projekts ein. Und dann bei LOCATION den Ordner, in dem Sie Ihr neues Projekt unterbringen wollen. (Wenn Sie die Vorgabe stehen lassen, schafft sich Unity seinen eigenen Ordner für die Spiel-Projekte.)

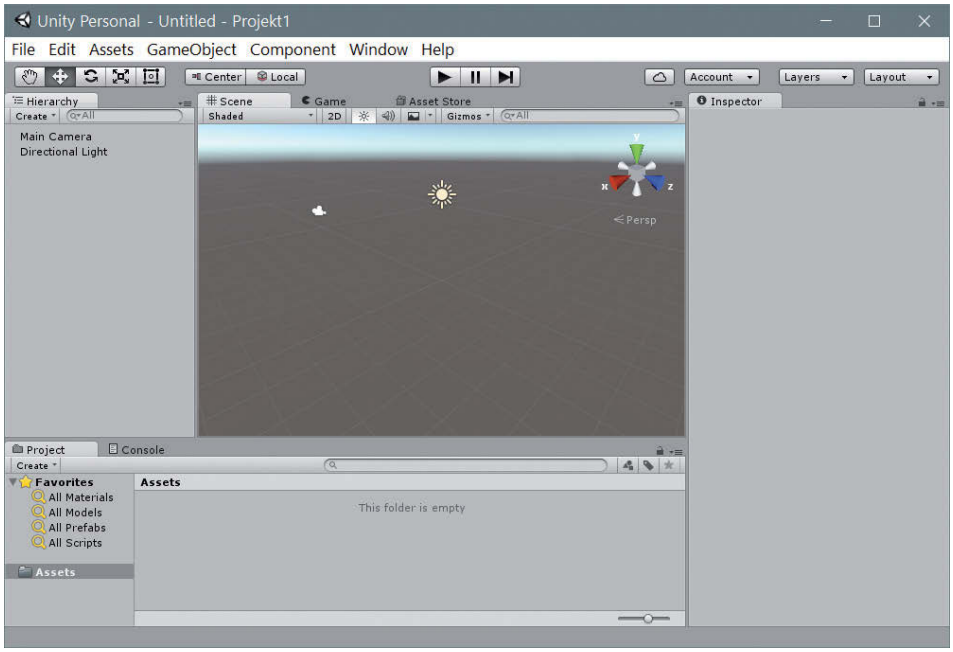
Projekt-Ordner

Ich benutze einen Ordner UNITY, in dem ich einen Unterordner PROJEKTE erstellt habe. Dieses Basis-Projekt könnte man schlicht und einfach BASIS1 oder BASIC1 nennen.

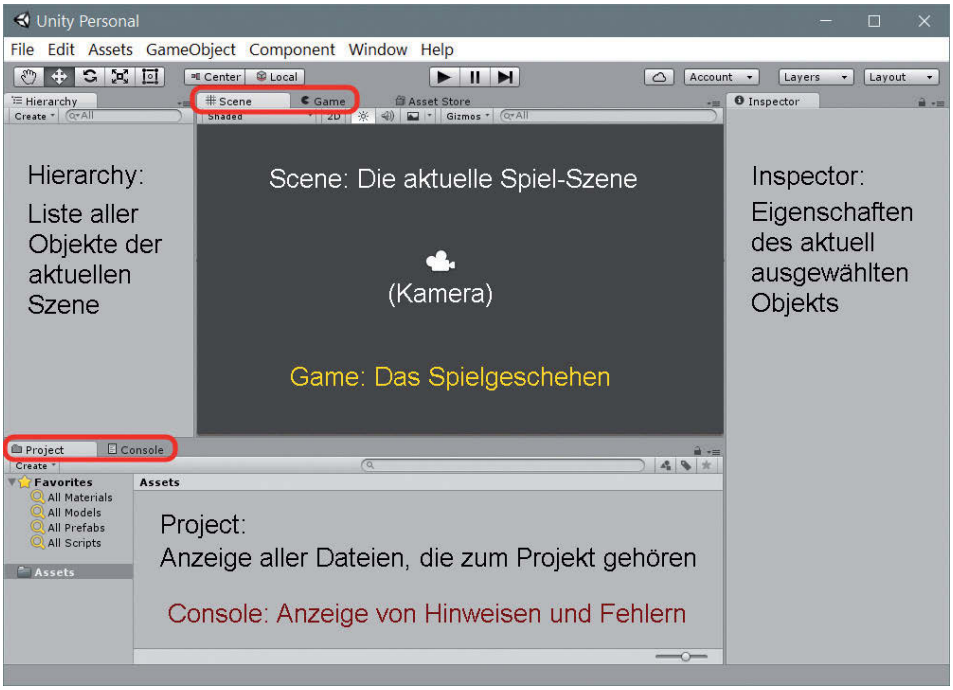


3. Achten Sie darauf, dass die Anzeige 3D rot und 2D grau aussieht. Bestätigen Sie Ihre Einstellungen mit einem Klick auf CREATE PROJECT.

Nach einer Weile zeigt sich Unity mit mehreren Fensterbereichen.



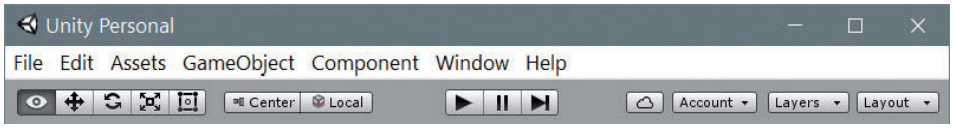
Die wichtigsten sind so aufgeteilt (dabei lassen sich zwei jeweils über Reiter zu einem zusätzlichen Fenster umschalten):



- Im SCENE-Fenster sieht man erst einmal nur die Kamera und eine Lichtquelle. Für ein Spiel brauchen wir dann noch mindestens ein weiteres Objekt wie eine Kugel oder eine Figur.
- Dahinter liegt das GAME-Fenster: Hier können Sie Ihr Spiel in Echtzeit ablaufen sehen, wenn es durch eine der darüberliegenden kleinen Schaltflächen gestartet wurde (die mit dem Dreieck).
- Im HIERARCHY-Fenster ist das Wichtigste bis jetzt die MAIN CAMERA. Dort stehen dann später alle weiteren Objekte, die zur Szene eines Spiels gehören. (Jedes Spiel kann natürlich mehrere Szenen haben.)
- Das PROJECT-Fenster erfasst die Ordner mit dem gesamten Zubehör für das ganze Spiel. Dazu gehören natürlich u.a. auch Programmteile. Bilder, die als Spielobjekt eingesetzt werden sollen (wie z.B. eine Kugel oder eine Figur), lassen sich einfach mit der Maus aus einem Ordnerfenster unter Windows hier hineinziehen. Womit die entsprechende Datei ins Projekt kopiert wird.
- Dahinter liegt das CONSOLE-Fenster, das sich bei Fehlern meldet. Außerdem lassen sich dort Werte z.B. von Spiel-Objekten anzeigen.

- Damit Sie die Eigenschaften eines Objekts nicht nur anschauen, sondern auch bearbeiten können, gibt es das INSPECTOR-Fenster.

Mit dem Hauptmenü und vielen der darunterliegenden Symbole bekommen Sie es nicht jetzt, aber später noch genug zu tun.

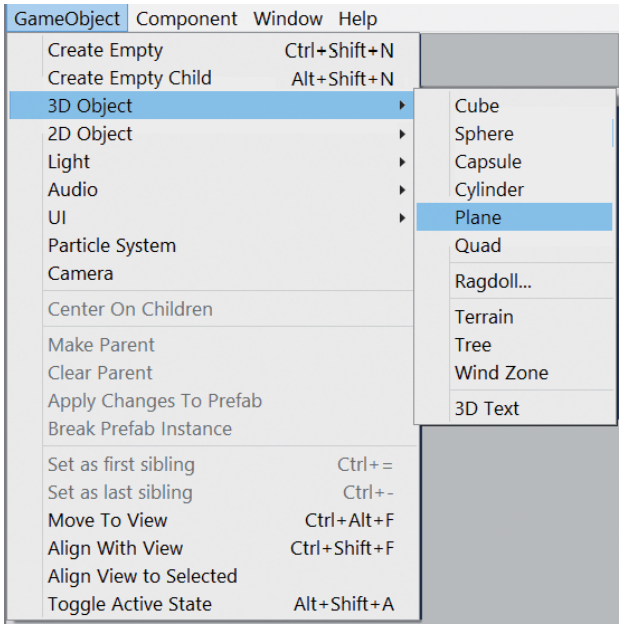


1.2 Spielfeld und Spielfigur

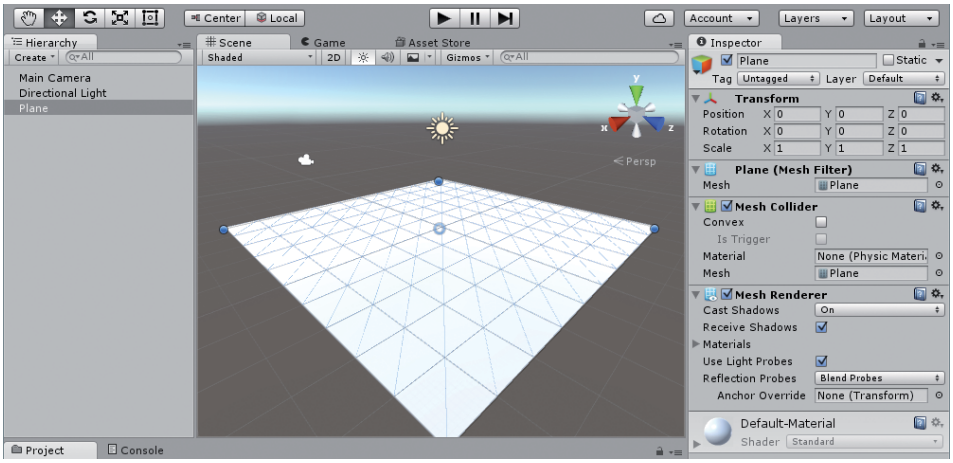
Das einfachste Spiel besteht aus zwei Elementen, in Unity auch GAMEOBJECT genannt: dem Spielfeld oder der Spiele-Welt und einer Figur, die natürlich auch ein Gegenstand sein kann – je nach Art des Spiels.

Unity spendiert dazu noch eine Kamera und die passende Beleuchtung. Beide Elemente sind schon in der HIERARCHY-Liste aufgeführt. Außerdem können Sie die Symbole für diese beiden Objekte auch im SCENE-Fenster sehen. Und so fügen Sie weitere Objekte hinzu:

1. Klicken Sie oben in der Menüleiste auf GAMEOBJECT und dann im Menü auf 3D OBJECT. Im Zusatzmenü wählen Sie den Eintrag PLANE.

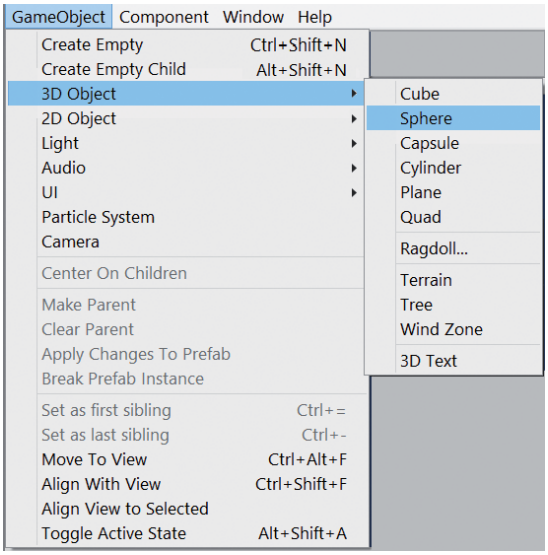


Anschließend erscheint im SCENE-Fenster sofort das neue Objekt, unser *Spielfeld*. Genauer ausgedrückt ist es hier nichts weiter als eine Ebene oder ein Rechteck. Zusätzlich tauchen rechts im INSPECTOR-Bereich zahlreiche Informationen über das neue Objekt auf. Damit beschäftigen wir uns später.

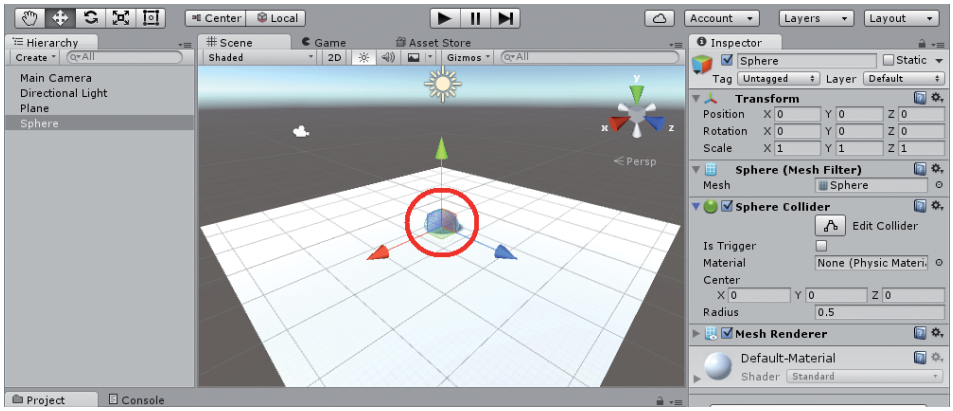


Damit das Spielfeld nicht lange so einsam bleiben muss, brauchen wir jetzt eine *Spielfigur*, die sich bewegen soll. Dies kann natürlich eine echte Figur sein, hier im Basis-Projekt aber soll uns ein Ball bzw. eine Kugel genügen.

