

Thomas Theis



4., aktualisierte Auflage

Einstieg in Unity

2D- und 3D-Spiele entwickeln

*Ideal für
Programmier-
anfänger*

- ▶ Schritt für Schritt zum eigenen Computerspiel
- ▶ Mit C#-Programmierkurs für Unity-Skripte
- ▶ 18 fertige Spiele als Vorlage für eigene Ideen



Alle Beispielprojekte zum Download



Rheinwerk
Computing

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?

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.

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, auch unter physikalischen Bedingungen.

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 Ihren Wünschen entsprechend weiter zu verändern und selbstständig eigene Spiele zu entwickeln.

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, doch 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 detailreich gestaltete Spielfiguren können zudem nur mit externen Programmen erstellt werden und müssen danach zunächst in Unity importiert werden.

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 des Rheinwerk Verlags, ganz besonders bei Anne Scheibe und Almut Poll.

1.2 Wie entsteht der programmierte Spielablauf?

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 zusätzlich eine kostenfreie Version der Entwicklungsumgebung *Visual Studio* der Firma Microsoft angeboten.

In Kapitel 19 finden Sie zusätzlich einen reinen C#-Programmierungskurs, 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#-Programmierungskurs auch 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

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

Zunächst wird der *Unity Hub* installiert. Dabei handelt es sich um eine Anwendung zur Verwaltung

- Ihrer Unity-Projekte und
- Ihrer Unity-Editoren, mit denen Sie Ihre Unity-Projekte entwickeln.

Sie finden den Unity Hub mithilfe Ihres Browsers über die folgende Adresse:

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

Scrollen Sie auf der Seite ein wenig nach unten und betätigen Sie für Windows den Link **DOWNLOAD FÜR WINDOWS**, siehe Abbildung 1.1. Es wird die Datei *UnityHubSetup.exe* heruntergeladen – aktuell (im Februar 2023) mit einer Größe von ca. 112 MB –, 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.

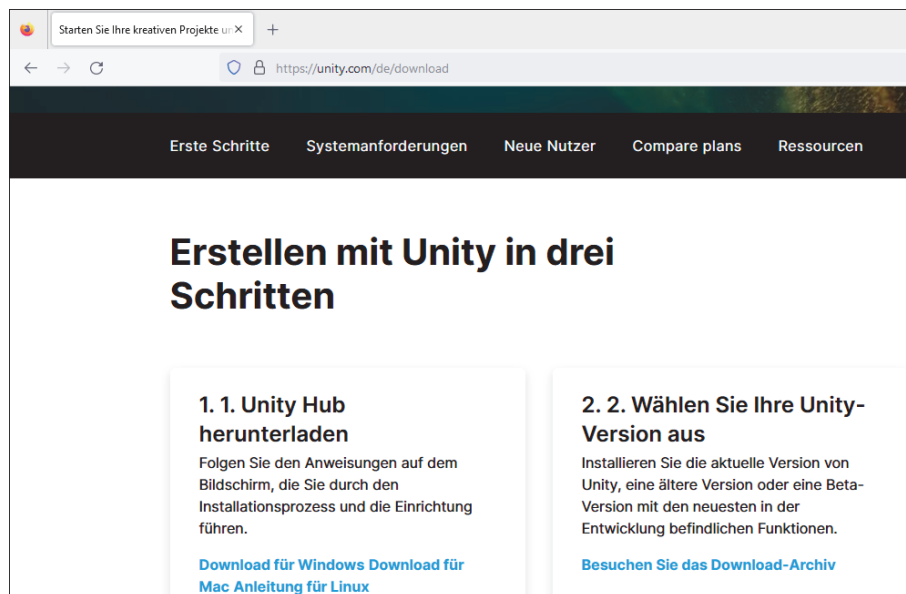


Abbildung 1.1 Unity-Website, Unity Hub herunterladen

Da der Unity Hub nicht aus dem Windows Store stammt, müssen Sie zu Beginn der Installation bestätigen, dass er installiert werden soll. Anschließend 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.

Es erscheint das Dialogfeld **INSTALL UNITY EDITOR**. Hier könnten Sie die LTS-Version des Unity Editors installieren, aktuell 2021.3.16.f1 LTS. Dabei handelt es sich um eine Version, die lange Zeit unterstützt wird (engl.: *Long Term Support*, kurz *LTS*).

In diesem Buch wird allerdings mit einer neueren Version gearbeitet, und zwar mit Version 2022.2.1. Diese Installation wird erst später durchgeführt. Daher betätigen Sie hier den Link **SKIP INSTALLATION**. Es erscheint der Unity Hub mit verschiedenen Reitern auf der linken Seite, aktuell in der Version 3.4.1, siehe Abbildung 1.2.

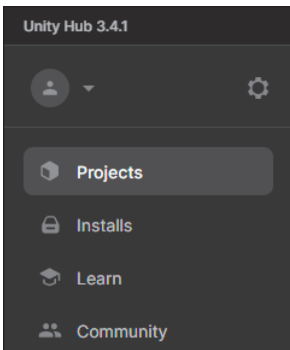


Abbildung 1.2 Unity Hub

1.5 Unity-Lizenz erhalten

Für die Arbeit mit diesem Buch benötigen Sie die kostenfreie Lizenz *Unity Personal*. Über die Schaltfläche **PREFERENCES**, erkennbar am Zahnrad-Symbol in Abbildung 1.2, gelangen Sie auf das gleichnamige Dialogfeld, siehe Abbildung 1.3.

Betätigen Sie ganz rechts auf der Registerkarte **LICENSES** den Button **ADD** und wählen Sie im Dialogfeld **ADD NEW LICENSE** den Link **GET A FREE PERSONAL LICENSE**. Anschließend erhalten Sie Ihre Lizenz durch Zustimmung zu den Bedingungen über die Schaltfläche **AGREE AND GET PERSONAL EDITION LICENSE**. Während des Lizenzierungsvorgangs müssen Sie sich bei den Erstellern von Unity registrieren.

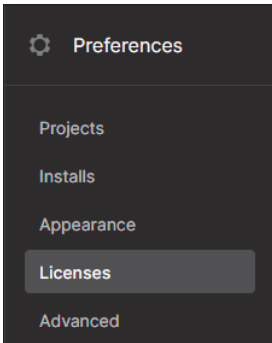


Abbildung 1.3 Preferences

Anschließend erscheint im Dialogfeld PREFERENCES die Lizenz PERSONAL mit Aktivierungsdatum. Schließen Sie das Dialogfeld wieder.

