Kapitel 1
Einführung

Unity ist eine Entwicklungsumgebung zur Erstellung und Gestaltung von Computerspielen, sowohl unter Windows als auch unter Linux und unter macOS. Unity-Projekte können darüber hinaus auch zu Lern- und Trainingszwecken genutzt werden.

Es gibt unterschiedliche Lizenzmodelle für Unity. Falls Sie Unity zu Übungs- und Lernzwecken nutzen, können Sie die Lizenz *Unity Personal* kostenfrei verwenden.

1.1 Was machen wir mit Unity?

Ihre Projekte

Unity bietet eine Vielfalt von Möglichkeiten. In diesem Einsteigerbuch beschäftigen wir uns mit den wichtigsten Elementen, die Ihnen eine selbstständige Gestaltung und Programmierung Ihrer Projekte ermöglichen.

Physik

Wir werden mit einfachen Flächen und Körpern arbeiten, die Ihnen ein Verständnis für Elemente im zweidimensionalen Raum und im dreidimensionalen Raum vermitteln. Daraus bauen wir gemeinsam Schritt für Schritt eine ganze Reihe verschiedener Spiele auf. Dabei lernen Sie, wie die Elemente aufeinander reagieren, besonders unter physikalischen Bedingungen.

Abläufe gestalten

Sie lernen die Elemente der Programmierung mit C# von Grund auf kennen, um vielseitige Abläufe gestalten zu können. Dadurch werden Sie in die Lage versetzt, die vorhandenen Spiele nach Ihren Wünschen weiter zu verändern und selbstständig eigene Spiele zu entwickeln.

Verständnis

Wir werden nicht einfach vorgefertigte komplexe Elemente miteinander kombinieren, wie sie z. B. in großer Zahl im *Asset Store* von Unity angeboten werden. Diese Elemente besitzen zwar eine Fülle von Fähigkeiten und bieten zahlreiche optische Effekte, allerdings trägt das reine Einsetzen und punktuelle Verändern dieser Elemente nur wenig zum Verständnis ihres komplexen Aufbaus bei. Sie vereinfachen auch nicht das Verständnis für den programmierten Spielablauf. Viele reichhaltig gestaltete Spielfiguren

können zudem nur mit externen Programmen erstellt werden und müssen danach zunächst in Unity importiert werden.

2D-Spiele

Die Begriffe *zweidimensional* und *dreidimensional* kürze ich in diesem Buch häufig mit *2D* und *3D* ab. Wir entwickeln zunächst reine 2D-Spiele. Dabei arbeiten wir bereits mit vielen Unity-Elementen, die später auch für die Entwicklung von 3D-Spielen benötigt werden.

2D-Spiele finden ausschließlich auf der Ebene des Bildschirms statt. Sie sind übersichtlicher und einfacher zu erfassen als 3D-Spiele. Sie können sich also auf das Erlernen der Spieleentwicklung konzentrieren und müssen sich nicht gleichzeitig mit der Orientierung, der Drehung und der Perspektive im dreidimensionalen Raum befassen.

Mein Dank: Für die Hilfe bei der Erstellung dieses Buchs bedanke ich mich bei dem ganzen Team vom Rheinwerk Verlag, ganz besonders bei Almut Poll und Anne Scheibe.

1.2 Wie entsteht der programmierte Spielablauf?

C#

Zur Programmierung der logischen Abläufe in den Unity-Projekten können Sie mit der Programmiersprache C# (sprich: C-Sharp) arbeiten. Alle benötigten Elemente der Sprache lernen Sie im Verlauf der Projekte an der passenden Stelle kennen. Bei Unity wird der Begriff *C#-Scripte* statt des Begriffs *C#-Programme* verwendet. Daher verwende ich diesen Begriff ebenfalls.

Zur Entwicklung der C#-Scripte wird bei der Installation der aktuellen Unity-Versionen eine kostenfreie Version der Entwicklungsumgebung *Visual Studio* der Firma Microsoft angeboten. Alternativ können Sie auch die Entwicklungsumgebung *MonoDevelop* installieren, die aus einer älteren Unity-Version stammt und von den Unity-Entwicklern weiterhin gepflegt und zur Verfügung gestellt wird.

Programmierkurs

In Kapitel 19, »Ein Programmierkurs in C#«, finden Sie zusätzlich einen reinen C#-Programmierkurs, in dem ohne die eigentlichen Unity-Elemente gearbeitet wird. Im Vordergrund stehen diejenigen Elemente der Sprache, die Sie in den Unity-Projekten besonders benötigen. Neben dem Erlernen von C# anhand der Unity-Projekte können Sie den C#-Programmierkurs als Ergänzung für das bessere Verständnis und als Nachschlagewerk nutzen, falls Sie etwas über bestimmte Elemente von C# erfahren möchten.

1.3 Dateiendungen anzeigen lassen

Windows Explorer

Für eine Installation unter Windows und für die nachfolgende Projektentwicklung ist es nützlich, Windows so einzustellen, dass die Endungen der Dateinamen angezeigt werden. Im Explorer von Windows 10 geht das wie folgt:

- Klappen Sie die Liste Optionen auf der Registerkarte Ansicht auf, und wählen Sie Ordner- und Suchoptionen ändern.
- Wechseln Sie im Dialogfeld Ordneroptionen auf die Registerkarte Ansicht.
- Entfernen Sie die Markierung bei Erweiterungen bei bekannten Dateitypen ausblenden.
- Bestätigen Sie Ihre Änderungen über die Schaltfläche OK.

1.4 Unity Hub installieren

Zur Installation und Verwaltung von Unity und zur Verwaltung Ihrer eigenen Unity-Projekte empfehle ich zunächst, die Anwendung *Unity Hub* zu installieren. Nachfolgend nenne ich diese Anwendung vereinfacht den Unity Hub.

Adresse

Sie finden sie mithilfe Ihres Browsers über die folgende Adresse:

https://unity3d.com/de/get-unity/download

Betätigen Sie die Schaltfläche Unity Hub herunterladen, siehe Abbildung 1.1.

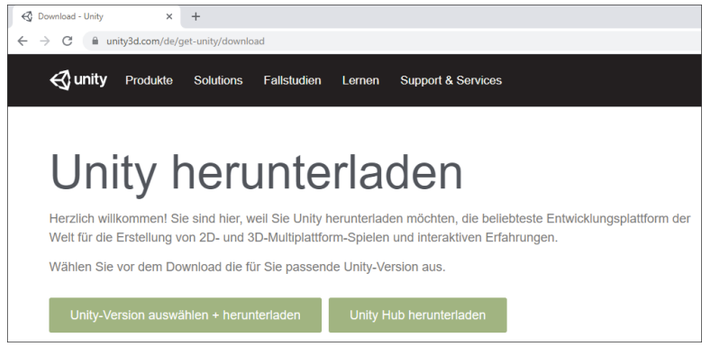

Abbildung 1.1 Unity-Website, Unity Hub herunterladen

Installationsdatei

Unter Windows wird die Datei *UnityHubSetup.exe* heruntergeladen, siehe Abbildung 1.2, die Sie anschließend zum Start der Installation aufrufen können. Unter Linux und unter macOS handelt es sich um andere Dateien. Deren Aufruf wird in Abschnitt 22.3, »Unity unter anderen Betriebssystemen«, beschrieben.

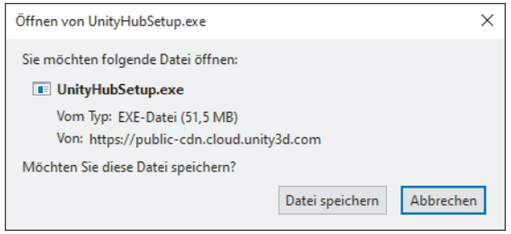

Abbildung 1.2 Installationsprogramm für den Unity Hub

Zu Beginn der Installation müssen die allgemeinen Lizenzbedingungen akzeptiert werden. Im nächsten Dialogfeld wählen Sie das Zielverzeichnis für die Installation aus. Nach der Installation können Sie den Unity Hub ausführen lassen. Zunächst erscheint gegebenenfalls ein Windows-Sicherheitshinweis, in dem Sie den blockierten Zugriff des Unity Hubs explizit zulassen sollten.

Unity-Lizenz

Falls Sie Unity erstmals nutzen, haben Sie noch keine Unity-Lizenz. In diesem Fall erscheinen die Einstellungen (engl. *Preferences*) des Dialogfelds Unity Hub mit der Seite License Management zur Verwaltung der Unity-Lizenz, siehe Abbildung 1.5.

Zur Erlangung einer Lizenz benötigen Sie als Erstes eine Unity ID. In der Fußzeile erscheint die Meldung You need to be logged in to manage your license. Betätigen Sie daneben den Link Login.

Unity-ID

Es erscheint das Dialogfeld Unity Hub Sign in, siehe Abbildung 1.3. Falls Sie noch keine Unity ID haben, betätigen Sie den Link create one, um Ihre persönliche Unity ID zu erzeugen. Melden Sie sich anschließend mit Ihrer E-Mail und Ihrem Passwort an.

