Max Schlosser

Spieleentwicklung mit Unity

Das umfassende Handbuch

- Grundlagen, Tutorials, Best Practices
- Physik, User Interfaces, Animationen, Components, Audio
- Viele Beispielprojekte, C#-Kurs und Übungen mit Lösungen

Alle Beispielprojekte zum Download

Rheinwerk Computing

Kapitel 1
Einführung

Es geht los! Am Anfang geht es darum, welche Grundlagen du benötigst, wie dieses Buch dir helfen möchte und wie du beim Lesen am besten lernen kannst.

Das Entwickeln von Spielen ist eines der großartigsten Dinge, die du dir vornehmen kannst. Kaum ein anderes Hobby macht es möglich, deine Kreativität praktisch grenzenlos auszuleben und den Ideen, die du schon immer in deinem Kopf hattest, ganz allein Leben einzuhauchen. Das Gefühl, der eigenen Familie oder deinem Freundeskreis ein Spiel zu zeigen, das es in dieser Form noch nicht gibt und das du selbst gebaut hast, ist einzigartig.

Gleichzeitig fordert die Spieleentwicklung dein Gehirn: Um das Herz eines Spiels zu realisieren, muss man dem Computer beibringen, wie das Spiel funktioniert. Dafür muss die Spiellogik programmiert werden – ein spannender Prozess, bei dem du nicht darum herumkommst, deine grauen Zellen anzustrengen. Zuallererst also ein großes Kompliment, dass du dich für das fesselnde, vielseitige und große Thema der Spieleentwicklung begeisterst.

1.1 Das Problem vieler Einsteiger

Wenn du dich dem großen Gebiet der Spieleentwicklung annäherst, wirst du von den vielen Dingen, die zu beachten sind, geradezu erschlagen: ein Programm mit vielen Fenstern, dazu kryptisch aussehender Programmquellcode und unverständliche Fehlermeldungen. Das ist für viele Einsteiger, die sich zum ersten Mal mit der Materie beschäftigen, schon nach wenigen Minuten demotivierend.

Viele Bücher, Videos oder Blogs versuchen die Dinge deshalb anders anzugehen. Anstatt die Grundlagen geordnet zu vermitteln, werden »einfach so« Spiele entwickelt. Es wird dabei Schritt für Schritt gezeigt, wie du beim Lesen zu einem ganz bestimmten Ergebnis kommst – zum Beispiel indem ganz bestimmte Beispiel-Games entwickelt werden. Dabei werden theoretische Konzepte verkürzt erklärt, und zwar nur dann, wenn sie für das aktuelle Beispiel benötigt werden. Es ist häufig nicht schwer, diese Schritte nachzuvollziehen und das gezeigte Spiel somit 1 : 1 nachzubauen, indem du das Gezeigte einfach »abschreibst«.

Trotzdem ergibt sich aus diesem Vorgehen mehr als nur ein entscheidendes Problem. Nachdem du verschiedene Spiele nach einer Anleitung nachentwickelt hast, möchtest du natürlich deine eigenen Ideen umsetzen. Da du davor aber nur anhand ganz konkreter Beispiele gelernt hast, stößt du auf verschiedene Schwierigkeiten:

- Du kennst bestimmte Techniken nicht, die du für dein Spiel brauchst. Das liegt daran, dass diese Techniken in den gezeigten Beispielen nicht benötigt wurden und du ihnen so noch nie begegnet bist.
- Du hast nicht gelernt, wie du selbstständig an Probleme herangehst, da du bei den gezeigten Spielen in jedem Schritt an die Hand genommen wurdest. Es fällt dir gleichzeitig schwer, mit anderen Quellen nach Lösungsansätzen zu suchen.
- Du unterschätzt die für die Umsetzung eines Spiels notwendige Planung, da in den gezeigten Beispielen »für dich« geplant wurde.

1.2 Wichtige Ansätze des Buches

Das Buch setzt sich als oberstes Ziel, zu verhindern, dass du auf die im letzten Abschnitt angesprochenen Schwierigkeiten stößt. Dafür setze ich verschiedene Techniken ein.

1.2.1 Die Mischung macht's – Theorie und Praxis

Du weißt nun, dass ein Buch, das die grundlegende Theorie nur hin und wieder in Praxisbeispiele einpflegt, auf lange Sicht nicht beim Lernen hilft. Genauso ist es mit einem Buch, das lediglich aus trockener Theorie besteht, die nie an praktischen Beispielen angewendet wird – du würdest beim Lesen einschlafen. Aus diesem Grund gehe ich einen ausgewogenen Mittelweg zwischen Theorie und Praxis, der dich optimal vorbereiten soll, um im Anschluss oder schon während des Lesens deine eigenen Spielideen umzusetzen. Ich will nicht, dass du *meine* Spiele entwickelst, sondern *deine*!

Theoretische Erklärungen halte ich möglichst allgemein, damit du sie später auf viele eigene Ideen anwenden kannst – dieses Buch ist also auf eine Art ein nützliches Handbuch. Dadurch ist es möglich, das Buch als »Nachschlagewerk« für bestimmte Themen zu nutzen. Darüber hinaus pflege ich in den theoretischen Einheiten viele praktische Beispiele ein. Diese zeigen dir, wie man die Theorie in der Praxis nutzen kann, um verschiedene Probleme zu lösen. Die im folgenden Abschnitt 1.2.2 erläuterten Elemente zum eigenen praktischen Arbeiten mit Übungen verfestigen die Theorie.

1.2.2 Übung macht den Meister

Um das Gelernte auf die Probe zu stellen, stelle ich in allen Kapiteln eigene Übungen bereit. Diese Übungen greifen die Theorie an konkreten Aufgaben auf, die das zuvor erworbene Wissen auf die Probe stellen. Auch wenn du die Übungen selbstständig lösen sollst, werden für alle Aufgaben Musterlösungen bereitgestellt. An diesen siehst du, wie du das Wissen aus den Kapiteln verwenden kannst, um die Übung zu meistern. Musterlösungen findest du immer am Ende eines Kapitels (so ist der Drang, direkt zur Lösung zu greifen, hoffentlich geringer).

Eine besondere Bedeutung haben die sogenannten Übungskomplex-Kapitel. Diese beinhalten keine neuen theoretischen Inhalte, sondern nur Aufgaben, die das Wissen der letzten Kapitel aufgreifen und kombinieren. Teilweise umfassen die Übungen dabei sogar ganze »Minispiele«, die du selbstständig umsetzen sollst. Für die Aufgaben aus den Übungskomplexen stehen ebenfalls vollständige Lösungen mitsamt einer Erklärung bereit.

1.2.3 Community

Vielleicht hast du schon in der Schul- oder Studienzeit gemerkt, dass gemeinsam zu lernen nicht nur mehr Spaß macht, sondern deutlich produktiver sein kann (wenn man sich nicht gegenseitig ablenkt). Aus diesem Grund bietet das Buch eine Möglichkeit, mit der du dich schnell mit anderen Unity-Einsteigern vernetzen kannst.

Das hat mehrere Vorteile:

- Du kannst eigene Lösungen teilen. Damit zeigst du nicht nur, dass du dir selbst Gedanken für einen Lösungsweg gemacht hast, sondern gibst anderen die Chance, dir weiteres Feedback zu geben.
- Du kannst dir andere Lösungen ansehen und diese mit deinen eigenen Ansätzen vergleichen. Durch solche Perspektiven kannst du viele Probleme deutlich besser verstehen, als wenn du allein lernst.

Ich verwende für das Buch die Plattform Discord, die verschiedene Möglichkeiten bietet, um sich per Text- und Sprachchat auszutauschen. Eine ausführliche Anleitung zur Einrichtung der Anwendung sowie zur Nutzung der Kanäle findest du auf der Webseite des Buches unter *www.rheinwerk-verlag.de/5711*. Dort ist ein Link hinterlegt, über den du dem Discord-Server mit nur einem Mausklick beitreten kannst. Falls du Discord noch nie genutzt hast: Das Erstellen eines neuen Discord-Accounts geht schnell und ist vollkommen kostenlos.

Falls du Discord bereits nutzt, kannst du über den folgenden Link direkt beitreten: *https://r-wrk.de/9539discord*

1.3 Lernen mit dem Buch

Nachdem klar ist, welchen Problemen das Buch wie begegnet, geht es nun um das eigentliche Lesen. Was sind also Voraussetzungen zum Lesen, wie erreichst du den besten Lerneffekt und was kannst du, wenn du dich durch das komplette Buch gekämpft hast?

1.3.1 Voraussetzungen für das Buch

»Was muss ich denn schon können, um hier mitzukommen?« Diese Frage hast du dir sicher schon einmal gestellt, bevor du eine Webseite angeklickt, ein Buch aufgeschlagen oder ein Video abgespielt hast. Deswegen soll diese wichtige Frage hier in aller Kürze beantwortet werden: Du brauchst für deine Reise in die Entwicklung von Games absolut keine Vorkenntnisse. Um dieses Buch zu lesen, musst du noch nie irgendeine Umgebung für die Spieleentwicklung geöffnet oder auch nur eine Zeile Quellcode geschrieben haben. Wenn du bereits Erfahrung in der Spieleentwicklung sammeln konntest oder eine Programmiersprache (grundlegend) beherrschst, wird dir das allerdings beim Verstehen vieler Themen helfen.

»Harte« Voraussetzung für das Entwickeln von Spielen ist selbstverständlich ein lauffähiger Computer. Für das Herunterladen und Einrichten des Programms Unity und den Download der Dateien, die du für die Beispiele benötigst, ist zudem einmalig eine stabile Internetverbindung notwendig. Danach kannst du auch offline an deinen Spielen arbeiten, allerdings könnte dir das Internet oft beim Recherchieren helfen. Eine Art Nebenvoraussetzung ist die Sprache Englisch, um die du während des Lesens nicht herumkommen wirst, wie Abschnitt 1.3.3 genauer erläutert.

Die wichtigste Voraussetzung für das Lernen ist jedoch deine Motivation. Dein Wille, eigene Spiele zu bauen, kommt dir besonders bei den Übungen zugute, da du mit jedem kleinen Minispiel, das du baust, deinem Ziel näher rückst, eigene Spiele zu kreieren. Sieh dieses Buch dabei insgesamt als eine große Tür, die ich dir öffne. Selbst wenn der Eingang weit offen steht, musst du den Schritt hindurch selbst gehen. Dafür gibt es einige Tipps, mit denen du dein Lernergebnis optimieren kannst.

1.3.2 Wie du am besten mit dem Buch lernst

Du hast mit Sicherheit deinen eigenen, ganz persönlichen Lernstil. Dennoch gibt es ein paar grundlegende Tipps, die dir das Lernen mit dem Buch insgesamt erleichtern sollen. Das sind nur drei allgemeine Grundregeln:

1. **Nimm dir Zeit für die Theorie**. Viele theoretische Themen sind umfangreich. Bevor du versuchst, die Theorie in der Praxis anzuwenden, solltest du die Kapitel des-

halb aufmerksam lesen. Nur so erhältst du den notwendigen Gesamtüberblick, der dir hilft, verschiedene Themen zu verstehen und im Anschluss gezielt einzusetzen.

2. **Versuche Übungen selbstständig zu lösen.** Auch wenn für die Aufgaben Musterlösungen bereitstehen, wirst du vom Abschreiben nicht viel lernen. Versuche dir die Lösung deshalb selbst herzuleiten und Probleme, auf die du stößt, auf eigene Faust zu lösen. Dabei kannst du ebenfalls andere Quellen einbeziehen, um den Lösungsweg besser zu verstehen. Es ist nicht schlimm, wenn du beim Bearbeiten der Übungen noch einmal zurückblätterst oder im Internet recherchierst. Im Gegenteil: So lernst du nach und nach, wie du Probleme selbstständig löst – schließlich kannst du, wenn du eigenständig ein Spiel entwickelst, auch jederzeit Lösungen recherchieren.
3. **Wiederhole Stoff bei Wissenslücken.** Besonders beim Üben ist es völlig normal, dass theoretisch gelernte Dinge nicht sofort abrufbereit sind. An dieser Stelle solltest du nicht einfach zur Lösung gehen, sondern die Grundlagen wiederholen, indem du ein paar Seiten zurückblätterst. Dich sollte niemand beim Lernen hetzen (nein, auch nicht du selbst!), also nimm dir Zeit. Das Gute an einem Buch ist ja, dass es nicht weglaufen kann.

Grundsätzlich ist das Buch so konzipiert, dass du es von vorn nach hinten durchlesen kannst. Wenn du der Meinung bist, ein bestimmtes Thema bereits sehr gut zu beherrschen, kannst du einzelne Kapitel oder Abschnitte einfach überspringen. Gleichzeitig bauen die Kapitel aufeinander auf. Wird für einen Themenkomplex Wissen aus einem vorangegangenen Kapitel benötigt, wird es deshalb nicht wiederholt. Bevor du ein Kapitel überspringst, solltest du dir also sicher sein, dass du alle im Kapitel untergebrachten Themen gut beherrschst. Es kann dabei auch helfen, die Übungsaufgaben als Selbsttest durchzuführen.

1.3.3 Englisch als Sprache für das Buch

Im Buch wirst du immer wieder über englische Begriffe stolpern. Das fängt schon bei der Software Unity an, in der alle Optionen standardmäßig in englischer Sprache angezeigt werden. Beim Programmieren ist Englisch ebenfalls nicht mehr wegzudenken. Im Quellcode, der für die Programmierung geschrieben wird, sind viele Begriffe vorgegeben – und natürlich sind diese ebenfalls in englischer Sprache. Und als wäre das nicht genug, werden im Bereich der Softwareentwicklung nur die wenigsten technischen Begriffe streng ins Deutsche übersetzt.

Es wäre also gelogen, zu behaupten, dass Englisch für das Buch keine Rolle spielt. Das heißt selbstverständlich nicht, dass du Englisch studiert haben musst, immerhin kannst du englische Optionen anklicken, indem du die einzelnen Buchstaben vergleichst. Wenn du dich aber langfristig auf die Welt der Entwicklung von Software oder Spielen einstellen willst, solltest du Englisch als deinen stetigen Begleiter aner-

kennen. Aber versprochen: Während des Lesens wird dir niemand vorschreiben, in welchem Umfang du die englische Sprache verwendest. Wenn du dich der Sprache erst annäherst, gibt es viele verschiedene Tipps, die du befolgen kannst, um sie während des Lesens noch besser zu lernen. Am wichtigsten sind dabei Vokabeln, die im technischen Bereich häufig genutzt werden. Zum Üben kannst du zum Beispiel englische Begriffe, die du noch nicht kennst, in einem Wörterbuch recherchieren, auch online. So lernst du neue Wörter, die du selbst einsetzen kannst, zum Beispiel beim Schreiben von Quellcode.

So oder so: Die englische Sprache wird dir keinen Strich durch die Rechnung machen. Wenn du Lust am Lernen hast, wirst du selbst diese Hürde mit Leichtigkeit überspringen.

1.3.4 Was kannst du nach dem Lesen?

Die wichtigste Frage zum Schluss: Was kannst du, wenn du das Buch vollständig durchgelesen hast? Wenn du nur stumpf Seite für Seite liest, lautet die ernüchternde Antwort wohl: Nicht wirklich viel. Die Hintergründe wurden bereits in Abschnitt 1.3.2, »Wie du am besten mit dem Buch lernst«, erklärt. Hier wird es also wichtig, dass du selbstständig lernst, Übungen löst, Lücken wiederholst und dir später eigene Ziele setzt. Im Optimalfall hast du nach dem aufmerksamen Lesen der theoretischen Grundlagen und dem selbstständigen Durcharbeiten der Übungen mehrere, wirklich nützliche Skills erlangt:

- Du kannst eigene Spielideen konzeptionieren und anschließend selbstständig in Unity umsetzen. Dabei kannst du abschätzen, aus welchen Komponenten dein Spiel bestehen wird und wie aufwendig diese in ihrer Umsetzung sind.
- Du hast einen Überblick über zahlreiche wichtige Funktionen von Unity erlangt und weißt, wo und wie sie dir das Leben erleichtern.
- Du besitzt ein generelles Verständnis für allgemeine Programmierkonzepte und fortgeschrittene Kenntnisse in der Programmiersprache C#, die für das Entwickeln von Spielen mit Unity verwendet wird.

[i]

Ein Spiel programmieren oder hübsch machen?

Dieses Buch nimmt dich in erster Linie dabei an die Hand, die Logik deiner Spiele mit Unity umzusetzen. Das Aussehen steht somit in vielen Beispielen nicht im Vordergrund, immerhin kann man ein Spiel auch ohne ein wunderschönes Design spielen. Das hindert dich allerdings nicht daran, dich weiter mit der »Verschönerung« auseinanderzusetzen – schließlich ist es wichtig, dass dein Game nicht nur gut funktioniert, sondern auch ansehnlich ist. Die Grundlagen, die Unity dafür bietet, lernst du in diesem Buch kennen. Wie ein Spiel am Ende hübsch wird, ist eine eigene Wissenschaft für sich – und die erfordert sehr viel Übung!

1.4 Vorstellung der Kapitel

Damit du einen groben Überblick über das Buch bekommst, stelle ich dir im Schnellüberblick alle Kapitel kurz vor:

- In **Kapitel 2, »Die Unity-Engine«,** wird das für das Buch verwendete Programm Unity kurz vorgestellt und beantwortet, warum gerade diese Software eine gute Wahl für die Spieleentwicklung ist. Auch wirst du Unity installieren und dein erstes Projekt einrichten.
- **Kapitel 3, »Grundlegende Konzepte in der Engine«,** erläutert die wichtigsten grundlegenden Konzepte, die für das allgemeine Verständnis von Unity notwendig sind. Du lernst, wie du dich in deiner Spieleumgebung bewegst und einfache Objekte in deiner virtuellen Welt anordnest.
- In **Kapitel 4, »Das erste Script«,** geht es zum ersten Mal an die Programmierung. Du wirst Schritt für Schritt lernen, wie ein neues Script aufgebaut ist. Gleichzeitig wirst du zum ersten Mal eigenen Quellcode schreiben, um Ausgaben in Unity anzuzeigen.
- **Kapitel 5, »Grundlegende Konzepte der Sprache C#«,** thematisiert, aufbauend auf dem vorangegangenen Kapitel, weitere Konzepte der Programmiersprache C#. Dieses Kapitel ist für die Programmierung insgesamt wichtig, da viele der vorgestellten Konzepte in allen Programmiersprachen auftauchen – du lernst also fürs Leben!
- **Kapitel 6, »Scripting in Unity«,** schlägt eine Brücke zwischen der Programmierung in C# und der Einbindung von Scripts in Unity. Hier wird es richtig interessant, denn du lernst, welche grundlegenden Funktionen unterstützt werden und wie verschiedene Scripts in deinem Spiel miteinander »kommunizieren« können. Außerdem setzt du mit dem Wissen eine einfache Steuerung um.
- **Kapitel 7, »Übungskomplex 1«,** ist der erste Übungskomplex des Buches. Hier geht es beim Großteil der Übungen um den Einstieg in die Programmierung, auch im Zusammenhang mit Unity. Mit den in den letzten Kapiteln erlangten Kenntnissen werden erste komplexere Aufgaben gelöst, die dein Wissen in der Programmierung und im Umgang mit Unity fordern.
- **Kapitel 8, »Physik«,** erklärt die Physik in Unity. Mithilfe einfacher physikalischer Simulationen siehst du, wie einfach die Nutzung der verschiedenen von Unity vorgegebenen Komponenten ist. Auch wirst du lernen, auf welche Weise Scripts mit der Spielphysik interagieren können.
- In **Kapitel 9, »Fortgeschrittene Scripting-Themen«,** werden komplexe Themen der Programmierung im Rahmen mit Unity vorgestellt. Egal, ob es um die Nutzung von Zeitintervallen oder das Speichern einfacher Spieldaten geht: Viele dieser Techniken werden für viele deiner Spiele nützlich sein.

- **Kapitel 10, »Prefabs«,** dreht sich um eine Einheit in Unity, die die Wiederverwendung bestimmter Spielelemente erheblich erleichtert. Anhand verschiedener Beispiele wird der Mehrwert solcher Prefabs deutlich gemacht.
- **Kapitel 11, »Übungskomplex 2«,** beinhaltet den zweiten Übungskomplex und damit zahlreiche Übungen, in denen kleine Simulationen und sogar Minispiele entwickelt werden sollen. Dabei sind nahezu alle Konzepte der vorangegangenen Kapitel zur effizienten Lösung der vorgegebenen Probleme relevant.
- **Kapitel 12, »Objektorientierte Programmierung«,** dreht sich um den zentralen Begriff der Objektorientierung innerhalb der Programmierung. Nach einer Vorstellung der grundlegenden Ideen dieses Paradigmas werden zahlreiche Konzepte, die die Sprache C# in diesem Rahmen unterstützt, genau erläutert.
- In **Kapitel 13, »2D-Inhalte«,** werden bestimmte 2D-Elemente von Unity hervorgehoben. Für das Bauen von Spielen im zweidimensionalen Raum können diese viele lästige Arbeiten abnehmen.
- **Kapitel 14, »User Interfaces«,** erläutert die zahlreichen Unity-Elemente zum Erstellen von Benutzeroberflächen. Grafiken, Texte, Buttons oder ganze Layouts können mit vielen vorgefertigten Elementen realisiert werden.
- **Kapitel 15, »Visualisierung«,** schneidet mehrere Unity-Systeme an, die für eine Visualisierung der Spielumgebung genutzt werden können. Dazu zählen Partikelsysteme, von Unity bereitgestellte Beleuchtungskomponenten und Terrain-Werkzeuge, mit denen komplexe 3D-Landschaften gebaut werden können. Du lernst auch, wie du dein Projekt zu einem fertigen Game umwandelst, das von allen gespielt werden kann.
- **Kapitel 16, »Animation«,** stellt vor, wie Animationen in Unity erstellt oder importiert und anschließend dynamisch in das Spielgeschehen eingebunden werden. Es wird dabei zwischen 2D- und 3D-Animationen unterschieden. Du lernst außerdem, wie du Animationen dynamisch wechselst und mit deinem Spielgeschehen verbindest.
- **Kapitel 17, »Sound«,** zeigt, auf welchen Wegen Unity das Einbinden von Musik und Soundeffekten erlaubt.
- **Kapitel 18, »Navigation«,** stellt das Navigationssystem der Engine vor. Mit diesem können automatisierte Wegfindungen im Spiel eingebaut werden, die zum Beispiel Gegner wie von Zauberhand um Hindernisse zum Spieler steuern lassen.
- In **Kapitel 19, »Fortgeschrittene Konzepte der Sprache C#«,** geht es um die fortgeschrittensten C#-Techniken, mit denen komplexe Ideen im Quellcode umgesetzt werden können.
- **Kapitel 20, »Übungskomplex 3«,** umfasst Übungen, die solide Kenntnisse in allen wichtigen Bereichen von Unity voraussetzen. Du setzt kleinere Szenarien und sogar zwei vollwertige Spielideen um.

- Mit **Kapitel 21** schließt sich der letzte **»Übungskomplex 4«** an. Dieser ist anders aufgebaut als die anderen Übungskomplexe, da die vollen Lösungen nur online bereitgestellt werden. Für die Übungen – vier komplette Spiele – werden im Buch lediglich verschiedene Hinweise zur Problemlösung gegeben.
- **Kapitel 22, »Ausblick«,** bildet den Abschluss des Buches. In einem Nachwort sowie einem Ausblick wird reflektiert, was du durch das Lesen gelernt hast. Außerdem wird beantwortet, auf welchen Wegen du das Wissen bestmöglich einsetzen oder erweitern kannst.

Wie du siehst, liegt eine ganze Menge an Themen vor dir. Also nichts wie rein ins Geschehen!

Kapitel 3
Grundlegende Konzepte in der Engine

Es wird Zeit, Unity richtig kennenzulernen! Mit Sicherheit bist du aufgeregter als vor deinem ersten Schultag.

Nachdem alle Programme installiert sind, kann es so richtig losgehen: In diesem Kapitel lernst du die wichtigsten Grundkonzepte kennen, die du im Umgang mit dem Programm und deiner virtuellen Welt benötigst. Nachdem du Unity zum ersten Mal geöffnet hast, mag dich der Anblick des Programms überrumpeln. Wer kann schon auf den ersten Blick verstehen, wofür die ganzen Fenster da sind und was Unity überhaupt für uns generiert hat? Damit du dir diese Fragen nicht selbst beantworten musst, stelle ich dir zunächst die wichtigsten Grundelemente und Fenster vor, die du in einem neuen Projekt findest.

3.1 Der erste Überblick über Unity

Werfen wir also zunächst einen Blick auf den Unity Editor, der sich dir zeigt, nachdem du ein neu erstelltes Projekt geöffnet hast (siehe Abbildung 3.1).

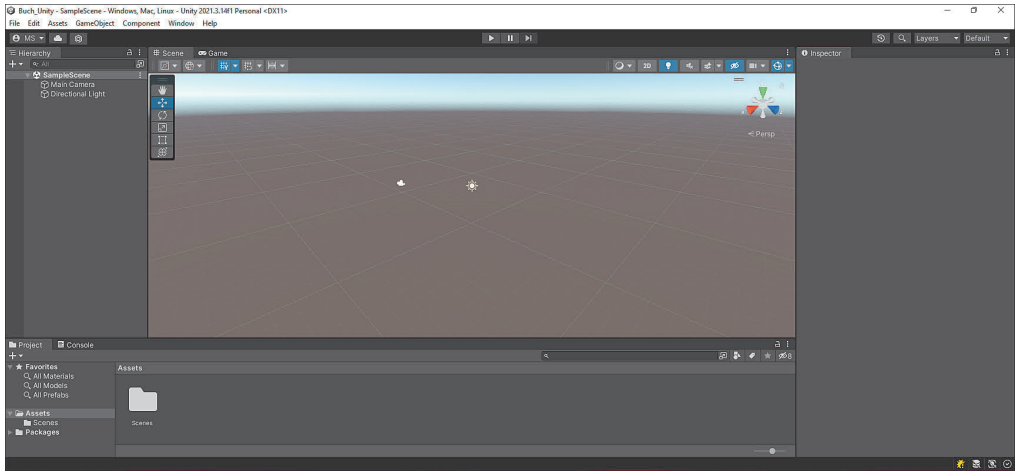


Abbildung 3.1 Die Ansicht der Unity Engine nach dem Öffnen des neu erstellten Projekts

Wenn die Engine beim Öffnen anders aussieht

Generell können alle Fenster innerhalb von Unity frei angeordnet werden. Darüber hinaus gibt es sogenannte *Layouts*. Layouts sind eine gespeicherte Anordnung von Fenstern, die in Unity zu sehen sind. Wenn Unity anders aussieht, dann hast du wahrscheinlich die Anordnung der Fenster geändert. Das ist aber kein Problem: Es gibt eine einfache Möglichkeit, um das Standardlayout wiederherzustellen. Klicke dafür in der oberen Menüleiste auf WINDOW • LAYOUTS • DEFAULT (siehe Abbildung 3.2).

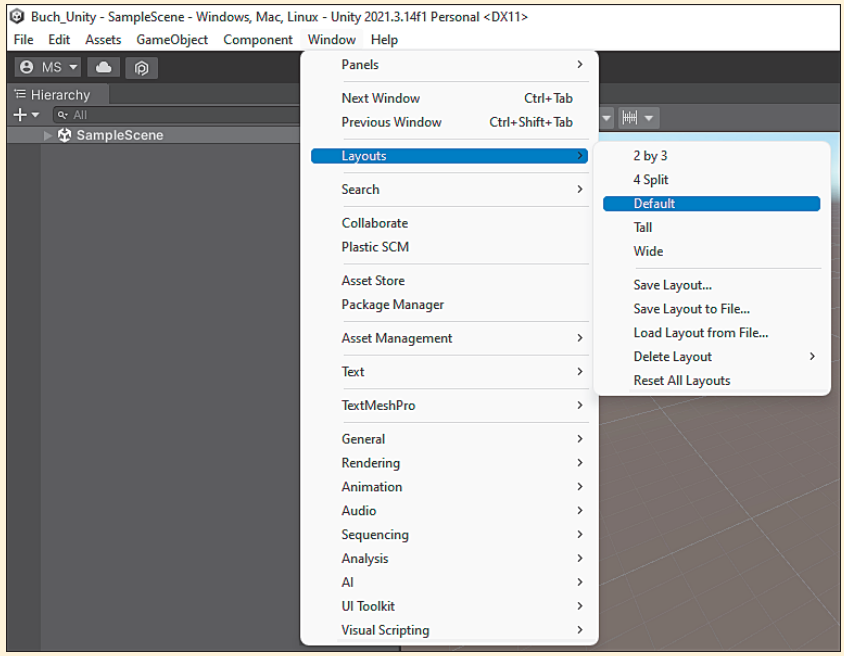


Abbildung 3.2 Zurücksetzen des Editor-Layouts auf den Standard

3.1.1 Scene

Schon auf den ersten Blick ist zu sehen, dass in der Mitte des Fensters eine große, virtuelle Welt für dich erstellt wurde (siehe Abbildung 3.1). Da du genau darin deine Spielideen verwirklichen wirst, werfen wir zunächst einen genaueren Blick auf diese Umgebung. Der Begriff, der dir dafür nie wieder aus dem Kopf gehen sollte, ist der einer *Scene*. Eine Scene ist eine Einheit, in der du eine Spielwelt in Unity bauen kannst. Ein Spiel kann dabei aus vielen Scenes bestehen. Das ermöglicht es, verschiedene Einheiten deines Spiels in jeweils eigenen Scenes abzulegen.

Stell dir ein Spiel vor, in dem du einen Charakter mit einer Steuerung durch unterschiedliche Level springen lässt. Mit Unity könntest du diese Idee sehr gut in mehrere

Scenes unterteilen. Die erste Scene könnte so zum Beispiel das Hauptmenü sein, mit dem das eigentliche Spiel über verschiedene Buttons gestartet oder beendet werden kann. Außerdem könnte jedes einzelne Level in einer eigenen Scene abgelegt werden. So bleibt die Aufteilung in Unity schön übersichtlich: Wenn du ein bestimmtes Level ändern möchtest, kannst du dich einfach an der jeweiligen Scene austoben.

Unity schreibt dir nicht vor, wann du eine neue Scene erstellen musst. So wäre es theoretisch möglich, dein gesamtes Spiel in nur einer einzigen Scene abzulegen. Du würdest dabei schnell merken, dass du die Übersicht verlierst, besonders dann, wenn du eine größere Spielidee hast. Die Unterteilung deines Spiels in verschiedene Scenes wirst du somit fast automatisch lernen und anwenden, es sei denn, du liebst das pure Chaos. Im Laufe des Buches wirst du noch viel mehr über Scenes lernen.