Sobald eine neue Version des Unity Hubs erscheint, wird Ihnen ein Update vorgeschlagen. Ich empfehle Ihnen, dem Vorschlag zu folgen.

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 mehrerer Versionen von Unity und den schnellen Umstieg zwischen den Versionen zu ermöglichen.

1.6 Unity-Version installieren

Aufgrund der kurzen Zeitabstände beim Erscheinen neuer Unity-Versionen kann es gut sein, dass Sie eine Version installieren, die neuer ist als die Version 2022.2.1, die ich für dieses Buch genutzt habe. Das ist aber kein Problem.

Zur Installation einer Unity-Version betätigen Sie im Unity Hub auf der Registerkarte INSTALLS ganz rechts die Schaltfläche INSTALL EDITOR. Es erscheint das Dialogfeld INSTALL UNITY EDITOR, siehe Abbildung 1.4. Betätigen Sie auf der Registerkarte OFFICIAL RELEASES die Schaltfläche INSTALL bei der für Sie aktuellen Version. In meinem Fall ist die Version 2022.2.1 bereits installiert.

Es erscheint das Dialogfeld INSTALL UNITY Hier ist die Standardoption MICROSOFT VISUAL STUDIO COMMUNITY 2022 bereits markiert. Sie können diese Option beibehalten.

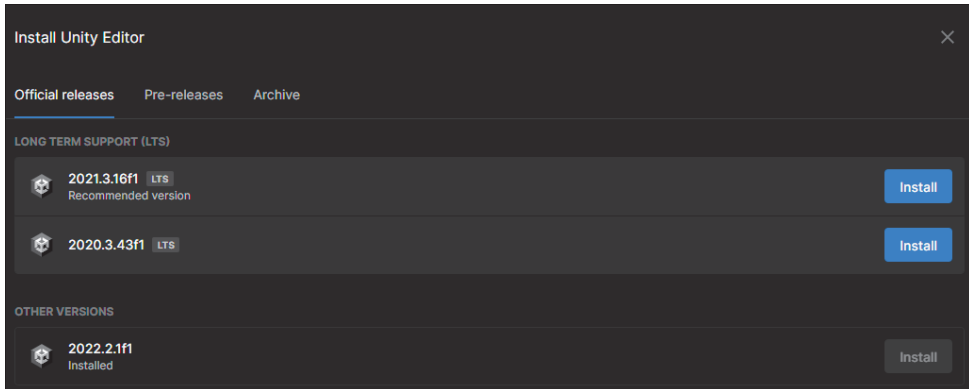


Abbildung 1.4 Unity-Version hinzufügen

Weitere Komponenten werden zurzeit noch nicht benötigt, können aber jederzeit nachinstalliert werden. Nach Betätigung der Schaltfläche CONTINUE akzeptieren Sie auf dem nächsten Dialogfeld die Lizenzbedingungen und betätigen anschließend die Schaltfläche INSTALL. Der ausgewählte Unity Editor wird installiert. Anschließend wird der *Visual Studio Installer* installiert, danach *Visual Studio* selbst.

In Ihrem ersten Unity-Projekt werden Sie in Abschnitt 4.3.1, »Stellen Sie ›Visual Studio‹ ein«, eine weitere notwendige Einstellung vornehmen, damit *Visual Studio* anschließend in Ihren sämtlichen Unity-Projekten automatisch genutzt wird.

1.7 Beispielprojekte und Assets

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/einstieg-in-unity-2d-und-3d-spiele-entwickeln> 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.

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.

In Abschnitt 22.6 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 nach Bearbeitung dieses Buchs erworben haben, sind Sie in der Lage, den Aufbau dieser Projekte zu erkennen und sie mit eigenen Ideen selbstständig weiterzuentwickeln.

1.8 Upgrade eines Unity-Projekts

Wie bereits erwähnt, haben Sie möglicherweise eine neuere Version installiert als die Version 2022.2.1, die ich für dieses Buch genutzt habe. Sie stoßen gegebenenfalls auf diesen Zusammenhang, falls Sie eines meiner Beispielprojekte nach dem Kopieren öffnen möchten.

Es ist aber kein Problem, mithilfe des Unity Hubs ein Upgrade für ein Projekt durchzuführen. Es kann auch vorkommen, dass die Entwicklung eines eigenen Unity-Projekts zunächst unter einer bestimmten Version beginnt und später unter einer anderen Version fortgesetzt wird.

Nehmen wir an, Sie haben das Beispielprojekt *TomsJumpAndRun* kopiert, das Sie in Kapitel 3, »Spielen Sie ein 2D-Jump&Run-Spiel«, genauer kennenlernen werden. Zunächst soll die neue Version des Unity Editors ausgewählt werden. Klicken Sie dazu in der Registerkarte PROJECTS auf die Versionsbezeichnung, die Sie in der Zeile des betreffenden Projekts in der Spalte EDITOR VERSION finden.

Es erscheint ein Dialogfeld mit den verschiedenen Optionen für die aktuell installierten Unity-Versionen. Wählen Sie die gewünschte Option aus und betätigen Sie die Schaltfläche OPEN WITH ..., anschließend die Schaltfläche CHANGE VERSION im Dialogfeld CHANGE EDITOR VERSION und die Schaltfläche CONTINUE im Dialogfeld OPENING PROJECT ...

Die Konvertierung nimmt eine gewisse Zeit in Anspruch. Anschließend kann das Unity-Projekt weiterbearbeitet werden. Der Wechsel auf eine ältere Version wäre ebenso möglich, auf die gleiche Art und Weise.

Die Beispielprojekte, die Screenshots und die Erläuterungen in diesem Buch sind mithilfe der Unity-Version 2022.2.1 entstanden.

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

Kapitel 9

Das erste 3D-Projekt

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.

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

Erstellen Sie ein neues Projekt mit dem Namen *DreiDimensionen*. Wählen Sie das Template 3D CORE aus. Damit erhalten 3D-Projekte die passenden Voreinstellungen.

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.

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:

- ▶ Erzeugen Sie einen Würfel mithilfe des Menüpunkts 3D OBJECT • CUBE. Er verbleibt im Zentrum der Darstellung an der Position 0/0/0. Die Namen der 3D-Spielobjekte bleiben in diesem Projekt unverändert. Der Würfel heißt also *Cube*.
- ▶ 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.
- ▶ 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.
- ▶ 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.

Nach der Markierung des Spielobjekts *Cube* im Zentrum der Darstellung sind die Achsen gut zu erkennen. In der SCENE VIEW ergibt sich ein Bild wie in Abbildung 9.1.

