

3

Quickstart: 3D-Entwicklung mit Unity

Jetzt, wo du über die Grundlagen von Virtual Reality Bescheid weißt, können wir die ersten Schritte in Unity machen. In diesem Kapitel wirst du Unity installieren, ein neues Projekt anlegen und die Grundlagen von Unity erlernen. Außerdem wirst du ein erstes Test-Projekt erstellen und es auf deiner Virtual-Reality-Brille testen.

■ 3.1 Installation

Als Erstes musst du natürlich eine Version von Unity installieren. Wie in der Einführung zum Buch bereits erwähnt, kannst du dir entweder die neuste Version herunterladen oder die Version verwenden, die ich beim Schreiben dieses Buches verwendet habe. Wählst du die neuste Version, kann es eventuell passieren, dass einzelne Menüs oder Funktionen etwas anders aussehen oder bedient werden müssen. Ich empfehle dir deswegen, die Version *Unity 2017.1.0f3* aus dem *Unity Download Archiv* zu verwenden.

Sowohl ältere als auch die neuste Version findest du auf der offiziellen Unity-Webseite: <https://unity.com>

Zu dem Download-Bereich kommst du mit einem Klick auf **HOLEN SIE SICH UNITY**. Über **PROBIEREN SIE PERSONAL** bzw. auf Englisch **TRY PERSONAL** gelangst du zum Download der neusten Version. Wenn du auf der Seite aber ganz nach unten scrollst, findest du dort unter *Resources* den Punkt **ÄLTERE UNITY-VERSIONEN** bzw. **OLDER VERSIONS OF UNITY**. Hinter diesem Link befindet sich das *Download Archiv*. Scrolle dort zu dem Punkt *Unity 2017.1.0* vom 10. Juli 2017 und wähle in dem *Downloads (Win)*-Menü **UNITY-INSTALLATIONSPROGRAMM** aus. Der Download startet anschließend automatisch.

Alternativ habe ich dir in dem Info-Kasten auch Direkt-Links zu beiden Download-Seiten hinterlegt.



Unity Download

Direkt-Link zur Version 2017.1f3 im Unity-Download-Archiv:

https://unity3d.com/de/get-unity/download?thank-you=update&download_nid=47505&os=Win

Download der neuesten Version:

<https://store.unity.com/download?ref=pers>

Dort klickst du auf **DOWNLOAD INSTALLER**.

Nachdem der Download abgeschlossen ist, kannst du das Installationsprogramm starten. Folge den Anweisungen des Installationsprogramms und klicke auf **NEXT**. Stimme im nächsten Bildschirm den Lizenzvereinbarungen zu und klicke erneut auf **NEXT**.

Im nächsten Bildschirm fragt dich Unity, welche Komponenten du installieren möchtest. Hier musst du, wie in Bild 3.1 zu sehen, folgende Komponenten auswählen:

- Unity 2017
- Documentation
- Standard Assets
- Example Project
- Microsoft Visual Studio Community 2017

Wenn du diesem Buch mit einer *Samsung GearVR*- oder *GoogleVR*-Brille folgst, musst du zusätzlich noch die Android-Unterstützung auswählen:

- Android Build Support



Fehlermeldung beim Starten der Installation?

Wenn du beim Starten der Installation folgende Meldung erhältst „*Unity ... was not able to verify whether Visual Studio 2017 is installed.*“, ist das nicht weiter schlimm. Bestätige die Meldung mit OK und fahre mit der Installation fort. Diese Meldung bedeutet lediglich, dass du selber entscheiden musst, ob du *Visual Studio 2017* installieren möchtest oder ob du es bereits installiert hast, weil die automatische Erkennung fehlgeschlagen ist.

Visual Studio 2017 bereits installiert?

Solltest du *Visual Studio 2017* bereits installiert haben, bietet dir Unity ggf. an, die *Microsoft Visual Studio Tools for Unity* zu installieren. Aktiviere diese Option, wenn sie angezeigt wird. Sollte weder *Visual Studio 2017* noch diese Option angezeigt werden, hast du bereits alle benötigten Werkzeuge auf deinem PC installiert.

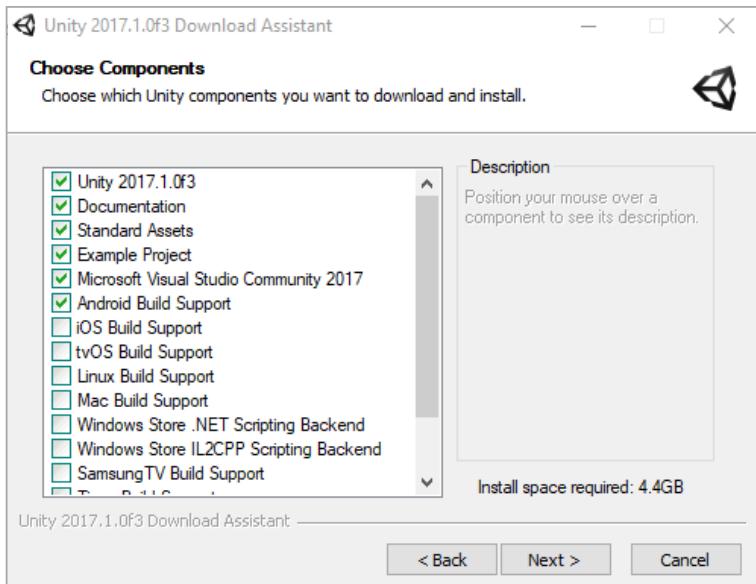


Bild 3.1 Im Unity-Download-Assistenten solltest du alle im Bild ausgewählten Komponenten installieren.

Im nächsten Fenster kannst du das Zielverzeichnis für den Download und die Installation auswählen. Lässt du für das Download-Verzeichnis das Standardverzeichnis aktiv, werden die heruntergeladenen Dateien automatisch gelöscht, nachdem die Installation abgeschlossen ist.

Wenn du *Visual Studio* installierst, musst du in dem darauffolgenden Fenster den Lizenzvereinbarungen von Microsoft zustimmen. Nach einem weiteren Klick auf **NEXT** startet der Installationsvorgang.

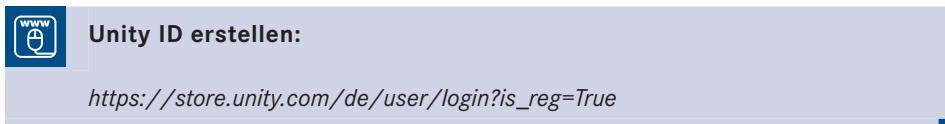
Während der Installation werden die benötigten Komponenten aus dem Internet heruntergeladen, deswegen kann die Installation einen Moment dauern. In der Zwischenzeit kannst du aber schon im nächsten Kapitel eine *Unity ID* anlegen und mit den darauffolgenden Kapiteln fortfahren.

Während der Installation öffnen und schließen sich verschiedene Fenster, in denen zum Beispiel *Visual Studio 2017* heruntergeladen wird. Du musst hier nicht eingreifen, sondern kannst einfach abwarten, bis alle Komponenten vollständig installiert sind.

3.1.1 Die Unity ID

Um weiter machen zu können, benötigst du nun eine Unity ID. Dieses Benutzerkonto muss einmalig erstellt werden und wird für die Anmeldung bei allen Unity Services verwendet. Zu den Unity Services gehören neben Lizenz- und Team-Verwaltung auch der *Asset Store* oder das offizielle Forum. Besuche zum Erstellen der Unity ID die in dem Kasten angegebene Adresse und füllle das Formular aus.

Am Ende der Registrierung erhältst du eine Bestätigungs-E-Mail an die von dir angegebene E-Mail-Adresse. Darin befindet sich ein Link, den du anklicken musst, um deine *Unity ID* zu aktivieren.



The screenshot shows a web browser window. On the left, there is a blue icon containing a white computer mouse and the word "www". To its right, the text "Unity ID erstellen:" is displayed in bold black font. Below this, a URL is shown in a text input field: https://store.unity.com/de/user/login?is_reg=True.

3.1.2 Android SDK & Java Development Kit(GearVR, GoogleVR)

Neben Unity selbst musst du zum Erstellen von Android-Apps noch das *Java Development Kit* und das *Android Software Development Kit* installiert haben. Nachfolgend erkläre ich dir, wo du die Werkzeuge herunterladen kannst und was du bei der Installation beachten musst.

3.1.2.1 Java Development Kit

Das JDK wird benötigt, um Java-Anwendungen entwickeln zu können. Da Android zu einem großen Teil auf Java basiert, wird dieses benötigt, wenn du eine Anwendung für Android entwickeln möchtest. Da das „JDK“ regelmäßig aktualisiert wird, kann ich dir keinen direkten Link geben, sondern nur eine Anleitung für den Download.



The screenshot shows a web browser window. On the left, there is a blue icon containing a white computer mouse and the word "www". To its right, the text "Java Development Kit 8 – Download-Seite" is displayed in bold black font. Below this, a URL is shown in a text input field: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>.

Öffne die Webseite aus dem Info-Kasten oben. Auf der offiziellen Seite von Oracle findest du die aktuelle JDK 8-Version als Download für alle unterstützten Betriebssysteme. Wähle in dem obersten Kasten mit dem Titel *Java SE Development Kit 8u ...* zunächst den Punkt **ACCEPT LICENSE AGREEMENT** aus, um den Java-Lizenzberechtigungen zuzustimmen. Danach kannst du die für dein System passende Version herunterladen, indem du rechts neben der Betriebssystem-Bezeichnung auf den Download-Link klickst (z.B. **JDK-8U...-WINDOWS-X64.EXE** für Windows 64-Bit).

Java SE Development Kit 8u144		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
<input type="radio"/> Accept License Agreement	<input checked="" type="radio"/> Decline License Agreement	
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.89 MB	jdk-8u144-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.83 MB	jdk-8u144-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.65 MB	jdk-8u144-linux-i586.rpm
Linux x86	179.44 MB	jdk-8u144-linux-i586.tar.gz
Linux x64	162.1 MB	jdk-8u144-linux-x64.rpm
Linux x64	176.92 MB	jdk-8u144-linux-x64.tar.gz
Mac OS X	226.6 MB	jdk-8u144-macosx-x64.dmg
Solaris SPARC 64-bit	139.87 MB	jdk-8u144-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.18 MB	jdk-8u144-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u144-solaris-x64.tar.Z
Solaris x64	96.99 MB	jdk-8u144-solaris-x64.tar.gz
Windows x86	190.94 MB	jdk-8u144-windows-i586.exe
Windows x64	197.78 MB	jdk-8u144-windows-x64.exe

Bild 3.2 Die relevanten Felder auf der Oracle-Download-Seite

Nach dem Download kannst du das JDK installieren. Bei der Installation musst du nichts weiter beachten. Falls du nicht das vorausgewählte Installationsverzeichnis verwendest, merke dir den Pfad, da du ihn später noch angeben musst.

Nach der Installation öffnet sich eventuell automatisch ein weiteres Installationsfenster, in dem du einen Installationspfad für die *Java Runtime Environment* (JRE) angeben must. Diese Installation wird benötigt, um Java-Programme ausführen zu können, und ist wahrscheinlich bereits auf deinem PC vorhanden. In diesem Fall öffnet sich das Fenster nicht und die Installation ist abgeschlossen.

3.1.2.2 Android Software Development Kit

Das *Android SDK* mit den *Platform Tools* und dem *Google USB Treiber* benötigst du, um Anwendungen für Android zu entwickeln. Am einfachsten ist das *SDK* als Teil der Entwicklungsumgebung *Android Studio* zu installieren, da hier ein Großteil der Einrichtung automatisch abläuft und dies den Vorgang deutlich vereinfacht.

Öffne die in dem Kasten angegebene Webseite zum Download von *Android Studio* und dem *Android SDK*.


Android Studio Download:

<https://developer.android.com/studio/index.html>

Klicke auf der Seite auf die Schaltfläche **DOWNLOAD ANDROID STUDIO**. Es öffnet sich dann ein Fenster mit den Lizenzbestimmungen für diese Software. Bestätige über die Checkbox, dass du sie gelesen hast und einverstanden bist. Mit einem Klick auf **DOWNLOAD ANDROID STUDIO FOR WINDOWS** startet dann der Download.

Sobald der Download abgeschlossen ist, kannst du den *Android Studio*-Installer starten und den Anweisungen des Installationsprogramms folgen. Bei der Auswahl des Installationsver-

zeichnisses ist es wichtig, dass du dir den Installationspfad von dem *Android SDK* merkst, falls du ihn änderst.¹

Ist die Installation abgeschlossen, lass die Option *Start Android Studio* aktiviert und klicke auf **FINISH**. Wir müssen *Android Studio* einmalig starten, um unter anderem den *Google USB-Treiber* zu installieren.

Bei deinem ersten Start von *Android Studio* wirst du mit einem Einrichtungs-Bildschirm empfangen. Klicke auf **NEXT**. Wähle in dem nächsten Bildschirm den Installationstyp *Standard* aus. Dort siehst du, wie in Bild 3.3 dargestellt, eine Übersicht der SDK-Komponenten, die im nächsten Schritt noch automatisch installiert werden. Klicke auf **FINISH**, um die gewählten Optionen zu bestätigen. *Android Studio* lädt nun die benötigten Dateien herunter.

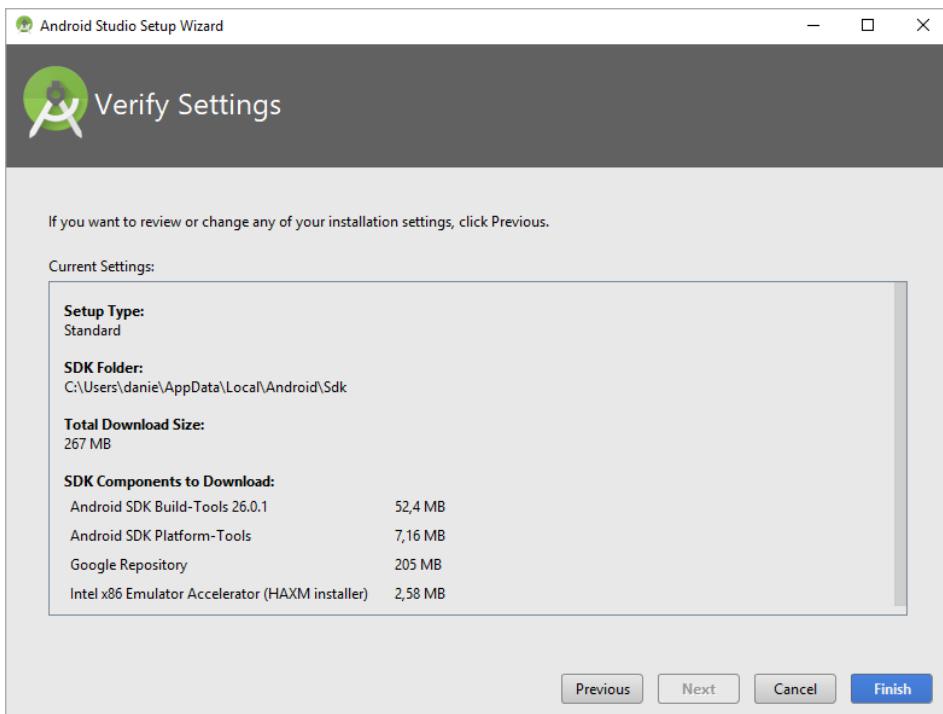


Bild 3.3 So sieht die Übersicht der zu installierenden Komponenten aus.

Nachdem alle benötigten Dateien heruntergeladen und eingerichtet wurden, öffnet sich der Willkommens-Bildschirm von *Android Studio*. Von hier aus können wir jetzt den *SDK Manager* starten, um unter anderem den *Google USB-Treiber* zu installieren. Klicke, wie in Bild 3.4 zu sehen, in dem Willkommens-Bildschirm unten rechts auf **CONFIGURE** gefolgt von **SDK MANAGER**.

¹ Das *Android SDK* benötigt ca. 3,2 GB an Festplattenspeicher. Falls du mehrere Festplatten verwendest, zum Beispiel eine SSD und eine HDD, kannst du das *SDK* auch auf einer anderen Festplatte installieren. Du musst dir in diesem Fall nur den Installationspfad merken, da du ihn später noch angeben musst.

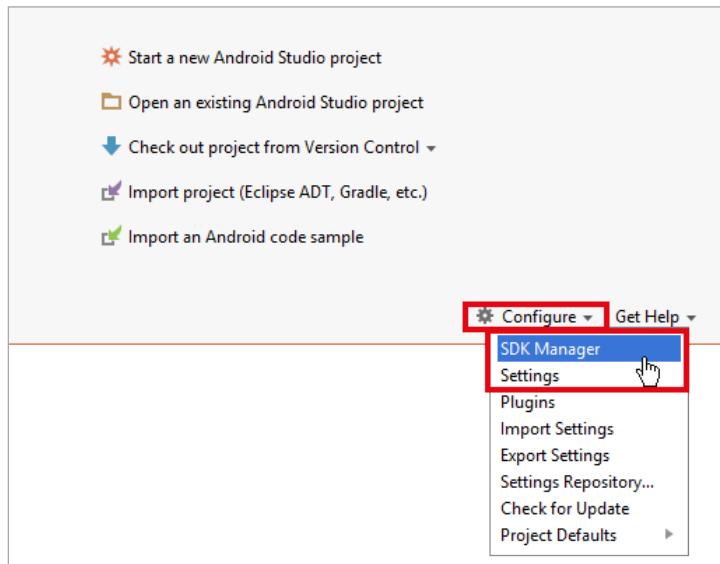


Bild 3.4 Öffne den SDK Manager.

Es öffnet sich ein Einstellungsfenster, in dem du das Android SDK verwalten kannst. Auf der rechten Seite siehst du verschiedene Komponenten und Werkzeuge, die du installieren kannst, indem du die Checkbox daneben aktivierst.

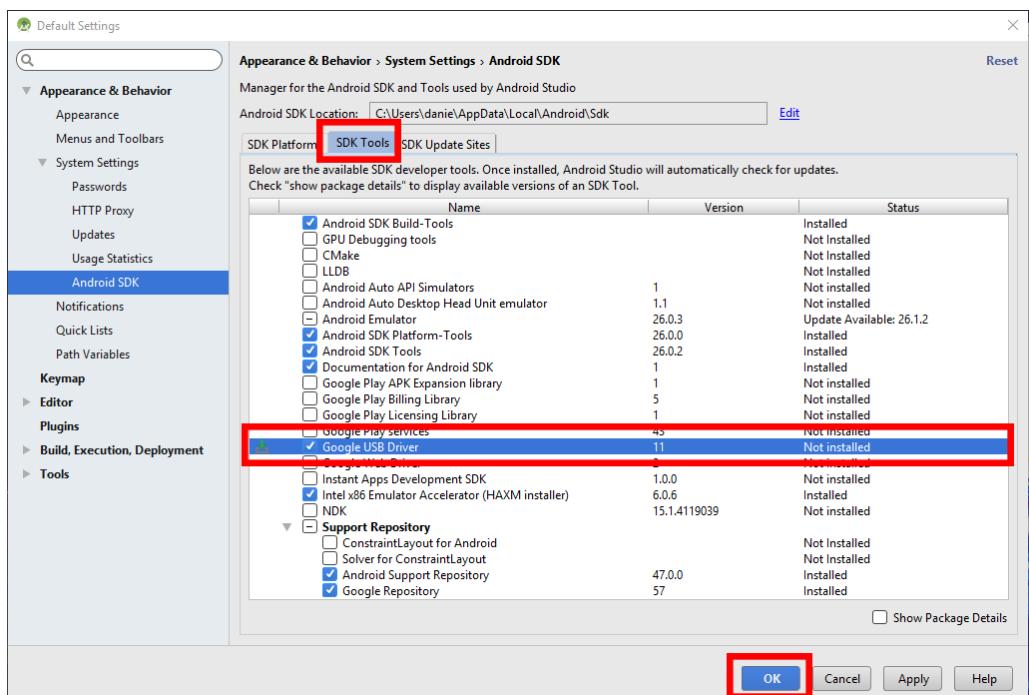


Bild 3.5 So installierst du den Google USB-Treiber.

Klicke auf den in Bild 3.5 markierten Reiter **SDK TOOLS** und aktiviere dort den Eintrag *Google USB Driver*. Klicke anschließend auf **OK**. Es öffnet sich zunächst ein Dialogfenster, das dich darüber informiert, welche Komponenten nun installiert werden. Wenn du die Meldung mit **OK** bestätigst, öffnet sich ein Fenster, in dem du den Lizenzbestimmungen der Komponenten zustimmen musst. Klicke auf **ACCEPT** und starte mit einem Klick auf **NEXT** den Download. Sobald der Treiber installiert ist, schließt du mit einem Klick auf **FINISH** den *SDK Manager* und kannst anschließend auch den Willkommens-Bildschirm von Android Studio schließen. Hier bist du nun fertig und kannst Unity starten.

■ 3.2 Der erste Start

Beim ersten Start von Unity musst du dich zunächst mit deiner *Unity ID* einloggen. Wenn du die Option **REMEMBER ME** auswählst, wird dieser Schritt zukünftig übersprungen.

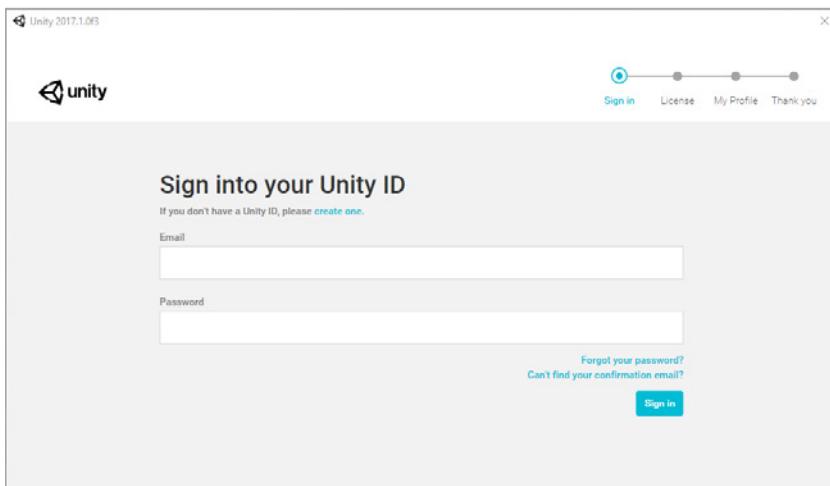


Bild 3.6 Der Login-Bildschirm beim ersten Start

Anschließend wählst du in dem darauffolgenden Fenster, dass du die *Personal*-Version verwenden möchtest. Nach einem Klick auf **NEXT** wird sich das Popup-Fenster aus Bild 3.7 öffnen. Hier musst du angeben, wie viel deine Firma im letzten Jahr verdient hat, wenn du Unity kommerziell verwendest. Wenn du deine Spiele erst einmal privat entwickelst, kannst du die Option **I DON'T USE UNITY IN A PROFESSIONAL CAPACITY** auswählen. Die Frage bezieht sich auf den aktuellen Zustand. Nur weil du angibst, Unity derzeit nicht kommerziell zu verwenden, heißt es nicht, dass du Unity deswegen niemals kommerziell verwenden darfst. Bestätige deine Auswahl mit **NEXT**.

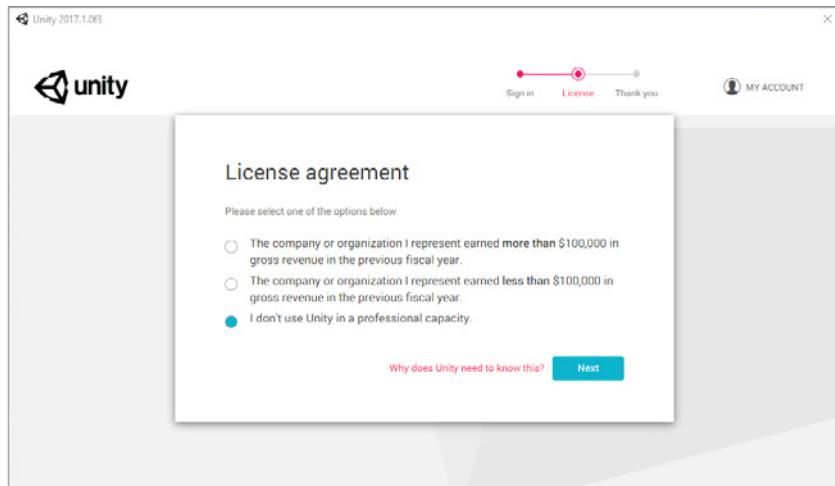


Bild 3.7 Die Unity-Umfrage beim ersten Anwendungsstart

Nach einem kurzen Moment kannst du mit einem Klick auf **START USING UNITY** den Einrichtungsassistenten abschließen und gelangst in die Projekt-Auswahl.

■ 3.3 Ein neues Projekt anlegen

Jedes Mal, wenn du Unity startest, gelangst du zunächst in die Projekt-Auswahl. Hier siehst du immer die Projekte, die du zuletzt geöffnet hast. Du kannst über die **OPEN**-Schaltfläche außerdem weitere Projekte von deiner Festplatte laden. Wir wollen an dieser Stelle aber ein neues Projekt anlegen. Klicke dazu auf die **NEW**-Schaltfläche.

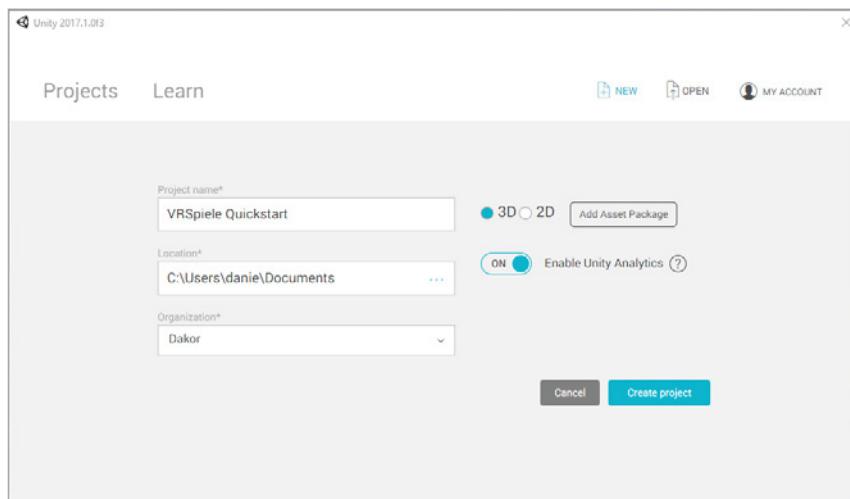


Bild 3.8 Die Projekt-anlegen-Ansicht

Beim Anlegen eines neuen Projektes kannst du folgende Einstellungen vornehmen:

Der **Projektname** wird zum einen für den Projektordner auf deiner Festplatte und zum anderen als Anwendungstitel für dein Spiel verwendet. Beides kann aber zukünftig noch unbenannt werden.

Die **Location** gibt an, wo das Projekt auf deiner Festplatte gespeichert werden soll. Hier solltest du beachten, dass der Ordner im Laufe der Entwicklung durchaus sehr groß werden kann. Während der Ordner am Anfang nur wenige MB groß ist, sind Projektordner zwischen 3 und 4 GB keine Seltenheit.

Die **Organization** lässt du einfach auf dem Standardwert, diese Einstellung wird erst relevant, wenn du in einer Firma und einem größeren Team arbeitest. Ebenso lässt du den Schalter für die Projektart auf **3D**.

Bei der Option **Enable Unity Analytics** musst du dir überlegen, ob du Unitys Analyse-Tool verwenden möchtest. Damit erhältst du Einsicht in Statistiken, die Unity über dein Projekt gesammelt hat.

Wenn du alles nach deinem Wunsch eingestellt hast, wähle **CREATE PROJECT**.

■ 3.4 Die Unity-Oberfläche



Bild 3.9 Die Standardansicht des Unity Editors

Die Standardansicht des Unity Editors kann in fünf Bereiche aufgeteilt werden. Als Erstes findest du hier eine kurze Übersicht, bevor wir uns jeden Bereich nochmals im Detail ansehen:

1. Project- und Console-Bereich

Dieser Bereich hat zwei Reiter, zum einen *Project* und zum anderen *Console*. Standardmäßig ist der Reiter *Project* aktiviert. Dies ist die Projektansicht, in der alle Dateien, die zu deinem Projekt gehören, angezeigt werden. Der Bereich wird auch *Project Browser* genannt.

In dem *Console*-Reiter werden, während du dein Spiel testest, Debug-Informationen, Warnungen und Fehler angezeigt.

2. Hierarchy

Hier werden alle *GameObjects* der aktuellen Szene angezeigt. Jedes Modell, jedes Licht und jede Kamera erhalten hier einen entsprechenden Eintrag und können hier wiedergefunden und ausgewählt werden. Über *Drag and Drop* kannst du *GameObjects* in diesem Fenster umsortieren und hierarchisch sortieren.

3. Scene-, Game- und Assets-Store-Bereich

Dieser Bereich hat standardmäßig drei Reiter.

Der *Scene*-Reiter ist dabei der wichtigste von den dreien. Hier erhältst du in Echtzeit eine Vorschau deiner aktuellen Szene. Diese Ansicht wird auch *Scene View* genannt.

Die *Game*-Ansicht zeigt dir, während du dein Spiel im Editor testest, die Perspektive, wie sie auch der Spieler im fertigen Spiel haben würde. Diese Ansicht wird auch *Game View* genannt.

Der *Asset Store* ist ein integrierter Store, in dem du Modelle, Sounds und Plug-ins herunterladen und kaufen kannst.

4. Toolbar-/Werkzeug-Bereich

Dieser Bereich enthält zum einen die klassische Toolbar (*File*, *Edit* etc.) und zum anderen die Unity-Toolbar.

5. Inspector-Bereich

Der *Inspector* ist ein dynamischer Bereich, dessen Inhalt sich immer dem anpasst, was du gerade tust. Hast du in der *Scene View* oder *Hierarchy* ein *GameObject* ausgewählt, zeigt er dir zum Beispiel detaillierte Informationen zu dem *GameObject* an. In diesem Bereich werden jedoch auch diverse Einstellungsfenster angezeigt, wenn du sie über die Toolbar aufrufst. In dem *Inspector*-Bereich gibt es noch den Reiter *Services*, den du für den Anfang allerdings erst einmal nicht benötigst.