2. Klicken Sie erneut auf GAMEOBJECT und auf 3D OBJECT. Diesmal wählen Sie im Zusatzmenü den Eintrag SPHERE.

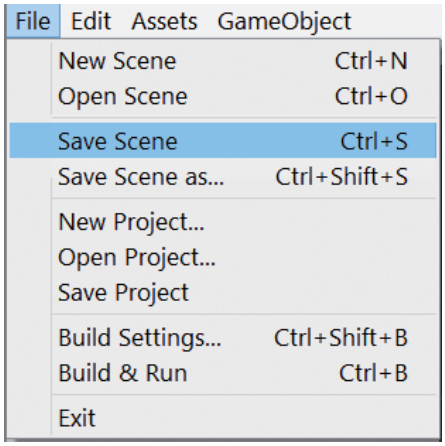


Wir haben nun die Basiselemente eines Spiels, wobei die Typen PLANE und SPHERE natürlich nur eine Ersatzrolle spielen. Sie lassen sich (später) durch beliebige andere Objekte ersetzen.

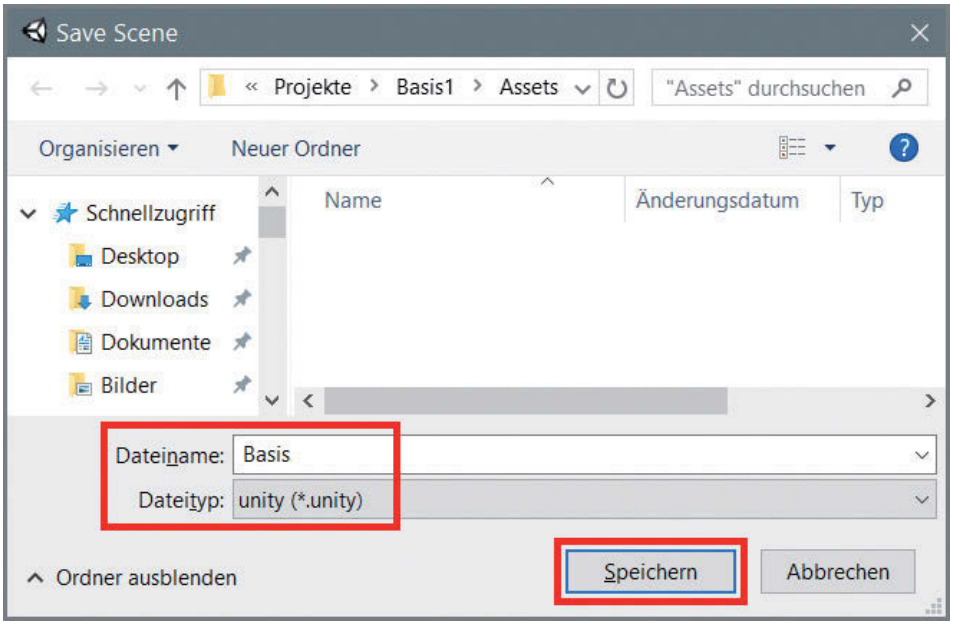


Sehr genau kann man das neue Objekt nicht erkennen, aber bald sieht man mehr. Denn wir werden unsere »Spielfigur« natürlich noch bearbeiten. Zuvor aber sollten wir die ganze Szene unbedingt erst einmal sichern.

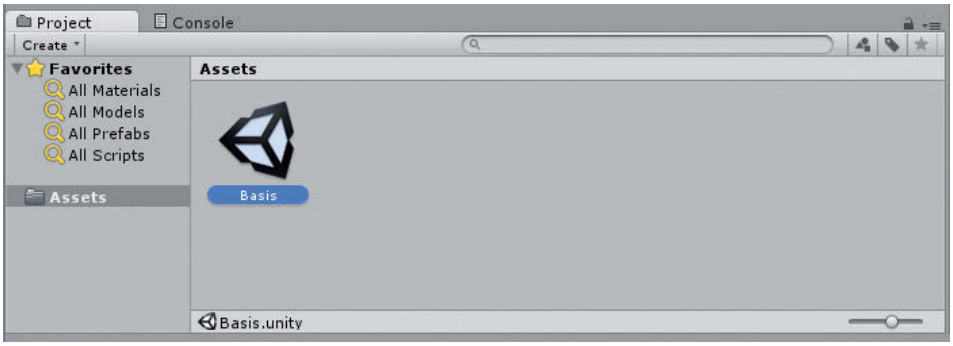
3. Klicken Sie dazu auf FILE und dann auf SAVE SCENE. (Alternativ geht das auch mit Strg + S.)



4. Im Dialogfeld SAVE SCENE können Sie der Szene einen Namen geben (der sich später auch wieder ändern lässt). Klicken Sie abschließend auf SPEICHERN.



Nun gibt es im PROJECT-Fenster unter ASSETS ein neues Element. Die Dateikennung für eine Unity-Szene ist (logischerweise) UNITY.



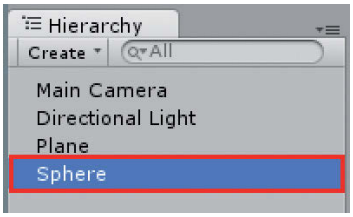
Assets – was sind das?

Darunter versteht man in Unity sämtliche Zutaten zu einem Spiel. Wie Sie unter dem Eintrag FAVORITES sehen, gehören dazu u.a. Materialien und alle Spiel-Objekte, die in unserem Fall in der Szene-Datei untergebracht sind. Und eine Menge anderer Elemente, die Sie im Laufe dieses und der nächsten Kapitel noch kennenlernen werden.

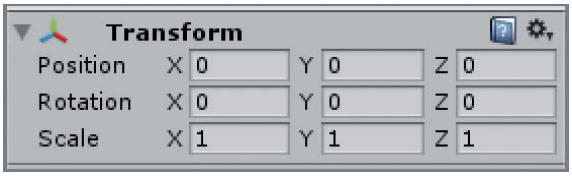
1.3 Position – Rotation – Scale

Die Ebene (= das Spielfeld) und die Kugel (= die Spielfigur) hat Unity beim Erzeugen in der sogenannten Welt-Mitte positioniert.

1. Klicken Sie im HIERARCHY-Fenster auf den Eintrag SPHERE, sodass er markiert ist.

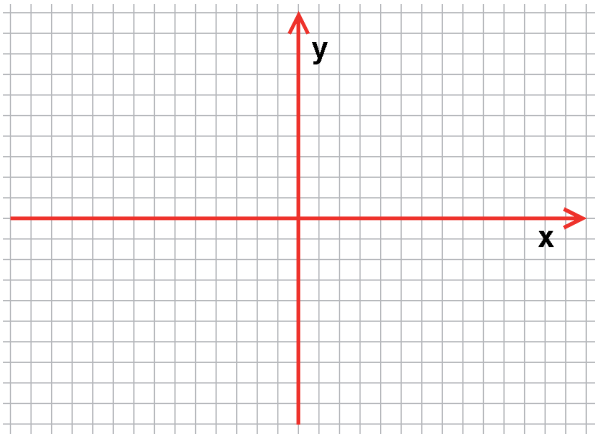


Nun sehen Sie im INSPECTOR-Fenster die zugehörigen Eigenschaften. Uns interessiert jetzt nur das, was unter TRANSFORM steht.



Weil es sich bei der Kugel (Typ `Sphere`) um ein dreidimensionales Objekt handelt, sind dazu auch drei Koordinaten nötig, um Lage und Größe des Objekts eindeutig festzulegen.

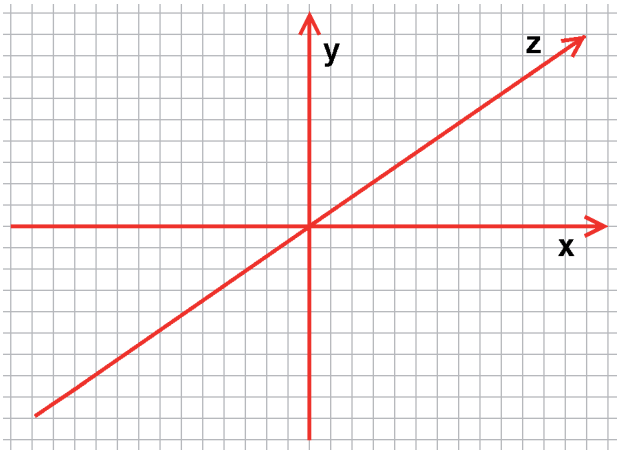
Im klassischen Koordinatensystem bildet die x-Achse die Horizontale (Waagerechte) und die y-Achse die Vertikale (Senkrechte) ab.



Entlang der x-Achse geht es also nach links oder rechts, entlang der y-Achse nach oben oder unten. Auch in Unity befindet sich der Ursprung genau in der Mitte der Spiel-Welt. Der 2D-Punkt dort hat die Koordinaten (0 | 0).

Weil Spiele in Unity in der Regel 3D-Spiele sind, genügen zwei Achsen nicht. Sondern es gibt noch eine dritte, z-Achse genannt. An der entlang geht es nach vorn oder nach hinten.

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.



Nehmen wir jetzt die Einstellungen unter TRANSFORM einmal genauer unter die Lupe.

	X	Y	Z	mögliche Aktion
POSITION	0	0	0	Verschieben
ROTATION	0	0	0	Drehen
SCALE	1	1	1	Größe ändern

Das bedeutet: Die Position der Kugel ist genau in der Welt-Mitte, einem 3D-Punkt mit den Koordinaten (0 | 0 | 0). Es gibt keine Verdrehung (was man bei einer Kugel ohnehin nicht bemerken würde), und der Durchmesser der Kugel beträgt eine Längeneinheit. Unity misst in Metern, also haben wir es mit einem 1 m »dicken« 3D-Objekt zu tun.

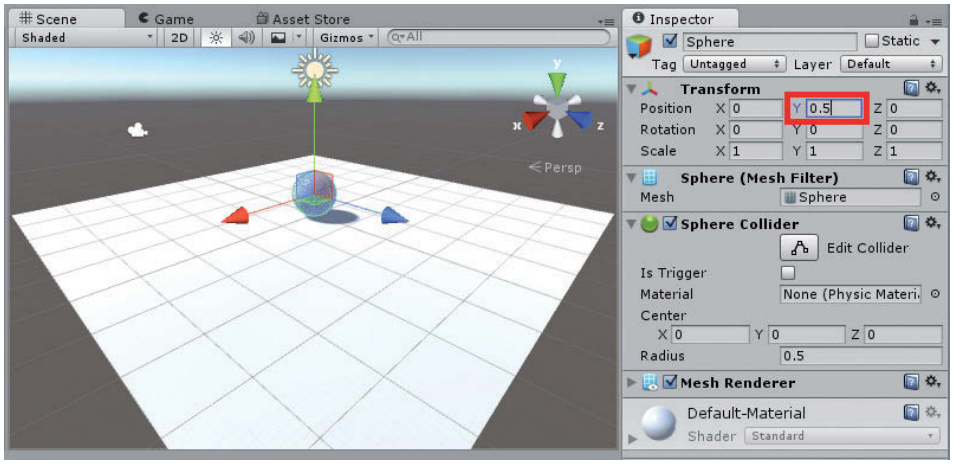
Basis-Ebene

Wenn Sie wollen, können Sie einmal zum Element PLANE umschalten. Auch da finden Sie im INSPECTOR-Fenster die gleichen Werte. Nur steht hier die 1 bei SCALE jeweils für 10 m. Eine Ebene ist also in Unity direkt nach der Erzeugung erst einmal 10 m lang und 10 m breit (und hat übrigens keine Höhe oder »Dicke«).

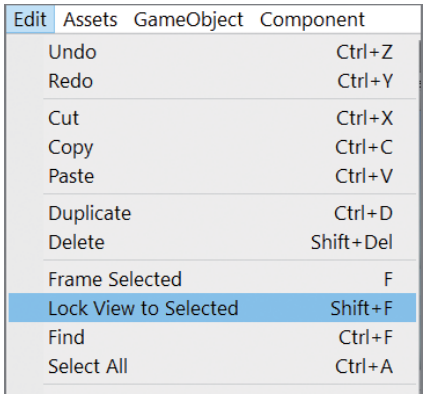
Wir lassen die Vorgabemaße von Unity weitgehend so stehen, aber nicht ganz, denn wenn Sie genau hinschauen, dann liegt die Kugel jeweils zur Hälfte oberhalb und unterhalb der Ebene. Wir wollen aber, dass sich die Spielfigur *auf* dem Spielfeld befindet. In unserem Fall soll die Kugel später ja auf der Ebene herumrollen. Deshalb müssen wir bei der Position eine Änderung vornehmen.

2. Klicken Sie im INSPECTOR-Fenster hinter POSITION auf das Y-Feld und geben Sie **0,5** (oder **0.5**) ein.

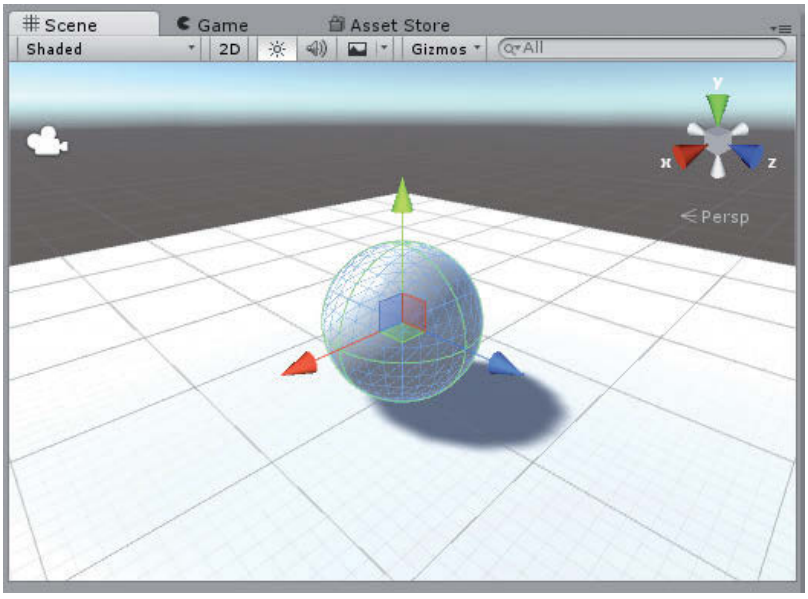
Sofort liegt die Kugel *auf* der Ebene und wirft sogar einen passenden Schatten.