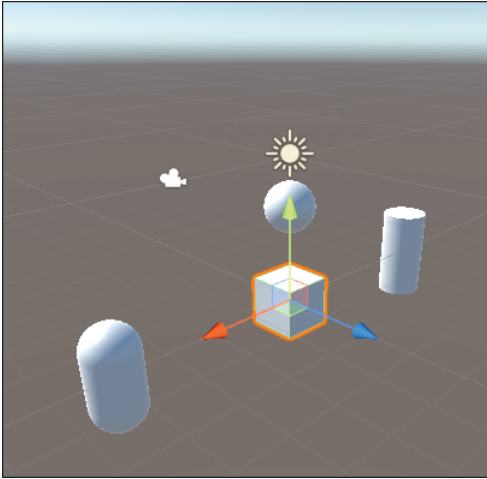


Abbildung 9.1 Einfache 3D-Objekte in der »Scene View«

Hinweis

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

9.1.3 Farbiges Oberflächenmaterial

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*.

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

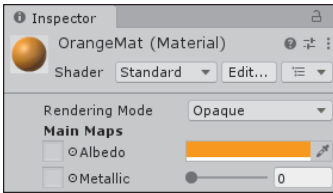


Abbildung 9.2 Wichtigste Farbe für ein Oberflächenmaterial

Zur Zuordnung der Oberflächenmaterialien klappen Sie bei den einzelnen Spielobjekten in der Komponente MESH RENDERER den Bereich MATERIALS auf, siehe Abbildung 9.3. Ziehen Sie das jeweilige Material-Asset auf die Eigenschaft ELEMENT 0. Die Kugel wird orangefarben, die Kapsel hellblau, siehe Abbildung 9.9.

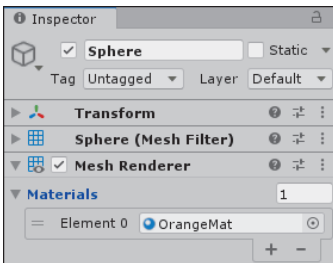


Abbildung 9.3 Oberflächenmaterial zuordnen

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.

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

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

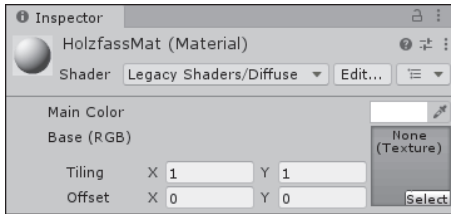


Abbildung 9.4 Typ der Schattierung einstellen

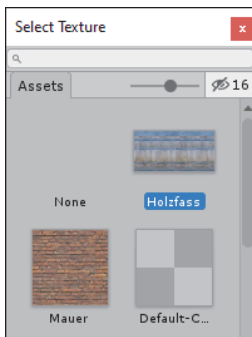


Abbildung 9.5 Textur auswählen

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.

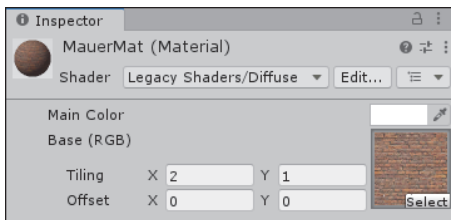


Abbildung 9.6 Tiling einstellen

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.

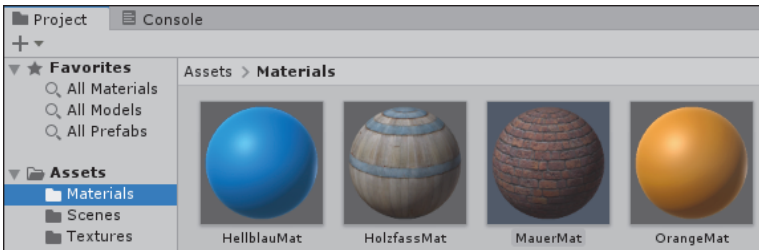


Abbildung 9.7 Vier Materialien

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

9.1.5 Oberflächenmaterial wechseln

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«

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.

Das C#-Script im neuen Asset-Verzeichnis `SCRIPTS` 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

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 Handsymbol in der kleinen Symbolleiste `TOOLS`, die Sie links oben in der `SCENE VIEW` sehen:

- ▶ *Verschieben* Sie die Ansicht in der `SCENE VIEW` durch Verschieben der Maus bei gedrückter linker Maustaste nach links, rechts, oben oder unten.
- ▶ *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.
- ▶ *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.

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



Abbildung 9.8 Gizmo

Hinweis

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.

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

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.

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 ganz einfach, stellt aber eine gute Übung dar.

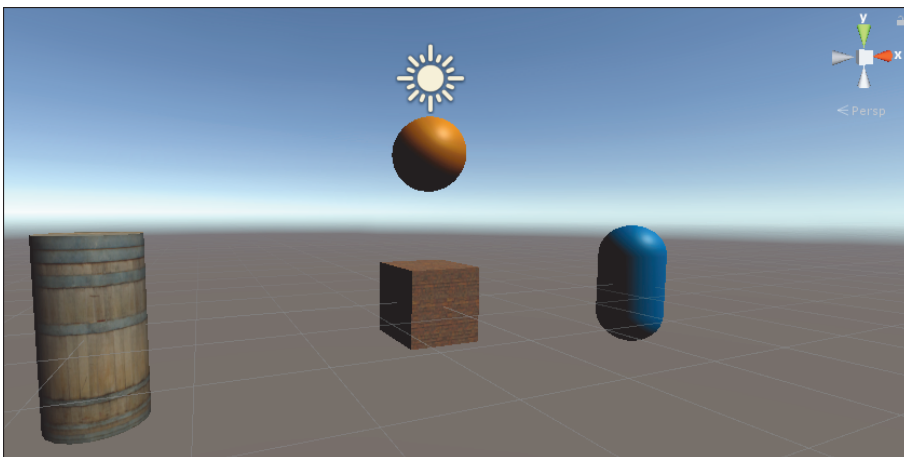


Abbildung 9.9 Gewünschte Ansicht

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. Dabei werden die Transformwerte der Kamera geändert.

9.2 Verschieben und Drehen

Bei der Verschiebung und besonders bei der Drehung von 3D-Objekten sollten Sie eine gute Vorstellung vom dreidimensionalen Raum 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.

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 zweidimensionale Objekte sinnvoll.