Das Layout des Editors kannst du nach deinen Wünschen anpassen, indem du die einzelnen Registerkarten einfach mittels *Drag and Drop* in einen anderen Bereich ziehst. Ich werde mich in diesem Buch jedoch stets auf das Standardlayout beziehen.

3.4.1 Drag & Drop

Damit es nicht zu Missverständnissen kommt, hier nochmals eine kurze Bergriffserklärung: Drag & Drop oder auf Deutsch „Ziehen und Ablegen“ ist eine der wichtigsten Bedienmethoden in Unity. Der Begriff beschreibt die Bedienmethode, die du zum Beispiel auch in *Microsoft Windows* oder *Apples MacOS* nutzt, um Dateien und Fenster mit der Maus zu verschieben: Du klickst mit der linken Maustaste auf das Objekt, das du verschieben möchtest,

und hältst die linke Maustaste gedrückt. Jetzt ziehst du das Objekt mit gedrückter linker Maustaste in den Bereich oder auf das Feld, in dem du es ablegen bzw. loslassen möchtest. Wenn ich in diesem Buch zum Beispiel schreibe „Ziehe die Grafik auf das angezeigte Feld“, meine ich damit, du sollst die jeweilige Grafik mit der oben beschriebenen Methode auf das jeweilige Feld ziehen und loslassen.

3.4.2 Project Browser

Im *Project Browser* findest du alle Assets, die zu deinem Projekt gehören. Die Ansicht ist in zwei Bereiche aufgeteilt: Links siehst du die Ordnerstruktur deines Projektes. Darüber befinden sich, mit gelben Icons versehen, ein paar Shortcuts für häufig verwendete Suchanfragen. Wenn du ein bestimmtes Asset in deinem Projekt suchst, kannst du es über die Suchleiste, welche sich in der oberen rechten Ecke des *Project Browsers* befindet, bequem suchen und finden. Tippe dazu einfach einen Teil des Namens in das Suchfeld ein und der rechte Teil des *Project Browsers* wird dir alle passenden Dateien anzeigen. Du musst für eine Suche nicht den Anfang des Dateinamens kennen. Wenn du zum Beispiel nach *up* suchst, wird auch **Pickup** oder **LookUpArrow** gefunden.

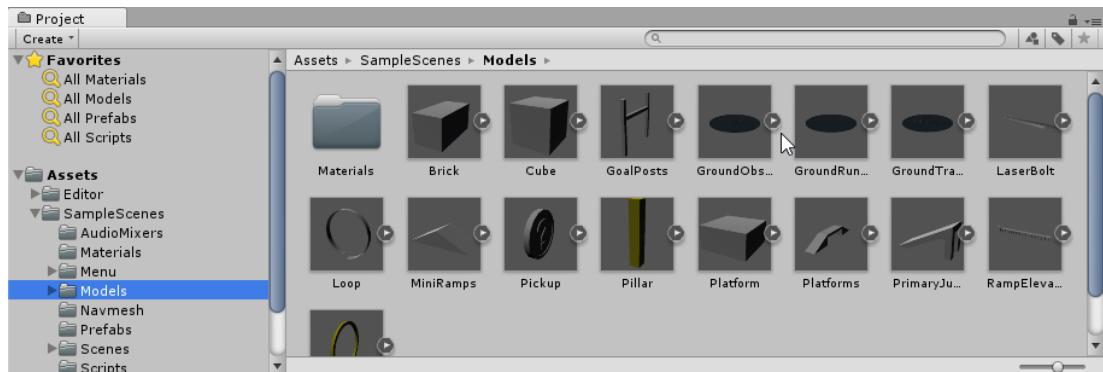


Bild 3.10 Der Project Browser

Im Project Browser kannst du zusätzlich auch nahezu jede Art von Asset anlegen. Wähle dazu links den gewünschten Zielordner aus und öffne mit der rechten Maustaste das *Kontextmenü* für den jeweiligen Ordner. Alternativ kannst du auch im rechten Bereich an einer freien Stelle das Kontextmenü öffnen. Im Kontextmenü findest du dann ganz oben den Eintrag **CREATE**. Wählst du ihn aus, siehst du ein Auswahlmenü mit allen Assets-Typen, die du anlegen kannst. Über dieses Menü kannst du auch, wie in Bild 3.11 zu sehen, weitere Ordner (*Folder*) anlegen, um dein Projekt besser zu sortieren.

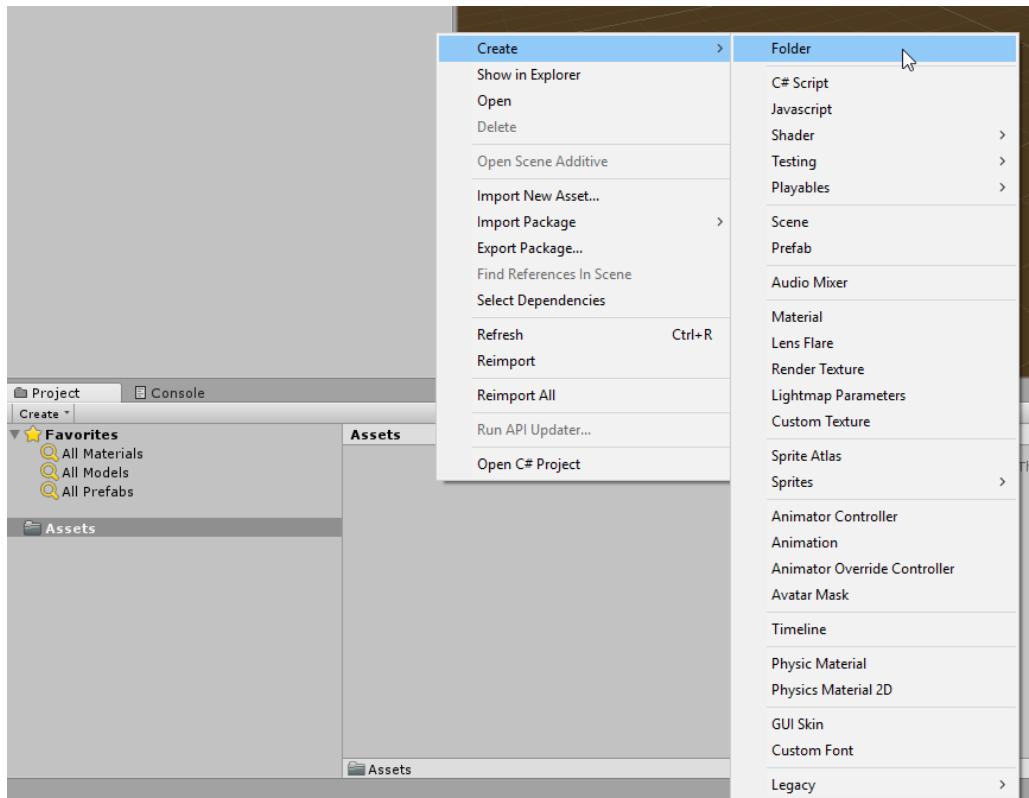


Bild 3.11 Neue Assets im Project Browser anlegen

3.4.3 Console

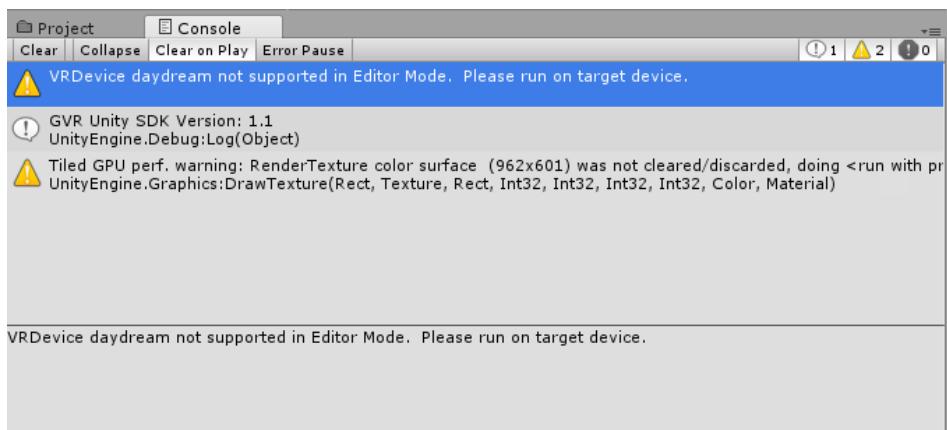


Bild 3.12 So sieht die Console im Unity Editor aus.

Die *Console* findest du in der Standardansicht als einen kleinen Reiter neben dem *Project Browser*. Du kannst über die Reiter jederzeit zwischen Console und Project Brower hin und her wechseln. In der Console werden hilfreiche Informationen sowie Warnungen und Fehlermeldungen angezeigt, wenn etwas nicht funktioniert, wie es soll. *Scripts* können hier jederzeit sogenannte *Debug-Logs* hineinschreiben. Diese Ausgaben dienen dazu, Fehler in der Anwendung (*Bugs*) zu finden. Diese Debug-Informationen können Aufschluss darauf liefern, wann und warum ein Fehler aufgetreten ist.

Über die Schaltflächen oben rechts in dem *Console*-Fenster kannst du die Ausgaben nach ihrem Typ filtern. Jede Schaltfläche bestimmt jeweils, ob ein bestimmter Ausgabe-Typ angezeigt wird oder nicht. Von links nach rechts sind das die Typen: *Debug*, *Warning* und *Error or Exception*. Dies wird später bei komplizierten *Scripts* mit vielen Ausgaben noch interessant. Standardmäßig sind alle Typen aktiviert.

Wie du eigene *Debug-Logs*, *Warnings* und *Errors* in die Console schreibst, werden wir uns in einem späteren Kapitel noch ansehen. Derzeit musst du dir nur merken, wenn etwas schief geht, findest du in der *Console* häufig genauere Informationen darüber.

3.4.4 Hierarchy

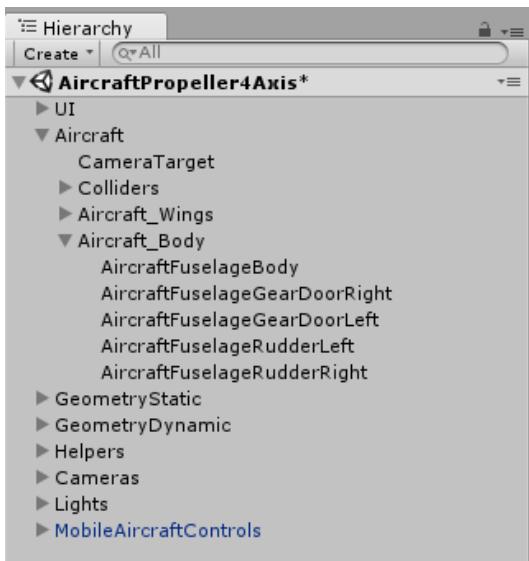


Bild 3.13 Die Hierarchy einer Beispiel-Scene mit einem Flugzeug

Die *Hierarchy* zeigt alle *GameObjects* in der aktuellen *Scene* an. Mit der linken Maustaste kannst du *GameObjects* in der *Hierarchy* markieren. Hältst du dabei STRG gedrückt, kannst du mehrere Objekte gleichzeitig auswählen. Die in der *Hierarchy* markierten Objekte werden in der *Scene View* farblich hervorgehoben markiert und im *Inspector* findest du Details zu dem ausgewählten Objekt.

Ziehst du ein 3D-Modell aus dem *Project Browser* in die *Scene View*, wird automatisch ein entsprechendes GameObject angelegt, welches bereits alle *Components* besitzt, damit das 3D-Modell in der Scene angezeigt wird. Das dadurch angelegte GameObject siehst du dann automatisch auch in der *Hierarchy*.

Oben links in dem *Hierarchy*-Fenster befindet sich das **CREATE**-Menü. Dieses Menü erlaubt dir, eine Vielzahl von vorkonfigurierten GameObjects zu erstellen. Dazu gehören zum Beispiel einfache geometrische Formen wie Würfel oder Kugeln, aber auch Lichter, Audio-Quellen und Kameras.

Ähnlich wie im *Project Browser* kannst du auch in der *Hierarchy* nach einzelnen Objekten suchen, indem du einen Teil des Namens in die Suchmaske oben rechts in der *Hierarchy* eingibst.



Mit einem Doppelklick auf ein GameObject in der Hierarchy wird die Kamera in der Scene View automatisch so verschoben, dass das ausgewählte Objekt zu sehen ist.

3.4.4.1 Parenting

Parenting, oder auch „Eltern-Kind-Beziehung“, ist in Unity ein wichtiges Konzept zum Erstellen von Szenen. Du kannst jedem GameObject beliebig viele andere GameObjects als Kinder zuweisen. Das übergeordnete GameObject nennt man dann *Parent* oder *Eltern-Objekt* und die untergeordneten GameObjects *Children* oder *Kinder*. Alle Kinder sind abhängig von ihrem *Parent*. Das bedeutet, wenn sich der Parent bewegt, dreht oder seine Größe ändert, ändern sich die Kinder entsprechend auch. Die Kinder können sich aber zusätzlich noch, immer relativ zu dem *Parent*, selber bewegen, drehen oder skalieren.

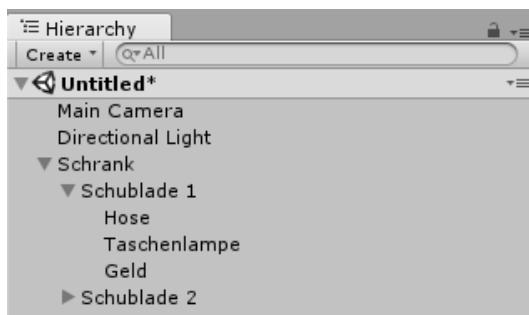


Bild 3.14 Ein Beispiel für Eltern-Kind-Beziehungen

Als Beispiel kann man sich einen Schrank mit Schubladen vorstellen, wie er in der Hierarchy in Bild 3.14 dargestellt wird. Der „Schrank“ ist das Eltern-Objekt. Die Schubladen sind jeweils Kinder des Schrankes und die Gegenstände in den Schubladen sind Kinder der Schubladen. Wenn du den Schrank in eine andere Ecke schiebst, bewegen sich alle Schubladen und deren Inhalte mit, sodass ihre Position relativ zu ihrem jeweiligen Eltern-Objekt gleich bleibt. Diese relative Position nennt sich *lokale Position*. Ebenso gibt es auch eine

lokale Rotation, welche die Rotation von den Kindern zu den Eltern beschreibt. Das Gegenstück zu diesen lokalen Werten sind die *globalen Positionen und Rotationen*. Die globale Position der Schublade berechnet sich aus der Position des Schrankes plus der lokalen Position der Schublade.

Während Änderungen am Parent sich immer auf alle Kinder auswirken, können die Kinder ihre lokale Position und Rotation ändern, ohne dass es das Eltern-Objekt beeinflusst. In unserem Beispiel schauen wir uns zur Veranschaulichung das Öffnen der Schubladen an: Öffnet man eine Schublade, verändert sich ihre lokale Position, da sie ihren Abstand zum Schrank ändert. Der gesamte Inhalt der Schubladen bewegt sich mit der Schublade mit. Durch diese Änderung an den Kindern des Schrankes bleibt die Position des Schrankes aber natürlich unverändert. Genauso können sich die Gegenstände in der Schublade hin und her bewegen, ohne dass sich die Schublade bewegt. Bewegt man aber die Schublade, bewegen sich auch die Gegenstände darin.

In Unity kannst du einem Objekt ein Kind-Objekt zuweisen, indem du in der *Hierarchy* ein *GameObject* mittels Drag & Drop auf das gewünschte Eltern-Objekt ziehst. In der *Hierarchy* werden die *GameObjects* dann wie in Bild 3.14 verschachtelt dargestellt.

Parenting verwendet man in Unity für bewegliche Objekte, wenn man andere Objekte auf eine einfache Art und Weise an ihnen befestigen möchte oder wenn Objekte ein Teil von etwas anderem sind. Außerdem verwendet man Parenting häufig auch, um Ordnung in der *Hierarchy* zu halten: Zum Beispiel kann man alle Lichter unter einem *GameObject* „Lights“ gruppieren, um sie schnell wiederzufinden.

3.4.5 Scene View

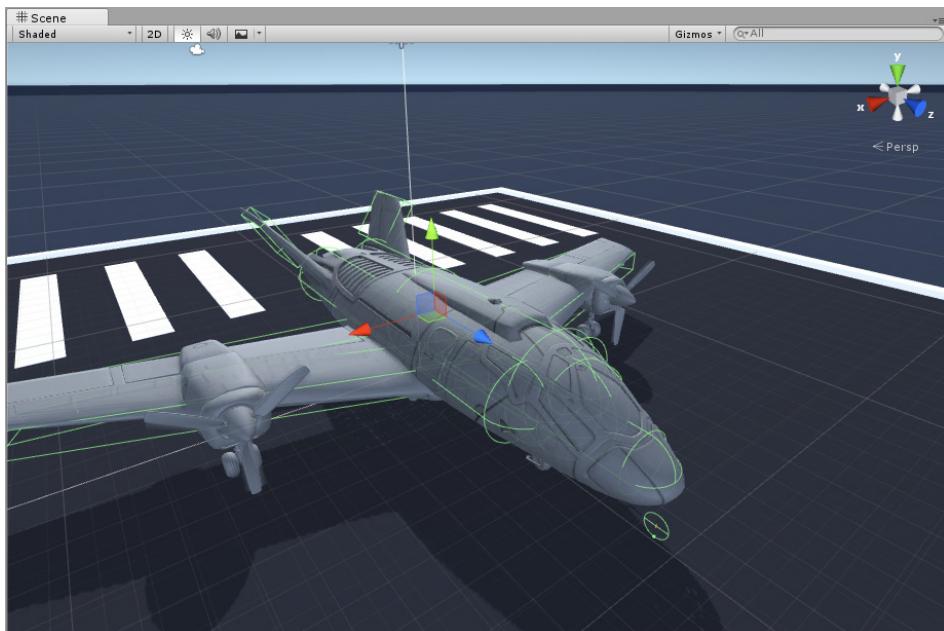


Bild 3.15 Die Szenen-Ansicht oder auch Scene View

Das große Fenster in der Mitte, welches die Beschriftung „Scene“ trägt, nennt sich *Scene View*. Dieses Fenster ist eines der wichtigsten Fenster im Unity Editor. Es zeigt dir eine Echtzeit-Ansicht von deinem Level (in Unity *Scene* genannt).

Hältst du die **RECHTE MAUSTASTE** gedrückt, kannst du mit der **MAUS** und **W, S, A, D** in der *Scene* herumfliegen. Die Tasten **E** und **Q** erlauben es dir, senkrecht hoch und runter zu fliegen. Über einen Klick mit der linken Maustaste kannst du in der *Scene View* Objekte markieren. Die markierten *GameObjects* werden dann auch in der *Hierarchy* markiert.

In der *Scene View* kannst du das ausgewählte *GameObject* über die *Transform-Tools* verschieben, drehen und skalieren.

3.4.5.1 Transform-Tools

Unity bietet dir unterschiedliche Werkzeuge an, mit denen du ein ausgewähltes *GameObject* verändern kannst. Diese Werkzeuge findest du links in der *Unity-Toolbar*.



Bild 3.16 Die Transform-Tools

Von links nach rechts gibt es folgende Werkzeuge:

1. **Hand-Tool** – Mit diesem Werkzeug kannst du die Perspektive in der *Scene View* verschieben. Es zählt eigentlich nicht zu den sogenannten *Transform-Tools*, befindet sich aber in der gleichen Menü-Gruppe.
2. **Translate-Tool** – Das Kreuz aus Pfeilen aktiviert das Verschieben-Werkzeug. Ist dieses Werkzeug aktiv, werden an dem ausgewählten *GameObject* in der *Scene View* Pfeile angezeigt, mit denen du das *GameObject* verschieben kannst.
3. **Rotate-Tool** – Die beiden Pfeile aktivieren das Rotations-Werkzeug. Bei diesem Werkzeug werden in der *Scene View* mehrere Kreise angezeigt, mit denen du das ausgewählte *GameObject* drehen kannst.
4. **Scale-Tool** – Die Schaltfläche mit dem Quadrat und den vier Pfeilen aktiviert das Skalierungs-Werkzeug. Dieses Werkzeug erlaubt dir, *GameObjects* in der *Scene View* zu vergrößern oder zu verkleinern.
5. **Rect-Tool** – Diese Schaltfläche ist ein Werkzeug, das vor allem für *User Interfaces* (also Menüs etc.) verwendet wird. Es erlaubt dir, Bilder, Schaltflächen und andere Elemente auf einer zweidimensionalen Ebene zu verschieben und in der Größe zu ändern. Dabei docken die Elemente an allen passenden Kanten an. Dieses Werkzeug erlaubt dir daher, aufgeräumt und einheitlich aussehende *User Interfaces* zu erstellen.

3.4.5.1.1 Lokale und globale Änderungen

Je nach Werkzeug werden dir unterschiedliche Helfer (*Handels*) an dem ausgewählten *GameObject* angezeigt, mit denen du das *GameObject* verändern kannst. Klickst du in der *Unity-Toolbar* auf die Schaltfläche **LOCAL**, ändert sich ihre Beschriftung zu **GLOBAL**. Damit kannst du umstellen, ob sich das Werkzeug an den lokalen oder globalen Achsen orientieren soll. Klickst du erneut auf dieselbe Schaltfläche, schaltest du wieder zu **LOCAL** um.

- **Local** ist am besten dann geeignet, wenn du das ausgewählte GameObject innerhalb des *Parent Objects* oder relativ zu den aktuellen Werten verändern möchtest. In den meisten Fällen ist dies die Option, die du verwenden möchtest.
- **Global** erlaubt es dir hingegen, ein Objekt relativ zu den globalen Achsen zu verändern. Die Änderung ist dann also unabhängig von seiner derzeitigen Position und Rotation. Die globalen Achsen kannst du stets mithilfe des Hilfselementes oben rechts in der *Scene View* ansehen.

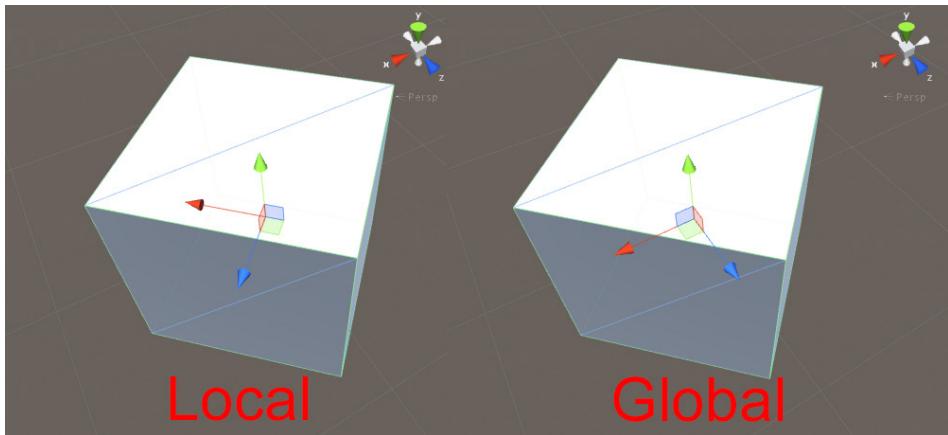


Bild 3.17 Der Unterschied zwischen Local und Global

3.4.5.1.2 Snapping

Mithilfe der Taste **STRG** auf deiner Tastatur kannst du das sogenannte *Snapping* aktivieren. Wenn du diese Taste gedrückt hältst und dabei eines der Transform-Tools verwendest, verändert sich das ausgewählte GameObject in festen Schritten. Möchte man mehrere GameObjects in einer Reihe anordnen oder exakt um 90° drehen, hilft diese Funktion enorm.

3.4.6 Inspector

Der *Inspector* ist das lange Fenster rechts im Unity Editor. Dieses kontextsensitive Fenster ist der Ort, wo du so gut wie jede Einstellung vornehmen wirst. Je nach Kontext, also je nachdem, was du zuletzt angeklickt hast, passt sich der Inhalt dieses Fensters entsprechend an. Es ist also egal, ob du ein GameObject verändern oder Einstellungen für den Editor selbst bearbeiten möchtest, in den meisten Fällen erscheinen die benötigten Optionen in diesem Fenster. Am häufigsten wirst du den *Inspector* aber nutzen, um *GameObjects* und ihre *Components* zu bearbeiten, weshalb wir uns diesen Fall nochmals genauer ansehen:

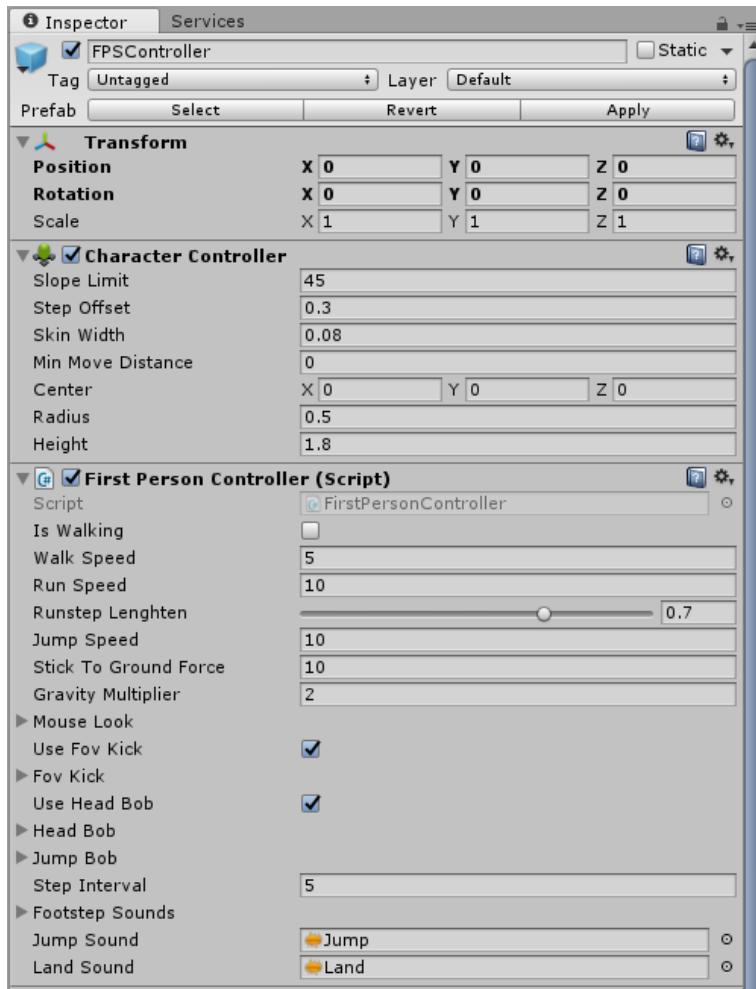


Bild 3.18 Der Unity Inspector zeigt dir immer die Einstellungen, die du gerade brauchst.

In Bild 3.18 siehst du beispielhaft, wie der *Inspector* aussieht, wenn du ein GameObject ausgewählt hast. In diesem Beispiel wurde ein GameObject mit dem Namen *FPSController* ausgewählt, was eine Spielfigur ist, die durch den Spieler gesteuert werden kann (*First Person Shooter Controller*). Das GameObject besitzt verschiedene Components, welche diese Funktionen ermöglichen. In Bild 3.18 sind zwei Components sichtbar: *Character Controller* und *First Person Controller*.

Components erlauben es dir in der Regel, einige ihrer Eigenschaften im *Inspector* anzupassen, sodass sie nicht nur für eine bestimmte Situation verwendet werden können. Das Ziel ist es, das Component so flexibel zu gestalten, dass es möglichst häufig wiederverwendet werden kann, ohne dass man Änderungen vornehmen muss.

Im *Inspector* kannst du die *Components* eines GameObjects dann konfigurieren: Möchtest du die Laufgeschwindigkeit ändern, musst du nicht das *First Person Controller*-Skript bear-

beiten, sondern änderst einfach im *Inspector* die Eigenschaft *Walk Speed*. Genauso kannst du zum Beispiel die Größe dieser Spielfigur einfach in dem *Character Controller* anpassen.

3.4.6.1 Inspector Felder

Die meisten Components können über den *Inspector* konfiguriert werden. Welche Eigenschaften im *Inspector* geändert werden können, ist im *Component* selbst gespeichert und wurde von dem jeweiligen Entwickler bestimmt. Die Bedienelemente sehen dabei immer ähnlich aus und werden durch einen Schriftzug (*Label*) beschrieben.

Was du genau mit den jeweiligen Eigenschaften änderst und welche Werte Sinn ergeben, sollte das Label oder die Dokumentation des *Components* deutlich machen. Teilweise kannst du mit der Maus über das Label fahren, um einen Hinweis (*Tooltip*) zu erhalten.

Standardmäßig gibt es gewisse Arten von Eingabefeldern, welche du immer wieder sehen wirst. Ich stelle dir hier deshalb erst einmal die gängigsten vor, es gibt jedoch noch mehr. In Bild 3.19 siehst du die Bedienelemente, die du im *Inspector* am häufigsten sehen wirst.

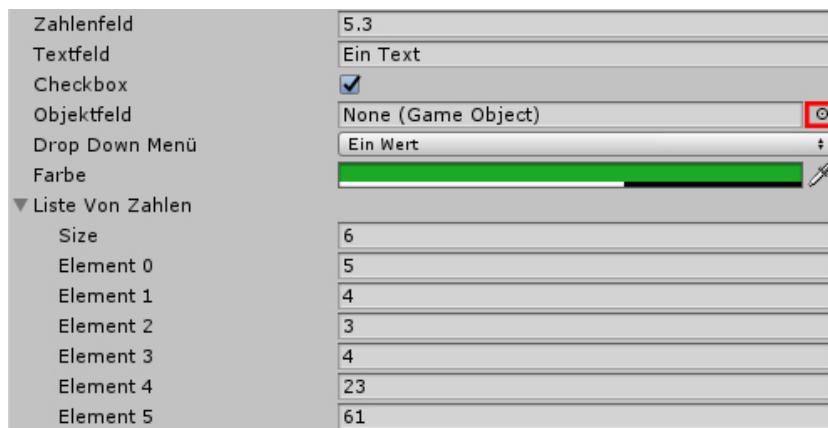


Bild 3.19 Das sind die am meisten vorkommenden Bedienelemente im *Inspector*.

