2. Auflage

# Spiele entwickeln mit UNREAL ENGINE 4

jonas RICHARTZ

## Programmierung mit Blueprints: Grundlagen & fortgeschrittene Techniken

Im Internet: Die Games zum Buch mit Projektdateien und weiterführende Videotutorials

HANSER

Richartz

# Spiele entwickeln mit Unreal Engine 4

Jonas Richartz

# Spiele entwickeln mit Unreal Engine 4

## Programmierung mit Blueprints.

## Grundlagen & fortgeschrittene Techniken

2., erweiterte Auflage

HANSER

Die Autoren:

*Jonas Richartz*, Leichlingen

*Benedikt Engelhard*, Nürnberg (Kapitel 23, Virtual Reality mit Unreal Engine 4)



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: Katrin Powik, Lassan
Umschlagdesign: Marc Müller-Bremer, München, www.rebranding.de
Umschlagrealisation: Stephan Rönigk
Gesamtherstellung: Kösel, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

Print-ISBN: 978-3-446-45290-9
E-Book-ISBN: 978-3-446-45369-2

# Inhalt

# Vorwort

Der Traum, ein eigenes Computerspiel zu erstellen, ist mittlerweile keine Seltenheit. Aufgrund der heutigen Möglichkeiten ist dies auch leichter als je zuvor. Ein kleines Spiel kannst du in wenigen Schritten zusammenstellen, und oft ist nur deine eigene Fantasie dein Limit.

Ich möchte dir in diesem Buch die Unreal Engine 4 vorstellen und dir alle möglichen Grundlagen dazu beibringen. Außerdem verdeutliche ich dir die Gedankengänge, die während der Entwicklung erforderlich sind. Die Unreal Engine 4 ist die neueste und bis dato beste Engine von Epic Games, die dir vollkommen kostenlos zur Verfügung gestellt wird. Du brauchst keine komplizierte Programmiererfahrung, um Spiele mit dieser Engine zu entwickeln, und die Community steht dir jederzeit mit Rat und Tat beiseite!

Ich hätte nie gedacht, jemals in meinem Leben ein Buch zu schreiben, bin aber sehr froh, die Gelegenheit dafür erhalten zu haben. Ich möchte mich bei Sieglinde Schärl bedanken, die mir die Tür zu diesem Buch geöffnet und mich während des Schreibens gut begleitet hat. Weiter gilt mein Dank Sylvia Hasselbach für die weitere Betreuung, Jürgen Dubau für die Korrekturen sowie dem ganzen Team des Hanser Verlags, die alle geholfen haben, dieses Buch zustande zu bringen.

Besonderer Dank gilt Benedikt Engelhard für das VR-Kapitel sowie nochmals Sylvia Hasselbach für die Betreuung der zweiten Auflage.

Sehr dankbar bin ich meiner Mutter, meinem Vater und meinem Bruder für ihre Unterstützung während der Monate des Schreibens. Ein besonderes Dankeschön geht an Christian Albrecht für seine Unterstützung und die Erstellung meiner Website.

Nicht zu vergessen der Dank an den Community Manager Chance Ivey sowie alle bei Epic Games für die fantastische Engine. Last, but not least danke ich meiner Community auf YouTube für die netten Kommentare, die mich überhaupt motiviert haben, mich an ein solches Buch zu wagen! Wenn mein Buch euch allen hilft, tolle Spiele zu erstellen, hat sich die Mühe gelohnt!

Leichlingen im August 2017

*Jonas Richartz*

# 1 Einleitung

Computerspiele zu entwickeln ist sehr viel Arbeit und kann auch nur mit jahrelanger Erfahrung bewältigt werden. So zumindest lautete die Meinung vieler in den vergangenen Jahren. Doch wie in keiner anderen Industrie verändert sich die Branche rund um die Spieleentwicklung rasant. Der Indie-Markt ist größer als je zuvor, und AAA-Spiele werden immer aufwendiger und detaillierter.

Um da mithalten zu können, benötigt man eine Engine, welche einem die Arbeit erheblich vereinfacht und zukunftsorientiert ist. Die Unreal Engine 4 von Epic Games gehört zu einer der modernsten und innovativsten Engines auf dem offenen Markt – egal ob es sich um PC, Konsolen, Mobile oder Web-Entwicklung handelt und ob man ein Spiel in 2D oder 3D entwickelt. Die Unreal Engine 4 läuft problemlos auf Windows- und Mac-Systemen (an Linux wird zurzeit noch gearbeitet), und durch ständige Updates werden eventuelle Probleme schnell behoben und neue Elemente hinzugefügt.

Bisher war es in den meisten Fällen undenkbar, dass bei einer großen Engine ohne Unmengen an Mehrkosten auch der Source-Code mitgeliefert wird. Mit der Unreal Engine 4 bekommt man jedoch erstmalig die Engine und den C++ Source-Code kostenlos. Besonders an der Unreal Engine 4 ist jedoch auch das visuelle Scripting. Noch nie war die Entwicklung von komplexen Mechaniken so einfach wie mit Blueprints, auf die sich dieses Buch hauptsächlich beziehen wird.

Die Unreal Engine 4 findet sich unter *www.unrealengine.com*, sie wird oft mit UE 4 und manchmal nur mit UE oder Unreal abgekürzt.

## 1.1 Was brauche ich?

Du brauchst zuerst einen PC oder Laptop, auf dem du entwickeln willst. Empfohlen wird ein Windows 64-Bit-System (auf dem ich dieses Buch schreibe), es funktioniert aber auch unter Mac. 8 GB Ram und eine DirectX-11-fähige Grafikkarte wären für gute Performance wünschenswert, aber nicht zwingend notwendig.