Für alle drei Spielobjekte gilt: Entfernen Sie in der Komponente `RIGIDBODY` die Markierung bei der Eigenschaft `USE GRAVITY`, sodass die 3D-Objekte keiner Schwerkraft unterliegen. Setzen Sie den Wert für `ANGULAR DRAG` auf 0, damit einer Drehung kein Widerstand entgegengesetzt wird.

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«

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.

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

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.E))
    GetComponent<Rigidbody>().AddTorque(5, 0, 0);
else if (Input.GetKeyDown(KeyCode.R))
    GetComponent<Rigidbody>().AddTorque(-5, 0, 0);
}

```

Listing 9.3 Klasse »Cylinder«, Ergänzung

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«

Die drei C#-Scripte sind den Spielobjekten *Cube*, *Cylinder* und *Capsule* zugeordnet. Betätigen Sie die im Programmcode notierten 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.

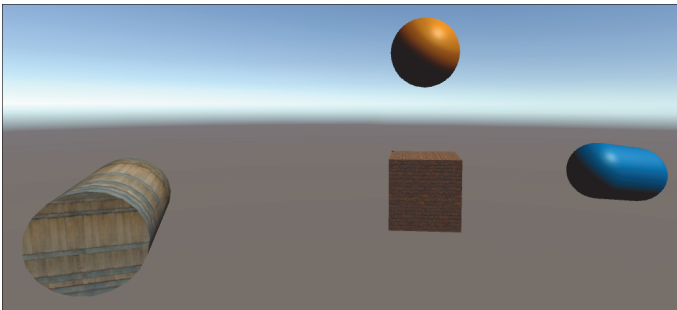


Abbildung 9.10 Zylinder, Würfel und Kapsel drehen sich

9.2.2 Animiert verschieben

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 (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.

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.

Hinweis

Wird einem neu deklarierten Verweis unmittelbar ein neues Objekt desselben Typs zugewiesen, kann bei der Erzeugung des Objekts der Name des Typs weggelassen werden, also:

```

Vector3 zielPunkt = new (0, -2, 0); statt
Vector3 zielPunkt = new Vector3 (0, -2, 0);

```

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

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.

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). Stellen Sie unterschiedliche Zielpunkte und Laufzeiten ein, und starten Sie die Bewegung erneut. Versehen Sie das Spielobjekt `Sphere` kurzzeitig mit einem `Rigidbody` (ohne Schwerkrafteinwirkung), und beobachten Sie die Auswirkungen.

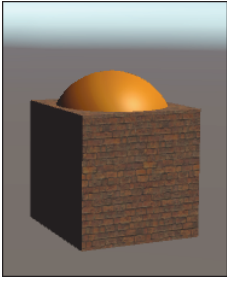


Abbildung 9.11 Kugel bewegt sich von A nach B

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.

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.

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«

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 (bei $-2/1/-7$) und dem Nullpunkt des Koordinatensystems. Die Parameter der Methode `Translate()` stehen in demselben Verhältnis zueinander.

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

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 nicht gedrehten 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

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.

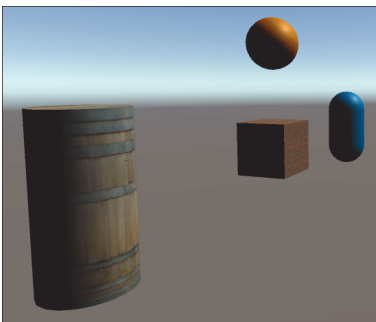


Abbildung 9.12 Kamera wird um Zylinder gedreht

9.2.4 Animiert drehen

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 (new Vector3 (0, 0, 0),
                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.

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 ☐ gestartet, falls sie zurzeit nicht ausgeführt wird. Dabei wird der Startzeitpunkt festgehalten.

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.

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.

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

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.
E	Zylinder erhält positives Drehmoment um die X-Achse.
R	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 »DreiDimensionen«

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.

Inhalt

1	Einführung	17
1.1	Was machen wir mit Unity?	17
1.2	Wie entsteht der programmierte Spielablauf?	18
1.3	Dateiendungen anzeigen lassen	18
1.4	Unity Hub installieren	19
1.5	Unity-Lizenz erhalten	20
1.6	Unity-Version installieren	21
1.7	Beispielprojekte und Assets	22
1.8	Upgrade eines Unity-Projekts	23
2	Das erste 2D-Projekt	25
2.1	Erstellung eines neuen Projekts	25
2.2	Wichtige Bereiche im Unity Editor	26
2.3	Das Spielobjekt »Main Camera«	28
2.4	Assets importieren	29
2.5	Spielobjekte einfügen	30
2.6	Ändern der Hierarchie	31
2.7	Eine Szene speichern	33
2.8	Die Komponente »Transform«	33
2.8.1	Die Eigenschaften der Transform-Komponente	34
2.8.2	Werte in der »Inspector View« ändern	35
2.9	Die Ansicht in der »Scene View«	36
2.9.1	Positionswerte mithilfe der Maus ändern	37
2.9.2	Rotationswerte mithilfe der Maus ändern	38
2.9.3	Scale-Werte mithilfe der Maus ändern	39

3	Spieren Sie ein 2D-Jump&Run-Spiel	41
3.1	Wie geht das Spiel?	41
3.2	Unsere ersten Unity-Elemente	43
3.2.1	Assets	44
3.2.2	Spielobjekte	45
4	Entwickeln Sie ein 2D-Jump&Run-Spiel	49
4.1	Erzeugen Sie Projekt und Assets	49
4.2	Fügen Sie Spielobjekte ein	50
4.2.1	Schaffen Sie einen Hintergrund	50
4.2.2	Erzeugen Sie das Spielfeld	51
4.2.3	Setzen Sie den Affen auf den Boden	53
4.3	Erstellen Sie den Spielablauf	56
4.3.1	Stellen Sie »Visual Studio« ein	56
4.3.2	Führen Sie die Klasse »Spieler« ein	57
4.3.3	Bewegen Sie den Affen	61
4.3.4	Anzeige von Fehlern	65
4.3.5	Verhindern Sie die Drehung	66
4.3.6	Begrenzen Sie die Bewegung zur Seite	67
4.3.7	Treffen Sie die Bananen	68
4.3.8	Meiden Sie die Tiger	72
4.3.9	Die geschweiften Klammern	74
4.3.10	Die Tiger bewegen sich	75
4.4	Gestalten Sie die Benutzeroberfläche	77
4.4.1	Erstellen Sie die erste Anzeige	77
4.4.2	Sammeln Sie Punkte	79
4.4.3	Verlieren Sie Leben	82
4.4.4	Messen Sie die Spielzeit	83
4.4.5	Speichern Sie Werte dauerhaft	85
4.4.6	Geben Sie den Benutzern Hinweise	87
4.4.7	Starten Sie ein neues Spiel	91
4.4.8	Beenden Sie die Anwendung	94
4.4.9	Ideen für Ihre Erweiterungen	95
4.5	Erzeugen Sie eine ausführbare Version	96

5 Ein 2D-Breakout-Spiel 99

5.1	Führen Sie das Spiel aus	99
5.2	Erzeugen Sie Projekt und Assets	100
5.2.1	Fügen Sie ein Audio-Asset ein	101
5.2.2	Erstellen Sie ein 2D-Material	101
5.2.3	Lernen Sie 2D-Materialien kennen	102
5.2.4	Erzeugen Sie ein Prefab	104