Scenes in den Downloadmaterialien

Für das Downloadprojekt wird ebenfalls mit einzelnen Scenes gearbeitet. So liegt beispielsweise jede Musterlösung für eine Übung in genau einer Scene.

Für einfache Spiele wirst du mit nur einer Scene gleichzeitig arbeiten. Wenn du eine andere Scene zeigen willst, zum Beispiel weil ein neues Level geladen werden soll, dann wird die zuvor geöffnete Scene verworfen und anschließend die neue Scene gezeigt. Ähnlich ist es bei der Bearbeitung einer Scene im Unity Editor. Scenes werden in eigenen Dateien abgelegt und getrennt voneinander bearbeitet. Wenn du also an einer anderen Scene arbeiten möchtest, öffnest du einfach die entsprechende Datei. Das mag erst einmal umständlich klingen, aber du wirst sehen, dass dir diese Unterteilung erheblich helfen wird, den Überblick über dein Spiel zu behalten.

3.1.2 Überblick über die wichtigsten Fenster

Da du nun weißt, dass wir es bereits mit einer geöffneten Scene zu tun haben, werfen wir nun einen Blick auf die verschiedenen Fenster, die im Standardlayout von Unity, das in Abbildung 3.3 gezeigt wird, zu sehen sind.

Du wirst die virtuelle Welt deiner Scene Stück für Stück aufbauen – egal, ob es um große Landschaften, einzelne Häuser oder das Platzieren von Gegnern geht. Es ist also nur wenig verwunderlich, dass Unity ein Fenster bereitstellt, das dir das Bauen einer Scene über viele nützliche Werkzeuge erleichtert: die sogenannte *Scene View* ❶. Da die Benutzung und die einzelnen Werkzeuge in Abschnitt 3.3, »Orientierung in der Scene View«, genauer erklärt werden, reicht es an dieser Stelle zu wissen, dass dies der Ort ist, an dem du die Spielwelt deiner Scene zusammensetzt – kein Wunder also, dass das Fenster im Standardlayout einen so großen Teil des Editors einnimmt.

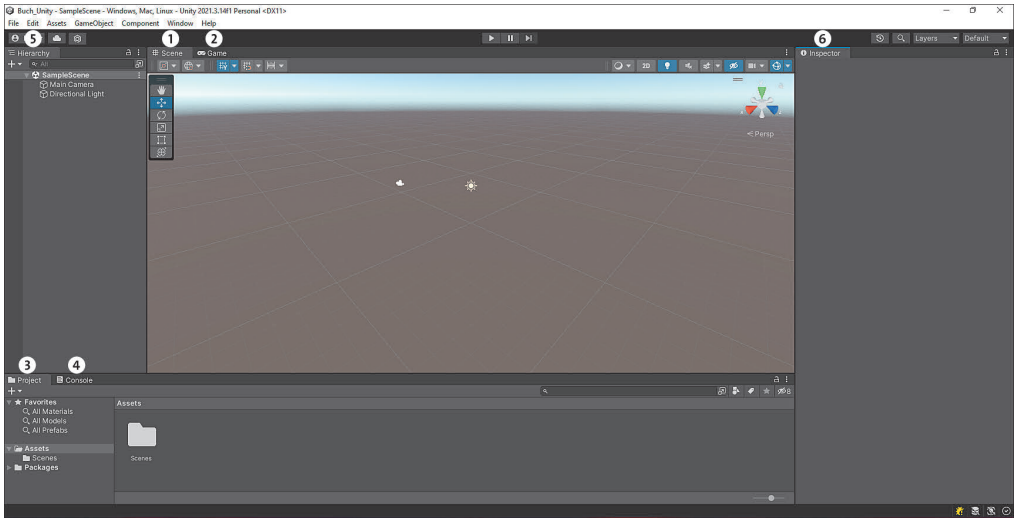


Abbildung 3.3 Übersicht über die Fenster des Standardlayouts

Ein weiteres Unterfenster in der Scene View

Wenn deine Scene View, wie in Abbildung 3.4 gezeigt, ein weiteres kleines Fenster beinhaltet, kannst du dieses einfach ausblenden. Dafür reicht ein Rechtsklick auf das Fenster und ein Klick auf COLLAPSE.

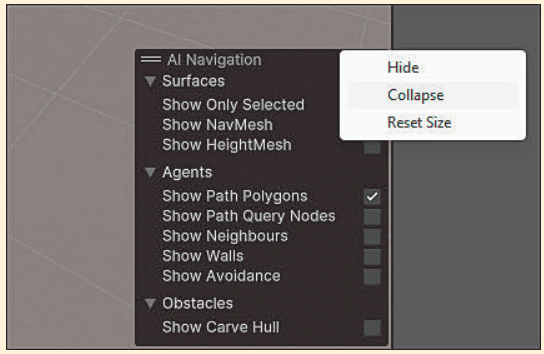


Abbildung 3.4 Ausblenden des Unterfensters in der Scene View

Das Fenster der *Game View* ❷ ist zunächst nicht zu sehen, wenn das Standardlayout geladen wird. Grund dafür ist, dass dieses Fenster unter dem der Scene View liegt. Durch einen einfachen Klick auf den Tab GAME wird die Game View geöffnet.

Die Game View stellt das Bild dar, das man sehen würde, wenn man das Spiel startet (siehe Abbildung 3.5). Im Moment sieht diese Ansicht nach einem leeren Himmel aus, da du in deiner Scene noch keine Änderungen vorgenommen hast. Über die Game View kannst du dein Spiel außerdem unkompliziert testen. Es wäre schließlich ner-

vig, wenn du dein Spiel jedes Mal neu bauen müsstest, um zu sehen, wie sich kleine Korrekturen auswirken.

Besonders praktisch ist dabei, dass du auch Anpassungen, die du während des Spielens vornimmst, live testen kannst. So kannst du schnell mit Werten in deinem Spiel experimentieren. Wie fühlt sich die Charaktersteuerung an, wenn die Sprunghöhe nur halb so hoch ist? Ist der Gegner zu stark, wenn er mit doppelter Geschwindigkeit auf den Spieler zustürmt? Die Game View wird dir die Antwort geben!

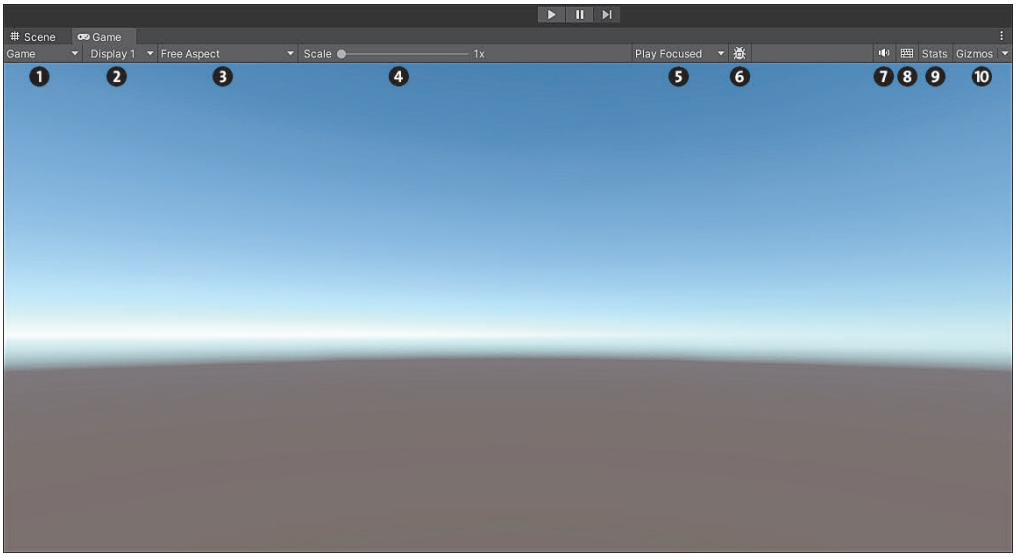


Abbildung 3.5 Ansicht der Game View im neu erstellten Projekt

Die Game View bietet mehrere Werkzeuge, die das Testen des Spiels vereinfachen. Sehen wir uns diese anhand von Abbildung 3.5 einmal genauer an:

❶ **Simulationsmenü**: Über dieses Menü kann gewählt werden, ob die Game View als Testfenster in der Engine oder über einen Simulator geöffnet wird. Wird die Einstellung SIMULATOR gewählt, kann das Spiel auf simulierten mobilen Endgeräten – also Handys und Tablets – getestet werden. Das ist praktisch, wenn du schnell ausprobieren möchtest, wie dein Spiel auf verschiedenen Displaygrößen aussieht. Auch stehen in der Simulator-Ansicht neue Einstellungen bereit, die du im Rahmen des Buches aber nicht benötigen wirst. Aus diesem Grund schauen wir uns diesen Anzeigemodus nicht näher an.

❷ **Display-Menü**: Hier wählst du, über welche Anzeigequelle die Game View angezeigt wird.

❸ **Aspect-Menü**: In diesem Menü kannst du eine spezifische Auflösung oder ein festes Seitenverhältnis für die Spielansicht festgelegen. Ist diese Einstellung auf FREE ASPECT gesetzt, wird die Auflösung an der Größe des Fensters deiner Game View

festgemacht. Wenn du eine andere Größe wählst, ändert sich die Spielansicht im Fenster so, dass das gewählte Seitenverhältnis zustande kommt.

❹ **Scale-Slider**: Erlaubt das Hereinzoomen in das Spielgeschehen. Dadurch, dass lediglich in das Fenster hineingezoomt wird, wirkt das Bild oft pixelig, so, als ob du in eine Bilddatei hineinzoomen würdest. Um die Spielwelt aus der Nähe zu betrachten, solltest du daher eher die Scene View nutzen.

❺ **Fokusmenü**: Hier kannst du einstellen, ob die Game View beim Testen fokussiert wird oder ob sie über die Einstellung PLAY MAXIMIZED innerhalb des Editors als Vollbildfenster gestartet wird.

❻ **Frame-Debugger-Button**: Wenn du diesen Button anklickst, öffnet sich ein Fenster, mit dem du exakte Informationen zu dem Bild, das gerade auf dem Bildschirm zu sehen ist, abrufen kannst. Diese Infos sind aber eher fortgeschritten, sodass ich das Fenster zunächst nicht näher beleuchte.

❼ **Lautsprecher-Button**: Zeigt an, ob beim Testen Spielgeräusche zu hören sein sollen.

❽ **Tastatur-Button**: Aktiviert oder deaktiviert Unity-Tastaturkürzel während des Spielens. Damit kannst du beispielsweise verhindern, dass Tastaturkürzel aus Unity mit Tastaturkombinationen aus deinem Spiel kollidieren.

❾ **Stats-Button**: Ein Klick auf STATS zeigt verschiedene technische Statistiken, die während des Testens nützlich sein können.

❿ **Gizmos-Button**: Wenn du auf GIZMOS klickst, aktivierst oder deaktivierst du die Anzeige sogenannter Gizmos im Spielgeschehen. Was genau Gizmos sind, wirst du in Abschnitt 3.3.4 lernen. Über den Pfeil an der rechten Seite kann zudem genau eingestellt werden, welche Gizmos beim Aktivieren der Option zu sehen sind.

Du siehst, es gibt ziemlich viele Einstellungen, die dir das Testen deines Games so einfach wie nur möglich machen sollen. Doch die wichtigsten Elemente der Game View fehlen noch: die Buttons, die das Testen des Spiels starten, beenden und steuern (siehe Abbildung 3.6).

Abbildung 3.6 Kontroll-Buttons der Game View vor dem Spielen (oben) und während des Spielens (unten)

Die Funktionen der Buttons sind nahezu selbsterklärend. Über den Play-Button ❶ in Abbildung 3.6 wird das Spiel in der Game View gestartet. Dabei wird die Scene, die aktuell geöffnet ist, als Startpunkt für den Spielstart gewählt. Wenn du den Button nach

dem Starten erneut betätigst, wird das Spiel wieder beendet. Probiere es ruhig aus: Wenn du das Spiel jetzt startest, kannst du bereits auf deine (noch) leere Welt schauen. Während des Spielens kann die Game View mit dem Pause-Button ❷ pausiert werden. Wird der Button bereits vor dem Starten des Spiels gedrückt, wird das Spiel sofort beim Start pausiert. Der Überspringen-Button ❸ springt einen winzigen Moment (einen sogenannten *Frame*) im Spielgeschehen weiter, während das Spiel pausiert ist. Das ist besonders praktisch, wenn du genau nachvollziehen willst, wie sich deine Spielwelt in winzigen Abschnitten verändert.

Egal, ob Bilder, programmierte Scripts oder Musik, was wäre ein Spiel ohne eine Menge Dateien? Natürlich werden alle für das Spiel benötigten Daten in deinem Projekt abgelegt. Damit du den Überblick über die Dateien nicht verlierst, gibt es im unteren Teil des Editors das PROJECT WINDOW (❸ in Abbildung 3.3). In diesem hast du eine Art Ordner, in dem du die bereits abgelegten Dateien ansehen kannst (siehe Abbildung 3.7). Außerdem kannst du Dateien erstellen, verschieben und löschen.

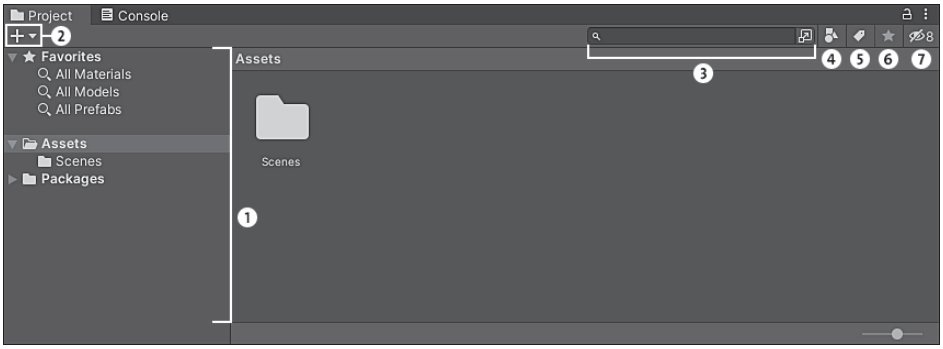


Abbildung 3.7 Übersicht über das Project Window

Neben der großen Anzeige für den aktuell geöffneten Ordner beinhaltet das Project Window einige praktische Tools:

❶ **Schnellzugriff**: Ein praktischer Schnellzugriff auf Suchanfragen, die als Favorit gespeichert wurden. Dabei sind standardmäßig bereits Favoriten angelegt worden, die du aber selbst erweitern kannst. Im unteren Teil der Seitenleiste sind die verschiedenen Ordner des Projekts hierarchisch aufgelistet.

❷ **Plus-Button**: Öffnet ein Menü, mit dem neue Dateien erstellt werden können. Das Erstellen bezieht sich dabei auf den aktuell im Project Window geöffneten Ordner.

❸ **Suchleiste**: Hier kann nach Dateinamen gesucht werden. Über den Button an der rechten Seite kann ein umfangreiches Unity-Suchmenü geöffnet werden.

❹ **Typsuche**: Erlaubt es dir, die Inhalte des Project Windows nach bestimmten Dateitypen zu filtern.

❺ **Labelsuche**: Ermöglicht das Filtern nach bestimmten Labels, die du einzelnen Projektinhalten zuordnen kannst.

❻ **Favoriten-Button**: Speichert die aktuelle Suche als Favorit in der Seitenleiste ❶. Dieser Button ist ausgegraut, wenn gerade keine Suche durchgeführt wird.

❼ **Hide-Button**: Versteckt oder zeigt die Dateien bestimmter Pakete im Project Window. Die Zahl auf dem Button zeigt an, wie viele Pakete aktuell gezeigt oder versteckt werden.

Im Projekt wurde der Ordner *Scenes* automatisch beim Erstellen des Projekts generiert, genau deshalb kannst du ihn im Project Window sehen. Wenn du den Ordner mit einem Doppelklick öffnest, findest du eine Datei mit dem Namen *SampleScene*. Wie du dir vielleicht schon denkst, ist dies eine Scene, die Unity standardmäßig für dich generiert hat – eine wirklich zuvorkommende Engine!

Im Standardlayout verbirgt sich unter dem Project Window der Tab der *Konsole* (Console, ❹ in Abbildung 3.3). Die Konsole bringt uns das erste Mal näher in Richtung der Programmierung. Der Grund dafür: Sie wird genutzt, um sinnvolle Informationen auszugeben, die mit programmierten Scripts zu tun haben. Dazu zählen neben Fehlermeldungen auch bestimmte Warnungen oder informative Textausgaben, die du selbst auslöst. Selbst wenn die Konsole im Moment wie zu erwarten leer ist, wirst du sie schon bald sinnvoll nutzen, um deine ersten eigenen Scripts zu testen.

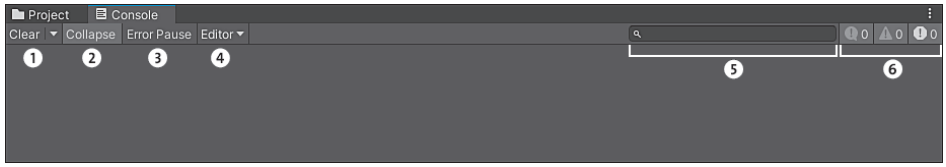


Abbildung 3.8 Übersicht über die Konsole

Die Konsole hält ebenfalls einige nützliche Tools für dich bereit, die wir uns anhand von Abbildung 3.8 genauer ansehen:

❶ **Clear-Button**: Leert den Inhalt der Konsole. Über den kleinen Pfeil an der Seite kann zudem eingestellt werden, ob die Konsole in bestimmten Momenten automatisch geleert werden soll.

❷ **Collapse-Button**: Ist Collapse aktiviert, werden identische Meldungen, die mehrmals hintereinander auftreten, in einer einzigen Meldung »zusammengefasst«. Dabei wird durch eine kleine Ziffer angegeben, wie oft die Meldung aufeinanderfolgend aufgetreten ist. Diese Option ist sehr praktisch, wenn eine identische Meldung dauerhaft und oft hintereinander erscheint. Die Konsole bleibt so übersichtlicher und andere Fehler gehen nicht so schnell in der Masse unter.

❸ **Error-Pause-Button**: Wenn Error Pause angeklickt wurde, pausiert das Spiel sofort, wenn während des Testens ein Fehler auftritt.

❹ **Targets-Menü**: Über das Menü kann die Konsole mit anderen Quellen deines Spiels verbunden werden, um auch externe Meldungen anzuzeigen. Über einen Klick auf <Enter IP> kann die IP-Adresse eines externen Entwicklungs-Builds oder eines Geräts eingegeben werden. Diese Technik ist für das fortgeschrittene Testen angedacht und wird im Buch nicht benutzt.

❺ **Suchleiste**: Wird genutzt, um Konsolenmeldungen nach bestimmten Begriffen abzusuchen.

❻ **Meldungs-Buttons**: Hier kannst du verschiedene Meldungen ein- und ausblenden. Dabei stehen die Buttons von links nach rechts für Fehlermeldungen, Warnungen und informative Ausgaben. Die Zahl neben dem jeweiligen Symbol steht für die aktuelle Anzahl von Meldungen des jeweiligen Typs.

Widmen wir uns dem Fenster am linken Bildschirmrand: In der Hierarchy (❺ in Abbildung 3.3) werden alle Inhalte, die sich in der aktuell geladenen Scene befinden, hierarchisch aufgeführt (siehe Abbildung 3.9).

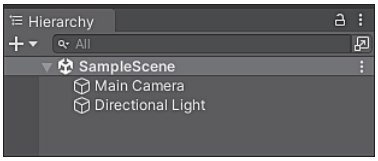


Abbildung 3.9 Übersicht über die Hierarchy

An der Hierarchy ist gut zu sehen, welche Inhalte Unity beim Erstellen des Projekts automatisch für dich generiert hat: Ganz oben ist der Name der geladenen Scene zu sehen. Der Name der Scene entspricht dabei dem Namen der Datei, die du bereits im Project Window sehen konntest: Sample Scene. Mit dem kleinen Pfeil am Namen der Scene können alle Inhalte, die der Scene untergeordnet sind, ein- und ausgeklappt werden. Unter der Scene sind zudem zwei Einträge zu sehen, die automatisch für dich generiert wurden: Main Camera und Directional Light. Welche Bedeutung diese beiden Einträge haben, wirst du in Kürze sehen.

Der Inspector (❻ in Abbildung 3.3) liefert detaillierte Informationen über Dinge, die in deiner Scene abgelegt worden sind. Aus diesem Grund bleibt das Fenster leer, wenn kein Eintrag aus der Scene selektiert ist. Du wirst den Inspector und die verschiedenen Einstellungsmöglichkeiten, die er bietet, deshalb am praktischen Beispiel kennenlernen.

3.1.3 Pakete für das Projekt installieren

Ein weiterer wichtiger Bestandteil deines Spiels sind sogenannte *Pakete*. Mithilfe von Paketen können Inhalte für dein Spiel oder den Editor, die bereits entwickelt wurden, in das Projekt geladen werden. So kannst du Pakete, die von Unity oder anderen Ent-

wicklern und Entwicklerinnen bereitgestellt werden, für dein Projekt verwenden, um dir das Leben einfacher zu machen.

Du kannst später sogar eigene Pakete entwickeln: Stell dir zum Beispiel vor, dass du eine Steuerung für einen Spielcharakter entwickelt hast. Damit du die Steuerung nicht für jedes Spiel neu entwickeln musst, könntest du stattdessen ein Paket erstellen, das du in den verschiedenen Projekten einbindest. Neben programmierten Inhalten kann es sich aber beispielsweise auch um grafische Inhalte, Erweiterungen für den Editor oder visuelle Effekte handeln.

Unity stellt eine Vielzahl von Paketen bereit, die über ein praktisches Fenster installiert und verwaltet werden können. Dieses Fenster heißt *Package Manager* und kann über das Menü WINDOW • PACKAGE MANAGER in der oberen Menüleiste geöffnet werden (siehe Abbildung 3.10).

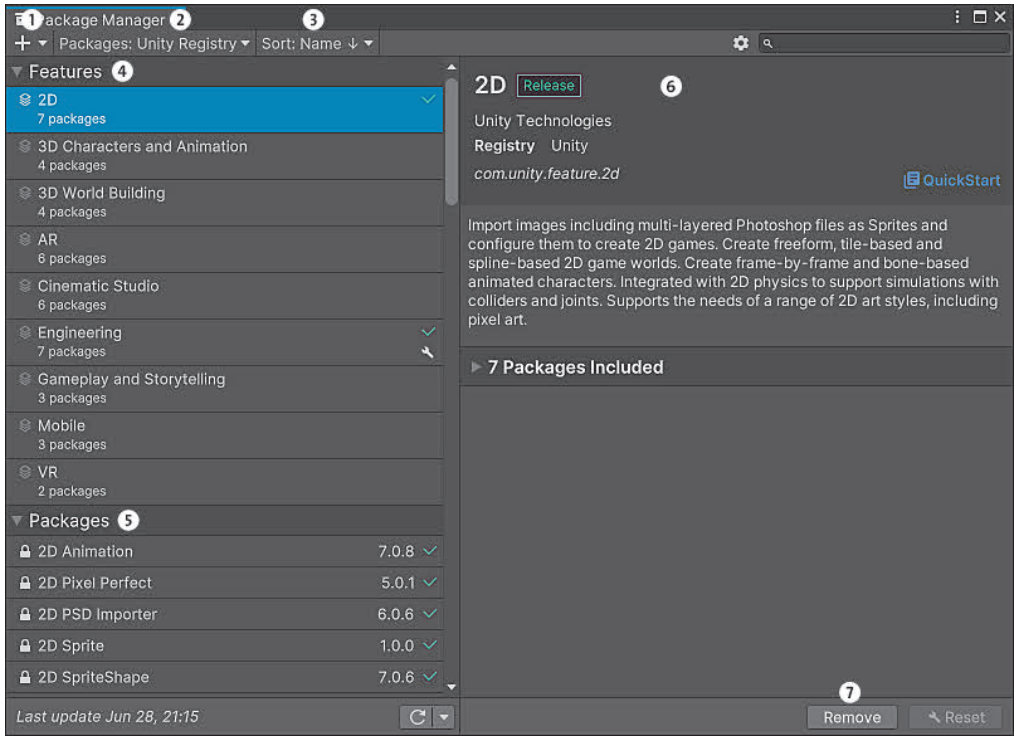


Abbildung 3.10 Der Package Manager bietet eine Übersicht zur Verwaltung von Paketen, die für das Projekt verwendet werden.

In dem Menü aus Abbildung 3.10 stehen verschiedene Bereiche zur Verfügung, die wir uns genauer ansehen wollen:

❶ Über den Button können neue Pakete installiert werden. Eine Installation ist dabei zum Beispiel aus Package-Dateien oder externen Systemen möglich.

❷ Mithilfe dieses Menüs wird ausgewählt, welche Paketauswahl angezeigt wird. Dabei stehen verschiedene Optionen zur Verfügung:

- Unity Registry: offizielle Pakete von Unity
- In Project: Pakete, die im Projekt installiert wurden
- My Assets: Pakete, die über den Unity Asset Store (dieser wird in Kürze erläutert) hinzugefügt wurden
- Built-In: vorinstallierte Pakete

❸ Legt die Sortierung der Anzeige fest.

❹ Zeigt sogenannte *Features*, die zur aktuellen Auswahl passen. Ein Feature besteht aus mehreren Paketen, die zusammenhängend einen bestimmten Zweck erfüllen. Je nach gewähltem Template des Projekts werden bestimmte Features vorinstalliert.

❺ Zeigt einzelne Pakete, die zur aktuellen Auswahl passen.

❻ Zeigt eine Übersicht des aktuell ausgewählten Packages oder Features.

❼ Über den Button kann das gewählte Feature im aktuellen Projekt installiert (oder deinstalliert) werden. Nach der Installation stehen die Inhalte des Pakets innerhalb des Projekts zur Verfügung. Manche Pakete benötigen besondere Schritte zur Einrichtung. Diese sind aber meist in der Paketbeschreibung hinterlegt.

Neben den offiziell von Unity angebotenen Paketen gibt es die Möglichkeit, weitere Pakete von anderen Entwicklern und Entwicklerinnen herunterzuladen. Dazu stellt Unity eine Website bereit, die den Namen *Asset Store* trägt. Der Asset Store ist über die offizielle Website erreichbar: *https://assetstore.unity.com/*

Die Website ist gut bedienbar und stellt viele Werkzeuge bereit, um Packages zu sortieren oder gezielt zu suchen. Aus diesem Grund erläutere ich den Umgang mit der Website nicht näher. Wichtig ist, dass viele angebotenen Packages kostenpflichtig sind. Gleichzeitig musst du beachten, welche Lizenzen für Packages verwendet werden – je nachdem ist beispielsweise die Nennung des Urhebers oder der Urheberin eines Packages notwendig, wenn das Spiel verkauft werden soll. Nach dem Kauf eines Pakets (oder wenn das Package von Beginn an kostenlos war), erscheint auf der Webseite des Packages ein Button, mit dem es in die persönliche Asset-Liste aufgenommen wird. Danach siehst du das Paket im Package Manager, wenn die Auswahl Packages: My Assets selektiert ist. Wichtig ist dabei, dass für den Login im Unity Asset Store der gleiche Login wie für Unity Hub verwendet wird – ansonsten kann das Package im Manager nicht gefunden werden.

3.1.4 Ordnerstruktur des Projekts

Beim Begutachten des Project Windows konntest du sehen, dass der Ordner SCENES für dich erstellt wurde. Doch dein erstelltes Projekt besteht nicht nur aus einem Ordner, der eine Scene beinhaltet. Schauen wir uns also auf dem PC den Ordner an, in dem das Projekt generiert wurde (siehe Abbildung 3.11):

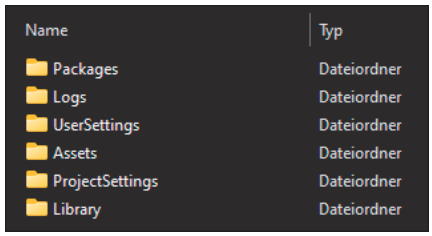


Abbildung 3.11 Inhalt des erstellten Projektordners

- *Logs*: Hier werden verschiedene Berichte erstellt, die im Zusammenhang mit den von Unity verwendeten Prozessen stehen. Ausgaben, die von der Konsole erzeugt werden, zählen im Normalfall nicht dazu.
- *Packages*: Dieser Ordner beinhaltet Informationen zu Paketen, die für das Projekt benötigt werden. Standardmäßig werden beim Erstellen des Projekts mit einem Template verschiedene Pakete automatisch eingebunden. Installierst du später neue Pakete, werden Informationen zu diesen im Ordner gespeichert. So wird sichergestellt, dass das Projekt die notwendigen Pakete lädt, wenn es auf einem neuen Rechner geöffnet wird.
- *Library*: Speichert die Daten der verschiedenen Dateien und Pakete, die im Projekt verwendet werden. Es handelt sich dabei um die für dein System notwendigen Formate. Wird der Ordner gelöscht, wird er beim nächsten Öffnen des Projekts neu erzeugt und alle Dateien werden neu für dein System konvertiert. Das ist einer der Prozesse, der beim ersten Starten des Projekts so lange gedauert hat.
- *Temp*: Enthält temporäre Dateien, die Unity erstellt und verwaltet, während das Projekt geöffnet ist. Wird das Projekt geschlossen, wird dieser Ordner automatisch gelöscht und erst beim nächsten Öffnen des Projekts wieder neu generiert.
- *UserSettings*: Sichert die nutzerspezifischen Einstellungen, wie etwa erstellte Layouts für den Unity Editor oder Einstellungen für die erweiterte Suche.
- *ProjectSettings*: Speichert verschiedene Einstellungen, die direkt zum Projekt gehören. Diese Einstellungen, die zum Beispiel die Standardtastenbelegung oder Grafikeinstellungen für das Projekt umfassen, sind wichtig, damit das Spiel korrekt gespielt werden kann.
- *Assets*: Hier liegen alle Dateien, die für die eigentliche Entwicklung des Spiels anfallen.

Assets in der Spieleentwicklung

In der Spieleentwicklung mit Unity werden alle Inhalte, die mit der Entwicklung des eigentlichen Games zusammenhängen, als *Asset* bezeichnet. Von 3D-Modellen und Bildern, die in der Scene platziert werden, über Audiodateien, visuelle Effekte bis hin zu Scripts wirst du noch häufig über das Synonym Asset stolpern. Außerhalb von Unity wird dieser Begriff in der Spiele- und Softwareentwicklung häufig benutzt, wenn es um Dateien geht, die für die Anwendung notwendig sind.