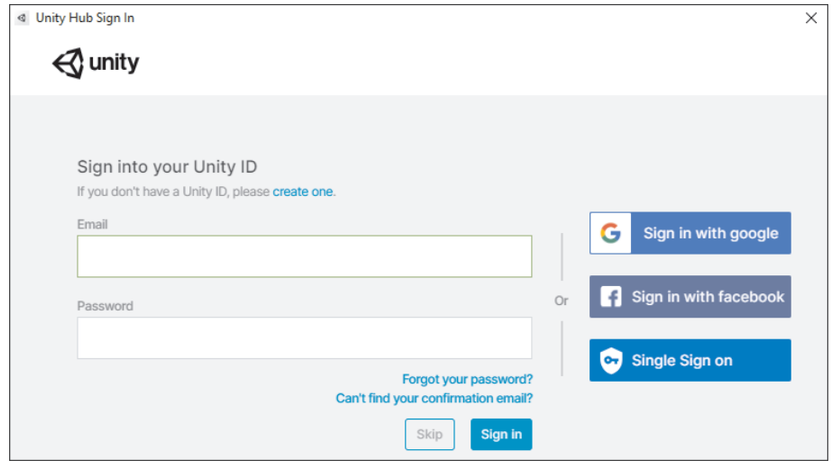

Abbildung 1.3 Anmeldung mit Unity ID

Unity Personal

Nach der erfolgreichen Anmeldung betätigen Sie im Dialogfeld Preferences die Schaltfläche Activate New License. Wählen Sie im Dialogfeld New License Activation die kostenfreie Lizenz Unity Personal und danach diejenige Option, die bei Ihnen zutrifft, siehe Abbildung 1.4.

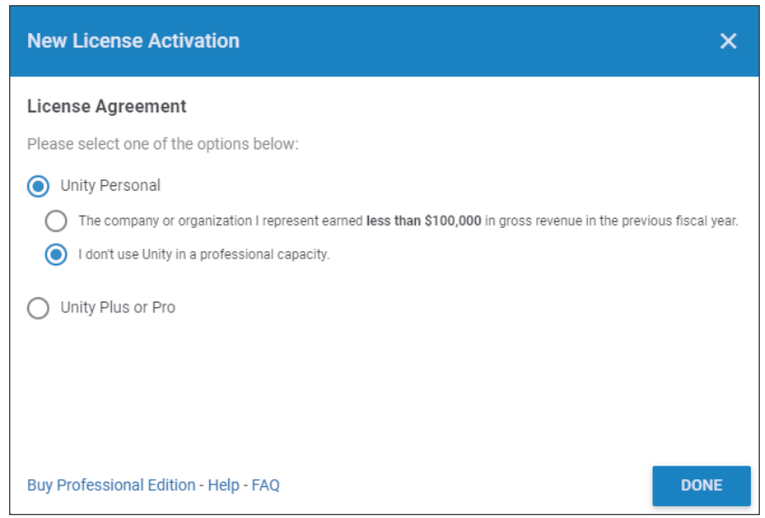

Abbildung 1.4 Auswahl der Lizenz

Nach der Betätigung der Schaltfläche DONE kehren Sie wieder zum Dialogfeld PREFERENCES zurück. Ihre Lizenz wird wie in Abbildung 1.5 angezeigt.

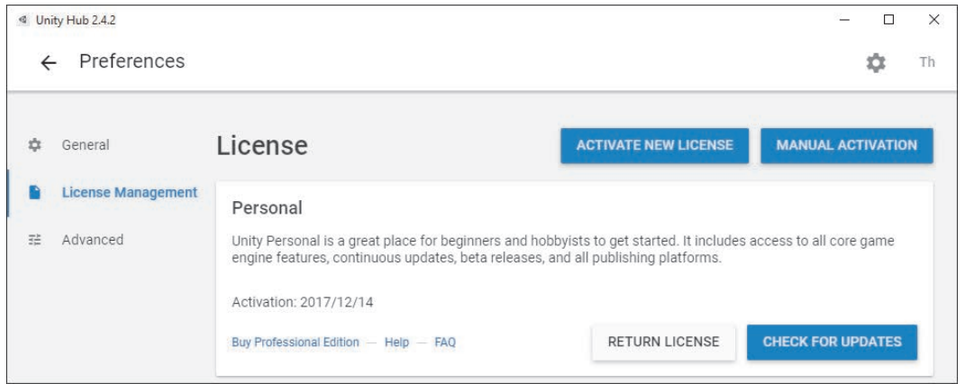

Abbildung 1.5 Lizenz »Personal«

Verlassen Sie das Dialogfeld PREFERENCES über den Pfeil oben links. Sie gelangen zur Hauptseite des Dialogfelds UNITY HUB. In Abbildung 1.6 sehen Sie die Version 2.4.2. Sobald eine neue Version oder ein neues Release (= Unterversion) des Unity Hubs erscheint, wird Ihnen ein Update vorgeschlagen. Ich empfehle, dem Vorschlag zu folgen.

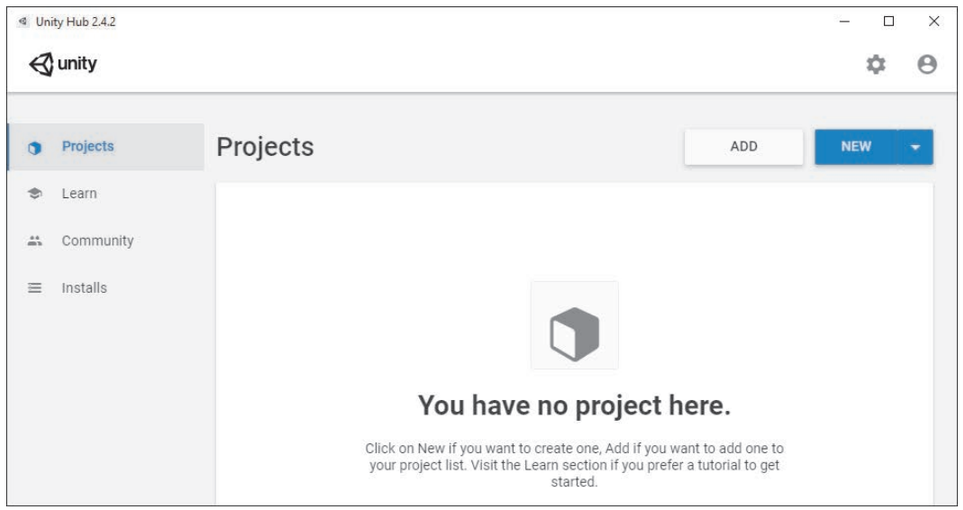

Abbildung 1.6 Unity Hub, Projects

Auf der linken Seite befindet sich ein Menü, mit dessen Hilfe Sie verschiedene Seiten aufrufen können. Auf der Seite PROJECTS werden später Ihre eigenen Unity-Projekte angezeigt.

Parallele Nutzung

Neue Versionen, neue Releases und neue Vorversionen von Unity erscheinen parallel und in kurzen Zeitabständen. Alle werden ständig weiterentwickelt und können von Ihnen gleichzeitig genutzt werden. So können Sie permanent von der Modernisierung von Unity profitieren. Der Unity Hub ist dafür vorgesehen, Ihnen die parallele Nutzung und den schnellen Umstieg zu ermöglichen.

Auf der Seite INSTALLS werden später die verschiedenen Unity-Versionen angezeigt, die Sie installiert haben, siehe Abbildung 1.7.

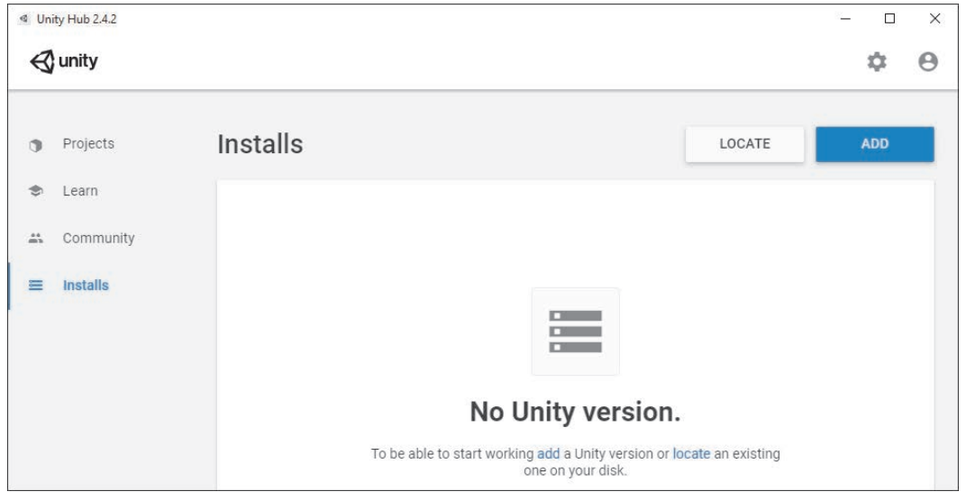

Abbildung 1.7 Unity Hub, Installs

1.5 Unity-Version installieren

Version auswählen

Zur Installation einer Unity-Version betätigen Sie auf der Seite INSTALLS die Schaltfläche ADD, siehe Abbildung 1.7. Es erscheint die erste Seite des Dialogfelds ADD UNITY VERSION, siehe Abbildung 1.8. Es gibt ein empfohlenes Release mit Langzeitunterstützung (engl.: Long Term Support, kurz LTS), offizielle Releases und Vor-Releases. Wählen Sie z. B. das Vor-Release Beta 2 zu Unity 2021.1.0 aus, und betätigen Sie die Schaltfläche NEXT.