- **Zahlfeld:** Hier kannst du der Eigenschaft eine beliebige Zahl zuweisen. Zahlfelder können manchmal auch auf ganze Zahlen beschränkt sein oder über einen Schieberegler verfügen.
- **Textfeld:** Hier kannst du einen beliebigen Text eintragen, der dann von dem *Component* benutzt wird. Die Textlänge ist nicht auf die Breite des Textfeldes im *Inspector* beschränkt. Zeilenumbrüche sind aber nicht möglich.
- **Checkbox:** Hiermit kannst du eine bestimmte Eigenschaft aktivieren oder deaktivieren.
- **Objektfeld:** Objektfelder sind universale Felder, in denen Verweise (auch *Referenzen* genannt) gespeichert werden können. Objektfelder sind meist auf einen bestimmten Typ beschränkt: In Bild 3.19 können dem Feld zum Beispiel nur *GameObjects* zugewiesen werden. Das *GameObject* kann dabei entweder aus deiner aktiven Scene oder aus deinen gespeicherten Assets stammen. Für das Zuweisen eines Wertes hast du zwei Möglichkeiten: Entweder du ziehst das gewünschte Objekt mittels *Drag and Drop* aus der *Hierarchy* oder dem *Project Browser* auf das Feld oder du verwendest das *Auswahlmenü*. Das Aus-

wahlmenü öffnest du über den in Bild 3.19 rot markierten kleinen Kreis rechts neben dem Feld. Das Auswahlmenü zeigt dir dann, wie in Bild 3.20, eine Liste aller zu der Einschränkung passenden Objekte an. In dem Menü kannst du über die zwei Reiter **SCENE** und **ASSETS** auswählen, ob es dir Objekte aus der aktuellen *Scene* oder dem *Project Browser* anzeigen soll.

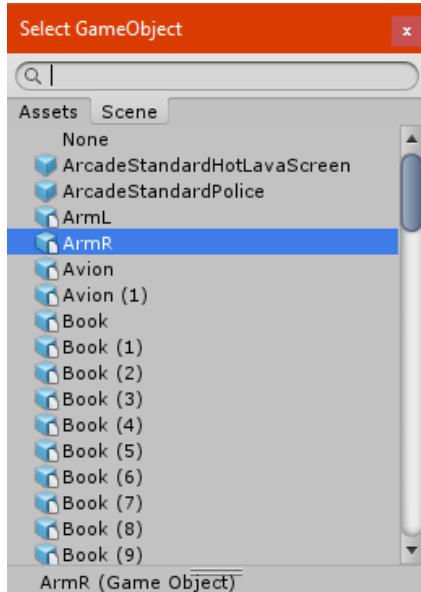


Bild 3.20 Hier kannst du ein GameObject aus deiner Scene oder deinen Assets auswählen.

- **Drop-down-Menü:** Hier hast du die Möglichkeit, aus einer Auswahl von Optionen den gewünschten Wert auszuwählen.
- **Farbe:** Bei diesem Feld kannst du eine Farbe angeben. Wenn du auf das Feld rechts neben dem *Label* klickst, öffnet sich ein neues Fenster mit einer Farbpalette und der Möglichkeit, eine Farbe über Zahlenwerte anzugeben. Das Fenster hat die Überschrift „Color“, wird aber meist als *Color-Picker* (dt. „Farbwähler“) bezeichnet.

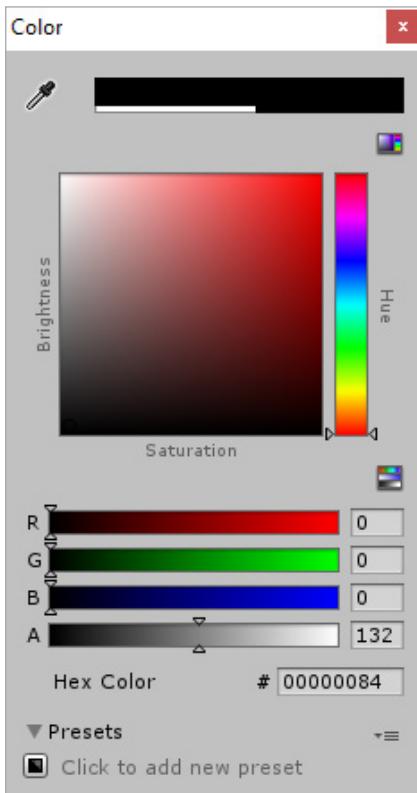


Bild 3.21 Der „Color-Picker“ von Unity erlaubt das Auswählen einer Farbe.

Jede Farbe hat in Unity vier Kanäle, aus denen sie gemischt wird: Rot, Grün, Blau und Alpha. Der Alpha-Wert bestimmt, wie transparent die Farbe ist. Ein Alpha-Wert von 0 würde bedeuten, die Farbe ist vollkommen durchsichtig. Der maximale Wert für jeden Kanal ist 255. Wenn du in der Farbpalette eine Farbe auswählst, kannst du in den Boxen darunter die Farbmischung für die einzelnen Kanäle ablesen. Über die Pipette oben links kannst du eine beliebige Farbe, die gerade auf deinem Monitor zu sehen ist, kopieren.

- **Liste:** Siehst du vor einem Label einen kleinen Pfeil, kannst du darauf klicken, um verborgene Elemente einzublenden. Bei Listen gibt das erste Feld im *Inspector* dann immer die Größe der Liste an. Wenn du mehr oder weniger Einträge benötigst, kannst du die Zahl hinter dem *Size-Label* ändern. Direkt darunter findest du die einzelnen Einträge der Liste. In Bild 3.19 sind es zum Beispiel fünf Zahlen. Es gibt nicht nur Zahlenlisten: Listen existieren für nahezu jede Eigenschaften-Art: Listen von Texten, Zahlen, Checkboxen oder auch Objekten sind zum Beispiel ebenfalls möglich.

3.4.6.2 Unity-Handbuch – so hilfst du dir selbst

Wie ich schon in der Einleitung erklärt habe, werde ich dir nicht jedes einzelne *Component* und jede einzelne Einstellung im Detail erklären, sondern mich auf die wichtigen beschränken. Sollten dir meine Beschreibungen nicht ausreichen, kannst du jederzeit einfach im

offiziellen *Unity-Handbuch* nachschauen. Jedes offizielle Unity Component hat im *Inspector* eine Verknüpfung zum passenden Handbucheintrag. Dort werden alle Funktionen und Eigenschaften in der Regel ausführlich beschrieben.

Um das Handbuch für ein bestimmtes Component schnell zu öffnen, kannst du einfach auf das blaue Hilfe-Icon, rechts neben dem *Component*-Namen, klicken. In Bild 3.22 ist das Hilfe-Icon rot markiert. Das Handbuch existiert derzeit nur in Englisch, kann aber auch offline aufgerufen werden, wenn du es bei der Installation nicht deaktiviert hast.



Bild 3.22 Die Verknüpfung zum Handbuch an einem Transform-Component

3.4.6.3 Components zu einem GameObject hinzufügen

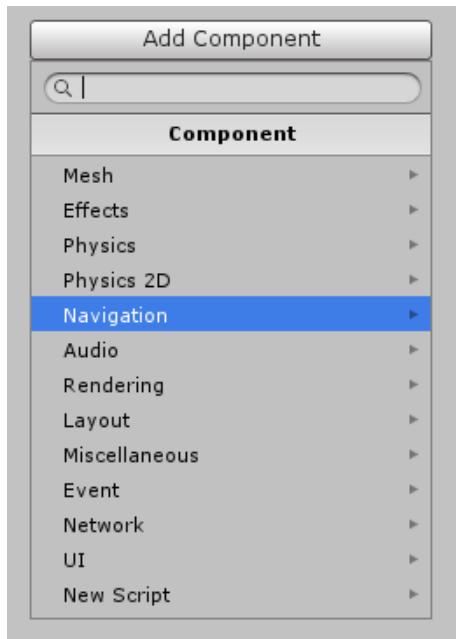


Bild 3.23 Das „Add Component“-Menü des *Inspectors*

Im *Inspector*, unter dem letzten vorhandenen *Component*, findest du die **ADD COMPONENT**-Schaltfläche. Hierüber kannst du weitere *Components* hinzufügen. Wenn du auf die Schaltfläche klickst, öffnet sich ein Menü, in welchem du aus verschiedenen Kategorien wählen kannst. In jeder der Kategorien befinden sich dann diverse passende Components. Wenn du einen Teil des Namens des Components kennst, kannst du ihn auch einfach in das Suchfeld oben in dem Menü eingeben. Die Liste aller Components wird dann entsprechend gefiltert. Wenn es sich bei dem *Component* um ein *Script* handelt, kannst du es auch mittels Drag & Drop aus dem *Project Browser* in den *Inspector*-Bereich ziehen.

3.4.6.4 Components von einem GameObject entfernen, kopieren und einfügen

Um ein Component wieder von einem GameObject zu entfernen, musst du mit der rechten Maustaste auf den Namen des Components im *Inspector* klicken. In dem dann erscheinenden Kontextmenü musst du **REMOVE COMPONENT** wählen.

In dem gleichen Kontextmenü findest du auch die Optionen **COPY COMPONENT** und **PASTE COMPONENT AS NEW**, mit denen du Components inklusive ihrer Konfiguration kopieren und auf demselben oder auf anderen GameObjects einfügen kannst.

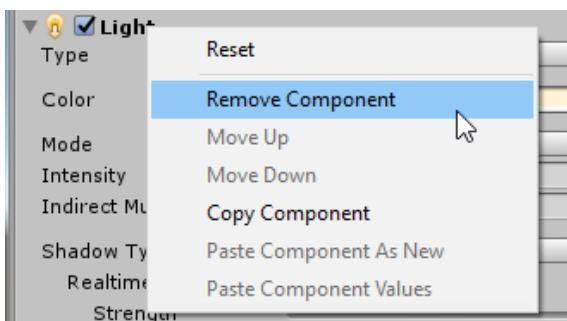


Bild 3.24 Das Component-Kontextmenü

■ 3.5 Scenes

Unity arbeitet mit sogenannten „Scenes“. Am besten stellst du dir jede *Scene* als ein Level deines Spiels vor. Gibt es ein Wüsten-Level und ein Schnee-Level, gäbe es zum Beispiel eine Wüsten-Scene und eine Schnee-Scene. Im Vergleich mit Mehrspieler-Spielen kannst du dir jede *Scene* auch als eine *Map* des Spiels vorstellen, die über einen Ladebildschirm geladen werden muss.

3.5.1 Unity in der dritten Dimension

Falls du dich in deiner Schulzeit jemals gefragt hast, wann man im Leben Vektorrechnung braucht, hast du hier die Antwort: wenn man Videospiele entwickeln möchte.

Mit Unity können sowohl 2D- als auch 3D-Spiele entwickelt werden. Wenn du ein Virtual-Reality-Spiel entwickelst, arbeitest du aber grundsätzlich in einem 3D-Koordinatensystem. Das bedeutet, ein beliebiger Punkt in deiner *Scene* wird durch drei Koordinaten beschrieben: *x*, *y* und *z*. Wie in Bild 3.25 dargestellt, beziehen sich in Unity die *x*- und *z*-Koordinaten auf die Position in der horizontalen Ebene und die *y*-Koordinate auf die vertikale Position.

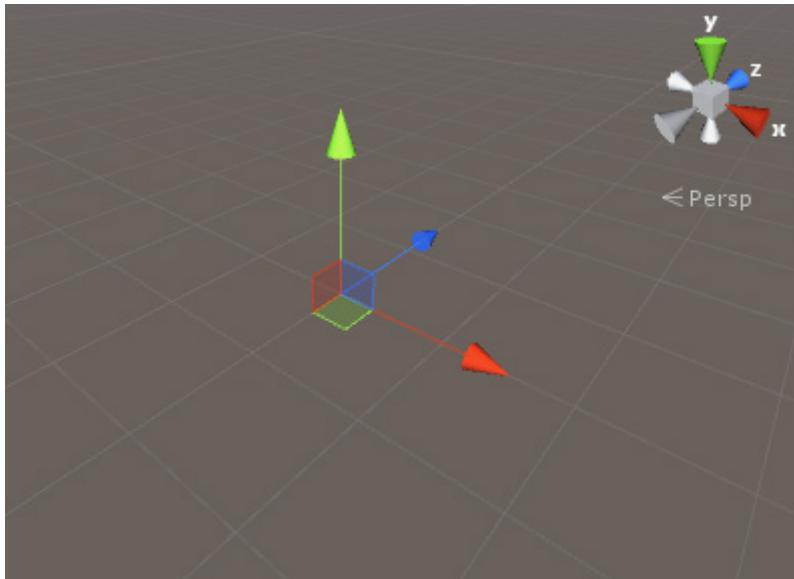


Bild 3.25 Die Pfeile zeigen jeweils in die positive Richtung der Achse.

Ich werde in diesem Buch für Vektoren die Schreibweise (x, y, z) verwenden. Der Vektor $(3, 6, 8)$, bedeutet beispielsweise: x ist 3, y ist 6 und z ist 8.

Ein Vektor muss nicht zwingend eine Position im Raum beschreiben. In Unity werden Vektoren unter anderem auch zur Angabe von Richtungen (*Richtungsvektor*) sowie Skalierungen und Rotationen verwendet. In diesen Fällen beschreiben die einzelnen Werte des Vektors stets die Veränderung auf der (oder Rotation um die) jeweiligen Achse x , y oder z .

Unity arbeitet mit *Unity-Einheiten*. Im Normalfall ist in Unity aber alles darauf ausgelegt, dass *eine Unity-Einheit* im Welt-Koordinatensystem auch *einem Meter* entspricht. Dadurch fällt es beim Erstellen von *Scenes* deutlich leichter, den richtigen Maßstab einschätzen zu können, als bei einer Software, die sich nicht an realen Maßen orientiert. Ich werde in diesem Buch manchmal „Meter“, manchmal „Einheiten“ verwenden, je nachdem, was gerade passender ist.

3.5.2 GameObjects

GameObjects sind die einzelnen Bestandteile, aus denen sich die *Scenes* zusammensetzen. An und für sich ist ein *GameObject* aber erst einmal nur ein fast leerer Container ohne Funktion. Erst durch das Hinzufügen von *Components* erhalten die *GameObjects* die Fähigkeit, 3D-Modelle, Lichter oder auch den Spieler darzustellen. *GameObjects* können aber auch nur dazu dienen, Spiellogik auszuführen, und müssen nicht unbedingt in der *Scene* sichtbar sein.

Ein neu angelegtes *GameObject* enthält immer ein *Transform*-Component. Das *Transform* enthält Informationen zur Position, Rotation und Skalierung des *GameObjects*. Außerdem

werden im Transform auch die Eltern-Kind-Beziehungen des jeweiligen GameObjects verwaltet. Ein frisches GameObject ist also zunächst nur ein unsichtbares Objekt in der *Scene*, welches an einer bestimmten Stelle liegt, eine bestimmte Größe hat und in eine bestimmte Richtung gedreht ist.

Neben den vielen vorgefertigten Components, mit denen Unity ausgeliefert wird, kannst du auch eigene *Scripts* schreiben und einem GameObject damit eigene, individuelle Funktionen geben. Sowohl die offiziellen als auch die selber erstellten *Components* kannst du über den *Inspector* konfigurieren.

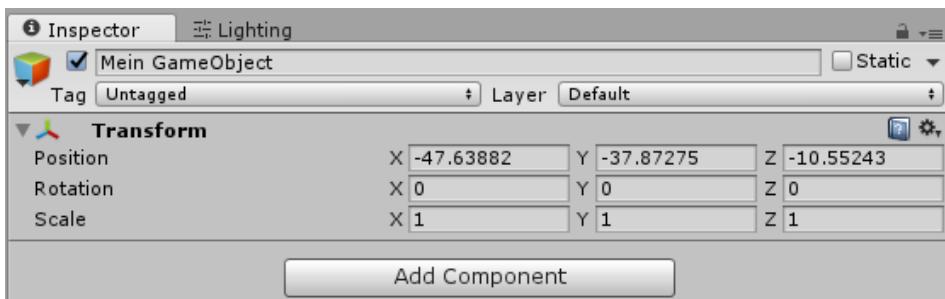


Bild 3.26 Der GameObject-Kopf im Inspector

Im *Inspector* haben GameObjects einen einheitlichen „Kopf“, den du in Bild 3.26 siehst. Unterhalb des Kopfes findest du die jeweiligen *Components* des *GameObjects*, stets beginnend mit dem *Transform*-Component.

Im GameObject-Kopf hast du folgende Einstellungsmöglichkeiten (von links nach rechts):

- **Gizmo** (der Würfel links): Hierüber kannst das *Gizmo-Icon* des GameObjects ändern. Wenn du hier ein anderes Icon auswählst, wird das GameObject in der *Scene View* dauerhaft sichtbar und mit dem gewählten Icon angezeigt.
- **Active State** (die Checkbox links): Mit der Checkbox kannst du das GameObject inaktiv schalten und auch wieder aktivieren. Wenn du das *GameObject* deaktivierst, ist es so, als wäre es vorübergehend gar nicht mehr in der Scene; mit dem Vorteil, dass du es jederzeit wieder aktivieren kannst (auch während das Spiel läuft).
- **Name** (das Textfeld): In dem Textfeld kannst du den Namen des GameObjects lesen und festlegen.
- **Static**: Diese Checkbox gibt an, ob das *GameObject* statisch ist. Statische GameObjects können nur im Editor, aber nicht zur Laufzeit bewegt werden. Statische Elemente wie Gebäude und Bäume sollten stets als *Static* markiert werden, da Unity die Darstellung des GameObjects dann automatisch optimieren kann. Hierzu erfährst du später noch mehr.
- **Static-Optionen**: Der Pfeil rechts neben der *Static*-Checkbox öffnet die *Static-Optionen*. Hier kannst du bestimmen, welche Teile der Unity Engine dieses GameObject als statisch betrachten sollen. (Nutzt du die Checkbox, ist die Funktion automatisch für alle Teile aktiviert.)

Beispiel: Ist nur die Option *Lightmap Static* aktiviert, bedeutet das für die Licht-Engine in Unity, dass dieses GameObject statisch ist, und sie berechnet eine statische *Lightmap* für das GameObject. Der Clou: Da alle anderen Teile der Engine das GameObject nicht als

statisch ansehen, kann das *GameObject* trotzdem noch bewegt werden, während das Spiel läuft. Die Lichtdarstellung auf dem *GameObject* ändert sich dann jedoch nicht.

- **Tag:** Mit Tags kannst du besondere Spielelemente markieren bzw. klassifizieren, um sie später leichter wiederzufinden oder ein *GameObject* von anderen unterscheiden zu können. Das wichtigste und bekannteste Tag ist das *Player*-Tag, mit dem das Spieler-*GameObject* markiert wird.
- **Layer:** Hier kannst du dem ausgewählten *GameObject* einen *Layer* (dt. „Ebene“) zuordnen. Du kannst dann an einer anderen Stelle beispielsweise einstellen, dass bestimmte Lichtquellen alle *GameObjects* auf einem bestimmten *Layer* nicht anstrahlen sollen; oder dass *GameObjects* auf *Layer A* nicht mit *GameObjects* auf *Layer B*, aber mit Objekten auf dem *Default-Layer* zusammenstoßen können.

3.5.3 Prefabs

Nicht jedes *GameObject* möchte man per Hand anlegen und verwalten. Häufig benötigt man viele Kopien eines identischen Objektes, die höchstens minimal voneinander abweichen. Für diesen Fall kannst du ein fertig konfiguriertes *GameObject* inklusive seiner Kinder und mit allem Drum und Dran in einem *Prefab* speichern.

Das gespeicherte *Prefab* findest du dann in deinem *Project Browser*, von wo aus du es beliebig oft in die *Scene* ziehen kannst. Jede Instanz des *Prefabs* entspricht dann einer exakten Kopie des Originals. Wenn du das *Prefab* nachträglich bearbeitest, werden die Änderungen auf alle bereits in der *Scene* vorhandenen Instanzen übernommen. Zusätzlich hast du die Möglichkeit, jedes durch ein *Prefab* erzeugte *GameObject* nach dem Anlegen noch individuell im *Inspector* anzupassen. Wenn du eine Eigenschaft änderst, wird diese dann nicht mehr durch das *Prefab* kontrolliert und das dazugehörige Label wird **fett** dargestellt. Änderst du denselben Wert jetzt in dem *Prefab*, wird der Wert bei *diesem* *GameObject* nicht überschrieben. Du kannst Änderungen an einem aus einem *Prefab* erzeugten *GameObject* aber auch auf das ursprüngliche *Prefab* und so auf alle anderen Instanzen übertragen: Das geschieht über die *Prefab-Tools*. Diese in Bild 3.27 dargestellte Werkzeuleiste ist bei *GameObjects*, die aus einem *Prefab* erzeugt wurden, im *GameObject*-Kopf sichtbar.

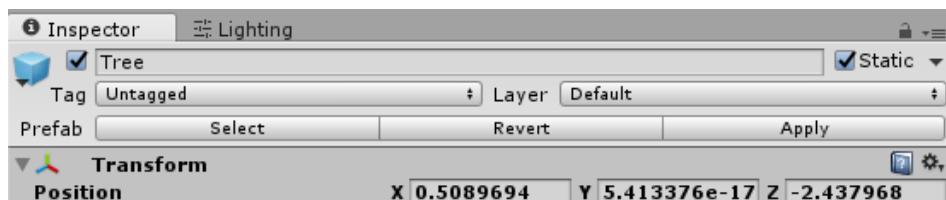


Bild 3.27 Die Prefab-Tools findest du im *GameObject*-Kopf von Instanzen eines *Prefabs*.

Über die Prefab-Tools kannst du die Änderungen in dem ursprünglichen *Prefab* speichern und dadurch auch alle existierenden Instanzen aktualisieren. Klicke dafür einfach auf **APPLY**. Möchtest du Änderungen an einer einzelnen Instanz rückgängig machen, kannst du das über den **REVERT**-Button tun. Dann werden wieder alle Werte durch das dazugehörige *Prefab* kontrolliert. Die **SELECT**-Schaltfläche öffnet das dazugehörige *Prefab* im *Project Browser*.

Alternativ können auch einzelne Änderungen rückgängig gemacht werden. Klicke dazu einfach mit der rechten Maustaste auf das fett gedruckte Label und wähle in dem Kontextmenü **REVERT TO PREFAB VALUE**.

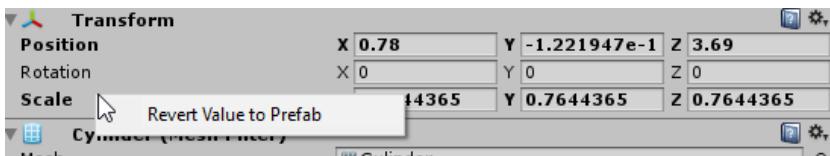


Bild 3.28 „Revert Value to Prefab“ sorgt dafür, dass der Wert wieder durch das Prefab kontrolliert wird.

■ 3.6 Die wichtigsten Components

Wie du jetzt bereit erfahren hast, geben die *Components* den *GameObjects* ihre Eigenschaften und werden auch verwendet, um Code auszuführen. Bevor wir praktisch loslegen, möchte ich dir kurz die wichtigsten Components vorstellen. Es gibt noch einige andere, mit denen wirst du aber am Anfang nur indirekt arbeiten. Einige der Components werden wir uns in späteren Kapiteln noch im Detail ansehen, an dieser Stelle geht es erst einmal darum, dass du einen ersten Eindruck erhältst.

3.6.1 Camera und Audio Listener

Die *Camera* stellt die Augen des Spielers dar. An dem gleichen *GameObject* wie die *Camera* befindet sich meist auch der *Audio Listener*, welcher die Ohren des Spielers darstellt. Bei einem Virtual-Reality-Spiel musst du dich für gewöhnlich nicht um die Konfiguration der *Camera* kümmern, da dies von dem jeweiligen SDK übernommen wird.

Lediglich die beiden Werte für die *Clipping Planes* sind je nach Spiel für dich interessant: Die *Near Clipping Plane* gibt an, wie nah ein 3D-Objekt vor der *Camera* gezeichnet werden kann, bevor es ausgeblendet wird. Die *Far Clipping Plane* bestimmt anders herum, wie weit ein 3D-Objekt von der *Camera* maximal entfernt sein darf, damit es noch gerendert wird.

Bei Virtual-Reality-Spielen musst du in der Praxis z. B. häufig die *Near Clipping Plane* anpassen, wenn du ein Spiel mit einem Cockpit entwickelst oder der Spieler einen Helm trägt. Solche sehr nahen Objekte würden sonst ausgeblendet werden. Mit der *Far Clipping Plane* kannst du die Sichtweite reduzieren, was wiederum Rechenleistung spart. Objekte, die durch die *Far Clipping Plane* abgeschnitten werden, werden jedoch nicht weich aus- und eingebendet, sondern verschwinden plötzlich und poppen genauso plötzlich wieder auf. Diese Technik macht also in erster Linie in verwinkelten Levels Sinn, wo dieser Effekt nicht sichtbar ist.

In der *Scene View* kannst du anhand der weißen Linien die Richtung und das Blickfeld der *Camera* erkennen. Zusätzlich ist unten rechts ein Live-Vorschaubild der *Camera* zu sehen, wenn du das dazugehörige *GameObject* auswählst.

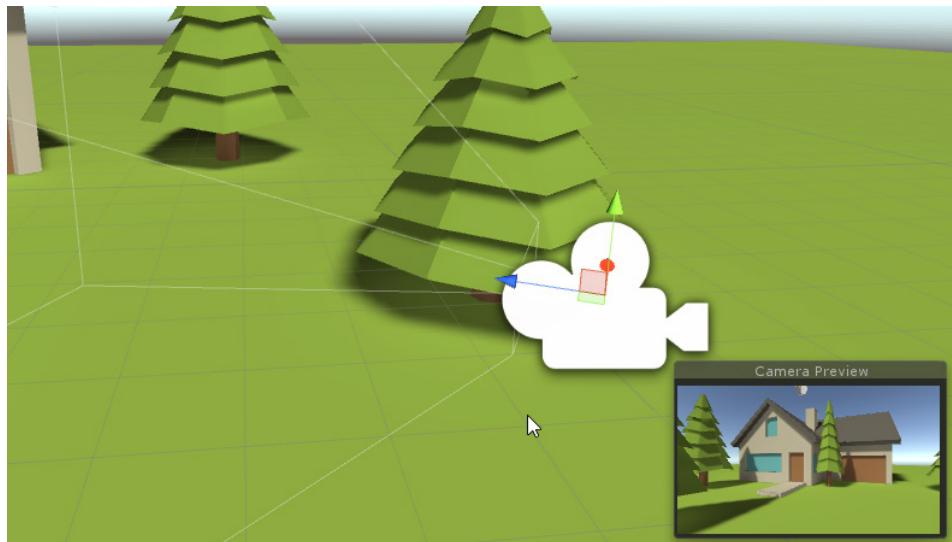


Bild 3.29 Ein GameObject mit einem Camera-Component in der Scene View

3.6.2 Lights

Damit deine *Scene* nicht komplett dunkel ist und sie auch ansprechend aussieht, benötigst du Lichter (engl. *Lights*). Vorkonfigurierte Lichter kannst du über das *Create*-Menü in der *Hierarchy* anlegen. In diesem Fall wird automatisch ein neues GameObject angelegt, welches bereits über eine entsprechendes *Light*-Component verfügt. Alternativ kannst du auch zu einem existierenden GameObject ein *Light*-Component hinzufügen. Das Light-Component findest du in dem **ADD COMPONENT**-Menü des *Inspectors* in der Kategorie *Rendering*.

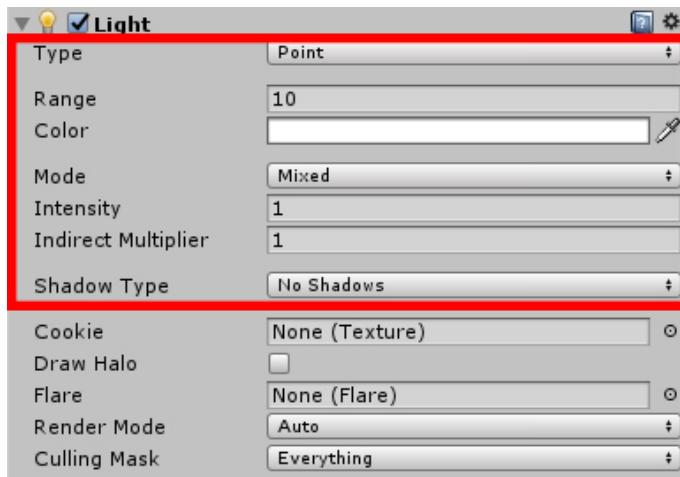


Bild 3.30 Das Light-Component mit den Standardwerten

Das *Light*-Component ist universal einsetzbar und kann unterschiedliche Arten von Licht imitieren. Es kann zum Beispiel wie eine *Sonne* konfiguriert werden, aber auch wie das *Spotlight* einer Taschenlampe. Die Details der Licht- und Schattenberechnung in Unity 2017 schauen wir uns in einem späteren Kapitel noch im Detail an, deswegen gibt es hier nur eine Kurzfassung:

Über *Type* kannst du den Lichttyp bestimmen. Ein *Point Light* strahlt in alle Richtungen, ein *Spot Light* nur in eine Richtung, wie eine Taschenlampe. Ein *Directional Light* liefert paralleles Licht, das nicht von einem nahen Punkt ausgeht, so wie die Sonne.

