
carsten SEIFERT
jan WISLAUG
Best-seller
3. Auflage
SPIELE ENTWICKELN MIT Unity 5
2D- UND 3D-GAMES MIT UNITY UND C# FÜR DESKTOP, WEB & MOBILE
Aktualisiert auf Unity 5.6
HANSER
»Lässt von der Programmierung bis zur Umsetzung von Spielekonzepten keine Fragen offen.« c't

Seifert / Wislaug

Spiele entwickeln mit Unity 5

Carsten Seifert
Jan Wislaug

Spiele entwickeln mit Unity 5

2D- und 3D-Games mit Unity und C# für Desktop, Web & Mobile

3., aktualisierte Auflage

HANSER

Die Autoren:

Carsten Seifert, Süderbrarup, *www.hummelwalker.de*

Jan Wislaug, Witten, *www.jan-wislaug.de*

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

Lektorat: Sylvia Hasselbach
Copy editing: Sandra Gottmann, Münster-Nienberge
Erstellung der Dateien für das Beispiel-Game: Alexej Bodemer, Stuhr, *www.alexejbodemer.de*
Umschlagdesign: Marc Müller-Bremer, München, *www.rebranding.de*
Umschlagrealisation: Stephan Rönigk
Gesamtherstellung: Kösel, Krugzell
Printed in Germany

Print-ISBN: 978-3-446-45197-1
E-Book-ISBN: 978-3-446-45368-5

Inhalt

Vorwort

Für viele von uns sind Computerspiele und Handygames heutzutage allgegenwärtige Wegbegleiter. Egal ob auf dem Smartphone, dem Tablet, installiert auf dem heimischen PC oder direkt aufgerufen im Browser werden sie von uns täglich genutzt. Manchmal dienen sie als Zeitvertreib, bis der nächste Bus kommt, manchmal sind sie aber auch Bestandteil eines intensiven Hobbys.

Aber nicht nur das Spielen kann Spaß machen, auch das Entwickeln dieser Games kann begeistern. Sowohl im Freizeitbereich als auch in der Arbeitswelt wird der Beruf des Spieleentwicklers immer beliebter. Es ist also kein Wunder, dass mittlerweile viele, teilweise sogar staatlich anerkannte, Studiengänge existieren, die sich dem Entwickeln von Computerspielen widmen.

In diesem Buch möchten wir Ihnen Unity, eine weit verbreitete Entwicklungsumgebung für Computerspiele und auch andere Anwendungen, näherbringen. Wir erklären und erläutern ausführlich, wie Sie mit diesem Werkzeug Spiele entwickeln können. Dabei richtet sich das Buch sowohl an Einsteiger und Umsteiger als auch an Spieleentwickler, die mit Unity nun richtig durchstarten möchten.

Als ursprünglicher Autor dieses Buches möchte ich, Carsten Seifert, mich an dieser Stelle ganz besonders bei meiner Frau Cornelia bedanken, die mich während des Schreibens so geduldig unterstützt hat und mir jederzeit beim Formulieren und Korrigieren hilfsbereit zur Seite stand.

Weiter möchte ich Sieglinde Schärl, Kristin Rothe und dem gesamten Hanser-Verlag-Team danken, die mir nicht nur das Schreiben dieses Buches ermöglicht haben, sondern auch bei der Arbeit an den ersten beiden Auflagen des Buches jederzeit mit Rat und Tat zur Seite standen.

Auch danke ich ganz herzlich Alexej Bodemer, der für das Beispiel-Game dieses Buches alle 3D-Modelle, Texturen und Musikdateien entworfen und zur Verfügung gestellt hat.

Ich, Jan Wislaug, möchte mich als überarbeitender Autor vor allem bei Sylvia Hasselbach vom Hanser Verlag bedanken. Sie stand mir bei der Arbeit an der dritten Auflage immer freundlich mit Rat und Tat beiseite.

Darüber hinaus danke ich Carsten Seifert für die gute Zusammenarbeit und dafür, dass sich sein Werk dank der guten Struktur so toll überarbeiten ließ.

Zusammen möchten wir beide Will Goldstone und Unity Technologies danken, die uns die bis dato aktuellsten Beta-Versionen zur Verfügung gestellt haben.

Nicht zuletzt danken wir auch der gesamten Community, die zum einen Carsten Seifert auf seinem Blog und seinen sozialen Kanälen begleitet hat und zum anderen Jan Wislaug bei seiner Überarbeitung auf die neue Unity-Version unterstützte.

Carsten Seifert und *Jan Wislaug*

im Juli 2017

1 Einleitung

Computerspiele gehören heutzutage zu den beliebtesten Freizeitgestaltungen unserer Zeit. Mit Zunahme der Popularität ist aber auch der Anspruch an diese Spiele gestiegen. Während in den ersten Jahren bis Jahrzehnten dieser jungen Branche noch ein einziger Programmierer ausreichte, um alle notwendigen Aufgaben zu erledigen, werden anspruchsvolle Computerspiele heutzutage meist von großen Teams umgesetzt. Hier arbeiten 3D-Modellierer, Grafiker, Sounddesigner, Level-Designern und natürlich auch Programmierer aus unterschiedlichen Sparten Hand in Hand.

Um den stetig wachsenden Ansprüchen zu genügen, sind aber auch die Werkzeuge der Entwickler ständig mitgewachsen. Eines dieser Werkzeuge ist Unity. Unity ist eine Spieleentwicklungsumgebung für Windows- und Mac OS X-Systeme und wird von der aus Dänemark stammenden Firma Unity Technologies entwickelt. Mit ihr können Sie sowohl interaktive 3D- als auch 2D-Inhalte erstellen. Wir sprechen deshalb von Inhalten und nicht nur von Spielen, weil Unity zwar eigentlich für die Entwicklung von 3D-Spielen gedacht war, mittlerweile aber auch immer häufiger Anwendung in anderen Bereichen findet. So wird es beispielsweise für Architekturvisualisierungen genutzt, im E-Learning-Bereich eingesetzt oder in der Digital-Signage-Branche für das Erstellen digitaler Werbe- und Informationssysteme genommen.

Da Unity ursprünglich für die Entwicklung von 3D-Spielen konzipiert wurde, lautet die Internet-Adresse der Firma *unity3d.com*. Dies ist der Grund, weshalb die Entwicklungssoftware auch gerne mal „Unity3D“ genannt wird, was aber eben nicht ganz korrekt ist.

1.1 Multiplattform-Publishing

Eine besondere Stärke von Unity ist die Unterstützung von Multiplattform-Publishing. Das bedeutet, dass Sie in Unity ein Spiel einmal entwickeln können, das Sie dann aber für mehrere Plattformen exportieren können. Aktuell werden mehr als 25 Plattformen unterstützt, die sich grob in folgende Kategorien einteilen lassen:

- Die sogenannten „Standalones“. Hierzu gehören: Windows Desktop, Mac, Linux (und Steam OS)

- Die mobilen Smartphone-/Tablet-Plattformen: iOS, Android, Windows Phone, Fire OS
- Die Spielekonsolen: PlayStation 4, PlayStation Vita, Xbox One, Nintendo Switch, Wii U, Nintendo 3DS
- Die Kategorie „Smart-TV“. Hier werden folgende Systeme unterstützt: AndroidTV, Samsung SMART-TV, tvOS

In Verbindung mit diesen Plattformen bietet Unity die Möglichkeit, Inhalte direkt für verschiedene Virtual-Reality-Systeme zu entwickeln. Hierzu gehören: Occulus-Rift, Google-Cardboard, Steam-VR, Playstation-VR, Gear-VR, Microsoft Hololens und Daydream.

Auch für Web-Inhalte bietet Unity entsprechende Features. Mit der Plattform „WebGL“, die im Browser läuft und auf HTML5 basiert, können Sie Nutzern direkten Zugriff auf Ihre Inhalte geben.

Bei den erwähnten Spielekonsolen müssen Sie allerdings noch eine Einschränkung beachten: Sie müssen hier extra Lizenzen erwerben, die zum einen nicht günstig und zum anderen nur für Firmen verfügbar sind, die von den jeweiligen Konsolenherstellern auch als Entwickler akzeptiert wurden. Deshalb werden wir die Konsolenentwicklung in diesem Buch auch außen vor lassen und nicht näher beleuchten.

1.2 Das kann Unity (nicht)

Unity bringt eine ganze Reihe an nützlichen Werkzeugen mit, um Spiele und andere 2D- und 3D-Anwendungen zu entwickeln. So gibt es neben einer ausgeklügelten Physik-Engine auch Tools für Partikeleffekte, zur Landschaftsgestaltung oder auch für Animationen. Außerdem wird Unity mit einer extra angepassten Version der Softwareentwicklungsumgebung MonoDevelop ausgeliefert, in der die Programmierung umgesetzt und das Debugging vorgenommen werden kann.

Eines ist Unity allerdings nicht: Es ist keine 3D-Modellierungssoftware. Unity bietet zwar von Haus aus einige 3D-Grundobjekte an, sogenannte Primitives, die für kleinere Aufgaben genutzt werden können, aber für richtige Modellierungsaufgaben sollten Sie auf die dafür entwickelten Spezialtools wie 3ds Max oder das kostenlose Blender zurückgreifen.

1.3 Lizenzmodelle

Sie haben die Möglichkeit, auf verschiedene Modelle zurückzugreifen, wenn Sie Unity nutzen möchten. Zum eine gibt es eine komplett kostenfreie Variante, welche bereits alle oben genannten Zielplattformen unterstützt und eine komplette Spieleentwicklung bis hin zum fertigen Spiel erlaubt. Es gibt jedoch auch einige Einschränkungen, die sich allerdings nur auf erweiterte oder visuelle Features beziehen. So müssen Sie zum Beispiel auf die dunkle Oberfläche (Skin) Ihres Unity-Programms verzichten oder haben auch keinen Zugriff auf den Premium-Support von Unity.

Bei der kostenpflichtigen Variante unterscheidet man die drei Abo-Modelle „Plus“, „Pro“ und „Enterprise“, welche jeweils monatlich zu bezahlen sind. Je teurer das Abo ist, umso mehr Funktionen stehen zur Verfügung.

Da sowohl die kostenfreie als auch die kostenpflichtige Edition kommerziell genutzt werden darf, gibt es beim Einsatz der kostenlosen Version die Vorgabe, dass nur Firmen bzw. Entwickler diese einsetzen dürfen, die nicht mehr als 100 000 US-Dollar Umsatz in einem Geschäftsjahr machen.

1.4 Aufbau und Ziel des Buches

Mit diesem Buch werden Sie lernen, auf Basis von Unity eigene 2D- und 3D-Spiele zu entwickeln. Sie werden sich mit den unterschiedlichen Tools der Game Engine vertraut machen und die Skript-Programmierung erlernen. Dabei steht aber nicht das Ziel im Fokus, wirklich jede einzelne Funktion und Möglichkeit zu beleuchten, die Unity dem Entwickler anbietet. Vielmehr zeigen wir Ihnen, wie die unterschiedlichen Bereiche von Unity funktionieren und miteinander zusammenarbeiten. Denn der Funktionsumfang dieser Spieleentwicklungsumgebung ist mittlerweile so umfangreich geworden, dass es gar nicht mehr möglich ist, alle Tools und deren Möglichkeiten bis ins letzte Detail in einem einzigen Buch ausführlich zu behandeln. Daher liegt der Schwerpunkt auf den Kernfunktionen und wichtigen Optimierungstechniken, die in Unity für die 2D- und 3D-Spieleentwicklung bereitgestellt werden, ergänzt um Anwendungsbeispiele und Praxistipps.

Möchten Sie weiter in die Tiefe eines speziellen Tools gehen oder mehr Informationen zu bestimmten Scripting-Klassen erhalten, empfehlen wir Ihnen die mit Unity mitgelieferten Hilfe-Dokumente, die Sie über das Help-Menü erreichen. Dort finden Sie sowohl ein ausführliches Manual über die in Unity integrierten Tools sowie eine *Scripting-Referenz* über alle Unity-Klassen und deren Möglichkeiten. Letztere finden Sie auch über die Hilfe von MonoDevelop, der mitgelieferten Programmierumgebung.

Spieleentwicklung wird häufig auch als Spieleprogrammierung bezeichnet, auch wenn viele Aufgaben in der Spieleentwicklung heutzutage nicht mehr programmiert, sondern mithilfe von Tools erledigt werden. Nichtsdestotrotz ist der Programmieranteil bei der Entwicklung eines Spiels doch immer noch sehr hoch. Deshalb wird auch das Buch zunächst mit zwei größeren programmierbezogenen Kapiteln beginnen. Das erste behandelt allgemeine Grundlagen der Programmierung, das zweite geht auf die Unity-spezifischen Themen ein. Erst danach werden wir in die 3D-Welt von Unity eintauchen und die verschiedenen Werkzeuge behandeln. Der Aufbau ist deshalb so gewählt, weil wir in den folgenden Kapiteln immer wieder kleinere Skript-Beispiele zeigen, die Einsatzmöglichkeiten in der Praxis demonstrieren.

Ganz grundsätzlich ist die Reihenfolge der einzelnen Kapitel so gewählt, dass die Inhalte aufeinander aufbauen. Aus diesem Grund kommen auch erst zum Ende des Buches die beiden Beispiel-Games, eines für 2D und eines für 3D. In diesen werden alle angesprochenen Themen noch einmal aufgegriffen und die gesamten Zusammenhänge in der Praxis gezeigt. Nichtsdestotrotz können Sie gerne diese Kapitel auch vorziehen. Speziell das 2D-Spiel eignet sich hierfür gut.

Im Buch werden wir Hinweiskästen, wie den obigen, einsetzen, um Hinweise und Tipps zu geben. Je nach Typ des Hinweises werden diese mit unterschiedlichen Icons ausgezeichnet.


Praxistipps

Hinweise zu weiterführenden Inhalten im Internet


Allgemeine Hinweise

1.5 Weiterentwicklung von Unity

Die Spieleindustrie gehört zu den Branchen, die sich aktuell am schnellsten verändern. Kein Wunder also, dass auch Unity ständig weiterentwickelt wird und neue Funktionen erhält. Sollten Sie Unterschiede zwischen dem Buch und Ihrer Unity-Version erkennen, wird dies sicher der ständigen Weiterentwicklung von Unity geschuldet sein.

Aber nicht nur der Funktionsumfang wird ständig weiterentwickelt, auch das Lizenzmodell von Unity ist nicht von Veränderungen ausgeschlossen. In den letzten Jahren hat sich dieses zunehmend geändert. Seit es die kostenlose Version von Unity gibt, wurde deren Funktionsumfang gravierend erweitert (genauso wie der der Pro-Version). So standen anfangs nur Basisfunktionalitäten zur Verfügung. Später kamen dann die Mobile-Export-Möglichkeiten hinzu, bis 2015 schließlich auch alle anderen Pro-Features der Engine in die kostenlose Version integriert wurden, wobei noch eine Ausnahme besteht. So lässt die Personal Edition mittlerweile zwar den *Customizable Splash Screen* zu, aber es erscheint vor jedem Spiel, das mit dieser Version erstellt wurde, immer noch der Unity-Schriftzug. Besitzen Sie allerdings Unity Professional, können Sie auch das anpassen.

Aktuell arbeitet Unity mit Facebook zusammen, um die Plattform „Facebook Gameroom“ weiter voranzubringen. Auch der Bereich „Virtual Reality“ wird kontinuierlich ausgebaut. In kommenden Versionen der Software wird sich also immer wieder etwas tun, um ein noch breiteres Spektrum an Funktionen zu bieten. Auf der Firmen-Website von Unity Technologies können Sie sich immer über den aktuellen Entwicklungstand informieren.