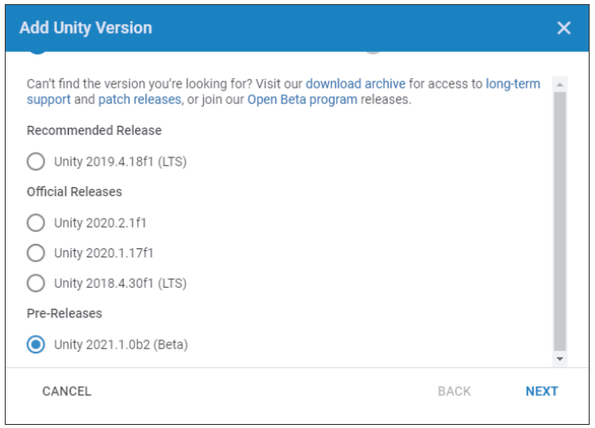

Abbildung 1.8 Unity-Version hinzufügen

Komponenten hinzufügen

Anschließend erscheint die zweite Seite des Dialogfelds ADD UNITY VERSION, siehe Abbildung 1.9. Hier können Sie weitere Komponenten zur Installation auswählen, u. a. eine Version von *Visual Studio* der Firma Microsoft. Alternativ können Sie *MonoDevelop* installieren, wie es in Abschnitt 1.6, »MonoDevelop installieren«, beschrieben wird.

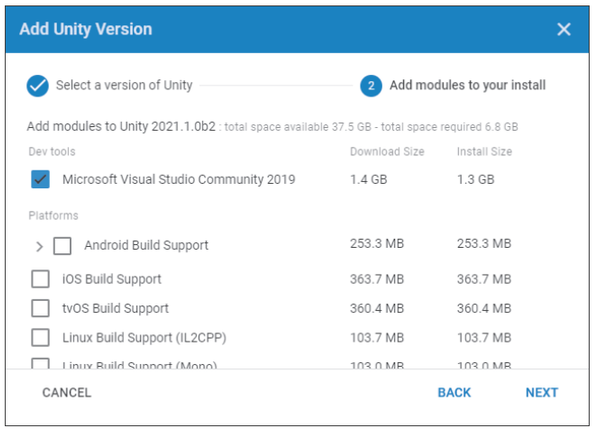

Abbildung 1.9 Weitere Komponenten

Weitere Komponenten werden zurzeit noch nicht benötigt. Sie könnten im Übrigen jederzeit nachinstalliert werden. Daher wird als Nächstes die Schaltfläche DONE betätigt.

Die eigentliche Installation startet und nimmt eine gewisse Zeit in Anspruch. Es erscheint ein neues Element mit einem Fortschrittsbalken am oberen Rand, an dem Sie den Stand der Installation erkennen können, siehe Abbildung 1.10. Über das Menü mit den drei Punkten könnten Sie eine laufende Installation jederzeit abbrechen.

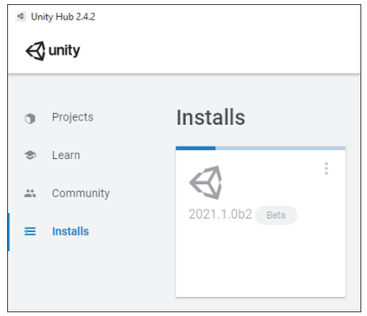

Abbildung 1.10 Unity Hub, laufende Installation

Visual Studio

Haben Sie *Visual Studio* als Zusatzmodul ausgewählt, gilt Folgendes:

- Nach einiger Zeit erscheint das Dialogfeld END USER LICENSE AGREEMENT für Visual Studio, in dem die allgemeinen Lizenzbedingungen akzeptiert werden müssen.
- Nach der Fertigstellung der Unity-Installation sollte der Unity Hub beendet und ein Neustart des Rechners durchgeführt werden.
- Sie nehmen in Ihrem ersten Unity-Projekt eine weitere notwendige Einstellung vor, damit Visual Studio anschließend in Ihren sämtlichen Unity-Projekten automatisch genutzt wird, siehe Abschnitt 4.3.1, »Wählen Sie die Entwicklungsumgebung«.

Im Folgenden gehe ich davon aus, dass Sie eine Unity-Version installiert haben.

1.6 MonoDevelop installieren

MonoDevelop kann unter allen Unity-Versionen installiert und genutzt werden. Rufen Sie zum Download von MonoDevelop in Ihrem Browser die folgende Seite auf:

https://unity3d.com/de/get-unity/download/archive

Wählen Sie den Reiter 2017.X aus. Die im Dezember 2020 aktuelle 2017er-Version (hier: 2017.4.40) steht oben in der Liste. Wählen Sie in der Liste Downloads (Win) das Unity-Installationsprogramm aus, siehe Abbildung 1.11.

Abbildung 1.11 Unity-Website, Version 2017.4.40 herunterladen

Es wird die Installationsdatei *UnityDownloadAssistant-2017.4.40f1.exe* heruntergeladen. Starten Sie die Installation. Wählen Sie unter den Komponenten ausschließlich MonoDevelop aus, siehe Abbildung 1.12.

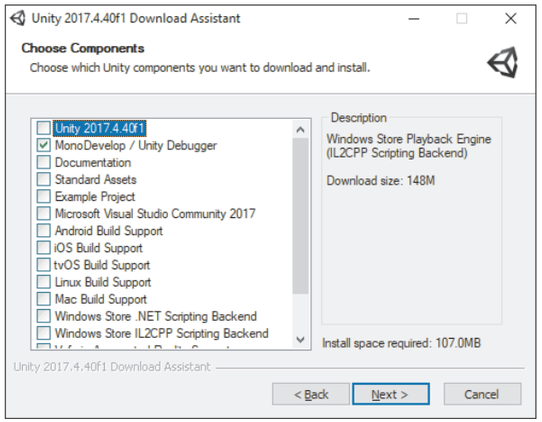

Abbildung 1.12 Komponente MonoDevelop

Die Unity-Version 2017.4.40 selbst muss nicht installiert werden. Sie können als Installationsverzeichnis ebenfalls *C:\UnityHub* auswählen.

In Ihrem ersten Unity-Projekt nehmen Sie eine weitere notwendige Einstellung vor, damit MonoDevelop anschließend in Ihren sämtlichen Unity-Projekten automatisch genutzt wird, siehe Abschnitt 4.3.1, »Wählen Sie die Entwicklungsumgebung«.

1.7 Beispielprojekte und Assets

Beispielprojekte

In diesem Buch arbeite ich mit einer Reihe von Beispielprojekten, die Ihnen sowohl als Hilfestellung bei der eigenen Entwicklung als auch zum Kennenlernen und Ausprobieren in ausführbarer Form dienen. Sie stehen in den Materialien zum Buch über die Webseite *https://www.rheinwerk-verlag.de/5293* zum Download zur Verfügung.

Bei Unity wird jedes Projekt in einem eigenen Projektverzeichnis gespeichert. Legen Sie zur besseren Übersicht ein eigenes Oberverzeichnis an, z. B. *C:\UnityProjekte*. Kopieren Sie alle Verzeichnisse mit meinen Beispielprojekten nach dem Download in dieses Verzeichnis.

Assets

In den Beispielprojekten nutze ich eine Reihe von einfachen vorgefertigten Elementen, sogenannte *Assets*. Sie finden sie, zusammen mit den bereits erwähnten Projektverzeichnissen, im Verzeichnis *FreieAssets* in den Materialien zum Buch.

Bonusprojekte

In Abschnitt 22.6, »Bonusprojekte«, werden zwei Bonusprojekte beschrieben, die sich ebenfalls in den Materialien zum Buch befinden. Sie sind den bekannten Spielen *Pacman* und *Frogger* nachempfunden. Mit den Fähigkeiten, die Sie aus diesem Buch erworben haben, sind Sie in der Lage, den Aufbau dieser Projekte zu erkennen und sie mit eigenen Ideen selbstständig weiterzuentwickeln.

1.8 Unity-Projekte und Unity-Versionen

Aufgrund der kurzen Zeitabstände beim Erscheinen neuer Unity-Versionen kommt es häufig vor, dass die Entwicklung eines Unity-Projekts zunächst unter einer bestimmten Version beginnt und später unter einer anderen Version fortgesetzt wird. Beim Öffnen eines Projekts in einer neueren Version ist ein Upgrade notwendig. Mithilfe des Unity Hubs ist das leicht möglich.