3. Um sich das Ganze mehr aus der Nähe anzusehen, klicken Sie im SCENE-Fenster auf die Kugel. Dann klicken Sie auf EDIT und wählen LOOK VIEW TO SELECTED. Oder Sie drücken die Tastenkombination ⇧+F.



Und unsere »Spielfigur« rückt ein ganzes Stück näher.



Aufgabe 1.1

Speichern Sie die Szene zuerst einmal wieder. Dann ändern Sie die Kugelgröße z. B. auf 3 m Durchmesser und sorgen dafür, dass sie weiterhin *auf* der Ebene liegt. Probieren Sie ruhig auch andere Größen aus. Beachten Sie, was passiert, wenn Sie unter SCALE nur **einen** Wert ändern.

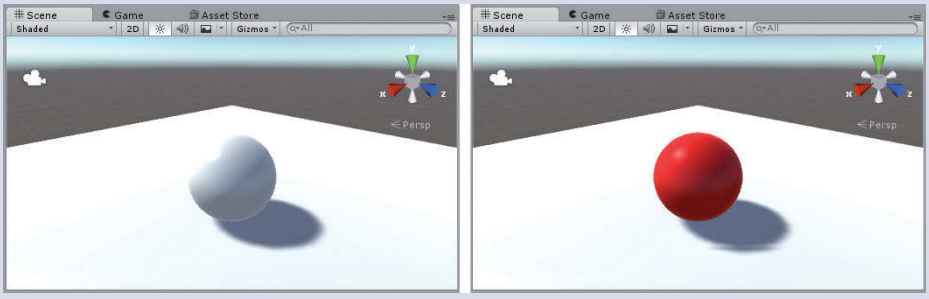
Mit welcher Kugelgröße Sie arbeiten wollen, entscheiden Sie. Und ist Ihnen die Spielfläche zu klein, dann können Sie auch deren Größe ändern.

1.4 Material und Farbe

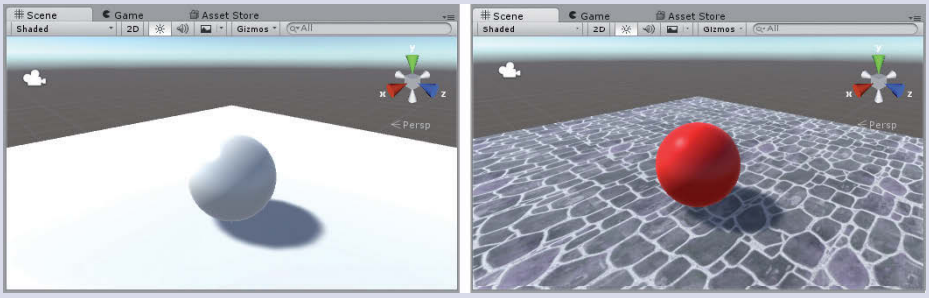
Bevor wir Bewegung ins Spiel bringen, wollen wir erst beiden Objekten ein etwas bunteres Aussehen verpassen. Das geht aber nicht direkt, sondern wir müssen dazu einen kleinen Umweg machen.

Fleisch und Haut

Jedes Objekt ist zunächst optisch »nackt«. Das heißt, es wird im SCENE-Fenster (und im GAME-Fenster) von Unity weiß bis grau dargestellt, wenn es beleuchtet ist.



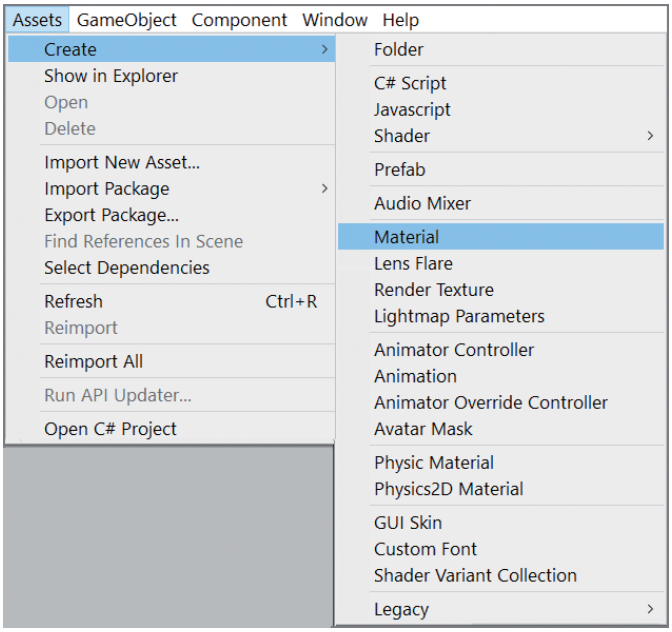
Will man das Objekt einfärben, so benötigt man zuerst ein *Material*. Das lässt sich direkt einfärben oder man kann darauf ein Bild bzw. Muster auftragen, die sogenannte *Textur*. Unity sorgt dafür, dass sich die Textur wie eine Haut an das Objekt anschmiegt.



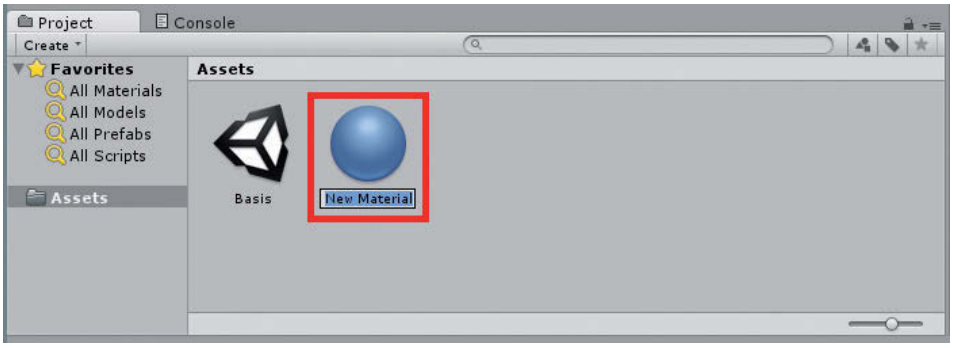
Während das Material also quasi das Fleisch ist, das natürlich schon eine Farbe haben kann, macht die Textur als Haut die eigentliche Struktur aus. Material allein kann immer nur einfarbig sein. Echte Haut ist ja auch nicht durchgehend einfarbig, die Textur ist dann sozusagen die Musterung.

Verpassen wir nun der Spielfigur etwas Farbe.

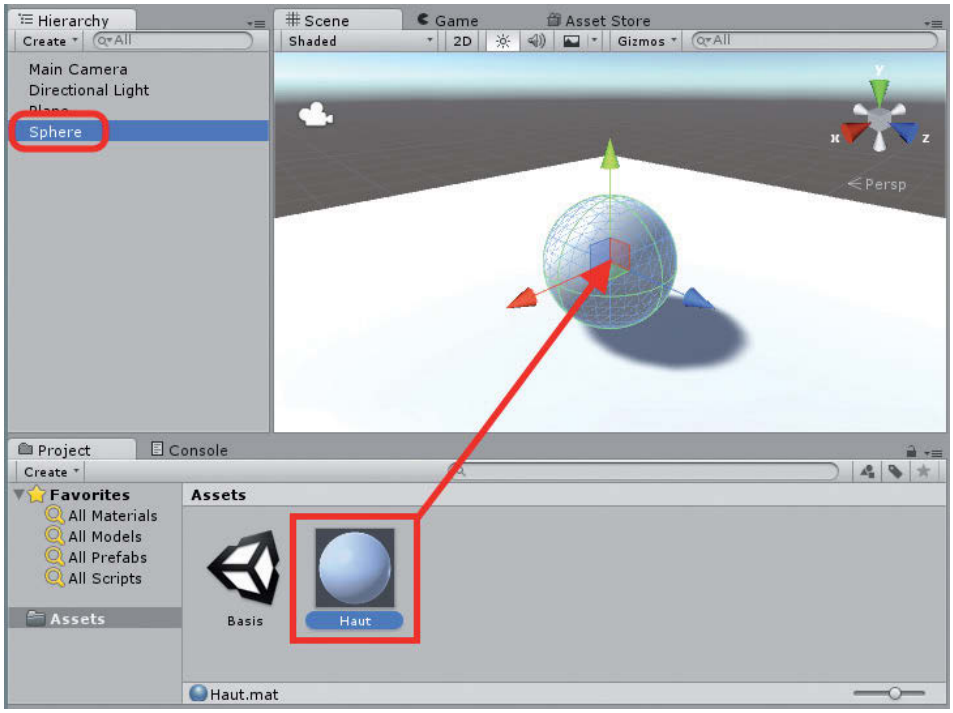
1. Klicken Sie im Hauptmenü auf ASSET und dann auf CREATE. Wählen Sie im Zusatzmenü den Eintrag MATERIAL.



Im PROJECT-Fenster taucht nun ein neues Symbol auf. Der Name NEW MATERIAL ist markiert und kann geändert werden.

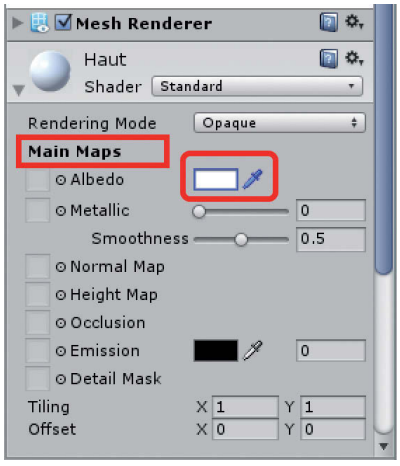


2. Geben Sie dem Material einen neuen Namen, z. B. **Haut** oder einfach nur **Material1**.
3. Ziehen Sie das Material-Symbol mit der Maus direkt auf das *markierte* Objekt im SCENE-Fenster.

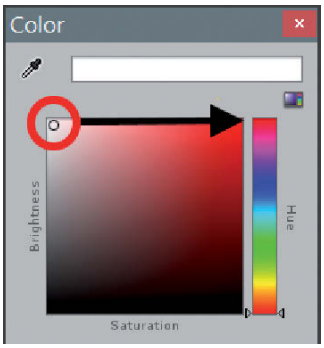


Nun hat die Kugel – unsere Spielfigur – ihr »Fleisch«. Auf eine Textur können wir hier (noch) verzichten. Es reicht, wenn wir der Kugel eine durchgehende Farbe geben.

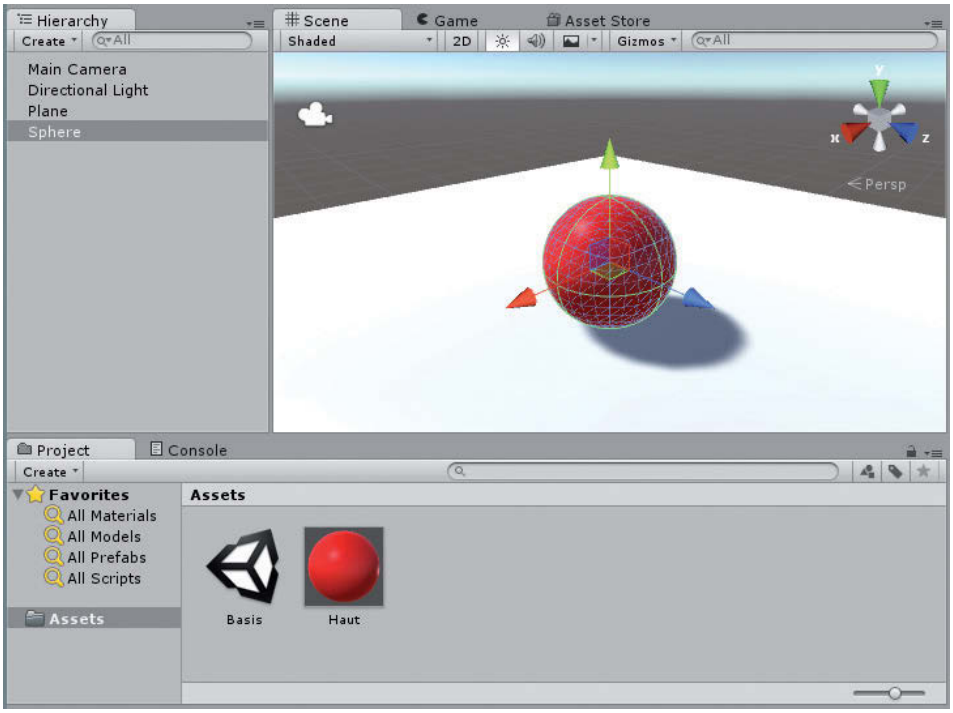
4. Suchen Sie im INSPECTOR-Fenster den Namen, den Sie dem Material gegeben haben (bei mir ist es HAUT). Klicken Sie auf den Bereich, sodass darunter die zugehörigen Werte zu sehen sind.



5. Klicken Sie auf die kleine Fläche rechts unter MAIN MAPS. Damit öffnen Sie ein Auswahlfeld für die Farbe.



6. Voreingestellt ist Weiß. Im Farbfeld können Sie nun mit der Maus herumrühren, um sich eine Farbe zu mischen. Ich schlage **Rot** vor, dazu ziehen Sie den (sehr) kleinen weißen Kreis einfach direkt nach rechts. Anschließend machen Sie das Fensterchen mit Klick auf das X rechts oben wieder zu.