1.6 Online-Zusatzmaterial

Auf einer Unterseite meiner Homepage stelle ich Ihnen einen Bereich zur Verfügung, in dem Sie Zusatzmaterialien erhalten. Im oberen, für jedermann sichtbaren Abschnitt, befindet sich eine Sammlung an kleinen Ausbesserungen und Hinweisen zum Buch. Darunter finden Sie die Beispielprojekte, welche aber nur nach einem erfolgreichen Login sichtbar sind.

Passwortgeschützter Bereich für Zusatzmaterialien zum Buch

URL: *http://jan-wislaug.de/Unitybuch_ZusatzMaterialien.htm*

Benutzername: Leser

Passwort: **Unity5Buch!**

In dem geschützten Bereich finden Sie Links zu Projektdaten unter anderem für:

- Ein touch-basiertes 2D-Beispiel-Game mit allen dazugehörigen Ressourcen
- Ein 3D-Beispiel-Game (Dungeon Crawler) mit allen dazugehörigen Ressourcen
- Ein Anwendungsbeispiel für eine Auto-Steuerung inklusive eines 3D-Modells mit zusätzlichen Skripten
- Ein Beispiel für eine Sprite-Animation
- Ein Anwendungsbeispiel für Parallax Scrolling
- Eine Vorlage für ein einfaches Übungsprojekt
- Mehrere ergänzende Video-Tutorials

2 Grundlagen

In diesem Kapitel werden Sie die Oberfläche von Unity sowie deren grundlegende Bedienung kennenlernen. Wir werden auf die Struktur eines Unity-Projektes eingehen und die grundlegenden Prinzipien von Unity behandeln.

2.1 Installation

Sollten Sie noch keine Installationsdatei haben, besuchen Sie zunächst die Seite *https://store.unity.com* bzw. *http://www.unity3d.com* und laden sich dort die aktuelle Unity-Version herunter. Nach dem Download können Sie Unity installieren. Da die Installation eigentlich selbsterklärend ist, sollten Sie lediglich darauf achten, dass Sie eine Komplettinstallation machen und keine Teile davon ausnehmen. Profis können natürlich selber bewerten, was sie benötigen, Anfängern würden wir aber immer eine Komplettinstallation empfehlen.

Wenn Sie die Installation abgeschlossen haben und Unity das erste Mal starten, wird sich das Projekt-Fenster von Unity zeigen. Über dieses können Sie ein existierendes Projekt öffnen oder ein neues Projekt erstellen. Außerdem finden Sie in der unteren Leiste des Fensters Web-Links zum Community-Bereich, zur Online-Dokumentation sowie zum Tutorial-Bereich von Unity Technologies.

Mehr zum Öffnen und Erstellen eines Projektes erfahren Sie im Abschnitt „Das Unity-Projekt“. Für den ersten Start können Sie einfach die Standardeinstellungen übernehmen und über NEW PROJECT ein neues Projekt erstellen.

2.2 Oberfläche

Die Oberfläche von Unity besteht aus mehreren frei anpassbaren Fenstern (auch Tabs genannt), die sich innerhalb von Unity übereinander und nebeneinander andocken sowie auch außerhalb des Hauptfensters verschieben lassen.

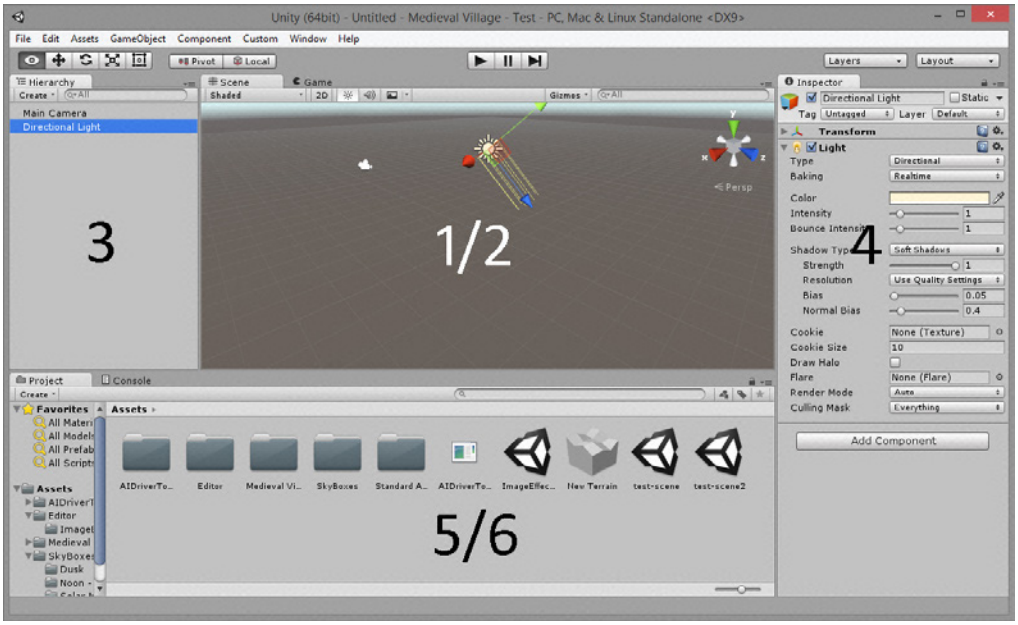

Bild 2.1 Default-Ansicht von Unity

Die Fenster (Tabs) besitzen zwar unterschiedliche Aufgaben, gehören funktional aber alle zusammen und werden deshalb auch kontinuierlich untereinander synchronisiert.

1. **Scene View** dient dem interaktiven Gestalten von Szenen und 3D-Welten.
2. **Game View** dient als Vorschau des fertigen Spiels. Dieses Fenster wird in Bild 2.1 von der *Scene View* verdeckt, wird aber von Unity automatisch nach vorne gebracht, wenn das Spiel gestartet wird.
3. **Hierarchy** zeigt alle in der Szene existierenden Objekte (*GameObjects*) in deren Hierarchiestruktur an.
4. **Inspector** zeigt alle Komponenten und öffentlichen Parameter des aktuell selektierten *GameObjects* an.
5. **Project Browser** dient dem Anzeigen und Verwalten aller digitalen Inhalte (auch *Assets* genannt), die zu dem Projekt gehören.
6. **Console** dient dem Anzeigen von Fehler- und Hinweismeldungen. Dieses Fenster befindet sich in der Default-Ansicht hinter dem *Project Browser*. Um die Meldungen zu sehen, müssen Sie auf den Reiter des Fensters klicken.

Abgesehen von der *Game View* und dem *Console -Tab* können Sie den Tabs nicht nur Informationen entnehmen, sondern auch die Objekte verändern.

Sie können die Tab-Anordnungen in Unity jederzeit speichern und auch zurücksetzen. Hierfür stellt Unity im oberen Hauptmenü über WINDOW/LAYOUTS entsprechende Funktionen und Standardanordnungen zur Verfügung. Als Schnellzugriff dient hier das *Layouts-Dropdown-Menü*, das Sie ganz rechts im oberen Bereich von Unity finden.

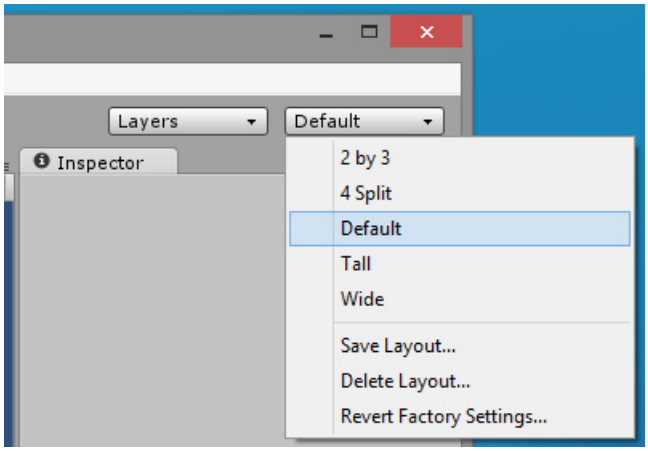

Bild 2.2 Schnellzugriff auf die Layout-Funktionen

Wenn Sie mit SAVE LAYOUT eigene Fenster-Anordnungen speichern, stehen diese Layouts nicht nur in diesem Projekt zur Verfügung, sondern auch in allen anderen Unity-Projekten. Mit DELETE LAYOUT löschen Sie diese natürlich ebenfalls in allen Projekten. Sollten Sie dabei eine Standardsicht aus Versehen gelöscht haben, können Sie über REVERT FACTORY SETTINGS alle Layout-Einstellungen wieder auf die Werkseinstellungen zurücksetzen.

2.2.1 Hauptmenü

Oberhalb aller Subfenster und Toolbars befindet sich das Hauptmenü von Unity. Viele Teile dieses Menüs finden Sie zusätzlich noch einmal als Schnellzugriff-Menüs in anderen Bereichen von Unity wieder, wie z. B. das *Layouts-Drop-down-Menü* (siehe oben).

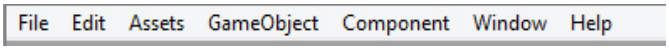

Bild 2.3 Hauptmenü

Hinter diesen Hauptmenüpunkten verbergen sich folgende Funktionen:

- **File** beinhaltet alle Funktionen, die sich mit dem Erstellen und Speichern von Dateien auf Projekt- und Szene-Ebene beschäftigen.
- **Edit** besitzt vor allem Einstellungen zum Verändern von Daten.
- **Asset** beschäftigt sich mit dem Verwalten und Erstellen neuer *Assets*, z. B. von Skripten und Materialien.
- **GameObject** beschäftigt sich vor allem mit dem Erstellen verschiedener, vorkonfigurierter *GameObjects*.
- **Component** bietet alle Standardkomponenten von Unity an.
- **Window** bietet vor allem weitere Fenster/Tabs an, die Sie anzeigen können.
- **Help** besitzt hauptsächlich Quellen und Links zu weiterführenden Informationen für Unity. Hier gelangen Sie auch zum „Unity Manual“ und zu der „Scripting Reference“, wo Sie viele weiterführende Informationen finden.

2.2.2 Scene View

Das Fenster mit der Überschrift „Scene“ wird als *Scene View* bezeichnet und dient dem interaktiven Positionieren und Verändern von Objekten innerhalb einer Szene bzw. eines Levels. Sie können hier neue Objekte hinzufügen, diese verschieben, rotieren und auch skalieren. Ein wichtiges Werkzeug hierfür sind die *Transform-Tools*, die Sie in der Toolbar direkt unter dem Hauptmenü oben links finden. Über Maus und Tastatur können Sie zudem durch die Szene navigieren (siehe Abschnitt 2.2.2.1, „Navigieren in der Scene View“).

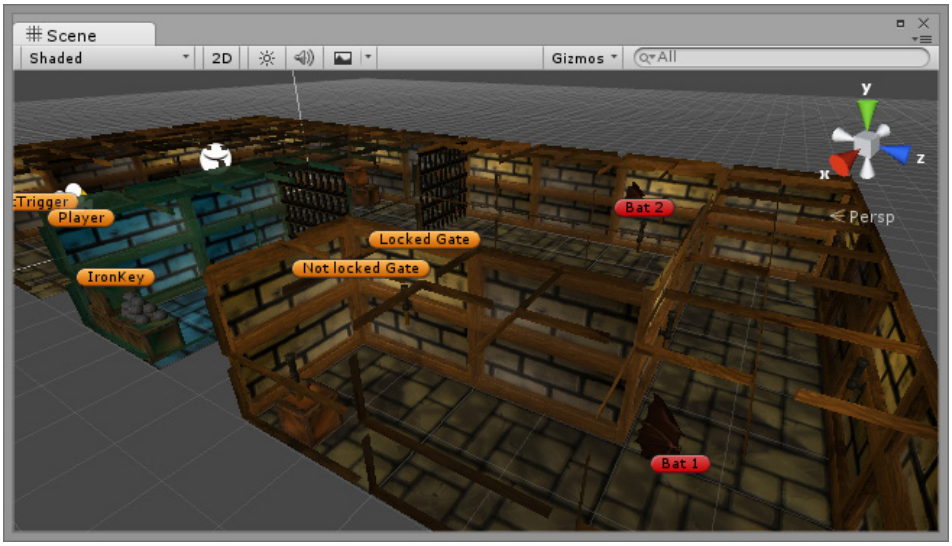

Bild 2.4 Scene View

Möchten Sie ein neues *GameObject* in der *Scene View* platzieren, können Sie einfach ein *GameObject* aus dem *Project Browser* in die *Scene View* hineinziehen und fallen lassen. Anschließend können Sie es noch verschieben und mit den erwähnten *Transform-Tools* modifizieren (siehe Abschnitt 2.2.4, „Toolbar“).

Am oberen Rand der *Scene View* befindet sich eine schmale *Control Bar*, die Ihnen über mehrere Untermenüs verschiedene Einstellmöglichkeiten bietet (siehe Bild 2.4).

- **Rendering-Menü** wechselt die Ansicht zwischen verschiedenen Darstellungsarten (Texturiert, Drahtgitter usw.).
- **2D-Button** wechselt zwischen der normalen 3D-Ansicht und einer speziellen Darstellungsform, die besonders für 2D-Games und das GUI-Design geeignet ist. Mehr zu dieser Ansicht erfahren Sie in Abschnitt 2.2.2.2.
- **Beleuchtungs-Button** schaltet die Darstellung der Lichtquellen bzw. deren Auswirkungen in der *Scene View* ein/aus.
- **Sound-Button** schaltet den Ton ein/aus.
- **Effects-Toggle** ermöglicht, Effekte in der Darstellung der *Scene View* ein und auszuschalten. Über einen kleinen Pfeil neben dem EFFECTS-Button, können Sie auch einzelne Effekte in der Szene aktivieren/deaktivieren. Hierzu gehören Skybox, Fog, Flares und Animated Materials . . .).

- **Gizmos -Menü** schaltet typbezogen Hilfsgrafiken in der *Scene View* ein/aus (Gizmos sind grafische Hilfsmittel, um nicht sichtbare Objekte wie Kameras, Soundquellen etc. darzustellen).
- **Suchmaske**, um Objekte in der Szene zu suchen (dabei wird alles ausgegraut, außer die gefundenen Objekte)

2.2.2.1 Navigieren in der Scene View

Unity bietet unterschiedliche Möglichkeiten, durch die *Scene View* zu navigieren. Manche Funktionen sind hierbei auch auf unterschiedliche Weise zu erreichen. Im Folgenden möchten wir Ihnen die wichtigsten vorstellen.

- **Objekte selektieren** Sie mit der linken Maustaste.
- **Ansicht in der Höhe und Seite verschieben** Sie mit der mittleren Maustaste bzw. durch Drücken auf das Mausrad. Sie können auch über die Pfeiltasten navigieren oder alternativ auch das Verschiebewerkzeug der *Transform-Tools* aktivieren (Shortcut: Q) und dann die linke Maustaste nutzen (siehe *Transform-Tools*). Wenn Sie große Szenen besitzen, können Sie die Geschwindigkeit des Verschiebens über das zusätzliche Drücken der UMSCHTASTE SHIFT noch erhöhen.
- **Ansicht in der Tiefe und Seite verschieben** Sie über die Pfeiltasten. Hierbei bewegen Sie sich den Pfeiltasten entsprechend durch die Szene.
- **Ansicht drehen** Sie über die rechte Maustaste. Alternativ können Sie auch die Kombination aus ALT-Taste und der linken Maustaste nutzen.
- **Ansicht zoomen** Sie über das Drehen des Mausrads. Alternativ können Sie auch die ALT-Taste und die rechte Maustaste drücken, während Sie den Mauszeiger bewegen.
- **Objekte fokussieren** Sie, indem Sie ein Objekt in der Scene View oder in der *Hierarchy* selektieren und dann die Taste F drücken. Die Ansicht wird dann so gesetzt, dass das Objekt im Mittelpunkt steht.