Ein Blick in den Ordner *Assets* zeigt zwei Inhalte: den Ordner *Scenes* und eine Datei *Scenes.meta*. Im Ordner *Scenes* sind zwei weitere Dateien enthalten: *SampleScene.unity* und *SampleScene.unity.meta*.

Vielleicht erkennst du, dass es einen Zusammenhang zwischen dem Ordner *Assets* und dem Project Window aus Unity gibt – immerhin ist dort ebenfalls der Ordner *Scenes* zu sehen, der die Datei *SampleScene* beinhaltet. Merk dir also: Das Project Window zeigt den Ordner *Assets* an. Da Unity diesen Ordner vorsieht, um Dateien für unser Spiel zu speichern, ist es auch gar nicht notwendig, die anderen Ordner im Project Window anzuzeigen.

Vielleicht fragst du dich zum Schluss noch, was es mit den Dateien auf sich hat, die die Endung *.meta* besitzen. Diese Dateien werden automatisch für jede Datei, die im Ordner *Assets* liegt, angelegt und werden als sogenannte *Metadateien* bezeichnet. Unter einer Metadatei versteht man eine Datei, die weitere Informationen zu der eigentlichen Datei speichert. Diese Informationen benötigt lediglich Unity, um intern mit den Dateien arbeiten zu können. Genau aus diesem Grund wirst du nie selbst solche Metadateien erstellen oder bearbeiten müssen. Um die Übersichtlichkeit zu behalten, werden alle Metadateien im Project Window versteckt – auch, wenn Unity sie im Hintergrund für dich anlegt und verwaltet.

Wichtige Ordner für die Weitergabe eines Projekts

Nicht alle Dateien, die Unity nach dem ersten Start generiert, sind für die Weitergabe des Projekts notwendig. Es wäre unpraktisch, wenn du das Projekt auf einem anderen PC aufsetzen oder an jemanden weitergeben möchtest und dafür Ordner kopierst, die gar nicht notwendig sind. Stell es dir am Beispiel des Ordners *UserSettings* vor: Eine Freundin, die dein Projekt auf ihrem PC mit einem eigenen Unity Editor ansieht, möchte nicht das gleiche Layout wie du nutzen, nur weil es dein Projekt ist.

Beim Weitergeben deines Projekts sind von den oben genannten Ordnern deshalb lediglich drei notwendig:

- *Assets*
- *Packages*
- *ProjectSettings*

Alle anderen Ordner werden automatisch generiert, sobald das Projekt erneut geöffnet wird, und sollten deshalb nicht weitergegeben werden.

3.2 Game Objects und Components

Für die nächsten Schritte wird die Unterscheidung zwischen zwei wichtigen Begriffen notwendig, die dich bei der Arbeit mit Unity ständig begleiten werden.

3.2.1 Game Objects

Sehen wir uns dafür zuerst noch einmal die HIERARCHY des neu erstellten Projekts an. Wie du schon beim Kennenlernen der Hierarchy gemerkt hast, sind unter dem Namen deiner Scene, SAMPLESCENE, zwei Einträge vermerkt (siehe Abbildung 3.12).

Abbildung 3.12 Unter der Hierarchy wurden zwei Einträge erstellt.

Hinter den beiden Begriffen MAIN CAMERA und DIRECTIONAL LIGHT verbirgt sich die gleiche Struktur: ein *Game Object*. Oder um es mit anderen Worten auszudrücken: In der Scene, die beim Erstellen des Unity-Projekts für dich angelegt wurde, wurden automatisch zwei Game Objects generiert.

In dem Zusammenhang kannst du dir zunächst eine Faustregel merken, die deine Arbeit in Unity stets begleiten wird: Alles ist ein Game Object. Wie es der Name andeutet, stellt jedes Game Object ein eigenes Objekt dar, das innerhalb deiner Spielwelt existiert. Egal, ob dein Spielcharakter, ein einfacher Baum oder eine Kugel, die von einer Kanone abgefeuert wird: In Unity wird jeder Bestandteil deiner Spielwelt mithilfe eines Game Objects dargestellt.

Du wirst das Konzept noch besser verstehen, wenn du dir die beiden Game Objects anschaust, die bereits in der Scene angelegt wurden. Wirf dafür noch einmal einen Blick in deine Hierarchy, wo beide Objekte unter dem Namen der Scene aufgelistet sind. Wenn du eines der beiden Game Objects anklickst, wird es innerhalb deiner Scene View hervorgehoben. So kannst du schnell einordnen, wo sich das jeweilige Game Object in der Scene befindet.

Bevor wir uns genauer anschauen, wie du Game Objects in der Scene verändern kannst, ist jedoch viel interessanter, wie ein solches Game Object aufgebaut ist. Wenn

du ein Game Object in der Hierarchy anwählst, siehst du nicht nur, dass es in der Scene hervorgehoben wird, sondern auch, dass dein Inspector zum Leben erwacht. Während er davor leer aussah, ist die rechte Seite nun mit einer Menge an Informationen gefüllt (siehe Abbildung 3.13).

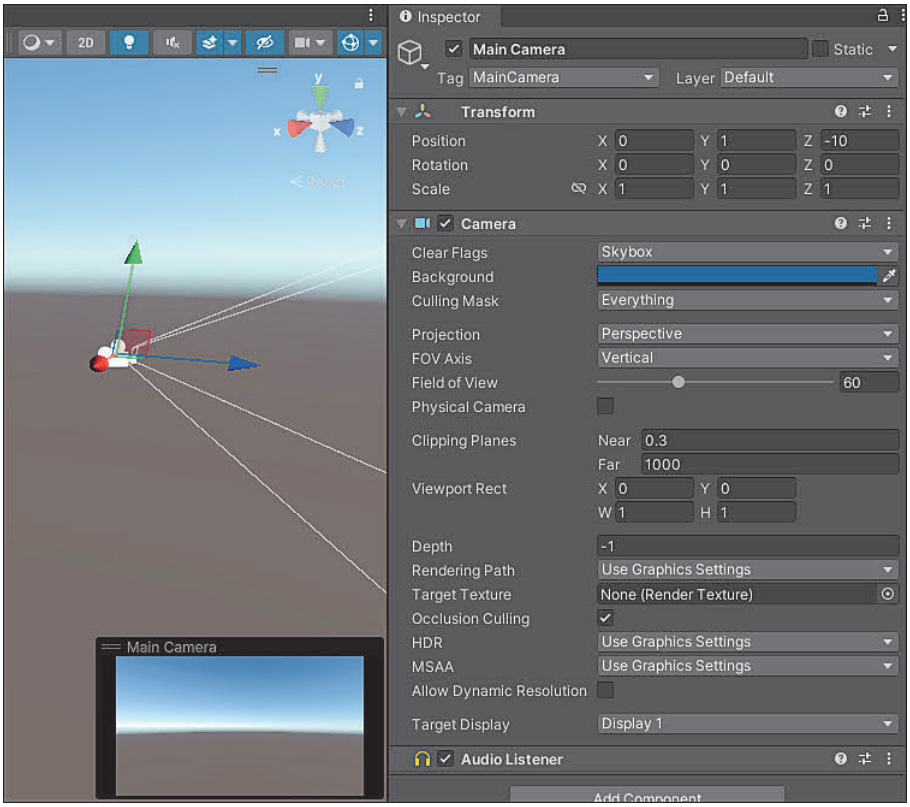


Abbildung 3.13 Beim Anklicken eines Game Objects in der Hierarchy wird es in der Scene hervorgehoben und der Inspector mit Informationen gefüllt.

Diese Informationen bringen uns zum nächsten wichtigen Begriff, der direkt mit Game Objects zusammenhängt.

3.2.2 Components

Das Konzept der sogenannten *Components* wird besser verständlich, wenn du es an einem Beispiel kennenlernst. Dafür schauen wir uns das Game Object an, das mit dem Namen Main Camera automatisch generiert wurde. Beim Anklicken erscheinen im Inspector Informationen, die zu genau diesem Game Object gehören (siehe Abbildung 3.14).

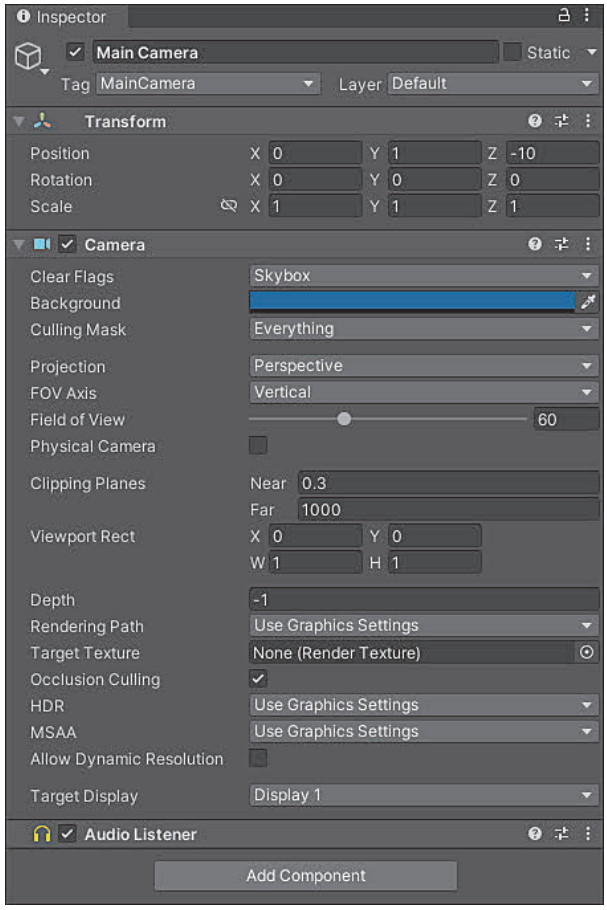


Abbildung 3.14 Ansicht des Inspectors, nachdem das Game Object »Main Camera« in der Hierarchy angeklickt wurde

Im oberen Bereich sind verschiedene Randinformationen zum Objekt abgelegt. Dazu gehört beispielsweise der Name des Objekts, Main Camera, der auch in der Hierarchy zu sehen war. Die beiden Einträge Tag und Layer sind ebenfalls Einstellungen, die das Game Object selbst betreffen. Diese werden wir später genauer unter die Lupe nehmen.

Interessant wird es darunter: Du siehst verschiedene kleine Abschnitte, die wiederum Einstellungen beinhalten. Im Fall des ausgewählten Objekts Main Camera tragen diese Abschnitte die Namen Transform, Camera und Audio Listener. Bei jedem dieser Abschnitte handelt es sich um eine sogenannte Component, die auf dem Game Object liegt. Wie du sehen kannst, besitzt jede Component eigene Eigenschaften. Durch diese kann jede einzelne Component genau eingestellt werden.

Auch wenn du noch nicht weißt, welche Aufgabe die verschiedenen Components erfüllen, die auf dem Game Object abgelegt wurden, lässt sich eine wichtige Grunderkenntnis ableiten. Das Game Object ist eine Art Box, die verschiedene Components in sich stapeln kann. Diese Components besitzen wiederum Eigenschaften, mit deren Hilfe genau eingestellt werden kann, wie sich die Component verhält. Oder um es mit anderen Worten zu sagen: Das Game Object ist selbst nur eine leere Hülle, der durch verschiedene Components Leben eingehaucht wird. Magisch!

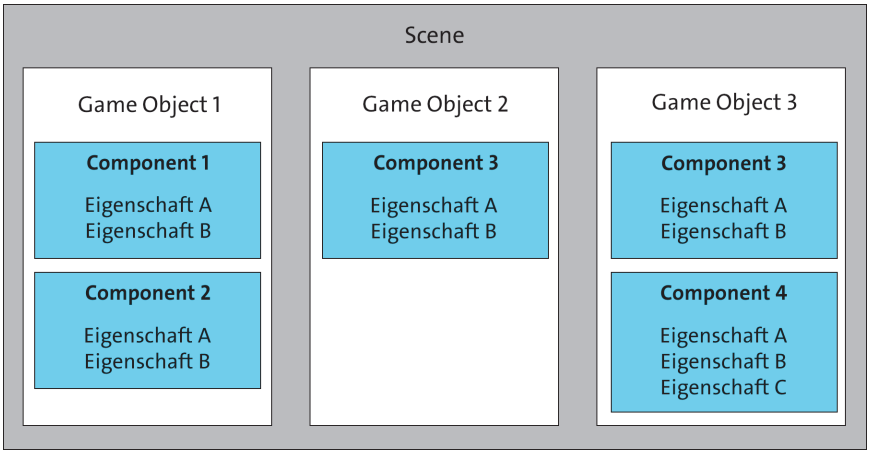


Abbildung 3.15 Skizze zur Beziehung zwischen Scene, Game Objects und Components

Wie du in der Skizze der Abbildung 3.15 erneut sehen kannst, besitzt eine Scene verschiedene Game Objects. Jedes Game Object wiederum besitzt beliebig viele Components, die wiederum selbst Eigenschaften besitzen, mit denen sie genauer konfiguriert werden. Wie viele Eigenschaften eine Component besitzt, ist von der jeweiligen Component abhängig. Wie du ebenfalls in Abbildung 3.15 sehen kannst, kann die gleiche Component von beliebig vielen verschiedenen Game Objects genutzt werden. So wird im Beispiel Component 3 von Game Object 2 und Game Object 3 genutzt. Ein wichtiger Unterschied: Die Objekte nutzen **die gleiche** Component, aber nicht **dieselbe**. In anderen Worten: Game Object 2 besitzt eine eigene Version der Component und Game Object 3 wiederum eine andere. Beide Components können somit auf jedem Game Object unterschiedlich eingestellt werden. Das ist ein gravierender Unterschied, beide Components hängen nicht zusammen!

Das generelle Konzept verwirrt dich vielleicht. Wozu brauchen wir überhaupt Game Objects, wenn sowieso die Components die gesamte Arbeit übernehmen? Warum gibt es verschiedene Components, wenn das Game Object doch nur eine Aufgabe im Spiel erfüllt? Ein praktisches Beispiel wird dir helfen, diese und ähnliche Fragen zu beantworten.

3.2.3 Beispiel: Ein Würfel ohne Haut? Components in der Praxis verstehen

Um den Zusammenhang zwischen Game Objects und Components besser zu verstehen, wollen wir zum ersten Mal ein eigenes Game Object erstellen. Als Beispiel dafür wirst du einen einfachen Würfel in der 3D-Scene ablegen.

Unity bietet dir die Möglichkeit, über ein einfaches Menü verschiedene geometrische Körper zu generieren, darunter zum Beispiel Kugeln, Zylinder oder auch Würfel. Beginnen wir also, indem du einen solchen Würfel in der Scene anlegst. Wähle dazu in der oberen Menüleiste die Option GameObject • 3D Object • Cube (siehe Abbildung 3.16). Das gleiche Menü kannst du öffnen, indem du in der Hierarchy den Plus-Button drückst oder mit der rechten Maustaste klickst. In Unity führen viele Wege zum Ziel!

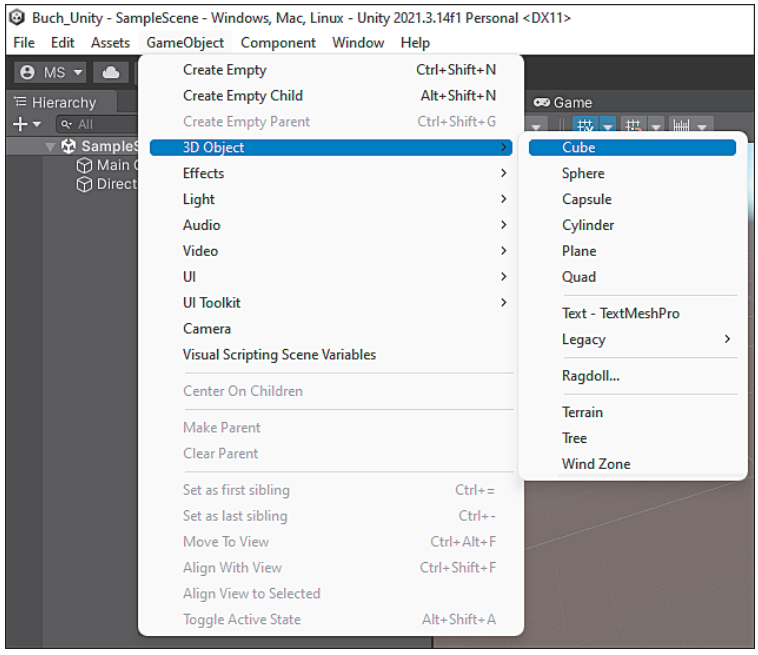


Abbildung 3.16 Menüauswahl, um ein einfaches würfelförmiges Game Object anzulegen

Nachdem du auf die Option geklickt hast, solltest du sehen, dass in deiner Hierarchy ein neues Game Object angelegt wurde. Kurz nach dem Erstellen kannst du einen Namen für das neue Objekt eintippen. Der gewählte Standard Cube reicht aber für den ersten Moment aus. Natürlich kann der Name des Game Objects im Nachhinein angepasst werden. Der Würfel befindet sich jetzt in der Scene und du kannst ihn deshalb in der Scene View finden. Wenn der Würfel angewählt ist, wird er in der Scene View außerdem orange umrandet (siehe Abbildung 3.17). Auf dem Würfel werden gleichzeitig drei bunte Pfeile angezeigt. Diese sind ein Werkzeug zum Verschieben, das du schon bald nutzen wirst.

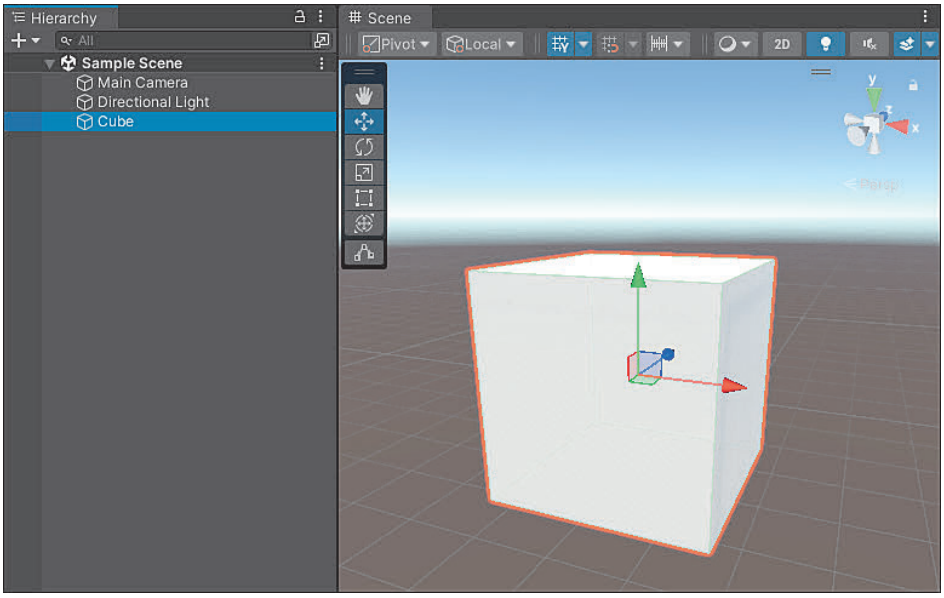


Abbildung 3.17 Der neu erstellte Würfel ist in der Hierarchy und in der Scene View sichtbar.

[i]

Wenn du deinen Würfel nicht mehr in der Scene sehen kannst

Wenn du deinen neu erstellten Würfel nicht sehen kannst, so hast du dich vielleicht durch die Scene View bewegt. Dadurch kann es sein, dass du den Würfel gerade nicht anschaust oder er einfach sehr weit von deinem Blickpunkt in der Scene View entfernt ist.

Du kannst aber einfach zum Würfel springen, indem du den Namen des Game Objects, Cube, in der Hierarchy doppelt mit der linken Maustaste anklickst. Alternativ kannst du das Objekt einmal anklicken, sodass es selektiert wird und anschließend die Taste F auf der Tastatur antippen. Auf beiden Wegen sollte dein Würfel dann in der Scene View *fokussiert* werden.

Schauen wir uns an, welche Components das neu erstellte Game Object Cube besitzt. Nach dem Selektieren des Objekts kannst du alle Components im Inspector betrachten (siehe Abbildung 3.18).

Ähnlich wie die Main Camera, die wir uns ebenfalls genauer ansehen werden, besitzt der neu erstellte Würfel verschiedene Components: Transform, Cube (Mesh Filter), Mesh Renderer und Box Collider. Auch wenn du noch nicht genau weißt, welche Bedeutung diese Components besitzen, ist der Grundgedanke der gleiche: Unser Würfel wird nur durch die Components zum Würfel. Diese Eigenschaft ist noch viel besser zu sehen, indem eine der Components testweise entfernt wird. Nehmen wir dafür die Component mit dem Namen Mesh Renderer. Über die drei Punkte in

der Kopfzeile oder einen Rechtsklick auf die Kopfzeile selbst kann ein Bearbeitungsmenü geöffnet werden. Über einen Klick auf Remove Component wird die Component Mesh Renderer vom Würfel entfernt (siehe Abbildung 3.19).

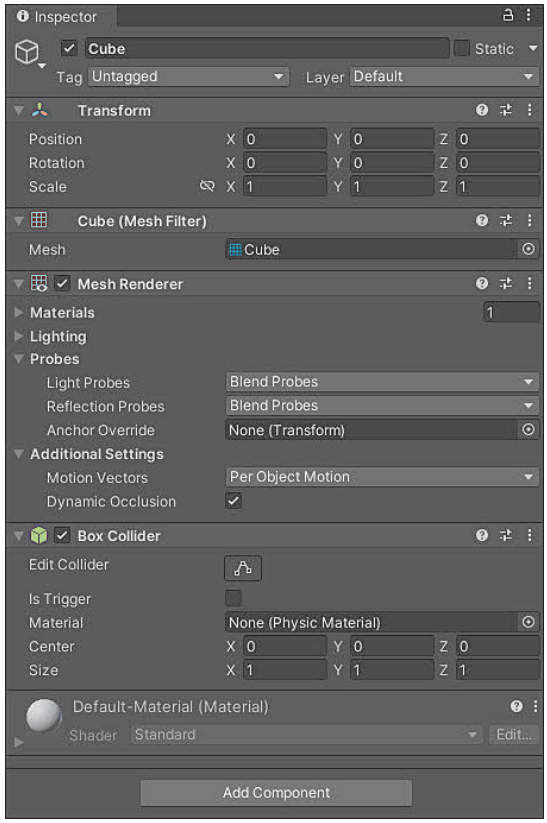


Abbildung 3.18 Die Components des neu erstellten Würfels

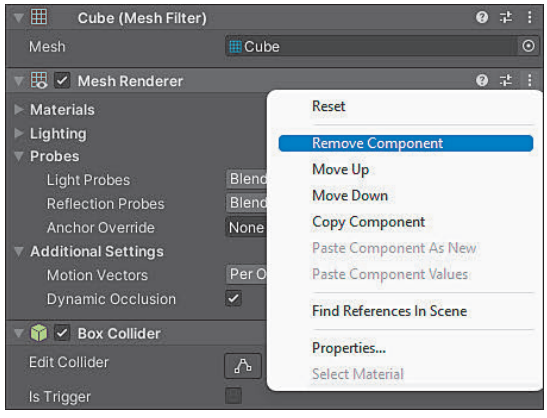


Abbildung 3.19 Menü zum Entfernen der Component »Mesh Renderer«

Das Ergebnis lässt nicht lange auf sich warten: Unser armer Würfel hat seine Oberfläche verloren! Wenn du den Würfel nicht mehr auswählst, ist er komplett unsichtbar. Beim Anwählen erscheint lediglich ein grüner Umriss des Würfels (siehe Abbildung 3.20).

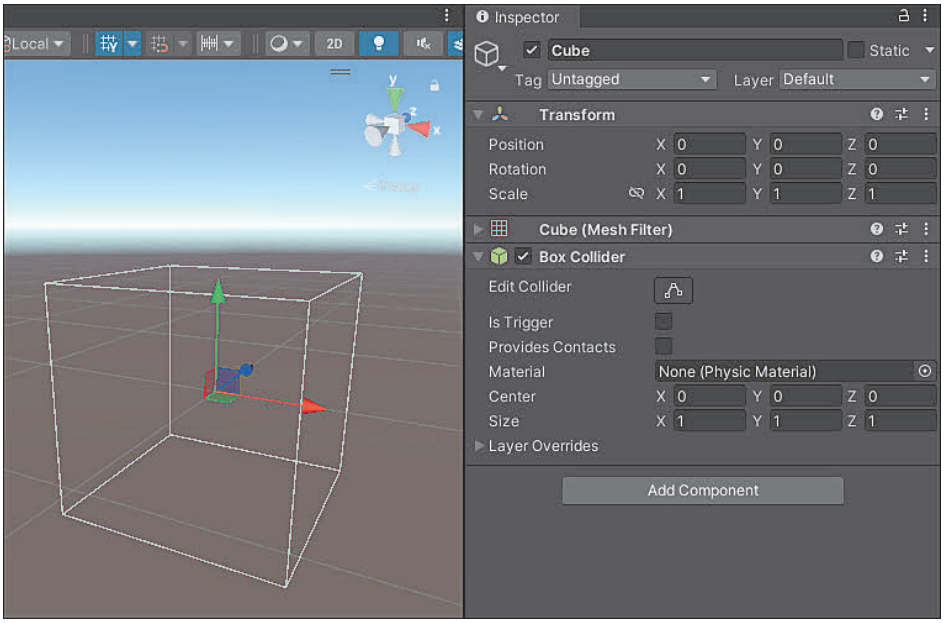


Abbildung 3.20 Der nackte Würfel nach dem Entfernen der Component

Offensichtlich war die Component MESH RENDERER dafür zuständig, dass der Würfel in der Scene sichtbar ist. Indem diese Fähigkeit entfernt wird, ist der Würfel nicht mehr zu sehen – was natürlich nicht bedeutet, dass er nicht immer noch unsichtbar in der Scene liegen würde. Schließlich hast du lediglich die Component entfernt, die für das Darstellen der Würfeloberfläche notwendig ist.

Was also, wenn wir unserem Würfel seine Haut »wiedergeben« wollen? Ganz logisch – dem Game Object muss eine passende Component hinzugefügt werden, mit der es in der Scene sichtbar wird. Du weißt bereits, dass die Component MESH RENDERER genau das kann. Fügen wir also dem Würfel diese Component wieder hinzu und schauen, was passiert. Wähle dazu zuerst wieder das Game Object CUBE aus und klicke im INSPECTOR auf den Button ADD COMPONENT. Dadurch öffnet sich ein Menü, in dem du alle Components hinzufügen kannst, die dir zur Verfügung stehen. Es gibt eine große Menge an verfügbaren Components, doch du weißt, welche Component gebraucht wird. Somit kannst du in der Suchleiste des Menüs »Mesh Renderer« eingeben und dem Würfel diese Component mit einem Klick auf die linke Maustaste wieder hinzufügen (siehe Abbildung 3.21).

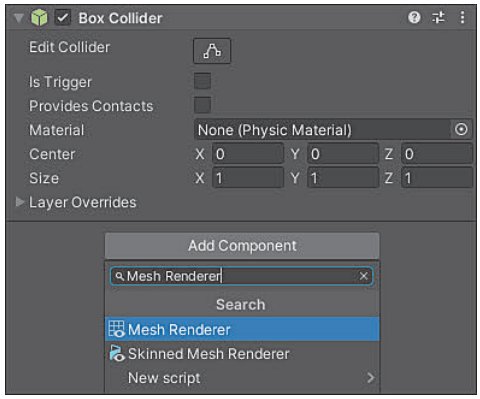


Abbildung 3.21 Hinzufügen einer neuen »Mesh Renderer«-Component über den Inspector

Das Ergebnis könnte deine Augen kurz brennen lassen. Unser Würfel ist wieder sichtbar, allerdings in einem grellen Pink. Falls du dich darüber wunderst: Das ist lediglich eine Voreinstellung. Damit Unity weiß, wie der Würfel am Ende aussehen soll, muss das Aussehen der Oberfläche in der Component MESH RENDERER festgelegt werden. Da wir diesen Schritt bisher nicht getan haben, nimmt Unity eine Standardfarbe. Und das ist, wie in Abbildung 3.22 zu sehen ist, ein wunderschönes, kaum zu übersehendes Pink.

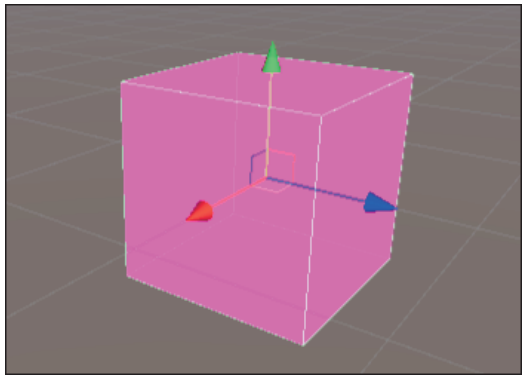


Abbildung 3.22 Die Ansicht des Würfels, nachdem eine neue »Mesh Renderer«-Component hinzugefügt wurde

Auch wenn der Würfel nun etwas verschandelt aussieht, war dieses Experiment sehr wichtig, um besser zu verstehen, wie genau Components funktionieren. Du hast gesehen, dass ein Game Object nur von den Components ausgemacht wird, die auf ihm liegen. Dieses Konzept macht Components sehr flexibel und gut wiederverwendbar. Einem Game Object werden neue Funktionen hinzugefügt, indem ihm neue Components zugewiesen werden. Genauso können Funktionen wieder entfernt werden, wenn bestimmte Components vom Game Object gelöscht werden. Stell dir für ein

Spiel vor, dass eine Springmaus eine Component besitzt, mit der sie springen kann. Wenn du diese Component entfernst, wäre es plötzlich nur noch eine normale Maus.

Um weitere Augenschmerzen zu vermeiden, kannst du den Würfel schlussendlich löschen, er hat uns in den Experimenten genügend Dienste erwiesen. Wähle nach einem Rechtsklick auf das Game Object in der HIERARCHY dafür die Option DELETE (siehe Abbildung 3.23). Alternativ kannst du den Würfel in der HIERARCHY anwählen und mit der Taste Entf löschen.