Im SCENE-Fenster erwartet Sie nun eine farbige Kugel (in meinem Falle rot). Und auch das Material hat eine neue Farbe.

7. Speichern Sie die Szene über FILE und SAVE SCENE bzw. mit Strg+S.

1.5 Material und Textur

Eigentlich würde das genügen, denn das Spielfeld darf natürlich weiß bleiben (später lässt es sich ja z. B. durch eine Landschaft austauschen). Aber schöner wäre es schon, wenn die weiße Ebene nicht nur Farbe, sondern auch eine Textur hätte.

JPG oder PNG

Dazu brauchen Sie allerdings Bild-Dateien vom Format JPG oder PNG. Die sollten nicht etwa Fotos, sondern Muster enthalten.

Das Format JPG lässt sich besser komprimieren, braucht also weniger Speicherplatz, in PNG lässt sich eine Farbe als durchsichtig festlegen.

Breite und Höhe der Bilder sollten ein Vielfaches von 8 sein, also z. B. 128, 256, 512, 1024 Pixel.

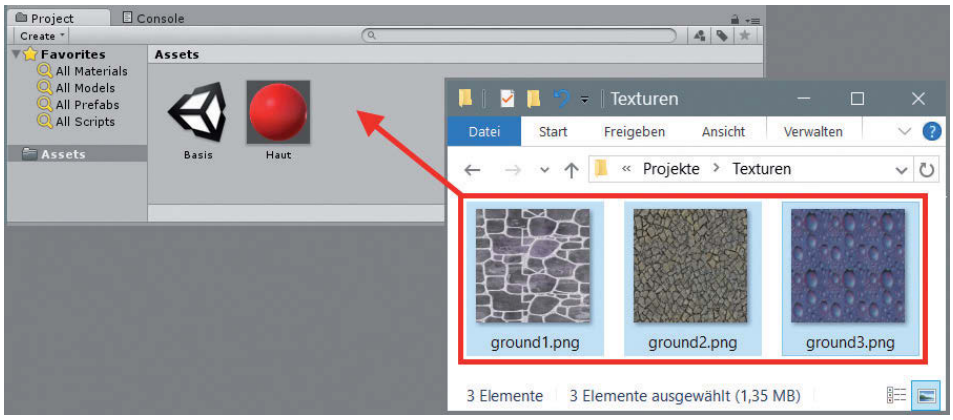
1. Besorgen Sie sich ein paar Bilder mit Texturen, die zu einer Bodenfläche passen könnten.

Download zum Buch

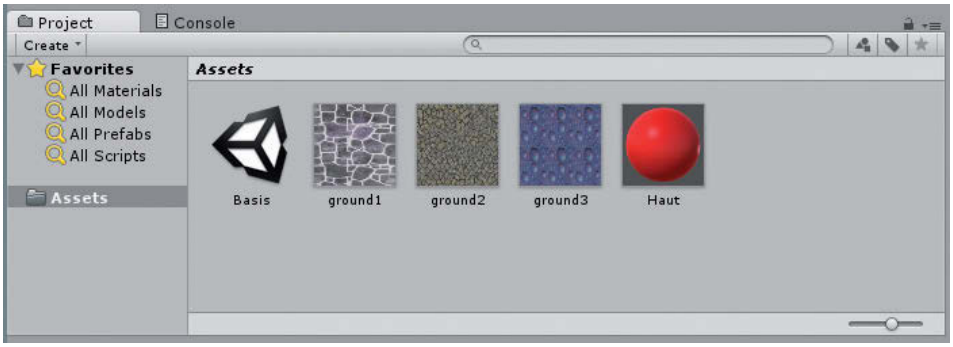
Sie können sich auch im TEXTURES-Ordner bedienen, der zum Download von der *mitp*-Homepage zur Verfügung steht:

`http://www.mitp.de/362`

2. Öffnen Sie den Ordner, in dem Sie die Bild-Dateien untergebracht haben, markieren Sie die Textur oder die Texturen, die Sie für Ihr Projekt verwenden wollen, und ziehen Sie sie mit der Maus ins PROJECT-Fenster von Unity.

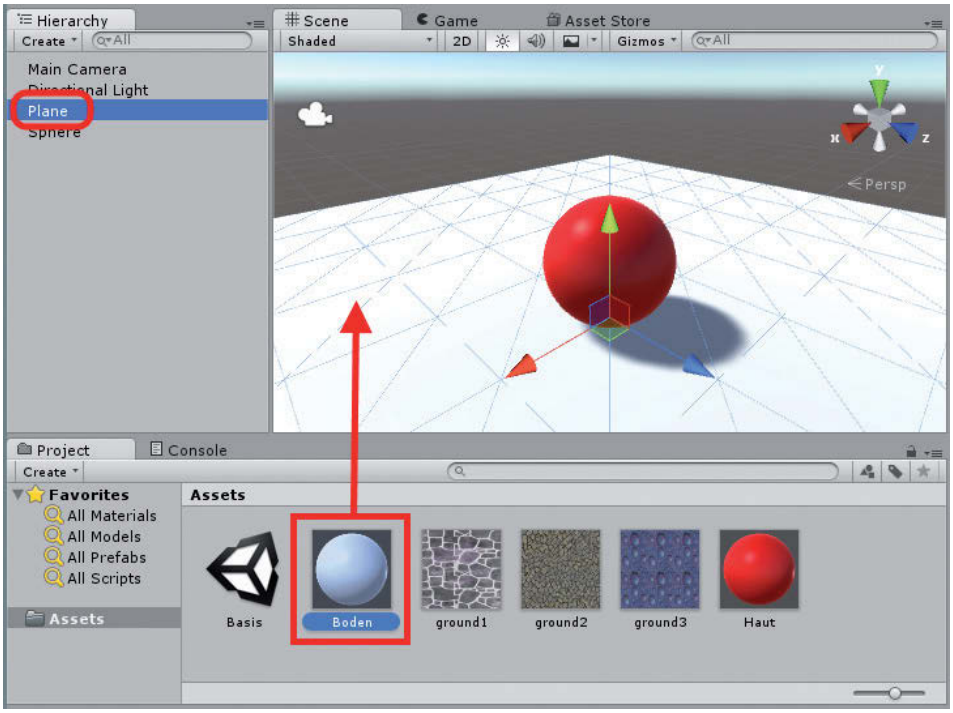


Nun hat sich im ASSETS-Ordner schon einiges an Zutaten angesammelt.

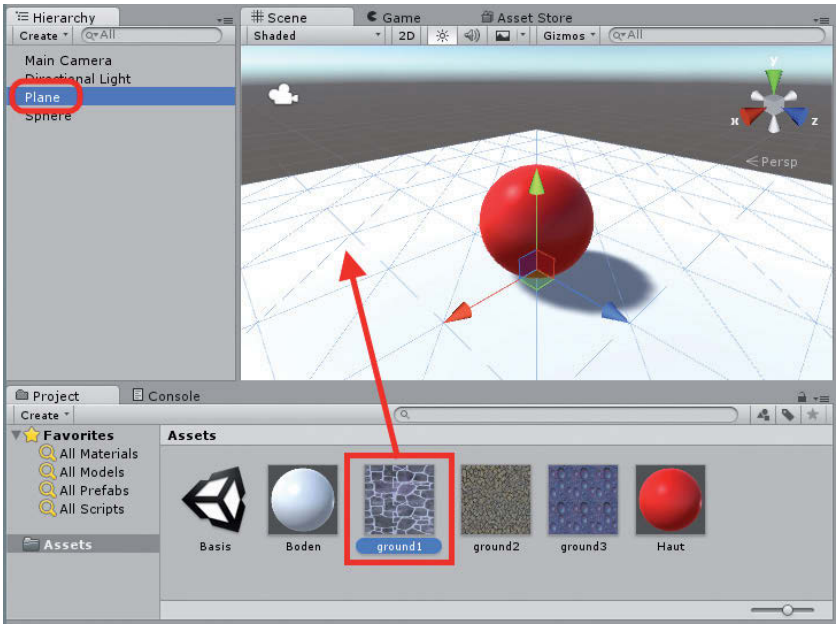


Jetzt sollten Sie gleich ausprobieren, wie Ihrem Spielfeld eine neue »Haut« steht.

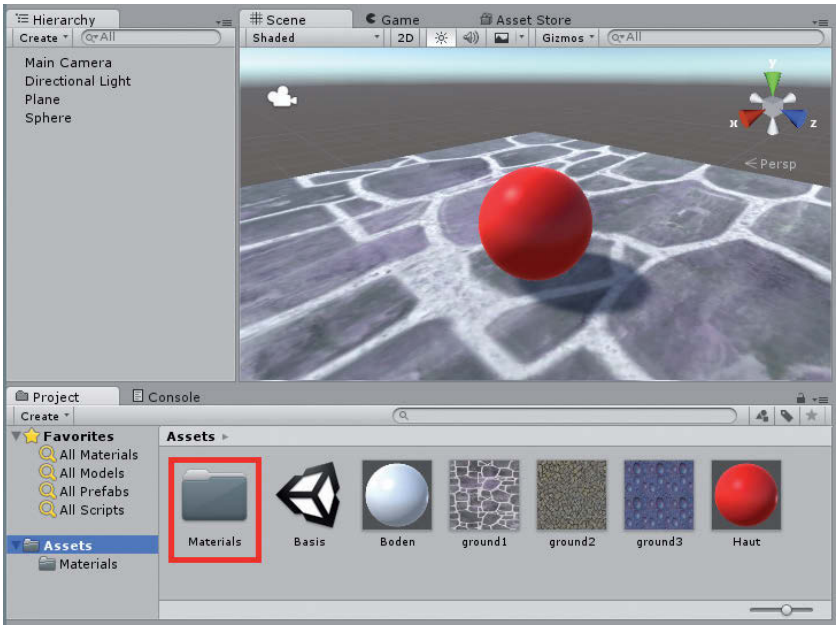
3. Zuerst erzeugen Sie ein weiteres Material, nennen es z. B. **Boden** (oder **Material2**) und weisen es dem Objekt PLANE zu.



4. Anschließend wiederholen Sie die Zuweisung mit einer der Texturen im PROJECT-Fenster.



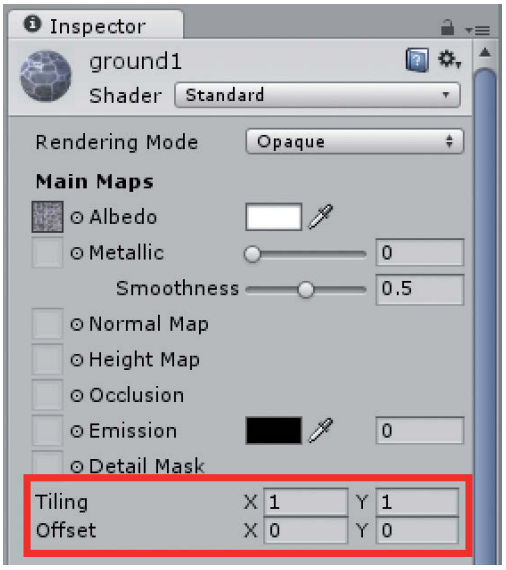
Nun hat das Spielfeld »Fleisch und Haut«. Außerdem gibt es einen neuen Ordner mit dem Namen MATERIALS.



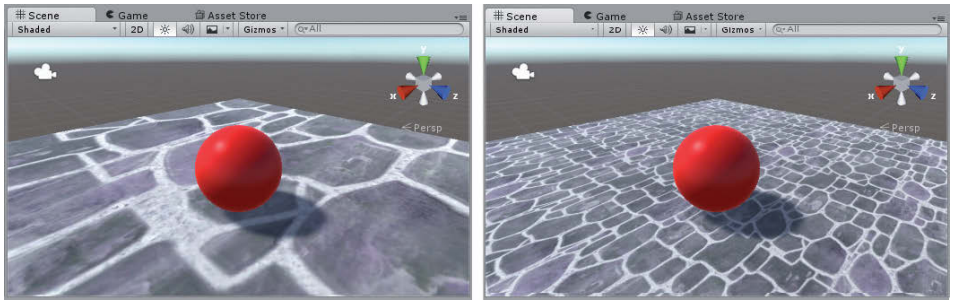
Man könnte nun alles so lassen, aber ich finde die Textur zu großflächig.

5. Wechseln Sie im PROJECT-Fenster in den neuen Ordner MATERIALS. Klicken Sie dort auf das neue Symbol mit dem Namen der Texturdatei (bei mir GROUND1).

6. Im INSPECTOR-Fenster sehen Sie die zugehörigen Einstellungen. Suchen Sie weiter unten den Eintrag TILING. Ändern Sie die 1 in eine größere Zahl, z. B. 5.



Damit ändert sich die Dichte der Textur. Bei 1 wird die Textur einmal über die gesamte Fläche gelegt, bei 5 wird sie dort fünfmal nebeneinander und hintereinander verteilt. Damit ist sie natürlich entsprechend kleiner und die Musterung wird dichter.



7. Zu guter Letzt sollten Sie die Szene mal wieder speichern.

Aufgabe 1.2

Wie wäre es, wenn Sie auch der Kugel eine Textur verpassen? Experimentieren Sie bei beiden Spiel-Objekten mit anderen Farben und Mustern.

1.6 Organisatorisches

Bevor wir uns mit der Steuerung der Spielfigur über das Spielfeld beschäftigen, wollen wir den beiden Objekten einen passenderen Namen geben.

1. Klicken Sie dazu im HIERARCHY-Fenster jeweils auf den Namen der betreffenden Objekte, also SPHERE und PLANE.
2. Dann drücken Sie die Taste F2. Nun können Sie die Namen ändern.