Baking bestimmt, ob dieses Licht in Echtzeit oder vorberechnet ist. Vorberechnete („Baked“) Lichter sind viel performanter, jedoch können sie nicht auf Änderungen in der Scene eingehen, die erst während des Spielens passieren. Außerdem werden von vorberechneten Lichtern nur GameObjects erleuchtet, die als *Static* markiert sind. *Realtime*-Lichter können auch dynamische Objekte erhellen und einen realistischen Schattenwurf berechnen. Besonders auf mobilen Plattformen sollte man jedoch bestmöglich darauf verzichten, da es sonst, speziell in VR, schnell zu Performance-Einbrüchen kommt. Mixed-Lichter kombinieren *Baked* und *Realtime* und stellen einen guten Kompromiss dar. Wann du am besten welchen Typ verwendest, schauen wir uns in einem späteren Kapitel an, da dies von mehreren verschiedenen Faktoren abhängig ist.

Für den Anfang auch interessant sind die Eigenschaften *Range* (dt. „Reichweite“), *Color*, *Intensity* und *Indirect Multiplier* (Intensitätsmultiplikator für reflektiertes Licht). Mit diesen Parametern kannst du Lichtstärke und Farbe der einzelnen Lichter bestimmen.

Der letzte interessante Punkt ist noch die Option *Shadow Type*. Hier kannst du bestimmen, ob Objekte, die von diesem Licht angestrahlt werden, einen Schatten werfen sollen oder nicht. *Soft Shadows* sehen besser aus als *Hard Shadows*, kosten jedoch auch mehr Rechenzeit.

3.6.3 Mesh Filter und Mesh Renderer

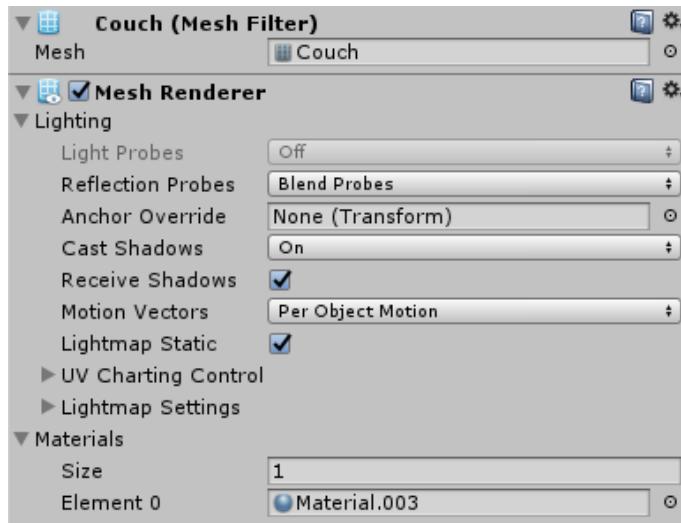


Bild 3.31 Mesh Renderer und Mesh Filter wirst du in den meisten Fällen immer zusammen auffinden.

Dieses *Component* wirst du so gut wie nie per Hand anlegen müssen, da du 3D-Modelle eigentlich immer über den *Project Browser* in die *Scene* ziehen wirst und das *GameObject* dann automatisch konfiguriert wird. Der *Mesh Renderer* benötigt ein *Mesh Filter*-Component, in dem das 3D-Modell angegeben wird, das dargestellt werden soll. Im *Inspector*-Bereich des *Mesh Renderers* kannst du die Darstellung des jeweiligen Modells (*Mesh*) anpassen.

- **Light Probes:** Hier kann angegeben werden, in welchem Modus die Lichtberechnung basierend auf *Light Probes* stattfinden soll. *Light Probes* ermöglichen, dass auch Objekte, die nicht statisch sind, bei vorberechnetem Licht eine Helligkeit und einen Farbton haben, der zur Umgebung passt, ohne dass man *Realtime*-Lichter verwenden muss. Mehr dazu folgt noch später in Kapitel 6.1.
- **Reflection Probes:** Wenn diese Eigenschaft aktiviert ist und sich eine *Reflection Probe* in der Scene befindet, erhält die Oberfläche eine Reflexion der Umgebung. Wählst du die Option *Blend Probes and Skybox* aus, wird auch die *Skybox* in die Reflexion mit einbezogen.
- **Anchor Override:** Wenn *Light* oder *Reflection Probes* verwendet werden, kann hier ein Transform angegeben werden, das als Referenzpunkt für diese Berechnungen verwendet wird.
- **Cast Shadows:** Hier kannst du einstellen, ob das dargestellte Objekt einen Schatten werfen soll.
- **Receive Shadows:** Gibt an, ob das Modell Schatten empfangen soll. Das bedeutet, dass der Schatten anderer 3D-Modelle auf diesem Modell dargestellt werden kann.
- **Motion Vectors:** Wenn diese Eigenschaft aktiviert ist, werden für dieses Objekt Bewegungsvektoren gespeichert. Dies wird für manche Bildeffekte wie *Motionblur* benötigt, kostet jedoch ein wenig Rechenleistung.
- **Materials:** Hier kannst du die *Materials* angeben, mit denen das Modell dargestellt werden soll.

3.6.4 Collider

Collider-Components bestimmen die Form eines *GameObjects*, um Kollisionen berechnen zu können. Besitzt ein *GameObject* keinen *Collider*, kollidiert es auch nicht mit anderen Objekten. Kollisionen sind also unabhängig davon, ob ein *GameObject* ein 3D-Modell darstellt oder nicht oder wie das 3D-Modell aussieht. Damit Kollisionen funktionieren, müssen beide betroffenen *GameObjects* über mindestens einen *Collider* verfügen.

Da Kollisionen sehr rechenaufwendig sein können, verwendet man meist nicht die tatsächliche Form des originalen 3D-Modells als *Collider*, sondern baut die Modelle in vereinfachter Form aus geometrischen Grundformen nach. Unity stellt dafür folgende *Collider*-Varianten zur Verfügung:

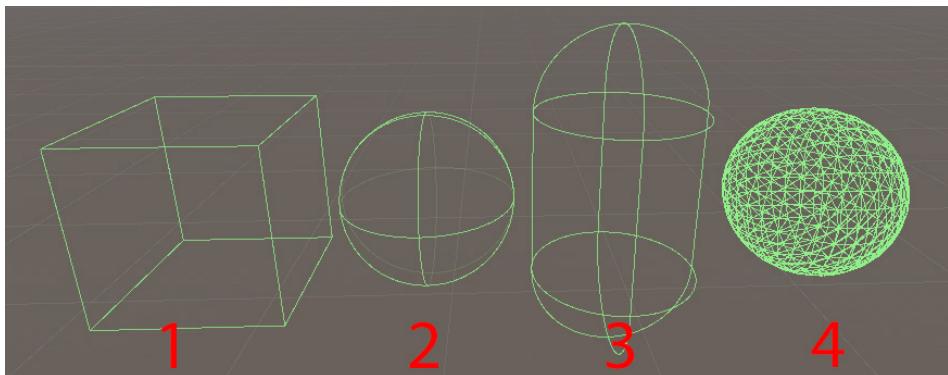


Bild 3.32 Die drei einfachen Collider und ein komplexer Mesh-Collider

1. **Box Collider:** Box Collider haben die Form eines Quaders. Dieser Collider ist der performanteste, da er der einfachste Collider ist.
2. **Sphere Collider:** Dieser Collider stellt eine einfache Kugel dar, deren Radius angepasst werden kann.
3. **Capsule Collider:** Dieser Collider hat die Form einer Kapseltablette. Es können die Länge und der Radius angepasst werden.
4. **Mesh Collider:** Der Mesh Collider nimmt exakt die Form des ihm zugewiesenen Meshs (3D-Modells) an. Ist am gleichen GameObject ein *Renderer* vorhanden, wird standardmäßig das *Mesh* des Renderers verwendet. Das bedeutet, in diesem Fall entspricht die Kollision exakt dem dargestellten Modell. Du solltest diesen Collider jedoch möglichst nicht verwenden, da dein Spiel sehr unperformant werden kann, wenn du ausschließlich diesen Collider-Typ verwendest. Es empfiehlt sich deshalb, das Modell mithilfe von einem oder mehreren einfachen Collidern nachzubauen. *Mesh Collider* können alternativ auch über ihre *Convex*-Eigenschaft automatisch vereinfacht werden.

3.6.5 Rigidbody

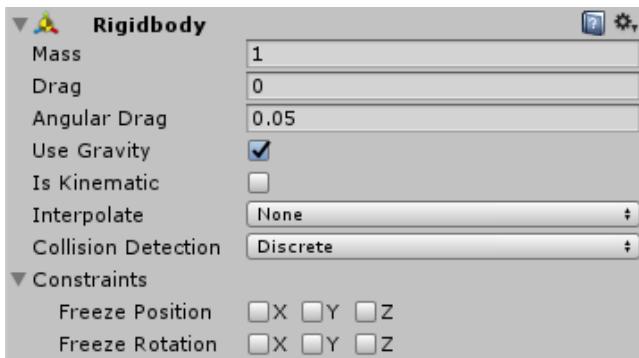


Bild 3.33 Der Inspector-Bereich des Rigidbody-Components

Wenn du das *Rigidbody*-Component zu einem GameObject hinzufügst, aktivierst du Unitys Physik-Simulation für dieses Objekt. Position und Rotation werden dann zur Laufzeit durch die Physik-Engine bestimmt. Für eine korrekte Simulation der Physik benötigt das GameObject zusätzlich noch einen oder mehrere *Collider*, da Unity ansonsten die Form des Objektes nicht kennt. Aus Performance-Gründen funktionieren *Rigidbodys* allerdings nicht mit *Mesh Collidern*, die nicht *Convex* sind.

Über *Mass* kannst du das Gewicht des GameObjects festlegen und mittels *Drag* den Luftwiderstand. *Use Gravity* bestimmt zudem, ob das Objekt von der Schwerkraft beeinflusst wird.

3.6.6 Audio Source

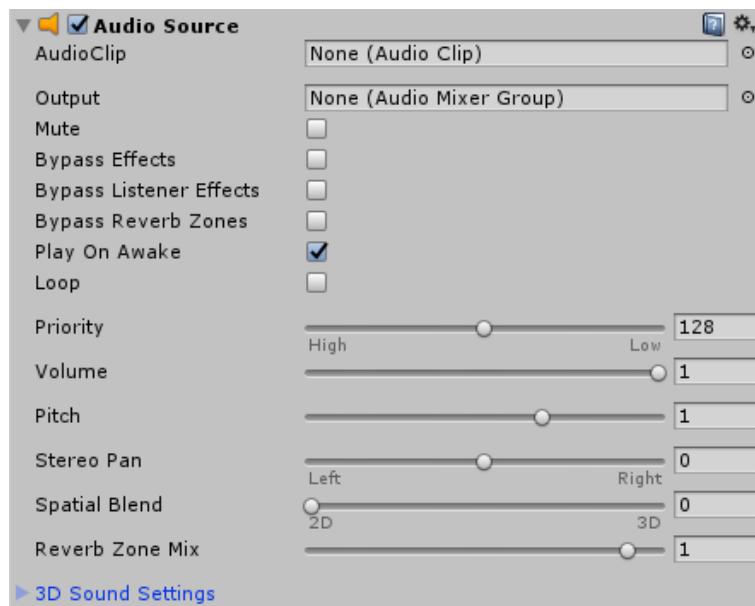


Bild 3.34 Das Audio Source-Component im Inspector

Audio Sources (dt. „Tonquellen“) werden dazu verwendet, Sounds abzuspielen. Dabei können *Audio Sources* sowohl positionsabhängiges 3D-Audio abspielen als auch 2D-Hintergrund-Sounds. Letztere sind unabhängig von der Position des Spielers und der Position der Quelle immer gleich laut.

- **Audio Clip:** Hier musst du den Sound-Effekt angeben, der abgespielt werden soll.
- **Play On Awake:** Wenn du diese Option aktivierst, wird der *Audio Clip* direkt abgespielt, wenn dieses GameObject aktiviert wird.
- **Loop:** Hier stellst du ein, ob der *Audio Clip* in einer Endlosschleife abgespielt werden soll.
- **Volume:** Hier kannst du die Laustärke der *Audio Source* festlegen.

- **Spatial Blend:** Hier stellst du ein, ob es sich bei dieser *Audio Source* um eine 2D- oder 3D-Soundquelle handelt. Im 3D-Fall sind die Lautstärke und die Position des Audioclips abhängig von der Position der Audio Source und der Position des Spielers.

3.6.7 Scripts/Behaviours

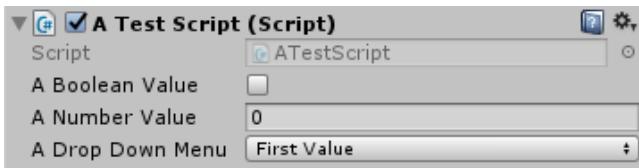


Bild 3.35 Ein Script erkennst du zum einen an dem Icon, welches die Programmiersprache angeibt, und zum andere an dem „(Script)“ hinter dem Namen. Beides ist bei einem von Unity mitgelieferten Component so nicht vorhanden.

Die wahrscheinlich interessantesten *Components* sind die sogenannten *Behaviours*, welche häufig auch *Scripts* genannt werden. Denn diese Components sind nicht Funktionen, die bereits in Unity integriert sind, sondern Quellcode-Dateien, welche im Projekt-Ordner liegen. Das bedeutet, du kannst existierende *Scripts* nach Belieben bearbeiten und auch eigene *Scripts* erstellen. Mit der Hilfe dieser *Scripts* kannst du alles machen, was du mit dem Unity Editor per Hand machen kannst, und noch einiges mehr. *Scripts* können, wie du bereits erfahren hast, bestimmte Eigenschaften dem *Inspector* zugänglich machen, sodass man sie über den Inspector konfigurieren kann. Wie so ein *Script* im Inspector aussieht, siehst du in Bild 3.35.

Die Details der Script-Programmierung sehen wir uns in einem späteren Kapitel an.

■ 3.7 Assets

Mit einem leeren Projekt kommst du nicht weit. Um ein Spiel erstellen zu können, benötigst du *Assets*. *Assets* sind alle möglichen Ressourcen, die du in deinem Projekt verwendest: Textures, Materials, Sounds, Scenes, Prefabs, 3D-Modelle etc. Nachfolgend findest du eine Übersicht wichtiger Asset-Typen und jeweils ein paar Informationen zu ihnen:

- **Textures:** Textures sind Grafiken, welche in *Materials* verwendet werden, um das Aussehen eines Objektes zu beschreiben. Textures sind in der Regel quadratisch und so angelegt, dass man sie beliebig oft horizontal und vertikal wiederholen kann, ohne dass sichtbare Kanten zu sehen sind („nahtlos“ bzw. „seamless“).

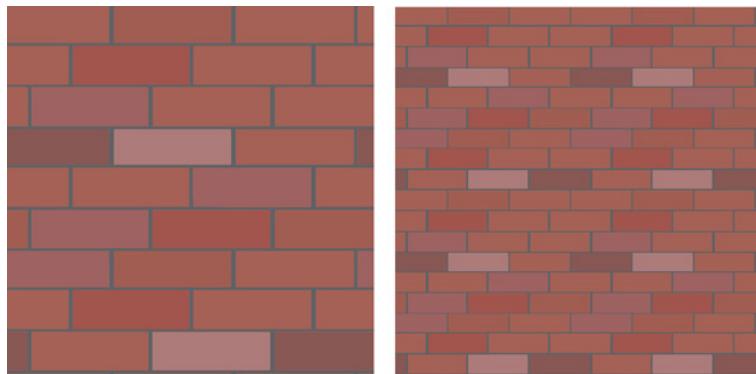


Bild 3.36 Beispiel für eine Texture mit Ziegelmuster. Links das Original, rechts horizontal und vertikal wiederholt.

- **Sprites:** Sprites sind besondere Arten von *Textures*, welche in Unity in erster Linie für *User Interfaces* wie z.B. Menüs verwendet werden. Sprites sind der Regel freigestellt, haben also einen transparenten Hintergrund. Sie können anders als andere Texture-Typen meist nicht nahtlos wiederholt werden und sind nicht dafür gedacht, auf Oberflächen von 3D-Objekten verwendet zu werden. Sprites sind in der Regel darauf ausgelegt, dass sie zur *Camera* ausgerichtet werden, da sie keine Tiefe haben.



Bild 3.37 Beispiel für eine Sprite-Grafik, wie sie in einem User Interface verwendet werden könnte

- **Materials:** Ein Material ist ein Informationscontainer, der beschreibt, wie die Oberfläche eines 3D-Modells aussieht. Neben den zu verwendenden *Textures* speichert das Material auch, welcher *Shader* verwendet werden soll. Die Shader-Konfiguration, welche zum Beispiel bestimmt, ob die Oberfläche spiegelt oder matt ist, wird ebenfalls in dem Material gespeichert, da sie von Material zu Material anders ausfallen kann.

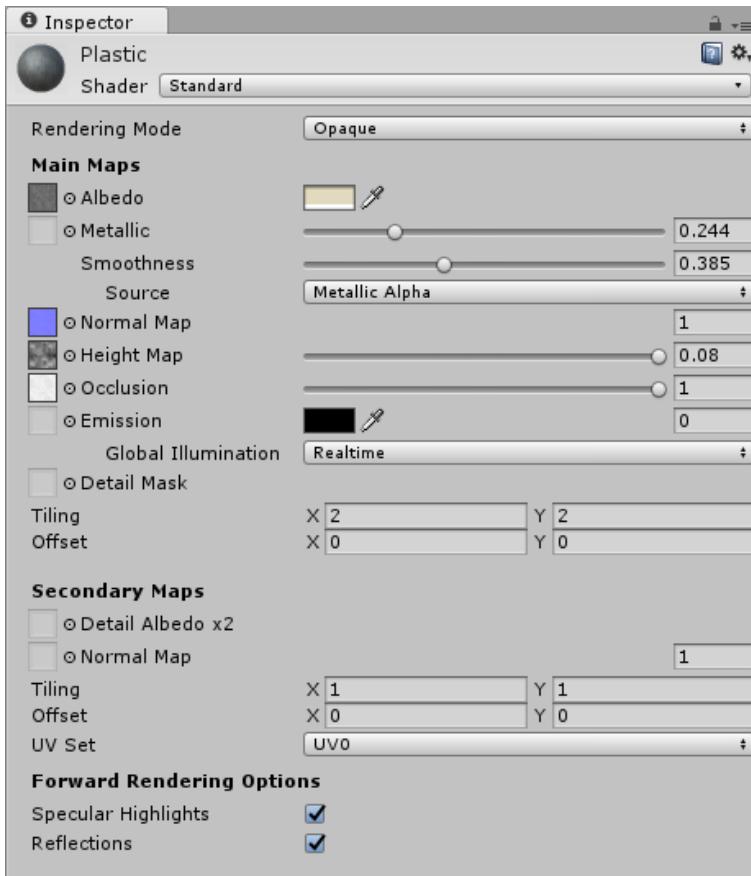


Bild 3.38 Beispiel für ein konfiguriertes Material mit „Standard“-Shader

- **Shader:** Technisch gesehen sind Shader kleine Programme, die auf deiner Grafikkarte ausgeführt werden. Sie bestimmen, welche Parameter für die Berechnung einer Oberfläche verwendet werden und wie die Darstellung berechnet wird. Unity bietet dir neben dem „Standard“-Shader auch einfachere „Mobile“-Shader an, die für schwächere Plattformen optimiert sind.
- **Sounds („Audio Clips“):** Sound-Dateien können zum Beispiel im WAV- oder MP3-Format in dein Projekt geladen werden, um sie mit einer *Audio Source* abzuspielen. Unity unterscheidet nicht zwischen Soundeffekten und Musik, der Unterschied liegt lediglich darin, wie du mit dem *Audio Clip* umgehst. Zur besseren Übersicht empfiehlt es sich, bei großen Projekten einen „Music“- und einen „Sounds“-Ordner anzulegen, anstelle alle *Audio Clips* in Ordnern zu sammeln.
- **3D-Modelle:** Aktuelle Speicherformate für 3D-Modelle können ziemlich komplex sein. In dem gängigen Format „FBX“ wird zum Beispiel häufig nicht nur das reine *Mesh* (die Modelldaten) gespeichert, sondern auch alle benötigten *Textures*, *Materials* und beliebig viele *Animations*. Zwar sind alle Inhalte, bis auf das *Mesh* optional, aber in vielen Fällen findest du alles, was du brauchst in einer einzelnen FBX-Datei.

Wenn du ein 3D-Modell importierst, sucht Unity automatisch in der Datei und um die Datei herum nach den benötigten *Textures* und legt entsprechende *Materials* automatisch an. *Animations* werden ebenfalls automatisch importiert, wenn sie vorhanden sind. Wenn ein Modell mehrere Animationen hat, wird in der Praxis häufig eine FBX-Datei pro Animation erstellt, auch wenn theoretisch mehrere Animationen in einer Datei gespeichert werden können. Diese Vorgehensweise hat den Vorteil, dass du einzelne Animationen bequem hinzufügen, bearbeiten oder löschen kannst, ohne dass die anderen Animationen dadurch beeinflusst werden.

Am besten lässt es sich in Unity mit dem genannten *FBX-Format* arbeiten. Es werden jedoch alle gängigen 3D-Formate unterstützt. Bei proprietären Formaten, die nur von bestimmten Anwendungen genutzt werden (wie „.blend“, „.max“, „.ma“) muss jeweils eine funktionierende Kopie der dazugehörigen Software auf deinem Computer installiert sein, damit Unity mit ihnen umgehen kann.



Mehr Details zu den unterstützten 3D-Formaten findest du in der Unity-Doku:

<https://docs.unity3d.com/Manual/3D-formats.html>

3.7.1 Dateien importieren

Wenn du neue *Assets* zu deinem Projekt hinzufügen (*importieren*) möchtest, kannst du die Dateien einfach per Drag & Drop aus dem Windows Explorer in den *Project Browser* ziehen. Am besten öffnest du vorher bereits den Zielordner im *Project Browser*, damit die Dateien direkt am richtigen Ort landen.

Alle *Assets* befinden sich in dem gleichnamigen Ordner in deinem Projektverzeichnis. Die Ordnerstruktur innerhalb des Assets-Ordners ist, was du im *Project Browser* siehst. Du kannst neue Assets auch direkt in den Assets-Ordner deines Projektes kopieren, wenn du sie hinzufügen möchtest. Die neuen Dateien werden automatisch von dem Unity Editor importiert, sobald du den Unity Editor öffnest.

Schaust du dir die Dateien in dem Assets-Ordner im *Windows Explorer* an, wirst du zu jedem *Asset* immer noch eine *.meta*-Datei finden, welche im Project Browser nicht sichtbar ist. Diese Datei enthält Meta-Informationen und wird von Unity beim Import der Datei angelegt. Löschst du die „.meta“-Datei, zwingst du Unity dazu, das dazugehörige Asset erneut zu importieren, und verlierst dadurch unter Umständen Änderungen, die du im Unity Editor an der Datei vorgenommen hast.

3.7.1.1 Dateien sortieren

Für eine bessere Übersicht hat es sich in Unity eingebürgert, für jeden Asset-Typ einen eigenen Unterordner in dem Projekt anzulegen. In der Regel macht es außerdem Sinn, die Dateien in inhaltliche Kategorien zu sortieren: zum Beispiel alle 3D-Modelle von Gebäuden in einen Ordner und die Inneneinrichtung in einen anderen.

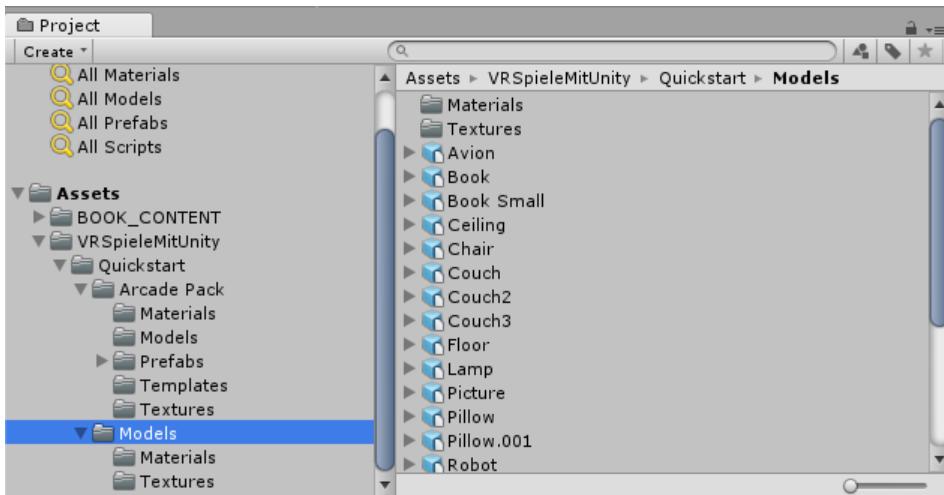


Bild 3.39 So könnte die Sortierung beispielsweise aussehen.

3.7.2 Unity Packages

Um das Teilen und Verbreiten von *Assets* zu vereinfachen, gibt es in Unity die Möglichkeit, ein oder mehrere Assets, inklusive all ihrer Abhängigkeiten, als ein sogenanntes *Unity Package* zu exportieren. Diese Pakete können dann mit wenigen Klicks in andere Projekte importiert werden, ohne dass man einzelne Dateien hin und her kopieren muss. Vereinfacht gesagt kannst dir *Unity Packages* als ein Zip-Archiv mit einer anderen Dateiendung und ein paar Zusatzinformationen vorstellen.

3.7.2.1 Unity Packages exportieren

Um Dateien aus deinem Projekt in ein Unity Package zu exportieren, musst du im *Project Browser* mit der rechten Maustaste auf eine Datei oder einen Ordner klicken und dann in dem Kontextmenü **EXPORT PACKAGE...** auswählen.

Je nach Anzahl und Größe der Dateien kann das Vorbereiten des Packages einen Moment dauern. Sobald die Dateien erkannt wurden, zeigt dir das Export-Fenster eine Liste aller Dateien an, die in das Unity Package gespeichert würden. Häufig sind das nicht nur die Dateien, die du im *Project Browser* ausgewählt hastest, sondern noch diverse andere Dateien. Das liegt daran, dass Unity automatisch Abhängigkeiten erkennt und anbietet, alle benötigten Dateien ebenfalls zu dem *Unity Package* hinzuzufügen. Exportierst du zum Beispiel ein *Material*, welches auf eine *Texture* zugreift, bietet Unity dir automatisch an, dieses *Texture* auch zu exportieren. In den meisten Fällen ist das sehr praktisch, kann aber auch dazu führen, dass dein Unity Package unnötig groß wird. Wenn du zum Beispiel ein *Script* exportierst, wird jedes Script, das auf dieses zugreift, und wiederum alle Scripts, die auf diese zugreifen, als Abhängigkeit hinzugefügt. Bei komplexen *Scripts* landen auf diese Weise schnell mal so gut wie alle *Scripts* des Projekts im Export-Fenster.

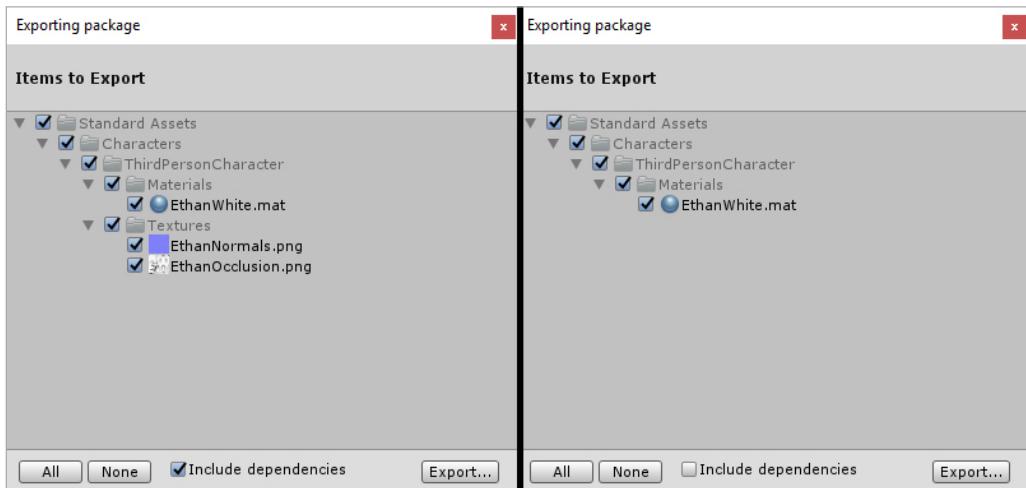


Bild 3.40 Das Exportfenster für ein Material mit und ohne Abhängigkeiten

Über die Checkboxen kannst du einzelne Assets von dem Export ausschließen, indem du den Haken entfernst. Über die **All**- und **None**-Schaltflächen kannst du alle Checkboxen auf einmal umschalten.

Möchtest du nicht, dass Unity automatisch alle Abhängigkeiten mit in das Unity Package legt, kannst du die Funktion über die Checkbox **INCLUDE DEPENDENCIES** deaktivieren. Ob es Sinn ergibt, diese Funktion zu verwenden, hängt vom Einsatzzweck des Packages ab. Im Zweifelsfall solltest du die Option aktiviert lassen, da dann in jedem Fall alle benötigten Dateien enthalten sind.

Mit einem Klick auf **EXPORT...** kannst du das *Unity Package* speichern.

3.7.2.2 Unity Packages importieren

Wenn du ein *Unity Package* importieren möchtest, kannst du das Package entweder einfach mit einem Doppelklick im Windows Explorer öffnen oder du ziehst es mittels Drag and Drop in den *Project Browser* des Unity Editors. Anschließend öffnet sich in beiden Fällen ein Fenster des Unity Editors. Dieses Import-Fenster zeigt dir an, was bei dem Importvorgang verändert würde.