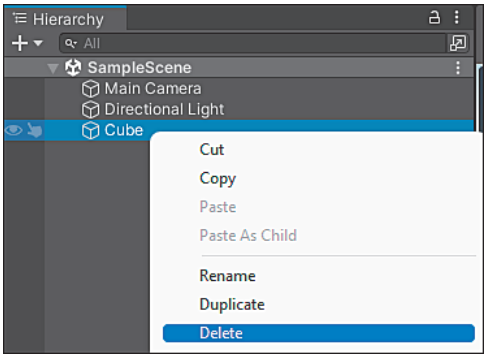


Abbildung 3.23 Genug getan, der Würfel wird aus der Scene gelöscht.

3.2.4 Das Game Object als leerer Behälter

Das vorangegangene Experiment verdeutlicht noch einmal, dass ein Game Object an sich nichts ist außer einem leeren Behälter. Dieser Behälter wartet nur auf praktische Components, die ihm schlussendlich Leben einhauchen.

Dieses Konzept lässt sich gut nachvollziehen, indem du ein vollkommen leeres Game Object in deiner Scene anlegst. Natürlich könntest du genauso einfach einen neuen Würfel erzeugen und alle Components von diesem entfernen. Dankenswerterweise hat Unity aber vorgebeugt und eine Option eingebaut, um ein neues, leeres Game Object zu erzeugen. Klicke dafür in der Menüleiste auf GAMEOBJECT • CREATE EMPTY. Wie beim Würfel kannst du diese Option über den Plus-Button oder einen Rechtsklick in der Hierarchy erreichen. Im Anschluss kannst du einen beliebigen Namen für das leere Game Object auswählen.

Wie zu erwarten ist kein neues 3D-Objekt in der Scene zu sehen (siehe Abbildung 3.24). Stattdessen siehst du lediglich die drei Pfeile, die auch beim Würfel angezeigt wurden. Der INSPECTOR ist ebenfalls deutlich leerer als beim Anklicken des Würfels. Statt drei Components ist lediglich ein Eintrag lesbar: TRANSFORM. Unser »leeres« Game Object ist also doch gar nicht so leer. Was es mit dieser besonderen Transform-Component auf sich hat, wirst du schon im nächsten Abschnitt 3.2.5 lernen. Neben dieser automatisch hinzugefügten Component handelt es sich aber um ein unbeschriebenes Blatt Papier. Das Game Object ist – um es schlicht und ergreifend zu sa-

gen – im Moment völlig sinnlos. Es erfüllt keine Funktion und ist noch nicht einmal in der Scene sichtbar. Sinnloser geht es kaum!

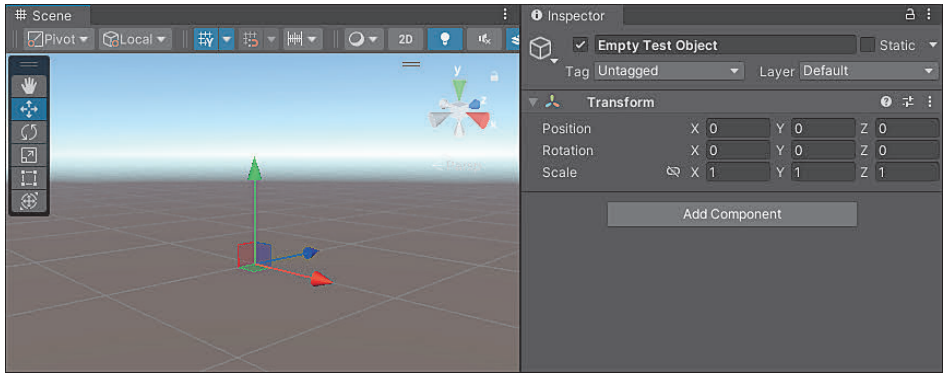


Abbildung 3.24 Ein neu erstelltes, leeres Game Object in der Scene

Du siehst also noch einmal, dass Game Objects nur über hinzugefügte Components zum Leben erwachen. Würden wir diesem leeren Game Object die exakt gleichen Components hinzufügen wie dem Würfel aus dem letzten Beispiel (und die Einstellungen aller Components exakt so übernehmen, wie sie beim Würfel waren), würde das Game Object genau den Würfel zeigen.

3.2.5 Die Transform-Component

Alle Game Objects, die du bisher kennengelernt hast, haben unterschiedliche Components benutzt. Dennoch hatten sie alle eine Component gemeinsam, selbst wenn du das Game Object vollkommen leer generiert hast: die *Transform-Component*. Schauen wir uns also an, warum diese besondere Component von keinem Game Object wegzudenken ist. Um die Wirkung dieser Component besser zu zeigen, verwenden wir wieder einen Würfel. Generiere dir also zunächst einen neuen Würfel in deiner Scene, genauso, wie du es bereits in Abschnitt 3.2.3, »Beispiel: Ein Würfel ohne Haut? Components in der Praxis verstehen«, gemacht hast.

Unter der Transform-Component sind drei Eigenschaften aufgeführt: Position, Rotation und Scale. Anhand dieser Begriffe kannst du dir gut herleiten, wofür die Transform-Component da ist: Sie beschreibt, wie dein Game Object in deiner Scene platziert wird.

Wie es der Name andeutet, legt die Eigenschaft *Position* fest, wo im Raum das Game Object liegt. Unity verwendet dabei ein dreidimensionales Koordinatensystem. Das wollen wir uns im Detail anschauen. Die Position hat einen Zahlenwert für X, Y und Z. Falls dich das an Mathematik erinnert: Du hast recht, wir haben es hier mit einem

grundlegenden geometrischen System zu tun. Aber keine Sorge, wir lassen es ruhig angehen und schauen uns die Bedeutung der drei Buchstaben der Skizze in Abbildung 3.25 an.

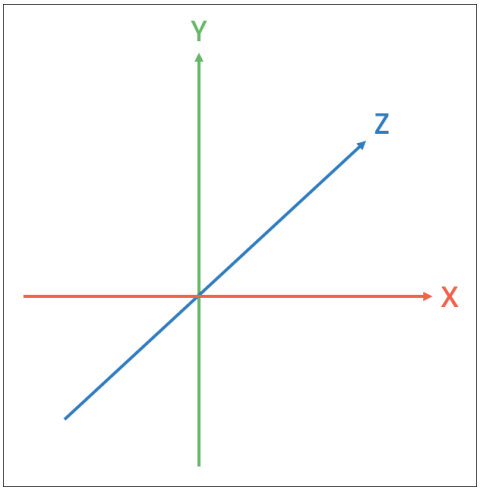


Abbildung 3.25 Exemplarische Ansicht eines Unity-3D-Koordinatensystems

Wie du sehen kannst, steht jeder der Buchstaben für eine bestimmte Gerade im dreidimensionalen Raum – eine sogenannte *Achse*. Dabei trägt in Unity die Achse von links nach rechts die Bezeichnung X, von unten nach oben die Bezeichnung Y und die Achse von hinten nach vorn die Bezeichnung Z. Im geometrischen Kontext sprechen wir von einem sogenannten *Koordinatensystem*.

Bei einer Sammlung der drei Werte für X, Y und Z sprechen wir von einer *Koordinate*. Wie du schon in Abbildung 3.25 sehen konntest, besitzt jede Achse eine bestimmte Richtung. In dieser Richtung wird entlang einer Achse der Wert von X, Y oder Z größer. Bewegt man sich in die andere Richtung, wird der Wert kleiner. Der Wert kann dabei auch negativ (kleiner als 0) werden. Eine Koordinate wird oft in der Schreibweise (X, Y, Z) dargestellt. Die Schreibweise (1, –1, 3) entspricht also einer Koordinate mit den Werten 1 für X, –1 für Y und 3 für Z. Der »Startpunkt«, an dem sich alle drei Achsen treffen, wird als *Koordinatenursprung* bezeichnet. Der Koordinatenursprung besitzt somit die Koordinate (0, 0, 0).

Koordinaten lassen sich sehr gut visualisieren. Stell dir vor, wir wollen unsere Spielfigur, eine kleine Kugel, im Unity-Koordinatensystem an die Koordinate (2, 3, 1) setzen. Dafür müssen wir lediglich zwei Einheiten nach rechts (X), drei Einheiten nach oben (Y) und eine Einheit nach vorn (Z) gehen. Anschließend befindet sich die Kugel an einem neuen Koordinatenpunkt. Die Zusammensetzung dieser Koordinate wird in Abbildung 3.26 visualisiert.

Du weißt nun also, dass die Werte für die Einordnung des Game Objects im virtuellen Raum verwendet werden. Als Nächstes wollen wir uns im Detail anschauen, welche Bedeutung die einzelnen Eigenschaften der Transform-Component haben.

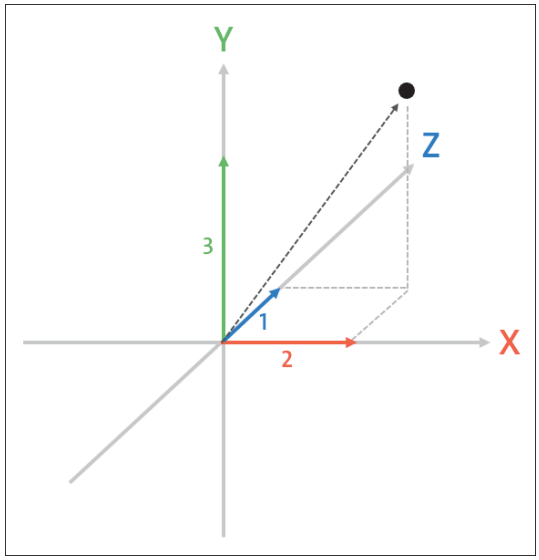


Abbildung 3.26 Visualisierung des Koordinatenpunktes (2, 3, 1) im Unity-Koordinatensystem

[i]

Für den Würfel sind bereits Zahlenwerte für die Position eingetragen

Deine Position hat bereits Zahlenwerte? Das ist sogar sehr wahrscheinlich. Standardmäßig wird beim Erstellen eines neuen Game Objects die Position dort festgelegt, wo deine Maus zuletzt in der Scene View aktiv war. Du kannst die Position aber einfach auf den Koordinatenursprung zurücksetzen. Der umständliche Weg dafür ist, einfach für alle Werte unter X, Y und Z die Zahl 0 einzutragen.

Es gibt allerdings auch einen einfacheren Weg. Wenn du auf die drei Punkte in der Kopfzeile der Transform-Component klickst oder einen Rechtsklick in die Kopfzeile machst, kannst du Reset wählen. Alle Werte der Position werden dann ebenfalls auf den Standardwert 0 zurückgesetzt.

Wie du es dir anhand der Skizze in Abbildung 3.26 herleiten kannst, wird das Objekt je nachdem, wie du die Zahlen veränderst, im Raum verschoben. Hier kannst du experimentieren: Trage neue Zahlenwerte für die Position ein und beobachte in der Scene View, wie sich dein Würfel bewegt.

Hier gibt es eine nützliche Technik, mit der du Zahlenwerte im Inspector noch einfacher anpassen kannst: Wenn du mit der Maus über einen Eigenschaftswert wie X

fährst, siehst du neben dem Mauscursor kleine Pfeile (siehe Abbildung 3.27). Hältst du nun die linke Maustaste gedrückt, kannst du die Maus seitlich nach links oder rechts bewegen, um den Wert zu verändern. Wenn du die Maus nach links ziehst, wird der Wert immer kleiner, während sich der Wert beim Ziehen nach rechts erhöht. Das ist sehr praktisch, um Zahlenwerte schnell zu verändern. Während sich der Wert ändert, kannst du in der Scene View beobachten, wie sich dein Würfel in Echtzeit bewegt.

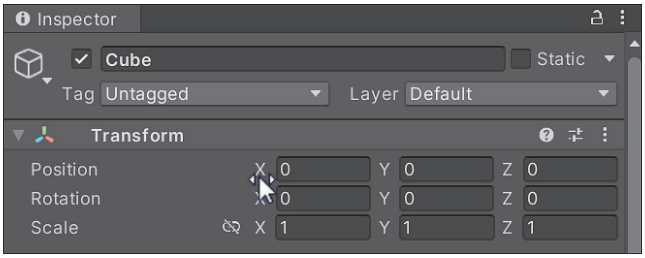


Abbildung 3.27 Wenn du mit der Maus über eine Zahleneigenschaft fährst, die linke Maustaste gedrückt hältst und die Maus nach links oder rechts bewegst, wird der Wert kleiner oder größer.

Anhand der Vorkenntnisse, die du nun zum Koordinatensystem hast, kannst du gut nachvollziehen, wie ein Verschieben zustande kommt. Setzen wir als Beispiel unseren Würfel manuell auf die Koordinate (–1, 1, 3), indem du die Werte direkt in die einzelnen Felder der Positionseigenschaft einträgst. Das bedeutet in anderen Worten vom Koordinatenursprung ausgehend: um eine Einheit nach links, um eine Einheit nach oben und um drei Einheiten nach vorn. Du kannst die Änderungen der Position des Würfels noch besser sehen, wenn du in die Game View wechselst.

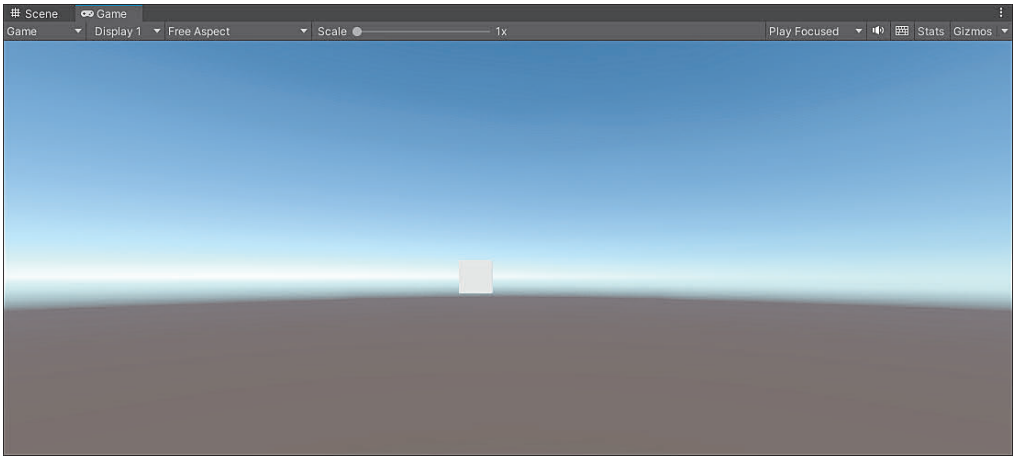


Abbildung 3.28 Das Verschieben des Würfels in der Game View

In Abbildung 3.28 solltest du die leichte Verschiebung besonders in die linke Richtung aus der Mitte der Spielansicht erkennen. Versuche, noch weiter mit der Positionseigenschaft herumzuexperimentieren. Du wirst so am besten verstehen, wie das dreidimensionale Koordinatensystem funktioniert.

Die zweite Eigenschaft der Transform-Component ist die *Rotation*, die auch als Drehung des Game Objects bezeichnet werden kann. Diese bezieht sich auf die Drehung entlang der jeweiligen Achse. Anders als die Position wird die Rotation als Gradzahl angegeben. Die Gradzahl besitzt dabei einen Wert von 0 Grad (keine Drehung) bis 360 Grad (eine volle Drehung). Ist der Wert größer als 360, beginnt eine neue Umdrehung. Es können auch Werte gesetzt werden, die kleiner als 0 sind. Die Drehung geht dann in die andere Richtung, bis beim Wert –360 eine volle Drehung erreicht wird.

Am besten siehst du, wie die Rotation funktioniert, wenn du deinen Würfel in der Szene rotieren lässt (nutze dafür zum einfachen Testen das Ziehen mit der Maus wie in Abbildung 3.27) und dabei beobachtest, welchen Einfluss die Veränderung der Werte auf die Drehung hat. Ein Beispiel für eine Rotation des Würfels wird in Abbildung 3.29 angewendet.

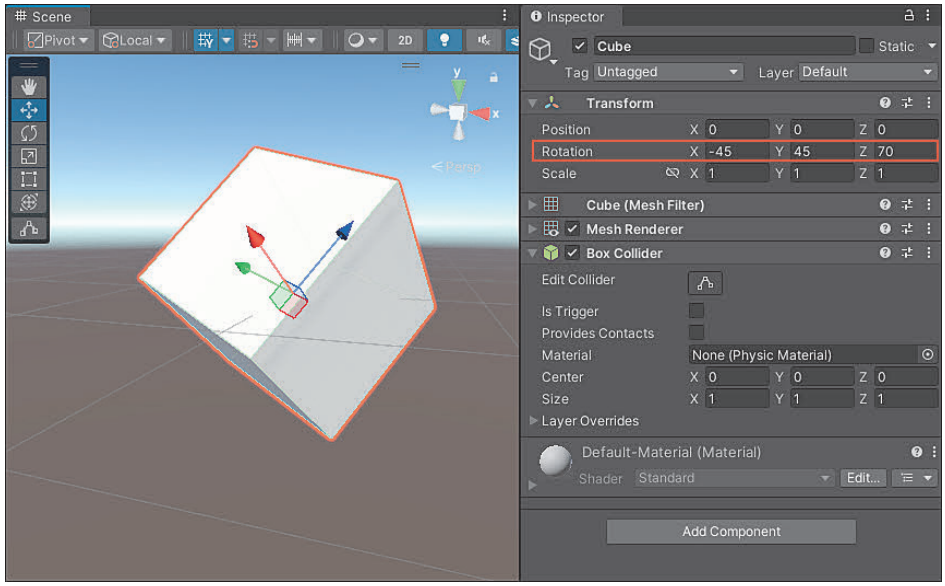


Abbildung 3.29 Beispielhafte Rotation des Würfels

Die letzte Eigenschaft der Transform-Component ist *Scale*, die Skalierung des Game Objects. Diese gibt die Größe des Game Objects an und kann genutzt werden, um das Objekt zu strecken oder zusammenzustauchen. Wie du siehst, besitzt diese Eigen-

schaft standardmäßig den Wert (1, 1, 1). Das bedeutet, dass das Objekt auf allen Achsen mit 100 % seiner Originalgröße dargestellt wird.

Veränderst du den Scale-Wert, so wird das Objekt entlang der jeweiligen Achse gestreckt oder zusammengedrückt. Ein Wert zwischen 0 und 1 bedeutet dabei, dass das Objekt entlang der Achse verkleinert wird. Ein Wert von 0,5 steht so zum Beispiel für die Hälfte der Originalgröße. Alle Werte größer als 1 skalieren die jeweilige Achse höher. Wird ein negativer Wert verwendet, findet die Skalierung in die jeweils andere Richtung der Achse statt – so, als würdest du die Rotation spiegeln.

Wie bei allen anderen Eigenschaften lohnt es sich hier, die Veränderung der Werte am Beispiel des Würfels nachzuvollziehen. So kann aus dem gängigen Würfel schnell eine klobige Platte werden (siehe Abbildung 3.30).

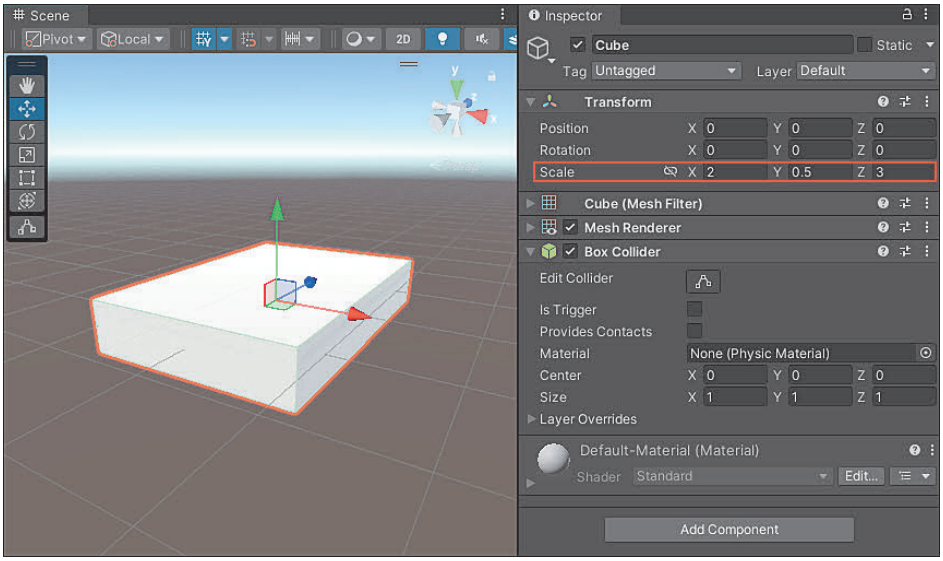


Abbildung 3.30 Beispielhafte Skalierung des Würfels

Oft ist es bei der Skalierung gewünscht, dass alle drei Werte gleichmäßig gesetzt werden, damit das Objekt nicht auf einer Achse unnatürlich verzerrt aussieht. Du kannst das mit einem Bild vergleichen: Wenn du das Bild nur nach rechts vergrößerst, wird der restliche Teil des Bildes verzerrt, und das sieht nicht gut aus. Dafür bietet Unity bei der Scale-Eigenschaft das Klammersymbol an, die sogenannte *Constrained Proportions*. Wenn diese aktiviert sind, wird das Verhältnis (die Proportion) zwischen allen drei Werten immer gleich gehalten.

Aktivieren wir diese Eigenschaft auf dem skalierten Würfel aus Abbildung 3.30 und verändern anschließend einen Wert der Scale-Eigenschaft, so werden alle anderen Eigenschaften gleichmäßig mit verändert (siehe Abbildung 3.31).

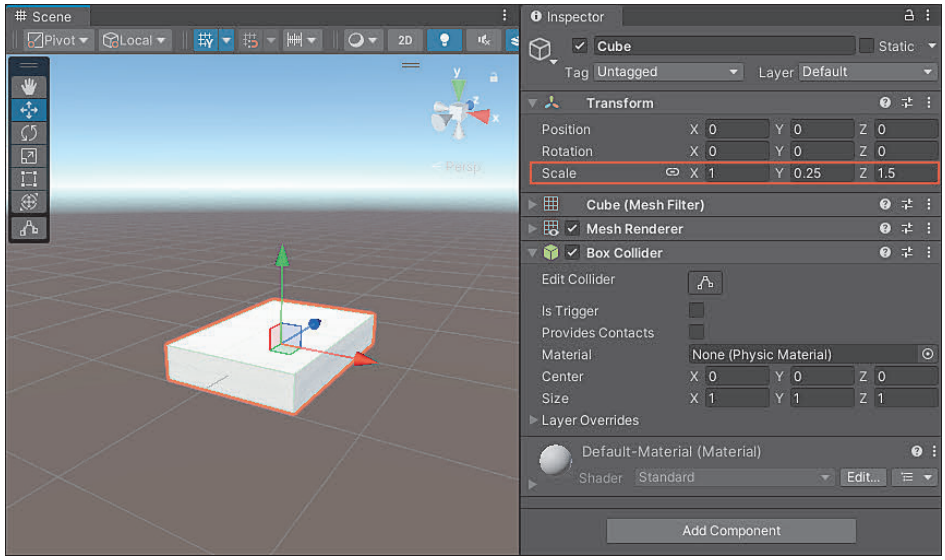


Abbildung 3.31 Skalieren des veränderten Würfels mit aktivierten Constrained Proportions

3.2.6 Übung 03.01: Objektumformung

Durch die Transform-Component kannst du aktiv werden und erste eigene Inhalte für ein richtiges Spiel bauen. Auch wenn die von Unity gegebenen Objekte trist aussehen, sollst du dich in dieser Übung austoben und mithilfe der Transform-Component verschiedene 3D-Objekte verwandeln, und zwar in richtige Dinge! Verwende dafür lediglich die Eigenschaften der Transform-Component. Verwandle folgende Grundobjekte:

- Cube in den Schiefen Turm von Pisa (einen etwas geneigten Turm)
- Sphere in ein Ei
- Cylinder in ein Floß

Die Grundformen Sphere und Cylinder kannst du, ebenso wie einen Cube, über Game Object • 3D Object in der Menüleiste finden.

3.2.7 Licht und Kamera – die Standardobjekte der 3D-Scene

Eine Frage bleibt dennoch offen: Welche Bedeutung haben die beiden Game Objects, die bereits in der Scene angelegt worden sind? Die Namen geben bereits einen guten Hinweis, dennoch wollen wir dieser Frage noch auf den Grund gehen. Schauen wir uns dafür noch einmal die beiden Game Objects an, die automatisch in der Scene angelegt wurden.

Main Camera

Die Main Camera ist, wie es die deutsche Übersetzung »Hauptkamera« andeutet, dafür zuständig, dass wir beim Spielen etwas sehen können. Gewissermaßen werden die Augen des Spielers durch die Main Camera dargestellt. Der Ort und die Ausrichtung der Kamera ist dafür ausschlaggebend, was beim Starten des Spiels zu sehen ist. Unsere Kamera steht vollkommen still, sodass wir ein Bild sehen, das sich nicht bewegt.

Doch was, wenn keine Kamera mehr in unserer Scene ist? Wenn das Game Object Main Camera testweise gelöscht wird, kann in der Game View keine Anzeige mehr erfolgen – immerhin wäre das so, als würden wir unserem Spieler die Augen verbinden. Stattdessen erscheint die Meldung No cameras rendering (siehe Abbildung 3.32).

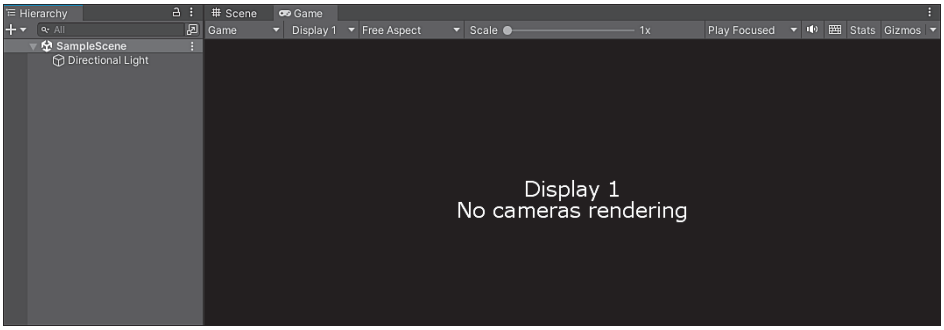


Abbildung 3.32 Ansicht der Game View nach Löschen der »Main Camera«

Eine Kamera ist also unabdingbar, damit beim Starten des Spiels überhaupt etwas zu sehen ist. Zuständig dafür ist die Component Camera, die auf dem Game Object Main Camera liegt. Die Component besitzt eine Reihe nützlicher Einstellungen, die wir nicht alle im Detail ansehen wollen. Wir konzentrieren uns auf die wichtigsten Eigenschaften, mit denen du am häufigsten zu tun hast, wenn du eine Kamera für deine Scene einrichtest:

- Clear Flags: Gibt an, was in den Bereichen angezeigt wird, an denen keine Game Objects mit einem Aussehen liegen. Du legst also den Hintergrund deiner Scene fest. Es stehen mehrere Optionen zur Auswahl, wobei zwei besonders wichtig sind:
  - Skybox: Es wird ein vordefinierter Hintergrund angezeigt, der im Regelfall ein Himmel ist. Das ist der Standard in unserem 3D-Template, wodurch der voreingerichtete blaue Himmel im Hintergrund zu sehen ist. Wie du eine eigene Skybox einrichtest, lernst du in Abschnitt 15.3.3, »Eine Skybox einrichten«.
  - Solid Color: Es wird eine bestimmte Farbe im Hintergrund angezeigt, die du im Inspector festlegen kannst.

- Projection: Legt fest, wie Objekte durch die Kamera dargestellt werden. Hier wird zwischen zwei Formen unterschieden, die du wählen kannst:
  - Perspective: Objekte werden so dargestellt, wie du es im echten Leben erwartest. Dabei werden Objekte, die weiter von der Kamera entfernt sind, kleiner dargestellt als Objekte, die direkt vor der Kamera stehen. Dadurch ist diese Einstellung besonders für 3D-Spiele ein Standard. Immerhin sehen wir im echten Leben quasi auch so: Die Sonne ist ziemlich groß, von der Erde aus gesehen wirkt sie aber nur wie ein kleiner Ball.
  - Orthographic: Alle Objekte werden in der gleichen Größe dargestellt, egal, wie nah sie an der Kamera liegen. Dieser Effekt wird vor allem bei 2D-Spielen genutzt, da diese oftmals keine »Tiefe« in die Scene hinein benötigen.

Der Unterschied zwischen perspektivischer Ansicht und orthografischer Darstellung ist sehr wichtig dafür, wie Spieler dein Game am Ende sehen. Abbildung 3.33 zeigt einen Vergleich der beiden Darstellungen, wobei in beiden Fällen drei versetzt hintereinander angeordnete Würfel in der Scene liegen.

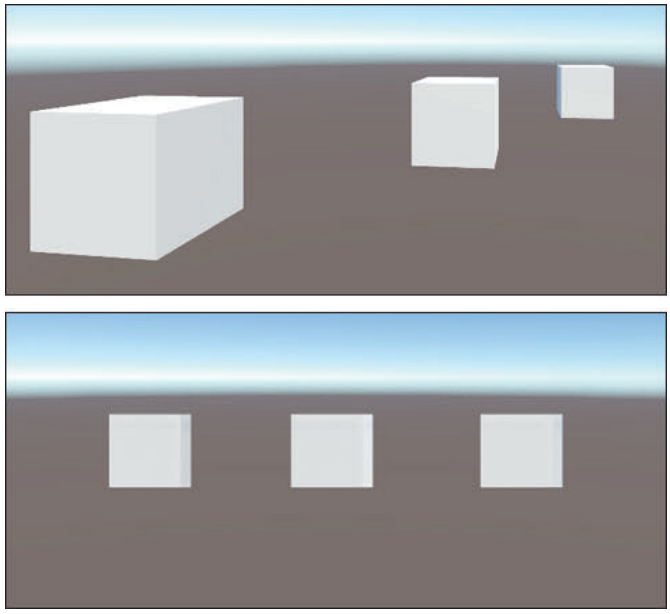


Abbildung 3.33 Darstellung der gleichen Scene in perspektivischer Ansicht (oben) und orthografischer Ansicht (unten)

Je nachdem, welche Perspektive die Kamera verwendet, stehen weitere Einstellungen bereit, um festzulegen, welchen Bereich eine Kamera erfasst:

- FOV Axis (nur bei Perspective): Legt die Achse fest, entlang der das Sichtfeld der Kamera aufgespannt wird (Horizontal oder Vertical).

- Field of View (nur bei Perspective): Setzt den Winkel, den das Sichtfeld der Kamera erfasst.
- Size (nur bei Orthogprahic): Legt die Größe des Bereichs fest, den die Kamera aufnimmt.

Eine Übersicht über alle weiteren Einstellungen der Component findest du in der Dokumentation unter *https://docs.unity3d.com/Manual/class-Camera*.