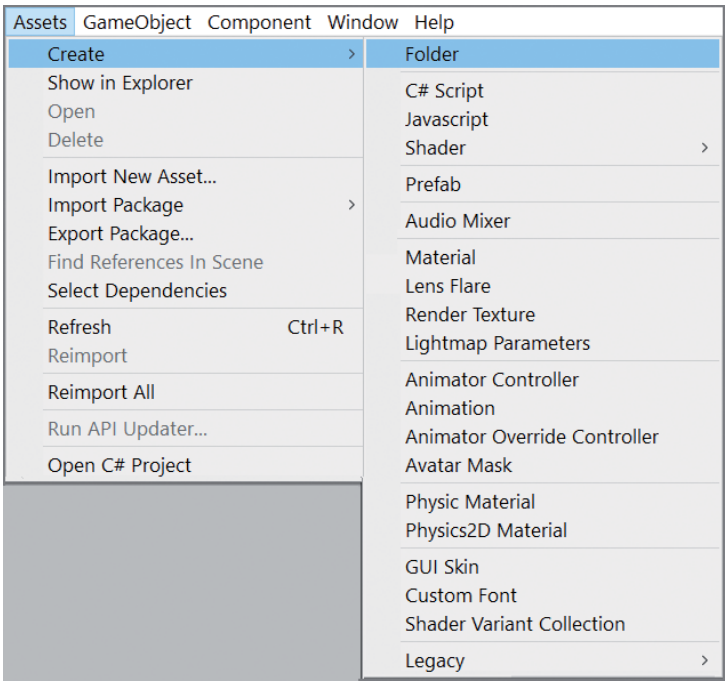
Wie Sie sehen, habe ich mich für englische Begriffe entschieden: **Player** und **Gamezone**, Sie können aber auch **Spielfigur** und **Spielfeld** verwenden – oder was Sie wollen.

Umbenennen und Löschen

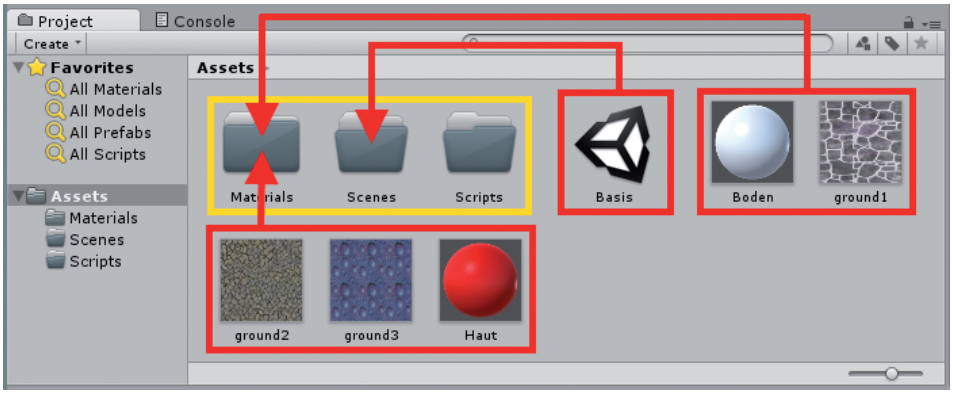
Grundsätzlich kann man in Unity fast alles umbenennen. So auch die Elemente im PROJECT-Ordner (und seinen Unterordnern). Dabei wird die gleiche Taste benutzt wie bei Windows: F2. Sollten Sie mal ein Element löschen wollen, geht das mit Entf. Und mit Strg+Z machen Sie eine Aktion wieder rückgängig.

Da wir gerade beim Ordnen sind: Unity hat es uns vorgemacht, indem es für das Material einen eigenen Unterordner in ASSETS erstellt hat. Das tun wir jetzt auch für einige andere Elemente im PROJECT-Fenster.

1. Sorgen Sie dafür, dass Sie im ASSETS-Ordner sind, dann klicken Sie auf ASSETS und CREATE und im Zusatzmenü auf FOLDER.



2. Benennen Sie den Ordner gleich in **Scenes** um.
3. Wiederholen Sie das Ganze für einen weiteren Ordner namens **Scripts**.



4. Und nun verschieben Sie das Symbol für die Szene in den Ordner SCENES, alle anderen Elemente in den Ordner MATERIALS.

Damit sind unsere Aufräumungsarbeiten (erst einmal) abgeschlossen.

Szene verschwunden?

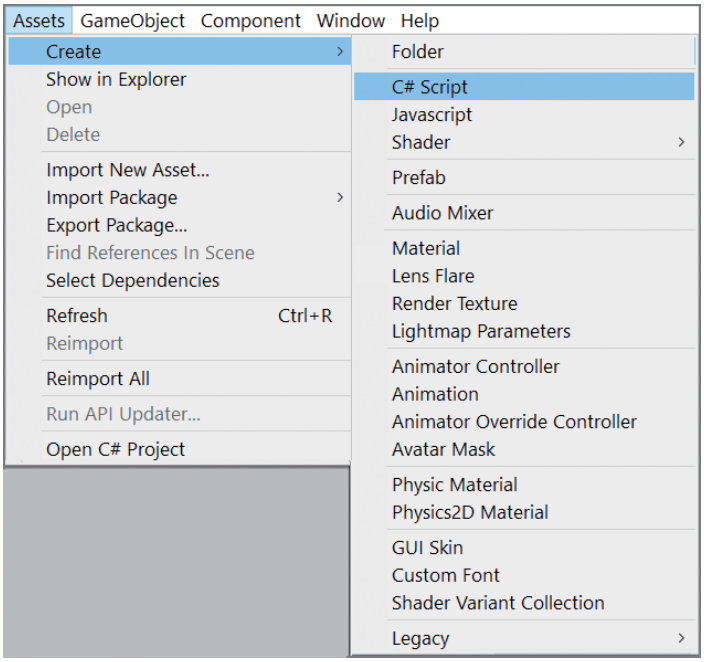
Wenn Sie ein Projekt in Unity bearbeiten, speichert es die meisten Daten automatisch. Die Szene aber müssen Sie selbst speichern. Falls Sie ein Projekt einmal neu laden, kann es sein, dass Sie im SCENE-Fenster nichts von Ihrer vorher erstellten Szene sehen. Dann öffnen Sie einfach im PROJECT-Fenster den SCENES-Ordner und *doppelklicken* auf das Szene-Symbol.

1.7 Ein Script für den Player

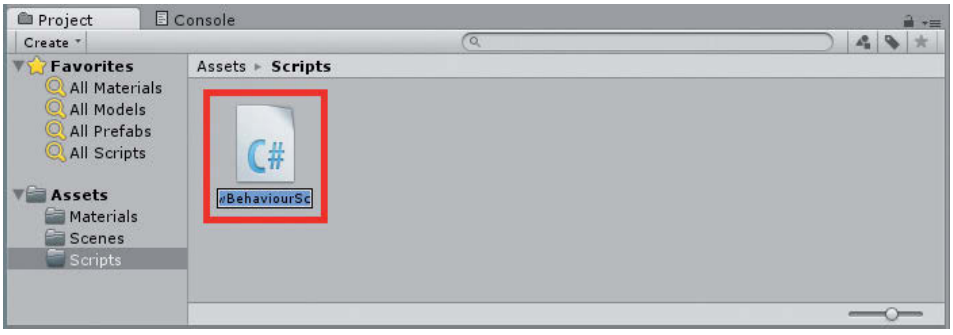
Sehen wir jetzt zu, dass wir den »Player« in Bewegung setzen. In Unity erstellt man dazu eine Art Drehbuch, *Script* genannt. Darin steht, was die Spielfigur zu tun hat.

Ein Ordner ist noch leer, aber wir kümmern uns darum, dass auch der »gefüttert« wird.

1. Wechseln Sie in den SCRIPTS-Ordner.
2. Dann klicken Sie auf ASSETS und CREATE und im Zusatzmenü auf C# SCRIPT.

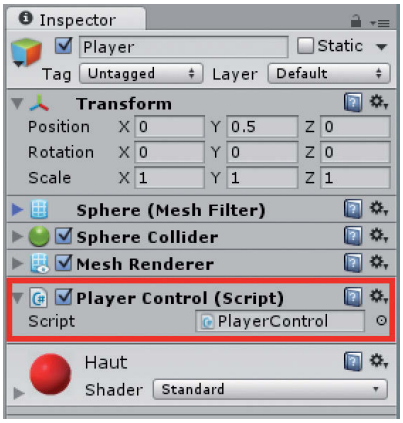


3. Geben Sie dem Script gleich einen Namen, z. B. **PlayerControl**.

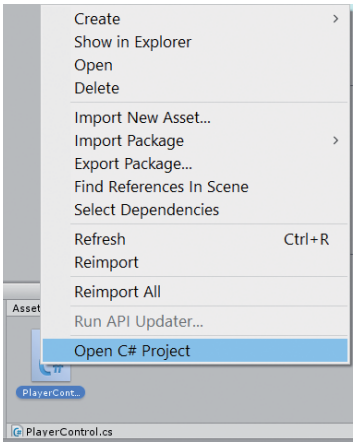


4. Schieben Sie das Script-Symbol auf die Kugel im SCENE-Fenster.

Damit ist das »Drehbuch« mit der Spielfigur verbunden. Wenn Sie im INSPECTOR-Fenster mal genauer hinschauen, sehen Sie, dass dort das Script als neue Komponente aufgetaucht ist. (Für die Abbildung habe ich die oberen Komponenten-Infos mal zugeklappt.)



5. Und nun klicken Sie mit der *rechten* Maustaste auf das Script-Symbol und im Kontextmenü auf OPEN C# PROJECT.

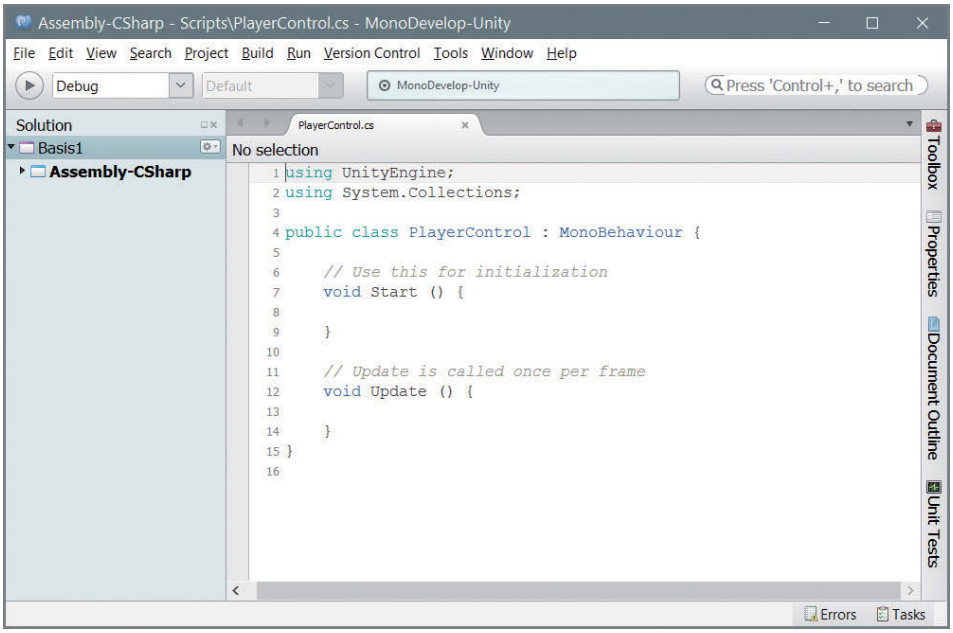


Gestartet wird nun ein weiteres Programm, mit dem Sie ein Script bearbeiten können.

Visual Studio oder MonoDevelop?

Je nachdem, welche Einstellung Sie bei der Installation vorgenommen haben, ist das *MonoDevelop*, das Unity automatisch mitinstalliert hat, oder *Visual Studio* von Microsoft.

Ich benutze hier MonoDevelop, weil Visual Studio nicht jeder hat. Man kann aber auf der Microsoft-Seite eine Version kostenlos herunterladen.



Wie Sie sehen, braucht ein Script seine eigene Programmiersprache, in der sich präzise Anweisungen formulieren lassen. Diese Sprache ist hier C# (gesprochen C-sharp). Und deren Wortschatz besteht aus englischen Wörtern.

Die Datei, mit der wir es hier zu tun haben, heißt bei mir PLAYERCONTROL.CS. Sie ist nichts weiter als eine Textdatei, in der Unity auch schon etwas hineingepackt hat:

```
using UnityEngine;
using System.Collections;

public class PlayerControl : MonoBehaviour {

  // Use this for initialization
  void Start () {
  }

  // Update is called once per frame
  void Update () {
  }
```

So etwas nennt man auch *Quelltext*. Hier wird wie in einem Drehbuch näher beschrieben, wie sich ein Objekt verhalten soll.

Kennen Sie die Programmiersprache C#?

Wenn Sie noch nie programmiert haben, dann sagt Ihnen das natürlich nichts oder nicht viel. Dann empfehle ich Ihnen den Crash-Kurs, den Sie im Anhang finden.

Dort werden die Grundlagen der Programmiersprache C# vorgestellt. Sie können sich aber auch einfach darauf einlassen und weiter hier mitmachen. Auch so werden Sie mit der Zeit mehr und mehr verstehen.

Die ersten beiden Zeilen beginnen mit `using`. Die dahinterstehenden Namen bezeichnen sogenannte Bibliotheken, die den Wortschatz der Sprache C# erweitern:

In `UnityEngine` befinden sich die Klassen und Methoden zur direkten Programmierung u. a. der Spiel-Objekte. Darüber lässt sich z. B. die Position unserer Spielfigur während des Spielverlaufs verändern. `System.Collections` enthält wichtige Strukturen, die die allgemeinen Fähigkeiten von C# erweitern.