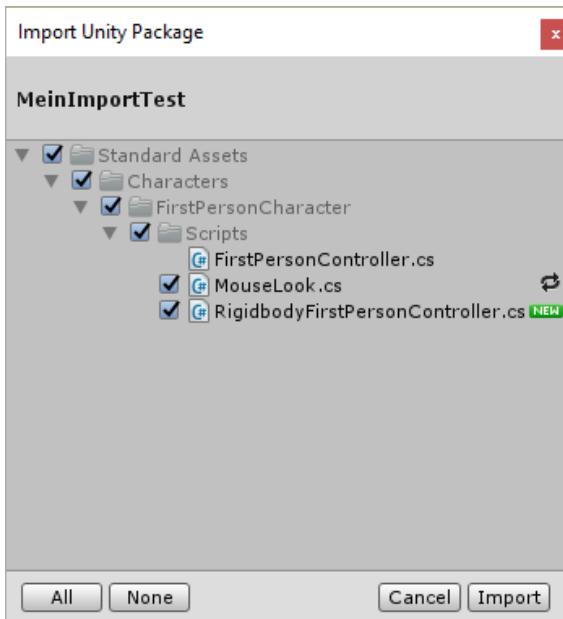


Bild 3.41 So sieht das Import-Fenster beispielhaft aus, wenn du ein Unity Package importierst.

Wie du in Bild 3.41 sehen kannst, werden dir zu jeder Datei in dem *Unity Package* ein paar Informationen angezeigt. Auf deren Basis kannst du dann mittels der Checkbox daneben für jede Datei einzeln entscheiden, ob du sie in dein Projekt importieren möchtest oder nicht.

- Dateien, die in deinem Projekt noch nicht vorhanden sind, werden mit einem grünen *NEW*-Icon markiert (in Bild 3.41, „RigidbodyFirstPersonController.cs“).
- Dateien, die bereits mit demselben Namen in deinem Projekt vorhanden sind, werden mit einem Aktualisierung-Symbol markiert, wenn sich die Version im *Unity Package* von der Version in deinem Projekt unterscheidet (in Bild 3.41, „MouseLook.cs“).
- Dateien, die bereits in deinem Projekt vorhanden sind und sich inhaltlich nicht von der Datei in dem *Unity Package* unterscheiden, werden dir zwar im Import-Fenster angezeigt, bieten aber keine Möglichkeit, sie zum Import zu markieren (in Bild 3.41Bild 3.1, „FirstPersonController.cs“).

Die Tasten **ALL** und **NONE** bieten dir wie beim Export die Möglichkeit, alle Dateien im Unity Package mit einem Klick an- oder abzuwählen.

Über die **IMPORT**-Schaltfläche startest du den Importvorgang für die ausgewählten Dateien. Die Dateien werden *nicht* in den aktuell ausgewählten Ordner des *Project Browsers* importiert, sondern mit der Ordnerstruktur, die im Import-Fenster angezeigt wird, in dem Assets-Ordner eingefügt.

3.7.3 Asset Store

Der *Unity Asset Store* ist, wie der Name schon sagt, ein Shop, wo du alle möglichen Assets für deine Unity Projekte kaufen und teilweise auch kostenlos herunterladen kannst. Die Preise sind häufig auf Hobby- und Indie-Entwickler zugeschnitten, sodass du für wenig Geld gute Inhalte kaufen kannst. Allerdings ist dies auch ein Nachteil: Die Modelle sind günstig und du kaufst nicht das alleinige Nutzungsrecht. Das bedeutet, Assets, die du aus dem Asset Store kaufst, werden häufig 1:1 auch in anderen Projekten verwendet. Gerade am Anfang solltest du dich davon aber nicht abschrecken lassen. Der Asset Store erlaubt dir nämlich auch als Einzelperson, dich auf den Teil der Arbeit zu konzentrieren, der dir Spaß macht. Den Rest kannst du dir aus dem *Asset Store* herunterladen.

Im *Unity Asset Store* findest du unter anderem auch von *Unity Technologies* erstellte Beispieldaten, die du kostenlos herunterladen und ausprobieren kannst. Alle enthaltenen Modelle kannst du außerdem für deine Spiele verwenden.

Den Asset Store öffnest du im Unity Editor über die Toolbar unter **WINDOW/ASSET STORE**. Nachdem du einen Inhalt gefunden hast, kannst du ihn mit einem Klick auf **DOWNLOAD** direkt herunterladen und importieren.

Du kannst den Asset Store auch in deinem Browser aufrufen, hier findest du dann anstelle der Download-Schaltfläche die Schaltfläche **ADD TO DOWNLOADS**, welche die jeweilige Datei in deiner *Download-Bibliothek* speichert. Deine Download-Bibliothek findest du wiederum in dem Asset Store-Fenster des Unity Editors. Mit einem Klick auf die in Bild 3.42 markierte Schaltfläche öffnest du eine Liste von allen Assets, die du bisher gekauft und oder heruntergeladen hast.



Bild 3.42 Der Unity Asset Store

3.7.4 Import Settings

Wählst du im *Project Browser* ein beliebiges *Asset* aus, werden dir im *Inspector* einige Informationen zu diesem Asset angezeigt. In den meisten Fällen beinhalten diese Informationen eine Vorschau des Assets und auch ein paar Import-Einstellungen. Bei Grafiken kannst du zum Beispiel angeben, ob es sich um eine *Texture* für ein 3D-Modell oder ein *Sprite* für ein User Interface handelt (*Texture Type*). Häufig kannst du hier auch die Auflösung deiner Grafiken anpassen, um Performance und Speicherbedarf bei deiner App zu sparen.

Jede Asset-Art hat ihre eigenen Import-Einstellungen, diese werden jedoch in der Regel erst für dich interessant, wenn du eigene Assets erstellst.

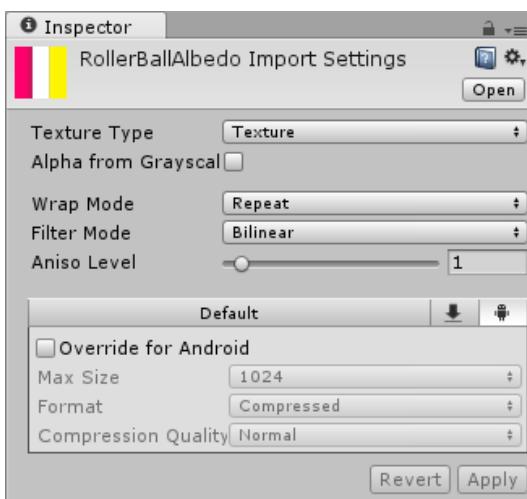


Bild 3.43 Import Settings für eine Texture

■ 3.8 Projektvorbereitung (GearVR, GoogleVR)

Bevor du anfängst, deine erste VR-Anwendung zu entwickeln, musst du noch ein paar Einstellungen in dem Unity Editor ändern, wenn du für Android entwickeln möchtest. Standardmäßig ist die Zielplattform auf Computer mit Windows, Mac und Linux eingestellt, was wir zunächst anpassen müssen. Außerdem musst du dem Unity Editor noch mitteilen, wohin du das Android SDK und das JDK installiert hast.

3.8.1 Zielplattform zu Android wechseln

Die Zielplattform kannst du zwar jederzeit ändern, jedoch dauert dieser Vorgang länger, je später du diese Aufgabe im Projekt erledigst. Bei einem Wechsel der Zielplattform müssen nämlich alle Assets für die neue Zielplattform neu importiert werden. Machst du es direkt am Anfang eines Projektes, dauert der Vorgang deshalb auch nur wenige Sekunden.

Wähle ganz oben links in der Toolbar **FILE/BUILD SETTINGS....**. Dadurch öffnet sich das *Build Settings*-Fenster, das du auch in Bild 3.44 siehst.

Hier musst du unter *Platform* die Option **ANDROID** auswählen und klickst anschließend unten links auf die Schaltfläche **SWITCH PLATFORM**, um die Auswahl für dieses Projekt dauerhaft zur Zielplattform zu machen.

Der Vorgang dauert einen kurzen Moment. Wenn die Zielplattform erfolgreich gewechselt wurde, sollte das Unity-Symbol, welches sich in Bild 3.44 neben „PC, Mac & Linux...“ befindet, neben „Android“ angezeigt werden. Wenn alles geklappt hat, kannst du das Fenster schließen.

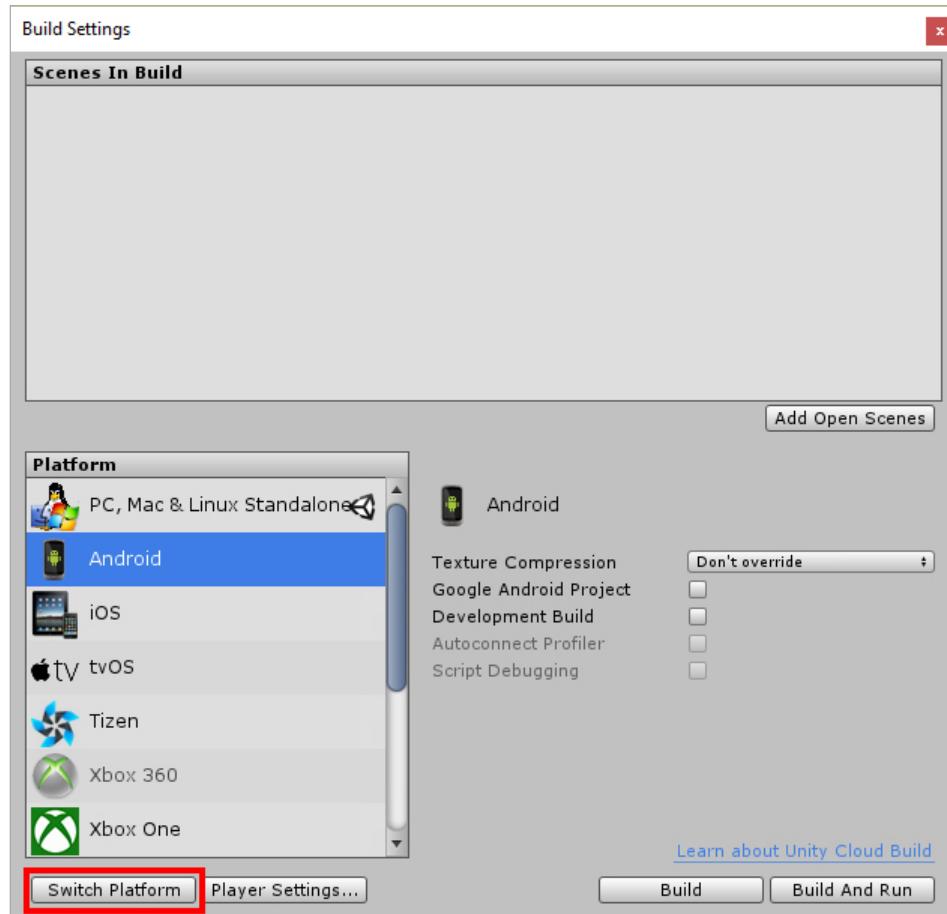


Bild 3.44 In Build Settings kannst du die Zielplattform für dein Projekt wechseln.



Manchmal beendet sich Unity bei einem Plattformwechsel von alleine.
In dem Fall musst du Unity einfach erneut starten.

3.8.2 Externe Tools angeben (Android SDK, JDK)

Bevor du nun richtig loslegen kannst, musst du noch prüfen, ob Unity das zuvor installierte *Android SDK* und das *Java Development Kit* automatisch erkannt hat. Um das zu prüfen, wähle in der Toolbar, ganz oben, **EDIT/PREFERENCES....**. In dem Einstellungsfenster musst du den Punkt **EXTERNAL TOOLS** auswählen.

In dem darauffolgenden Fenster musst du prüfen, ob Unity automatisch die richtigen Pfade für *Android SDK* und *JDK* gefunden hat.

Sollten die Pfade leer oder nicht korrekt sein, kannst du den korrekten Pfad über die Schaltfläche **BROWSE** anpassen. Sowohl für das *Android SDK* als auch für das *JDK* musst du das Stammverzeichnis der Installation auswählen.

Folgende Pfade werden von den Installationsprogrammen standardmäßig verwendet:

- Java Development Kit: *C:\Programme\Java\jdk1.8.0_...*\
- Android SDK: *C:\Users\Benutzername\AppData\Local\Android\sdk*\

(Der Ordner ist versteckt, beachte dazu die Hinweise unten)

Der „*AppData*“-Ordner ist standardmäßig allerdings *versteckt* und wird nicht angezeigt. Deshalb verrate ich dir nun folgenden Trick, um schnell in das Verzeichnis zu wechseln:

1. Klicke auf **BROWSE**. Es öffnet sich ein Auswahlfenster, in dem du den SDK-Ordner auswählen sollst.
2. Klicke in die Adresszeile des Auswahlfensters und gib dort, wie in Bild 3.45 zu sehen, folgenden Pfad ein: *%APPDATA%\..\Local\Android\sdk*
Drücke anschließend **ENTER** oder verwende die **PFEIL**-Schaltfläche, um in das Verzeichnis zu wechseln.
3. Du befindest dich jetzt in dem Standardverzeichnis des *Android SDK* und kannst den aktuellen Ordner „*sdk*“ mittels eines Klicks auf **ORDNER AUSWÄHLEN** bestätigen.

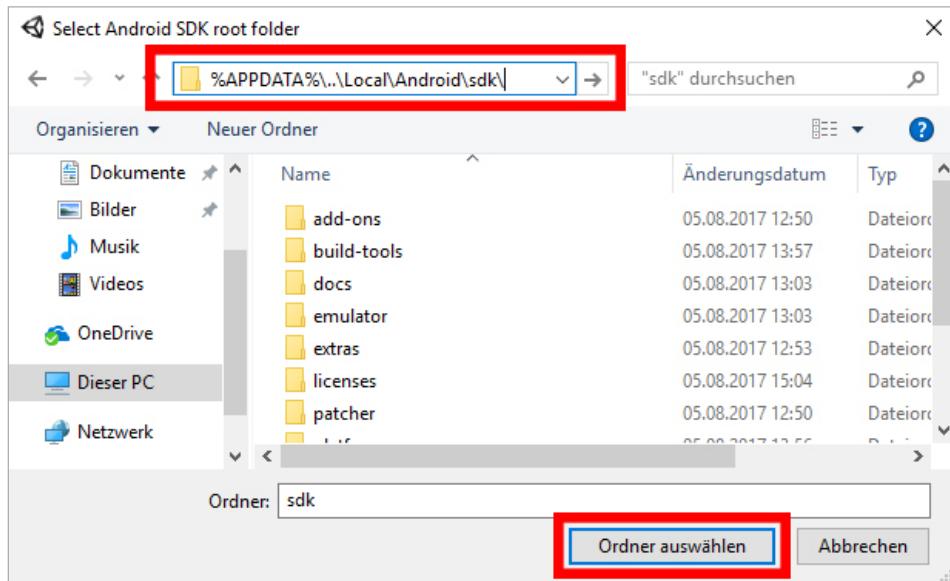


Bild 3.45 Über diese Abkürzung kannst du das Android SDK in seinem Standardverzeichnis finden.



Wenn du versehentlich ein falsches Verzeichnis auswählst, öffnet sich das Auswahlfenster sofort wieder und lässt dich deine Wahl korrigieren. Du musst den *sdk*-Ordner und den *jdk1.8.0_...*-Ordner auswählen.

Wenn du für das SDK und das JDK, wie in Bild 3.46, das richtige Verzeichnis ausgewählt hast, kannst du das Fenster über das X schließen.

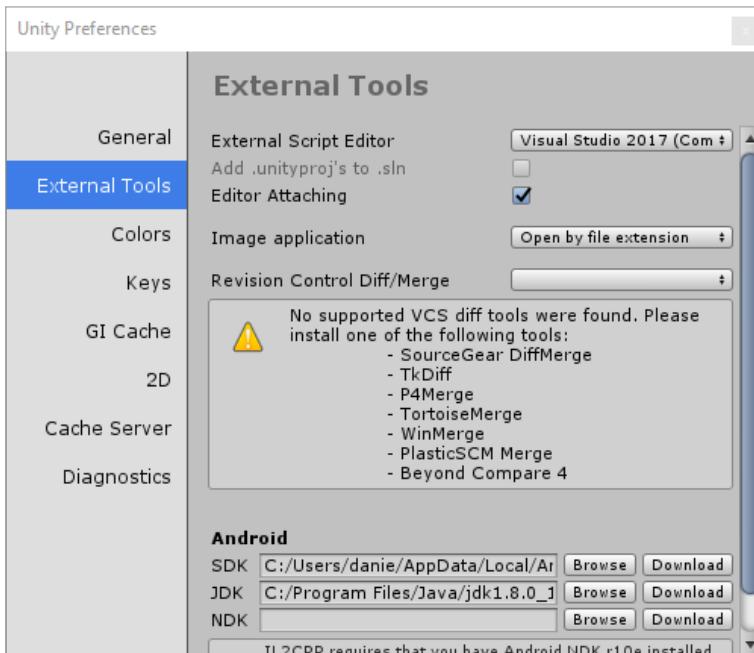


Bild 3.46 Die Einstellungen für die externen Anwendungen

■ 3.9 Test Scene erstellen und starten

So viel zur Theorie! Du solltest nun ausreichend Hintergrundwissen haben, damit wir mit dem Entwickeln einer einfachen Scene beginnen können. Dafür wirst du das Wissen aus den vorherigen Kapiteln benötigen. Wenn du also nicht genau weißt, was du tun sollst, blättere einfach noch einmal ein paar Seiten zurück. Ich werde dir jedoch auch ausreichend Hinweise geben, damit du den einzelnen Schritten gut folgen kannst.

In diesem Abschnitt werden wir ein Jugendzimmer bauen, das du dir dann mit deiner Virtual-Reality-Brille ansehen kannst. Bild 3.47 zeigt, wie das Ergebnis dieses Kapitels aussehen könnte:



Bild 3.47 So ein Jugendzimmer ist unser Ziel für dieses Beispiel.

Ich habe für dich ein *Unity Package* mit ein paar einfachen Modellen, Texturen und Materialien vorbereitet, welches du benötigst, um den nächsten Schritten folgen zu können. Das Package findest du auf der begleitenden Webseite.



Unity Package benötigt!

<http://www.vrspieleentwickeln.de/zusatz/>

Das „Quickstart“-Unity Package findest du unter **Kapitel 3**.

(Erinnerung: Das Passwort findest du in Kapitel 1.6.)

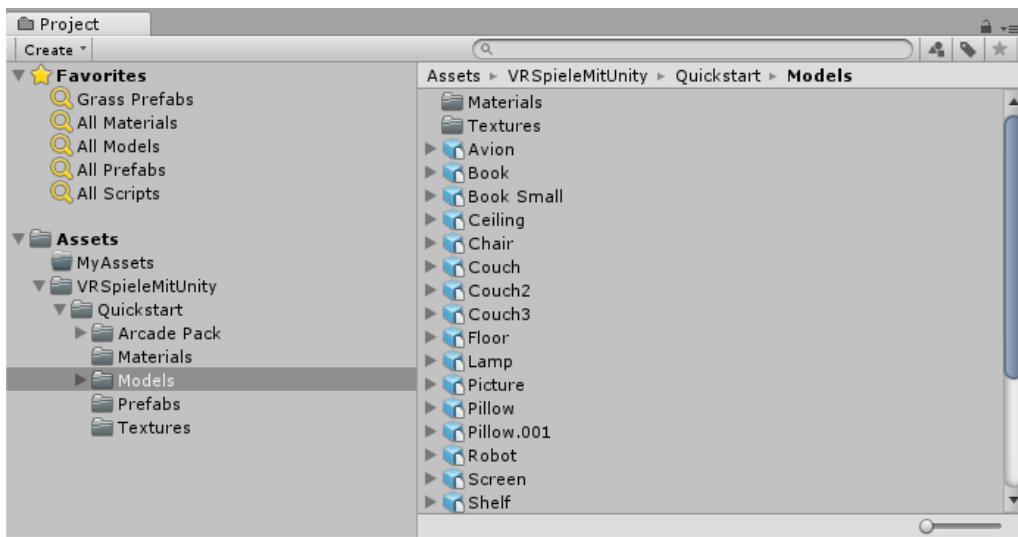


Bild 3.48 Diese Ordner solltest du nach dem Importieren in deinem Project Browser finden.

In dem Ordner *Models* findest du diverse 3D-Modelle, die wir in dem Beispiel verwenden werden. In dem *Models*-Ordner findest du außerdem einen *Materials*- und einen *Textures*-Ordner, welche jeweils die *Textures* und *Materials* enthalten, die aus den FBX-Dateien der Modelle stammen. Der Ordner *Arcade Pack* enthält ein Set von verschiedenen Arcade-Automaten, die ich für dich herausgesucht habe und die wir in dem Jugendzimmer aufstellen werden.

3.9.1 Rohbau für den Raum erstellen

Als Erstes erstellen wir einen Raum für unser Jugendzimmer. Woraus besteht ein Raum? Genau: aus einem Boden, vier Wänden und einer Decke. Los geht's.

1. Zu Beginn erstellen wir eine neue *Scene*.

(In der Toolbar: **FILE/NEW SCENE**)

2. Eine neue Scene enthält immer eine *Camera* und ein *Directional Light*, welche du in der *Hierarchy* sehen solltest.

3. Öffne jetzt im *Project Browser* den Ordner *VRSpieleMitUnity/Quickstart/Models*. Dort findest du ein Modell mit dem Namen *Floor* (dt. „Boden“). Ziehe das Modell aus dem *Project Browser* in die *Scene View*.

4. In der *Scene View* siehst du jetzt eine braune Fläche und in der *Hierarchy* ein neues Game-Object mit dem Namen *Floor*.

5. Um eine gute Übersicht zu behalten, ist es sinnvoll, GameObjects um den Ursprung, also um Position $(0, 0, 0)$ herum, anzzuordnen. Verschiebe das „Floor“-GameObject so, dass es exakt an der Position $(0, 0, 0)$ liegt. Das machst du am einfachsten, indem du das *Game-*

Object in der *Hierarchy* auswählst und dann im *Inspector* unter *Transform* die Position auf $(0, 0, 0)$ setzt.

6. Markiere das *GameObject* im *Inspector* als *Static*.

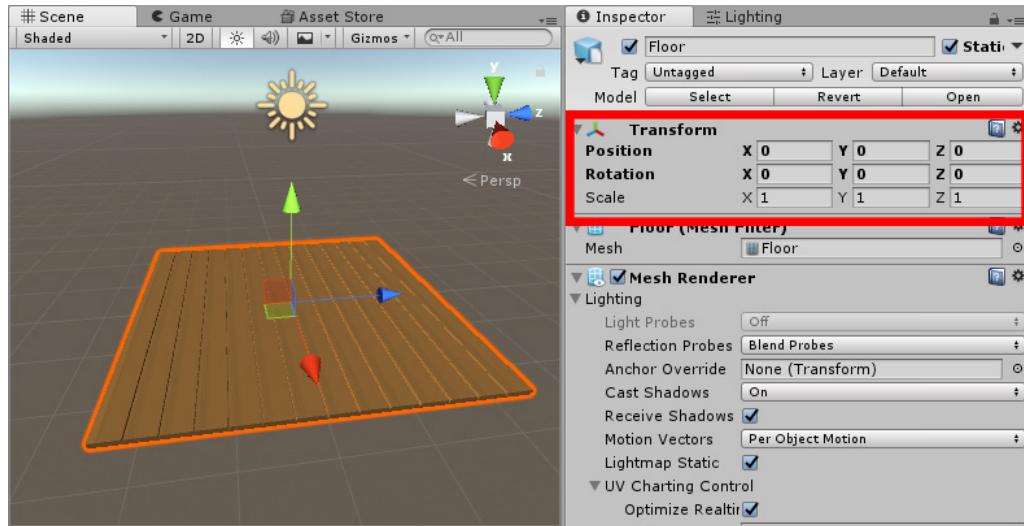


Bild 3.49 In dem rot markierten Bereich kannst du die lokale Position, Rotation und Skalierung eines *GameObjects* bestimmen.

Als Nächstes fügen wir eine Wand hinzu. Das geschieht genauso wie beim Boden über Drag & Drop und anschließende Positionierung über den *Inspector*:

1. In dem Ordner *VRSpieleMitUnity/Quickstart/Models* findest du ein Modell mit dem Namen *WindowWall* (dt. „Fensterwand“). Ziehe das Modell aus dem *Project Browser* in die *Scene View*.
2. Diese Wand musst du jetzt an einen der Enden des Bodens positionieren. Setze die *Position* dafür auf $(-2.5, 0, 0)$.
3. Dir fällt wahrscheinlich auf, dass das Wand-Modell von einer Seite durchsichtig ist. Das liegt daran, dass dieses Modell für Innenraum-Scenes optimiert ist. Da der Spieler ohnehin immer nur die Innenseite der Wand sieht, wird die unsichtbare Seite weggelassen, um Rechenleistung zu sparen. Das *GameObject* muss allerdings noch so gedreht werden, dass auch die richtige Seite nach innen zeigt.
Setze deshalb im *Inspector* die *Rotation* auf $(0, 180, 0)$, um das *GameObject* um 180° zu drehen.
4. Markiere das *GameObject* als *Static*.

Bild 3.50 zeigt, wie das *GameObject* nun aussehen sollte.

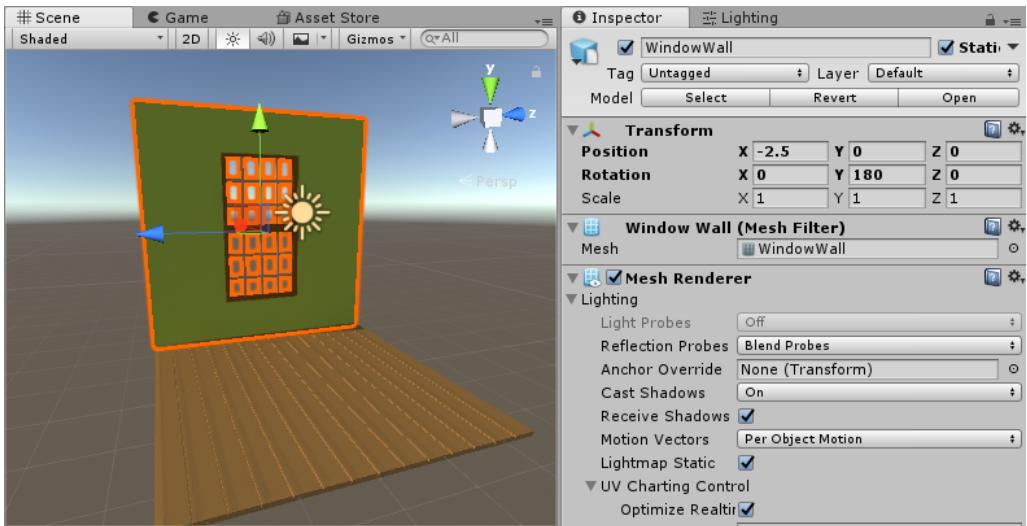


Bild 3.50 So sollte deine Scene mit dem neuen GameObject aussehen.

Eine Wand geschafft, fehlen nur noch drei. Kleine Erinnerung: Wenn du die rechte Maustaste gedrückt hältst, kannst du mit **W**, **S**, **A**, **D** und **MAUSBEWEGUNGEN** durch die Scene fliegen.

1. Für die restlichen drei Wände verwenden wir das „Wall“-Modell, welches du ebenfalls unter *VRSpieleMitUnity/Quickstart/Models* findest. Ziehe das Modell aus dem *Project Browser* in die *Scene View*.
2. Positioniere das neue „Wall“-GameObject an der Position $(0, 0, -2.5)$ und mit einer Rotation von $(0, 0, 0)$ und markiere die Wand als *Static*.
3. Wiederhole Schritt 1 und 2 für eine weitere Wand, verwende aber als Position $(2.5, 0, 0)$ und als Rotation $(0, 270, 0)$.
4. Wiederhole Schritt 1 und 2 ein weiteres Mal für die letzte Wand, verwende diesmal als Position $(0, 0, 2.5)$ und als Rotation $(0, 180, 0)$.

Spätestens jetzt bemerkst du auch einen großen Vorteil der Wände, die von einer Seite durchsichtig sind: Du kannst noch bequem von außen hineinschauen. Unter anderem aus diesem Grund ist auch die Decke, die wir als Nächstes einfügen werden, von nur einer Seite modelliert.

1. Wie die anderen Modelle auch, findest du die Decke ebenfalls in dem *VRSpieleMitUnity/Quickstart/Models*-Ordner. Ziehe das *Ceiling*-Modell ebenfalls in die Scene.
2. Positioniere sie exakt über dem Boden an der Position $(0, 5, 0)$.
3. Markiere das *GameObject* als *Static*.

Du solltest jetzt einen vollständigen Raum mit vier Wänden, einem Fenster, einem Boden und einer Decke in deiner Scene haben. Wie in Bild 3.51 ist eine der vier Wände immer nicht sichtbar, weil du von hinten durch diese hindurchschauest. Wenn du aber in der Scene um den Raum herumfliegst, kannst du überprüfen, dass alle Wände vorhanden und korrekt positioniert sind.

Kontrolliere nochmals, ob auch wirklich alle GameObjects, die du erzeugt hast, im *Inspector* oben rechts als *Static* markiert sind.

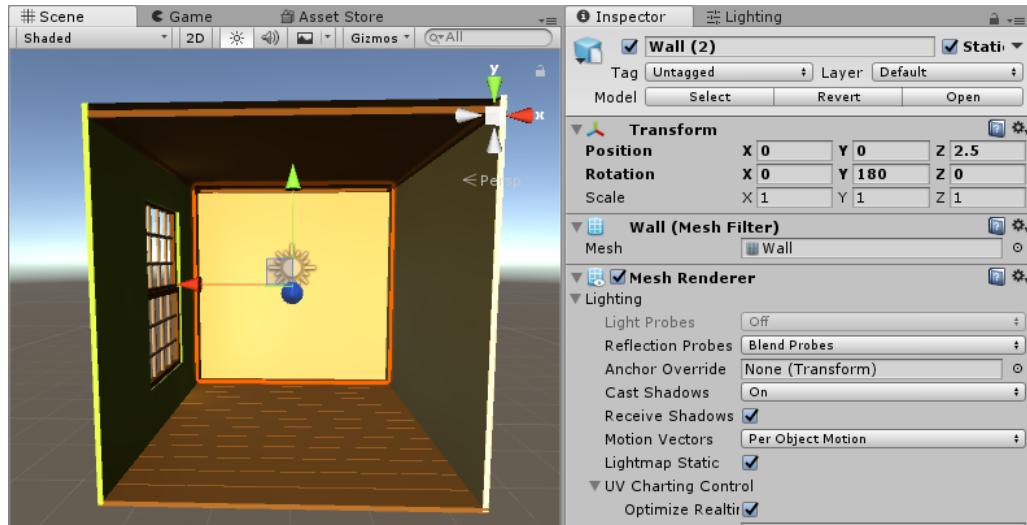


Bild 3.51 So sollte der Raum in deiner Scene nun aussehen.