Beim Öffnen eines Projekts wird zunächst die gewünschte Version ausgewählt. Klicken Sie dazu auf den kleinen Pfeil in der Spalte Unity-Version. Es erscheint eine Liste der installierten Unity-Versionen zur Auswahl. In Abbildung 1.13 wird das am Beispiel des Projekts *TomsJumpAndRun* aus Kapitel 3, »Spielen Sie ein 2D-Jump&Run-Spiel«, gezeigt. Es wurde zunächst unter der Unity-Version 2019.4.14f1 entwickelt und soll unter der Unity-Version 2020.1.12f1 weiterentwickelt werden.

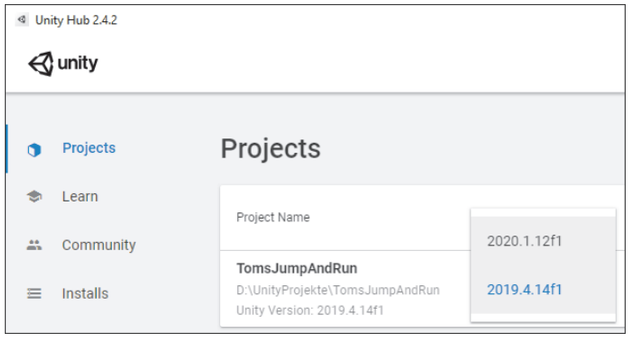

Abbildung 1.13 Auswahl einer Unity-Version

Nach Auswahl der Unity-Version wird das Projekt geöffnet, indem die gesamte Projektzeile angeklickt wird. Es erscheint das in Abbildung 1.14 dargestellte Dialogfeld. Die Konvertierung nimmt eine gewisse Zeit in Anspruch. Anschließend kann das Unity-Projekt weiterbearbeitet werden.

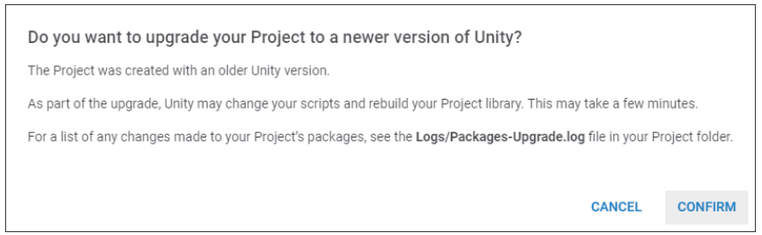

Abbildung 1.14 Upgrade eines Projekts

Die Beispielprojekte, die Screenshots und die Erläuterungen in diesem Buch sind mithilfe der Unity-Version 2021.1.0b2 entstanden, also der zweiten Beta-Vorversion der Unity-Version 2021.1.0.

Die Erfahrungen der letzten Jahre zeigen, dass in neuere Unity-Versionen zwar Verbesserungen einfließen, aber die grundsätzliche Orientierung erhalten bleibt. Daher werden die Inhalte dieses Buchs Sie noch sehr lange bei der Entwicklung von Unity-Projekten unterstützen.

Kapitel 9
Das erste 3D-Projekt

Ähnliche Elemente

Mit Einführung der dritten Dimension werden Ihre Projekte räumlicher und realistischer. Gleichzeitig werden sie auch komplexer. Nach dem Bearbeiten der bisherigen Kapitel haben Sie aber einen Vorteil: Sie haben bereits viele Unity-Elemente und grundsätzliche Abläufe in 2D-Projekten kennengelernt und können sie in 3D-Projekten in gleicher oder ähnlicher Form weiterverwenden.

Verständnis

In einem ersten 3D-Projekt mit dem Namen *DreiDimensionen* werden mithilfe von einfachen 3D-Objekten einige Grundlagen erläutert. Ähnlich wie im Einführungsprojekt *ZweiDimensionen* aus Kapitel 2, »Das erste 2D-Projekt«, handelt es sich nicht um ein Spielprojekt, sondern dient Ihrem Verständnis für typische 3D-Elemente. Einige der Abläufe können Sie zudem in Ihren eigenen 3D-Projekten einsetzen.

9.1 Grundlagen eines 3D-Projekts

Sie erfahren mehr über die Kamera sowie die Elemente und Materialien, aus denen einfache 3D-Objekte zusammengesetzt sind. Zudem gestalten Sie die dreidimensionale Ansicht in der Scene View.

9.1.1 Kamera, Skybox und Licht

3D-Projekt anlegen

Erstellen Sie ein neues Projekt mit dem Namen *DreiDimensionen*. Achten Sie darauf, dass die Option 3D markiert ist. Damit erhalten 3D-Projekte die passenden Voreinstellungen.

Horizont

Wie bei 2D-Projekten gibt es zu Beginn bereits das Spielobjekt `Main Camera`. Wir setzen es auf die Position –2/1/–7 und schauen damit aus einer leicht erhöhten Position von links vorn auf die Szene. In der Komponente Camera des Spielobjekts `Main Camera` finden Sie die Eigenschaft Clear Flags. Sie steht auf dem Standardwert `Skybox`. Mit dieser Einstellung wird der Eindruck einer realen Szene inklusive eines Himmels und eines Horizonts erzeugt. Dagegen würde der Hintergrund z. B. mithilfe des Werts `Solid Color` nur in einer einheitlichen Farbe dargestellt.

Licht-Objekt

Außerdem gibt es in 3D-Projekten zu Beginn ein Licht-Objekt. Es sorgt für eine Beleuchtung der Elemente und dient zur Erzeugung von Schatten und zur weiteren Verbesserung der räumlichen Wirkung. Sie können in einem Projekt mehrere Licht-Objekte gleichzeitig einsetzen.

Hier handelt es sich um ein Licht-Objekt des Typs *Directional Light*. Wir belassen es bei den Standardwerten, also bei der Position 0/3/0 und bei der Rotation 50/330/0.

Hinweis

Es gibt u. a. die folgenden Typen von Licht-Objekten:

- Ein Objekt des Typs *Directional Light* beleuchtet die gesamte Szene aus einer bestimmten Richtung. Die Lichtstärke ist überall gleich. Die Wirkung ist ähnlich wie beim Licht der Sonne.
- Ein Objekt des Typs *Point Light* strahlt von einem bestimmten Punkt aus in alle Richtungen. Die Lichtstärke nimmt mit der Entfernung zu diesem Punkt ab. Die Wirkung ist ähnlich wie beim Licht einer Glühlampe.
- Ein Objekt des Typs *Spot Light* strahlt ebenfalls von einem bestimmten Punkt aus. Die Lichtstärke nimmt ebenso mit der Entfernung zu diesem Punkt ab. Allerdings wird das Licht mithilfe eines Winkels eingeschränkt, sodass sich ein Lichtkegel ergibt. Die Wirkung in der Szene ist ähnlich wie beim Licht eines Scheinwerfers.

9.1.2 Einfache 3D-Objekte

Sie können in Ihren 3D-Unity-Projekten beliebige komplexe 3D-Objekte einsetzen. Diese 3D-Objekte können mithilfe von verschiedenen Modellierungsprogrammen außerhalb von Unity erstellt werden. Im Unity Editor werden aber bereits einige einfache 3D-Objekte bereitgestellt, die für viele Projekte ausreichen. Einige dieser Objekte werden nachfolgend eingefügt. Klappen Sie dazu jeweils das Menü der Hierarchy View mithilfe eines Klicks auf das Pluszeichen auf:

Würfel

- Erzeugen Sie einen Würfel mithilfe des Menüpunkts 3D Object • Cube. Er verbleibt in der Mitte, und zwar an der Position 0/0/0.

Kugel

- Erzeugen Sie als Nächstes mithilfe des Menüpunkts 3D Object • Sphere eine Kugel. Positionieren Sie sie bei 0/2/0. Sie wird also auf der positiven Y-Achse verschoben und befindet sich oberhalb des Würfels.

Kapsel

- Es folgt eine Kapsel, die mithilfe des Menüpunkts 3D Object • Capsule erstellt wird. Sie wird auf der positiven X-Achse verschoben, und zwar an die Position 4/0/0.

Zylinder

- Als Letztes erstellen Sie über den Menüpunkt 3D Object • Cylinder noch einen Zylinder. Er wird auf der negativen X-Achse an die Position –4/0/0 verschoben.

In der Scene View ergibt sich ein Bild wie in Abbildung 9.1.