2.2.2.2 Scene Gizmo

Oben rechts innerhalb der *Scene View* finden Sie das *Scene Gizmo*, das die Orientierung des Koordinatensystems zur aktuellen Sicht der *Scene View* darstellt. Das *Scene Gizmo* ist dabei wie ein Kompass und zeigt Ihnen, von welcher Seite Sie aktuell die Szene betrachten.

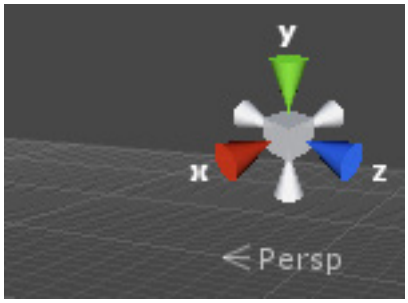

Bild 2.5 Scene Gizmo

Wenn Sie auf die unterschiedlichen Achsen des *Scene Gizmos* klicken, wechselt die Ansicht dabei in die jeweilige Seiten-, Drauf- oder Frontansicht. Ein Klick auf den mittleren Würfel ändert zudem die Ansicht zwischen orthogonal und perspektivisch.

- **Orthogonal** ignoriert Tiefenverhältnisse. So werden beispielsweise zwei gleich große Objekte immer gleich groß dargestellt, egal wie weit diese vom Betrachter entfernt sind.
- **Perspektivisch** stellt Objekte „natürlich" dar und berücksichtigt die Entfernung vom Betrachter.

Unter dem *Scene Gizmo* wird Ihnen in Textform zusätzlich angezeigt, welche Ansicht Sie aktuell gewählt haben. Die drei parallel verlaufenden Linien bedeuten hierbei orthogonal, die auseinandergehenden Linien bedeuten perspektivisch. Ein Klick auf diesen Text wechselt ebenfalls zwischen diesen beiden Ansichtsformen.

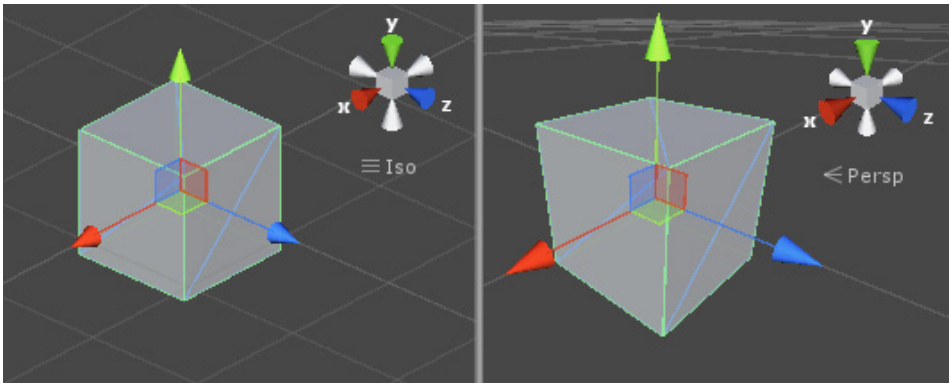

Bild 2.6 Vergleich: Orthogonal (links) vs. Perspektivisch (rechts)

2D-Button

Der *2D-Button*, den Sie in der *Control Bar* der *Scene View* finden, ändert lediglich die Ansicht auf „Orthogonal" und dreht die räumliche Betrachtung so, dass der Nutzer in Richtung der positiven Z-Achse schaut, während die Y-Achse nach oben zeigt. Er ruft also nur eine Standardansicht auf, die sich für 2D-Spiele als praktisch erwiesen hat. Ansonsten nimmt dieser Button keinen Einfluss auf das Spiel.

2.2.3 Game View

Die *Game View* dient dem Testen Ihres Projektes. Sie zeigt das Spiel aus Sicht der Kamera und ermöglicht Ihnen, das Spiel so zu betrachten, wie es später nach der Erstellung aussehen wird. Wenn Sie das Spiel über die Play-Taste starten, können Sie über dieses Fenster Ihr Spiel testen. Das Spiel befindet sich dann im sogenannte *Play Mode*.

Bild 2.7 Game View

Wie die *Scene View* besitzt auch dieses Fenster am oberen Rand eine schmale *Control Bar* mit verschiedenen Funktionen.

- **Size-Menü** gibt Ihnen die Möglichkeit, zum Testen das Bildschirmformat vorzugeben. Es können Seitenverhältnisse sowie feste Pixelwerte vorgegeben werden, wie das Bild in der *Game View* dargestellt werden soll.
- **Maximize on Play-Button** wechselt bei Aktivierung beim Testen automatisch in einen Fullscreen-Modus.
- **Mute Audio-Button** deaktiviert/aktiviert den gesamten Sound, während sich das Spiel im *Play Mode* befindet.
- **Stats-Button** blendet eine kleine Statistik in der *Game View* ein, die verschiedene Performance-Daten darstellt.
- **Gizmos -Menü** schaltet typbezogen die Hilfsgrafiken in der *Game View* dazu.

2.2.4 Toolbar

Direkt unter dem eigentlichen Hauptmenü finden Sie die Toolbar mit ihren verschiedenen Werkzeugen.

Bild 2.8 Toolbar

Transform-Tools

Ganz links liegen die **Transform-Tools**, mit denen Sie das aktuell selektierte Objekt in der *Scene View* verändern können. Je nach Tool wechselt dabei das Handle des selektierten *GameObjects*, mit dem Sie dann das Objekt dementsprechend verändern können.

- **Hand-Tool** (Hand) dient keiner direkten Transform-Modifikation. Es erlaubt Ihnen das Verschieben der *Scene View*-Perspektive per Maus und gedrückter linker Maustaste.
- **Translate-Tool** (Kreuz) verschiebt das selektierte Objekt in der Szene.
- **Rotate-Tool** (drehende Pfeile) dreht das Objekt.
- **Scale-Tool** skaliert das Objekt bzw. Mesh in die verschiedenen Achsen. Selektieren Sie hierfür eins der drei Enden und ziehen Sie dieses in die jeweilige Richtung. Über den grauen Cube in der Mitte des Objektes skalieren Sie das Objekt in alle Richtungen gleichzeitig.
- **Rect-Tool** ermöglicht, das Objekt an den Rechteck-Handles zu verändern. Dieses Tool eignet sich vor allem zum Modifizieren von Texturen des Typs „Sprite“ und anderen 2D-Objekten. Durch das Verschieben einer Ecke verändern Sie die Größe des Objektes. Hier können Sie durch das zusätzliche Halten der UMSCH-Taste auch alle Seiten gleichzeitig skalieren. Bewegen Sie Ihren Mauszeiger (ohne gedrückte Maustaste) bei einem Handle etwas nach außen, erscheint ein Rotations-Symbol. Wenn Sie nun die linke Maustaste drücken, können Sie das Objekt um den *Pivot*-Punkt drehen.

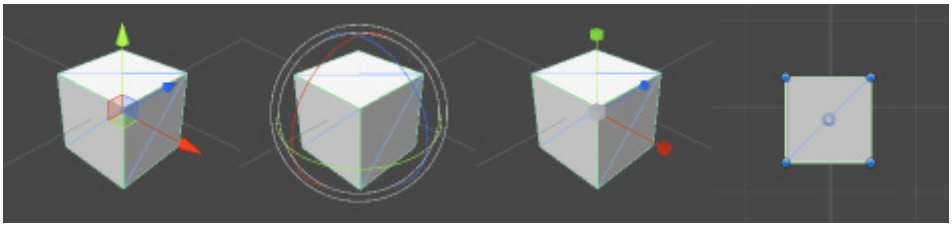

Bild 2.9 Handles der unterschiedlichen Transform-Tools (Translate-, Rotate-, Scale- und Rect-Tool)

Snapping

Häufig ist es nützlich, wenn man Objekte in fest definierten Schritten verschieben, drehen oder skalieren kann. Dieses Verfahren, das *Snapping* genannt wird, können Sie anwenden, indem Sie beim Nutzen des jeweiligen Transform-Tools die STRG-Taste gedrückt halten.

Die Raster, die hierfür gelten, können Sie in den *Snap Settings* einstellen, die Sie über EDIT/SNAP SETTINGS erreichen. Dort können Sie für das Verschieben, Drehen und Skalieren die Raster separat einstellen.

Vertex-Snapping

Ein besonderes Snapping-Feature ist das **Vertex-Snapping**. Es dient dazu, ein Objekt nicht in einem festdefinierten Raster zu verschieben, sondern Objekte so zu positionieren, dass ein Vertex des eigenen Objektes genau auf der Vertex-Position eines anderen Objektes liegt. Auf diese Weise können Sie z. B. ohne großen Aufwand unterschiedliche 3D-Modelle genau nebeneinander platzieren.

Selektieren Sie hierfür ein Objekt in der Szene und drücken Sie anschließend die Taste V. Nun können Sie durch Bewegen der Maus einen Vertex des Objektes auswählen (halten Sie hierbei keine Maustaste gedrückt!). Wenn Sie einen passenden gefunden haben, drücken Sie die linke Maustaste und bewegen die Maus zu einem Vertex eines anderen Objektes, wo das Objekt schließlich positioniert werden soll. Das erste Objekt wird nun Vertex an Vertex platziert.

Zu den *Transform-Tools* gehören auch die beiden daneben liegenden Funktionsknöpfe, die sogenannten **Gizmo DisplayToggles**.

Mit diesen Toggle-Buttons können Sie folgende Funktionen ausführen:

- **Pivot/Center** definiert das Transformationszentrum. *Pivot* ist hierbei der Original-Nullpunkt des Modells, *Center* nimmt die Mitte des gerenderten Objektes.
- **Local/Global** legt fest, ob sich die Transformationsachsen auf das lokale oder das globale Koordinatensystem beziehen sollen.

Transform-Tools

Eine Demonstration aller*Transform-Tools* und des „Vertex-Snappings“ finden Sie in Video-Form im Online-Bereich des Buches. In diesem Zusammenhang werden auch *Sprites* näher erläutert.

Play Controls

Der nächste Abschnitt sind die **Play Controls**. Mit diesen können Sie Ihr Spiel in der *Game View* testen.

Ein erstes Drücken auf die Play-Taste startet das Spiel (Play-Modus), ein zweiter Klick beendet es. Der zweite Button der *Play Controls* pausiert das Spiel und der dritte spielt es in kleinen Schritten ab. Letzteres eignet sich beispielweise zum Testen von Kollisionen, Animationen oder Partikeleffekten.

Änderungen im Play-Modus

Alle Änderungen, die Sie im Play-Modus in der Szene machen, werden nach dem Stoppen wieder rückgängig gemacht.

Beachten Sie, dass Änderungen im *Project Browser* nicht rückgängig gemacht werden. Ändern Sie also im Play-Modus ein *Asset*, z. B. ein *Material*, ein Skript oder einen *Audio Mixer*, werden diese Änderungen nicht rückgängig gemacht.

Als Nächstes folgt das **Layers Drop-down-Menü**. Über dieses können Sie die *GameObjects* dieser Layer in der *Scene View* sichtbar und unsichtbar schalten.

Ganz am Ende finden Sie schließlich das *Layouts-Drop-down-Menü*, über das Sie den Aufbau der Subfenster von Unity speichern, laden und verwalten können.

2.2.5 Hierarchy

Die *Hierarchy* zeigt alle *GameObjects* der aktuellen Szene an. Unity unterscheidet bei der Darstellung zwischen normalen *GameObjects* und *Prefab*-Instanzen, die blau dargestellt werden. *Prefab*-Instanzen sind Kopien einer Vorlage (*Prefab* genannt) und besitzen zusätzliche Funktionalitäten (siehe Kapitel 15 „Prefabs“).

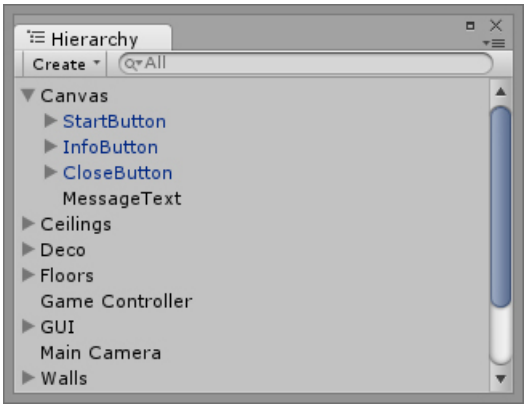

Bild 2.10 Hierarchy-Tab

Das *Hierarchy*-Fenster und die *Scene View* werden ständig synchronisiert, sodass ein selektiertes Objekt in der *Scene View* auch gleichzeitig in der *Hierarchy* markiert ist. Sie können auch ein Asset (z. B. ein 3D-Modell oder ein *Prefab*) aus dem *Project Browser* direkt in die *Hierarchy* ziehen, um es in einer Szene zu platzieren. Da bei diesem Verfahren keine konkrete Positionsvorgabe existiert, wird das Objekt zunächst auf die Position (0, 0, 0) gesetzt. Möchten Sie nun den Fokus auf das neue Objekt setzen, drücken Sie einfach die Taste F. Unity setzt hierbei den Fokus auf das aktuell in der *Hierarchy* selektierte Objekte und zeigt dieses an (siehe Abschnitt 2.2.2.1 „Navigieren in der Scene View“).

Das *Hierarchy*-Fenster besitzt oben links ein Drop-down-Menü, das einen Schnellzugriff auf das GAMEOBJECT-Menü erlaubt und Ihnen die Möglichkeit bietet, schnell neue *GameObjects* Ihrem Projekt zuzufügen. Rechts neben dem Menü finden Sie eine Suchmaske, um nach *GameObjects* zu filtern.

Außerdem können Sie über das Kontextmenü der rechten Maustaste *GameObjects* löschen, umbenennen, kopieren und noch einiges mehr. An dieser Stelle möchten wir noch kurz darauf hinweisen, dass das Duplizieren eines Objektes oder eines *Assets* in Unity generell über das Tastenkürzel STRG+D funktioniert, nicht Windows-typisch über STRG+C und STRG+V.

2.2.5.1 Parenting

Ein wichtiges Konzept von Unity ist das sogenannte *Parenting* bzw. Eltern-Kind-Konzept. Bei diesem kann ein beliebiges *GameObject* unter einem anderen Objekt hängen und erbt von diesem die relativen Transform-Eigenschaften wie Position und die Rotation. Besitzt ein *GameObject* ein oder mehrere Unterobjekte, wird dies durch ein kleines Dreieck am linken Rand dargestellt. Wenn Sie auf dieses klicken, werden die Unterobjekte wie in einer Ordnerstruktur auf- und zugeklappt.

Bild 2.11 Struktur eines Soldaten mit mehreren Kind-Objekten

Die im *Inspector* angezeigten Werte der *Transform*-Komponente sind bei einem Kind-Objekt immer lokale Werte, sie stellen die Position, Rotation, Skalierung also immer relativ zum Eltern-Objekt dar. Beträgt die Position eines Kind-Objektes nun (0, 0, 0), so befindet es sich im Zentrum des Eltern-Objektes. Verschieben Sie dieses Eltern-Objekt, wird das Kind-Objekt zwar mitverschoben, die Position bleibt aber im *Inspector* bei (0, 0, 0).