3.9.2 Scene speichern

Bevor du weitermachst, solltest du erst einmal deinen Fortschritt speichern. Lege dazu in deinem *MyAssets*-Ordner einen neuen Unterordner mit dem Namen *Scenes* an.² Du solltest danach also den Pfad *MyAssets/Scenes* in deinem Projekt haben.

Speichere anschließend die Scene über die Toolbar: **FILE/SAVE SCENE**. Wähle in dem Speichern-Dialog den soeben erstellten Ordner aus und gebe der Scene einen aussagekräftigen Namen (z. B. *Quickstart_Jugendzimmer*).

💡

Regelmäßig speichern

Speichere deinen Fortschritt regelmäßig. Sollte doch mal etwas abstürzen und deine Arbeit geht verloren, ist das sehr ärgerlich. Speichere deine Scene deshalb regelmäßig ab. Über das Tastenkürzel **STRG + S** kannst du jederzeit speichern, ohne dass dein Arbeitsfluss gestört wird.

² Zur Erinnerung: Klicke dazu mit der rechten Maustaste auf den *MyAssets*-Ordner und wähle *Create/Folder*.

3.9.3 Licht anpassen

Jetzt, wo der Rohbau für unser Jugendzimmer steht, sollten wir uns um die grobe Lichtsituation in dem Raum kümmern. Wenn du von außen hineinschaust, sieht zwar alles okay aus, sobald du in den Raum hineinfliest,stellst du aber fest, dass es dunkel ist. Das liegt daran, dass derzeit kein Sonnenlicht durch das Fenster in den Raum scheint und nur ein wenig indirektes Licht in den Raum fällt.

Um das zu ändern, musst du die Sonne, also das *Directional Light* in der Scene, so drehen, dass das Licht durch das Fenster fällt:

1. Wähle dazu zunächst das *Directional Light* in der Hierarchy aus.
2. Aktiviere nun das Rotations-Werkzeug in der Toolbar oben.
3. An dem Licht werden jetzt, wie in Bild 3.52, runde Helfer angezeigt. Greife und ziehe diese mit der Maus, um das GameObject zu drehen. Anhand der gelben Linien, die von dem Licht-Icon ausgehen, erkennst du die Strahlrichtung.
4. Wenn du mit der Strahlrichtung zufrieden bist, solltest du in dem *Light-Component* die Eigenschaft *Mode* auf *Baked* setzen. Damit sorgst du dafür, dass für alle statischen Objekte eine *Lightmap* berechnet wird.
5. Da es mir in dem Raum immer noch ein wenig zu dunkel war, habe ich außerdem die Intensität des Lichtes (*Intensity*) von 1 auf 1.1 und den Multiplikator für indirektes Licht (*Indirect Multiplier*) von 1 auf 1.5 erhöht.

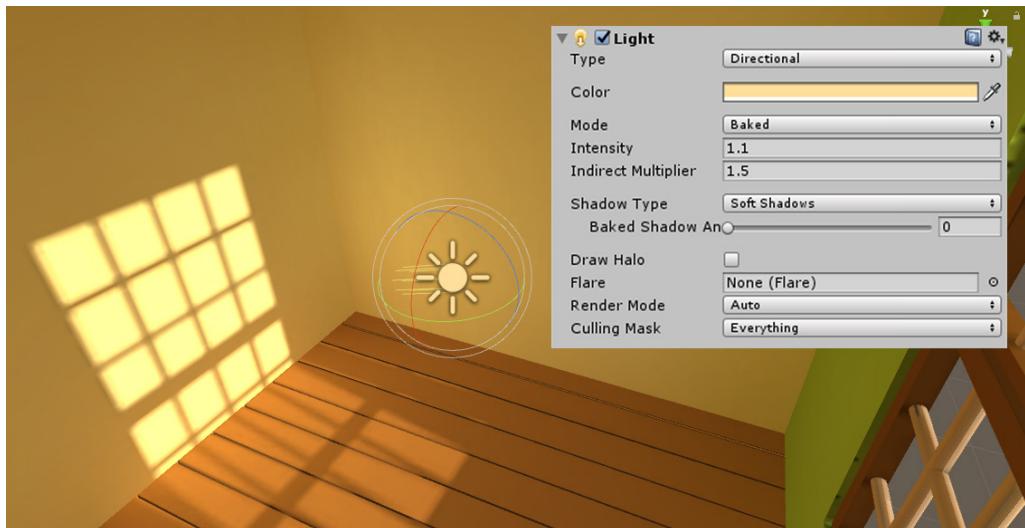


Bild 3.52 So könnte dein konfiguriertes Licht aussehen.

Sobald du mit dem Umstellen auf *Baked* dafür gesorgt hast, dass Unity Lightmaps berechnen soll, startet Unity auch mit der Berechnung. Unten rechts im Editor kannst du dann, die in Bild 3.53 dargestellte, Fortschrittsanzeige sehen. Sobald sie verschwindet, sind die Berechnungen abgeschlossen und die Lightmaps in der *Scene View* aktualisieren sich. Je mehr Objekte sich in der Scene befinden, desto länger dauert die Berechnung. Nimmst du

eine Änderung an der Scene vor, während die Berechnung läuft, wird die aktuelle Berechnung abgebrochen und mit der Änderung neu gestartet. Du musst also nicht warten, bis die Berechnung fertig ist, damit du weiterarbeiten kannst.

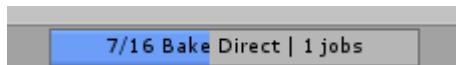


Bild 3.53 Die Fortschrittsanzeige gibt dir Aufschluss darüber, was Unity derzeit berechnet.

3.9.4 Collider hinzufügen

Weder der Boden noch die Wände besitzen derzeit einen Collider, das bedeutet, physikalische Gegenstände (*Rigidbody*s) würden einfach durch sie hindurchfallen. Gut, derzeit haben wir auch noch keine beweglichen Gegenstände im Raum, aber Böden und Wände mit Collidern auszustatten, hat noch einen anderen Vorteil: Wenn du ein Modell aus dem Project Browser in die *Scene View* ziehst, platziert sich das neue *GameObject* automatisch auf oder an Objekten, die einen *Collider* zu besitzen. Wie in Bild 3.54 dargestellt, wird das Füllen von Räumen mit Collidern an dem *GameObject* des Bodens und an *GameObjects* der Wände deutlich einfacher.

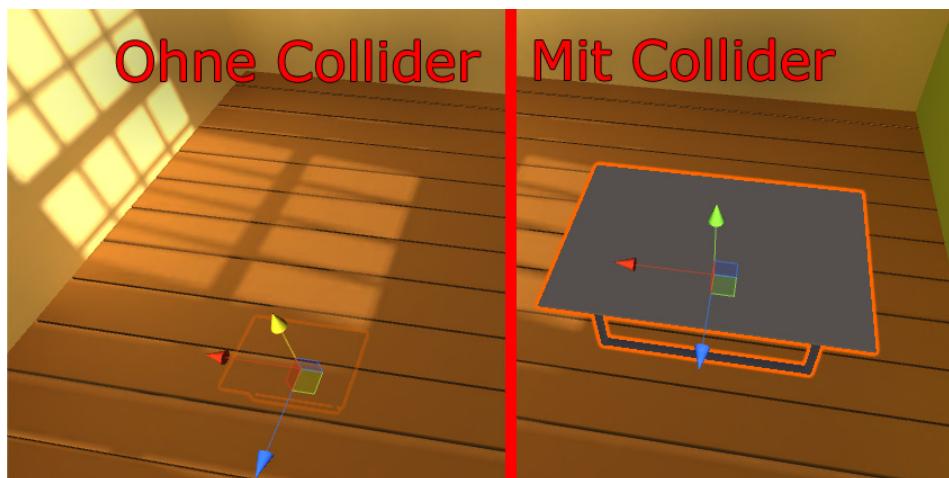


Bild 3.54 Links: Ohne Umgebungs-Collider landen neue GameObjects irgendwo. Rechts: Mit Umgebungs-Collider platzieren sich hineingezogene Objekte automatisch pixelgenau auf den Collidern.

Um uns die Arbeit zu erleichtern, fügen wir also erst einmal Collider zu dem Boden und den Wänden hinzu. Da sowohl der Boden als auch die Wände rechteckig sind, verwenden wir dafür einen *Box Collider*:

1. Wähle das *Floor*-GameObject in der Hierarchy aus.
2. Klicke im *Inspector* auf **ADD COMPONENT/PHYSICS/BOX COLLIDER**.

Der erzeugte *Box Collider* passt sich automatisch der Größe des durch den *MeshRenderer* dargestellten Modells, also unserem Boden, an.

3. Wiederhole die Schritte 1 und 2 für die GameObjects der Wände: *WindowWall*, *Wall*, *Wall (1)*, *Wall (2)*.

Wenn du das erledigt hast, kannst du die GameObjects in der Hierarchy auswählen und siehst dann, wie in Bild 3.55, in der *Scene View* eine grüne Box, die den Collider darstellt.

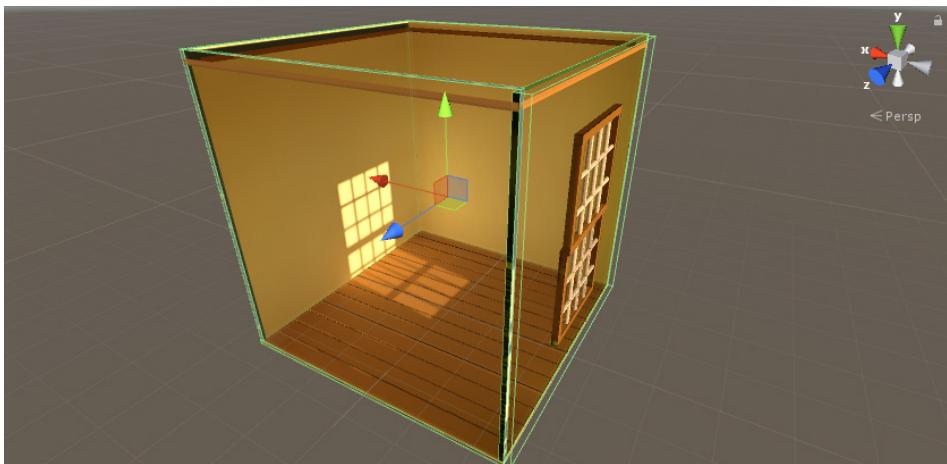


Bild 3.55 An den grünen Linien erkennst du die Collider.

(Für den Screenshot wurde die orange Outline in der *Scene View* über das Gizmo-Menü deaktiviert.)

3.9.5 Möbelstücke einräumen

Jetzt, wo der Raum steht, kannst du ihn mit Möbeln füllen. In diesem Abschnitt geht es zunächst um große Dinge wie Tische, Regale oder Couches. Um die Details kümmern wir uns im nächsten Abschnitt.

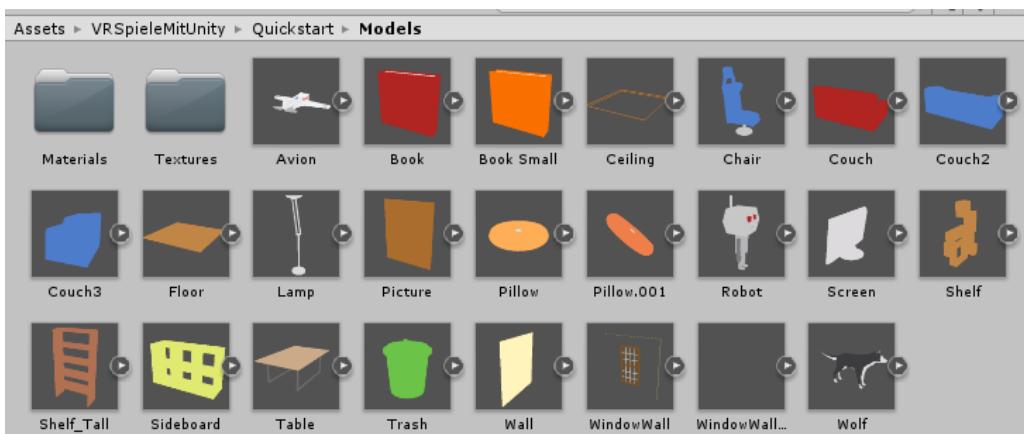


Bild 3.56 Diese Modelle stehen dir in dem Quickstart-Package zur Verfügung.

Bild 3.56 zeigt dir eine Übersicht, welche Modelle dir durch das Quickstart-Package in dem *VRSpieleMitUnity/Quickstart/Models*-Ordner zur Verfügung stehen. Dazu kommt noch der Arcade-Automat im dem *VRSpieleMitUnity/Quickstart/Prefabs*-Ordner.

Anders als die Böden und die Wände müssen und sollten diese Gegenstände nicht perfekt ausgerichtet sein. Schließlich soll das Jugendzimmer bewohnt aussehen und nicht wie in einem Katalog. Aus diesem Grund solltest du die Position, Rotation und Größe der GameObjects anpassen. Dafür kannst du die *Transform-Tools* aus der Toolbar verwenden. Verschiebe, drehe und skaliere mit der Maus! Dieses Vorgehen erzeugt ein natürlicheres Ergebnis als das Setzen runder Werte über den *Inspector*. Mache dich also, während du den Raum füllst, ein wenig mit den *Transform-Tools* vertraut.



Möbel auf einem Collider positionieren

Wenn du ein Prefab aus dem Project Browser in die Scene View ziehst, positioniert es sich automatisch auf Collidern, die in der Nähe des Ablageortes sind. In unserem Fall musst du dich, damit das korrekt funktioniert, aber in der Scene View *in* deinem Raum befinden, da du die Objekte ansonsten von außen an den Wänden positionierst.

Auf einzelnen Achsen verschieben & drehen

Häufig ist es einfacher, ein GameObject nur auf einer Achse gleichzeitig zu bearbeiten, als den Helfer mittig zu greifen und auf allen Achsen gleichzeitig zu verändern. Wähle also am besten die einzelnen Pfeile oder Kreise der Helfer aus.

Weise den GameObjects außerdem passende Collider zu, damit du später kleinere Gegenstände bequem auf ihnen platzieren kannst. In diesem Beispiel kannst du für alle Gegenstände den *Mesh Collider* verwenden, da die Modelle sehr einfach gehalten sind und wir uns zu diesem Zeitpunkt nicht zu lange mit dem Erstellen und Konfigurieren von Collidern aufzuhalten wollen – dazu kommen wir später noch.

Um einen Mesh Collider zu einem GameObject hinzufügen zu können, aktivierst du es in der Hierarchy oder Scene View und klickst dann im Inspector auf **ADD COMPONENT / PHYSICS / MESH COLLIDER**. Der Collider verwendet dann automatisch das *Mesh*, das der *Mesh-Renderer*, welcher sich am gleichen GameObject befindet, darstellt.



Static!

Vergiss außerdem nicht, jedes der platzierten GameObjects im Inspector als Static zu markieren!

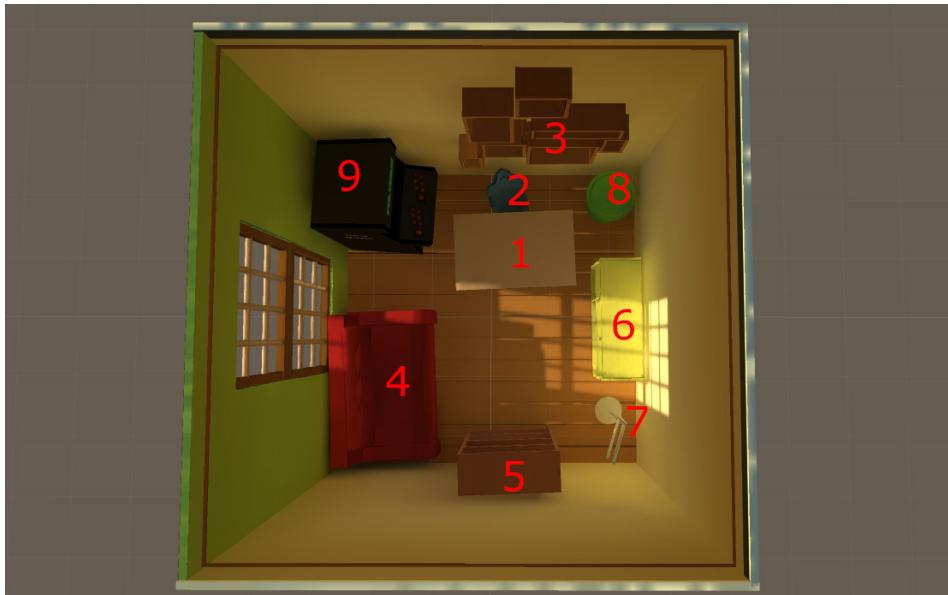


Bild 3.57 So könntest du zum Beispiel dein Jugendzimmer aufbauen.

In Bild 3.57 siehst du einen Vorschlag, wie du den Raum mit Möbeln füllen kannst. Nachfolgend liste ich dir auch nochmals auf, welches Objekt aus dem Bild welches Modell ist und was du beim Verwenden des Modells beachten solltest.

1. *Table*: Standardmäßig ist der Tisch ein wenig niedrig für einen Schreibtisch, deswegen habe ich den Tisch auf der *y-Achse* ein wenig größer skaliert: von 1 auf 1.4.
2. *Chair*
3. *Shelf*
4. *Couch*
5. *Shelf_Tall*
6. *Sideboard*: Das Sideboard war ursprünglich ein wenig zu groß, um gut in die Lücke zwischen Tisch und Wand zu passen, deswegen habe ich es mit dem Scale-Werkzeug auf allen Achsen ein wenig kleiner skaliert. Die neue Skalierung ist ungefähr (0.8, 0.8, 0.8).
7. *Lamp*: Für eines der folgenden Kapitel ergibt es Sinn, wenn die Lampe nicht direkt neben dem Arcade-Automaten steht.
8. *Trash*
9. *ArcadeCabinet_Polybius* aus dem Verzeichnis *VRSpieleMitUnity/Quickstart/Prefabs*. Dieser berühmt-berüchtigte Arcade-Automat darf in unserem Jugendzimmer nicht fehlen. Auf die Frage, ob du auch alle Kinder als *Static* markieren möchtest, solltest du mit **Yes** antworten.

Wie du deinen Raum füllst, bleibt natürlich dir überlassen, für die folgenden Kapitel solltest du aber den Tisch (1), die Lampe (7), den Arcade-Automaten (9) sowie irgendein Regal in deinem Jugendzimmer unterbringen.

Wenn du denkst, du hast den Raum gut ausgestattet, kannst du mit dem nächsten Kapitel fortfahren. Du kannst natürlich jederzeit noch Änderungen vornehmen.

3.9.6 Bücher hinzufügen

Wenn die Möbel stehen, können sie eingeräumt werden. In diesem Beispiel beginnen wir damit, ein paar Bücher in die Regale zu räumen. An diesem Beispiel werden wir uns nun auch einmal anschauen, wie man neue *Materials* erzeugt und mit *Prefabs* umgeht.

3.9.6.1 Prefabs erstellen und verwenden

In diesem Beispiel sollen alle Bücher, die wir in dem Raum platzieren, *Static* sein, damit sie in die Lichtberechnung der *Lightmap* aufgenommen werden. Anders als bei den großen Möbelstücken wirst du allerdings höchstwahrscheinlich nicht nur ein Buch platzieren, sondern einige. Aus diesem Grund ergibt es Sinn, diesen Vorgang ein wenig zu vereinfachen, indem wir ein vorkonfiguriertes Buch als *Prefab* speichern, damit wir es daraufhin wiederverwenden können.

Wir gehen Schritt für Schritt vor:

1. Als Erstes musst du das GameObject einmalig per Hand erstellen, damit wir eine Vorlage haben. Ziehe also das „Book“-Modell aus dem *VRSpieleMitUnity/Quickstart/Models*-Ordner in die *Scene View*, um ein „Book“-GameObject zu erzeugen.
2. Wähle das neu erzeugte „Book“-GameObject in der Hierarchy oder Scene View aus und markiere es im Inspector als *Static*.

Damit ist das GameObject, aus dem wir ein Prefab erstellen wollen, vorbereitet und wir können das *Prefab* anlegen:

3. Bevor du das *Prefab* speicherst, solltest du zunächst noch einen passenden Ordner anlegen, wo du das *Prefab* speichern kannst. Erzeuge also in deinem *MyAssets*-Ordner einen neuen Unterordner mit dem Namen *Prefabs*.
4. Öffne den neuen Ordner im *Project Browser* und ziehe dann das Book-GameObject aus der *Hierarchy* in den erzeugten *Prefabs*-Ordner in dem *Project Browser*.
5. Sobald du das GameObject losgelassen hast, wird das *Prefab* angelegt und im Project Browser sichtbar. Bild 3.58 beschreibt dir den Ablauf nochmals visuell.

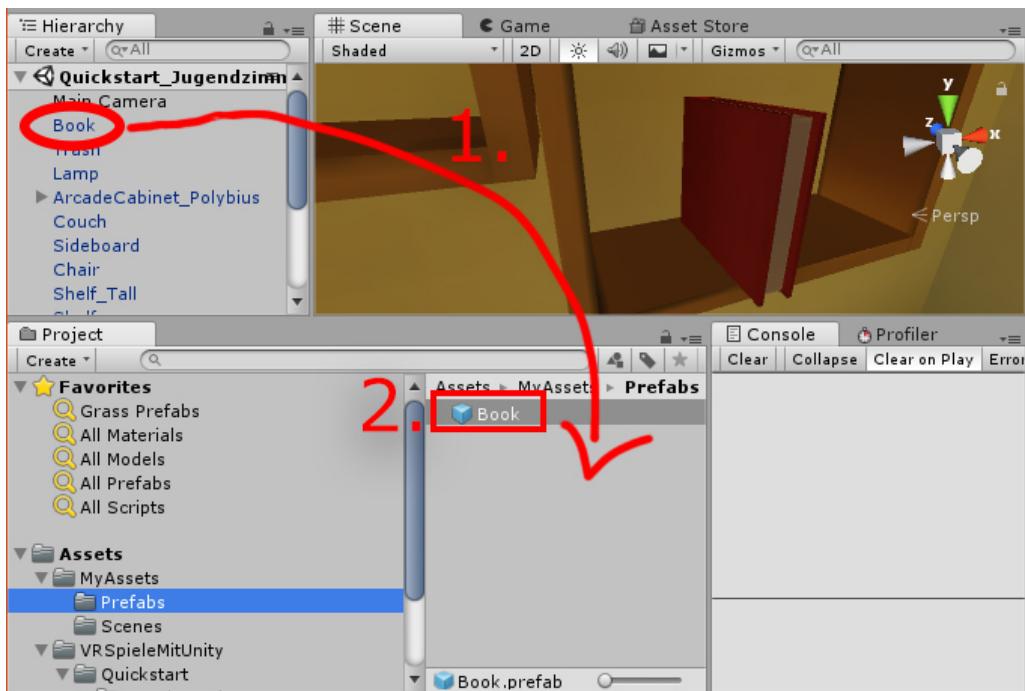


Bild 3.58 So erstellst du ein Prefab.

Das soeben erzeugte *Prefab* kannst du, wie ein 3D-Modell, einfach aus dem *Project Browser* in die *Scene View* ziehen, um eine Instanz davon zu erzeugen. Klickst du das neue GameObject *Book* (1) an, wirst du feststellen, dass es bereits als Static markiert ist – so wie es sein soll.

3.9.6.2 Prefabs Update

An dieser Stelle fragst du dich vielleicht, warum du das GameObject nicht einfach einmal erstellen und dann duplizieren kannst? Ein *Prefab* zu verwenden, hat diverse Vorteile: Zum einen ist es im Projekt gespeichert und nicht in der Scene. Das bedeutet, du könntest jetzt eine neue Scene erstellen und dort ebenfalls das *Prefab* einfügen. Das ist mit einfacherem Duplizieren nicht möglich. Außerdem ist es mit Prefabs auf eine sehr einfache Art möglich, komplexe GameObjects mittels eines Scripts in der Scene zu erzeugen, während das Spiel läuft.

Ein weiterer wichtiger Punkt ist aber die Möglichkeit, ein *Prefab* zu verändern und dadurch alle GameObjects, die auf diesem *Prefab* basieren, zu aktualisieren. Nachfolgend schauen wir uns das einmal anhand des *Book*-Prefabs an:

1. Wenn du es noch nicht gemacht hast, erzeuge mit dem *Prefab* mehrere Bücher in deiner Scene.
2. Wähle eines der erzeugten Book-GameObjects in der *Hierarchy* aus.
3. Füge im *Inspector* zu diesem GameObject einen *Box Collider* hinzu (**ADD COMPONENT/PHYSICS/BOX COLLIDER**).

4. Diese Änderung betrifft erst einmal nur dieses eine GameObject, alle anderen durch das *Prefab* erzeugten GameObjects bleiben davon noch unbeeinflusst.

Klicke deshalb jetzt im *GameObject-Kopf* im *Inspector* auf **APPLY**, um die Änderungen auf die anderen Book-GameObjects zu übertragen (siehe Bild 3.59).

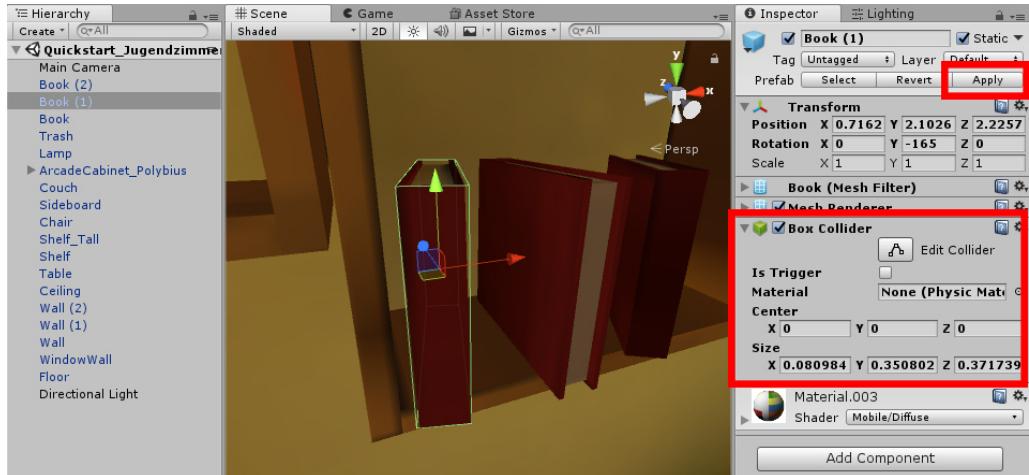


Bild 3.59 Mit der Apply-Schaltfläche kannst du alle am GameObject vorgenommenen Änderungen auf alle Instanzen des Prefabs übertragen.

Durch den Klick auf *Apply* wurden alle Änderungen an diesem Prefab, also das Hinzufügen des *Box Colliders*, auf das Prefab in deinem Projekt-Ordner übertragen. Das sorgt wiederum dafür, dass alle durch das *Prefab* erzeugten GameObjects aktualisiert werden und die Änderungen ebenfalls erhalten. In unserem Fall hat das den Effekt, dass alle Book-GameObjects in der Scene, die durch das *Prefab* erzeugt wurden, jetzt automatisch auch einen *Box Collider* erhalten haben, wie du in Bild 3.60 sehen kannst.

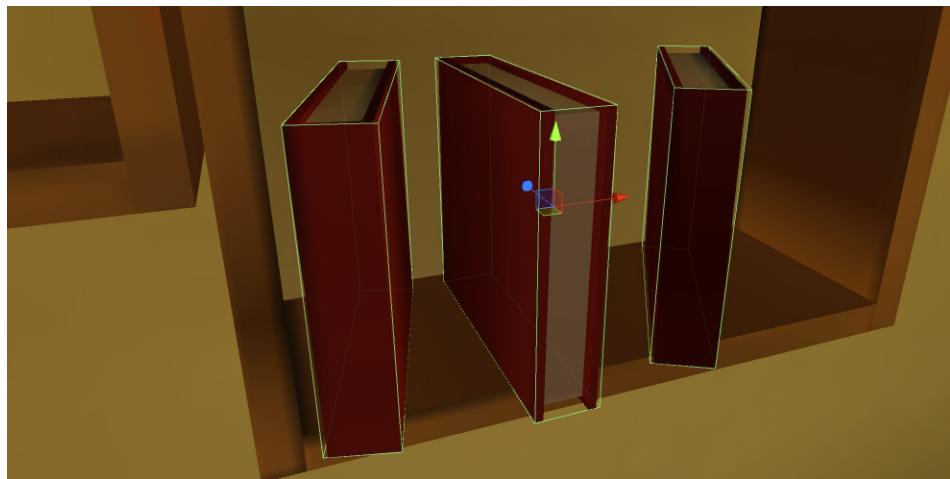


Bild 3.60 Die durch das Prefab erzeugten GameObjects wurden automatisch mit den Änderungen versorgt.

3.9.6.3 Alternatives Material erstellen

Um ein wenig mehr Variation in die Scene zu bringen, werden wir in diesem Kapitel ein weiteres Book-Prefab mit einer alternativen Farbgebung erzeugen. In diesem Zusammenhang werde ich dir zeigen, wie du neue Assets, in diesem Fall ein *Material*, im *Project Browser* anlegst.

1. Erstelle im *Project Browser* in deinem „MyAssets“-Ordner einen neuen Unterordner mit dem Namen *Materials*.
2. Öffne den Ordner im *Project Browser* und klicke in dem Ordner mit der rechten Maustaste an eine freie Stelle. Wähle in dem Kontextmenü **CREATE/MATERIAL** und gib anschließend den Namen *SimpleColorPalette_ALT* ein (ALT wie in „alternativ“).
3. Wähle das neue Material im *Project Browser* aus. Im Inspector siehst du dann die aktuelle Konfiguration des Assets.
4. Ganz oben im Inspector kannst du den Shader für das Material ändern. Klicke dafür neben dem *Shader*-Label auf **STANDARD** (das ist der aktuelle Wert) und wähle in dem Auswahlmenü **MOBILE/DIFFUSE**.