C#-Bibliotheken

C# hat zunächst einen nur kleinen, aber kompakten Grundwortschatz. Alles andere ist in Bibliotheken ausgelagert, die sich über `using` in ein Projekt einbinden lassen. Der Vorteil ist, dass jeder sich eigene Bibliotheken erstellen kann, die zum jeweiligen Entwicklungssystem passen. Und für Unity gibt es eine Menge Nützlichkeiten, mit denen C# erweitert wurde.

1.8 Klassen und Methoden

Alles, was wir an Drehbuch-Anweisungen für das Verhalten der Spielfigur in unser Script schreiben, wird in einer *Klasse* zusammengefasst. Ich habe dem Script und damit dieser Klasse den Namen PlayerControl gegeben. Ich will ja mit meinen Anweisungen die Spielfigur kontrollieren, also bestimmen, was sie macht. So sieht eine leere Klassendefinition aus:

```
public class PlayerControl : MonoBehaviour {
}
```

Hier sehen Sie schon eine Eigenheit von C#, die geschweiften Klammern. Die markieren immer den Anfang und das Ende einer bestimmten Struktur. Wie Sie später sehen werden, gilt das nicht nur für Klassen.

Das nachgestellte `MonoBehaviour` bedeutet, dass unsere Klasse von einer Basisklasse abgeleitet wurde und damit schon einige Grundfähigkeiten geerbt hat. Von `MonoBehaviour` sollte *jede* Script-Klasse abgeleitet werden. Damit hat sie automatisch einige Methoden – so wie diese zwei:

```
void Start () {
}
```

In `Start()` kommt alles hinein, was das Objekt bei seiner Erzeugung an Fähigkeiten haben soll. Diese Methode wird beim Spielstart *einmal* ausgeführt, noch *vor* jeder anderen Methode, also auch vor `Update()`. Die kommt direkt *nach* dem Spielstart zum Zug, sie wird in sehr kurzen Zeitabständen das ganze Spiel über *immer wieder* aufgerufen:

```
void Update () {
}
```

Hier stehen dann Anweisungen für den Spielverlauf. In den Zeilen direkt über den Methoden-Namen stehen sogenannte *Kommentare*. Sie werden mit zwei Schrägstrichen (`//`) eingeleitet. Damit lassen sich im Script eigene Bemerkungen einfügen (die natürlich auch auf Deutsch sein können).

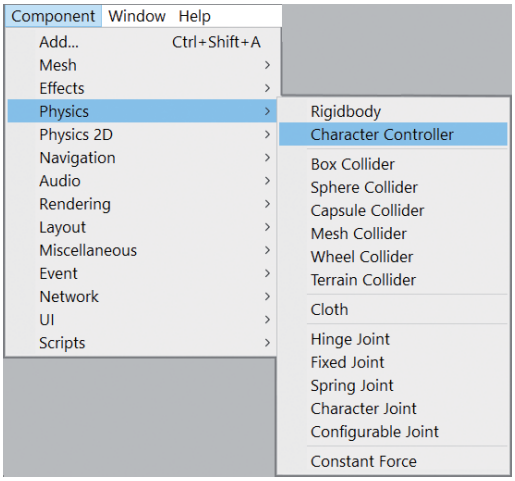
Klammer-Arten

Methoden haben in C# immer zwei Paare von Klammern: Die runden können Werte übernehmen oder leer sein (wie bei `Start()` und `Update()`), und zwischen die beiden geschweiften kommen unsere Anweisungen – wenn wir welche haben.

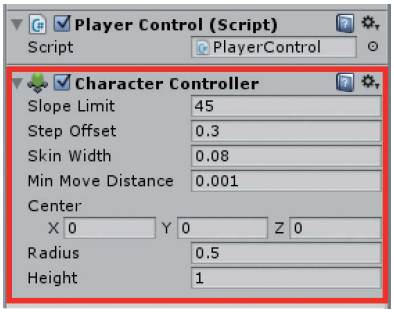
An dieser Stelle auch gleich ein Tipp, mit welchen Tastenkombinationen man an diese Klammern kommt: Für die runden Klammern sind ⇧+8 und ⇧+9 zuständig, und für die geschweiften Klammern AltGr+7 und AltGr+0.

Was wollen wir? Die Spielfigur soll sich mit den Pfeiltasten bewegen lassen. Eine Verbindung zwischen Figur und Script gibt es bereits. Für ein GameObject wie die Kugel (die unsere Spielfigur ist) bietet Unity keine Methode zum Bewegen an. Dafür brauchen wir eine zusätzliche Komponente.

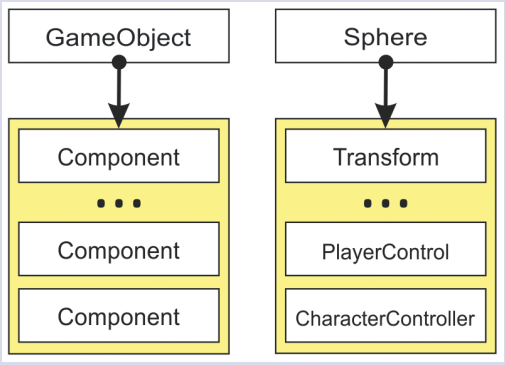
1. Wechseln Sie zu Unity und sorgen Sie dafür, dass im Hierarchy-Fenster der Eintrag Player markiert ist.
2. Klicken Sie dann auf Component und Physics und dann im Zusatzmenü auf Character Controller.



Und im INSPECTOR-Fenster taucht der Name der neuen Komponente auf.



Objekte und Komponenten



In Unity umfasst eine Szene mindestens ein Objekt vom Typ GAMEOBJECT. Dazu gehören die Kamera und eine Lichtquelle. In unserem Fall kommt noch eine Kugel als Spielfigur hinzu. Jedes Objekt hat verschiedene Komponenten (Typ COMPONENT).

Die Komponente TRANSFORM hat jedes Spiel-Objekt »von Geburt an«. Weitere Komponenten lassen sich (fast) beliebig hinzufügen (aber auch wieder entfernen).

Kehren wir zurück zum Script, wo wir nun auf bequeme Weise direkt auf die neue Komponente und deren Methoden zugreifen können. Zuvor aber müssen wir sie sozusagen im Drehbuch als »Schauspielerin« einführen. Das geschieht über diese *Vereinbarung*:

```
private CharacterController Player;
```

Nun muss der Name noch mit der Komponente der Spielfigur verknüpft werden. Das erledigen wir in der `Start`-Methode:

```
void Start () {
  Player = GetComponent <CharacterController> ();
}
```

Die Methode `GetComponent()` sorgt dafür, dass alles, was `CharacterController` zu bieten hat, nun im Script verfügbar ist. Und über den Namen `Player` kann man darauf zugreifen.

Schauen wir uns jetzt gleich den gesamten Quelltext an, mit dem wir die `Update`-Methode füllen:

```
void Update () {
  MoveVector.x = Input.GetAxis ("Horizontal");
  MoveVector.z = Input.GetAxis ("Vertical");
  Player.Move (MoveVector * Time.deltaTime * Speed);
}
```

Was Sie sicher schon erkennen und verstehen: Mit `Input` ist eine Eingabe gemeint, z. B. mit den Pfeiltasten. Und `Move` sorgt für Bewegung.

3. Ergänzen Sie das Script um die `Player`-Vereinbarung und die Methoden-Definitionen.

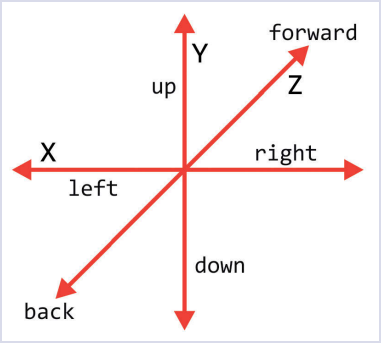
1.9 Input für den Vektor

Allerdings ist da noch ein *Vektor* im Spiel, den wir offenbar für die Bewegungsrichtung benötigen. Den vereinbaren wir so:

```
private Vector3 MoveVector = Vector3.zero;
```

Was sind Vektoren?

Der Begriff Vektor stammt aus der Mathematik. Mit einem Vektor lässt sich ein Punkt in einem Koordinatensystem festlegen. Mit einem Vektor kann man aber auch eine Bewegung beschreiben: in eine bestimmte Richtung und um eine bestimmte Strecke. (Genauer gesagt wäre ein Vektor dann ein symbolischer Pfeil.)



Der Name `Vector3` drückt aus, dass wir es hier mit dreidimensionalen Vektoren zu tun haben. (Es gibt in Unity auch die Typen `Vector2` und `Vector4`.)

Für einen solchen Vektor gibt es die Achsen `x`, `y` und `z`. Jede ist für zwei Richtungen zuständig. (Die englischen Namen in der Abbildung werden von Unity für die Kennzeichnung der jeweiligen Richtung benutzt.)

Da sich unsere Spielfigur (in der Basis-Version) nur in der Ebene bewegt, können wir auf die Veränderung des `y`-Werts verzichten. Allerdings braucht der Vektor einen Startwert, und der wird mit `Vector3.zero` auf die Koordinaten (0 | 0 | 0) festgelegt.

In der `Update`-Methode bekommt dann `x` und `z` jeweils einen anderen Wert, der von der Eingabe abhängt:

```
MoveVector.x = Input.GetAxis ("Horizontal");
MoveVector.z = Input.GetAxis ("Vertical");
```

Wir machen hier Gebrauch vom Input-Manager, den Unity uns anbietet. Grundsätzlich kann der alles managen, was mit Eingabe zu tun hat, ob die nun von einer Tastatur oder Maus, einem Joystick oder Gamepad oder durch Berührung des Displays oder von irgendwelchen Sensoren kommt.

Dazu verwendet Unity ein virtuelles Achsensystem mit den Namen `"Horizontal"` und `"Vertical"`. Ganz gleich, welches Eingabegerät im Einsatz ist, eine Spielfigur lässt sich dadurch steuern, dass der Wert dieser virtuellen Achsen abgefragt wird. Unity hat bereits eine ganze Reihe von Tasten vordefiniert, darunter die Pfeiltasten für Links-Rechts und Vorwärts-Rückwärts oder die Leertaste für Sprünge.

Nachdem der Vektor, dem ich den Namen `MoveVector` gegeben habe, mit den Daten gefüttert wurde, den Unity aus den Pfeiltasten gelesen hat, wird der Player entsprechend bewegt:

```
Player.Move (MoveVector * Time.deltaTime * Speed);
```

Dazu sind allerdings noch ein paar zusätzliche Faktoren nötig: `Time.deltaTime` gibt die Zeitspanne an, die zwischen zwei Aufrufen von `Update()` liegt. Das kann je nach Leistungsfähigkeit des Computers nur 30-mal sein oder auch mehr als 60-mal. `deltaTime` sorgt also dafür, dass sich die Spielfigur auf jedem Computer gleich schnell bewegt.

Und damit Sie da aber auch selbst ein Wörtchen mitreden können, vereinbaren wir zusätzlich eine Variable und weisen ihr einen Wert zu:

```
public float Speed = 5.0f;
```

Diese Vereinbarung sollte ebenso wie die von `MoveVector` direkt unter der Objektvereinbarung von `Player` stehen:

```
private CharacterController Player;
public float Speed = 5.0f;
private Vector3 MoveVector = Vector3.zero;
```

private und public

Sicher ist Ihnen aufgefallen, dass die Vereinbarungen entweder mit `private` oder mit `public` eingeleitet werden.

Alles, was innerhalb einer Klasse vereinbart und verwendet wird, ist zuerst eigentlich »Privatsache«, denn es wird normalerweise nur **innerhalb** der Klasse (und des Scripts) benutzt. Deshalb wird hier `private` vereinbart.

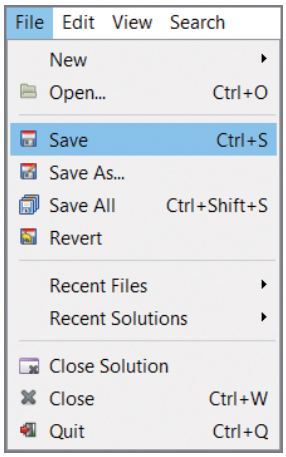
Anders ist es bei `public`: So vereinbarte Variablen sind auch außerhalb, also öffentlich verwendbar. Das erkennen Sie gleich, wenn Sie nach der Vereinbarung das Script gespeichert haben: Die Variable erscheint nun im INSPECTOR-Fenster und ihr Wert ist von außen veränderbar.

1.10 Programmlauf nach Drehbuch

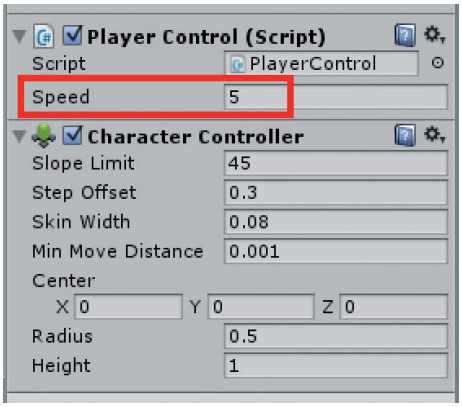
Hier nun noch einmal der komplette Quelltext für unser Script:

```
public class PlayerControl : MonoBehaviour {
  private CharacterController Player;
  public float Speed = 5.0f;
  private Vector3 MoveVector = Vector3.zero;

  void Start () {
    Player = GetComponent <CharacterController> ();
  }

  void Update () {
    MoveVector.x = Input.GetAxis ("Horizontal");
    MoveVector.z = Input.GetAxis ("Vertical");
    Player.Move (MoveVector * Time.deltaTime * Speed);
  }
}
```

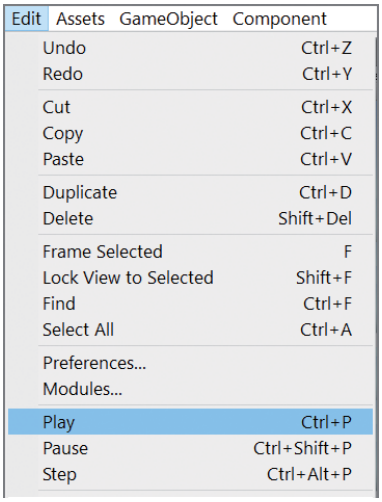
1. Wechseln Sie zu MonoDevelop und erweitern Sie den vorhandenen Quelltext um diese Zeilen. (Kommentare können Sie entfernen oder auch um eigene ergänzen.)
2. Klicken Sie (in MonoDevelop) auf FILE und SAVE, um die Datei zu speichern.