5.3	Fügen Sie Spielobjekte ein	105
5.3.1	Füllen Sie das Spielfeld	105
5.3.2	Erzeugen Sie einen Ziegel	106
5.3.3	Wiederholen Sie den Vorgang	108
5.3.4	Wiederholen Sie die Wiederholung	109
5.3.5	Erzeugen Sie unterschiedliche Ziegel	111
5.4	Erstellen Sie den Spielablauf	112
5.4.1	Senden Sie den Ball ab	112
5.4.2	Bewegen Sie den Spieler	114
5.4.3	Sammeln Sie Punkte	115
5.4.4	Verlieren Sie Leben	119
5.5	Gestalten Sie die Benutzeroberfläche	122
5.5.1	Exportieren Sie ein Unity Package	122
5.5.2	Importieren Sie ein Unity Package	123
5.5.3	Passen Sie die Benutzeroberfläche an	124
5.5.4	Punkte, Leben und Infos anzeigen	126
5.5.5	Messen Sie die Spielzeit	127
5.5.6	Zeigen Sie die vorherige Zeit an	129
5.5.7	Starten Sie ein neues Spiel	130
5.5.8	Beenden Sie die Anwendung	134
5.5.9	Ideen für Ihre Erweiterungen	135

6 Ein 2D-Spiel für zwei Spieler 137

6.1	Führen Sie das Spiel aus	137
6.2	Bereiten Sie das Spiel vor	139
6.2.1	Erzeugen Sie Projekt und Assets	139
6.2.2	Erzeugen Sie Spielfeld und UI	140
6.2.3	Gestalten Sie das Spielfeld	142

6.3	Erstellen Sie den Spielablauf	145
6.3.1	Führen Sie den Aufschlag aus	145
6.3.2	Bewegen Sie die Spieler vertikal	147
6.3.3	Bewegen Sie die Spieler horizontal	150
6.3.4	Sammeln Sie Punkte	152
6.3.5	Eine kleine Übung	154
6.3.6	Ideen für Ihre Erweiterungen	154
6.4	Künstliche Intelligenz	155
7	Ein Gedächtnistrainer als 2D-Projekt	157
7.1	Führen Sie das Training aus	157
7.2	Bereiten Sie das Training vor	158
7.3	Das Training für drei Zahlen	159
7.3.1	Verteilen Sie die Zahlen	159
7.3.2	Vermeiden Sie doppelte Positionen	162
7.3.3	Löschen Sie die Zahlen	164
7.3.4	Prüfen Sie die Reihenfolge	165
7.4	Die Erweiterung des Trainings	168
7.4.1	Machen Sie das Training leichter	168
7.4.2	Machen Sie das Training schwerer	169
7.4.3	Optimieren Sie das Training	172
7.4.4	Ideen für Ihre Erweiterungen	174
8	Ein 2D-Space-Shooter	177
8.1	Bereiten Sie das Spiel vor	178
8.1.1	Gestalten Sie die beiden Explosions-Prefabs	179
8.1.2	Erzeugen Sie Ihr Raumschiff und die Geschosse	180
8.1.3	Erstellen Sie die anderen Raumschiffe	181
8.1.4	Gestalten Sie die Energieanzeige	182
8.1.5	Erstellen Sie die Benutzeroberfläche	183
8.2	Erstellen Sie den Spielablauf	183
8.2.1	Bewegen Sie Ihr Raumschiff, und feuern Sie	184
8.2.2	Bewegen Sie die Geschosse nach dem Abfeuern	186

8.2.3	Bewegen Sie die anderen Raumschiffe	187
8.2.4	Lassen Sie die Raumschiffe explodieren	188
8.2.5	Kollidieren Sie mit den anderen Raumschiffen	190
8.2.6	Führen Sie weitere Änderungen der Energie herbei	192
8.2.7	Messen Sie die Zeit, und beenden Sie das Spiel	194
8.2.8	Ausführbare Version	196
8.2.9	Eine kleine Übung	196
8.2.10	Ideen für Ihre Erweiterungen	196

9 Das erste 3D-Projekt 199

9.1	Grundlagen eines 3D-Projekts	199
9.1.1	Kamera, Skybox und Licht	199
9.1.2	Einfache 3D-Objekte	200
9.1.3	Farbiges Oberflächenmaterial	201
9.1.4	Oberflächenmaterial mit Textur	202
9.1.5	Oberflächenmaterial wechseln	204
9.1.6	Ansicht in der »Scene View« gestalten	205
9.2	Verschieben und Drehen	207
9.2.1	Spielobjekte drehen	207
9.2.2	Animiert verschieben	210
9.2.3	Kamera bewegen	212
9.2.4	Animiert drehen	214
9.2.5	Übersicht	216

10 Eine 3D-Animation 217

10.1	Schaffen Sie die Voraussetzungen	217
10.1.1	Betrachten Sie die fertige Animation	217
10.1.2	Bauen Sie das Beispiel auf	218
10.2	Erstellen Sie die Animation	220
10.2.1	Legen Sie die Animation an	220
10.2.2	Drehen Sie das rechte Bein	221
10.2.3	Erstellen Sie weitere Keyframes	222
10.2.4	Stellen Sie die Keyframes ein	223
10.2.5	Verschieben Sie das rechte Bein	224

10.3	Arbeiten Sie mit dem »Animator Controller«	225
10.3.1	Gestalten Sie die States	225
10.3.2	Erstellen Sie die Parameter	226
10.3.3	Erzeugen Sie die Transitions	226
10.4	Fügen Sie das C#-Script hinzu	227
10.4.1	Verbinden Sie Bewegung und Animation	228
10.4.2	Vervollständigen Sie die Animation	229
10.4.3	Ideen für Ihre Erweiterungen	230

11 Ein 3D-Balancer 231

11.1	Führen Sie das Spiel aus	231
11.2	Bereiten Sie das Spiel vor	232
11.2.1	Erzeugen Sie Projekt und Assets	232
11.2.2	Erzeugen Sie Spielfeld und UI	233
11.2.3	Relative Transform-Werte	234
11.3	Erstellen Sie den Spielablauf	235
11.3.1	Drehen Sie die Platte	236
11.3.2	Bewegen Sie den Lederball und die Kamera	238
11.3.3	Ändern Sie die Punktzahl	240
11.3.4	Ideen für Ihre Erweiterungen	241

12 Ein 3D-Tetris 243

12.1	Führen Sie das Spiel aus	243
12.2	Bereiten Sie das Spiel vor	244