2.2.6 Inspector

Im *Inspector* werden alle Informationen und Komponenten des aktuell selektierten *GameObjects* angezeigt. Hier können Sie Komponenten hinzufügen, entfernen und Einstellungen an diesen vornehmen.

Bild 2.12 Inspector-Beispiel

Komponenten zuweisen

Sie können*Components* sowohl über den *Add Component*-Button des *Inspectors* einem *GameObject* zuweisen als auch via Drag & Drop aus dem *Project Browser*. Genauso können Sie aber auch diese vom *Project Browser* auf den *Hierarchy*-Eintrag oder direkt auf das *GameObject* in der *Scene View* ziehen.

2.2.6.1 Kopfinformationen im Inspector

Im Kopf des *Inspectors* finden Sie neben dem Namen des *GameObjects* viele weitere Informationen und Einstellmöglichkeiten.

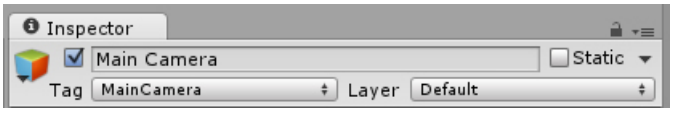

Bild 2.13 Kopfinformationen im Inspector

Ganz links können Sie ein **Icon** auswählen, das in der *Scene View* an der Position des *GameObjects* dargestellt werden soll. Klicken Sie hierfür auf das angezeigte Symbol und wählen Sie ein Icon aus. Das Zeichen wechselt dann auf das ausgewählte **Icon**. Ein Würfel-Symbol, wie in Bild 2.13 zu sehen, bedeutet, dass kein spezielles **Icon** angezeigt werden soll.

Daneben finden Sie eine Checkbox, über die Sie das gesamte Objekt mit all seinen Komponenten aktivieren und deaktivieren können. Deaktivierte Objekte werden in der *Hierarchy* grau dargestellt und in der *Scene View* nicht mehr angezeigt.

Rechts neben dem Namen finden Sie die **Static**-Checkbox. Über diese können Sie definieren, ob ein Objekt statisch ist, also ob es sich in seiner örtlichen Lage verändert bzw. verändern kann. Diese Einstellung ist lediglich für einige Spezialfunktionen wichtig, die sich auf nicht bewegende Objekte beziehen. Für welche Funktionen dieser Kenner genau gelten

soll, können Sie zudem über das rechte Drop-down-Menü zusätzlich einschränken (dargestellt durch den kleinen, nach unten zeigenden Pfeil).

In der zweiten Zeile können Sie dem *GameObject* zudem noch einen **Tag** und einen **Layer** zuweisen, auf die wir später noch zu sprechen kommen.

2.2.6.2 Variablen Werte zuweisen

Fast jede Komponente besitzt im *Inspector* eigene Parameter, mit denen Sie deren Funktionen parametrisieren können. Dabei können Sie aber nicht nur Werte in Form von Zahlen und Texten zuweisen, Sie können diesen teilweise auch Elemente aus dem *Project Browser* oder aus der aktuellen Szene zuweisen – je nach Typ und Aufgabe der Variablen. So besitzt z. B. jede *AudioSource* einen Parameter namens `AudioClip`, dem die abzuspielende Audiodatei, die im Project Browser zu finden ist, zugewiesen wird. Nehmen Sie einfach eine beliebige Audiodatei aus dem *Project Browser* und ziehen Sie diese auf die `AudioClip`-Variable.

Anstatt solche Objekte per Drag & Drop einer Variablen zuzuweisen, können Sie dies auch über ein Auswahlmenü machen, das Sie über ein kleines Kreis-Symbol neben dem jeweiligen Parameter erreichen (siehe Bild 2.14). Dieses öffnet einen Dialog, aus dem Sie dann auch das passende Objekt auswählen können. Möchten Sie hierbei der Variablen keinen Inhalt zuweisen, wählen Sie einfach den Eintrag *None* aus, der jedes Mal mit angeboten wird.

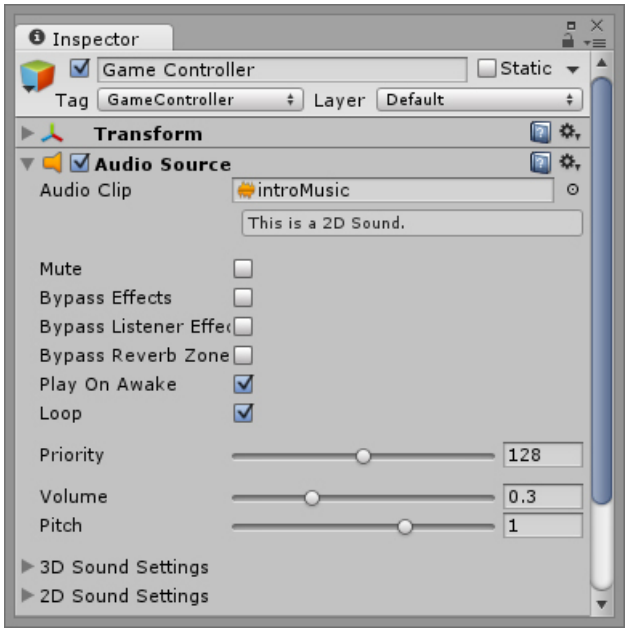

Bild 2.14 Darstellung öffentlicher Variablen im Inspector

2.2.6.3 Komponenten-Menüs

Jede Komponente (z. B. AudioSource, Transform) eines *GameObjects* besitzt ein Menü mit einigen Grundsatzfunktionen zum Anordnen, Zurücksetzen und Löschen der jeweiligen Komponente. Mit dem Befehl RESET setzen Sie z. B. alle Werte auf Default-Einstellungen zurück.

Dieses Menü erreichen Sie über das Zahnrad-Symbol im oberen rechten Bereich einer Komponente. Parallel dazu stehen die gleichen Funktionen auch über das Kontextmenü der rechten Maustaste zur Verfügung.

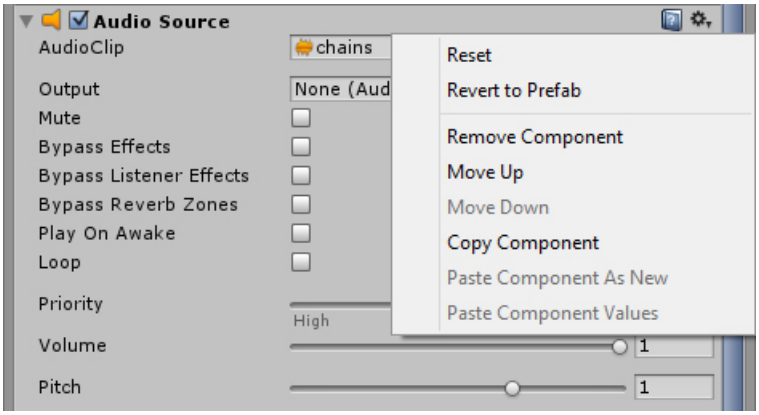

Bild 2.15 Komponenten-Menü im Inspector

2.2.6.4 Preview-Fenster

Wenn Sie im Project Browser ein Asset selektieren, erscheint im *Inspector* noch ein zusätzliches *Preview-Fenster*, in dem eine Vorschau des markierten Assets angezeigt wird. Je nach Asset-Typ bietet das Fenster noch zusätzliche Funktionen an. So wird z. B. bei einer Animation ein Start-Button angeboten, um diese zu starten, und ein Geschwindigkeitsregler, der die Schnelligkeit des Abspielens steuert.

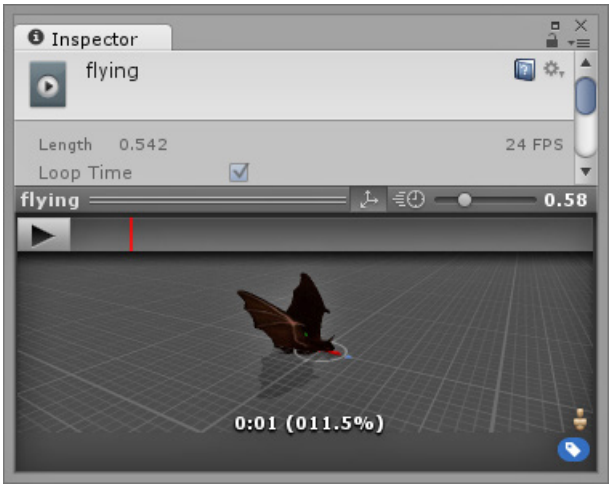

Bild 2.16 Preview-Fenster des Inspectors

Über das kleine blaue Schild-Symbol können Sie zudem jedem *Asset* ein oder mehrere Label zufügen, über die Sie diese später filtern können.

2.2.7 Project Browser

Im *Project Browser* befinden sich alle *Assets* des Projektes. Die linke Seite dieses Fensters zeigt dabei die Ordnerstruktur Ihrer Projektdateien an, die rechte Seite stellt den Inhalt des aktuell selektierten Ordners dar. Den Inhalt können Sie dabei über die Suchmaske im oberen Bereich durchsuchen und mithilfe des Type- und des Label-Filters aussortieren (siehe Abschnitt 2.2.7.1 „Assets suchen und finden“).

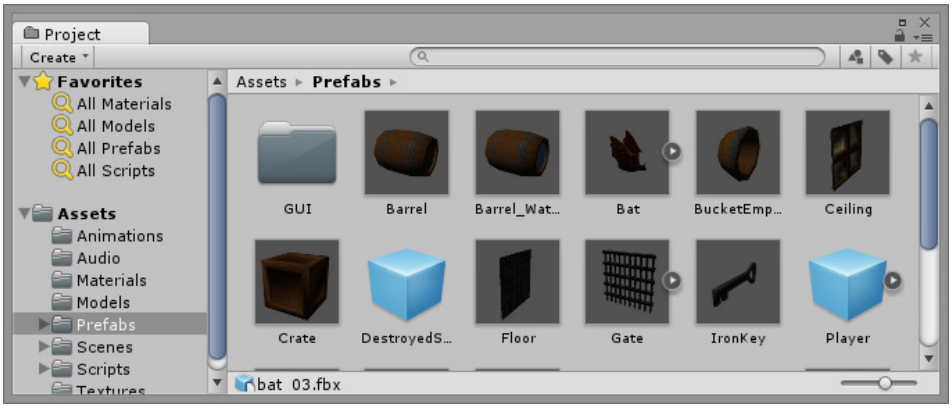

Bild 2.17 Project Browser

Mit dem Schieberegler unten rechts können Sie die Darstellung der Assets verändern. Dabei können Sie die Inhalte in einer Listendarstellung oder als Symbole anzeigen, deren Größe Sie mit dem Regler stufenlos festlegen können.

Über einen Rechtsklick in den *Project Browser* erhalten Sie eine große Auswahl an Funktionen. Der Bereich *Create* ist dabei ein Schnellzugriff auf das Hauptmenü ASSETS/CREATE, der auch über das Drop-down-Menu oben links im *Project Browser* angeboten wird. Beide Menüs bieten Ihnen somit die Möglichkeit, neue *Assets* wie Skripte, Materialien, *Shader* etc. zu erstellen. Aber auch Unterordner können Sie hier erstellen, die Sie für das Gliedern der *Assets* nutzen sollten.

2.2.7.1 Assets suchen und finden

Da sich im *Project Browser* alle Dateien befinden, die Sie in Ihrem Projekt nutzen können, wird die Menge der Dateien schnell ansteigen. Deshalb ist es sehr wichtig, von vornherein eine gute Struktur aufzubauen, nach der die Assets im *Project Browser* abgelegt werden.

Ordnung halten

Es ist empfehlenswert, im *Project Browser* für die unterschiedlichen Asset-Typen (Scripts, Materials, Textures ...) eigene Ordner zu erstellen und diese dort dann abzulegen. Bei größeren Projekten können Sie innerhalb dieser Ordner noch weitere Unterordner anlegen, um diese dann nach Aufgaben, Level o. Ä. zu sortieren.

Möchten Sie Ihre *Assets* nun noch zusätzlich gruppieren, z. B. nach Zusammengehörigkeit oder einer bestimmten Eigenschaft, können Sie dies mit sogenannten Labels machen, die Sie im *Preview-Fenster* den Assets zufügen können.

Über den *Project Browser* können Sie nun über das Label-Symbol, das Sie neben dem Suchfeld finden, danach filtern. Der Vorteil hierbei ist, dass sich die Filterung nun über alle Ordner erstreckt, deren Treffer dann auch alle im *Project Browser* angezeigt werden.

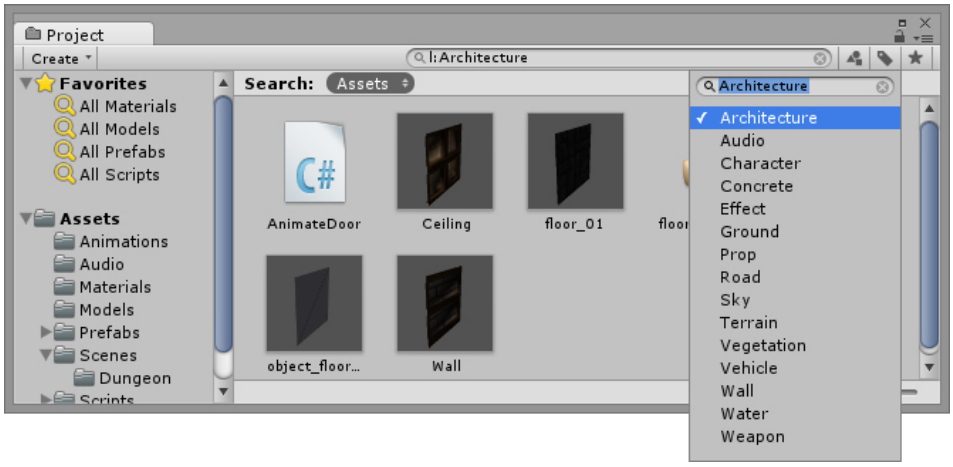

Bild 2.18 Filterung nach dem Label „Architecture“

Den *Asset-Type* können Sie ebenfalls in die Filterung mit einfließen lassen. Nutzen Sie hierfür das links neben dem Label-Filter liegende Symbol.

Um schließlich häufig genutzte Filter wiederzuverwenden, können Sie sie unter *Favorites* abspeichern. Nutzen Sie hierfür das Stern-Symbol, das Sie rechts neben dem Label-Symbol finden. Wollen Sie einen abgespeicherten Favoriten wieder löschen, können Sie dies über das Kontextmenü des jeweiligen Favoriten machen.

2.2.7.2 Assets importieren

Wenn Sie externe *Assets* (z. B. Texturen, Musikdateien, 3D-Modelle etc.) in Ihr Programm importieren wollen, können Sie diese einfach aus einem beliebigen Ordner Ihres Computers in den *Project Browser* hineinziehen und fallen lassen. Unity erstellt eine Kopie der Datei und importiert diese in das Projekt.

2.2.8 Console

Der *Console*-Tab, zu Deutsch „Konsole“, dient dem Anzeigen von Fehler- und Warnmeldungen. Sollten Sie beispielsweise Programmierfehler gemacht haben, zeigt Unity diese hier an. Aber auch Sie selber als Entwickler können hierüber Meldungen ausgeben.