Directional Light

Nun zum zweiten Game Object, das für dich angelegt wurde. In einer Welt ohne Sonne wäre es ziemlich dunkel. Nicht anders geht es da Unity. Genau aus diesem Grund enthält die neu generierte Scene standardmäßig ein Directional Light, durch das Objekte gut sichtbar werden. Welche Bedeutung das Licht besitzt, lässt sich gut sehen, wenn es testweise entfernt wird. Der Sinn kommt besonders dann zum Vorschein, wenn gleichzeitig noch unser altbekannter Würfel in der Scene liegt.

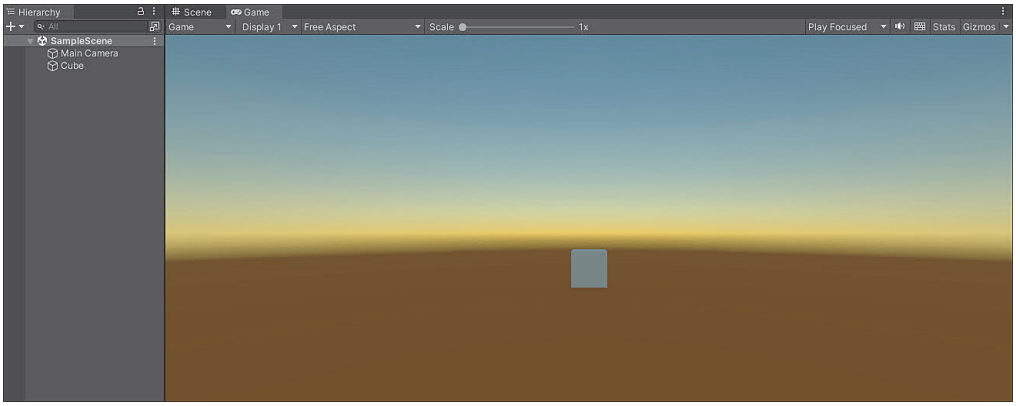


Abbildung 3.34 Ansicht der Game View mit Würfel nach Löschen des »Directional Light«

Wie du in Abbildung 3.34 sehen kannst, hat sich der Hintergrund der Scene bedeutend verdunkelt, der Himmel hat von einem satten Blau zu tristem Orange gewechselt. Außerdem wird der Würfel nicht mehr angeleuchtet. Da es kein Licht mehr in der Scene gibt, könnten Objekte wie der Würfel außerdem keinen Schatten mehr auf andere Game Objects der Scene werfen.

Zuständig für das Standardlicht ist die Component Light, die auf dem Game Object Directional Light liegt. Das Licht ist so konfiguriert, dass überall in der Scene Licht in einem gleichmäßigen Winkel strahlt. Man spricht dabei von *Directional Light*, von gerichtetem Licht. Das soll eine Sonne, die schräg vom Himmel scheint, auf möglichst einfache Weise simulieren. Die Details zu allen Einstellungen, die du für Licht in deiner Scene treffen kannst, lernst du in Abschnitt 15.2.1, »Die Light-Component«, kennen.

3.2.8 Game Objects und Components deaktivieren

Oftmals kann es praktisch sein, einzelne Game Objects oder auch Components zu deaktivieren. So kannst du bestimmte Objekte erst dann anzeigen, wenn sie im Spiel benötigt werden, oder sie wieder verstecken, wenn sie nicht länger gebraucht werden. Stell dir ein Spiel vor, bei dem nach Ende des Spiels eine Anzeige erscheint, die die erzielte Punktzahl zeigt: Du könntest alle Elemente vorbereiten und das zugrunde liegende Game Object deaktivieren. Wenn das Spiel beendet ist, kannst du die Anzeige dann einblenden, indem das Objekt aktiviert wird.

Das Deaktivieren eines Game Objects ist simpel: Nachdem das gewünschte Objekt selektiert wurde, muss in der linken oberen Ecke des Inspectors der gesetzte Haken entfernt werden (siehe am Beispiel des DIRECTIONAL LIGHT in Abbildung 3.35). Dadurch wird das Objekt deaktiviert und so behandelt, als ob es nicht in der Scene wäre. Somit funktionieren auch die Components nicht mehr, die auf dem Objekt liegen. Nach dem Deaktivieren wird der Name des Objekts zusätzlich in der HIERARCHY ausgegraut.

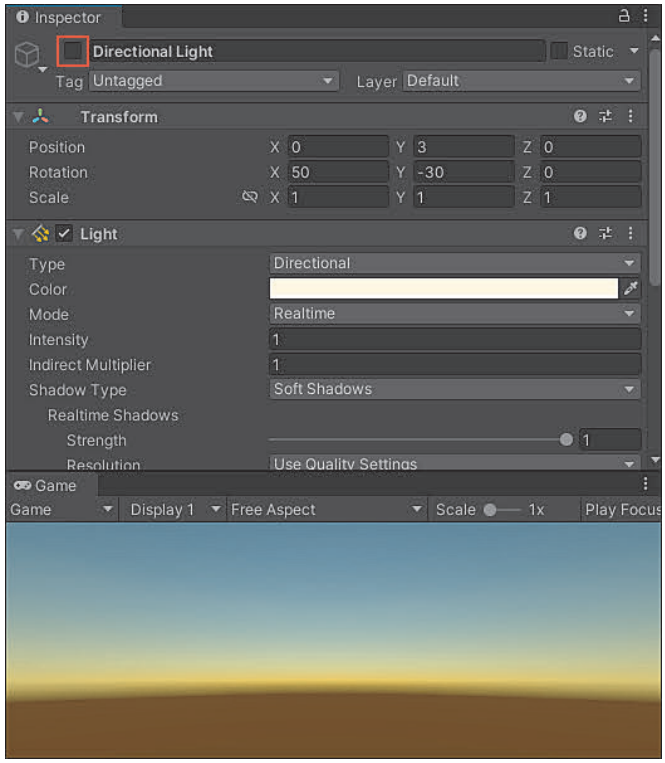


Abbildung 3.35 Ansicht der Game View, nachdem das »Directional Light« deaktiviert wurde

Es muss allerdings nicht immer das ganze Objekt deaktiviert werden. Stattdessen kann es einzelne Components treffen, die über den Haken links neben der Compo-

nent deaktiviert werden können (siehe Abbildung 3.36). Dadurch wird lediglich die gewählte Component so behandelt, als ob sie nicht mehr am Game Object enthalten wäre. Beim Deaktivieren der LIGHT-Component unseres DIRECTIONAL LIGHT würde das Licht ebenso in der Scene fehlen.

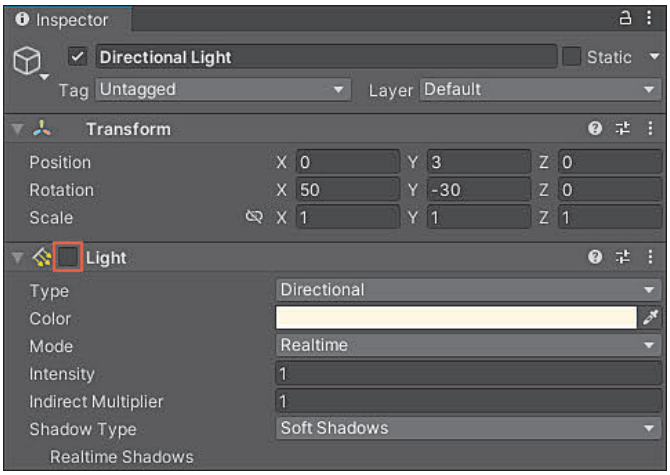


Abbildung 3.36 Deaktivieren der Component »Light« über den Inspector

Das Deaktivieren einzelner Game Objects oder Components kann übrigens besonders beim Testen helfen, um schnell zu ermitteln, wie sich das Spiel verhält, wenn bestimmte Bestandteile im Spiel fehlen. Das volle Potenzial wird sich aber erst entfalten, wenn du Game Objects oder Components deiner Scene automatisch mithilfe der Programmierung deaktivierst.

3.3 Orientierung in der Scene View

Du weißt nun, dass du deine virtuelle Welt in der Scene View baust und diese Welt aus Game Objects besteht, die verschiedene Components enthalten. Gerade deshalb ist es wichtig, dass du dich in der Scene View gut orientieren kannst und lernst, welche nützlichen Werkzeuge dir mit an die Hand gegeben werden, um eine Spielwelt nach deinen Vorstellungen bauen zu können. Und du kannst mir glauben: Wenn du erst mal ihre Macht kennst, willst du viele der Werkzeuge nicht mehr aus der Hand legen.

3.3.1 Maus- und Tastaturkürzel in der Scene View

Mit verschiedenen Maus- und Tastaturkürzeln ermöglicht es Unity, dich schnell in deiner virtuellen Welt zurechtzufinden. Im Folgenden lernst du die wichtigsten Kürzel kennen, die dir das Arbeiten mit der Scene View erleichtern. Dabei beziehen sich die angegebenen Kürzel auf die Standardeinstellungen von Unity, die überschrieben

werden können. Das Menü zum Einsehen aller Shortcuts ist über EDIT • SHORTCUTS erreichbar. Dort können bestimmte Tastaturkürzel hinzugefügt werden, die normalerweise nicht gesetzt sind. Unity erlaubt es, für nahezu jede denkbare Option Kürzel hinzuzufügen – das Menü gibt dir einen guten Überblick.

Das Drehen in der Scene View erfolgt, indem du die Maus bewegst, während die rechte Maustaste gedrückt gehalten wird. Du kannst dich, während du die rechte Maustaste gedrückt hältst, außerdem mit den Tasten W (nach vorn), A (nach links), S (nach hinten) und D (nach rechts) in der 3D-Welt bewegen. Die Bewegung nach vorn und hinten geht alternativ über die Tasten ↑ und ↓, zusätzlich kann durch das Scrollen des Mausrades in die Scene hinein- oder aus ihr herausgezoomt werden. Du kannst dich außerdem bewegen, indem du die mittlere Maustaste (das Scrollrad) gedrückt hältst und dabei die Maus bewegst. Nach links und rechts kannst du dich alternativ mit ← und → bewegen. Ein Game Object kann in der Scene View durch einen Klick mit der linken Maustaste selektiert werden.

3.3.2 Die Transform Toolbar

In Abschnitt 3.2.5 hast du die Transform-Component im Detail kennengelernt und gesehen, welche Wirkung das Verändern der Eigenschaften über den Inspector hat. Das manuelle Anpassen der Zahlenwerte ist allerdings eher unpraktisch, wenn du die Wirkung direkt in deiner Scene begutachten möchtest. Aus diesem Grund gibt es in der Scene View eine Reihe an nützlichen Tools, die das Verändern der Transform-Component direkt in der Scene View möglich macht. Das Tool, das du am häufigsten nutzen wirst, ist die *Transform Toolbar* (siehe Abbildung 3.37). Beachte, dass die Position der Toolbar durch das Verschiebensymbol innerhalb des Editor-Fensters verändert werden kann. Dadurch kann die Anordnung der einzelnen Buttons abweichen.

Abbildung 3.37 Die Transform Toolbar

Sehen wir uns die Tools im Einzelnen an. Jedes Werkzeug aus der Transform Toolbar kann zusätzlich über eine Taste angesteuert werden. Die Zusammenfassung für alle Werkzeuge findest du in Tabelle 3.1.

Tool	Beschreibung	Taste
View Tool ❶	Dient nur der Fortbewegung in der Scene. Wird es ausgewählt, kannst du dich durch Klicken mit der linken Maustaste so in der Scene bewegen, als ob du die mittlere Maustaste (das Scrollrad) drückst.	Q
Move Tool ❷	Die bisherige Standardauswahl. Bei diesem Tool werden am Game Object Pfeile angezeigt, die das Objekt entlang der jeweiligen Achse bewegen. Zwischen den Pfeilen sind kleine Flächen, mit denen das Objekt entlang einer Ebene bewegt wird. Probiere es einfach aus, das Bewegen von Objekten ist mit diesem Tool kinderleicht!	W
Rotate Tool ❸	Dient der Drehung eines Objekts. Es erscheinen verschiedene Kreise um das Objekt, die das Rotieren entlang einer bestimmten Achse erlauben.	E
Scale Tool ❹	Ermöglicht das Setzen der Skalierung. Auch hier kann jede Achse für sich in ihrer Größe bearbeitet werden. Alternativ können alle Achsen gleichmäßig verändert werden, indem der Würfel in der Mitte aller Achsen gezogen wird.	R
Rect Tool ❺	Wenn du dieses Tool wählst, erscheint ein Umriss, der das Skalieren auf einer zweidimensionalen Ebene ermöglicht. Dieses Tool ist besonders praktisch, wenn die Größe von 2D-Inhalten wie Bildern oder Elementen der Benutzeroberfläche verändert werden soll.	T
Transform Tool ❻	Vereint das Move Tool, das Rotate Tool und das Scale Tool. Alle Achsen zur Veränderung erscheinen gleichzeitig. Beim Einstieg ist das unübersichtlich, aber für Ungeduldige, die den Überblick bewahren können, ist dieses Tool sehr praktisch.	Y
Edit Bounding Volume ❼	Dient der Anpassung bestimmter Umrisse von Objekten. Diese Auswahl wird nur dann eingeblendet, wenn ein Objekt mit einer Collider-Component ausgewählt wird. Dazu gehören beispielsweise die von Unity bereitgestellten 3D-Grundobjekte wie der Würfel.	keine Taste

Tabelle 3.1 Übersicht über die Werkzeuge der Transform Toolbar

Eine Übersicht über die drei »Grundtools« für das Verändern der Position, der Rotation und der Skalierung wird in Abbildung 3.38 gegeben.

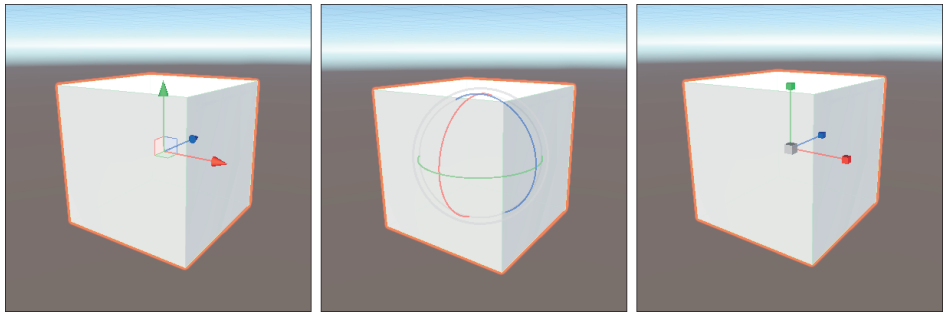


Abbildung 3.38 Die drei wichtigen Transform-Tools Move (links), Rotate (Mitte) und Scale (rechts)

Oft müssen Tools zur Anpassung der drei Haupteigenschaften POSITION, ROTATION und SCALE in festgelegten Schritten oder an bestimmten Punkten verwendet werden – beispielsweise, wenn ein Tisch immer genau 1 m weiter nach vorn verschoben werden soll. Dafür bietet Unity verschiedene Techniken zum sogenannten *Snapping* an:

- *Increment Snapping*: Wird verwendet, wenn die Werte in festen Schritten verändert werden sollen. Diese Form wird aktiviert, indem bei der Nutzung eines der Tools die Taste Strg gedrückt gehalten wird. Beim Move Tool wird so beispielsweise standardmäßig eine schrittweise Anpassung um den Wert 1 vorgenommen, beim Rotate Tool erfolgt eine Veränderung um jeweils 15 Grad. Die Increment-Werte können in einem Einstellungsmenü, das du in Abschnitt 3.3.5, »Weitere Tools der Scene View«, kennenlernst, beliebig verändert werden.
- *Vertex Snapping*: Erlaubt das Bearbeiten eines Objekts ausgehend von einem Eckpunkt. Wird die Taste V gedrückt gehalten, während das Move Tool aktiviert ist, springt das Symbol des jeweils ausgewählten Transform-Werkzeugs zum nächstgelegenen Eckpunkt des Objekts. Beim Würfel wird so beispielsweise die jeweils nächste Ecke ausgewählt. Durch das Bewegen der Maus können weitere Eckpunkte angesteuert werden.
- *Surface Snapping*: Damit können Objekte einfach neben- oder übereinander angeordnet werden, indem das Move Tool zu Überschneidungen mit anderen Objekten springt. Dazu müssen die Tasten Strg und ⇧ gedrückt gehalten werden. In der Mitte des Move Tools erscheint ein kleines Viereck, das umhergezogen werden kann. Das Objekt wird dabei an Oberflächen anderer Objekte ausgerichtet. Wichtig: Diese Technik funktioniert nur dann, wenn beide Objekte (daher das umhergezogene Objekt und das Objekt zur Ausrichtung) eine Collider-Component besitzen. Die 3D-Grundobjekte wie unser alter Freund, der Würfel, besitzen eine solche Component.

- *Grid Snapping*: Mit dieser Form wird das gewählte Objekt entlang eines virtuellen Rasters in der Scene ausgerichtet. Das Objekt springt dabei entlang der festgelegten Punkte des Rasters. Das Aktivieren des Grid Snappings wird genauer bei der Erläuterung der Tools der Scene View in Abschnitt 3.3.5 thematisiert.

Durch das parallele Drücken mehrerer Snapping-Tasten können die Arten kombiniert werden. Wird beispielsweise die Taste V parallel zur Taste Strg genutzt, wird der jeweilige Vertexpunkt durch das Increment Snapping verändert. Unity gibt dir viele praktische Möglichkeiten – und du solltest sie nutzen!

3.3.3 Übung 03.02: Tischlermeister

Mithilfe dieser Übung sollst du mit der Nutzung der Transform Toolbar warm werden. Deshalb besteht deine Aufgabe darin, verschiedene Möbel nachzubauen. Nutze dafür einfache 3D-Objekte, die dir Unity bereitstellt (wie zum Beispiel den altbekannten Würfel oder andere Objekte deiner Wahl). Lasse deiner Kreativität freien Lauf und baue die folgenden Möbel:

- Tisch (leicht)
- Stuhl (mittel)
- Schrank (mittel)
- Schubladenfach mit einer offenen Schublade (schwer)

3.3.4 Gizmos

Bestimmt ist dir beim Herumbewegen innerhalb der Scene View bereits aufgefallen, dass bestimmte Game Objects mit Symbolen gekennzeichnet werden, die im eigentlichen Spiel nicht zu sehen sind. Solche Symbole und Orientierungshilfen werden als *Gizmo* bezeichnet.

In einer neu generierten Scene sollten das Symbol einer Glühbirne (für die Lichtquelle des Directional Light) und einer Kamera (für die Main Camera) zu sehen sein. Du hast außerdem bereits gesehen, dass Game Objects beim Selektieren farblich hervorgehoben werden. Unity versucht damit, die Arbeit mit der Scene weiter zu vereinfachen. Gizmos können einzeln oder insgesamt deaktiviert werden, wenn sie dich stören.

3.3.5 Weitere Tools der Scene View

Zum Abschluss widmen wir uns den zahlreichen restlichen Tools, die in der Scene View enthalten sind (siehe Abbildung 3.39):

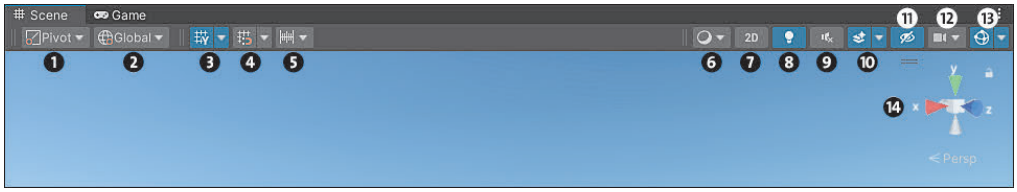


Abbildung 3.39 Die Toolleiste der Scene View

❶ **Gizmo-Positionsmenü**: Stellt ein, wo das Gizmo zur Bewegung eines Game Objects mit dem Move Tool erscheint. Mit der Einstellung PIVOT erscheint das Gizmo am eigentlichen Bewegungspunkt der Transform-Component. Über die Einstellung CENTER wird der Mittelpunkt aller Game Objects gewählt, die selektiert sind. Ist nur ein Game Object ausgewählt, ist egal, welche der beiden Einstellungen gewählt wurde.

❷ **Gizmo-Rotationsmenü**: Wird analog genutzt, um anzuzeigen, woran sich das Transform Rotation Tool ausrichtet. Mit der Einstellung LOCAL wird die Rotationsanzeige einheitlich mit Game Object geneigt, wenn dieses rotiert wird. Ist GLOBAL gewählt, bleibt das Rotations-Gizmo immer gleich ausgerichtet.

❸ **Grid-Button**: Kann das dünne Raster, das in der Scene zu sehen ist, ein- oder ausblenden. Über den kleinen Pfeil kann das Raster zudem auf eine bestimmte Ebene gesetzt, transparenter angezeigt oder verschoben werden.

❹ **Grid-Snapping-Button**: Aktiviert oder deaktiviert das Einrasten der Position entlang des Scene-Rasters, wenn das Move Tool verwendet wird. Die Option zur Ausrichtung entlang des Grids ist nur dann gestattet, wenn im Gizmo-Rotationsmenü die Einstellung GLOBAL gewählt ist. Über den Pfeil an der Seite kann das aktuell gewählte Objekt zudem an bestimmten Achsen des Rasters ausgerichtet werden.

❺ **Snap-Increment-Menü**: Bezieht sich auf das Increment Snapping, das, wie in Abschnitt 3.3.2, »Die Transform Toolbar«, erläutert, mit Strg genutzt wird. Für alle Hauptfunktionen – Move Tool, Rotation Tool und Scale Tool – kann im Menü der Abstand für das Snapping festgelegt werden.

❻ **Draw-Mode-Menü**: Hier kannst du konfigurieren, auf welche Weise die Scene in der Scene View dargestellt wird. So ist es beispielsweise möglich, lediglich die Umrisse von Objekten darzustellen.

❼ **2D-Button**: Erlaubt das Umschalten in eine 2D-Ansicht der Scene. Diese ist besonders beim Bauen von 2D-Spielen praktisch. Gleichzeitig kann die Ansicht gut in 3D-Spielen genutzt werden, wenn zweidimensionale Elemente, wie eine Benutzeroberfläche, angefertigt werden.

❽ **Licht-Button**: Aktiviert oder deaktiviert alle Lichtquellen in der Scene View.

❾ **Lautsprecher-Button**: Aktiviert oder deaktiviert Audioeffekte, die die Scene View beeinflussen.

❿ **Effekt-Button**: Über diesen Button können bestimmte visuelle Effekte in der Scene View aktiviert oder deaktiviert werden. Über den Pfeil an der Seite kann zudem genau eingestellt werden, welche Effekte in der Scene View sichtbar sein sollen. Dazu zählen zum Beispiel ein virtueller Himmel, Nebel oder Partikeleffekte.

⓫ **Augen-Button**: Hier können bestimmte Game Objects ein- oder ausgeblendet werden. Um ein Objekt für das Ausblenden freizugeben, muss in der HIERARCHY durch einen Klick mit der linken Maustaste neben dem Namen des Game Objects das Auge aktiviert werden (siehe Abbildung 3.40).

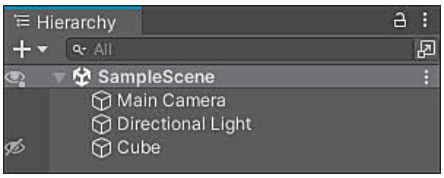


Abbildung 3.40 Ein Würfel, der in der Hierarchy als ausgeblendet markiert wurde

⓬ **Scene-Camera-Button**: Erlaubt das Konfigurieren diverser Einstellungen für die Kamera, durch die die virtuelle Welt in der Scene View zu sehen ist. Diese Einstellungen haben nichts mit anderen Kameras zu tun, die im Spiel eingesetzt werden.

⓭ **Gizmo-Button**: Zeigt oder versteckt alle Gizmos in der Scene View. Über den Pfeil an der Seite kann zudem genau eingestellt werden, welche Gizmos zu sehen sein sollen. Auch können verschiedene Symbole oder Anzeigeeinstellungen für Gizmos der Scene View getroffen werden.

⓮ **Scene View Orientation**: Zeigt die drei Achsen X (rot), Y (grün) und Z (blau). Durch einen Klick auf eines der verschiedenen Achsensymbole schwenkt die Kamera der Scene View auf exakt diese Achse. Das ist besonders praktisch, um das Spiel oder ein bestimmtes Objekt aus einer genauen Richtung zu begutachten – so kann mit einem Klick auf die Y-Achse beispielsweise in die Vogelperspektive gewechselt werden. Über einen Klick auf den mittleren Würfel des Buttons oder die Schrift unter den Achsen kann die Ansicht zudem zwischen PERSPECTIVE und ISOMETRIC umgeschaltet werden. Die perspektivische Ansicht ist so, wie du es in Abschnitt 3.2.7, »Licht und Kamera – die Standardobjekte der 3D-Scene«, für die Kameraoption PERSPECTIVE kennengelernt hast: Objekte sind kleiner, wenn sie weiter von der aktuellen Position in der Scene View entfernt sind. Die isometrische Ansicht zeigt alle Objekte unabhängig von ihrer Position in ihrer Skalierung an, so wie bei der Wahl von ORTHOGRAPHIC in der PROJECTION-Einstellung der Kamera.

3.4 Parenting

Bisher haben alle Objekte, die du in der Scene angeordnet hast, losgelöst voneinander existiert. Oft ist das aber nicht praktisch, da bestimmte Objekte zusammengehören. Denk über folgendes Beispiel nach: Du hast einen einfachen Spielcharakter, der aus einem Würfel (dem Körper) und einer Kugel (dem Kopf) besteht. Nach deinem bisherigen Wissen würden beide Objekte lose in der Scene liegen. Wenn du den Kopf beispielsweise nach rechts bewegst, bleibt der Körper einfach stehen. Genauso bleibt der Kopf an Ort und Stelle, wenn der Körper verschoben wird. Gruselig! Dieser Aufbau wird in Abbildung 3.41 dargestellt.

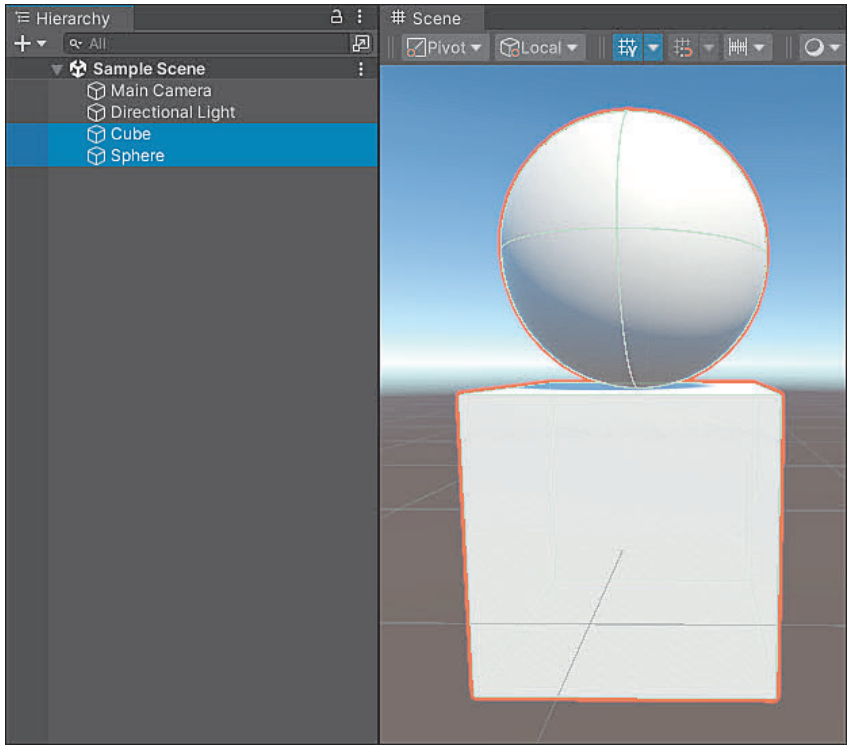


Abbildung 3.41 Der Beispielcharakter aus zwei getrennten Game Objects.

Natürlich möchtest du, dass sich beide Körperteile in einer gewissen »Beziehung« zueinander bewegen. Wenn sich der Körper bewegt, dann soll sich der Kopf natürlich mit bewegen – schließlich sind bei gesunden Menschen beide Elemente verbunden. Der Kopf hingegen soll sich unabhängig zum Körper bewegen und drehen können. Wenn du nickst, bleibt dein Körper im übertragenden Sinn immerhin vollkommen regungslos. Unity ermöglicht uns das Anlegen solcher Beziehungen über das sogenannte *Parenting*.

3.4.1 Grundidee des Parentings

Das englische Wort *parent* kann mit »Elternteil« übersetzt werden. Leider ist unser technisches Programm Unity nicht romantisch genug, um eine richtige Familie zu gründen. Dafür ist der Ansatz des Parentings an die Idee geknüpft, dass sich Kinder an ihre Eltern halten, also artige Kinder!

Gehen wir noch einmal vom Einstiegsbeispiel des Spielcharakters aus, der aus einem Würfel (Körper) und einer Kugel (Kopf) besteht (siehe Abbildung 3.41). Mithilfe von Parenting wollen wir beide Objekte miteinander verbinden, sodass der Kopf am Körper angebracht wird. Dafür soll der Körper als Parent für den Kopf gesetzt werden. Klicke dafür in der HIERARCHY die Kugel mit der linken Maustaste an und ziehe sie per Drag & Drop auf den Würfel. Dadurch sollte das Game Object der Kugel in der HIERARCHY eingerückt unter dem Würfel stehen. Außerdem werden beide Objekte markiert, wenn du nur den Würfel auswählst (siehe Abbildung 3.42).