3. Wechseln Sie zu Unity und schauen Sie dort ins INSPECTOR-Fenster. Wenn Sie wollen, können Sie den vorgegebenen Wert hinter SPEED ändern.



4. Speichern Sie jetzt zuerst noch einmal die Spiel-Szene.
5. Dann starten Sie das Programm. Dazu können Sie auf EDIT und PLAY klicken.



Oder Sie nutzen den Play-Button direkt über dem SCENE-Fenster.

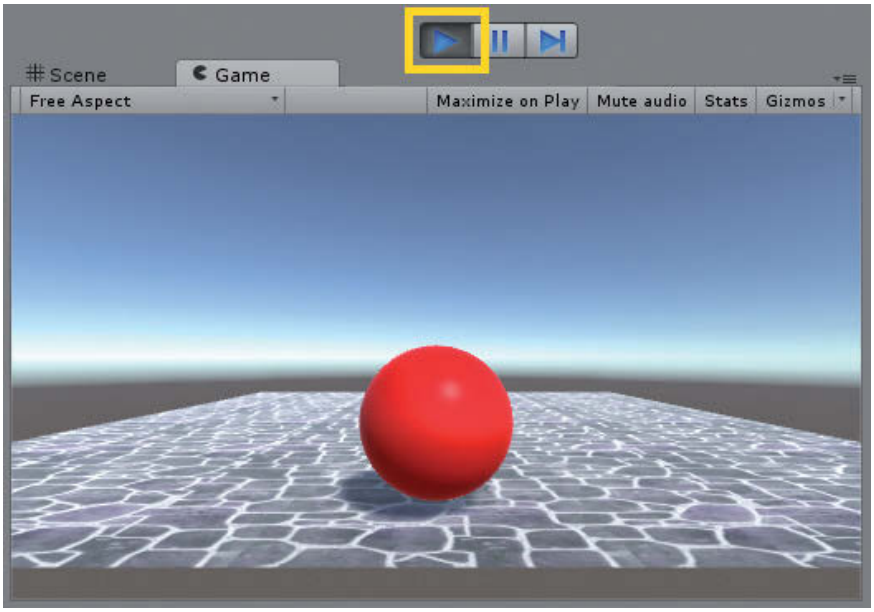
Und Unity schaltet ins GAME-Fenster um. Dort sehen Sie nun die Kugel bzw. die Spielfigur ein bisschen aus der Ferne. Das liegt daran, dass die Kamera von Unity etwas weiter weg positioniert wurde, um das ganze Spielfeld überblicken zu können.

Maximize

Damit Sie mehr vom Spiel sehen können, sollten Sie den Button MAXIMIZE ON PLAY oben rechts am GAME-Fenster anschalten.

Sie können natürlich zusätzlich auch das ganze Fenstersystem von Unity maximieren.

6. Probieren Sie Ihr Basis-Spiel aus, indem Sie die Pfeiltasten betätigen. Dabei können Sie auch zwei Tasten kombinieren, damit die Kugel auch mal schräg übers Spielfeld rollt.



Wie Sie sehen, bewegt sich die Kugel. (In der Abbildung habe ich sie schon ein bisschen nach vorn geholt.)

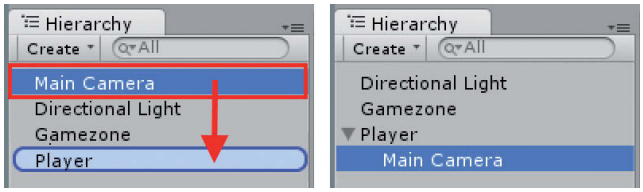
Aufgabe 1.3

Ersetzen Sie die Kugel (SPHERE) auch mal durch einen anderen Körper: einen Quader (CUBE), einen Zylinder (CYLINDER) oder eine Kapsel (CAPSULE).

1.11 Kameraführung

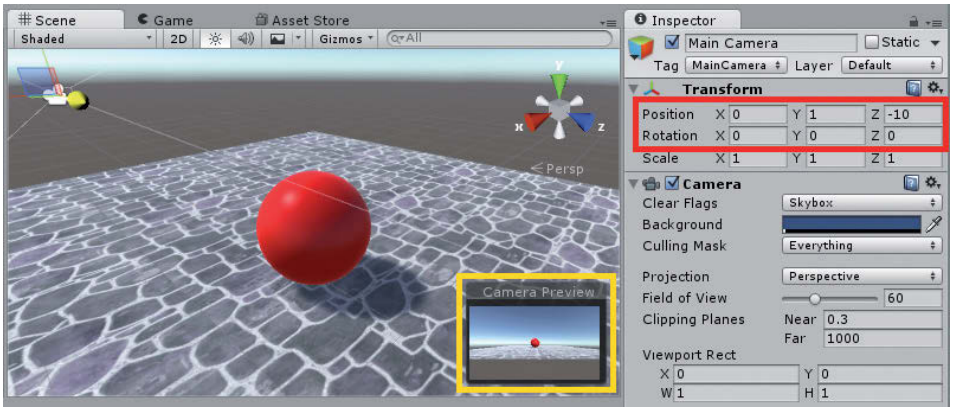
Die Spielfigur lässt sich nun mit den Pfeiltasten steuern. Bei all der Arbeit, die Sie bis jetzt geleistet haben, ist das aber doch ein bisschen dürftig. Ich habe zuletzt die Kamera erwähnt. Bei einem richtigen Spiel sollte die doch nicht starr sein, sondern der Figur folgen. Und darum kümmern wir uns jetzt.

1. Ziehen Sie im HIERARCHY-Fenster den Eintrag MAIN CAMERA nach unten auf den Eintrag PLAYER.



Von jetzt an ist die Kamera fest mit der Spielfigur verbunden: Wo Letztere hingeht, folgt ihr Erstere. Allerdings sind die beiden hier noch ein bisschen zu weit voneinander entfernt.

2. Sorgen Sie dafür, dass der Eintrag MAIN CAMERA markiert ist, und ändern Sie im INSPECTOR-Fenster unter TRANSFORM und hinter POSITION den z-Wert.



Wenn die Kamera markiert ist, erscheint unten rechts im SCENE-Fenster eine kleine Vorschau. Sie können aber auch direkt oben über den Reiter ins GAME-Fenster wechseln.

Ein bisschen Experimentieren ist schon nötig, bis Sie einen passenden Abstand gefunden haben. Vielleicht gefällt Ihnen die Kamerasicht dann aber immer noch nicht. Aber da lässt sich ja noch mehr einstellen:

- Die Höhe der Kamera im Vergleich zur Spielfigur kann man über den Y-Wert hinter POSITION ändern.
- Außerdem lässt sich die Kamera nach unten oder oben kippen, wenn Sie den X-Wert hinter ROTATION ändern.

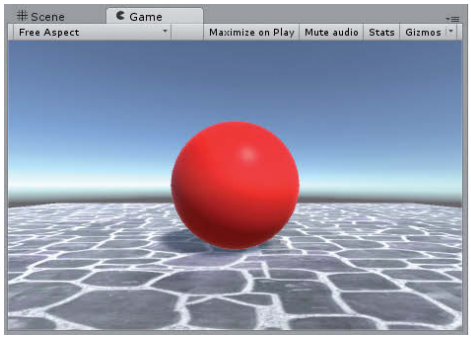
Meter oder Grad?

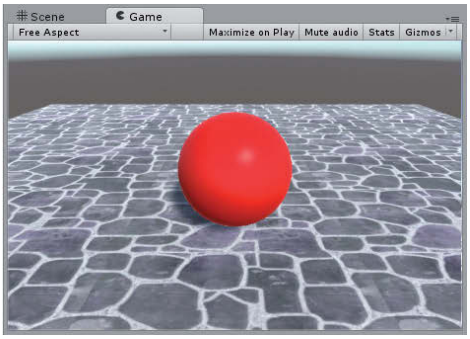
Beachten Sie, dass Unity die Zahlen hinter POSITION (und SCALE) als Meter und die bei ROTATION als Grad versteht.

Zwei Vorschläge für die Kameraperspektive sehen Sie in dieser Tabelle:

	X	Y	Z		X	Y	Z
Position	0	0	-2	Position	0	1	-2
Rotation	0	0	0	Rotation	30	0	0

Und hier sehen Sie, wie sich das bei mir ausgewirkt hat:





3. Speichern Sie die Szene noch einmal, ehe Sie das Programm über den Play-Button starten und die Kugel mit den Pfeiltasten durch die Gegend rollen (lassen).

Ein Basis-Projekt haben Sie jetzt. Tauschen Sie die Kugel gegen eine richtige Figur aus (wie wir es in Projekt 2 tun werden) und ersetzen Sie die Ebene z. B. durch eine richtige Landschaft (die erstellen wir in Projekt 4), dann können Sie wenigstens schon durch eine Spielwelt wandern.

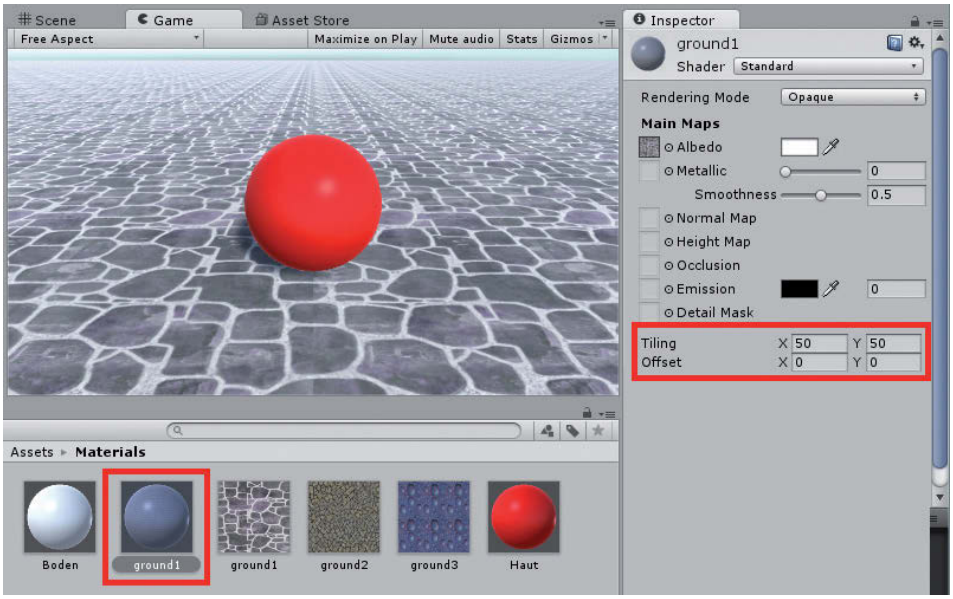
Bonus: Spiel ohne Grenzen?

Kümmern wir uns nun noch einmal um das Spielfeld. Zum Ersten ist es bis jetzt doch ein wenig klein. Das lässt sich leicht ändern, indem man die Maße der Ebene z. B. auf 100 m mal 100 m erweitert (oder mehr, wenn Sie wollen).

1. Markieren Sie das Objekt GAMEZONE und ändern Sie im INSPECTOR-Fenster die SCALE-Werte:

Scale	X = 10	Y = 1	Z = 10

2. Wechseln Sie im PROJECT-Fenster in den Ordner MATERIALS und markieren Sie dort das Symbol für das Spielfeld-Material. Passen Sie im INSPECTOR-Fenster die Werte für TILING an. Denn die Textur muss ja der neuen Objektgröße angepasst werden.



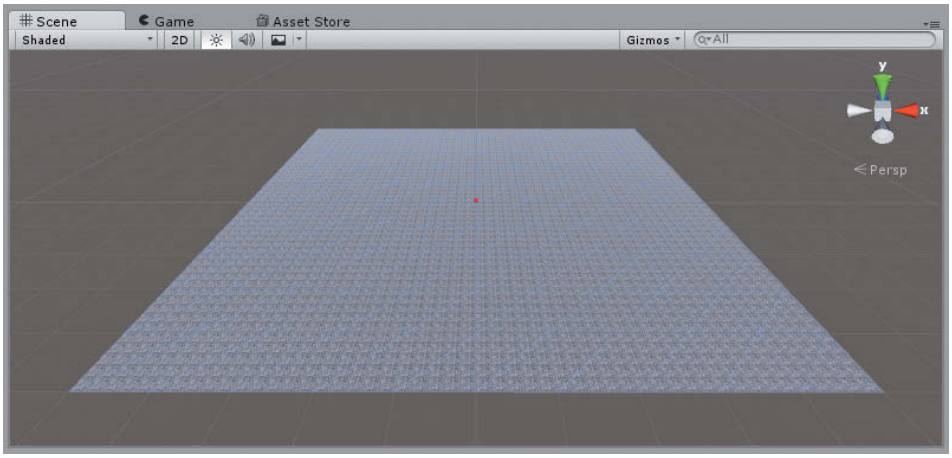
Was wir nun noch brauchen, sind Grenzen. Denn wie Sie ja schon bemerkt haben, lässt sich die Spielfigur beliebig weit über das Spielfeld hinaus bewegen. Das aber muss bei einem richtigen Spiel nicht sein.