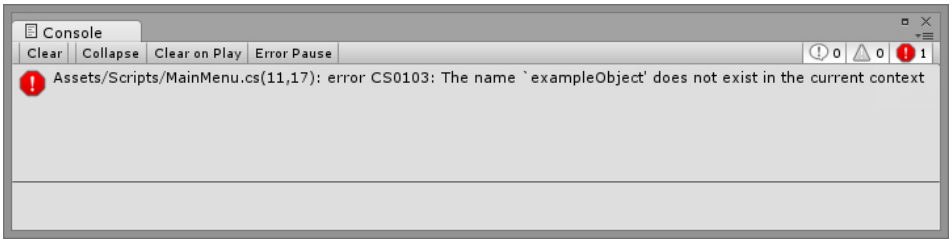

Bild 2.19 Console-Tab mit einer Fehlermeldung

Meldungen zu Fehlern, die zwingend behoben werden müssen (Errors), werden in dem Fenster rot dargestellt. Warnungen, also nicht kritische Fehler, erscheinen demgegenüber gelb. Normale Hinweismeldungen werden weiß angezeigt. Über die drei Symbole an der rechten Seite können Sie die Meldungen auch nach diesen Einstufungen filtern.

Neben den drei Filtermöglichkeiten besitzt die obere Konsolenleiste noch einige Buttons. Diese haben folgende Bedeutungen:

- **Clear** löscht alle Einträge.
- **Collapse** unterdrückt doppelt erscheinende Meldungen.
- **Clear on Play** löscht alle Einträge beim Start des Spiels über den Play-Button.
- **Error Pause** unterbricht das Spiel, sobald ein Eintrag mit `Debug.LogError("Fehlertext")` ausgeführt wird.

Mehr dazu im Kapitel 4 „Skript-Programmierung“.

2.3 Das Unity-Projekt

Das Gesamtgerüst eines Spiels bzw. einer Anwendung wird in Unity als Projekt bezeichnet. Es speichert alle allgemeinen Einstellungen und Daten wie beispielsweise den Namen des Spiels, die Tastenbelegungen oder auch Grafikeinstellungen. Das Spiel an sich findet aber nicht in einem Projekt statt, sondern in Szenen (Scenes), die ebenfalls in dem Projekt gespeichert werden. Jedes spielbare Projekt besitzt also mindestens eine Szene. Zum Anlegen und Öffnen von Projekten nutzt Unity das Projekt-Fenster, das sich übrigens auch öffnet, wenn Sie zum ersten Mal Unity starten (siehe „Einleitung“).

2.3.1 Neues Projekt anlegen

Über das Menü FILE/NEW PROJECT öffnen Sie das Projekt-Fenster, um ein neues Projekt zu erstellen. In der Textbox *Project name* wird Ihnen gleich ein neuer Name vorgeschlagen, den Sie natürlich auch ändern können. Oder Sie wählen gleich in der Textbox *Location* einen komplett neuen Pfad aus. Navigieren Sie zu einem Ordner, in dem Sie Ihr Projekt erstellen möchten, und bestätigen Sie die Auswahl.

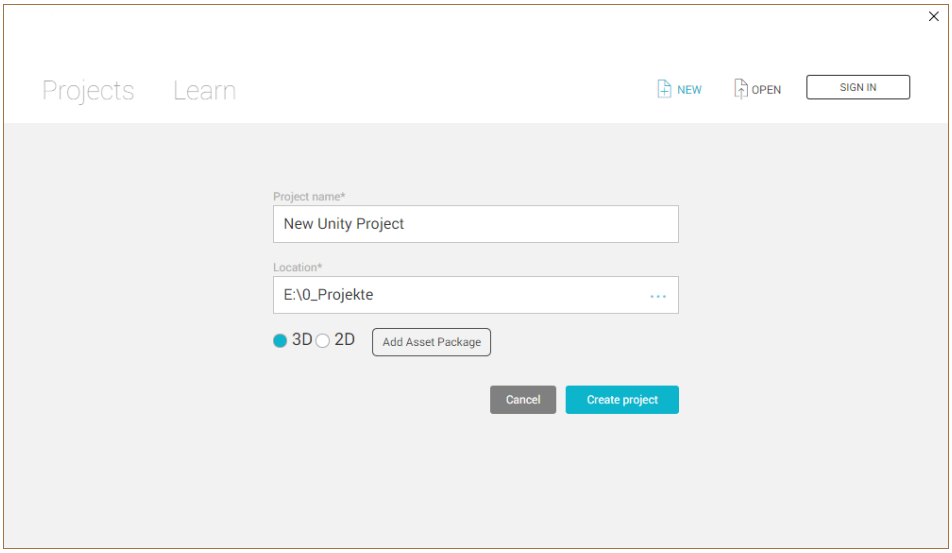

Bild 2.20 Projekt-Fenster: Neu-Modus

Über CREATE PROJECT erstellen Sie dann das neue Projekt.

Wenn Sie möchten, können Sie noch vor dem Erstellen über den Knopf ASSET PACKAGES verschiedene *Unity Packages* auswählen, die Sie von vornherein in Ihr Projekt importieren wollen. Sie können aber auch alle Packages im Nachhinein importieren, weshalb diese Auswahl nicht so essenziell ist.

Als Letztes können Sie noch entscheiden, ob Sie die Standardeinstellungen für ein 3D- oder ein 2D-Spiel nutzen wollen. Diese Auswahl hat keinen Einfluss auf die tatsächliche Spieleentwicklung, sondern lediglich auf einige Standardeinstellungen. Aber auch diesen Parameter können Sie später noch über EDIT/PROJECT SETTINGS/EDITOR mit dem Parameter *Default Behaviour Mode* ändern. Die Hauptunterschiede dieser beiden Möglichkeiten sind:

- **2D** setzt die Kamera wie auch die *Scene View*-Darstellung auf *Orthographic* (siehe Abschnitt 2.2.2.2 und Kapitel „Kameras, die Augen des Spielers“) und den *Texture Type* beim Texturen-Import auf den Default-Wert *Sprite*.
- **3D** setzt die Kamera standardmäßig auf *Perspective* und den *Texture Type* beim Texturen-Import auf den Default-Wert *Texture*. Außerdem wird in diesem Modus jeder Default-Szene gleich ein Licht-Objekt sowie eine Skybox zum Simulieren des Himmels zugefügt.

Haben Sie alle Einstellungen getätigt, erzeugen Sie mit dem Button CREATE Ihr neues Projekt.

Nach dem Erstellen des Projektes öffnet Unity dieses. Sollten Sie vorher Packages ausgewählt haben, werden diese natürlich im *Project Browser* angezeigt.

2.3.2 Bestehendes Projekt öffnen

Möchten Sie bereits erstellte Projekte öffnen, starten Sie über das Menü FILE/OPEN PROJECT das Projekt-Fenster im Öffnen-Modus. Es wird Ihnen eine Auflistung der zuletzt geöffneten Projekte angezeigt. Über OPEN OTHER . . . können Sie natürlich auch ältere Projekte auswählen. Navigieren Sie hier zum Hauptordner des jeweiligen Projektes und bestätigen Sie die Auswahl.

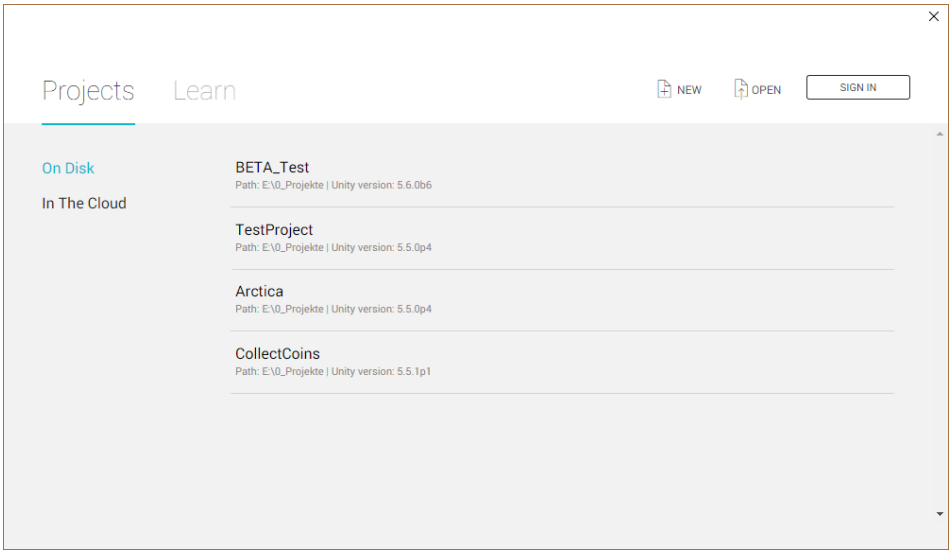

Bild 2.21 Projekt-Fenster: Öffnen-Modus

2.3.2.1 Ältere Projekte öffnen

Sollte das Projekt mit einer älteren Unity-Version erstellt worden sein, möchte Unity das Projekt „upgraden". Dabei werden Dateien auf das Format der neuen Version konvertiert. Es ist ratsam, jedes Projekt einmal zu sichern, bevor Sie diesen Upgrading-Prozess starten. Sollten Sie dies noch nicht getan haben, brechen Sie mit QUIT den Vorgang ab und machen Sie eine Sicherungskopie des Projektes.

Danach können Sie das Projekt noch einmal starten und dann den Upgrading-Prozess mit CONTINUE durchführen lassen. Dieser Vorgang kann je nach Projektgröße auch schon mal etwas länger dauern. In manchen Fällen erscheint nach diesem Prozess noch eine Extraabfrage zum Konvertieren einzelner Dateien. Wenn nichts Besonderes dagegen spricht, sollten Sie dies ebenfalls bestätigen.

2.3.3 Projektdateien

Jedes Unity-Projekt besteht aus mehreren Hauptordnern, die wir im Folgenden kurz vorstellen möchten:

- **Assets** speichert alle Ressourcen des Projektes. Alle Dateien und Ordner, die sich in diesem befinden, werden in dem *Project Browser* von Unity angezeigt.
- **ProjectSettings** speichert alle Projekteinstellungen wie *Tags*, *Player Settings* usw.
- **Library** dient als Speicher für Metadaten von importierten *Assets*.
- **Temp** dient zum Ablegen von temporären Dateien, die von Unity zum Beispiel beim Build-Vorgang erstellt werden.

2.3.4 Szene

Eine Szene können Sie sich wie einen Level oder wie eine Welt eines Spiels vorstellen, in der der Aufbau des Spiels mit seinen 3D-Modellen, Grafiken, Musikdateien etc. (allgemein auch Assets genannt) festgelegt wird.

Szene und Level

Häufig werden in Unity die Begriffe Scene und Level gleichgesetzt. Trotzdem können Spieleentwickler durch geschickte Programmierung bereits aus einer einzelnen Szene Spiele mit mehreren 100 Levels programmieren.

Eine neue Szene erzeugen Sie über das Menü FILE/NEW SCENE. Eine bestehende Szene öffnen Sie über FILE/OPEN SCENE. Jede Szene, die später im Spiel auch eingebunden werden soll, muss zuvor über FILE/BUILD SETTINGS dem Spiel zugefügt werden, wo es dann auch einen eindeutigen Index erhält, über den man die Szene auch dann identifizieren und starten kann.

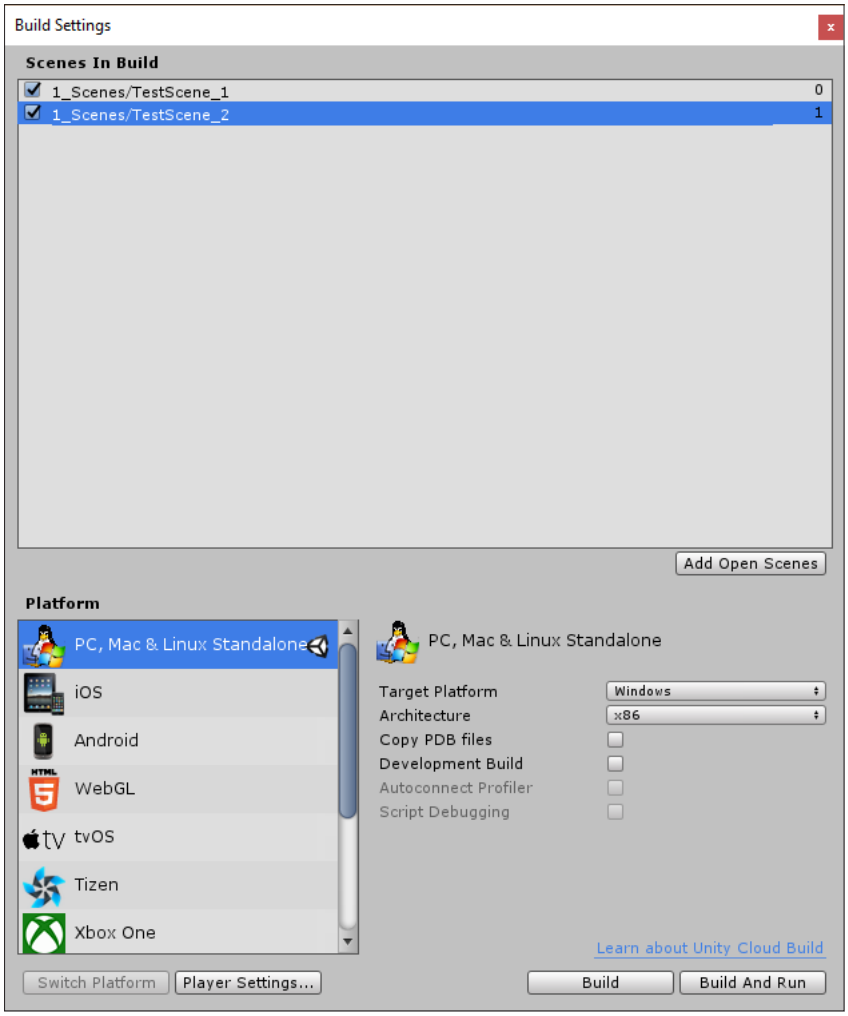

Bild 2.22 Build Settings mit den Scene Indices

2.3.5 Game Objects

Der wichtigste Baustein in einer Unity-Szene ist das *GameObject*. Jedes Objekt, das sich in einer Szene befindet, ist ein solches.

Trotzdem ist ein *GameObject* an sich zunächst einmal nichts anderes als ein leerer Container, der im Grunde nichts kann. Seine eigentlichen Eigenschaften und Fähigkeiten erhält ein *GameObject* erst durch sogenannte Komponenten bzw. *Components*, die man einem *GameObject* zufügen kann.

Dies ist ein ganz elementares Prinzip von Unity, das stets bedacht werden muss. Anstatt Eigenschaften eines *GameObjects* zu verändern, werden Sie normalerweise immer Parameter einer Komponente ändern, die natürlich zu dem jeweiligen *GameObject* gehört.

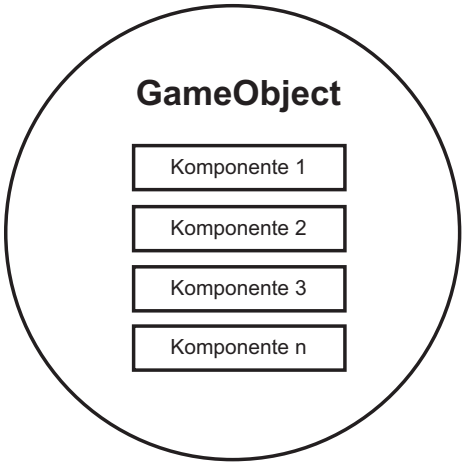

Bild 2.23 Schematische Darstellung eines GameObjects mit Komponenten

Das Spannende an diesem Verfahren ist nun, dass jedes *GameObject* alles sein kann. Um z. B. aus einer Kugel eine Lampe zu machen, brauchen Sie dieser lediglich eine Beleuchtungs-Komponente zu geben und die Komponenten entsprechend zu parametrisieren. Das war es schon.