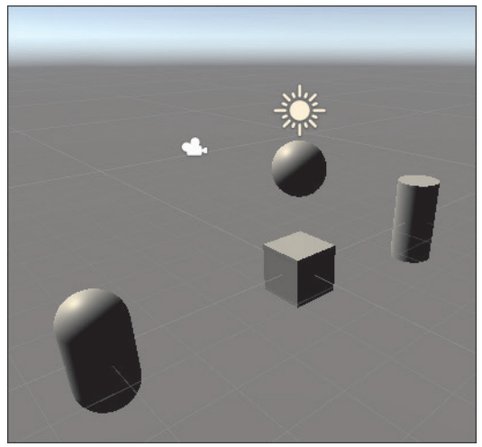

Abbildung 9.1 Einfache 3D-Objekte in der »Scene View«

Hinweis

Der Einfluss des Lichts, der Schattenwurf und die Skybox sind gut zu erkennen. Standardmäßig werden die Objekte perspektivisch dargestellt. Objekte, die weiter weg vom Betrachter sind, werden also kleiner dargestellt. Auch das trägt zur Verbesserung der räumlichen Wirkung bei.

9.1.3 Farbiges Oberflächenmaterial

Neues Material

Die Oberflächen der 3D-Objekte sollen eine Farbe erhalten. Erstellen Sie das Asset-Verzeichnis Materials. Öffnen Sie durch einen Klick auf das Pluszeichen das Menü der Project View, und erzeugen Sie in dem neuen Verzeichnis mithilfe des Menüpunkts Material ein neues Asset für Oberflächenmaterialien. Nennen Sie es `OrangeMat` als Abkürzung für *Orangefarbenes Oberflächenmaterial*.

Albedo

Markieren Sie das Asset, und stellen Sie in der INSPECTOR VIEW die wichtigste Farbe ein: Wählen Sie für die Eigenschaft ALBEDO im Bereich MAIN MAPS (siehe Abbildung 9.2) die RGB-Werte 255/128/0. Die Eigenschaft ALBEDO bestimmt das Reflexionsverhalten von Oberflächen, die angeleuchtet werden.

Erstellen Sie durch Kopie ein weiteres Asset für Oberflächenmaterialien, und nennen Sie es `HellblauMat`. Es erhält für die Eigenschaft ALBEDO die RGB-Werte 0/128/255.

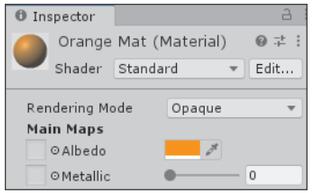

Abbildung 9.2 Wichtigste Farbe für ein Oberflächenmaterial

Material zuordnen

Zur Zuordnung der Oberflächenmaterialien klappen Sie bei den einzelnen Objekten in der Komponente MESH RENDERER den Bereich MATERIALS auf. Ziehen Sie das jeweilige Material-Asset auf die Eigenschaft ELEMENT 0. Die Kugel wird orangefarben, die Kapsel hellblau, siehe Abbildung 9.7.

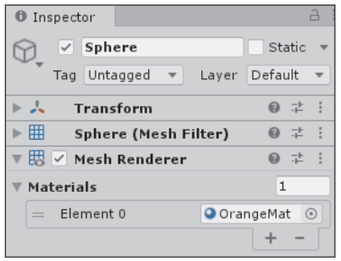

Abbildung 9.3 Oberflächenmaterial zuordnen

9.1.4 Oberflächenmaterial mit Textur

Eine Textur ist ein zweidimensionales Bild, das auf die Oberfläche eines dreidimensionalen Körpers gelegt wird, damit dieser realistischer wirkt. Erstellen Sie das Asset-Verzeichnis TEXTURES, und importieren Sie aus dem Verzeichnis *FreieAssets* die beiden Bilddateien *Holzfass.png* und *Mauer.png* in das neue Verzeichnis. Sie sollen als Texturen dienen.

Erstellen Sie im Asset-Verzeichnis MATERIALS ein neues Material, und nennen Sie es `HolzfassMat` als Abkürzung für *Oberflächenmaterial mit Holzfass-Textur*. Markieren Sie es, und wählen Sie in der Liste SHADER (dt.: Schattierung) statt des Eintrags STANDARD den Eintrag LEGACY SHADERS • DIFFUSE, siehe Abbildung 9.4.

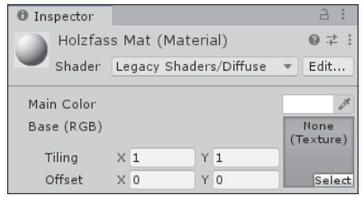

Abbildung 9.4 Typ der Schattierung einstellen

Bei diesem Typ der Schattierung haben Sie die Möglichkeit, nach Betätigung der Schaltfläche SELECT eine Textur einzustellen. Wählen Sie im Dialogfeld SELECT TEXTURE die zuvor importierte Textur HOLZFASS, siehe Abbildung 9.5.

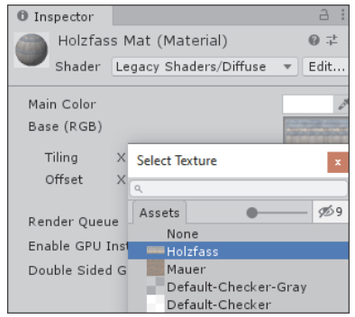

Abbildung 9.5 Textur auswählen

Erstellen Sie ein weiteres Material mit dem Namen `MauerMat`, und geben Sie ihm auf dieselbe Weise die Textur MAUER.

Hinweis

Standardmäßig wird ein Bild bei der Texturierung nur einmal auf einer Oberfläche abgebildet. Abhängig von der Größe der Oberfläche kann es aber realistischer wirken, wenn das Textur-Bild wie eine Wand- oder Bodenfliese mehrmals nebeneinander und übereinander abgebildet wird. Dieser Vorgang wird *Tiling* (engl.: to tile, dt.: Fliesenlegen) genannt.

Stellen Sie beim Material `MauerMat` den TILING-Wert für X auf 2, siehe Abbildung 9.6. In diesem Fall erscheint die Textur in X-Richtung zweimal nebeneinander auf der Oberfläche. Der Wert für Y bleibt beim Standardwert 1.

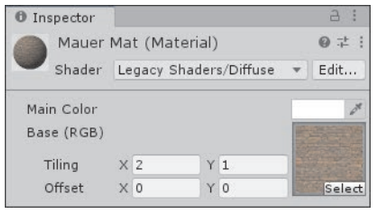

Abbildung 9.6 Tiling einstellen

> **Hinweis**
>
> Sind die Übergänge der neben- bzw. übereinanderliegenden Fliesen nicht sichtbar, ist eine Bilddatei besonders gut für das Tiling geeignet. Die Strukturen am linken und am rechten Rand eines Bilds sollten also ebenso zueinander passen wie diejenigen am oberen und am unteren Rand.

Anschließend sieht das Asset-Verzeichnis MATERIALS aus wie in Abbildung 9.7.

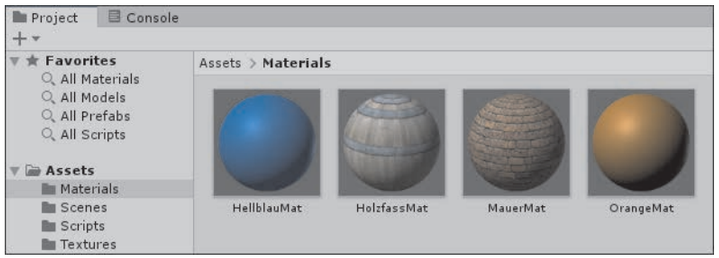

Abbildung 9.7 Vier Materialien

Weisen Sie dem Zylinder das Material `HolzfassMat` und dem Würfel das Material `MauerMat` zu.

9.1.5 Oberflächenmaterial wechseln

Klasse »Cylinder«

Sie können das Aussehen von 3D-Objekten zur Laufzeit eines Projekts verändern, indem Sie z. B. das Oberflächenmaterial wechseln, wie im nachfolgenden Code für den Zylinder:


```
using UnityEngine;
public class Cylinder : MonoBehaviour
{
   public Material hellblauMat;
   public Material holzfassMat;

   void Update()
   {
      if (Input.GetKeyDown (KeyCode.B))
         GetComponent<MeshRenderer>().material = hellblauMat;
      else if (Input.GetKeyDown (KeyCode.F))
         GetComponent<MeshRenderer>().material = holzfassMat;
   }
}
```

Listing 9.1 Klasse »Cylinder«

Mesh Renderer

Es werden die beiden öffentlich zugänglichen Variablen `hellblauMat` und `holzfassMat` vom Datentyp `Material` erstellt. Falls der Benutzer die Taste B oder die Taste F betätigt, erhält die Eigenschaft `material` der Komponente MESH RENDERER des zugehörigen Spielobjekts einen neuen Wert.

Material zuordnen

Das C#-Script wird dem Spielobjekt `Cylinder` zugeordnet. Im UNITY EDITOR werden die beiden Material-Assets auf die zugehörigen Slots gezogen.