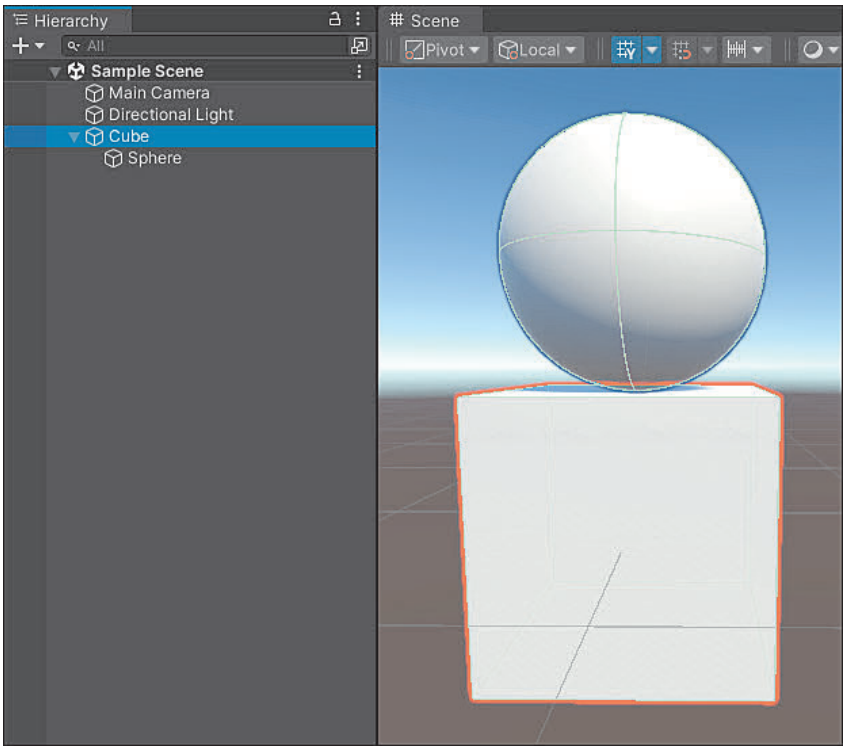


Abbildung 3.42 Die Kugel liegt in der Hierarchy unter dem Würfel.

Damit haben wir das eigentliche Hexenwerk vollbracht: Der Würfel (der Körper) ist nun ein *Parent (Elternteil)* der Kugel (des Kopfes). Umgedreht heißt das, dass die Kugel ein *Child (Kind)* des Würfels ist. Sehen wir uns an, was uns das Parenting bringt.

Wenn das Elternobjekt, der Würfel, verschoben wird, so wird das Kindobjekt, die Kugel, automatisch mit verschoben. So, als würde sie am Körper kleben! Genauso verhält es sich mit der Rotation und der Skalierung: Alle Änderungen, die am Elternobjekt des Würfels vorgenommen werden, werden automatisch auf das Kindobjekt übertragen. Andersherum ist es nicht so: Wenn die Transform-Eigenschaften auf dem Kindobjekt verändert werden, so bleibt das Elternteil völlig unbeeindruckt.

Wenn ein Game Object deaktiviert wird, dann werden alle Kindobjekte »versteckt« und funktionieren nicht länger. Das Elternobjekt und alle Kindobjekte können jederzeit wieder aktiviert werden, indem der Haken beim Elternobjekt im INSPECTOR wieder gesetzt wird.

3.4.2 Relativer Bezug zum Elternteil

Mit dem Parenting kommt ein weiterer wichtiger Aspekt, der dich am Anfang verwirren mag. Bleiben wir beim Beispiel aus Abbildung 3.42 und nehmen an, dass sich der Würfel an der Koordinate (0, 0, 0) befindet. Der Kopf liegt weiter oben, damit es so aussieht als würde er auf dem Körper sitzen, seine Position ist (0, 1, 0). Stell dir nun vor, dass wir den Körper, also das Elternteil, um eine Einheit nach rechts verschieben, sodass der Körper am Koordinatenpunkt (1, 0, 0) ist. Welche Koordinate hat nun der Kopf?

Da du weißt, dass sich der Kopf mit dem Körper bewegt, könntest du denken, dass sich die Position der Kugel mit dem Körper verändert hat und die neue Koordinate des Kopfes somit (1, 1, 0) ist. Beim Anklicken der Kugel wirst du dich aber wundern, denn wie in Abbildung 3.43 zu sehen, ist die Koordinate immer noch an derselben Position wie vor der Verschiebung (0, 1, 0).

Dieses Phänomen lässt sich natürlich erklären. Grund ist, dass Kindobjekte einen *relativen Bezug* zu ihrem Elternteil behalten. Das bedeutet, dass die Werte der Transform-Component beim Kind so angegeben werden, wie sie sich vom Elternteil unterscheiden. Bei der Position kannst du dir das so denken, als wäre das Elternobjekt der Koordinatenursprung für den Sphere-Kopf. Daher bleibt die Position weiterhin bei (0, 1, 0), was mit »vom Würfel ausgehend eine Einheit nach oben« übersetzt werden kann. Im Gegensatz dazu spricht man bei einer Einordnung in den Gesamtkontext der Scene von einem *absoluten Bezug*. Da der Würfel kein Elternobjekt besitzt, nutzt er permanent den absoluten Bezug des Scene-Koordinatensystems.

[i]

Relative und absolute Werte

Gleichermaßen kann man von *relativen und absoluten Werten* sprechen. Der Würfel besitzt vor dem Verschieben den absoluten Positionswert (0, 0, 0) und danach (1, 0, 0). Die relativen Werte des Würfels sind identisch, da er kein Elternobjekt besitzt. Die

Kugel dagegen besitzt vor und nach dem Verschieben die relative Position (0, 1, 0). Die absolute Position des Kopfes – und damit sein Bezug zum Koordinatenursprung (0, 0, 0) – verändert sich dagegen von (0, 1, 0) auf die Endposition (1, 1, 0). Nimm dir ruhig etwas Zeit, um diesen Unterschied zu verinnerlichen, er wird dich bei der Spieleentwicklung mit Unity begleiten!

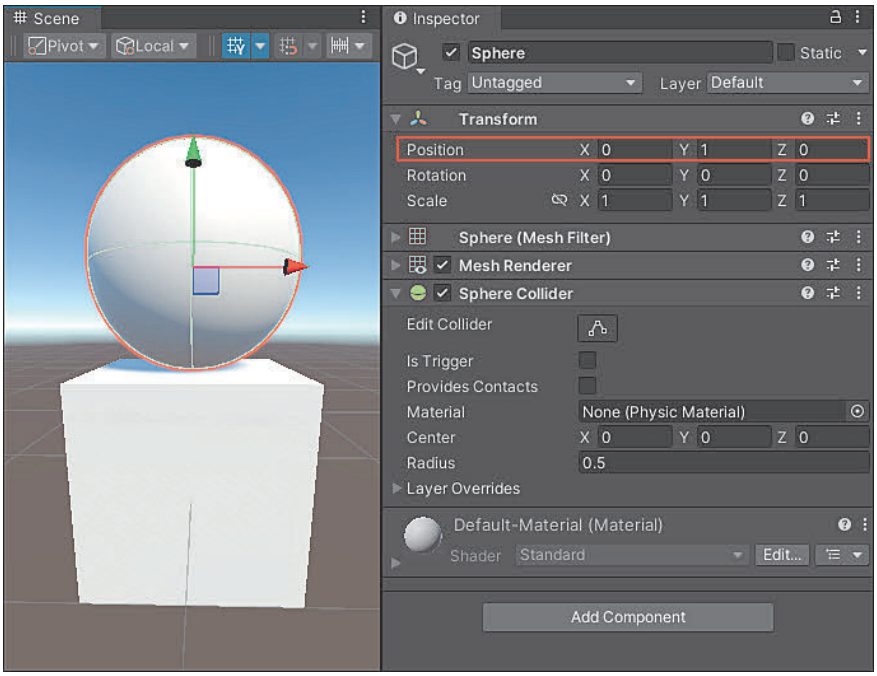


Abbildung 3.43 Die Koordinate des Kindobjekts hat sich nicht verändert, obwohl das Elternobjekt in X-Richtung verschoben wurde.

Auch wenn die bisherigen Informationen zu relativem und absolutem Bezug mit der Position erläutert wurden, gelten die gleichen Regeln für die Rotation und die Skalierung. Wird der Körper rotiert, wird der Kopf ebenfalls mit Bezug zum Elternobjekt rotiert, die relativen Werte verbleiben dabei dennoch unverändert. Genauso wird der Kopf mit dem Körper skaliert, wenn die Scale-Werte angepasst werden. Dennoch bleiben die Eigenschaften beim Objekt selbst auf dem Standard-Scale-Wert (1, 1, 1).

3.4.3 Verschachtelte Hierarchien und mehrere Kinder

Im echten Leben sind die meisten Eltern bereits mit einem Kind glücklich, doch Unity geht hier einen Schritt weiter. Die Parenting-Struktur ist nicht nur auf ein Kind limitiert. So ist Folgendes möglich:

- Ein Elternobjekt besitzt mehrere Kindobjekte.
- Ein Objekt, das bereits Kind eines Objekts ist, besitzt ebenfalls ein oder mehrere Kinder.

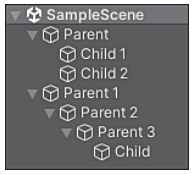


Abbildung 3.44 Beispiel für ein Elternobjekt mit mehreren Kindern (oben) und eine verschachtelte Parent-Hierarchie (unten)

In anderen Worten besitzt jedes Game Object maximal ein Elternteil – es ist nicht möglich, dass ein Game Object zwei Parents besitzt. Gleichzeitig darf es beliebig viele Kindobjekte besitzen. Ein Beispiel für verschiedene Eltern-Kind-Beziehungen in der Hierarchy ist in Abbildung 3.44 zu sehen.

Ein Beispiel für ein Elternteil, das mehrere Kinder besitzt, ist eine Hand mit fünf Fingern. Wenn sich die Hand bewegt, sollen sich alle fünf Finger mit bewegen. Dennoch funktioniert jeder Finger für sich allein: Wird ein Finger geneigt, interessiert das die eigentliche Hand und die anderen Finger nicht. Für eine verschachtelte Hierarchie eignet sich das Beispiel des Körpers ebenfalls gut. So könnten nacheinander Objekte angeordnet werden, die in einer Beziehung stehen: Oberkörper, Arm, Hand. Auch an diesem Beispiel wird deutlich, welchen Vorteil das Parenting hat.

3.4.4 Empty Parents – auch leere Eltern sind gute Eltern

Eine besonders praktische Anwendung findet das Parenting beim Bauen einer Scene, um verschiedene Objekte in der Scene zu gruppieren. Stell dir vor, du hast einen Wald, der aus 20 Bäumen besteht. Wenn alle Objekte, die die Scene dekorieren, einzeln herumliegen, wird die Hierarchy schnell unübersichtlich. Deshalb kann Parenting gut genutzt werden, um die Vielzahl an bestimmten Objekten unter einem Parent-Objekt zusammenzufassen, das selbst nur ein leeres Game Object ist. In diesem Fall ist ein leeres Game Object plötzlich doch gar nicht mehr so sinnlos!

Wie du es in Abschnitt 3.2.4, »Das Game Object als leerer Behälter«, gelernt hast, könntest du ein leeres Game Object erstellen und diesem sämtliche Kindobjekte manuell zuweisen. Unity bietet allerdings einen einfacheren Weg an. Zunächst werden alle Objekte, die einem neuen Parent-Objekt zugewiesen werden sollen, ausgewählt. Das geht über einzelnes Anklicken, während die Taste Strg gedrückt gehalten wird, oder indem das erste und letzte Objekt in der Reihe mit ⇧ angeklickt werden. Anschließend kann im Menü über einen Klick mit der rechten Maustaste die Option Create Empty Parent gewählt werden. Alle selektierten Objekte werden dann unter ein

neues Parent-Objekt gruppiert, dem ein neuer Name gegeben werden kann. Der Vorgang ist in Abbildung 3.45 dargestellt.

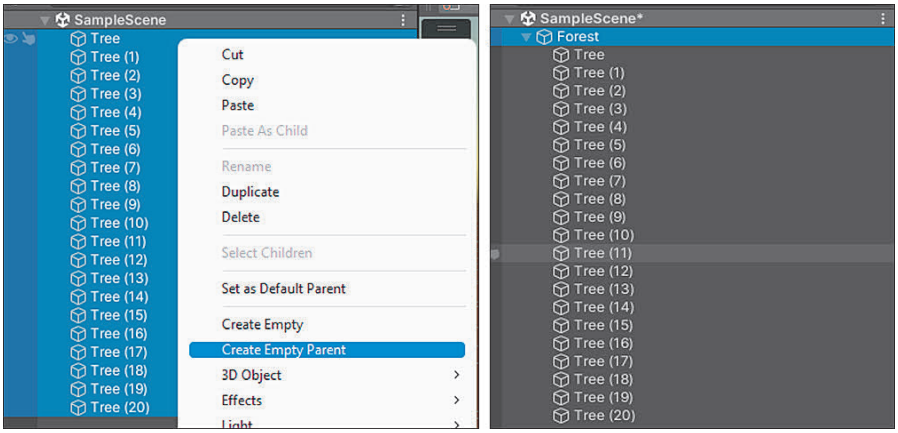


Abbildung 3.45 Erstellen eines leeren Elternteils (links) und Ansicht der Hierarchie nach dem Erstellen (rechts)

Über den kleinen Pfeil an der Seite des Elternobjekts FOREST könnten nun alle Kindobjekte eingeklappt werden. Die Hierarchy bleibt so wunderbar übersichtlich. Das Parenting hat über die Übersichtlichkeit und die relativen Bezüge der Transform-Component hinaus viele Vorteile, die du im Laufe des Buches kennenlernen wirst.

3.4.5 Übung 03.03: Frankensteins Monster

Deine nächste Aufgabe ist mindestens deiner Mutter geglückt: Du sollst einen eigenen Menschen bauen! Natürlich findet deine Aufgabe in Unity statt. Nutze dafür die 3D-Grundobjekte, die Unity anbietet. Achte darauf, sinnvolles und gut nutzbares Parenting anzuwenden. Dein Mensch soll mindestens die folgenden Elemente enthalten:

- Körper
- Arme und Beine
- Hände
- Kopf
- einen beliebigen Gegenstand (zum Beispiel einen Ball oder einen Zauberstab) in einer Hand

3.5 Musterlösungen für die Übungen

Bei allen Übungen in diesem Kapitel kann deine kreative Lösung stark von den gezeigten Musterlösungen abweichen. Für die genauen Werte, die in der Transform-

Component eingestellt wurden, kannst du die jeweiligen Beispiel-Scenes im Ordner *Kapitel 03* in den Buchdateien öffnen.

3.5.1 Lösung für Übung 03.01: Objektumformung

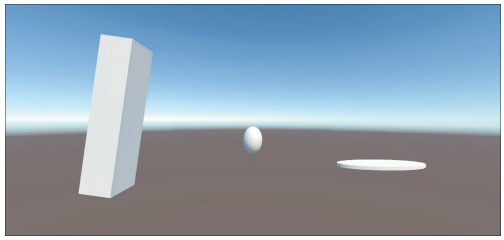


Abbildung 3.46 Musterlösung für Übung 03.01

3.5.2 Lösung für Übung 03.02: Tischlermeister

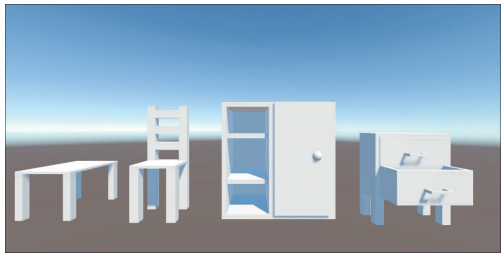


Abbildung 3.47 Musterlösung für Übung 03.02

3.5.3 Lösung für Übung 03.03: Frankensteins Monster

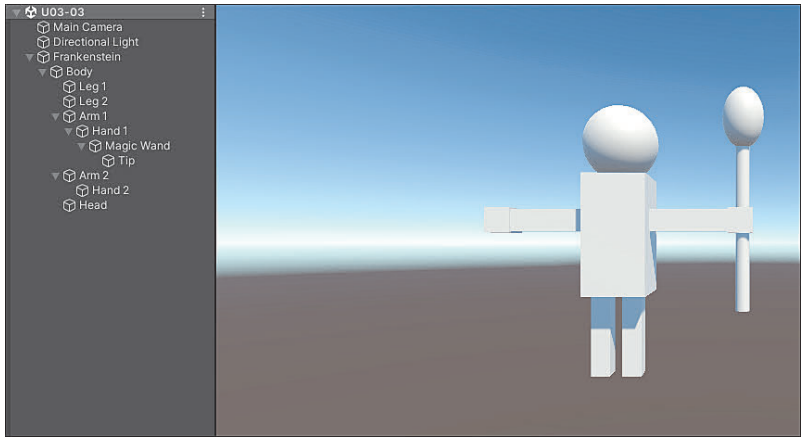


Abbildung 3.48 Musterlösung für ein … etwas kantiges Monster mit Zauberstab für Übung 03.03

Kapitel 17
Sound

Wurdest du jemals wegen Lärmbelästigung angezeigt? Ich hoffe nicht. Vielleicht denkst du sonst, dass Sound nicht gut ist – doch genau das Gegenteil ist der Fall!

Geräusche für Spiele können vielseitig sein: Von einer seichten Hintergrundmusik über eine Truhe, die sich knarzend öffnet, bis hin zur Fanfare bei einem Level-up können Geräusche in vielen Teilen deines Spiels eine Rolle spielen. Zwar sind die meisten Spiele auch ohne Geräusche spielbar, dennoch tragen Sounds stark dazu bei, dass man sich mehr in das Spiel hineinversetzen kann. Vielleicht möchtest du Sound sogar als Eingabe für dein Spiel nutzen: Stell dir zum Beispiel ein Game vor, in dem sich dein Charakter nur bewegt, wenn du in das Mikro schreist. Deine Nachbarn hätten definitiv weniger Spaß als du. Dieses Kapitel bringt dir alle wichtigen Grundlagen bei, die Unity zur Ein- und Ausgabe von Sound bereitstellt.

17.1 Grundlagen von Sound

Zunächst sehen wir uns die wichtigsten Grundlagen zu Sound an, indem wir Sounddateien in ein Unity-Projekt importieren und mithilfe von Components abspielen.

17.1.1 Importieren von Audioclips

Der erste Schritt zum Verwenden von Sounds besteht selbstverständlich darin, Sounddateien in das Projekt zu importieren. Unity unterstützt viele gängige Soundformate, wie zum Beispiel MP3, Ogg, WAV oder AIFF. Um den Import näher anzusehen, importieren wir eine Beispieldatei in das Projekt. Diese enthält ein kurzes »Blob«-Geräusch. Die Datei liegt dabei im Ogg-Format vor.

Material

Datei: *Kapitel 17/blob.ogg*

Quelle: *https://www.kenney.nl/assets/digital-audio*

Ein Sound, der in das Projekt geladen wurde, wird als *Audioclip* bezeichnet. Wenn du den Audioclip im Project Window auswählst, kannst du im INSPECTOR einige Einstellungen konfigurieren (siehe Abbildung 17.1).

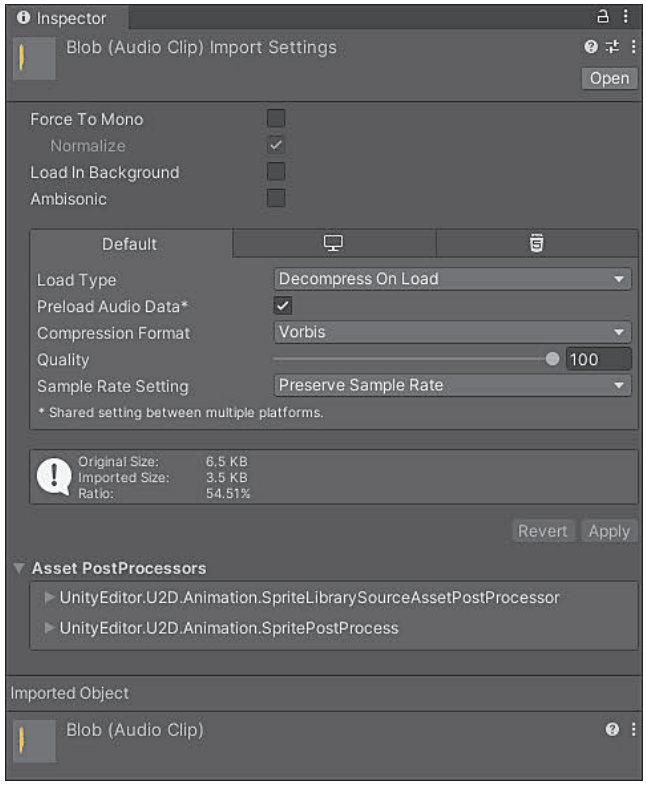


Abbildung 17.1 Der Audioclip kann über den Inspector angepasst werden.

Die Einstellungen, die du im Inspector setzen kannst, umfassen:

- FORCE TO MONO: Wenn diese Eigenschaft aktiviert wird, werden Geräusche, die auf mehreren Kanälen abgemischt wurden, vor dem Abspielen auf lediglich einen Kanal gesetzt.
- NORMALIZE: Beim Aktivieren dieser Eigenschaft wird beim Abmischen des Sounds auf Mono eine Normalisierung vorgenommen.
- LOAD IN BACKGROUND: Wenn diese Option aktiviert ist, wird die Sounddatei beim Spielen im Hintergrund geladen. Somit wird die eigentliche Ausführung des Spiels nicht blockiert.
- AMBISONIC: Diese Eigenschaft wird genutzt, wenn das Geräusch ein Ambisonic-Sound ist. Diese Art von Sounds wird je nach Orientierung des Hörers gedreht, sodass sie besonders für 360-Grad-Anwendungen nützlich sind.
- LOAD TYPE: Gibt die Art an, mit der ein Sound im Spiel geladen wird.

- Preload Audio Data: Wenn diese Option nicht aktiv ist, wird der Clip erst geladen, wenn er benötigt wird oder ein expliziter Scriptaufruf zum Laden gegeben wird.
- Compression Format: Gibt an, in welchem Format der Audioclip komprimiert wird.
- Quality: Gibt an, wie stark der Clip komprimiert wird. Der Standardwert 100 steht für die bestmögliche Qualität des Sounds.
- Sample Rate Setting: Einstellung für die verwendete Sample Rate, die für den Clip verwendet wird

Im Normalfall sind die Standardeinstellungen vollkommend ausreichend für eine gute Konfiguration. Nachdem der Clip im Projekt ist, kommen wir nun zum spannenderen Teil: Wir wollen den Clip in unserem Spiel hören!

17.1.2 Audio Listener

Wie soll man Geräusche ohne virtuelle Ohren hören? Genau aus diesem Grund stellt Unity eine Component bereit, die als »Ohr« für dein Spiel fungiert. Die Component trägt den Namen Audio Listener und nimmt Geräusche innerhalb der Scene auf und gibt diese an die Kopfhörer oder Lautsprecher des Spielers weiter.

Eventuell erinnerst du dich dunkel, dass standardmäßig am Game Object Main Camera eine solche Component platziert wird. Das ist eine sinnvolle Lösung – immerhin wird auf diesem Weg simuliert, dass die Ohren gleichzeitig bei den Augen liegen. Aus diesem Grund musst du die Component im Normalfall nicht separat einrichten. Wenn es für das Spiel sinnvoll ist, kann die Component auch einem anderen Objekt, wie zum Beispiel dem Spielcharakter, angefügt werden. Wichtig ist lediglich, dass nur ein Audio Listener pro Scene eingesetzt wird. Immerhin hast du auch nur ein Paar Ohren am Kopf! Die Component hat keine Einstellungsoptionen: Sobald sie in der Scene liegt, brauchst du dich um nichts mehr zu kümmern, damit Geräusche hörbar werden.

17.1.3 Audio Source

Es fehlt nur noch ein letztes Puzzleteil, um Geräusche im Spiel hören zu können: die eigentliche Quelle eines Geräuschs, die importierte Audioclips abspielt. Die einfachste Einrichtung dafür ist es, einen Audioclip aus dem Project Window in die Hierarchy oder Scene zu ziehen. Dabei wird ein Game Object generiert, das den Namen des jeweiligen Clips trägt. Auf dem Objekt wird zusätzlich eine Component mit dem Namen Audio Source hinzugefügt (siehe Abbildung 17.2). Rund um die Audio Source wird eine blaue Kugel angezeigt. Die Bedeutung dieser Kugel wird dir im Laufe des Abschnitts klarer.

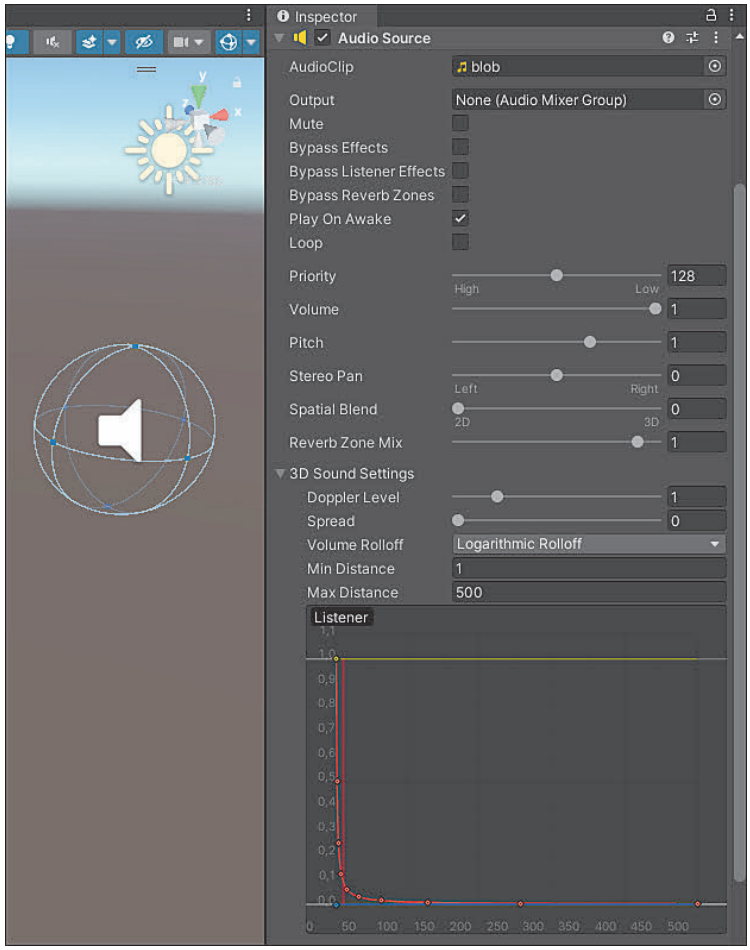


Abbildung 17.2 Ansicht nach dem Anlegen einer neuen Audio Source

Bevor wir uns die Eigenschaften der Component ansehen, wollen wir zunächst einen Test wagen. Wenn du das Spiel startest, solltest du sofort beim Starten das importierte »Blob«-Geräusch hören.

[i]

Wenn du kein Geräusch hören kannst

Wenn du das Geräusch nach Ziehen des Audioclips in die Scene nicht hörst, hat das wahrscheinlich einen der folgenden Gründe:

- Du hast keinen Audio Listener in der Scene. Stelle sicher, dass du die Component zum Beispiel der Main Camera anfügst.
- Du hast den Sound im Play Mode deaktiviert. Klicke im oberen Teil der Spielansicht auf das Lautsprechersymbol (dieses sollte danach blau sein) und starte das Spiel neu.

- Du hast Einstellungen an der Audio Source verändert, sodass der Sound nicht mehr hörbar ist. Setze den Clip dann am besten noch einmal neu in die Scene.

Wie genau das Geräusch zu hören ist, ist von einer Reihe an Einstellungen abhängig, die du für eine Audio Source festlegen kannst:

- AudioClip: Gibt den Clip an, der von der Audio Source abgespielt wird. Nach dem Ziehen eines Clips in die Scene wird diese Eigenschaft automatisch gesetzt.
- Output: Hier kannst du optional einen Audio Mixer (siehe Abschnitt 17.3.3) angeben, an den das Geräusch der Audio Source weitergegeben wird. Egal, ob diese Eigenschaft zugewiesen wird oder nicht, wird das Geräusch auch auf dem Audio Listener zu hören sein.
- Mute: Wenn diese Eigenschaft aktiviert ist, wird der Sound der Audio Source stumm geschaltet.
- Bypass Effects: Wenn diese Option aktiviert wird, werden alle Effekte deaktiviert, die auf die Audio Source angewendet werden.
- Bypass Listener Effects: Wenn diese Option aktiviert wird, werden alle Listener-Effekte deaktiviert, die auf die Audio Source angewendet werden.
- Bypass Reverb Zones: Wenn diese Option aktiviert wird, werden alle Effekte von Reverb Zones (siehe Abschnitt 17.3.5) deaktiviert, die auf die Audio Source angewendet werden.
- Play On Awake: Gibt an, ob die Audio Source beim Spielstart abgespielt werden soll. Wenn diese Eigenschaft deaktiviert ist, muss die Source manuell an einer geeigneten Stelle abgespielt werden.
- Loop: Gibt an, ob die Audio Source nach dem Ende des Clips neu starten soll (ob der Clip in einer Endlosschleife weiterläuft).
- Priority: Gibt die Priorität dieser Audio Source innerhalb der Scene an. Ein Wert von 0 meint dabei am wichtigsten, 256 am unwichtigsten. Beispielsweise sollte Hintergrundmusik einen Wert von 0 erhalten, um nicht mit anderen Soundeffekten »zu kollidieren«.
- Volume: Gibt die Lautstärke des Sounds an, wenn der Audio Listener unter Verwendung des 3D-Raumes eine Einheit neben der Audio Source platziert ist.
- Pitch: Setzt die Frequenz des Geräuschs. Somit kann der Sound höher oder tiefer klingen und schneller oder langsamer abgespielt werden.
- Stereo Pan: Setzt die Position eines 2D-Stereo-Pan-Feldes tendenziell nach links oder rechts. Mit dieser Einstellung kannst du beeinflussen, dass sich ein Sound mehr danach anhört, als würde er von rechts oder links kommen – und das beispielsweise unabhängig davon, wo er im 3D-Raum abgespielt wird. Diesen Effekt kannst du besonders auf Kopfhörern gut nachvollziehen.

- Spatial Blend: Gibt an, ob der Sound das 3D-System der Engine verwendet. Unter Verwendung der Standardeinstellung 2D ist der Sound immer zu hören, egal, wie weit die Audio Source vom Audio Listener entfernt ist. Diese Eigenschaft werden wir uns gleich näher ansehen.
- Reverb Zone Mix: Gibt an, wie der Sound unter der Nutzung von Reverb Zones (siehe Abschnitt 17.3.5) abgemischt wird.
- 3D Sound Settings: Unter diesem Bereich können verschiedene 3D-Einstellungen für den Sound vorgenommen werden. Diese sehen wir uns gleich näher an.