12.2.1	Erzeugen Sie Projekt und Assets	244
12.2.2	Erzeugen Sie Spielfeld und UI	245
12.2.3	Erstellen Sie das Würfel-Prefab	246
12.3	Erstellen Sie den Spielablauf	247
12.3.1	Bewegen Sie die Würfel	247
12.3.2	Eine »generische Liste«	248
12.3.3	Fügen Sie Elemente zur Liste hinzu	249
12.3.4	Entfernen Sie Elemente aus der Liste	252
12.3.5	Eine kleine Übung	255
12.3.6	Ideen für Ihre Erweiterungen	255

13 Ein Kopfrechentainer als 3D-Projekt 257

13.1 Führen Sie das Training aus	257
13.2 Bereiten Sie das Training vor	258
13.3 Erstellen Sie den Trainingsablauf	259
13.3.1 Erzeugen Sie die Aufgabe und die Lösungen	259
13.3.2 Mischen Sie die Lösungen	262
13.3.3 Sammeln Sie Punkte	264
13.3.4 Verlieren Sie Leben	266
13.3.5 Ideen für Ihre Erweiterungen	269

14 Golf spielen auf einem 3D-Terrain 271

14.1 Führen Sie das Spiel aus	271
14.2 Bereiten Sie das Spiel vor	272
14.2.1 Erzeugen Sie Projekt und Landschaft	272
14.2.2 Weisen Sie der Landschaft eine Textur zu	273
14.2.3 Erstellen Sie die drei Ebenen	274
14.2.4 Fügen Sie den Rand hinzu	276
14.2.5 Erstellen Sie die beiden Rampen	278
14.2.6 Setzen Sie Spieler und Ziel in die Landschaft	279
14.2.7 Arbeiten Sie mit einem »Physic Material«	280
14.3 Erstellen Sie den Spielablauf	281
14.3.1 Schlagen Sie den Spielball	281
14.3.2 Versetzen Sie das Ziel	284
14.3.3 Vermeiden Sie den Verlust des Spielballs	286
14.3.4 Ideen für Ihre Erweiterungen	287
14.4 Ein weiteres Terrain	288
14.4.1 Erzeugen Sie zehn Ebenen	288
14.4.2 Fügen Sie den linken und den rechten Rand hinzu	290
14.4.3 Fügen Sie den unteren und den oberen Rand hinzu	290
14.4.4 Erzeugen Sie die erste Rampe	291
14.4.5 Erstellen Sie alle Rampen links	292
14.4.6 Erstellen Sie alle Rampen rechts	292
14.4.7 Setzen Sie die Positionen	293

15 Jagen auf einem 3D-Terrain 295

15.1 Führen Sie das Spiel aus	295
15.2 Bereiten Sie das Spiel vor	297
15.2.1 Erzeugen Sie Projekt und Landschaft	298
15.2.2 Steuern Sie den Zufall	298
15.2.3 Erzeugen Sie die weiteren Spielobjekte	301
15.2.4 Erstellen Sie die drei Prefabs	303
15.2.5 Zoomen Sie mithilfe eines Sliders	303
15.3 Erstellen Sie den Spielablauf	306
15.3.1 Bewegen Sie den Jäger	306
15.3.2 Treffen Sie die Ziele	308
15.3.3 Die Ziele starten eine Abwehr	311
15.3.4 Die Abwehr wird gefährlich	312
15.3.5 Messen Sie die Zeit	314
15.3.6 Ideen für Ihre Erweiterungen	316

16 Eine Schlange aus 3D-Joints 317

16.1 Führen Sie das Spiel aus	317
16.2 Bereiten Sie das Spiel vor	318
16.2.1 Erzeugen Sie Assets und Platte	318
16.2.2 Erstellen Sie Schlange und Beute	319
16.2.3 Stellen Sie die gelenkigen Verbindungen her	320
16.3 Erstellen Sie den Spielablauf	321
16.3.1 Bewegen Sie die Schlange	322
16.3.2 Treffen Sie die Beute	323
16.3.3 Verkürzen Sie die Schlange	324
16.3.4 Zählen Sie die Punkte	326
16.3.5 Die Segmente treffen den Rand	327
16.3.6 Messen Sie die Zeit	328
16.3.7 Ideen für Ihre Erweiterungen	329

17 Ein Renntraining und ein Autorennen 331

17.1 Führen Sie das Renntraining aus	331
17.2 Führen Sie das Autorennen aus	333
17.3 Bereiten Sie das Renntraining vor	334
17.3.1 Erzeugen Sie Projekt und Fahrbahn	335
17.3.2 Konstruieren Sie das Fahrzeug	336
17.3.3 Fügen Sie die Wheel Collider hinzu	337
17.4 Erstellen Sie den Ablauf des Renntrainings	339
17.4.1 Beschleunigen Sie das Fahrzeug	339
17.4.2 Lenken Sie das Fahrzeug	340
17.4.3 Folgen Sie dem Fahrzeug mit der Kamera	342
17.4.4 Bauen Sie die Begrenzungen auf	344
17.4.5 Eine »Lichtschranke« an der Startlinie	346
17.4.6 Messen Sie die Rundenzeiten	347
17.5 Erweitern Sie das Renntraining zum Autorennen	350
17.5.1 Erzeugen Sie das zweite Fahrzeug	350
17.5.2 Steuern Sie die Fahrzeuge getrennt	351
17.5.3 Teilen Sie den Bildschirm auf	354
17.5.4 Eine dritte Kamera für den Überblick	355
17.5.5 Getrennte Rundenzeiten nach einem Countdown	356
17.5.6 Ideen für Ihre Erweiterungen	360

18 Erkunden Sie das Verlies 361

18.1 Führen Sie das Spiel aus	361
18.2 Bereiten Sie das Spiel vor	366
18.2.1 Die Planung des Verlieses	366
18.2.2 Der Aufbau einer Kammer	367
18.2.3 Erstellen Sie die ersten Spielobjekte	368
18.2.4 Bauen Sie das Prefab für die Kammer	369
18.2.5 Die Schlüssel, Schatzkisten und Sperren	371
18.2.6 Gestalten Sie die Benutzeroberfläche	373

18.3 Erstellen Sie den Spielablauf	375
18.3.1 Folgen Sie dem Spieler mit der Kamera	375
18.3.2 Erstellen Sie alle Kammern	377
18.3.3 Konfigurieren Sie die Kammern	378
18.3.4 Gehen Sie durch ein Tor	380