9.1.6 Ansicht in der »Scene View« gestalten

Handsymbol

Da wir nun einige Objekte zur Verfügung haben, können wir uns die Möglichkeiten veranschaulichen, die Ansicht in der SCENE VIEW zu gestalten. Klicken Sie zunächst auf das Hand-symbol in der oberen linken Symbolleiste:

- *Verschieben* Sie die Ansicht in der SCENE VIEW durch Verschieben der Maus bei gedrückter linker Maustaste nach links, rechts, oben oder unten. — Verschieben
- *Drehen* Sie die Ansicht in der SCENE VIEW um verschiedene Achsen durch Verschieben der Maus bei gedrückter rechter Maustaste nach links, rechts, oben oder unten. — Drehen
- *Zoomen* Sie in der SCENE VIEW durch Drücken und Festhalten der Taste Alt und gleichzeitiges Verschieben der Maus bei gedrückter rechter Maustaste nach links, rechts, oben oder unten. — Zoomen

Gizmo

Rechts oben in der SCENE VIEW befindet sich ein kleines Hilfselement, ein sogenanntes *Gizmo* (siehe Abbildung 9.8). Es zeigt die aktuelle Lage der drei Koordinatenachsen und die aktuelle Darstellungsart.

Hinweise

Wie in Abschnitt 2.8.1, »Die Eigenschaften der Transform-Komponente«, wenden Sie auch hier die *Linke-Hand-Regel* an: Der Daumen der linken Hand weist in Richtung der positiven X-Achse. Der Zeigefinger der linken Hand weist in Richtung der positiven Y-Achse. Der Mittelfinger der linken Hand wird, von der Handfläche aus gesehen, nach vorn gestreckt. Damit weist er in Richtung der positiven Z-Achse.

Der positive Teil der Achse (oder kurz: die positive Achse) ist derjenige Teil mit dem farbigen Kegel und der Achsenbezeichnung (x, y oder z). Die negative Achse ist diejenige mit dem nicht farbigen Kegel.

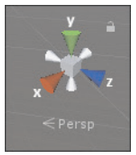

Abbildung 9.8 Gizmo

Isometrisch

Klicken Sie einmal auf den Schriftzug PERSP (Abkürzung für *perspektivisch*) unter dem Gizmo: Die Darstellungsart wechselt zu *isometrisch*, der Text des Schriftzugs auf ISO. Alle 3D-Objekte werden unabhängig von der Entfernung zum Betrachter gleich groß dargestellt. Der räumliche Effekt geht verloren. Ein erneuter Klick auf den Schriftzug wechselt wieder zur perspektivischen Darstellungsart.

Klick auf Achse

Klicken Sie nacheinander auf die drei positiven Achsen des Gizmos. Anschließend betrachten wir die Szene aus der jeweiligen positiven Achsenrichtung. Klicken Sie nacheinander auf die drei negativen Achsen. Anschließend betrachten wir die Szene aus der jeweiligen negativen Achsenrichtung.

Ansicht ändern

Versuchen Sie durch Verschieben, Drehen und Zoomen ungefähr die Ansicht in Abbildung 9.9 herzustellen: Die positive X-Achse weist nach rechts, die positive Y-Achse weist nach oben. Die Ansicht wird gedreht,

sodass die linke und die obere Seite des Würfels zu sehen sind. Das ist beim ersten Mal nicht einfach, stellt aber eine gute Übung dar.

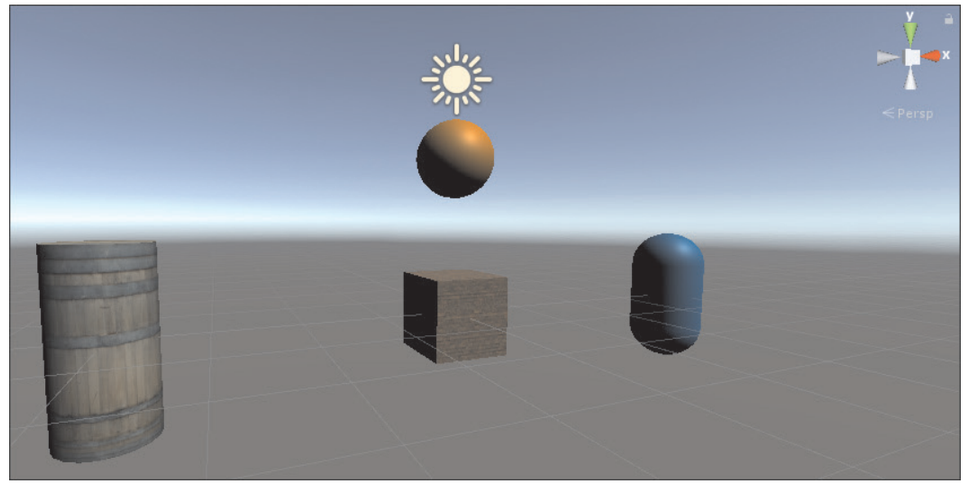

Abbildung 9.9 Gewünschte Ansicht

Align with View

Bei Bedarf können Sie nun auch die Ansicht in der GAME VIEW gleichartig gestalten. Wählen Sie dazu in der HIERARCHY VIEW das Spielobjekt `Main Camera` aus, und rufen Sie den Menüpunkt GAMEOBJECT • ALIGN WITH VIEW auf.

9.2 Verschieben und Drehen

Bei der Verschiebung und besonders bei der Drehung von 3D-Objekten sollten Sie eine gute Vorstellung des dreidimensionalen Raums haben. Es folgen einige Bewegungen der Objekte mithilfe von Programmcode im Projekt *DreiDimensionen*.

9.2.1 Spielobjekte drehen

Den 3D-Objekten werden *Drehmomente* zugeordnet. Damit werden Drehungen der 3D-Objekte um bestimmte Achsen des dreidimensionalen Koordinatensystems verdeutlicht.

Hinweis

Was ist ein Drehmoment? Nehmen wir an, die vorhandenen 3D-Objekte könnten sich nur drehen, aber ihre Position nicht verändern. Nehmen wir des Weiteren an, Sie stecken eine lange Stange durch ein 3D-Objekt, z. B. durch den Würfel, der im Zentrum steht, und zwar genau entlang der Y-Achse. Drücken Sie in X-Richtung außerhalb des Würfels gegen die Stange, dreht sich der Würfel um seine Z-Achse: Sie üben ein Drehmoment aus. Das Drehmoment ergibt sich aus der Formel *Kraft mal Hebelarm*. Es gilt:

- Je größer Ihre Kraft ist, desto größer ist das Drehmoment.
- Je größer die Entfernung Ihres Angriffspunkts vom Würfel ist, desto größer ist Ihr Hebelarm und damit auch das Drehmoment.

Richtige Kategorie wählen

Dem Zylinder, dem Würfel und der Kapsel wird jeweils eine Rigidbody-Komponente zugeordnet. Das geschieht in der INSPECTOR VIEW mithilfe der Schaltfläche ADD COMPONENT. Achten Sie bei 3D-Objekten darauf, die Komponente RIGIDBODY aus der Kategorie PHYSICS zu wählen. Wählen Sie *nicht* die Komponente RIGIDBODY 2D aus der Kategorie PHYSICS 2D, denn sie ist nur für *zwei*dimensionale Objekte sinnvoll.

Keine Schwerkraft

Entfernen Sie in der Komponente RIGIDBODY die Markierung bei der Eigenschaft USE GRAVITY, sodass die 3D-Objekte keiner Schwerkraft unterliegen.

Kein Drehwiderstand

Setzen Sie den Wert für ANGULAR DRAG auf 0, damit einer Drehung kein Widerstand entgegengesetzt wird.

Klasse »Cube«

Es folgt der Code für die Drehung des Würfels um die Y-Achse:

```
using UnityEngine;
public class Cube : MonoBehaviour
{
   void Update()
   {
      if (Input.GetKeyDown (KeyCode.U))
         GetComponent<Rigidbody>().AddTorque (0, 5, 0);
      else if (Input.GetKeyDown (KeyCode.I))
         GetComponent<Rigidbody>().AddTorque (0, -5, 0);
   }
}
```

Listing 9.2 Klasse »Cube«

AddTorque()

Betätigt der Benutzer die Taste U oder die Taste I, wird die Methode `AddTorque()` der Komponente RIGIDBODY aufgerufen. Damit wird dem 3D-Objekt ein Drehmoment zugeordnet, und zwar um die Achse, die mit den nachfolgenden Parametern für `x`, `y` und `z` festgelegt wird. In der Klasse `Cube` handelt es sich jeweils um die Y-Achse. Anschließend dreht sich das 3D-Objekt immer weiter, da der Drehung kein Widerstand entgegengesetzt wird (ANGULAR DRAG = 0).

Hinweis

Ein positiver Wert erzeugt eine Drehung in positiver Drehrichtung, ein negativer Wert entsprechend in negativer Drehrichtung. Zur Verdeutlichung der Drehrichtung um eine Achse kann wiederum die linke Hand genutzt werden: Der Daumen der linken Hand weist in Richtung der positiven Achse, um die gedreht wird. Die restlichen Finger der Hand werden leicht gekrümmt. Die Fingerspitzen weisen in die positive Drehrichtung um die jeweilige Achse.

Drehung ändern

Betätigt der Benutzer die Tasten U oder I mehrfach, werden weitere Drehmomente addiert. Das 3D-Objekt dreht sich dann schneller bzw. langsamer oder ändert seine Drehrichtung.