Wie du siehst, fallen sehr viele der Einstellungen in dem Material weg. Ein wenig vereinfacht ausgedrückt bedeutet das auch, dass viele Berechnungen wegfallen, weshalb dieser *Shader* schneller ausgeführt werden kann als der *Standard-Shader*.

Alles, was du jetzt tun musst, ist, dem Material ein *Base (RGB)-Texture* zuzuweisen. Das ist einfach die Textur, die der Shader für die Darstellung des Materials verwenden wird.

5. Klicke in dem quadratischen *Texture*-Feld, rechts, auf **SELECT**.
6. Es öffnet sich ein Auswahlfenster mit allen *Textures* in deinem Projekt. Wähle hier die Grafik mit dem Namen *SimpleColorPalette_ALT* aus.

Alternativ zu Schritt 5 und 6 kannst du die *Texture*, die du in dem *VRSpieleMitUnity/Quickstart/Models*-Ordner findest, auch via Drag & Drop aus dem *Project Browser* auf das Feld ziehen.

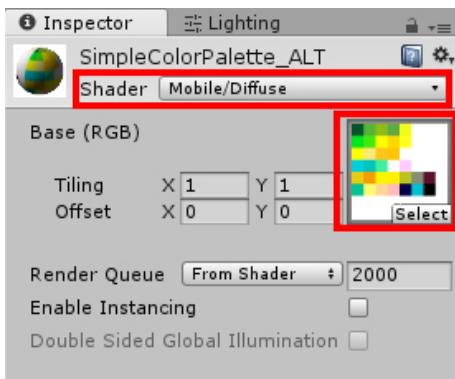


Bild 3.61 So sollte das von dir erstellte Material im Inspector aussehen.

³ Mehr Infos zu Shadern erhältst du in der erweiterten Unity-Einführung später in diesem Buch.

Jetzt musst du das Material nur noch einem der Bücher zuweisen. Das machst du entweder, indem du es per Drag & Drop aus dem *Project Browser* direkt auf eines der Bücher in der *Scene View* ziehst oder über die *Materials*-Eigenschaft des *Mesh Renderers* im *Inspector* des jeweiligen Buches. Teste am besten beides einmal aus. Das Buch erhält dann eine etwas dunklere, lila angehauchte Farbe.

Aus diesem dunkel eingefärbten Buch kannst du dann jetzt ein weiteres Prefab erzeugen.

1. Wähle das GameObject in der *Hierarchy* aus und ändere im *Inspector* ganz oben den Namen des GameObjects von *Book ...* nach *Book Alt*.
2. Jetzt kannst du das „*Book Alt*“-GameObject mittels Drag & Drop aus der *Hierarchy* an *eine freie Stelle* in deinem *MyAssets/Prefabs*-Ordner ziehen, um es als neues Prefab anzulegen. Ziehst du das GameObject auf ein bestehendes Prefab, ersetzt du dadurch das bereits vorhandene.

Damit hast du ein zweites Prefab mit einer alternativen Farbe erstellt, welches du jetzt auch zur Dekoration deiner Scene verwenden kannst. Das von dir erzeugte *SimpleColorPalette_ALT*-Material kannst du übrigens für alle Modelle aus dem *VRSpieleMitUnity/Quickstart/Models*-Ordner verwenden, um ihnen eine alternative Farbe zu geben.

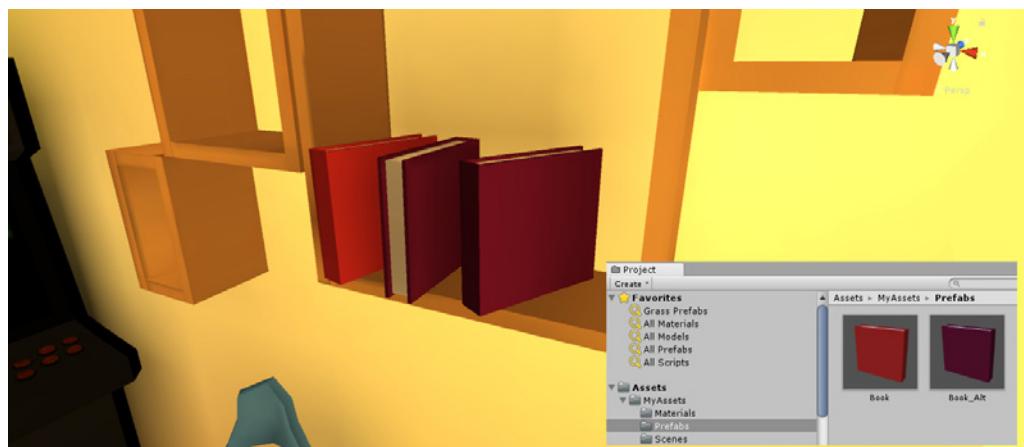


Bild 3.62 Die beiden Prefabs nebeneinander in der Scene View und im Project Browser

3.9.7 Weitere Details

Die Grundlagen in der Bedienung hast du jetzt drauf, deswegen geht es jetzt darum, die *Scene* mit diesem Wissen mit Leben zu füllen. In der Regel sind Schreibtische und Regale nämlich nicht leer, sondern gefüllt mit jeder Menge Dinge, die sich über die Zeit so angehämmelt haben.

In dem *VRSpieleMitUnity/Quickstart/Prefabs*-Ordner findest du einige Modelle, mit denen du die Tische und Regale füllen kannst. Außerdem stehen dir auch die verschiedenen *Prefabs* aus dem *Arcade Pack*-Ordner zur Verfügung.



Vergiss nicht, alle GameObjects als Static zu markieren, damit sie in die Lightmap aufgenommen werden!

Bild 3.63 zeigt dir, wie dein Jugendzimmer aussehen könnte, nachdem einige Details hinzugefügt wurden:



Bild 3.63 So könntest du das Jugendzimmer mit Details füllen.



Halte Ordnung in der Hierarchy

Du kannst leere GameObjects erzeugen und zusammengehörende GameObjects als Kinder anlegen (*Parenting*). Du kannst zum Beispiel alle Bücher, die in einem Regal stehen, in das Regal-GameObject gruppieren.

3.9.8 Lichter und Lichteinstellungen optimieren

Jetzt, wo das Zimmer so gut wie fertig ist, werden wir die Lichtsituation in diesem Raum noch ein wenig optimieren. Dafür werden wir zum einen die *Lighting Settings* ein klein wenig anpassen und zum anderen weitere Lichter zur Scene hinzufügen.

Als Erstes passen wir die Einstellungen an, damit wir beim Feintunen der Lichter bereits sehen können, wie die Lichtsituation final aussehen wird.

3.9.8.1 Lighting Settings optimieren

Was die einzelnen Optionen machen, schauen wir uns in einem späteren Kapitel noch im Detail an, für den Moment soll es reichen, dass ich dir eine gute Konfiguration für die Lichtberechnung vorstelle.

1. Öffne die *Lighting Settings* über die Toolbar **WINDOW/LIGHTING/SETTINGS**.
2. **DEAKTIVIERE** in der Kategorie *Realtime Lighting* die Option *Realtime Global Illumination*.
In der Kategorie *Lightmapping Settings*:
 3. **DEAKTIVIERE** die Option *Compress Lightmaps*.
 4. **AKTIVIERE** die Option *Ambient Occlusion*.

Sobald du deine Scene fertig eingerichtet hast, kannst du zudem die Option *Final Gather* aktivieren. Da diese Berechnung allerdings relativ lange dauert, empfiehlt es sich, sie erst einmal deaktiviert zu lassen. Sobald du denkst, dass du keine Änderungen mehr an der Scene vornehmen musst, kannst du sie aktivieren.

Dir ist vielleicht aufgefallen, dass die Scene durch diese Änderungen dunkler geworden ist. Das ist aber kein Problem, wir konfigurieren einfach das bestehende *Directional Light* neu und fügen an passenden Stellen neue Lichter hinzu.

Zunächst kümmern wir uns um das *Directional Light*:

1. Wähle das *Directional Light* in der Hierarchy aus und erhöhe den *Indirect Multiplier* auf 2.2.

Dadurch ist es schon mal ein wenig heller, um die Scene noch etwas weiter aufzuhellen, fügen wir ein *Point Light* hinzu:

2. Wähle in der *Hierarchy* dafür **CREATE/LIGHT/POINT LIGHT**. Verschiebe das Point Light so, dass es ungefähr 1,5 Meter mittig vor dem Fenster schwebt, also ungefähr bei $(-1.5, 3, 0)$.
3. Ändere außerdem folgende Einstellungen an dem neuen Point Light:

Mode: *Baked*, Intensity: 0.5, Shadow Type: *No Shadows*



Tipp für das Positionieren von Lichern

Bei Objekten ohne 3D-Modell kann es schon mal schwerfallen, sie mit den Transform-Tools zu positionieren, da man in der Scene View nicht exakt einschätzen kann, wo sich das GameObject gerade befindet. Wenn du ein GameObject in der Scene „verloren“ hast, positioniere es im Inspector einfach bei $(0, 0, 0)$ und verschiebe es von dort aus mit den Transform-Tools.

Jetzt, wo du den Raum ein wenig aufgehellt hast, können wir ein paar Detail-Lichter zu dem Raum hinzufügen. Wenn du auf dem Schreibtisch einen Monitor platziert hast, können wir diesen zum Beispiel ein wenig Licht auf die Wand dahinter werfen lassen. Das machst du am besten über ein Spotlight:

4. Erzeuge über das *Create*-Menü in der Hierarchy ein Spotlight (**CREATE/LIGHT/SPOTLIGHT**) und positioniere es vor dem Monitor. Drehe das Spotlight anschließend so, dass es von dem Monitor weg zeigt. Bild 3.64 zeigt dir, wie das aussehen könnte.

5. Ändere außerdem folgende Einstellungen an dem neuen Spotlight:

Spot Angle: ca. 110, Color: ein helles Blau (z.B. Hex: # 9FB4FFFF), Mode: *Baked*, Range: ca. 5, Shadow Type: *Soft Shadows*

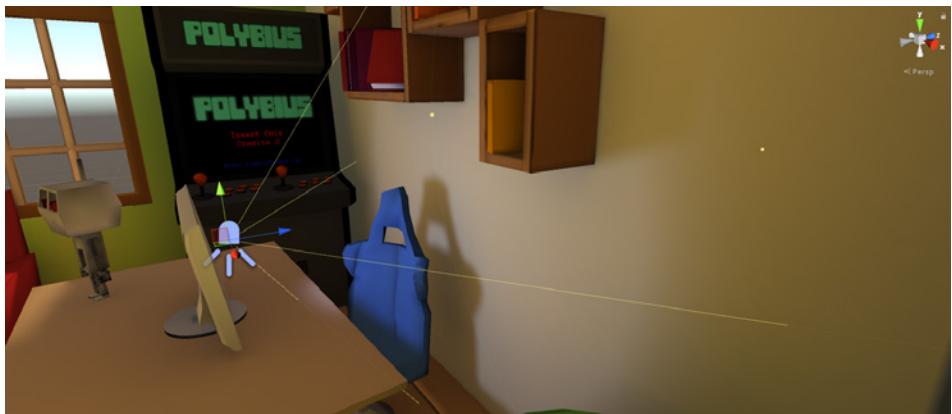


Bild 3.64 Dieses Spotlight simuliert Licht, das von dem Monitor ausgeht.

Ein weiteres *Baked Spotlight* mit *Soft Shadows* kannst du über die Lampe in der Ecke platzieren. Dabei solltest du es so konfigurieren, dass es mit einem großen Winkel (*Spot Angle*: 150) und gelblichem Licht (Hex: # *FFFAD0FF*) an die Decke strahlt. Die Intensität kannst du auf 0.4 herunterstellen, da dieses Licht nur zur Stimmung in dem Raum beitragen soll. Bild 3.65 zeigt, wie das Spotlight aussehen könnte.

Ebenfalls in Bild 3.65 zu sehen ist ein *Point Light*, das den Monitor und das Marquee⁴ des Arcade-Automaten anstrahlt, damit es so aussieht, als ob diese leuchten. Das *Point Light* solltest du circa einen halben Meter vor dem Automaten, auf einer Höhe zwischen Marquee und Monitor, platzieren. Für den Effekt in Bild 3.65 kannst du folgende Einstellungen verwenden: Range: 3.5, Color: die Farbe des Schriftzugs (Hex: # *4FAA98FF*), Mode: *Baked*, Intensity: 1.5, Shadow Type: *Soft Shadows*.



Bild 3.65 So sehen die beschriebenen Stimmungslichter aus.

Eine gute Position der Lichter ist wichtig für die Stimmung in dem jeweiligen Raum. Selbst die besten Modelle können grauenhaft aussehen, wenn du dir keine Mühe bei dem Erstellen

⁴ So nennt sich bei einem Arcade-Automaten der Schriftzug über dem Bildschirm.

der Lichter gibst. Andersherum können, wie in unserem Beispiel, einfache Modelle durch die richtige Lichtsituation auch stark aufgewertet werden.

3.9.9 Spieler positionieren

Bevor du nun die Scene zum ersten Mal testen kannst, musst du noch die Startposition und Perspektive des Spieles festlegen. Suche dazu das *Main Camera*-GameObject in der *Hierarchy*. Verschiebe das GameObject anschließend mit den *Transform-Tools* an einen geeigneten Ort. Wie du in Bild 3.66 sehen kannst, habe ich bei meinem Beispielaufbau die *Camera* mittig in den leeren Bereich zwischen Schreibtisch, Couch und Regal positioniert.

Die Rotation bestimmt, in welche Richtung der Spieler bei Spielstart schaut, und kann ebenfalls von dir entsprechend angepasst werden. Für die Höhe der Camera, also der Position auf der y-Achse, musst du je nach VR-Plattform einen anderen Wert wählen. Während bei den meisten Brillen ohne das dazugehörige *Unity Package* die Kamera auf Kopfhöhe positioniert werden muss, funktioniert die Erkennung der Körpergröße bei SteamVR automatisch. Hier muss die *Camera* deshalb ein wenig über Bodenhöhe platziert werden.

Oculus Rift, GearVR und GoogleVR:

Die Höhe, also die *Position* auf der y-Achse, sollte circa „1.8“ betragen. (Alternativ: deine Körpergröße in Metern aufgerundet)

SteamVR:

Die Höhe, also die *Position* auf der y-Achse, sollte „0.2“ betragen. SteamVR erkennt deine Größe dann automatisch und passt die *Camera* entsprechend an.



Bild 3.66 So könnte die Camera beispielsweise positioniert werden (Oculus Rift, GearVR, Daydream).

■ 3.10 VR-Unterstützung aktivieren

Damit du dir das Projekt auf deiner VR-Brille ansehen kannst, musst du erst einmal die VR-Unterstützung in Unity aktivieren. Mit aktivierter VR-Unterstützung startet die Anwendung automatisch auf deiner Brille und du kannst dich in der virtuellen Welt umsehen.

In diesem Quickstart-Beispiel werden wir zunächst nur Unitys interne VR-Unterstützung verwenden und noch nicht die, für jede Brille unterschiedlichen, Entwicklungswerkzeuge. Dies hat den Nachteil, dass die Motion-Controller deiner VR-Brille noch nicht mit diesem Projekt funktionieren werden (z.B. Oculus Touch, HTC Vive Controller, Daydream Controller, GearVR Tracked Remote).

Damit die Controller funktionieren, werden *zusätzlich* zu Unitys interner Unterstützung *Unity Packages* des jeweiligen Headset-Herstellers benötigt. Diese Packages erweitern dann die Funktionalität der Unity-internen VR-Unterstützung um Funktionen wie die Motion-Controller oder auch besondere Einstellungen, die speziell für die jeweilige Brille vorgesehen sind. Diese Unity Packages werden in Regel *Software Development Kits* (kurz *SDKs*, dt. „Software-Entwicklungswerkzeuge“) genannt. Die *SDKs* der einzelnen Hersteller werden wir uns später noch im Detail ansehen. Damit du dich möglichst schnell zum ersten Mal in deiner selbst erstellten virtuellen Welt umsehen kannst, verzichten wir in diesem Beispiel hier darauf.

3.10.1 Unitys VR-Unterstützung aktivieren

Um Unitys interne Virtual-Reality-Unterstützung zu aktivieren, musst du zunächst die *Player Settings* öffnen. Dazu wählst du in der Toolbar **EDIT/PROJECT SETTINGS/PLAYER**. Die *Player Settings* werden jetzt im *Inspector* angezeigt.



GearVR und GoogleVR: Android-Einstellungen

Falls es nicht automatisch geschehen ist, musst du auf den Reiter mit dem kleinen „Android“-Icon klicken, um zu den Player Settings für die Android-Plattform zu gelangen.

Die Option für die VR-Unterstützung findest du in der Kategorie **OTHER Settings**. Dort musst du die Checkbox *Virtual Reality Supported* **AKTIVIEREN**. Durch das Aktivieren der VR-Unterstützung erscheint darunter eine Liste. Zu dieser Liste musst du über die Hinzufügen-Schaltfläche („+“) das Virtual-Reality-SDK hinzufügen, das für deine Brille benötigt wird.

Um zu diesem Zeitpunkt unvorhersehbare Fehler zu vermeiden, solltest du alle anderen Einträge aus der Liste entfernen, wenn welche vorhanden sind. Das machst du, indem du den Eintrag mit einem Klick auswählst und anschließend über die Entfernen-Schaltfläche („-“) löscht.

Welches SDK du für welche VR-Brille auswählen musst, erfährst du in Tabelle 3.1.

Tabelle 3.1 Welche Brille benötigt welches SDK?

SDK	VR-Brille
Oculus	Oculus Rift, Samsung GearVR
OpenVR	SteamVR-kompatible Brillen (HTC Vive, OSVR, Oculus Rift)
Daydream	Daydream-kompatible Brillen
Cardboard	Cardboard-Brillen

■ 3.11 Player Settings anpassen (GearVR, GoogleVR)

Wenn du ein Spiel für Android entwickelst, musst du noch ein paar andere Einstellungen in den *Player Settings* vornehmen.

Für eine optimale Performance musst du als Erstes die Option *Multithreaded Rendering AKTIVIEREN*. Die Option findest du ebenfalls in der *Other Settings*-Kategorie unter *Rendering*.

Um dein Spiel auf deinem Smartphone installieren zu können, musst du außerdem noch ein paar Informationen zu deiner App angeben. Diese Optionen findest du alle in der *Other Settings*-Kategorie unter *Identification*.

Wir beginnen mit dem *Bundle Identifier*. Diese Kennung wird verwendet, um deine App auf den Smartphones und in den App-Stores eindeutig zu identifizieren. Für gewöhnlich verwendet man hier die eigene Webseite und den jeweiligen App- oder Projektnamen.

Aufbau: *[Domain Endung (Top Level Domain)].[Domain-Name].[App-Name]*

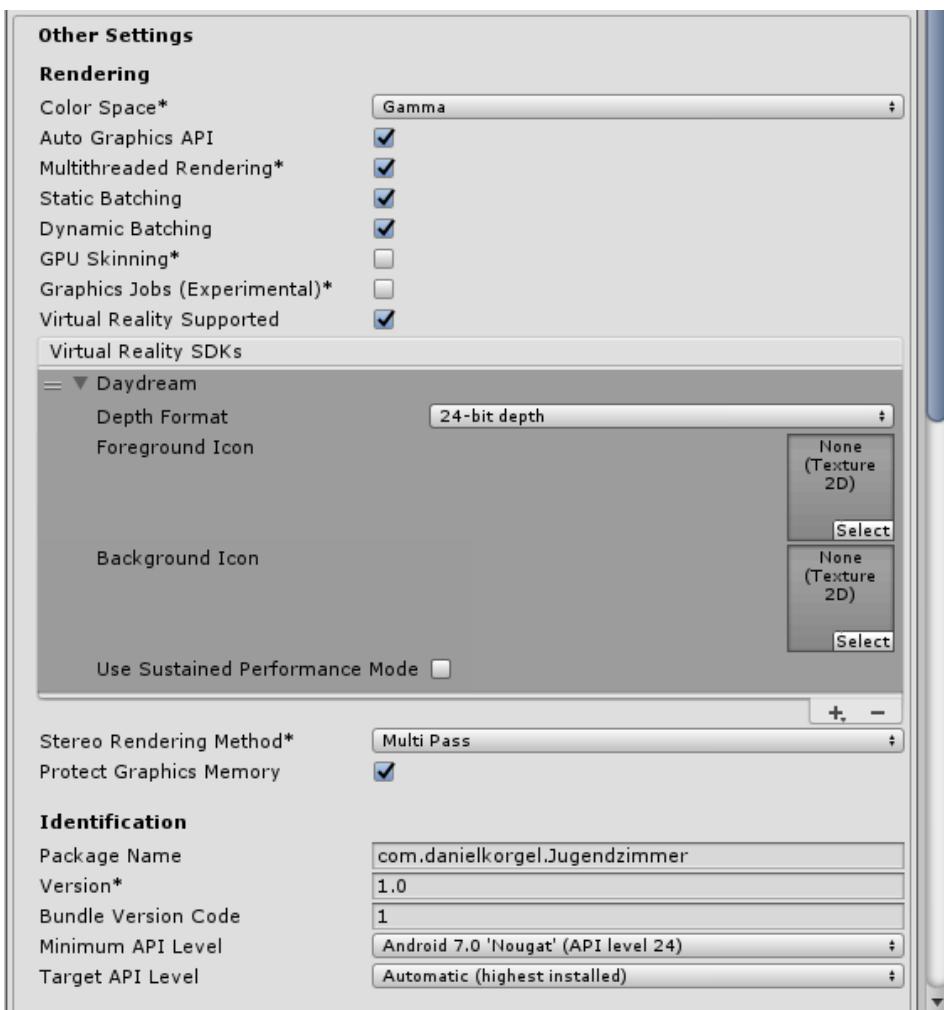
Beispiel: *de.meine-webseite.MeineErsteApp*

Wenn du keine eigene Webseite hast, kannst du dir hier auch einen Wert ausdenken. Wichtig ist, dass der gesamte *Identifier* eindeutig ist und von keiner anderen App auf deinem Handy (und später im Store) verwendet wird. Die meisten Leute verwenden in diesen Fällen zum Beispiel ihren Vor- und Nachnamen, gefolgt vom Projektnamen (zum Beispiel *de.Max-Mustermann.MeinVRSpiel*).

Außerdem muss noch das *Minimum API Level* angepasst werden. Mit diesem Wert gibst du an, welche Android-Version deine App mindestens benötigt, um zu funktionieren. Wenn deine App später im *PlayStore* ist, wird sie bei Geräten mit älteren Android-Versionen nicht zum Download angeboten. Die VR-Brillen-Hersteller geben diesen Wert für ihre jeweilige Brille in der Regel vor. Du musst deshalb das *Minimum API Level* entsprechend der Tabelle 3.2 auswählen.

Tabelle 3.2 Minimum API Level nach VR-Brille

VR-Brille	Minimum API Level
GearVR	19 („KitKat“, Android 4.4)
Daydream	24 („Nougat“, Android 7.0)
Cardboard	19 („KitKat“, Android 4.4)

**Bild 3.67** Ein Beispiel für die Player Settings eines Daydream-Projektes

■ 3.12 Anti-Aliasing aktivieren

Anti-Aliasing beschreibt den Vorgang der Kantenglättung bei der Darstellung von Grafiken auf einem Bildschirm. Anti-Aliasing verhindert, dass Kanten von Gegenständen auf den Bildschirmen flackern und sich unschöne „Treppen“ an den Kanten bilden („Aliasing“). In Virtual Reality fallen diese Kanten besonders stark auf, weil der Abstand zum Bildschirm bei einer VR-Brille deutlich kleiner ist und das Bild durch die Linsen zusätzlich vergrößert wird. Bei einem VR-Spiel ist es deshalb empfehlenswert, mindestens *zweifaches Anti-Aliasing* aktiviert zu haben, auch wenn man dafür Abstriche in anderen Bereichen machen muss.

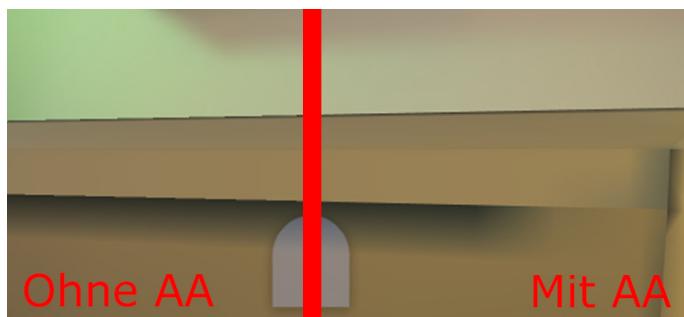


Bild 3.68 Der Vergleich einer Kante im Spiel, ohne und mit Anti-Aliasing

Dieser Abschnitt ist in erster Linie für Android-Projekte, also *GearVR* und *GoogleVR* interessant, da hier Anti-Aliasing standardmäßig vollkommen deaktiviert ist. Am PC ist es bereits aktiviert, allerdings hast du hier, je nachdem wie leistungsstark dein PC ist, die Möglichkeit, den Wert ein wenig höher zu stellen.

Um die Anti-Aliasing-Einstellungen zu ändern, musst du die *Quality Settings* aufrufen. Wähle dafür in der Toolbar: **EDIT/PROJECT SETTINGS/QUALITY**.

Im *Inspector* siehst du nun die *Quality Settings*. Wenn deine Zielplattform Windows ist, ist für dich die Spalte mit dem *Stand-alone-Icon* (einem Pfeil nach unten) interessant; wenn du für *Android* entwickelst, entsprechend die Spalte mit dem *Android-Icon*.

Der *grüne Haken* in jeder Spalte bestimmt die standardmäßig aktivierte Qualitätsstufe, in Unity *Level* genannt, für die jeweilige Plattform. Der Einfachheit halber werden wir diese Qualitätsstufe bearbeiten, auf diese Weise sind die Änderungen sofort sichtbar und die Qualitätsstufe muss nicht erst aktiviert werden. Wähle also das Qualitätslevel aus, bei dem für deine Plattform der grüne Haken ist. Das ausgewählte Level wird dann dunkelgrau hinterlegt. Wenn die Qualitätslevel bei dir so aussehen wie in Bild 3.69, müsstest du für den PC *Ultra* und für *Android Medium* auswählen.

Unter der Qualitätsstufen-Tabelle findest du die aktuelle *Rendering*-Konfiguration für die ausgewählte Qualitätsstufe. Hier kannst du unter dem Punkt *Anti Aliasing* die Stärke der Kantenglättung anpassen. Aber Vorsicht, Kantenglättung benötigt viel Rechenleistung!

Wenn du für *Android* entwickelst, wähle bei dem Punkt *Anti Aliasing* den Wert **2x MULTI SAMPLING** aus.

Wenn du für den PC entwickelst, sollte „2x Multi Sampling“ bereits aktiviert sein. Sollte dein PC jedoch deutlich besser als die Mindestanforderungen für deine VR-Brille sein, kannst du hier auch vierfaches oder eventuell sogar achtfaches Anti-Aliasing aktivieren. Ich empfehle dir jedoch, für den Anfang erst einmal einen kleineren Wert zu wählen, damit deine erste eigene VR-Erfahrung nicht direkt ruckelt und Unwohlsein bei dir verursacht.

Die anderen Einstellungen in den *Quality Settings* kannst du zunächst auf ihrem Standardwert lassen.



Bild 3.69 In den Quality Settings kannst du die Qualitätsstufen („Levels“) für die Grafik festlegen.

■ 3.13 Die Scene in VR testen

Nun ist es endlich so weit, schon gleich wirst du erstmals die von dir erstellte Welt in *Virtual Reality* erleben können! Der Weg dorthin unterscheidet sich allerdings wieder abhängig davon, ob du eine VR-Brille verwendest, die direkt am PC angeschlossen wird, oder ob du eine mobile VR-Brille verwendest. Bei mobilen VR-Brillen musst du nämlich zunächst eine App erstellen und auf deinem Smartphone installieren, während du mit einer PC-Brille die Scene direkt im Unity Editor in Virtual Reality testen kannst.

3.13.1 Am PC in VR testen (Oculus Rift, SteamVR)

Als Entwickler eines VR-Spiels, das als Zielplattform den PC hat, hast du Vorteil, dass du auf derselben Plattform entwickelst, wie deine VR-Brille läuft. Aus diesem Grund musst du nicht, wie zum Beispiel bei Android-Apps, eine Kopie deines Spiels auf ein anderes Gerät kopieren, um es auszuprobieren zu können.

Bei einer PC-VR-Brille musst du an dieser Stelle nichts anderes tun, als auf den **PLAY**-Button in der Mitte der *Unity-Toolbar* zu klicken und deine VR-Brille aufzusetzen.

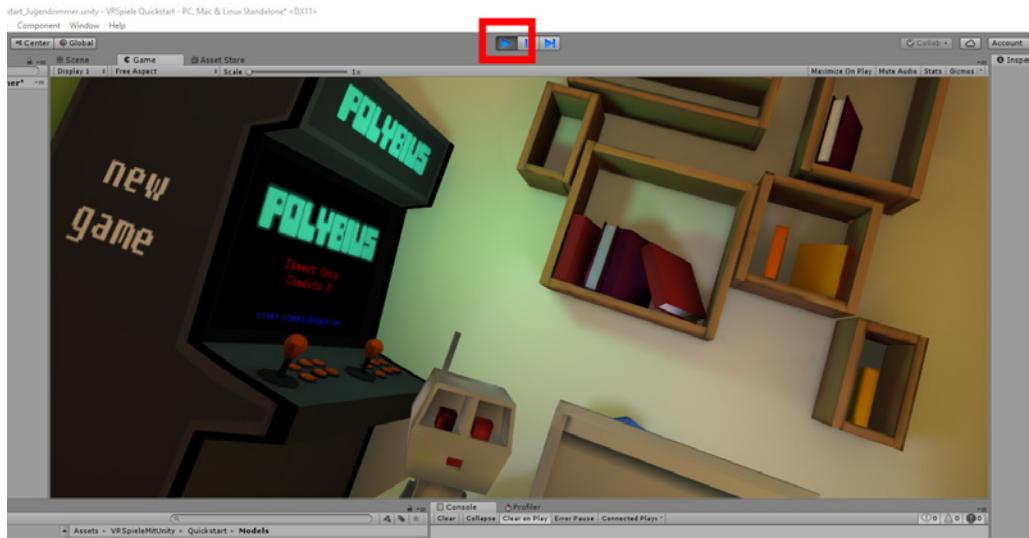


Bild 3.70 Nur ein Klick, und schon ist die Scene auf deiner VR-Brille.