Möchten Sie jetzt ein einfaches *GameObject* Ihrer Szene zufügen, können Sie dies über das Hauptmenü GAMEOBJECT/CREATE EMPTY machen. Diese *GameObjects* werden auch „Empty GameObjects“ genannt, obwohl sie gar nicht wirklich „empty“ sind. Denn auch diese besitzen bereits eine Komponente, und zwar die *Transform*-Komponente. Diese ermöglicht Ihnen, das *GameObject* in der Szene zu platzieren – was ja auch durchaus Sinn macht, wenn man im Editor arbeitet. Ein komplett leeres *GameObject* können Sie nur via Code erzeugen, worauf wir aber noch im Kapitel „Skript-Programmierung“ zu sprechen kommen.
Components

Components (Komponenten) werden *GameObjects* zugefügt und verleihen dem *GameObject* seine eigentlichen Fähigkeiten und bestimmen deren Verhalten.

Unity liefert bereits eine ganze Reihe fertiger *Components* mit, die beispielsweise der Kollisionserkennung dienen, Sound abspielen können oder Partikeleffekte erzeugen. Auch selbst programmierte Skripte gelten in Unity als Komponenten. Auf die genauen Zusammenhänge werden wir aber im bereits erwähnten Kapitel 4 „Skript-Programmierung“ noch genauer eingehen.

Wenn Sie ein *GameObject* in der *Hierarchy* oder direkt in der Szene selektieren, werden Ihnen im *Inspector* alle Komponenten dieses *GameObjects* angezeigt. Ein schematisch dargestelltes *GameObject* wie in Bild 2.23 könnte dann in der Praxis wie in Bild 2.24 aussehen. Das dort selektierte *GameObject* besitzt eine *Transform*-Komponente (dargestellt durch den dreiachsigen Handle) und eine *Light*-Komponente (dargestellt durch das Glühlampen-*Gizmo* und die gelbe Kugel, die die Beleuchtungsreichweite zeigt).

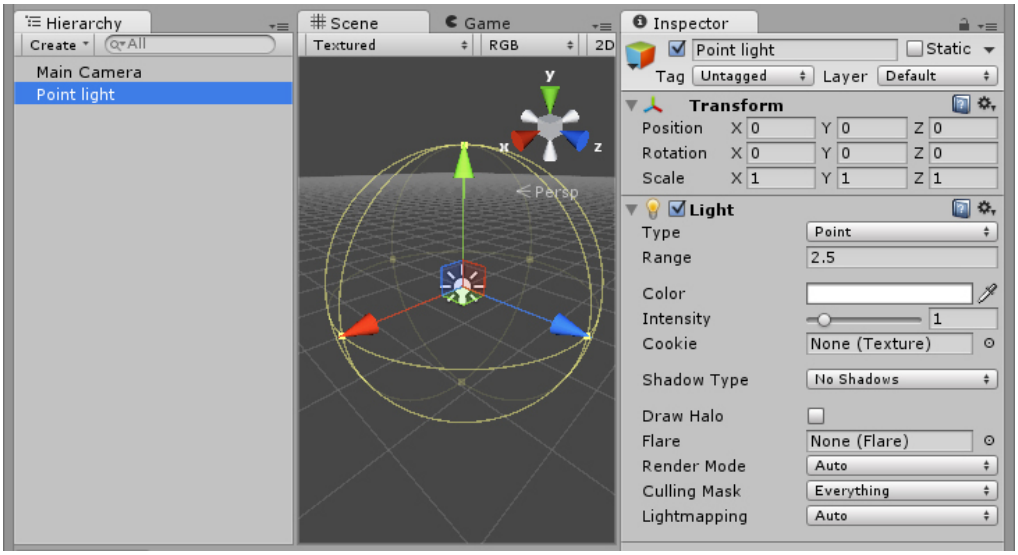

Bild 2.24 GameObject mit Komponenten

Ein *Component* ist also ein ganz wichtiger Bestandteil eines *GameObjects*, weshalb ein *Component* aber auch nicht alleine in einer Szene existieren kann, sondern immer einem *GameObject* zugefügt sein muss.

2.3.6 Tags

Das englische Wort „Tag“ bedeutet nichts anderes als Etikett, und genau das ist ein *Tag* auch. Es ist ein Typenschild für *GameObjects*, um diese beim Programmieren besser identifizieren zu können.

Sie weisen den Tag im *Inspector* zu, wo Sie im oberen Bereich ein Drop-down-Menü mit allen *Tags* finden. Jedes *GameObject* kann nur mit einem einzigen Tag gekennzeichnet werden. Standardmäßig stellt Unity bereits einige wichtige Tags wie z. B. „Player“, „MainCamera“ oder „GameController“ bereit. Über den Menüpunkt ADD TAG können Sie aber auch weitere *Tags* definieren und anschließend zuweisen.

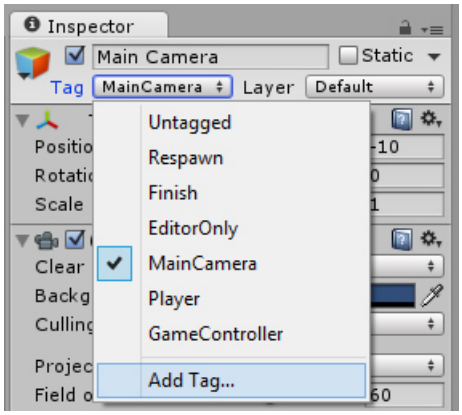

Bild 2.25 Tag zuweisen

2.3.7 Layer

Eine weitere Kategorisierung neben *Tags* sind *Layer*. Über diese werden meist funktional zusammengehörige Objekte markiert, wie z. B. UI-Objekte oder Objekte, die von bestimmten Funktionalitäten betroffen sind.

So gibt es zum Beispiel bei Kameras die Möglichkeit festzulegen, welche *Layer* von der Kamera gerendert werden sollen. Alle anderen Objekte werden ignoriert und nicht von dieser dargestellt.

Layer weisen Sie ebenfalls im *Inspector* zu. Auch dort können Sie vordefinierte wählen und auch eigene erstellen.

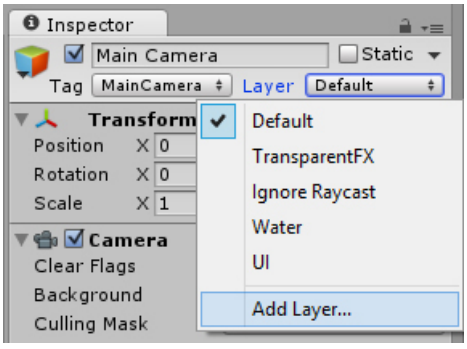

Bild 2.26 Layer zuweisen

2.3.8 Assets

Als *Asset* werden alle digitalen Inhalte bezeichnet, die sich im Unity-Projekt und damit im Ordner „Assets“ des jeweiligen Projektes befinden. Das können beispielsweise Audiodateien, Texturen, Materialien, Skripte oder auch 3D-Modelle sein.

Den Ordner „Assets“ müssen Sie nicht lange suchen. Er wird mit allen seinen Unterordnern und enthaltenen Dateien im *Project Browser* angezeigt. Jeder *Asset*-Typ besitzt eigene Import-Settings, anhand derer Sie bestimmen, wie diese in Unity importiert werden sollen.

Auch wenn wir in diesem Buch einige *Import Settings* vorstellen werden, so würde das Erläutern aller verfügbaren Parameter den Rahmen sprengen. Deshalb möchten wir an dieser Stelle auf die Hilfe-Funktionen hinweisen, die Ihnen bei allen *Import Settings* in Form eines Fragezeichens zur Verfügung gestellt werden.

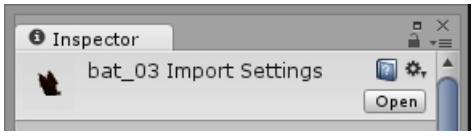

Bild 2.27 Hilfe-Funktion in den ImportSettings

2.3.8.1 UnityPackage

UnityPackage ist ein Container-Format, welches für den Austausch von *Assets* zwischen unterschiedlichen Unity-Projekten gedacht ist. Dabei bleiben die Verknüpfungen und Strukturen der einzelnen *Assets* untereinander erhalten.

Auf diese Weise kann beispielsweise eine fertig konfigurierte Spielerfigur ohne Probleme von einem Projekt in ein anderes geladen werden, ohne dass Komponenten und Assets neu zugewiesen werden müssen. Ein *UnityPackage*besitzt die Dateiendung *.unitypackage*.

2.3.8.2 Asset Import

Um *Assets* im *UnityPackage*-Format zu importieren, gehen Sie im Hauptmenü auf ASSETS/IMPORT PACKAGE. Alternativ können Sie die gleichen Menüpunkte aber auch im *Project Browser* über die rechte Maustaste erreichen.

Dort können Sie über CUSTOM PACKAGE ein beliebiges *UnityPackage* von Ihrer Festplatte importieren. Zusätzlich stehen Ihnen dort aber auch noch von Unity mitgelieferte Asset-Pakete zur Verfügung, die sogenannten *Standard Assets*. Das sind thematisch zusammenhängende *Assets*, die Unity im *UnityPackage*-Format standardmäßig mitliefert und direkt im Menüzweig einbindet.

Egal ob Sie nun *Standard Assets* oder eine eigene *UnityPackage*-Datei auswählen, in beiden Fällen öffnet sich nach der Wahl des Paketes ein Auswahldialog, über den Sie noch einmal bestimmen können, welche Dateien genau aus dem Paket importiert werden sollen.

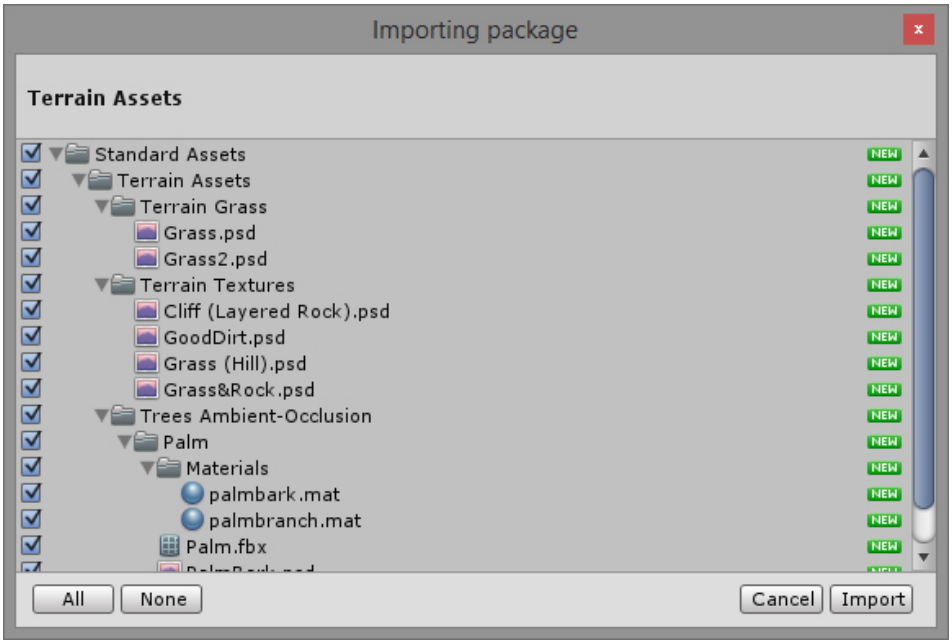

Bild 2.28 Asset Import-Dialog

2.3.8.3 Asset Export

Den Export als *UnityPackage* erreichen Sie sowohl über das Kontextmenü der rechten Maustaste im *Project Browser* als auch über das Hauptmenü im Bereich ASSETS.

Selektieren Sie hierfür beliebig viele *Assets* und Ordner im *Project Browser*, die Sie exportieren möchten, und wählen dann die Funktion EXPORT PACKAGE. Es öffnet sich ein Fenster, in dem noch einmal alle ausgewählten Dateien angezeigt werden, um hier ggf. noch einige wieder auszuschließen. Drücken Sie dann EXPORT, und es wird ein *UnityPackage* mit den jeweiligen Dateien erstellt.

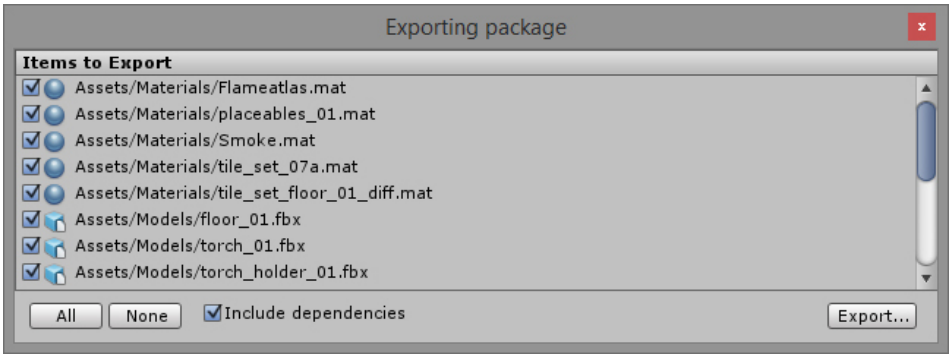

Bild 2.29 Asset Export-Dialog

2.3.8.4 Asset Store

Unity besitzt einen integrierten Zubehör-Shop, den *Asset Store*. In diesem finden Sie eine große Auswahl an kostenlosen und kostenpflichtigen *Assets* aller Art. Dabei reicht das Spektrum von kleinen 3D-Modellen und einfachen Skripten über komplexe Wettersimulationen bis hin zu *Editor Extensions*, die Unity um ganz neue Funktionen erweitern. Solche *Editor Extensions* können z. B. Visual Scripting Tools sein, mit denen Sie nicht mehr Code programmieren müssen, sondern diesen über das Anordnen von Grafiken und Symbolen automatisch erzeugen können.

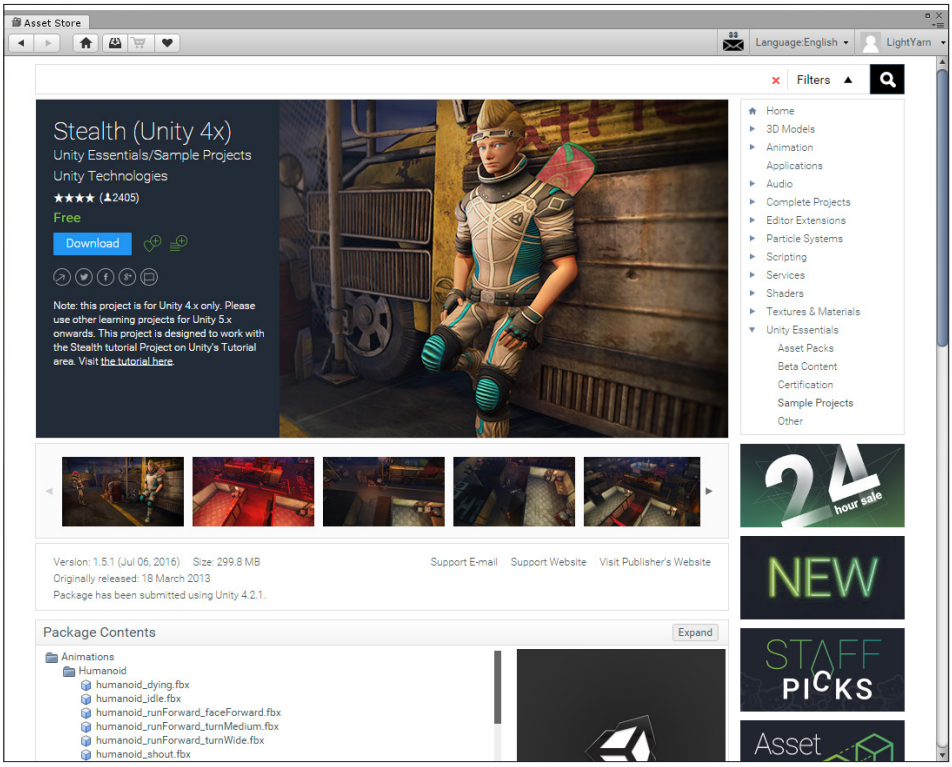

Bild 2.30 Asset Store

Aber nicht nur einzelne Assets finden Sie in dem Store, dort gibt es auch ganze Templates für Projekte und Anwendungsbeispiele für bestimmte Techniken. Auch Unity Technologies selbst stellt dort mittlerweile seine kostenlosen Beispiele und Tutorial-Projekte hinein, sodass Sie auch dort viele kostenlose Assets finden, die, wie die *Standard Assets* auch, vom Hersteller selbst stammen.