Zylinder drehen

Es sollen zwei weitere Drehungen hinzukommen. Ergänzen Sie zunächst die Methode `Update()` der Klasse `Cylinder` wie folgt, damit eine Drehung des Zylinders um die X-Achse ermöglicht wird:

```
void Update()
{
   ...
   else if (Input.GetKeyDown(KeyCode.Y))
      GetComponent<Rigidbody>().AddTorque(5, 0, 0);
   else if (Input.GetKeyDown(KeyCode.X))
      GetComponent<Rigidbody>().AddTorque(-5, 0, 0);
}
```

Listing 9.3 Klasse »Cylinder«, Ergänzung

Kapsel drehen

Erstellen Sie wie folgt die Klasse `Capsule`, damit eine Drehung der Kapsel um die Z-Achse ermöglicht wird:

```
using UnityEngine;
public class Capsule : MonoBehaviour
{
```

```
    void Update()
    {
        if (Input.GetKeyDown (KeyCode.A))
            GetComponent<Rigidbody>().AddTorque (0, 0, 5);
        else if (Input.GetKeyDown (KeyCode.S))
            GetComponent<Rigidbody>().AddTorque (0, 0, -5);
    }
}
```

Listing 9.4 Klasse »Capsule«

Testen

Die drei C#-Scripte sind den Spielobjekten `Cube`, `Cylinder` und `Capsule` zugeordnet. Betätigen Sie die genannten Tasten, und führen Sie sich anhand der Drehungen die Achsen, Drehmomente und Drehrichtungen vor Augen. In Abbildung 9.10 sehen Sie eine Momentaufnahme, nachdem alle drei Objekte in Drehung versetzt wurden.

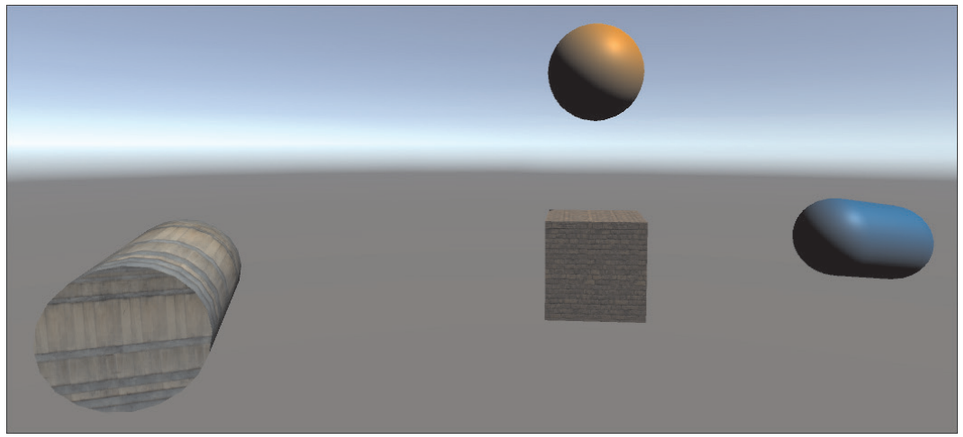

Abbildung 9.10 Zylinder, Würfel und Kapsel drehen sich

9.2.2 Animiert verschieben

Klasse »Sphere«

Ein Spielobjekt soll sich mithilfe von Programmcode wie bei einer Animation in einer bestimmten Zeit von einem Startpunkt A zu einem Zielpunkt B bewegen. Erstellen Sie die Klasse `Sphere` mit folgendem Code:

```
using UnityEngine;
public class Sphere : MonoBehaviour
{
```



```
    bool bewegung;
    float zeitGesamt;
    Vector3 startPunkt;
    Vector3 streckeGesamt;
    float bewegungZeitStart;

    void Start()
    {
        bewegung = false;
        zeitGesamt = 5;
        startPunkt = new Vector3(0, 2, 0);
        Vector3 zielPunkt = new Vector3(0, -2, 0);
        streckeGesamt = zielPunkt - startPunkt;
    }

    void Update()
    {
        if (Input.GetKeyDown (KeyCode.H) && !bewegung)
        {
            bewegung = true;
            bewegungZeitStart = Time.time;
        }

        if (bewegung)
        {
            float zeitAnteil =
                (Time.time - bewegungZeitStart) / zeitGesamt;
            Vector3 streckeAnteil = zeitAnteil * streckeGesamt;
            transform.position = startPunkt + streckeAnteil;
            if(zeitAnteil >= 1)
                bewegung = false;
        }
    }
}
```

Listing 9.5 Klasse »Sphere«, Verschiebung von A nach B

Zunächst werden einige Variablen deklariert, die innerhalb der gesamten Klasse gelten. Sie werden innerhalb der Methoden erläutert.

Gesamtzeit und Gesamtstrecke

In der Methode `Start()` werden die Daten für die gewünschte Bewegung festgelegt. Die boolesche Variable `bewegung` wird auf `false` gesetzt. Sie steht

nur während der Bewegung auf `true`. Die Gesamtzeit für die Bewegung wird auf fünf Sekunden gesetzt. Den Variablen für den Startpunkt und den Zielpunkt der Bewegung werden `Vector3`-Werte zugewiesen. Die Gesamtstrecke ist ebenfalls ein `Vector3`-Wert. Sie entspricht dem Vektor vom Startpunkt zum Zielpunkt.

In der Methode `Update()` wird die Bewegung nach Betätigung der Taste H gestartet, falls sie zurzeit nicht ausgeführt wird. Dabei wird der Startzeitpunkt festgehalten.

Anteilige Verschiebung

Nach dem Start der Bewegung wird das 3D-Objekt kontinuierlich weiterbewegt. Es wird der Anteil an der Gesamtzeit berechnet, der seit dem Startzeitpunkt vergangen ist. Daraus wird der zugehörige Anteil an der Gesamtstrecke berechnet. Anschließend wird das 3D-Objekt auf die Position gesetzt, die sich aus dem Startpunkt und diesem Streckenanteil ergibt. Die Bewegung wird gestoppt, falls die Gesamtzeit abgelaufen ist. Das 3D-Objekt hat in diesem Fall den Zielpunkt erreicht.

Testen

Ordnen Sie das C#-Script dem Spielobjekt `Sphere` zu, und testen Sie die Bewegung. Da das Spielobjekt keinen Rigidbody besitzt, bewegt es sich durch das Spielobjekt `Cube` hindurch (siehe Abbildung 9.11). Das gilt auch, wenn sich das Spielobjekt `Cube` zurzeit drehen sollte. Stellen Sie unterschiedliche Zielpunkte und Laufzeiten ein, und starten Sie die Bewegung erneut. Versehen Sie das Spielobjekt `Sphere` kurzzeitig mit einem Rigidbody, und beobachten Sie die Auswirkungen.

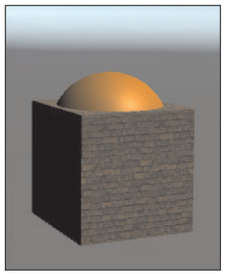

Abbildung 9.11 Kugel bewegt sich von A nach B

9.2.3 Kamera bewegen

Sie können nicht nur die Spielobjekte, sondern auch die Kamera bewegen. Damit vergrößern Sie das mögliche Spielfeld. Zudem wird das Spiel anschaulicher. Folgende Bewegungen werden verdeutlicht:

- Die Kamera wird geradlinig bewegt.
- Die Kamera wird um sich selbst gedreht.
- Die Kamera wird um einen bestimmten Punkt gedreht.

Zoomen

In Kapitel 15, »Jagen auf einem 3D-Terrain«, finden Sie außerdem ein Projekt, bei dem der Benutzer selbst mit der Kamera herauszoomen und wieder hineinzoomen kann.

Klasse »KameraBewegen«

Es folgt der Code der Klasse `KameraBewegen`:

```
using UnityEngine;
public class KameraBewegen : MonoBehaviour
{
   void Update()
   {
      if (Input.GetKeyDown (KeyCode.C))
         transform.Translate (0.2f, -0.1f, 0.7f);
      else if (Input.GetKeyDown (KeyCode.V))
         transform.Translate (-0.2f, 0.1f, -0.7f);
      else if (Input.GetKeyDown (KeyCode.O))
         transform.Rotate(0, 5, 0);
      else if (Input.GetKeyDown (KeyCode.P))
         transform.Rotate(0, -5, 0);
      else if (Input.GetKeyDown (KeyCode.J))
         transform.RotateAround (new Vector3 (-4, 0, 0),
            new Vector3 (0, 1, 0), 5);
      else if (Input.GetKeyDown (KeyCode.K))
         transform.RotateAround (new Vector3 (-4, 0, 0),
            new Vector3 (0, 1, 0), -5);
   }
}
```

Listing 9.6 Klasse »KameraBewegen«

Name der Klasse

Die Klasse erhält nicht den Namen `Camera`, weil es bereits eine gleichnamige Komponente des Spielobjekts `Main Camera` gibt.