Das Erste, was mir (und Ihnen?) dazu einfällt, sind Mauern, die das gesamte Feld umgeben. Wenn man die sehen darf, wären Quader vom Typ CUBE die richtige Lösung. Die könnte man dann mit Texturen versehen, damit Quader z. B. wie Ziegelsteinmauern aussehen.

Oft aber will man lieber Grenzen, die unsichtbar sind. Auch die hat Unity zu bieten und diese sogenannten »Empty Objects« setzen wir hier ein.

Zuerst aber sollten wir dafür sorgen, dass man das komplette Spielfeld im Blick hat.

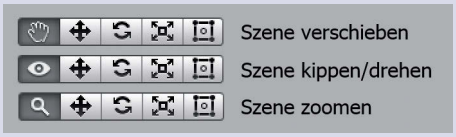
3. Maximieren Sie das Fenstersystem von Unity, sodass es den ganzen Bildschirm bedeckt. Dann klicken Sie ins SCENE-Fenster und benutzen das Scrollrad der Maus, um die Ansicht zu zoomen (also zu vergrößern oder zu verkleinern).



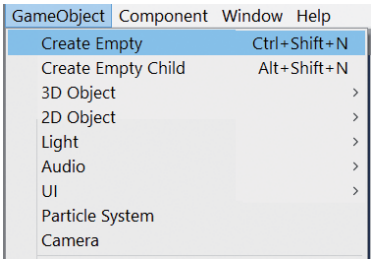
Ich habe die Sicht auf das Spielfeld auch gleich so gedreht, dass ich in Blickrichtung der Kamera darauf schauen kann.

Ansichtssachen

Ganz oben links, direkt unter dem Hauptmenü, gibt es mehrere Symbole. Klicken Sie auf das erste, das Hand-Symbol, dann lässt sich im SCENE-Fenster die Ansicht bei gedrückter linker Maustaste verschieben. Und bei gedrückter rechter Maustaste können Sie die Szene kippen und drehen. Nehmen Sie die Tasten Strg und Alt hinzu, lässt sich noch mehr machen. Probieren Sie's aus!



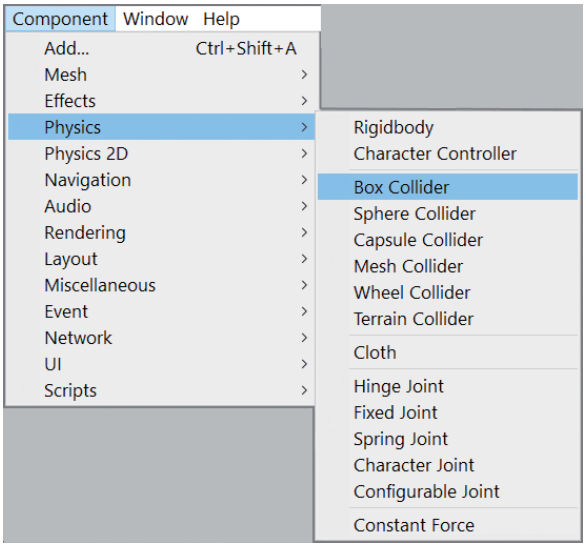
4. Klicken Sie jetzt im Hauptmenü auf GAMEOBJECT und dann auf CREATE EMPTY.



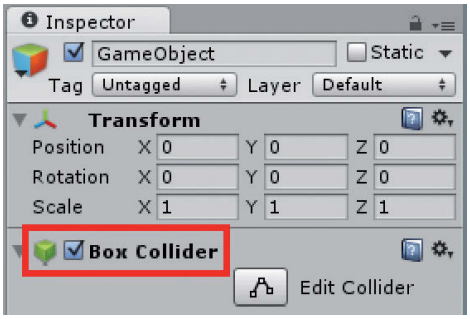
Wir haben nun ein ziemlich leeres (und nicht sichtbares) Objekt im Spiel, das nur über eine TRANSFORM-Komponente verfügt – wie Sie im INSPECTOR-Fenster sehen können. Dort lässt sich unsere Mauer passend positionieren und vergrößern.

Weil sie weiter für uns unsichtbar ist, tappen wir regelrecht im Dunkeln. Doch es gibt eine Möglichkeit, dem leeren Spielobjekt eine »Füllung« zu geben, womit sie für uns im SCENE-Fenster sichtbar, später aber im Spiel weiter unsichtbar bleibt.

5. Markieren Sie das neue Objekt im HIERARCHY-Fenster. Dann klicken Sie im Hauptmenü auf COMPONENT und PHYSICS und wählen dann im Zusatzmenü BOX COLLIDER.



Und sofort taucht im INSPECTOR-Fenster die neue Komponente auf. Was man mit einem COLLIDER anfangen kann, erfahren Sie schon bald.



So richtig viel sehen können Sie derzeit noch nicht. Aber das ändert sich, wenn Sie aus dem kleinen »Ding« eine passende Mauer machen. Dabei gehe ich davon aus, dass das Spielfeld 100 x 100 m lang und breit ist.

6. Ändern Sie im INSPECTOR-Fenster die Werte für Lage und Größe so um:

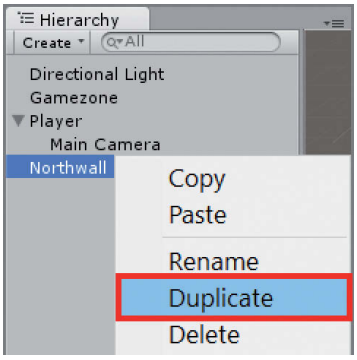
Position	X = 0	Y = 5	Z = **50**
Scale	X = **100**	Y = 10	Z = 1

7. Geben Sie der Mauer auch gleich im HIERARCHY-Fenster einen neuen Namen: **Northwall**.

Damit haben wir unseren ersten Grenzwall. Bei der Namensgebung für die neuen Objekte habe ich für mich die Blickrichtung der Kamera auf Norden festgelegt.

Um die anderen Mauern nicht jedes Mal neu erzeugen zu müssen, duplizieren wir einfach das vorhandene Objekt mehrmals und passen die neuen Objekte an.

1. Klicken Sie im HIERARCHY-Fenster mit der *rechten* Maustaste auf den Eintrag NORTHWALL. Wählen Sie im Kontextmenü die Option DUPLICATE.



Nun gibt es schon zwei Mauern.

2. Benennen Sie das neue Objekt in **Southwall** um. Dann passen Sie im INSPECTOR-Fenster die Werte für die neue Mauer so an:

Position	X = 0	Y = 5	Z = **-50**
Scale	X = **100**	Y = 10	Z = 1

3. Duplizieren Sie jetzt noch die beiden anderen Mauern, die dann **Eastwall** und **Westwall** heißen sollten. Und hier sind die Werte für Position und Skalierung:

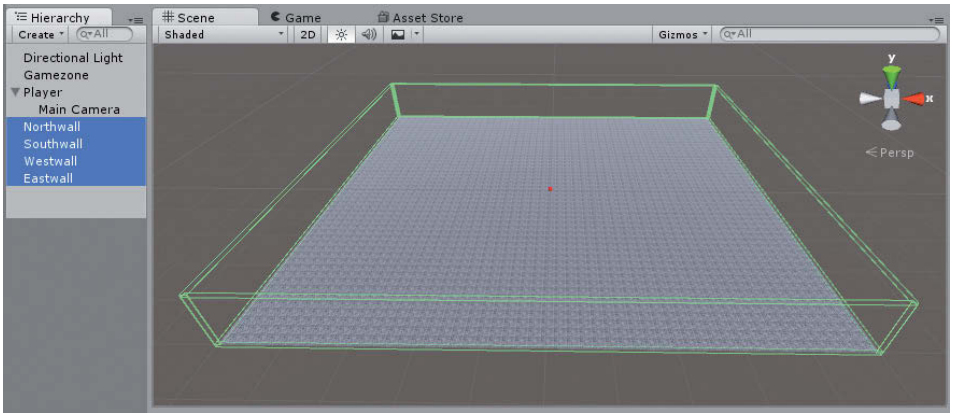
EASTWALL:

Position	X = **50**	Y = 5	Z = 0
Scale	X = 1	Y = 10	Z = **100**

WESTWALL:

Position	X = **-50**	Y = 5	Z = 0
Scale	X = 1	Y = 10	Z = **100**

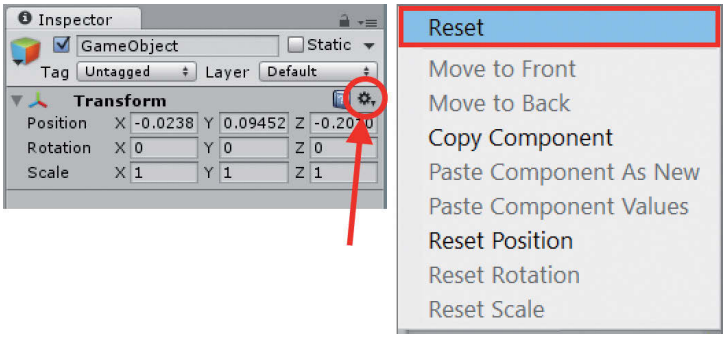
Womit das Mauerwerk komplett wäre. Wenn Sie (mit gedrückter Strg-Taste) alle WALL-Einträge im HIERARCHY-Fenster markieren, sehen Sie im SCENE-Fenster alle Grenzwälle zusammen.



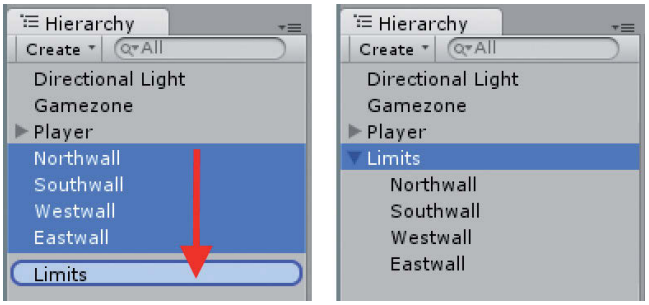
Diese vier Elemente packen wir jetzt noch in einen Container. Dazu verwenden wir ein weiteres leeres Objekt.

1. Dazu klicken Sie wieder im Hauptmenü auf GAMEOBJECT und CREATE EMPTY.
2. Geben Sie dem neuen Objekt im HIERARCHY-Fenster einen Namen wie z. B. **Limits**.

3. Sorgen Sie im INSPECTOR-Fenster dafür, dass alle POSITION-Werte auf 0 stehen. Klicken Sie dazu auf das kleine Zahnrad hinter TRANSFORM und dann im Kontextmenü auf RESET.



4. Markieren Sie im HIERARCHY-Fenster alle WALL-Objekte und ziehen Sie sie auf das LIMITS-Element.



Nun gibt es ein Objekt (mit vier Unter-Objekten).

5. Speichern Sie die Szene und starten Sie das Spiel. Und testen Sie die neuen Grenzen.

Dass die Spielfigur die unsichtbaren Mauern wahrnimmt und davon aufgehalten wird, verdanken wir der COLLIDER-Komponente. Wie deren Name schon sagt, haben wir es hier mit Kollisionen zu tun.

In Unity bekommen die meisten neu erzeugten Objekte außer der TRANSFORM- auch eine COLLIDER-Komponente mit auf den Weg. Treffen zwei solcher Objekte aufeinander, dann meldet der jeweilige Collider eine Kollision. Was in unserem Falle heißt: Die Spielfigur wird gestoppt. Vielleicht haben Sie schon bemerkt, dass die Kugel und die Objekte für die Grenzmauern nicht die gleiche Art von Collider haben. Werfen Sie doch mal einen Blick ins INSPECTOR-Fenster. Die wichtigsten Collider habe ich in einer Tabelle aufgeführt:

BOX COLLIDER	Der Bereich innerhalb eines Quaders wird auf Kontakt kontrolliert. Objekte vom Typ CUBE bekommen automatisch einen BOX-COLLIDER als Komponente.
SPHERE COLLIDER	Der Bereich innerhalb einer Kugel wird auf Kontakt kontrolliert. Objekte vom Typ SPHERE bekommen automatisch einen SPHERE-COLLIDER als Komponente.
CAPSULE COLLIDER	Eine Kapsel besteht aus einem Zylinder, an dessen Kreisflächen jeweils eine Halbkugel »klebt«. Dieser Bereich wird auf Kontakt kontrolliert. Objekte vom Typ CYLINDER und CAPSULE bekommen automatisch einen CAPSULE-COLLIDER als Komponente.

Mit Kollisionen und ihren Folgen werden wir uns in späteren Projekten noch beschäftigen.

Aufgabe 1.4

Verpassen Sie dem Spielfeld auch mal sichtbare Wände. Setzen Sie dazu über GAMEOBJECT|3D OBJECT|CUBE jeweils Quader ein, Sie können dieselben Werte für POSITION und SCALE verwenden wie bei den »Empty Objects«. Geben Sie den Mauern Material und Texturen.

1.12 Exit

Das erste Projekt können Sie hier (erst einmal) als beendet ansehen. Aber wahrscheinlich wird Ihnen immer wieder etwas einfallen, das Sie an Ihrem Werk gern noch verbessern würden.

Mein Basis-Projekt lässt sich von der Website des Verlags herunterladen. Es heißt BASIS1 und ist unter diesem Link zu finden:

`http://www.mitp.de/362`

Dazu gibt es ein zweites Basis-Projekt mit einigen Änderungen und um eine Maussteuerung erweitert: Es trägt den Namen BASIS2.

Duplikat des Basis-Projekts?

Wenn Sie für eigene Experimente das Basis-Projekt in seiner ursprünglichen Version behalten möchten, kopieren Sie einfach den kompletten Ordner und benennen ihn um. Dann haben Sie ein neues Projekt, mit dem Sie anstellen können, was Sie wollen.

Stichwortverzeichnis