Sie öffnen den *Asset Store* direkt aus Unity heraus über das Menü WINDOW/ASSET STORE. In einem Extra-Tab öffnet Unity dann den Store.

2.3.9 Frames

Als *Frame* wird ein einzelnes auf dem Monitor dargestelltes Bild bezeichnet. Vergleichen kann man dies am besten mit einem Daumenkino, bei dem durch das Abblättern mehrerer Einzelbilder einer fortlaufenden Bildfolge eine Bewegung vorgetäuscht werden kann. Ein einzelnes Bild stellt hierbei ein *Frame* dar. Und auch in Unity werden keine echten Bewegungen von den Objekten gemacht, sondern es werden von der Game Engine ganz schnell Bilder auf dem Monitor gezeichnet, auch Rendern genannt, die rasch hintereinander dargestellt werden.

2.4 Das erste Übungsprojekt

Um sich mit den vorgestellten Funktionalitäten vertraut zu machen, ist es am besten, diese in der Praxis anzuwenden. Im Folgenden erläutern wir deshalb ein mögliches Vorgehen, wie Sie ein kleines Übungsprojekt erstellen können, um die vorgestellten Funktionen kennenzulernen. In den folgenden Kapiteln werden wir uns ab und zu auch auf dieses Projekt beziehen, um Dinge zu testen.

1. Erstellen Sie als Erstes über FILE/NEW PROJECT ein neues Projekt mit dem *Setup-Default* „3D“. Sie können den von Unity vorgeschlagenen Projektnamen einfach übernehmen oder natürlich auch selber einen wählen.

Nach dem Start des neuen Projekts sollten Sie in dem *Hierarchy*-Fenster nun ein Objekt namens „Main Camera“ und eines namens „Directional Light“ vorfinden. „Main Camera“ ist Ihr Kamera-Objekt, aus dessen Sicht der Spieler das Spiel betrachtet. Überprüfen Sie, dass dieses Objekt auf die Koordinaten (0, 1, –10) positioniert ist und eine Rotation von (0, 0, 0) besitzt. „Directional Light“ ist das Hauptlicht, dessen Position und Ausrichtung zunächst keine besondere Rolle spielt.

2. Klicken Sie im *Project Browser* den Oberordner „Assets“ an und legen Sie über die rechte Maustaste CREATE/FOLDER einen neuen Ordner mit dem Namen „Scenes“ an.
3. Speichern Sie die neue Szene über das Hauptmenü *File/Save Scene as*. In dem *Save Scene*-Dialog wird Ihnen nun der neue Ordner „Scenes“ angeboten. Wählen Sie diesen aus, geben Sie Ihrer Szene den Dateinamen „Test“ und drücken Sie auf SPEICHERN.
4. Öffnen Sie nun im *Project Browser* den Ordner „Scenes“. Sie werden dort nun einen Eintrag finden, der die Szene „Test“ symbolisiert. Wenn Sie später weitere Szenen erzeugen, sollten Sie diese nun ebenfalls in diesem Ordner abspeichern.
5. Fügen Sie jetzt Ihrer Szene einen Würfel hinzu. Dies machen Sie über GAMEOBJECT/3D OBJECT/CUBE.
6. Markieren Sie den Würfel „Cube“ im *Hierarchy*-Fenster und drücken Sie F, um in der *Scene View* auf den Würfel zu fokussieren.
7. Verschieben Sie nun den Würfel auf die Position (0, 5, 0). Versuchen Sie dies sowohl mit den *Transform-Tools* wie auch direkt über die *Inspector*-Eigenschaften des Würfels. Nutzen Sie hierbei zum Drehen und Verschieben der Ansicht der *Scene View* auch die bereits vorgestellten Navigationsmöglichkeiten dieses Fensters.

8. Erzeugen Sie nun über GAMEOBJECT/3D OBJECT/PLANE eine Fläche und positionieren Sie diese auf der Position (0, 0, 0).
9. Starten Sie das Spiel über die Play-Taste. Ihre beiden Objekte sollten nun in der *Game View* zu sehen sein.
10. Wechseln Sie in die *Scene View* und verschieben Sie dort den „Cube“ auf Position (3, 5, 0).
11. Wechseln Sie wieder in die *Game View* und Sie werden sehen, dass der Würfel zur Seite verschoben wurde.
12. Stoppen Sie nun das Spiel. Der Würfel wird wieder auf (0, 5, 0) zurückgesetzt.
13. Als Nächstes fügen Sie dem Würfel eine Physik-Komponente zu, damit dieser von einer virtuellen Erdanziehungskraft angezogen wird. Markieren Sie hierfür den Würfel in der Hierarchy und klicken Sie im Hauptmenü auf COMPONENT/PHYSICS/RIGIDBODY.
14. Starten Sie das Spiel und beobachten Sie, wie der Würfel nach unten fällt und auf der Fläche liegen bleibt.
15. Speichern Sie die Szene über FILE/SAVE SCENE.
16. Beenden Sie das Spiel und fügen Sie nun die Fläche dem Würfel als Kind-Objekt hinzu.
17. Wenn Sie den Würfel verschieben, drehen oder auch skalieren, werden Sie sehen, wie sich die Fläche mitverändert.
18. Setzen Sie die Transform-Werte des Würfels wieder auf die Originalwerte zurück und starten noch einmal das Spiel. Da die Fläche ein Kind-Objekt vom Würfel ist, fällt dieser nun gemeinsam mit dem Würfel runter. Da die Objekte auf kein anderes Objekt mehr fallen können, schließlich gibt es keine anderen in der Szene, können Sie das Spiel nun wieder stoppen.

Da wir dieses Projekt im Laufe des Buches noch häufiger nutzen wollen, speichern Sie die Szene bitte nicht noch ein weiteres Mal, damit der Szenenzustand vom Schritt 16 bzw. 17 erhalten bleibt. Ansonsten können Sie jetzt noch weiter mit diesem Projekt die vorgestellten Funktionen testen.

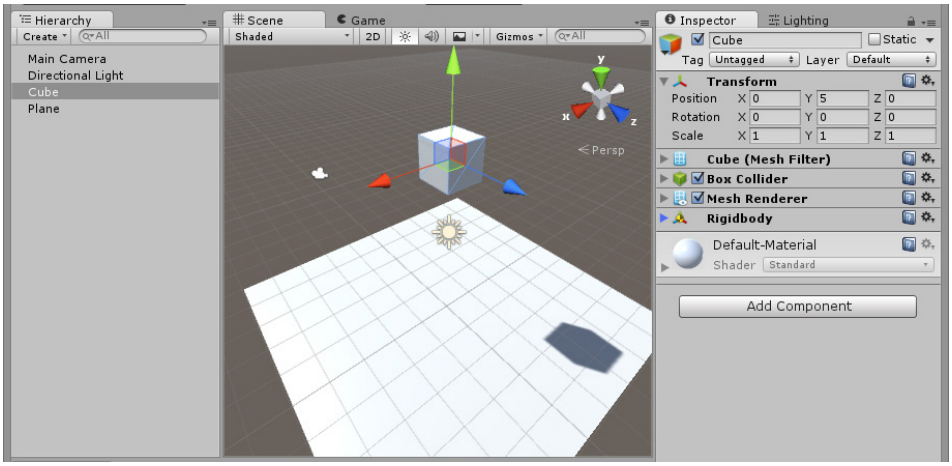

Bild 2.31 Erstes Übungsprojekt

Dieses erste kleine Übungsprojekt finden Sie auch im Online-Bereich des Buches.

3 C# und Unity

Auch wenn Unity bereits für viele Standardaufgaben fertige Komponenten mitliefert (zum Beispiel für die Bereiche Audio, Physik, Partikeleffekte oder auch das Rendering), muss die eigentliche, individuelle Spiellogik immer noch von Ihnen selber programmiert werden. In Unity wird dies mit Skripten gemacht, einzelne Textdateien, die beim Erstellen des Spiels von einem Programm (genannt Compiler) in für Computer verständliche Befehle übersetzt werden. Skripte verhalten sich in Unity für gewöhnlich wie Komponenten und werden auch dementsprechend an *GameObjects* angehängt.

Da wir in diesem Buch mit der Programmiersprache C# arbeiten, möchten wir Ihnen als Erstes diese Sprache und wie sie in Unity eingesetzt wird, etwas näherbringen. Sollten Sie sich bereits mit C# auseinandergesetzt haben, möchten wir Ihnen trotzdem empfehlen, dieses Kapitel durchzulesen, da hier auch einige Besonderheiten erläutert werden, die speziell in Unity gelten.

3.1 Die Sprache C#

C# wurde von der Firma Microsoft entwickelt und gehört zu den sogenannten *Objektorientierten Programmiersprachen*. Sie wurde entwickelt, um Anwendungen mit dem .NET Framework zu entwickeln, einer Plattform, die eine große Palette an Klassenbibliotheken, Programmierschnittstellen und Dienstprogrammen zur Verfügung stellt. Unity nutzt die Sprache aber in Kombination mit dem Mono-Framework, einer Open-Source-Variante des .NET Frameworks, wodurch C#-Anwendungen auch auf Nicht-Microsoft-Systemen betrieben werden können (bekannt auch als Mono-Projekt).

Da die Sprache sehr umfangreich ist, werden wir Ihnen im Folgenden aber nicht alle Möglichkeiten von C# vorstellen und den Fokus vor allem auf die Themen legen, die für Sie als Unity-Programmierer interessant sind. Sollten Sie weitere Informationen zur Sprache C# haben wollen, sollten Sie über die bekannten Suchmaschinen fündig werden. Außerdem gibt es viele gute Bücher, wie z. B. die Visual C#-Bücher von Walter Doberenz und Thomas Gewinnus, die die Sprache in allen ihren Facetten behandeln.

Sollten Sie einige Themen nicht gleich verstehen, verzweifeln Sie nicht. Sie werden sicher den einen oder anderen Aha-Effekt haben, wenn Sie auch die anderen Buchkapitel durchgearbeitet haben und diese Stellen später erneut durchlesen.

Als Einstieg möchten wir Ihnen noch ein kleines Skript zeigen, das Sie am Ende dieses Buch für das Beispiel-Game programmieren werden. Das Skript dient dem Verwalten der Lebensstärke und wird sowohl vom Spieler wie auch von den Gegnern genutzt.

Listing 3.1 HealthController-Skript

```
using UnityEngine;
using System.Collections;
public class HealthController : MonoBehaviour {
  public float health = 5;
  private bool isDead = false;
  void ApplyDamage(float damage) {
    health -= damage;
    if(health <= 0 && !isDead) {
      isDead = true;
      Dying();
    }
    else {
      Damaging();
    }
  }
  public virtual void Damaging()
  {
  }
  public virtual void Dying ()
  {
  }
}
```

3.2 Syntax

Zunächst einmal einige Worte zum grundsätzlichen Programmieren: Programmcode (auch Quelltext oder Source-Code genannt) besteht aus Code-Zeilen, die nacheinander abgearbeitet werden. Damit der Computer diese Zeilen auch versteht, werden diese mit einem Programm namens Compiler in Maschinensprache übersetzt, das sogenannte Kompilieren. Damit der Compiler auch weiß, wann das Ende einer einzelnen Codezeile erreicht ist, wird in C# am Ende ein Semikolon **;** geschrieben. Weitere wichtige Zeichen sind in C# die geschwungenen Klammern {}. Diese werden dafür genutzt, um zusammenhängende Codeblöcke zu kennzeichnen.

3.3 Kommentare

Neben dem eigentlichen Code wird es auch vorkommen, dass Sie Anmerkungen zu Ihrem Code hinzufügen möchten, die eben nicht vom Computer ausgeführt werden sollen. Solche nicht auszuführenden Zeilen werden auch Kommentare genannt. Einen Kommentar markieren Sie mit einem Doppelslash //. Alles, was dahinter bis zum nächsten Zeilenumbruch steht, gilt als nicht auszuführender Code. Was in dieser Zeile vor dem Doppelslash steht, gilt aber als ausführbar!

Listing 3.2 Einzeiliger Kommentar

```
int zahl1; //Dies ist eine Befehlszeile, die auch kompiliert wird.
zahl1 = 2;
```

Wenn Sie mehrere Zeilen als Kommentare markieren möchten, können Sie dies durch das Schreiben von /* am Anfang und */ am Ende erreichen:

Listing 3.3 Mehrzeiliger Kommentar

```
/*
Dies ist alter Code
int lifePoints;
lifePoints= 2;
*/
int lifePoints = 2; //Das ist der neue Code
```

Normalerweise werden Kommentare von der Entwicklungsumgebung farblich hervorgehoben, z. B. in Grün oder Rot.

3.4 Variablen

Variablen sind zunächst einmal nichts anderes als Platzhalter, um Werte zu speichern. Der Name einer Variablen muss immer eindeutig sein und wird von Ihnen vergeben. Auch wenn dieser zunächst einmal keine Rolle spielt, müssen Sie darauf achten, dass er keine Sonderzeichen, Umlaute oder Leerzeichen besitzt. Außerdem dürfen Variablennamen nicht mit einer Zahl beginnen.

3.4.1 Namenskonventionen

Um die Lesbarkeit von Programmcode zu vereinfachen, gibt es einige Konventionen, an die Sie sich halten sollten. Zum einen sollten die Variablennamen den Inhalt der Variablen beschreiben. Hierbei nutzen Sie am besten englische Begriffe, z. B. `speed`. Außerdem sollten Variablenbezeichnungen immer mit einem Kleinbuchstaben beginnen. Bei Namen, die aus mehreren Wörtern bestehen, sollten diese durch Großbuchstaben voneinander abgehoben werden, z. B. `enemySpeed`. Diese Schreibweise wird übrigens auch *Camel Case* genannt.