Translate()

In der Klasse `KameraBewegen` führt die Betätigung der Taste C oder der Taste V zu einer kurzen geradlinigen Bewegung der Kamera auf der Linie zwischen der Startposition der Kamera und dem Nullpunkt des Koordinatensystems. Die Kamera ist zu Beginn bei –2/1/–7 positioniert. Die Parameter der Methode `Translate()` stehen in demselben Verhältnis zueinander.

Rotate() Die Betätigung der Taste [O] oder der Taste [P] führt zu einer 5-Grad-Drehung der Kamera um ihre eigene Y-Achse herum. Die Parameter der Methode `Rotate()` geben den Winkel an, um den das Spielobjekt um die eigene X-, Y- und Z-Achse weitergedreht wird.

RotateAround() Die Betätigung der Taste [J] oder der Taste [K] führt zu einer 5-Grad-Drehung der Kamera um eine Achse herum, die parallel zur Y-Achse des Koordinatensystems durch die Mitte des Zylinders verläuft. Die drei Parameter der Methode `RotateAround()` geben Folgendes an:

- den Punkt, um den gedreht wird (hier die Position des Zylinders)
- die Lage der Drehachse, die durch den genannten Punkt verläuft (hier eine Achse parallel zur Y-Achse)
- den Winkel, um den das Spielobjekt um die genannte Achse weitergedreht wird

Testen Das C#-Script wird dem Spielobjekt `Main Camera` zugeordnet. Betätigen Sie die genannten Tasten, und führen Sie sich die Bewegung der Kamera und die Auswirkungen vor Augen. In Abbildung 9.12 sehen Sie die Ansicht, nachdem die Taste [J] mehrmals betätigt wurde.

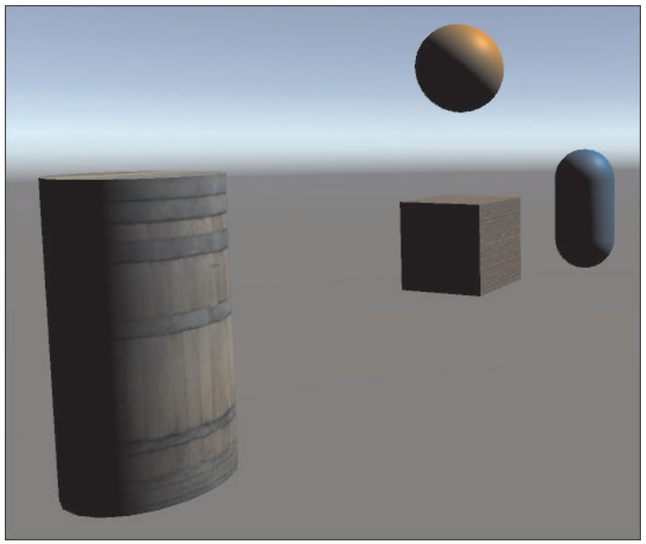

Abbildung 9.12 Kamera wird um Zylinder gedreht

9.2.4 Animiert drehen

Klasse »KameraBewegen«

Ein Spielobjekt soll sich mithilfe von Programmcode wie bei einer Animation in einer bestimmten Zeit und von einem Startwinkel A ausgehend so lange um eine bestimmte Achse drehen, bis ein Zielwinkel B erreicht ist. Das wird mithilfe des nachfolgenden Codes für das Spielobjekt `Main Camera` realisiert:

```
using UnityEngine;
public class KameraBewegen : MonoBehaviour
{
   bool bewegung;
   float zeitGesamt;
   float winkelGesamt;
   float bewegungZeitStart;
   float zeitAnteilAlt;

   void Start()
   {
      bewegung = false;
      zeitGesamt = 3;
      winkelGesamt = 90;
      zeitAnteilAlt = 0;
   }

   void Update()
   {
      if...
      else if (Input.GetKeyDown (KeyCode.L) && !bewegung)
      {
         bewegung = true;
         bewegungZeitStart = Time.time;
      }

      if (bewegung)
      {
         float zeitAnteil =
            (Time.time - bewegungZeitStart) / zeitGesamt;
         float winkelAenderung =
            (zeitAnteil - zeitAnteilAlt) * winkelGesamt;
         transform.RotateAround (Vector3.zero,
```

```
                new Vector3 (0, 1, 0), winkelAenderung);
            zeitAnteilAlt = zeitAnteil;
            Debug.Log (transform.eulerAngles.y);
            if(zeitAnteil >= 1)
                bewegung = false;
        }
    }
}
```

Listing 9.7 Klasse »KameraBewegen« mit Drehung von A nach B

Der Ablauf ähnelt demjenigen bei der Verschiebung von einem Startpunkt A zu einem Zielpunkt B aus Abschnitt 9.2.2, »Animiert verschieben«. Zunächst werden einige Variablen deklariert, die innerhalb der gesamten Klasse gelten. Sie werden innerhalb der Methoden erläutert.

Gesamtzeit und Gesamtwinkel

In der Methode `Start()` werden die Daten für die gewünschte Bewegung festgelegt. Die boolesche Variable `bewegung` wird auf `false` gesetzt. Sie steht nur während der Bewegung auf `true`. Die Gesamtzeit für die Bewegung wird auf drei Sekunden gesetzt und der Gesamtwinkel auf 90 Grad. Da sich der Winkel bei der Methode `RotateAround()` immer relativ zum vorhergehenden Winkel ändert, muss der Zeitpunkt der vorhergehenden Änderung gespeichert werden (hier in der Variablen `zeitAnteilAlt`).

In der Methode `Update()` wird die Bewegung nach Betätigung der Taste L gestartet, falls sie zurzeit nicht ausgeführt wird. Dabei wird der Startzeitpunkt festgehalten.

Anteilige Drehung

Nach dem Start der Bewegung wird das 3D-Objekt kontinuierlich weitergedreht. Es wird der Anteil an der Gesamtzeit berechnet, der seit dem Startzeitpunkt vergangen ist. Anschließend wird der Winkel berechnet, um den sich das Spielobjekt seit der letzten Änderung weiterdrehen soll. Diese Drehung wird mithilfe der Methode `RotateAround()` ausgeführt und findet hier um die Y-Achse herum statt, die durch das Zentrum des Koordinatensystems verläuft. Der aktuelle Zeitanteil wird zur Durchführung der nächsten Änderung gespeichert. Die Bewegung wird gestoppt, falls die Gesamtzeit abgelaufen ist.

Drehwinkel

Der aktuelle Drehwinkel um die Y-Achse wird mithilfe der statischen Methode `Log()` der Klasse `Debug` zur Kontrolle ausgegeben. Die Transform-Komponente besitzt die Untereigenschaft `eulerAngles`. Diese hat den Typ

`Vector3` und gibt die Drehung um die X-Achse, um die Y-Achse und um die Z-Achse in einem Winkel von 0 bis 360 Grad an.

Testen

Starten Sie die Bewegung. Stellen Sie unterschiedliche Zielwinkel und Laufzeiten ein, und starten Sie die Bewegung erneut. Beachten Sie auch die Kontrollausgabe des Drehwinkels in der Statuszeile.

9.2.5 Übersicht

Alle Tastencodes

Es folgt eine Tabelle mit allen Tastencodes im Projekt *DreiDimensionen* in der Reihenfolge ihres Einsatzes in diesem Kapitel:

Taste	Erläuterung
B	Zylinder erhält Material mit hellblauer Farbe.
F	Zylinder erhält Material mit Holzfass-Textur.
U	Würfel erhält positives Drehmoment um die Y-Achse.
I	Würfel erhält negatives Drehmoment um die Y-Achse.
Y	Zylinder erhält positives Drehmoment um die X-Achse.
X	Zylinder erhält negatives Drehmoment um die X-Achse.
A	Kapsel erhält positives Drehmoment um die Z-Achse.
S	Kapsel erhält negatives Drehmoment um die Z-Achse.
H	Kugel wird animiert von Punkt A nach Punkt B verschoben.
C	Kamera wird verschoben, vom Betrachter weg.
V	Kamera wird verschoben, zum Betrachter hin.
O	Kamera wird positiv um ihre Y-Achse gedreht.
P	Kamera wird negativ um ihre Y-Achse gedreht.
J	Kamera wird positiv um eine Achse gedreht, die parallel zur Y-Achse steht.
K	Kamera wird negativ um eine Achse gedreht, die parallel zur Y-Achse steht.
L	Kamera wird animiert um einen Punkt von Winkel A nach Winkel B gedreht.

Tabelle 9.1 Tastencodes im Projekt

Time.deltaTime

Möglicherweise vermissen Sie bei den Bewegungen und Drehungen den Faktor `Time.deltaTime`. Im vorliegenden Projekt gibt es aber nur

- einmalige Bewegungen auf eine neue Position,
- einmalige Drehungen auf eine neue Rotationsposition und
- kontinuierliche Änderungen mithilfe einer eigenen Zeitsteuerung.

Daher wird der Wert von `Time.deltaTime` nicht benötigt.