18.3.5 Nehmen Sie den Schlüssel aus einer Schatzkiste	383
18.3.6 Schließen Sie eine Sperre auf	387
18.3.7 Speichern Sie den Spielstand	389
18.3.8 Laden Sie den alten Spielstand	390
18.3.9 Ideen für Ihre Erweiterungen	392

19 Ein Programmierkurs in C# 393

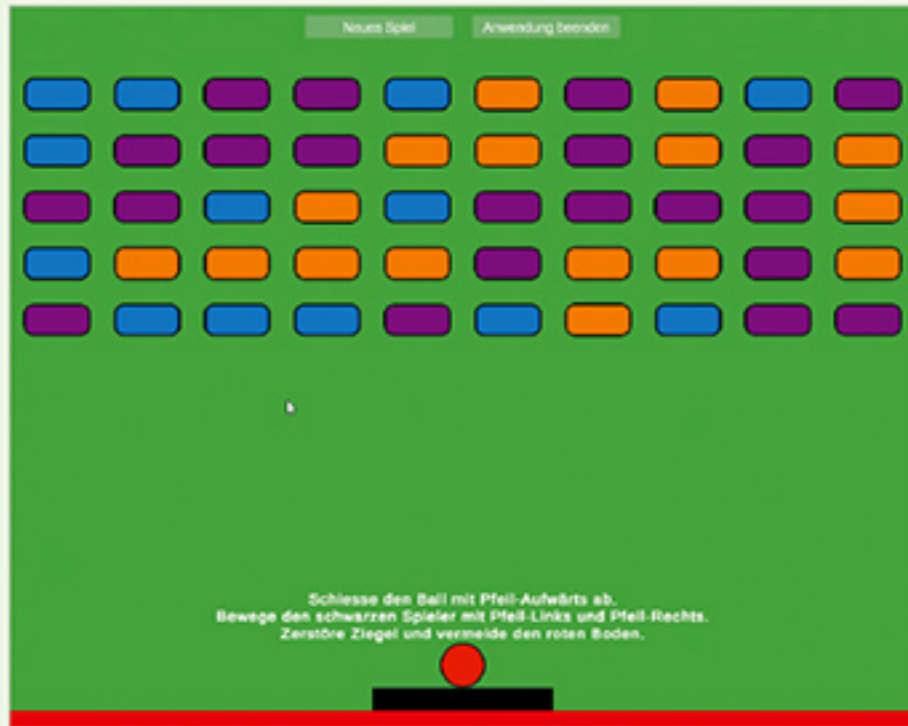
19.1 Das Unity-Projekt »Programmierkurs«	393
19.2 Grundlagen	394
19.2.1 Variablen und Datentypen	395
19.2.2 Rechenoperatoren	397
19.2.3 Division von ganzen Zahlen	399
19.2.4 Verzweigungen	400
19.2.5 Logische Verknüpfungen	402
19.2.6 Schleifen und Zufallszahlen	404
19.2.7 Schleifenabbruch mit »break«	407
19.3 Datenfelder	408
19.4 Zeichenketten	411
19.5 Methoden	413
19.5.1 Einfache Methode	413
19.5.2 Methode mit Parametern	414
19.5.3 Methode mit Rückgabewert	415
19.5.4 Methode mit Verweis-Parameter	416
19.6 Generische Listen	418
19.6.1 Hilfsmethode »AusgabeListe()«	420
19.6.2 »foreach«-Schleife	420
19.7 Daten auf der Festplatte	421
19.7.1 Daten speichern	422
19.7.2 Daten laden	423
19.7.3 Kontrolle der Daten	424

19.8	Objektorientierung	425
19.8.1	Die Spielobjekte im »Unity Editor«	425
19.8.2	Die Klasse »Spieler«	426
19.8.3	Änderungen aller Objekte der Klasse	428
19.8.4	Änderungen einzelner Objekte	429
20	Speichern Sie eine Highscore-Liste	431
20.1	Definition der eigenen Klasse	431
20.2	Nutzung der eigenen Klasse	433
20.2.1	Generische Liste erzeugen und füllen	434
20.2.2	Generische Liste anzeigen	435
20.2.3	Einen neuen Eintrag hinzufügen	436
20.2.4	Alles speichern, alles löschen	438
20.2.5	Der Anzeige-Schalter	439
21	Arbeiten Sie mit mehreren Szenen	443
21.1	Der Ablauf des Projekts	443
21.2	Der Aufbau der ersten Szene	444
21.3	Weitere Szenen	448
22	Allgemeine Hinweise	451
22.1	Projekte bearbeiten	451
22.1.1	Projekte umbenennen	451
22.1.2	Projekte kopieren	452
22.2	Unity Packages	453
22.2.1	Exportieren Sie ein Unity Package	453
22.2.2	Importieren Sie ein Unity Package	453
22.3	Unity unter anderen Betriebssystemen	455
22.3.1	Unity unter macOS installieren	455
22.3.2	Unity unter Ubuntu Linux installieren	456

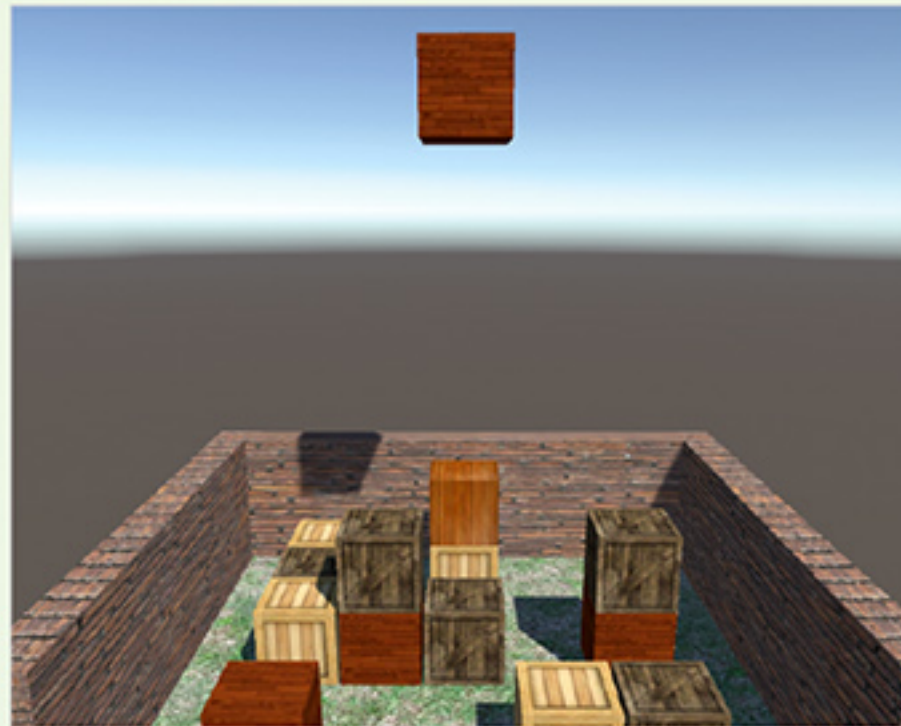
22.4	Browser-Anwendungen erstellen	456
22.5	Android-Apps erstellen	458
22.5.1	Änderungen im Code	458
22.5.2	Installieren Sie den Android Build Support	460
22.5.3	Stellen Sie die »Player Settings« ein	460
22.5.4	Führen Sie den Android-Build durch	462
22.5.5	Starten Sie die App unter Android	462
22.6	Bonusprojekte	463
22.6.1	Bonusprojekt »TomsFrogger«	463
22.6.2	Bonusprojekt »TomsPacman«	464
	Index	467

Spiele programmieren leicht gemacht

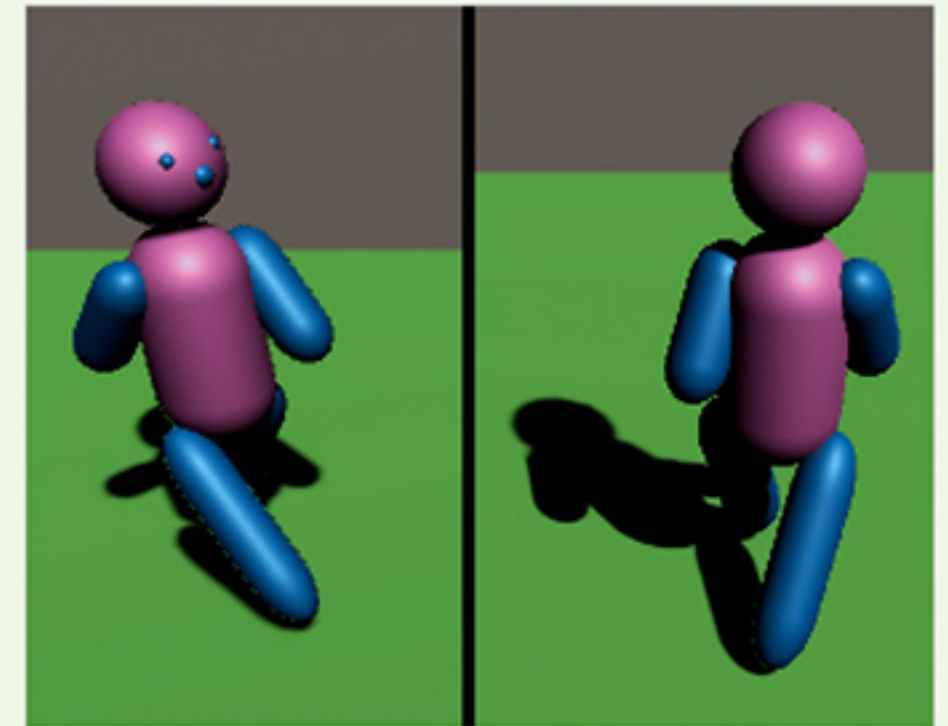
Nutzen Sie die geniale Game Engine, um eigene Spiele zu entwickeln! Unity bietet Ihnen mächtige Werkzeuge und eine Fülle von Effekten, die Sie nur einzubinden brauchen. Dieses Buch zeigt Ihnen, wie es geht – auch ohne Vorkenntnisse. Das Ziel: Computerspiele Marke Eigenbau.



Game-Klassiker in 2D



3D-Spiele mit Physik



Animationen erstellen

Programmierkurs für Anfänger

Das beste Spielerlebnis bieten Sie mithilfe von Unity-Skripten. Wie Sie diese Skripte in C# schreiben, erfahren Sie hier. Dazu gibt es einen leicht verständlichen Programmierkurs mit allen Grundlagen.

Nostalgie: Arcade-Klassiker nachbauen

Pong, Breakout, Autorennen: Bauen Sie die Retro-Games nach und lernen Sie dabei die Konzepte und Features von Unity kennen. Spiel für Spiel holen Sie immer mehr aus Unity heraus.

Übung macht den Meister

18 Games aus verschiedenen Genres laden Sie ein, kreativ zu werden: Probieren Sie Varianten aus, erweitern Sie die Spiele und entwerfen Sie eigene Ideen. So wird selbst das Üben zum Spiel.



Alle Projekte aus dem Buch als einsatzbereite Spiele zum Download



Thomas Theis ist Diplom-Ingenieur für Technische Informatik und arbeitet als Berater, Trainer und Dozent. Seit vielen Jahren gibt er Kurse für Einsteiger in verschiedene Programmiersprachen und -Plattformen, darunter JavaScript, C und C#.

Auf einen Blick

Grundlagen

- C#-Einstieg
- Unity installieren
- Projekte, Scenes und Assets verwalten
- Objekteigenschaften festlegen
- Collider und Trigger
- Physik simulieren
- Terrains generieren

Kult-Games

- 2D-Jump & Run
- Pong, Breakout und Co.
- 3D-Autorennen
- Space Shooter
- 3D-Adventure
- Geschicklichkeitsspiele
- 3D-Labyrinth

€ 29,90 [D] € 30,80 [A]



Gedruckt in Deutschland
Papier aus nachhaltiger Waldwirtschaft
Mineralölfreie Druckfarben