Wie du nach dem Einbinden des Clips in der Scene sehen kannst, wird die Einstellung Spatial Blend standardmäßig auf 2D gestellt. Das bedeutet, dass der Sound jederzeit in der gleichen Lautstärke zu hören ist – und das völlig unabhängig davon, wo sich der Audio Listener in Relation zur Audio Source befindet. Sehen wir uns also als Nächstes an, wie eine Audio Source für den 3D-Raum eingerichtet wird. Die Grundlage für 3D-Sound wird durch den Abschnitt 3D Sound Settings innerhalb der Component Audio Source konfiguriert. Dafür stehen einige separate Einstellungen zur Verfügung:

- Doppler Level: Gibt den Dopplereffekt der 3D-Soundquelle an. Ein Wert von 0 bedeutet, dass kein Dopplereffekt verwendet wird.
- Spread: Setzt den Ausbreitungswinkel der Soundquelle. Diese Option ist für 3D-Stereosound interessant.
- Volume Rolloff: Gibt an, wie schnell der Sound im 3D-Raum verblasst. Dir stehen drei Optionen zur Auswahl:
  - Logarithmic Rolloff: An der Quelle ist der Sound laut, aber beim Entfernen von der Audio Source wird die Lautstärke schnell leiser.
  - Linear Rolloff: Die Lautstärke sinkt gleichmäßig, wenn du dich von der Quelle entfernst.
  - Custom Rolloff: Du kannst eine separate Kurve bestimmen, die angibt, wie sich die Laustärke mit der Distanz zur Quelle verhält.
- Min Distance: Innerhalb dieser Distanz ist der Sound in voller Lautstärke zu hören.
- Max Distance: Außerhalb dieser Distanz ist der Sound nicht mehr zu hören. Zwischen Min Distance und Max Distance wird der Volume Rolloff verwendet, um die Lautstärke des Sounds zu bestimmen.

Mithilfe dieser Einstellungen können wir den »Blob«-Sound auf das 3D-System umstellen. Dafür musst du den Schieberegler der Option Spatial Blend zunächst von 2D auf 3D nach ganz rechts verschieben. Anschließend legen wir eine neue Max Distance mit einem Wert von 20 fest. (siehe Abbildung 17.3). Für das Testen hat die Main Camera (und somit auch der Audio Listener der Scene) die Standardposition (0, 0, –10).

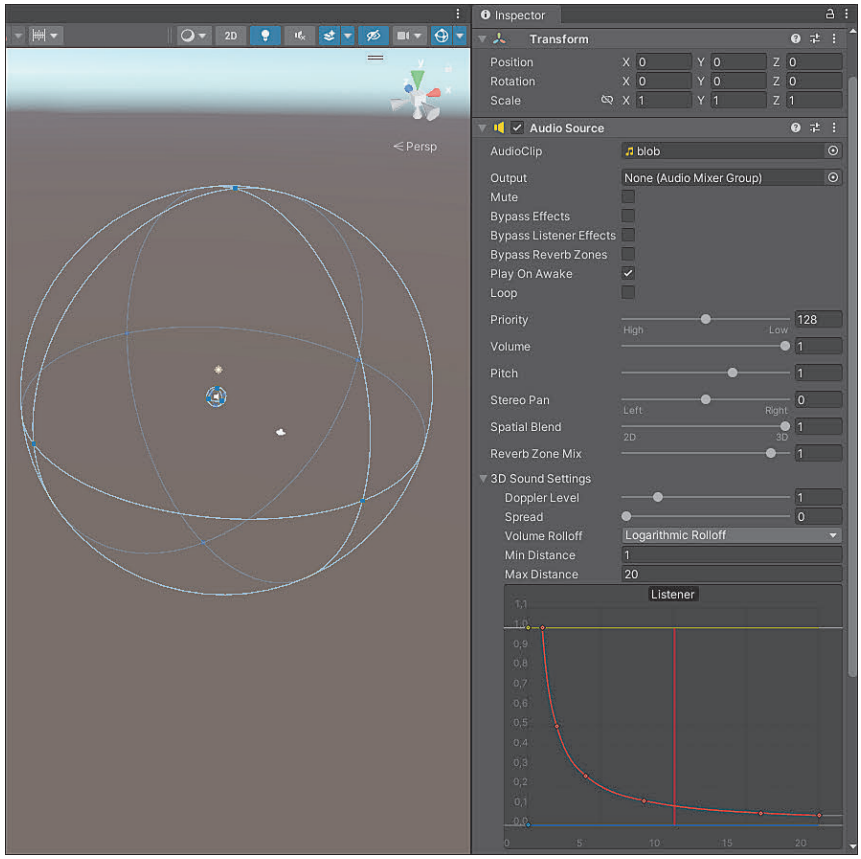


Abbildung 17.3 Einrichtung der Audio Source für den 3D-Raum

Startest du das Spiel, ist das Geräusch mit einer geringeren Lautstärke zu hören, als wenn du den Sound nicht der 3D-Umgebung anpasst. Mit dem Verändern der Distanzwerte siehst du, dass sich die Anzeige der blauen Kugel im Spiel verändert. Die innere blaue Kugel steht für die vergebene MIN DISTANCE ausgehend von der Audio Source, die äußere Kugel für die MAX DISTANCE. Die Kugeln haben zusätzlich kleine Punkte, mit denen du sie in der Scene anpassen kannst. Wenn du den Radius veränderst, werden die Distanzwerte in der Audio Source sofort angepasst.

Vielleicht ist dir außerdem aufgefallen, dass sich der Graph unterhalb der 3D-Einstellungen verändert. Er gibt die Kurve an, mit der sich die Lautstärke zwischen MIN DISTANCE und MAX DISTANCE verändert (standardmäßig wird die Lautstärke dabei natürlich leiser, daher flacht die Kurve ab). Innerhalb des Graphen kannst du die Punkte anklicken und verändern, um eine eigene Veränderungskurve für die Lautstärke zu erstellen. Außerdem werden Geraden für den Audio Listener in der Scene (vertikale Linie) sowie SPATIAL BLEND, SPREAD und REVERB ZONE MIX (horizontale Linien) angezeigt.

17.2 Sounds und Scripting

Auch, wenn viele Einstellungen im Inspector gesetzt werden können, so kommst du bei Sounds häufig nicht an Scripts vorbei. In diesem Abschnitt lernst du, welche Möglichkeiten sich durch Scripts im Zusammenspiel mit Sound bieten.

17.2.1 Die Klasse »AudioSource«

Die wichtigsten Aufgaben werden von der Klasse `AudioSource` der gleichnamigen Component übernommen. Die Eigenschaften, die von dieser Klasse bereitgestellt werden, stimmen zum größten Teil mit den Einstellungen aus dem Inspector überein. Wir wollen uns stattdessen vor allem auf die bereitgestellten Methoden konzentrieren, die mit einem Audioclip, der für die Audio Source hinterlegt wurde, interagieren können. Die wichtigsten Methoden zum Abspielen, Stoppen oder Pausieren eines Clips sind dabei:

- `void Play()`: Spielt den Clip der Audio Source ab.
- `void PlayDelayed(float delay)`: Spielt den Clip der Audio Source mit einer Verzögerung ab. Das übergebene `float` gibt die Verzögerung in Sekunden an.
- `void Stop()`: Stoppt den aktuellen Clip. Beim Stoppen kann der Clip später nicht an der gestoppten Stelle fortgesetzt werden.
- `void Pause()`: Pausiert den aktuell abgespielten Clip.
- `void UnPause()`: Setzt einen pausierten Clip fort.

Alle Bestandteile der Klasse »AudioSource«

Für eine Übersicht über alle verfügbaren Eigenschaften und Methoden kannst du die offizielle Dokumentation verwenden:

https://docs.unity3d.com/ScriptReference/AudioSource

Sehen wir uns an einem Beispiel an, wie ein Audioclip mithilfe eines Scripts pausiert und wieder abgespielt werden kann. Dafür verwenden wir einen längeren Audioclip, auf dem ein Rauschen zu hören ist.

Material

Datei: *Kapitel 17/rauschen.ogg*

Quelle: *https://kenney.nl/assets/sci-fi-sounds*

Zunächst richten wir die Audio Source ein: Nachdem der Clip in die Scene gezogen wurde, aktivieren wir die Einstellung Loop, damit das Rauschgeräusch in einer Endlosschleife abgespielt wird, wenn der Clip nicht pausiert ist. Außerdem ist wichtig,

dass die Eigenschaft Play On Awake, aktiviert ist – der Clip kann nämlich nur pausiert oder weitergespielt werden, wenn die Audio Source gestartet wurde. Die eingerichtete Audio Source ist in Abbildung 17.4 zu sehen.

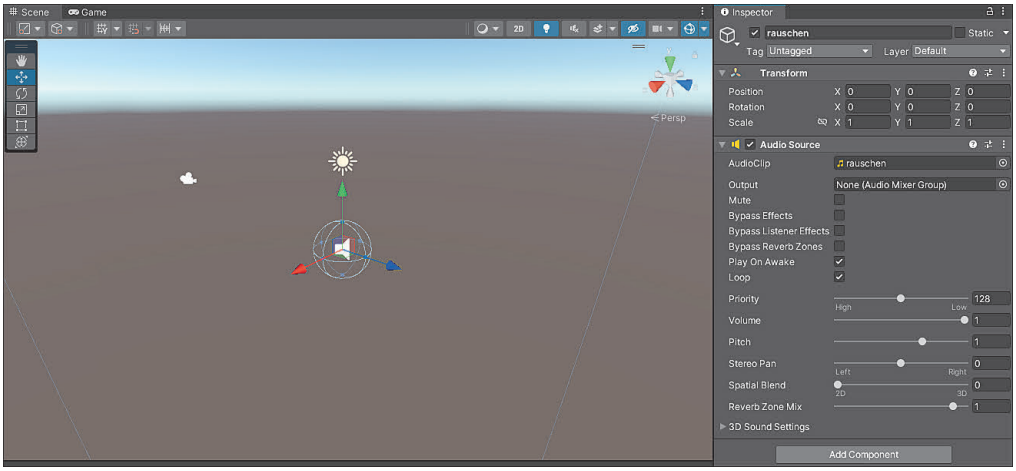


Abbildung 17.4 Die für das Rauschen eingerichtete Audio Source

Als Nächstes ist das Script für die Pause-Abspiel-Logik an der Reihe, das du in Listing 17.1 siehst. Zunächst wird die Component `AudioSource` in der `Awake`-Methode zwischengespeichert. Wir wollen das Pausieren oder Abspielen an das Drücken der Leertaste binden. Die dafür benötigte Logik kommt in die Eventmethode `Update`. Anschließend wird je nachdem, ob die Audio Source aktuell spielt, das Pausieren oder Weiterspielen über die jeweilige Methode ausgelöst. Für das Ermitteln, ob die Audio Source aktuell nicht pausiert ist, verwenden wir die Eigenschaft `isPlaying` der Component.

```
using UnityEngine;

public class AudioPauseTest : MonoBehaviour
{
  private AudioSource audioSource;

  private void Awake()
  {
    audioSource = GetComponent<AudioSource>();
  }

  private void Update()
  {
    if (!Input.GetKeyDown(KeyCode.Space)) return;
```

```
    if (audioSource.isPlaying)
    {
      audioSource.Pause();
    }
    else
    {
      audioSource.UnPause();
    }
  }
}
```

Listing 17.1 Beispiel für das Pausieren und Abspielen eines Audioclips mit der Leertaste

Das Script kannst du nun auf dem Game Object mit der Audio Source hinzufügen. Beim Starten des Spiels ist das Rauschen zu hören. Durch Drücken der Leertaste wird das Geräusch allerdings unterbrochen und nach nochmaligem Drücken wieder weiter abgespielt. So einfach kann eine Pausierungslogik aussehen!

Werfen wir außerdem noch einen Blick auf das Zuweisen eines neuen Clips zu einer Audio Source. Dafür wollen wir ein weiteres Beispiel entwickeln, mit dem beim Spielstart ein zufälliger Clip aus einer Reihe von mehreren Clips verwendet wird. Dafür importieren wir zunächst einen weiteren Clip, sodass wir zwei kurze Clips zum Testen im Projekt verwenden können.

Material

Datei: *Kapitel 17/pling.ogg*

Quelle: *https://kenney.nl/assets/interface-sounds*

Zunächst legen wir eine leere Audio Source an. Diese kannst du über GameObject • Audio • Audio Source im Menü generieren. Standardmäßig weisen wir keinen Clip zu, dafür deaktivieren wir die Einstellung Play On Awake (immerhin kann eine Audio Source erst dann, wenn ein Clip hinterlegt wurde, abgespielt werden).

Innerhalb des Scripts in Listing 17.2 stellen wir ein Array bereit, in dem die Clips aus dem Inspector heraus zugewiesen werden können. Deshalb wird das Attribut `SerializeField` verwendet. In der `Start`-Methode wird der Component Audio Source die Eigenschaft `clip` neu gesetzt – diese bestimmt den Audioclip, der von der Audio Source abgespielt wird. Mithilfe der Methode `Random.Range` weisen wir dabei ein zufälliges Element aus dem Array zu. Anschließend wird die Audio Source mit der Methode `Play` abgespielt.

```
using UnityEngine;

public class AudioSwitcher : MonoBehaviour
{
  [SerializeField] private AudioClip[] clips;

  private AudioSource audioSource;

  private void Awake()
  {
    audioSource = GetComponent<AudioSource>();
  }

  private void Start()
  {
    audioSource.clip = clips[Random.Range(0, clips.Length)];
    audioSource.Play();
  }
}
```

Listing 17.2 Beispiel für das Zuweisen eines zufälligen Audioclips bei Spielstart

Nach dem Zuweisen des Scripts musst du zunächst das Array im Inspector befüllen (siehe Abbildung 17.5). Dafür kannst du neben dem importierten Sound den Audioclip aus dem Einführungsbeispiel (siehe Abschnitt 17.1.1, »Importieren von Audioclips«) verwenden.

Abbildung 17.5 Die Audioclips werden im Inspector zugewiesen.

Wenn du das Spiel startest, ist einer der beiden zugewiesenen Clips zu hören. Das Script würde auch mit drei oder noch mehr Sounddateien funktionieren. Ein System dieser oder ähnlicher Art könntest du beispielsweise nutzen, wenn für eine bestimmte Aktion ein zufälliges Geräusch abgespielt werden soll. Ein Beispiel wäre eine Begrüßung, die ein Charakter im Spiel sagt: Anstatt immer nur »Hallo!« zu sagen, könnten zufällige Grüße wie »Moin!« oder »Hi!« erklingen. Das bringt trotz des geringen Aufwands eine Menge Abwechslung!

17.2.2 Übung 17.01: Die Neustarttaste

Schreibe ein Script, mit dem eine Audio Source neu gestartet wird, sobald eine bestimmte Taste auf der Tastatur gedrückt wird.

17.2.3 Übung 17.02: Lauter oder leiser, bitte!

Setz eine Audio Source in die Scene, die einen Clip in einer Endlosschleife abspielt. Verwende standardmäßig die Lautstärke 0,5. Sorge dafür, dass mithilfe von ↑ und ↓ die Lautstärke des Clips erhöht bzw. gesenkt wird. Achte darauf, dass die Lautstärke nicht leiser als 0 % oder lauter als 100 % wird.

Ein Tipp: Wenn du die passende Eigenschaft der Klasse `AudioSource` nicht findest, schau in der Dokumentation nach.

17.2.4 Clips unabhängig von einer Audio Source abspielen

Stell dir folgendes Szenario vor: In deinem Spiel gibt es Monster, die erlegt werden. Beim Töten eines Monsters soll dieses ein Geräusch von sich geben (ein leise sterbendes Monster wäre schließlich nur halb so interessant). Würde die Audio Source an dem Monster liegen, kämen wir an ein Problem: In unserem Spiel verschwinden Monster sofort, wenn sie sterben, deshalb wird das Game Object mitsamt der Audio Source automatisch beim Tod zerstört. Somit würde das Monster kein letztes Röcheln mehr von sich geben können. Das kannst du an dem einfachen Beispiel in Listing 17.3 austesten, bei dem du beim Drücken der Leertaste eine Audio Source über den gewohnten Weg abspielst und gleichzeitig das zugrunde liegende Game Object zerstörst.

```
using UnityEngine;

public class NoPlayClipDestroy : MonoBehaviour
{
  private AudioSource audioSource;

  private void Awake()
  {
    audioSource = GetComponent<AudioSource>();
  }

  private void Update()
  {
    if (Input.GetKeyDown(KeyCode.Space))
    {
      audioSource.Play();
```

```
      Destroy(gameObject);
    }
  }
}
```

Listing 17.3 Script zum Abspielen einer Audio Source, während das Objekt zeitgleich zerstört wird.

Wenn du das Script einem Game Object mit Audio Source anhängst und beim Spielen auf die Leertaste drückst, ist kein Geräusch zu hören. Stattdessen wird lediglich das Objekt, das die Audio Source enthält, zerstört. Du könntest zwar die Audio Source auf ein anderes Objekt auslagern, das nicht zerstört wird, und über die Verbindung zu dieser Source das Problem umgehen, allerdings ist das ziemlich aufwendig.

[i]

Umgehen des Problems

Das Beispiel ist zugegebenermaßen ein wenig konstruiert – immerhin könntest du beispielsweise auch einfach mithilfe einer Coroutine warten, bis das Geräusch fertig abgespielt wurde, und das Gegnerobjekt einfach danach zerstören. Bleiben wir der Anschaulichkeit halber aber bei dem Beispiel.

Wegen der genannten Probleme brauchen wir eine praktische Möglichkeit, mit der ein Audioclip abgespielt werden kann, ohne dass zuvor eine Audio Source in der Scene eingerichtet wurde. Für diesen Zweck wird von der Klasse `AudioSource` eine statische Methode bereitgestellt, die den Namen `PlayClipAtPoint` trägt. Der Methode wird der gewünschte Audioclip mitgegeben sowie ein Punkt genannt, an dem der Clip innerhalb der Scene abgespielt werden soll. Wir verwenden als Punkt die aktuelle Position des Game Objects. Außerdem kannst du optional ein `float` übergeben, das für die Lautstärke des abgespielten Clips steht. In Listing 17.4 nutzen wir die Methode: Wir erlauben das Übergeben eines Audioclips im Inspector. Wie in Listing 17.3 soll beim Drücken der Leertaste das Game Object zerstört werden, wobei der Audioclip dennoch im Hintergrund weiterspielen soll.

```
using UnityEngine;

public class PlayClipDestroy : MonoBehaviour
{
  [SerializeField] private AudioClip clip;

  private void Update()
  {
    if (Input.GetKeyDown(KeyCode.Space))
    {
      AudioSource.PlayClipAtPoint(clip, transform.position);
```

```
            Destroy(gameObject);
        }
    }
}
```

Listing 17.4 Abspielen eines Audioclips mit dem Zerstören des Game Objects über »PlayClipAtPoint«

Das Script kannst du nun einem Game Object in der Scene anhängen und einen beliebigen Audioclip zum Testen zuweisen. Wenn du das Spiel startest und auf die Leertaste drückst, wird das angelegte Game Object gelöscht. Gleichzeitig ist das Geräusch allerdings in der Scene zu hören. Der Grund zeigt sich, wenn du das Spiel kurz pausierst, während das Geräusch zu hören ist: Es wird automatisch ein neues Game Object angelegt, das lediglich die Component AUDIO SOURCE enthält. Der Audio Source wird der angegebene Clip zugewiesen, sodass der Sound an der übergebenen Position abgespielt wird (siehe Abbildung 17.6).

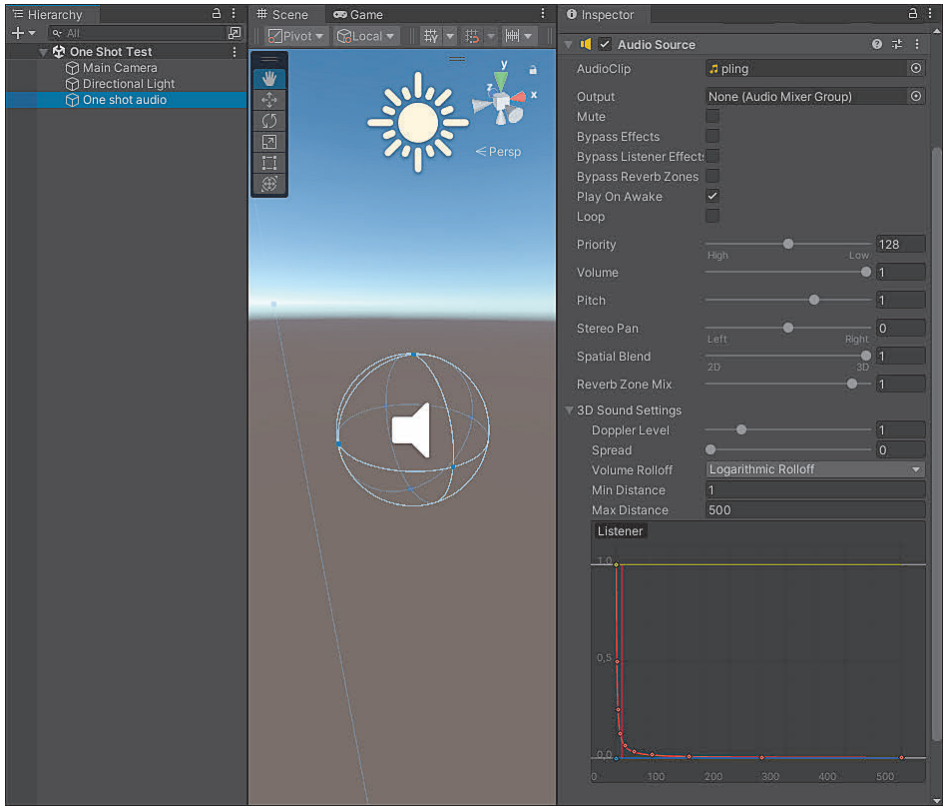


Abbildung 17.6 Durch die Methode »PlayClipAtPoint« wird ein Dummyobjekt mit eigener Audio Source angelegt.

Besonders praktisch: Nachdem das Geräusch wieder verstummt ist, wird das Dummyobjekt automatisch gelöscht. Dadurch »vermüllen« mehrere über `PlayClipAtPoint` abgespielte Geräusche deine Scene nicht. Gleichzeitig liegt der Nachteil dieser Methode auf der Hand: Dadurch, dass die Audio Source automatisch für dich angelegt wird, kannst du sie nicht ausgiebig konfigurieren. Das Nutzen von `PlayClipAtPoint` sollte deshalb eher eine Ausnahme für Szenarien bleiben, in denen sich die Methode anbietet.

17.3 Anpassung von Sounds

Nachdem du alle Basics zum Einrichten einer Audio Source sowie zum Abspielen über Scripts kennst, wollen wir uns nun den Bestandteilen der Engine widmen, mit denen Sounds im Spiel angepasst werden können.

Mehr Infos zu Audiokomponenten

Die einzelnen Einstellungen für die verschiedenen Anpassungselemente reichen teilweise tief in die Theorie von Sounds und ihrer digitalen Darstellung. Aus diesem Grund werde ich in den folgenden Abschnitten eher die generelle Funktionsweise der einzelnen Elemente vorstellen. Details kannst du der offiziellen Dokumentation entnehmen: *https://docs.unity3d.com/Manual/Audio*

17.3.1 Audiofilter

Unity stellt eine Reihe an Filtern bereit, die auf Audio Sources oder die gesamte Ausgabe angewendet werden können. Mithilfe von Filtern werden bestimmte Geräuschgruppen und Signale aus dem Ausgabeton ausgeschlossen. Um Filter anzuwenden, müssen bestimmte Components einem Game Object mit Audio Source oder Audio Listener hinzugefügt werden. Die Components, die als Filter eingesetzt werden können, sind dabei:

- **Audio Low Pass Filter**: Wendet einen Tiefpassfilter an, der hohe Frequenzen oberhalb eines angegebenen Schwellenwertes filtert.
- **Audio High Pass Filter**: Wendet einen Hochpassfilter an, der tiefe Frequenzen unterhalb eines angegebenen Schwellenwertes filtert.
- **Audio Echo Filter**: Wendet ein Echo auf den Sound an, mit dem der Sound in einer bestimmten Verzögerung wiederholt wird.
- **Audio Distortion Filter**: Verzerrt den Sound auf einer Stufe von 0 (gar nicht verzerrt) bis 1 (voll verzerrt).

- **Audio Reverb Filter**: Wendet einen besonderen Hall auf die Audio Source an. Viele der Einstellungen stimmen mit einer Reverb Zone überein, die du in Abschnitt 17.3.5 kennenlernst.
- **Audio Chorus Filter**: Wendet einen Choreffekt auf den Sound an.

Ein weiterer Hinweis: Da es möglich ist, mehrere Filter auf eine Audio Source oder den Audio Lister anzuwenden, ist die Reihenfolge der Components bedeutsam. Filter, die weiter oben im Inspector platziert sind, werden somit vor Filtern angewendet, die weiter unten angeordnet sind.

17.3.2 Übung 17.03: Filter ausprobieren

Teste verschiedene Filter mithilfe der Audio Sources aus, die du aktuell im Projekt importiert hast. Wenn du die Veränderungen hörst, bekommst du ein gutes Gefühl dafür, wann der Einsatz eines Filters sinnvoll sein könnte.

17.3.3 Audio Mixer

Du kannst dir sicher vorstellen, dass ein großes Spiel viele verschiedene Audio Sources innerhalb einer Scene haben kann. Stell dir vor, dass du diese Geräusche in einer Einheit steuern oder gar verändern willst – das ist eine schwierige Aufgabe. Doch Unity steht dir zur Seite! Wir sehen uns den *Audio Mixer* an, eine Einheit, mit der verschiedene Audioquellen kategorisiert und in einer Einheit gesteuert oder mit Effekten überlagert werden können. So behältst du nicht nur einen besseren Überblick über verschiedene Audio Sources in der Scene, sondern hast auch jederzeit eine zentrale Steuereinheit, die du sogar über Scripts ansteuern kannst.

Gleichzeitig ist es möglich, das Audiosignal zwischen mehreren Audio Mixern zu »verketten«. Somit können Veränderungen von Sounds durch komplexe Regelungen genauso angepasst werden, wie du es dir vorstellst. Das ist nicht nur für jeden DJ ein Paradies! Stell dir zum Beispiel vor, dass dein Spielcharakter im Spiel unter Wasser taucht und alle Sounds (bis auf die normale Spielmusik) so klingen sollen, als würden sie unter Wasser stattfinden. Mithilfe von Audio Mixern lässt sich ein solches Ziel gut erreichen. Du wirst deshalb lernen, wie du Audio Mixer erstellst, auf Audio Sources anwendest und mit Effekten manipulierst.

Der erste Schritt besteht im Anlegen eines Audio Mixers im Projekt. Dafür wählst du das Menü Assets • Create • Audio Mixer aus. Anschließend wird eine Datei für den Audio Mixer erstellt, die bei einem Doppelklick zu einem neuen Fenster für den Audio Mixer führt (siehe Abbildung 17.7).

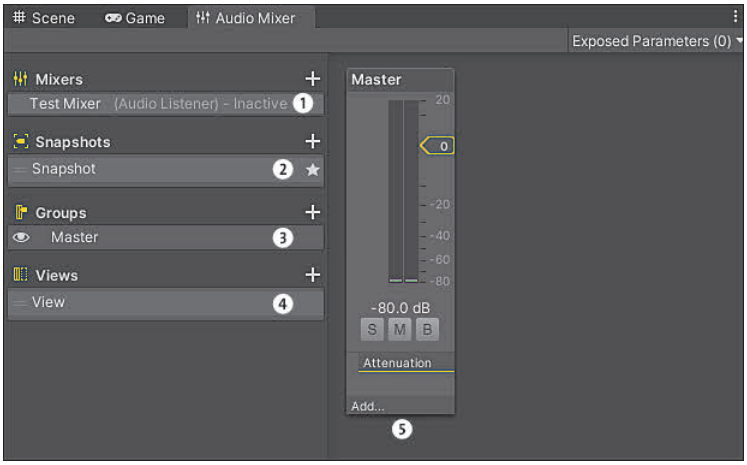


Abbildung 17.7 Ansicht eines neuen Audio Mixers im zugehörigen Audio-Mixer-Fenster

Die verschiedenen Optionen im Audio Mixer erfüllen dabei jeweils einen eigenen Zweck:

❶ MIXERS: Der Output Mixer für diesen Mixer. Durch diese Option kannst du mehrere Mixer verketten.

❷ SNAPSHOTS: Liste der Snapshots des Mixers. Snapshots beinhalten bestimmte Einstellungen für alle Parameter des Mixers und können im Spielverlauf verwendet werden, um Übergänge zu erzeugen.

❸ GROUPS: Enthält alle Gruppen des Mixers in einer Hierarchie. Standardmäßig ist nur eine Gruppe mit dem Namen MASTER enthalten, die das endgültige Audiosignal des Mixers zurückgibt.

❹ VIEWS: Beinhaltet Views, die bestimmte Anzeigeeinstellungen für Gruppen des Mixers festlegen können.

❺ **Group Strip Views**: In diesem Bereich werden die verschiedenen Gruppen des Mixers visuell angezeigt.

Eine neue Gruppe innerhalb des Mixers wird über das Plussymbol neben dem Bereich GROUPS erstellt. Je nach aktueller Auswahl wird die neue Gruppe in die bestehende Hierarchie eingeordnet, wobei die Gruppe MASTER das erste Elternelemente innerhalb der Gruppenanordnung darstellt (siehe Abbildung 17.8).

Das Zuweisen einer Ausgabegruppe des Audio Mixers ist simpel: Bei einer Audio Source muss lediglich die Einstellung OUTPUT neu zugewiesen werden. Wenn du auf den Punkt an der rechten Seite dieser Einstellung klickst, werden alle gefunden Audio Mixer mit ihren Gruppen aufgelistet. Durch das Anklicken einer Gruppe wird diese als Ausgabegruppe für die jeweilige Audio Source festgelegt (siehe Abbildung 17.9).