3.4.2 Datentypen

Damit der Compiler auch weiß, was für Werte in einer Variablen gespeichert werden dürfen (Zahlen, Buchstaben . . .), müssen Sie der Variablen noch sagen, von was für einem Datentyp sie ist. Dies wird als Variablendeklaration bezeichnet. Häufig genutzte Datentypen sind in der Unity-Spieleentwicklung:

- **string** für Text
- **int** für Ganzzahl
- **float** für Fließkommazahl
- **bool** für Boolean
- **enum** für Enumeration

In C# beginnt die Deklaration einer Variablen mit dem Datentyp gefolgt von dem Namen der Variablen. Am Ende folgt natürlich das Semikolon:

Listing 3.4 Variablendeklaration

```
int lifePoints;
float height;
string name;
```

Um diesen Variablen Werte zuzuweisen, wird in C # das Gleichzeichen genutzt:

Listing 3.5 Werte zuweisen

```
lifePoints = 2;
name = "Carsten";
height = 1.5F;
```

Wie Sie sehen, benutzen wir bei Texten zusätzlich zu dem Text an sich noch Anführungsstriche, um dem Compiler zu sagen, wo der zugewiesene Text beginnt und wo er endet. Wenn ich Kommazahlen zuweise (hierfür benötigen wir den Datentyp *float*), schreiben wir statt des Kommas einen Punkt. Zudem schreiben wir noch bei *float*-Variablen ein F hinter dem Wert, damit der Compiler weiß, dass es sich bei dem Wert auch tatsächlich um eine Fließkommazahl vom Typ *float* handelt. Alternativ gibt es nämlich in C# noch den Typ *double*, welcher aber einen größeren Zahlenbereich abdeckt als *float* und deshalb nicht in eine *float*-Variable „hineinpasst“:

Listing 3.6 Unterschied float und double

```
float gravity;
gravity = 9.81F;
double dblGravity;
dblGravity= 9.81;
```

In Unity ist aber *float* geläufiger als *double*, weshalb wir in Zukunft nur von *float*-Werten sprechen werden. Um den obigen Code etwas zu kürzen, können Sie gleich beim Deklarieren der Variablen diese auch mit einem Wert vorbelegen (auch Initialisierung genannt):

Listing 3.7 Variablendeklaration mit Initialisierung

```
float gravity = 9.81F;
```

Ein weiterer Typ, der sehr viel in der Spieleprogrammierung Verwendung findet, ist der Datentyp Boolean *(bool)*. Dieser Typ kann nur zwei Zustände annehmen: *TRUE* und *FALSE*, also wahr oder falsch. Ein typisches Beispiel hierfür ist eine Checkbox, also ein Haken in der GUI, um beispielsweise eine Funktion zu aktivieren oder zu deaktivieren. Je nach Zustand dieser Checkbox hat nun die boolesche Variable, die den Wert im Programmcode speichert, den Wert *TRUE* oder *FALSE*.

Listing 3.8 Boolesche Variable

```
bool isAttacking = false;
```

3.4.3 Schlüsselwort var

Neben dem Festlegen eines Datentyps mit *int*, *string* usw. gibt es in C# auch die Möglichkeit, eine Variable mit `var` zu definieren. In diesem Fall wird der Datentyp dieser Variablen erst mit dem Zuweisen des ersten Wertes bestimmt. Dann entscheidet der Compiler selber, welcher Datentyp hier der richtige ist, und legt diesen fest. Jede zukünftige Zuweisung, die einen anderen Typ erfordern würde, führt dann aber zu einem Fehler.

Listing 3.9 Variablendeklaration mit var

```
var lifePoint;
lifePoint = 5;
```

3.4.4 Datenfelder/Array

Wenn Sie mehrere Variablen eines Datentyps benötigen, können Sie diese über eine Array-Definition erstellen. Ein Array ist also kein Datentyp an sich, sondern eher eine Variation eines Datentyps.

3.4.4.1 Arrays erstellen

Ein Array definieren Sie durch eckige Klammern hinter dem eigentlichen Datentyp:

Listing 3.10 Array-Deklaration

```
int[] speedLimits;
```

Im Gegensatz zu einer normalen Variablen existiert das Array aber noch nicht durch die alleinige Definition der Variablen. Dies müssen Sie noch einmal extra machen, und zwar mit dem Schlüsselwort `new`. Diesen Vorgang nennt man auch Instanziieren. Zudem müssen Sie beim Instanziieren dem Array noch mitgeben, wie groß es sein soll, also aus wie vielen Integer-Variablen das Array bestehen soll. Das sieht dann wie folgt aus:

Listing 3.11 Array-Instanziierung

```
speedLimits= new int[5];
```

Oder etwas kürzer:

Listing 3.12 Array-Deklaration mit Instanziierung

```
int[] speedLimits = new int[5];
```

Sie können beim Instanziieren auch gleich jedem Element (Item) des Arrays auch einen Startwert mitgeben. Dies kann dann so aussehen:

Listing 3.13 Array-Initialisierung mit verschiedenen Werten

```
int[] speedLimits= new int[5]{30,50,80,100,120};
```

Wichtig ist hierbei, dass Sie auch tatsächlich so viele Werte in den geschweiften Klammern übergeben, wie das Array Elemente besitzt.

3.4.4.2 Zugriff auf ein Array-Element

Wenn Sie nun auf ein bestimmtes Element dieses Arrays zugreifen möchten, müssen Sie über eine Index-Zahl das richtige Item auswählen. Für Unerfahrene wird dies erst einmal eine Umstellung bedeuten, denn das erste Element eines Arrays hat den Index 0, nicht 1! Damit hat ein Array mit fünf Elementen den höchsten Index 4!

Listing 3.14 Array-Elemente mit Index ansprechen

```
int[] speedLimits= new int[5]{30,50,80,100,120};
speedLimits[0] = 20; // Wert betrug vorher 30
speedLimits[4] = 140; // Wert betrug vorher 120
```

3.4.4.3 Anzahl aller Array-Items ermitteln

Es kann sein, dass Arrays auch automatisch erstellt werden. Wenn Sie nun wissen möchten, aus wie vielen Items denn nun das Array besteht, können Sie dies über die Eigenschaft `Length` ermitteln:

Listing 3.15 Array-Größe ermitteln

```
int len = speedLimits.Length;
```

3.4.4.4 Mehrdimensionale Arrays

Sie haben auch die Möglichkeit, ein Array mit mehreren Dimensionen zu erstellen.

Stellen Sie sich vor, Sie möchten mit einem Array ein quadratisches Rasterspielfeld darstellen (ein Schachbrett zum Beispiel). Dann ist es sehr unübersichtlich, ein Array mit 64 Elementen zu erstellen. Einfacher ist es, ein Array mit zwei Dimensionen zu erstellen, wo der erste Index die X-Achse (beim Schach wären es die Buchstaben A bis H) darstellt und der zweite die Y-Achse. Eine solche Definition sieht in C# dann so aus:

Listing 3.16 Mehrdimensionales Array

```
int[,] tile= new int[8,8];
```

Bedenken Sie auch hier, dass jeder Index des Arrays mit einer 0 beginnt. Wenn Sie nun die Spielfiguren mit Zahlen verschlüsseln (0 bedeutet keine Figur, 1 stellt einen Bauer dar, 2 ist ein Turm), können Sie so kinderleicht die Positionen der Figuren in diesem Array speichern:

Listing 3.17 Mehrdimensionales Array als Schachbrett-Speicher

```
tile[0,0] = 2; //Auf A1 befindet sich ein Turm.
tile[0,1] = 1; //Auf A2 befindet sich ein Bauer.
tile[1,1] = 1; //Auf B2 befindet sich ein Bauer.
tile[0,2] = 0; //Auf A3 befindet sich keine Figur.
```

3.5 Konstanten

Es gibt immer wieder Werte, die nicht verändert werden sollen bzw. dürfen. Diese können Sie als Konstanten definieren, wodurch diese nicht mehr verändert werden können. Hierfür müssen Sie vor dem Datentyp das Schlüsselwort `const` schreiben. Zudem ist es notwendig, Konstanten bereits bei der Deklaration zu initialisieren:

Listing 3.18 Konstanten-Deklaration

```
const float gravity = 9.81F;
```

3.5.1 Enumeration

Mit *Enumerationen* können Sie lesbare Auflistungen von Konstanten erstellen. Was etwas kryptisch klingt, ist aber eine ganz praktische Sache. Stellen Sie sich vor, Sie wollen den Zustand einer Figur speichern. Dieser hat drei Zustände: Stehen, Gehen, Springen. Um abzufragen, in welchem Zustand sich die Figur gerade befindet, möchten Sie diesen nun in einer Variablen speichern. Hierfür definieren Sie zunächst einmal eine Enumeration (kurz *Enum*), die diese drei Werte besitzt. Wir wollen diese Enumeration einfach mal State nennen:

Listing 3.19 Enum-Deklaration

```
enum State {Idle,Walk,Jump}
```

Danach können Sie nun eine Variable vom Typ dieser Enumeration erstellen und dieser einen der drei Werte zuweisen.

Listing 3.20 Enumeration nutzen

```
State myState;
myState = State.Idle;
```

3.6 Typkonvertierung

Es ist auch möglich, eine Variable eines Typs in einen anderen Typ umzuwandeln. Voraussetzung ist natürlich, dass es auch vom Inhalt her passt. Eine *Integer*-Variable kann immer in eine *Float*-Variable umgewandelt werden. Der umgekehrte Weg funktioniert nur dann, wenn die *Float*-Zahl keinen Kommawert besitzt. Bei der Typkonvertierung wird vor dem zu konvertierenden Wert einfach in runden Klammern der Zieltyp angegeben:

Listing 3.21 Typkonvertierung

```
int lifePoints;
string lifePointsText = "2";
lifePoints = (int) lifePointsText;
```

Zahlen in ein *String* umzuwandeln, geht sogar noch einfacher. Diese besitzen von Haus aus eine Methode/Funktion (was dies ist, erklären wir gleich noch), die diese Aufgabe übernimmt. Hierfür geben Sie hinter der Zahlenvariablen einen Punkt an und danach den Text `ToString()`. Bereits beim Tippen des Punktes sollte eine Auswahl in MonoDevelop erscheinen, die diesen Befehl zur Verfügung stellt.

Listing 3.22 ToString-Beispiel

```
lifePointsText = lifePoints.ToString();
```

Diese Funktion ist sehr nützlich, da auf diese Weise die Integer-Variable wie ein String behandelt werden kann:

Listing 3.23 String-Verkettung

```
lifePointsText = "Sie besitzen " + lifePoints.ToString() + " Lebenspunkte.";
```

Wie Sie sehen, können Sie in C# mithilfe des +-Zeichens Strings aneinanderhängen. Dieses Verfahren wird auch als String-Verkettung bezeichnet und wird sehr häufig bei Textausgaben genutzt.

3.7 Rechnen

Wenn Sie mit Zahlen rechnen möchten, dann sollten Sie die folgenden Rechenbefehle kennen, die in C# genutzt werden:

Tabelle 3.1 Rechenoperatoren

Operator	Funktion
+	Addieren
-	Subtrahieren
*	Multiplizieren
/	Dividieren
%	Modulo (Division mit Rest)
++	Wert um 1 erhöhen
—	Wert um 1 vermindern

Das Ergebnis der Rechnung weisen Sie mit einem einfachen Gleichzeichen einer Variablen zu.

Listing 3.24 Beispielrechnung

```
int health = 5;
int damage = 1;
health = health - damage;
```

Sie können Rechnungen auch verkürzen. Nehmen wir an, Sie wollen einen Wert um eins erhöhen, dann können Sie auch die folgenden Schreibweisen nutzen:

Listing 3.25 Verkürzte Schreibweise einer Addition

```
lifePoints += 1;
```

Beim Sonderfall des Hochzählens bzw. Runterzählens um 1 geht es sogar noch kürzer:

Listing 3.26 Werte um eins erhöhen und reduzieren

```
lifePoints ++;
lifepoint --;
```

3.8 Verzweigungen

Nachdem Sie nun alle wichtigen Vorkenntnisse für das Programmieren kennengelernt haben, können wir endlich mit dem eigentlichen Programmieren beginnen. Denn hier geht es ja nicht darum, einfach nur Variablen zu definieren und mit Werten zu befüllen, vielmehr geht es darum, abhängig von diesen Werten Entscheidungen zu treffen.

Hier kommen sogenannte Verzweigungen ins Spiel. Sie entscheiden aufgrund von definierten Bedingungen, welche Codeblöcke als Nächstes ausgeführt werden sollen.

3.8.1 if-Anweisungen

Die einfachste Verzweigung ist die sogenannte *if-Anweisung*. Sie arbeitet wie eine Weiche, die abhängig von einer Bedingung einen Code ausführen lässt. Zudem kann ein Alternativcode angegeben werden, der ausgeführt werden soll, wenn die Bedingung nicht erfüllt ist, sodass sich der gesamte Code dann wie folgt verhält: „Wenn die Bedingung zutrifft, dann mache dies, ansonsten mache das."

Eine if-Anweisung wird mit dem Signalwort `if` eingeleitet, gefolgt von der Bedingung, die in runden Klammern eingeschlossen wird. In geschwungenen Klammern folgt dann der eigentliche Code. Der optionale Alternativcode wird dann mit dem Wort `else` eingeleitet.

Listing 3.27 Einfache if-Anweisung

```
if(lifePoints == 2) {
  message = "Du hast 2 Leben";
}
else {
  message = "Keine Ahnung, wie viele Leben Du hast." +
    "Aber es sind auf jeden Fall keine 2.";
}
```

Wenn der Codeblock bzw. der else-Zweig nur aus einer einzigen Zeile besteht, kann auch auf die geschwungenen Klammern verzichtet werden.

Listing 3.28 if-Anweisung ohne Klammern

```
if(lifePoints == 2)
  message = "Du hast 2 Leben";
```

Die Bedingungen der if-Anweisung werden dabei mit Vergleichsoperatoren angegeben, die Sie der Tabelle 3.2 entnehmen können.

Tabelle 3.2 Vergleichsoperatoren

Operator	Erläuterung
a == b	Vergleicht, ob a gleich b ist
a != b	Vergleicht, ob a ungleich b ist
a > b	Vergleicht, ob a größer b ist
a < b	Vergleicht, ob a kleiner b ist
a <= b	Vergleicht, ob a kleiner oder gleich b ist
a >= b	Vergleicht, ob a größer oder gleich b ist

if-Anweisungen können aber nicht nur auf einer Ebene existieren. Sie können auch ineinander verschachtelt sein, sodass weitere if-Bedingungen abgefragt werden können, wenn etwas zutrifft oder eben nicht zutrifft.

Listing 3.29 Verschachtelte if-Anweisung

```
if(lifePoints == 2) {
  message = "Du hast 2 Leben.";
```

```
}
else {
  if (lifePoints != 3) {
    message ="Du hast keine 2 und keine 3 Leben.";
  }
  else {
    message ="Du hast 3 Leben.";
  }
}
```

Bei booleschen Werten können Sie auch anstatt `if (isAlive == true){}` eine Kurzschreibweise `if (isAlive) {}` nutzen. Möchten Sie nun auf FALSE anstatt auf TRUE prüfen, können Sie das Ergebnis einfach mit einem Ausrufezeichen invertieren.

Listing 3.30 if-Anweisung mit negativem Bool-Wert

```
if (!isAlive) {
  Destroy(gameObject);
}
```

3.8.1.1 Komplexere if-Anweisungen

Ist die Code-Ausführung von mehreren Bedingungen abhängig, können Sie die Bedingungen mit Operatoren miteinander logisch verknüpfen. Die am meisten genutzten Verknüpfungen sind die sogenannten bedingten Operatoren.

Tabelle 3.3 Bedingte Operatoren

Operator	Funktion	Erläuterung
&&	UND	Wenn beide Operanden*TRUE* sind, ist auch das Ergebnis *TRUE*
||	ODER	Wenn mindestens ein Operand*TRUE* ist, ist auch das Ergebnis *TRUE*

In der folgenden *if-Anweisung* müssen beide Bedingungen *TRUE* sein, damit wir in den Ausführungsblock gelangen:

Listing 3.31 if-Anweisung mit zwei Bedingungen

```
if (health > 0 && lifePoints > 0) {
  //Code...
}
```

Beachten Sie, dass Sie auch hier ein Ausrufezeichen zum Negieren nutzen können. So wird der Code dann ausgeführt, wenn entweder `lifepoints` größer als 0 ist oder die boolesche Variable `isDetected` den Wert *FALSE* besitzt.

Listing 3.32 if-Anweisung mit zwei ODER-verknüpften Bedingungen

```
if (lifePoints > 0 || !isDetected) {
  //Code...
}
```