Der Startvorgang kann ein paar Sekunden dauern. Während dieser Ladezeit ist es vollkommen normal, dass der Unity Editor nicht mehr reagiert.

Um die Startzeit zu verkürzen, ist es empfehlenswert, im Vorfeld darauf zu achten, dass die jeweilige *VR-Runtime* bereits gestartet ist. Das bedeutet im Falle der *Oculus Rift*, dass die *Oculus Home*-Anwendung offen oder in der Taskleiste ist. Wenn du eine *SteamVR*-Brille wie die *HTC Vive* verwendest, bedeutet es, dass du über die **VR**-Schaltfläche in *Steam* bereits den *SteamVR*-Modus gestartet hast.

Sobald die Scene geladen wurde, wechselt Unity automatisch von der *Scene View* in die *Game View* und zeigt dir eine Kopie von dem, was du in dem Headset siehst.

Wenn du deine Anwendung auf der *Oculus Rift* testest, kann es außerdem vorkommen, dass du zuerst die *Health and Safety Warning* (dt. „Gesundheit- und Sicherheitswarnung“) bestätigen musst, bevor es losgehen kann.

3.13.1.1 Die Editor-Test-Steuerung

Mittig in der Unity-Toolbar befindest du die Schaltflächen, um das Editor-Testen zu steuern. Die Steuerung besteht aus drei Schaltflächen, die ich dir jeweils kurz vorstellen möchte.



Bild 3.71 Die Bedienelemente zur Steuerung der Test-Sessions

Von links nach rechts:

- **Play:** Die dreieckige Play-Schaltfläche startet eine Test-Session. Wenn du in einer Test-Session bist, kannst du nochmals auf das Play-Symbol klicken, um das Testen zu beenden.
- **Pause:** Wenn du in einer Test-Session bist, kannst du das Testen über diese Schaltfläche pausieren. Das ist zum Beispiel hilfreich, wenn du dir eine bestimmte Situation genau ansehen möchtest, ohne in Zeitdruck zu geraten. *Diese Taste beendet das Testen nicht!*
- **One Frame Forward:** Wenn du eine Test-Session pausiert hast, kannst du über diese Schaltfläche einen einzelnen Frame berechnen und sofort wieder pausieren. Dies ist hilfreich, wenn du dir eine bestimmte Situation genau ansehen möchtest und nicht im perfekten Moment pausiert hast.

Du kannst deine Scene, während du in einer Test-Session bist, weiterhin bearbeiten. Das ist zum Beispiel praktisch, wenn man ein Objekt perfekt positionieren und sich die Änderung sofort in VR-Ansehen möchte. **Änderungen, die du zur Laufzeit machst, werden allerdings nicht gespeichert.** Sobald du nochmals auf das *Play-Symbol* klickst und damit das Testen beendest, werden alle zur Laufzeit gemachten Änderungen wieder zurückgesetzt. Du musst dir also die gemachten Änderungen merken und anschließend außerhalb einer Test-Session erneut umsetzen.

3.13.2 Auf dem Smartphone in VR testen (GearVR, GoogleVR)

Deine erste Scene ist jetzt bereit, auch auf deinem Smartphone getestet zu werden. Doch bevor wir loslegen können, musst du auf deinem Android-Smartphone zunächst die versteckten *Entwickleroptionen* aktivieren.

3.13.2.1 Android-Entwickleroptionen aktivieren

Das Aktivieren der *Entwickleroptionen* ist schnell erledigt und funktioniert auf allen Geräten ähnlich. Lediglich die Optik der Menüs könnte sich von Hersteller zu Hersteller unterscheiden. Manche Hersteller entscheiden sich nämlich dazu, von dem offiziellen Google-Design abzuweichen und ein eigenes zu verwenden.

1. Gehe auf deinem Smartphone zunächst in die *Einstellungen*. Bei den meisten Geräten ziehst du dafür die Benachrichtigungsleiste nach unten und klickst auf das **ZAHNRAD-ICON**.

2. In dem Einstellungsmenü musst du nun den Punkt **ÜBER DAS TELEFON** auswählen. Auf manchen Geräten heißt dieser Punkt auch **GERÄTEINFORMATIONEN** oder ist im Unterpunkt **SYSTEM** versteckt.

Das ist in der Regel der letzte Eintrag in dem Menü.

3. Jetzt musst du den Eintrag *Build-Nummer* bzw. *Buildnummer* finden.

Bei den meisten Geräten musst du dafür in diesem Menü nur noch ganz unten in der Liste scrollen.

Auf manchen Geräten musst du allerdings zuerst noch das Untermenü **SOFTWAREINFO** aufrufen und findest dort erst den Punkt *Buildnummer*.

4. Klicke jetzt **sieben Mal** auf den Eintrag **BUILD-NUMMER**. Wenn du die Meldung erhältst, dass du nun ein Entwickler seist, hast du alles richtig gemacht.

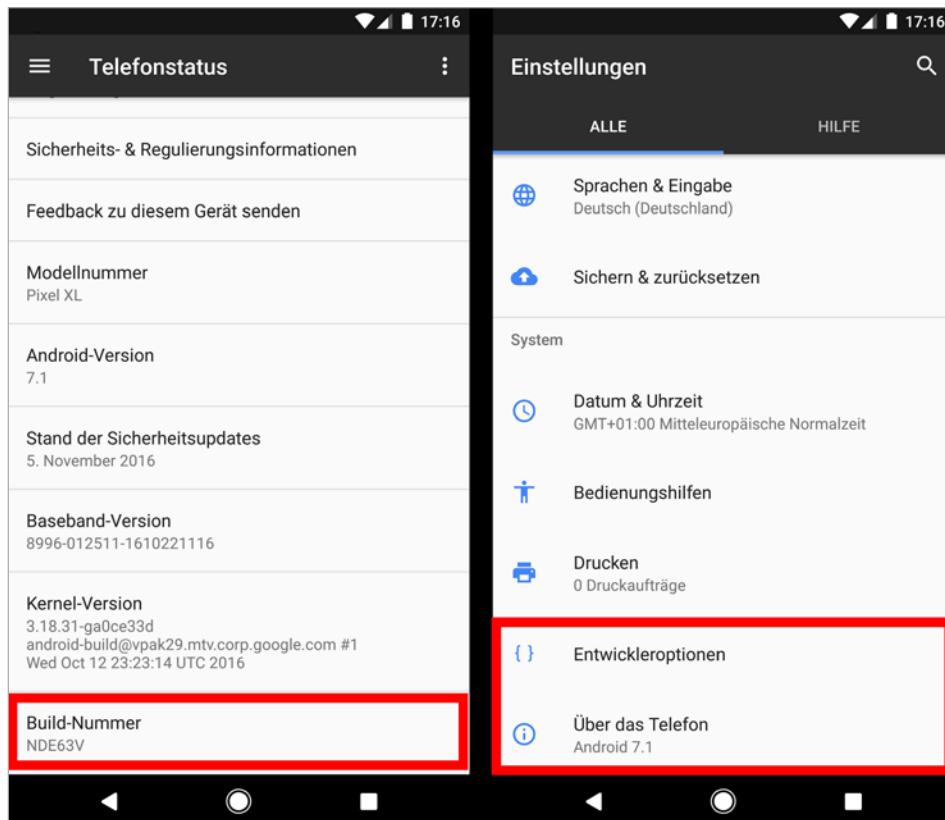


Bild 3.72 Markiert sind die für die Entwickleroptionen relevanten Menüpunkte (Foto: Google Pixel mit Android 7)

Wenn du jetzt wieder ein Menü wechselst, findest du dort einen neuen Eintrag **ENTWICKLEROPTIONEN**. Öffne dieses neue Untermenü, um die zahlreichen Optionen für Android-Entwickler zu sehen.

Scrolle herunter bis zu der Gruppe *Debugging*. Aktiviere hier, wie in Bild 3.73 links, die Option **USB-DEBUGGING**. Nachdem du auf den Eintrag geklickt hast, wird sich ein Pop-up-

Fenster mit einem Hinweis zu dem USB-Debugging öffnen. Lese dir den Hinweis durch und bestätige den Dialog mit **OK**.

Schließe dein Smartphone jetzt mit einem USB-Kabel an deinen Computer an. Es sollte sich dann auf deinem Handy ein weiteres *Popup* öffnen, in dem du, wie in Bild 3.73 rechts, das USB-Debugging für diesen Computer freigeben musst. Aktiviere die Option **VON DIESEM COMPUTER IMMER ZULASSEN** und wähle **OK**.

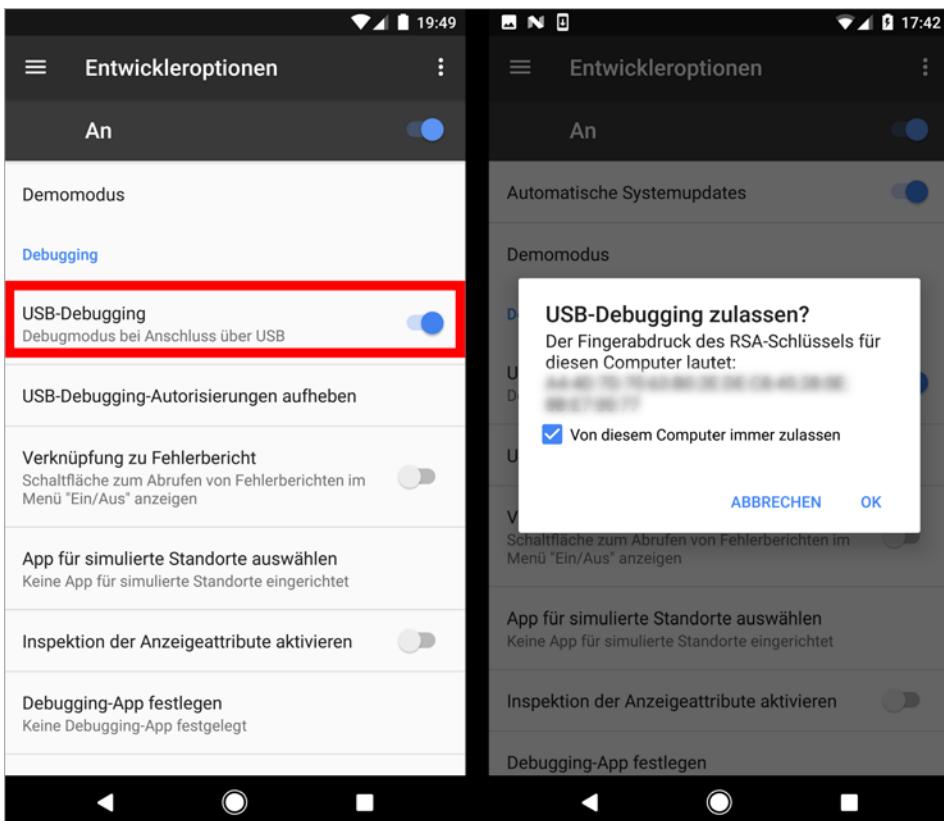


Bild 3.73 Die relevanten Schaltflächen, um USB-Debugging zu aktivieren (Foto: Google Pixel mit Android 7)



Die „USB-Debugging zulassen?“-Meldung erscheint nicht

Sollte die Meldung nicht erscheinen, solltest du als Erstes deinen PC einmal neu starten und anschließend die Option „USB Debugging“ in deinem Smartphone einmal aus- und wieder einschalten. Sollte das nicht helfen, blättere zurück zu Kapitel 3.1.2.2 und prüfe, ob die „Google USB Treiber“ korrekt installiert sind.

Das war es auch schon, dein Smartphone kann nun zum Testen und Entwickeln von Android-Apps verwendet werden.

3.13.2.2 Android Manifest-Fehler vorbeugen (GearVR)

Die Unity-Version *2017.1.0f3* enthält einen Fehler, der auftritt, wenn man versucht, einen GearVR-Build in Kombination mit einem aktuellen Android SDK zu erstellen. Um diesen Fehler zu umgehen, musst du das im Kasten verlinkte *Unity Package* in dein Projekt importieren.

Ohne das Package tritt beim Aufspielen der App ein Fehler beim Generieren der *AndroidManifest.xml* auf. Um den Fehler zu umgehen, enthält das *Unity Package* eine eigene *AndroidManifest*-Datei, mit der das Problem *Unable to merge android manifests* nicht auftritt.

Neuere Unity-Versionen sollen von diesem Problem nicht mehr betroffen sein. Probleme sollte das Unity Package bei einer neueren Version aber nicht verursachen, sodass du es sicherheitshalber trotzdem verwenden kannst.



GearVR Android Manifest Bugfix

<http://www.vrspieleentwickeln.de/zusatz/>

Auf der begleitenden Webseite in der Kategorie **Alle SDKs** findest du das *GearVR Android Manifest Bugfix*-Package. Lade es herunter und importiere es in dein Projekt, um GearVR Builds erstellen zu können.

3.13.2.3 Osig-Datei einfügen (GearVR)

Ich hatte in der Vorstellung der GearVR ja bereits angekündigt, dass du als Entwickler die Apps für die einzelnen Smartphones freigeben musst, wenn du sie außerhalb des Oculus-Stores installieren möchtest. Jetzt ist der Moment gekommen, an dem du dein Smartphone freischalten musst.

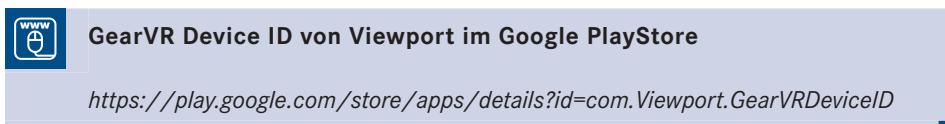
Der Freischaltvorgang erfolgt über eine Datei, die du auf der Oculus-Webseite generieren lassen und anschließend in einen speziellen Ordner in deinem Unity-Projekt legen musst. Diese Datei nennt sich *Osig*, was für „Oculus Signature File“ steht.

Um die Osig-Datei zu generieren, benötigst du als Erstes deine *Device ID*. Diese Device ID kann entweder mithilfe des Android SDK oder mit speziellen Apps aus dem *PlayStore* ausgelesen werden. Ich werde dir beide Wege beschreiben und du kannst dann selber entscheiden, welcher Weg für dich der geeigneter ist.

Osig mittels SideloadVR auslesen

In Googles *PlayStore* gibt es mehrere Apps, welche dir deine Device ID anzeigen können. Wenn du danach suchst, suche aber explizit nach GearVR-bezogenen Apps, da es in Android auch noch andere „Device IDs“ gibt.

In diesem Beispiel werden wir die App *GearVR Device ID* von dem Entwickler *Viewport* verwenden, da sie einfach zu bedienen ist und keine Berechtigungen bei der Installation benötigt.



Installiere die App, die ich dir in dem Kasten verlinkt habe, auf dem Samsung-Handy, das du freischalten möchtest. Nachdem die App installiert ist, kannst du sie starten und nach einem kurzen Ladevorgang zeigt sie dir wie in Bild 3.74 deine Device ID an. Je nach Smartphone kann deine Device ID auch deutlich kürzer oder länger als in dem Bild sein.



Bild 3.74 So einfach liest du die Device ID mittels einer App aus.

Osig mittels Android SDK auslesen

Über das *Android SDK* funktioniert das wie folgt:

In dem *sdk*-Ordner des *Android SDK* existiert ein Unterordner *platform-tools* und darin befindet sich ein Programm mit dem Namen *adb.exe*. Dieses Programm musst du mit dem Startparameter *devices* aufrufen, während dein Handy mit aktiviertem USB-Debugging an deinen PC angeschlossen ist.

1. Öffne die Windows *Eingabeaufforderung*.

2. Gib dort folgenden Befehl ein:

```
%APPDATA%\..\Local\Android\sdk\platform-tools\adb devices
```

(Wenn du das *Android SDK* nicht in das Standardverzeichnis installiert hast, musst du %APPDATA%\..\Local\Android\sdk\ durch deinen Installationspfad ersetzen.)

3. Du solltest dann eine Liste von angeschlossenen Geräten erhalten, in der, wie in Bild 3.75, auch deine Device ID angezeigt wird. Je nach Smartphone kann deine Device ID auch deutlich kürzer oder länger als in dem Bild sein.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. Alle Rechte vorbehalten.

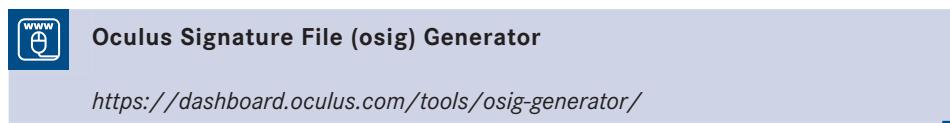
C:\Users\danie>APPDATA\Local\Android\sdk\platform-tools\adb devices
List of devices attached
0715f764    device
43a          device

C:\Users\danie>
```

Bild 3.75 So liest du deine Device ID über das Android SDK aus.

Osig generieren

Die Osig-Datei generierst du in dem Entwickler-Bereich der Oculus-Webseite. In dem Kasten unten habe ich dir die Adresse des Online-Werkzeugs eingefügt.



Oculus Signature File (osig) Generator
<https://dashboard.oculus.com/tools/osig-generator/>

Rufe die Webseite aus dem Kasten auf. Damit du Zugang zu dem Osig-Generator erhältst, musst du dich mit deinem Oculus- bzw. Facebook- Account einloggen. Dies ist der gleiche Account, den du auch in der *Oculus-App* auf deinem Handy verwendest.

Wenn du eingeloggt bist, siehst du das Formular auf Bild 3.76. Dort musst du einfach deine *Device ID* eintragen und auf **DOWNLOAD FILE** klicken. Es startet ein Download für eine Datei mit dem Namen *oculusig_{Device ID}*. Diese Datei darfst du auf keinen Fall umbenennen, da sie ansonsten nicht mehr funktioniert!

Speichere diese Datei an einem Ort auf deinem Computer, wo du sie gut wiederfindest. Du wirst sie immer wieder brauchen.

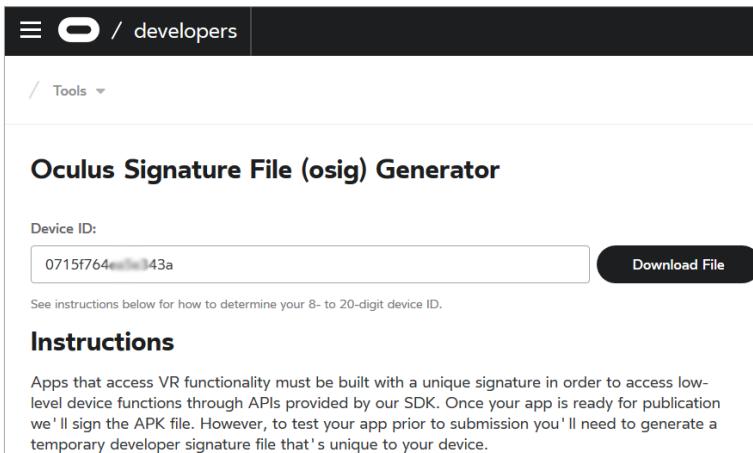


Bild 3.76 Nachdem du deine ID eingetragen hast, kannst du die Osig-Datei herunterladen.

Osig in das Projekt einfügen

Die Osig-Datei musst du jetzt in dein Unity-Projekt kopieren; allerdings nicht irgendwo hin, sondern in den Ordner */DEIN PROJEKT ORDNER/Assets/Plugins/Android/assets*. Wahrscheinlich existiert weder der Ordner noch der Dateipfad bereits in deinem Projekt, weshalb du ihn Schritt für Schritt im *Project Browser* anlegen musst.

Wie du Ordner anlegst, weißt du ja mittlerweile, deswegen werde ich es dir ersparen, das Anlegen des Pfades Schritt für Schritt zu beschreiben. Wichtig ist, dass du die Groß- und Kleinschreibung beachtest. Wenn du den Pfad angelegt hast, kannst du deine *oculusig_{Device ID}.osig*-Datei in den Ordner *assets* kopieren.

Bild 3.77 zeigt, wie die Ordnerstruktur und die Osig-Datei in deinem *Project Browser* aussehen sollten. Achte darauf, dass der *Plugins*-Ordner wie im Bild *neben* deinem *MyAssets*-Ordner liegt und *nicht darin*.

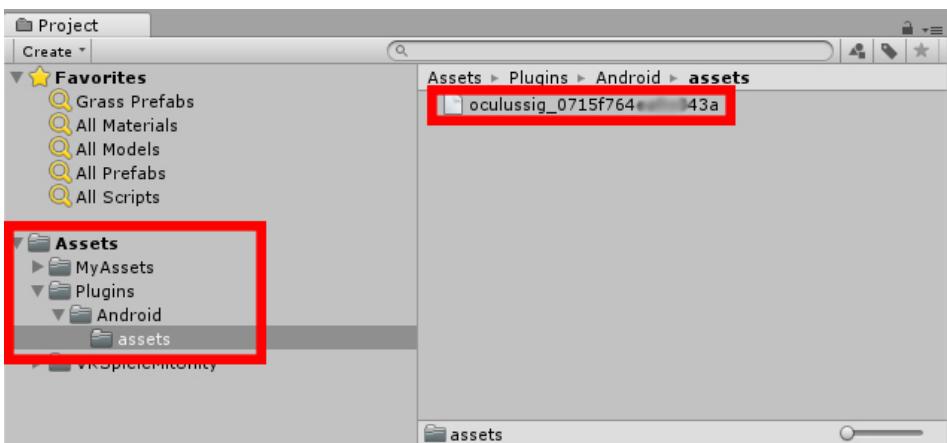


Bild 3.77 In deinem Project Browser musst du den Pfad „Plugins/Android/assets“ anlegen und dort die Osig-Datei einfügen.

3.13.2.4 Einen Unity Build erstellen

Jetzt bist du bereit, deinen ersten *Build* zu erstellen und ihn auf deinem Smartphone auszuführen. Im *Unity Editor* musst dazu zunächst die *Build Settings* öffnen. Wähle dafür in der Toolbar **FILE/BUILD SETTINGS....**. Es öffnet sich das Menü, in dem du auch die Zielplattform geändert hastest.

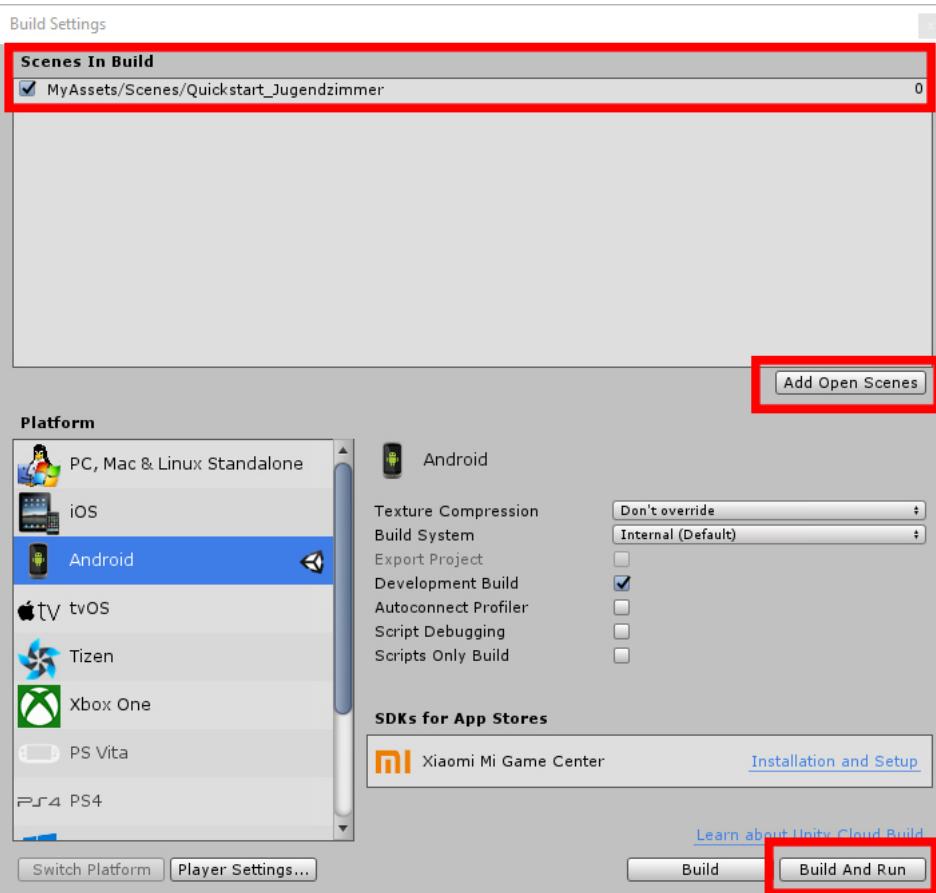


Bild 3.78 Die für *Build and Run* relevanten Schaltflächen

Nicht jede Scene in deinem Projekt wird automatisch zu deinen *Builds* hinzugefügt, das musst du per Hand vornehmen. Klicke auf die Schaltfläche **ADD OPEN SCENES**, um die aktuell geöffnete Scene zu dem *Build* hinzuzufügen. Wenn du später mehrere Scenes in einem *Build* haben solltest, bestimmt die Reihenfolge in der *Scenes in Build*-Liste, welche Scene bei Anwendungsstart als Erstes geladen wird. Die Start-Scene ist immer die mit dem Index 0. Über die Checkbox neben dem Scene-Namen kannst du Scenes vorübergehend aus dem Build entfernen und später wieder hinzufügen. Über die *Entf-Taste* auf der Tastatur kannst du Scenes aus der Liste löschen.

Am unteren, rechten Rand des Fensters findest du zwei Optionen: *Build* und *Build and Run*. Nachfolgend erkläre ich dir die Unterschiede:

- **Build:** Erstellt eine *.apk-Datei, welche du per Hand auf dein Smartphone kopieren und von dort aus installieren kannst
- **Build and Run:** Erstellt eine *.apk-Datei und installiert sie automatisch auf deinem Smartphone, wenn es via USB angeschlossen ist. Nach der Installation wird die App außerdem automatisch gestartet.

Wähle **BUILD AND RUN**, um den Build-Vorgang zu starten. Die apk-Datei kannst du an einem Ort deiner Wahl speichern, nur nicht in dem Asset-Ordner deines Projektes. Im *Unity Editor* öffnet sich anschließend eine Fortschrittsanzeige mit dem aktuellen Build-Status. Wenn die Fortschrittsanzeige verschwunden ist, sollte die App auf deinem Smartphone gestartet werden. Damit das automatische Starten problemlos funktioniert, sollte das Gerät zu diesem Zeitpunkt entsperrt sein. War dein Gerät gesperrt und die App wurde nicht automatisch gestartet, kannst du sie per Hand über die App-Übersicht deines Smartphones starten.

Wenn alles geklappt hat, solltest du dich in deiner Test-Scene wiederfinden und umsehen können.



Wenn der Build-Vorgang fehlschlägt, findest du entsprechende Fehlermeldungen in der *Console*.

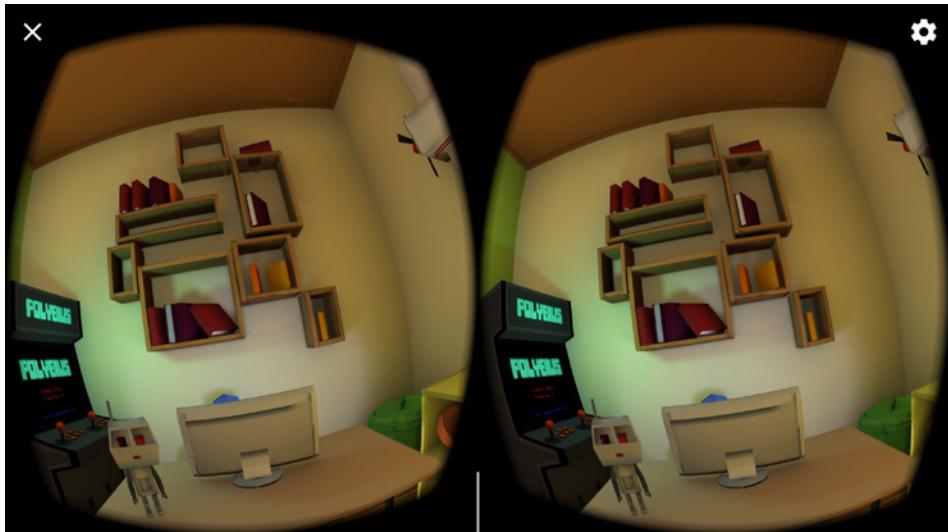


Bild 3.79 So sollte die Test-Scene auf deinem Smartphone aussehen.

GearVR: Thread Priority Exception

Wenn du beim Starten deiner App die Fehlermeldung *Thread Priority Exception. Make sure your APK is signed* siehst, stimmt etwas mit deiner Osig-Datei nicht. Blättere zu Kapitel 3.13.2.1.3 zurück und prüfe, ob sich vielleicht ein Fehler eingeschlichen hat.

■ 3.14 So geht es weiter

Die nächsten Kapitel werden dich noch tiefer in die Themen Virtual Reality, Unity und C#-Programmierung einführen. Das Wissen aus diesen Kapiteln ist eine Grundlage für die Beispielprojekte. Aber keine Angst, du musst die Kapitel nicht auswendig können, um die Beispielprojekte verstehen und umsetzen zu können. Die Kapitel sollen dir auch als Nachschlagewerk dienen, falls du in Zukunft Fragen zu einzelnen Themen hast.