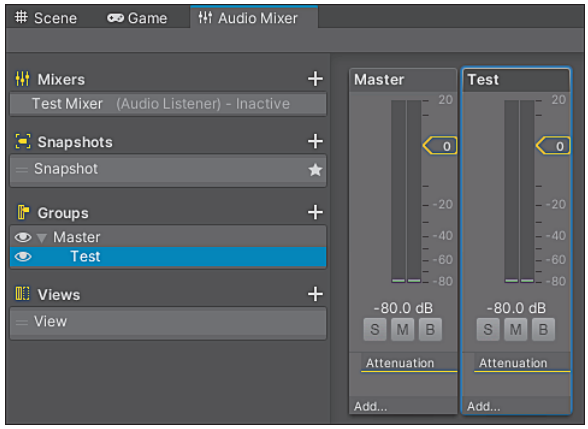


Abbildung 17.8 Ansicht nach dem Einordnen einer neuen Gruppe

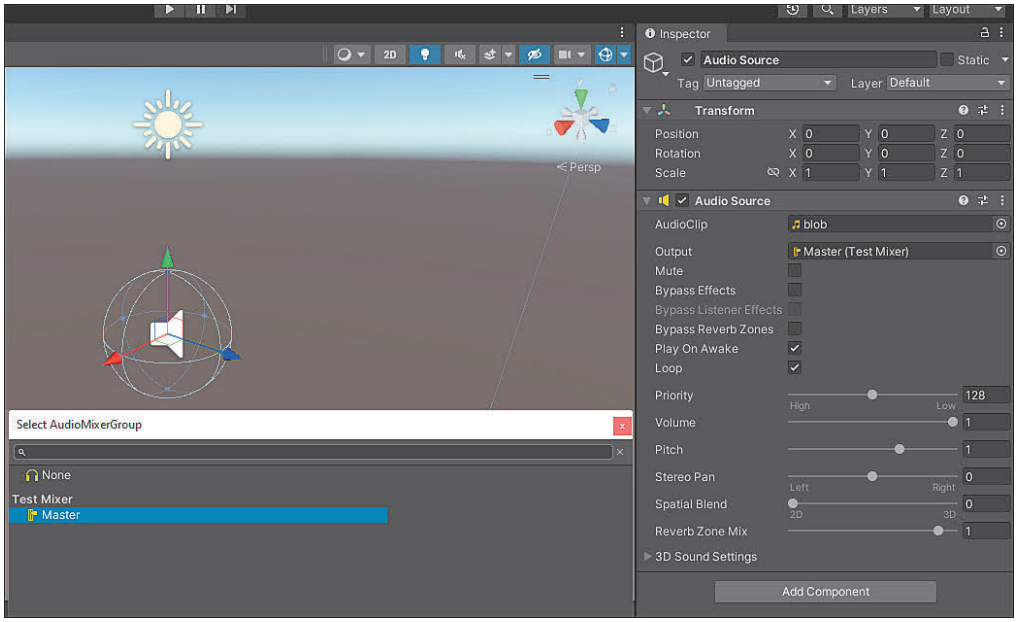


Abbildung 17.9 Die Master-Group wird der Audio Source zugewiesen.

Den Schritt der Zuweisung kannst du ebenso mit einem Script übernehmen, während das Spiel läuft. So kann sich der Sound schnell verändern – beispielsweise, wenn dein Spielcharakter unter Wasser taucht.

Der spannende Einsatz von Mixern zeigt sich dann, wenn du mit ihnen manipulierst, wie sich ein Geräusch anhört. Dafür kannst du im Mixer bestimmte Effekte festlegen, die die Ausgabe einer Gruppe verändern. Ähnlich wie bei den in Abschnitt 17.3.1 vorgestellten Filtern werden Effekte nacheinander auf eine Mixer Group angewendet.

Dabei werden auch hier Effekte, die weiter unten stehen, niedriger priorisiert als Effekte, die weiter oben gelistet werden. Insgesamt bieten die Effekte eine gute Möglichkeit, um schnell das Signal aller Audio Sources zu verändern, die die Gruppe verwenden.

Ein neuer Effekt wird über den Button ADD... unter einer Gruppe angelegt. Wie in Abbildung 17.10 zu sehen ist, stehen einige Effekte zur Auswahl.

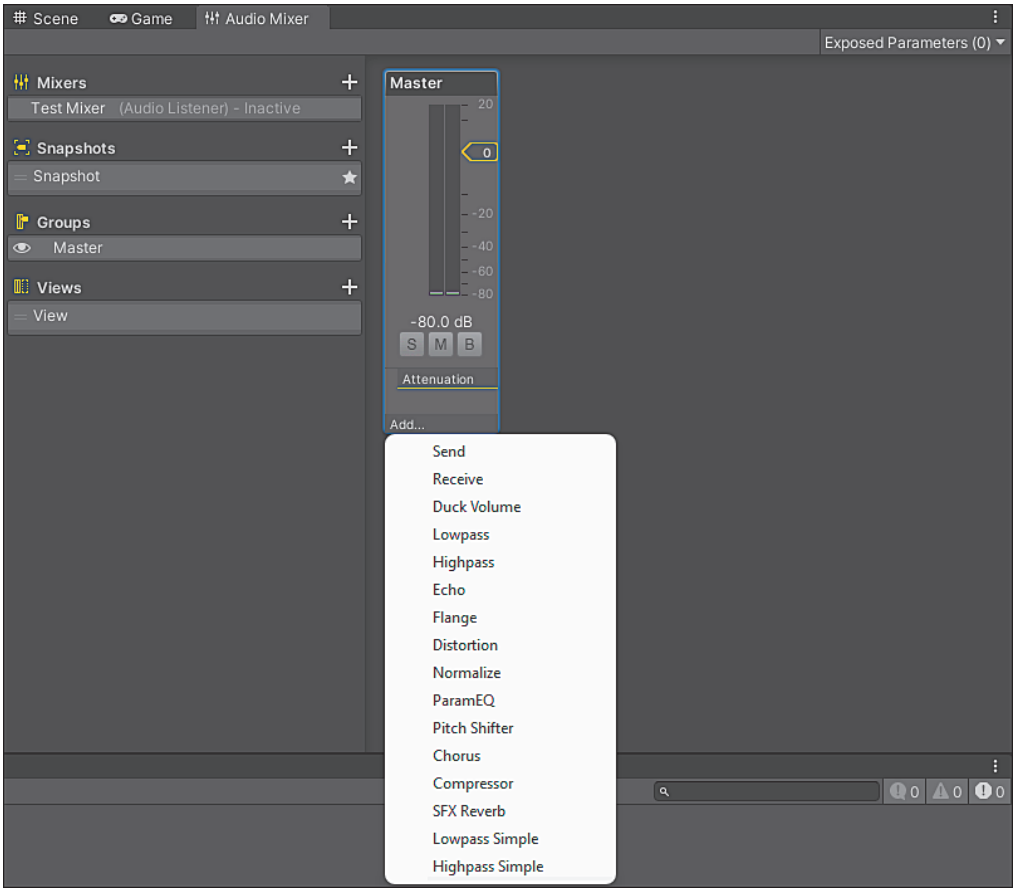


Abbildung 17.10 Hinzufügen eines neuen Effekts in einer Audio Mixer Group

Sehen wir uns an, welche Aufgaben die einzelnen Effekte im Groben erfüllen können:

- SEND und RECEIVE: Diese Effekte spielen eine besondere Rolle. SEND sendet das aktuelle Signal (beim Auftreten des Effekts) an eine bestimmte Effekteinheit weiter. RECEIVE bietet den Gegenpart dazu und wird mit der Ausgabe des SEND-Effekts verknüpft, um Effektausgaben entgegenzunehmen.
- DUCK VOLUME: Passt die Lautstärke des Audiosignals auf der Basis des Signals (oder eines anderen Signals) an.

- LOWPASS und HIGHPASS: Ähnelt dem Audio Low Pass/Audio High Pass Filter (siehe Abschnitt 17.3.1).
- ECHO: Ähnelt dem Audio Echo Filter (siehe Abschnitt 17.3.1).
- FLANGE: Verändert das Audiosignal, indem identische Signale so zusammengemischt werden, dass ein Signal um eine kleine Zeit verzögert wird.
- DISTORTION: Ähnelt dem Audio Distortion Filter (siehe Abschnitt 17.3.1).
- NORMALIZE: Verändert den Audiostream so, dass Unregelmäßigkeiten einem Zielwert angeglichen werden.
- PITCH SHIFER: Verändert den Pitch eines Audiosignals.
- PARAMEO: Legt lineare Filter fest, mit denen die Frequenzen des Audiosignals verändert werden können.
- CHORUS: Ähnelt dem Audio Chorus Filter (siehe Abschnitt 17.3.1).
- COMPRESSOR: Reduziert die Lautstärke von lauten Geräuschen oder verstärkt leise Geräusche.
- SFX REVERB: Ähnelt dem Audio Reverb Filter (siehe Abschnitt 17.3.1) sowie den Einstellungen der Reverb Zones (siehe Abschnitt 17.3.5).
- LOWPASS SIMPLE und HIGHPASS SIMPLE: Ähnelt dem Audio Low Pass und Audio High Pass Filter (siehe Abschnitt 17.3.1). Der Unterschied zum normalen Lowpass- und Highpass-Effekt besteht darin, dass kein Resonanzwert festgelegt werden kann, sondern nur der Schwellenwert für die jeweilige Frequenz.

Besonders das Konzept des Sendens (SEND) und Empfangens (RECEIVE) von Effekten zwischen Gruppen ist wichtig. Abbildung 17.11 zeigt ein Beispiel, bei dem ein mithilfe des Highpass-Effekts verändertes Audiosignal an den Receive-Effect einer anderen Gruppe weitergegeben wird. Auf diese Weise lassen sich schnell komplexe Zusammenhänge zwischen Audiosignalen darstellen.

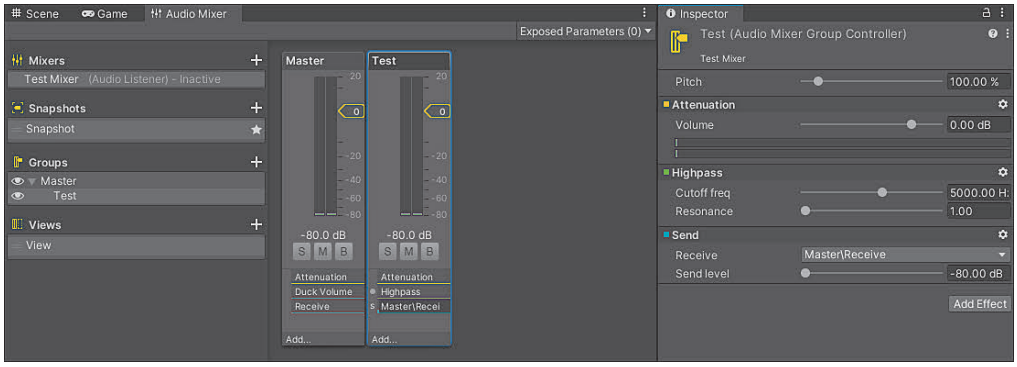


Abbildung 17.11 Senden eines Signals an eine andere Audio Mixer Group

Sounds richtig abmischen

Zugegeben, das Abmischen (und vor allem das gute Abmischen) von Sounds ist eine Wissenschaft für sich. Aus diesem Grund habe ich lediglich die absoluten Grundlagen des Themas angeschnitten. Für mehr Informationen steht dir die Dokumentation zur Seite: *https://docs.unity3d.com/Manual/AudioMixer*

So richtig interessant werden Audio Mixer Groups dann, wenn du ihre Einstellungen nach außen hin für Scripts öffnest. So kannst du beispielsweise über dein Optionsmenü darauf Einfluss nehmen, mit welcher Lautstärke Musik oder Soundeffekte in deinem Spiel hörbar sind, indem die Einstellungen direkt an die Audio Mixer Group weitergeleitet werden – so werden alle Audio Sources, die diese Group nutzen, parallel manipuliert. Genau dieses Beispiel wollen wir simulieren.

Zuerst musst du dafür einen Kanal einer Audio Mixer Group nach außen hin verfügbar machen. Für das Beispiel habe ich eine neue Gruppe erstellt, die im Spiel für alle Musikgeräusche stehen könnte. Nutzen wir im Beispiel die Eigenschaft Volume unter Attenuation. Zum Verfügbarmachen der Eigenschaft nach außen musst du nach einem Rechtsklick auf die Eigenschaft Expose ›Volume (of Music)‹ to Script wählen, wobei »Music« der Name der Mixer Group ist (siehe Abbildung 17.12).

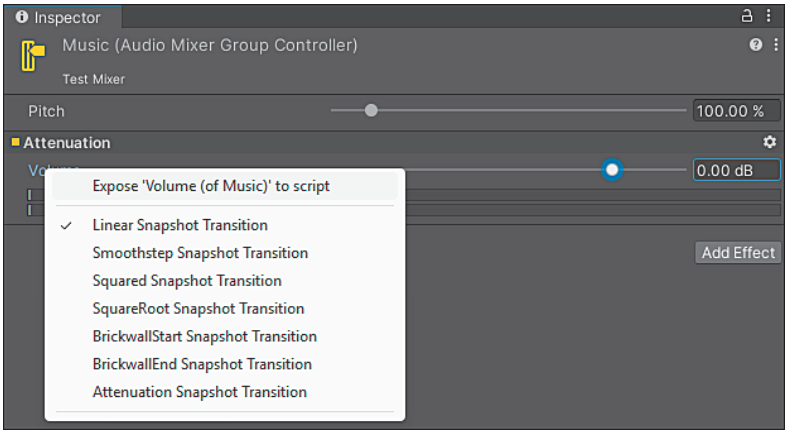


Abbildung 17.12 Der »Volume«-Parameter der Audio Mixer Group wird nach außen hin geöffnet.

Wie du in Abbildung 17.13 sehen kannst, taucht der Parameter danach in der Liste Exposed Parameters innerhalb des Audio-Mixer-Fensters auf. Standardmäßig heißt der neue Parameter MyExposedParam, mit einem Rechtsklick und der Option Rename kannst du den Parameter umbenennen. Nutzen wir als Namen Music-Volume.

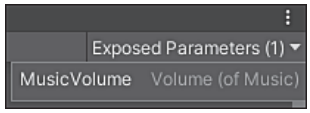


Abbildung 17.13 Der Parameter wurde unter dem Namen »MusicVolume« nach außen für Scripts geöffnet.

Als Letztes kümmern wir uns um das Script (siehe Listing 17.5) – zur Vereinfachung bauen wir kein ganzes Menü, sondern stellen die Lautstärke der Mixer Group über einen Slider im Inspector ein. Dafür kommt das Attribut `Range` zum Einsatz – hier sollen Werte zwischen –30 (die Musik ist quasi nicht mehr hörbar) und 0 (damit erhält die Musik ihre Originallautstärke) einstellbar sein. Für die Referenz zum `AudioMixer` kannst du ein einfaches `SerializeField` verwenden. Den Parameter setzt du über die Methode `SetFloat`, wobei du einen neuen Wert über den Namen zuweist. In unserem Fall trägt der Parameter den Namen `MusicVolume`, sodass wir den Wert testweise in jedem Frame neu setzen können. In einem richtigen Einstellungsmenü würde dieser Aufruf nur einmalig passieren, zum Beispiel wenn die Optionen gespeichert werden.

```
using UnityEngine;
using UnityEngine.Audio;

public class AudioMixerChanger : MonoBehaviour
{
  [SerializeField] [Range(-30f, 0f)] private float volume;
  [SerializeField] private AudioMixer mixerGroup;

  private void Update()
  {
    mixerGroup.SetFloat("MusicVolume", volume);
  }
}
```

Listing 17.5 Das Script stellt einen Slider bereit, mit dem der Parameter »MusicVolume« dauerhaft verändert wird.

Du kannst dein Script einfach testen, indem du einer Audio Source die Gruppe mit dem neuen Parameter zuweist und beim Script den Audio Mixer referenzierst, unter dem dieser Parameter liegt. Beim Spielen kannst du nun am Slider im INSPECTOR drehen, damit sich die Lautstärke der Mixer Group live ändert (siehe Abbildung 17.14). Würdest du mehrere Audio Sources mit deiner Mixer Group MUSIC verwenden, würden sie alle die neue Musiklautstärke nutzen.

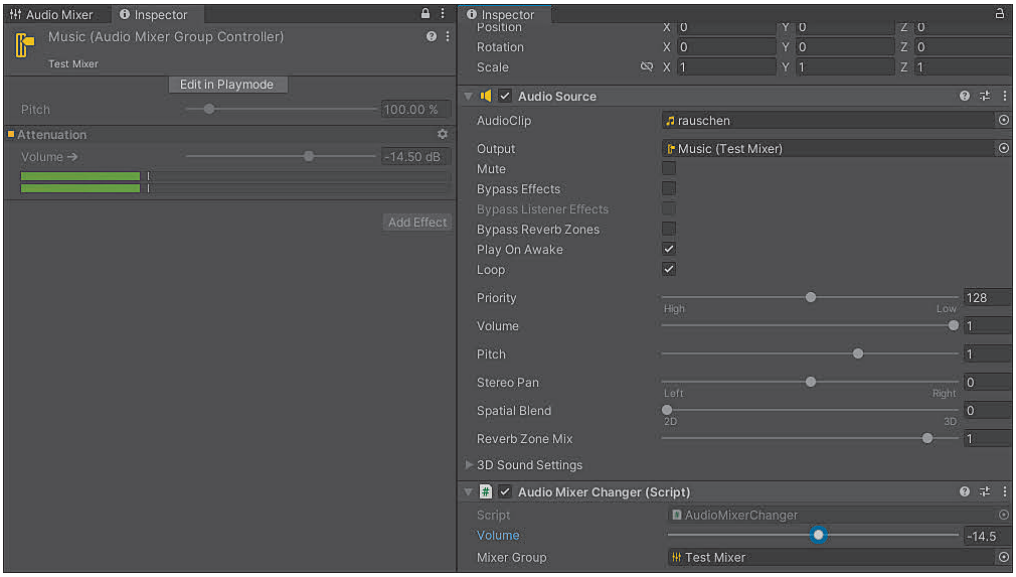


Abbildung 17.14 Über den Range-Slider kannst du live während des Spiels die Lautstärke ändern – die Änderung wird sofort auf den Mixerparameter übertragen, und Geräusche werden leiser oder lauter.

17.3.4 Übung 17.04: Mixe dein eigenes Audiosignal!

Probiere dich am Audio Mixer aus, indem du für ein bestimmtes Signal eine eigene Version mischst. Wenn du keine Ideen hast, was du erreichen möchtest, dann nimm dir eines der folgenden Ziele vor:

- Der Sound soll sich so anhören, als wäre man unter Wasser.
- Der Sound soll sich anhören wie im Weltall.
- Der Sound soll sich anhören, als würde er in einem alten Radio gespielt werden.

Experimentiere mit verschiedenen Effekten, um ein gutes Resultat zu erzielen.

17.3.5 Reverb Zones

Audio Reverb Zones werden eingesetzt, um in einem bestimmten Bereich einen Hall zu simulieren. Wie du an einem einfachen Beispiel sehen kannst, ist das Anlegen und Einrichten einer solchen Zone denkbar einfach. Eine neue Reverb Zone wird über GameObject • Audio • Audio Reverb Zone in der Scene angelegt (siehe Abbildung 17.15).

Ähnlich wie eine normale Einstellung wird ein Bereich angegeben, in dem der Hall seinen vollen Effekt entfaltet (MinDistance). Die MaxDistance gibt die Entfer-

nung an, bis zu der der Hall außerhalb des Minimalbereichs schwächer wird. Beide Einstellungen werden mit Kugeln in der Scene View angezeigt, die du wie bei einer Audio Source mithilfe der Maus anpassen kannst. Unterhalb der Einstellung REVERB-PRESET stehen zahlreiche Voreinstellungen für bestimmte Umgebungen bereit, für die ein Hall simuliert werden soll. Während GENERIC einen generischen Hall erzeugt, sind unter anderem Voreinstellungen für Höhlen (CAVE), Städte (CITY) oder Wälder (FOREST) zu finden. Mithilfe der Einstellung USER kannst du alle Parameter des Halls manuell konfigurieren.

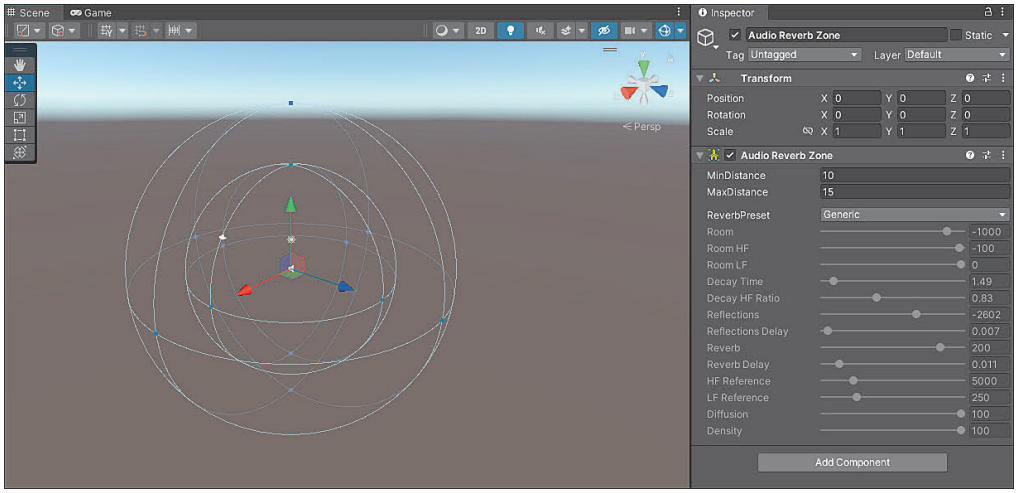


Abbildung 17.15 Ansicht einer neuen Reverb Zone

Den Effekt einer Reverb Zone kannst du am besten an einem Beispiel erkennen: Setze eine Audio Source (die am besten auf LOOP gestellt wird) innerhalb der Reverb Zone ein. Starte anschließend das Spiel und deaktiviere und aktiviere in Abständen das Game Object mit der Zone. Der Unterschied zwischen dem »normalem« Klang und dem Hall sollte gut erkennbar sein. Auf diesem schnellen Weg lassen sich schnell bestimmte Areale »halliger« anhören. Für Gebiete wie Höhlen oder geschlossene Räume kann das die Soundumgebung für dein Spiel deutlich aufhübschen!

17.4 Mikrofoneingaben

Auch wenn es für die meisten Spiele wohl eher uninteressant ist, sehen wir uns zuletzt an, wie du Mikrofoneingaben im Spiel verwerten kannst. Die Klasse `Microphone` stellt statische Eigenschaften und Methoden bereit, mit denen wir eine Aufnahme durchführen und in Unity abspielen wollen. Anhand eines Beispiels in Listing 17.6 sollst du kennenlernen, wie die eigene Stimme aufgenommen und auf einer Audio Source abgespielt werden kann.

Dafür müssen wir zunächst das passende Eingabegerät finden: Mithilfe von `Microphone.devices` wird ein `string`-Array mit gültigen Eingabegeräten zurückgegeben. Von diesen nehmen wir das erste gefundene Element. Die eigentliche Aufnahme wird über die Methode `Microphone.Start` begonnen. Als erster Parameter wird das gespeicherte Eingabegerät übergeben. Der zweite Parameter gibt als `bool` an, ob auch nach Ende der Aufnahme weiter aufgenommen werden soll. Der dritte Parameter gibt die Aufnahmezeit in Sekunden an. Der letzte Parameter gibt die Frequenz an, mit der aufgenommen werden soll. Ein Wert von 44.000 ist hier gängig. Die Methode gibt einen Audioclip zurück, den du im Anschluss an die Audio Source übergeben kannst. Zum Schluss wird die Audio Source, die vom Mikrofon aufgenommen wurde, abgespielt.

```
using UnityEngine;

public class MicrophoneTest : MonoBehaviour
{
  private AudioSource audioSource;

  private void Awake()
  {
    audioSource = GetComponent<AudioSource>();
  }

  private void Start()
  {
    string device = Microphone.devices[0];
    AudioClip clip = Microphone.Start(device, true, 3, 44000);

    audioSource.clip = clip;
    audioSource.Play();
  }
}
```

Listing 17.6 Beispiel für die Ausgabe des Mikrofons mit einer Verzögerung von 3 Sekunden

Das Script kannst du nun einer Audio Source anfügen, die die Eigenschaft Loop aktiviert hat. Erscheint beim Starten des Spiels ein Fehler, dass der Index das Array `Microphone.devices` »überlaufen« hat, konnte kein passendes Mikro an deinem PC gefunden werden. Wenn keine Exception erscheint, du aber dennoch nichts hörst, kannst du versuchen, einen anderen Index für `Microphone.devices` zu übergeben. Wenn hingegen alles funktioniert, solltest du nach 3 Sekunden deine eigene Stimme hören.

Die Stimme spielt danach immer weiter in einem dreisekündigen Abstand, bis du das Spiel beendest. Vielleicht wirkt das etwas gruselig, aber so siehst du immerhin, dass

Unity deine Stimme aufnehmen kann! Mithilfe des Clips, der vom Mikrofon »ausgelesen« wird, sowie mit den aktuellen Abspieldaten der Audio Source kannst du das Mikrofon nicht nur zum Hören deiner eigenen engelsgleichen Stimme, sondern auch als richtiges Eingabegerät für dein Spiel verwenden.

17.5 Musterlösung für die Übungen

17.5.1 Lösung für Übung 17.01: Die Neustarttaste

Durch den Aufruf der Methoden `Stop` und `Play` hintereinander wird die Audio Source unmittelbar nach dem Beenden von vorn abgespielt. Das kannst du dir für die Aufgabe zunutze machen.

```
using UnityEngine;

public class RestartAudio : MonoBehaviour
{
  private AudioSource audioSource;

  private void Start()
  {
    audioSource = GetComponent<AudioSource>();
  }

  private void Update()
  {
    if (Input.GetKeyDown(KeyCode.Space))
    {
      audioSource.Stop();
      audioSource.Play();
    }
  }
}
```

Listing 17.7 Musterlösung für Übung 17.01

Beim Drücken der Leertaste sollte deine verwendete Beispieldatei von vorn abgespielt werden.

17.5.2 Lösung für Übung 17.02: Lauter oder leiser, bitte!

Die eigentliche Logik zum Abfragen der Tasten ist simpel. Schwieriger wird es beim Neusetzen der Lautstärke, dabei wird der Eigenschaft `volume` der Audio Source ein

neuer Wert zugewiesen. Je nach gedrückter Taste wird der aktuelle Wert um »0.1« Einheiten erhöht oder gesenkt. Um zu verhindern, dass der Wert 0 unterschreitet oder 1 überschreitet, kannst du die Hilfsmethode `Mathf.Clamp01` verwenden.

```
using UnityEngine;

public class VolumeButtons : MonoBehaviour
{
  private AudioSource audioSource;

  private void Start()
  {
    audioSource = GetComponent<AudioSource>();
  }

  private void Update()
  {
    if (Input.GetKeyDown(KeyCode.UpArrow))
    {
      audioSource.volume = Mathf.Clamp01(audioSource.volume + 0.1f);
    }

    if (Input.GetKeyDown(KeyCode.DownArrow))
    {
      audioSource.volume = Mathf.Clamp01(audioSource.volume - 0.1f);
    }
  }
}
```

Listing 17.8 Musterlösung für Übung 17.02

Mithilfe von ↑ und ↓ kannst du nun die Lautstärke einer Audio Source direkt im Spiel steuern. Menüs, mit denen man die Lautstärke manuell einstellen kann, werden überbewertet!

17.5.3 Lösung für Übung 17.03: Filter ausprobieren

Da es hier eher ums Ausprobieren als um ein bestimmtes Ergebnis geht, gibt es zu dieser Übung keine Musterlösung.

17.5.4 Lösung für Übung 17.04: Mixe dein eigenes Audiosignal!

Bei dieser Übung sind deiner Kreativität keine Grenzen gesetzt – tob dich aus und versuche, die Funktionsweise von Audio Mixern besser zu verstehen.

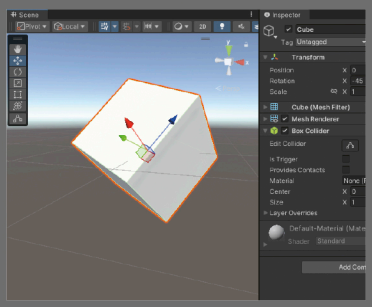
Inhalt

Spieleentwicklung mit Unity

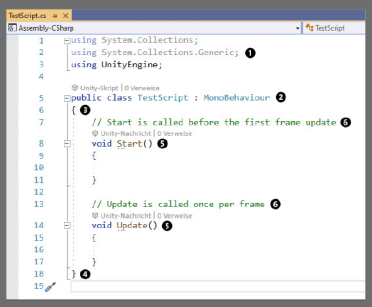
»Spiele entwickeln leicht gemacht«

Lebe deinen Traum – werde zum Game-Profi

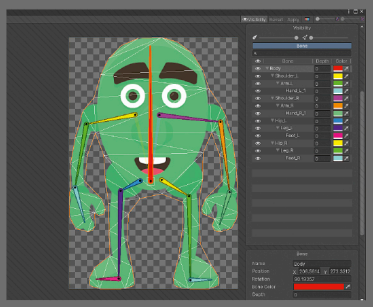
Du willst Spiele entwickeln mit Unity? Mit diesem Buch schaffst du es. Vorkenntnisse brauchst du nicht, denn du lernst alles hier: von den Grundkonzepten der Engine über Prefabs und Scripts bis zum komplexen 3D-Spiel. Viele Übungen helfen dir dabei, Unity zu beherrschen und eigene Ideen umzusetzen. Werde kreativ und entwickle dein erstes eigenes Spiel!



Unity kennenlernen



Scripts mit C# schreiben



Eigene Spiele programmieren

Unity installieren und los geht´s

Das Buch begleitet dich von der Installation von Unity bis zu fertigen 2D- und 3D-Spielen. Dabei lernst du Schritt für Schritt alles über Unity und die Spieleentwicklung mit C#. Wer etwas geübter ist, kann auch gezielt Themen nachschlagen.

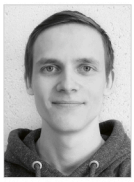
Mit Übung zum Erfolg

An vielen Übungen trainierst du deine Skills und lernst die Konzepte, das Scripting mit C# und die Logik dahinter gründlich kennen. Bau dir ein immer besseres Repertoire auf und mach aus deinen eigenen Ideen spielbare Games!

Professionelle Spiele in 2D und 3D

Ob in zwei oder drei Dimensionen: Übersichtliche UIs, robuster und sauberer Code, aufregende Spielewelten und realistische Effekte kommen nicht von ungefähr. Auch zu den Profi-Themen wie automatischer Bewegung gibt es viel Material zum Üben.

Alle Code-Beispiele, Projektmaterialien und Musterlösungen der Übungen stehen zum Download bereit.



Max Schlosser forscht an der Hochschule Mittweida im Bereich Medieninformatik und ist durch und durch Gaming-Fan. Unity setzt er in Spieleprojekten und in der Forschung ein. Sein umfassendes Wissen erklärt er leicht verständlich im Buch und auf seinem YouTube-Kanal »Schloool«.

Aus dem Inhalt

Rheinwerk Computing

Gedruckt in Deutschland
Papier aus nachhaltiger Waldwirtschaft
Mineralölfreie Druckfarben

€ 49,90 [D] € 51,30 [A]

Für Windows, macOS und Linux Spiele

ISBN 978-3-8362-9539-0