Vorkenntnisse im Bereich der Spieleentwicklung und Programmierung werden nicht benötigt. Da die Engine zurzeit nur auf Englisch erhältlich ist, sind Grundkenntnisse in Englisch

von Vorteil, aber auch ohne diese wirst du dich mithilfe des Buches zurechtfinden können. Generell ist sehr viel in Englisch verfasst, vor allem was Hilfen und Ressourcen online angeht, aber es sollte dich nicht davon abhalten, dich in die Engine einzuarbeiten.

Aber für dich am Wichtigsten sind die Motivation und die Lust, Neues zu lernen, um deinen Horizont zu erweitern. Zusätzlich zu diesem schicken Buch findest du auf meiner Webseite weitere Inhalte zum Herunterladen mit begleitenden Videos und Projektdateien.

Webseite: *www.jonasrichartz.de*

Password: **sd092qnbf13**

## 1.2 Was lerne ich?

Du wirst den kompletten Umgang mit den Basics der Engine lernen und dich nach Beendigung des Buches in der Engine zurechtfinden können, eigenständig neue Elemente in deinem Spiel einfügen können und den kompletten Denkprozess kennenlernen, der hinter der Spielentwicklung und einzelnen Features steht. Ich werde in diesem Buch eher auf die 3D-Entwicklung eingehen als die 2D-Entwicklung, was vor allem den Grund hat, dass die 3D-Entwicklung mit der Engine einfacher ist als die Entwicklung in 2D. Generell ist die Engine auf 3D-Entwicklung spezialisiert, auch wenn die Entwicklung von 2D-Spielen ebenfalls möglich ist. Ich werde auch eher auf die Entwicklung für Windows-Plattformen eingehen als auf Web, Konsolen, VR und Mobile. Die Engine unterstützt vom Prinzip her alle erdenklichen Plattformen, jedoch benötigt man bei der Entwicklung für Playstation 4 und Xbox One eigene Lizenzen von Sony und Microsoft. Mit den Lizenzen muss man dann zusätzlich Epic Games kontaktieren, um speziellen Code zu bekommen, der das Exportieren auf den jeweiligen Plattformen erlaubt. Grundsätzlich ist die Entwicklung für alle Plattformen aber sehr ähnlich und unterscheidet sich meist nur beim Vorgehen für das Exportieren auf den jeweiligen Plattformen. Natürlich muss man für Mobile und Web deutlich mehr auf die Performance achten als für Windows und Konsolen.

Du wirst **kein** C++ lernen, sondern die Handhabung von **Blueprints**, sodass du alles machen kannst, ohne eine einzige Zeile Code zu schreiben, dabei aber dennoch die Grundkenntnisse des Programmierens lernst. Das wird dir helfen, wenn du später im C++ Source-Code programmieren willst. Aber fürs Erste werden Blueprints mehr als genug sein, um fast alles umzusetzen, was du dir vorstellen kannst.

Du fragst dich nun sicherlich, was Blueprints sind und warum du ohne Programmierung auskommen kannst. Die Antworten findest du in **Kapitel 4**. Aber schon mal vorweg: Blueprints sind vom Prinzip nichts anderes als C++-Klassen, nur dass man keine Tausende Zeilen von Code schreiben muss, sondern mit Bausteinen arbeitet, welche im Hintergrund für dich Code ausführen. Anstatt also komplett mehrere Zeilen Code schreiben zu müssen, wirst du hier viele verschiedene Bausteine aneinanderknüpfen, um deine Logik aufzubauen, und hast somit alles auf einen Blick. Doch dazu später mehr.

Ich werde in diesem Buch alle wichtigen Bereiche der Engine durchgehen: von einfachen Kapiteln im Bereich der Landschaftserstellung und Licht- und Schattengestaltung bis hin zu komplexeren Themen wie Animationen, Netzwerke und die erste einfache KI (künstliche Intelligenz). Du wirst eine komplette Übersicht zu allen relevanten Themen bekommen, dazu zwischendurch einigen Aufgaben mit Lösungsvorschlägen im Videoformat. Des Weiteren versuche ich, dir immer kleine Beispiele zu zeigen, womit du dir ein besseres Bild über die Funktionsweise und Nützlichkeit der einzelnen Aspekte machen kannst.

Wie du vielleicht schon festgestellt hast, schreibe ich dieses Buch sehr offen, als spräche ich mit dir persönlich. Es ist mir wichtig, dass du dich wohlfühlst und dass nicht alles in einem Fremdwortwirrwarr endet.

## 1.3 Lizenzen

Wie bei fast jeder Engine üblich, musst du auch bei der Unreal Engine 4 gewissen Lizenzen zustimmen. Du kannst die Engine für nicht-kommerzielle und kommerzielle Projekte in den Bereichen der Spieleentwicklung kostenlos nutzen, aber auch für Film, Architektur und vieles mehr, was du mit der Engine anstellen kannst. Du darfst dir den vollen Source-Code kostenlos herunterladen und für den eigenen Nutzen abändern, jedoch nicht weiterverkaufen – es sei denn, du erstellst ein Plug-In, das du auf dem Unreal Engine Marketplace verkaufst. Es gibt hierbei keine verschiedenen Versionen der Engine, welche aufgeteilt ist in Free, Commercial oder Pro, sondern es gibt nur eine einzige Variante der Unreal Engine. Je nachdem, was du mit der Engine gestalten willst, ändert sich auch die Lizenz für dich, aber jedem Nutzer, sei es Privatperson, Schüler, Indie-Entwickler oder große AAA-Studios, steht die gleiche Engine zur Verfügung.

Solltest du dich dafür entscheiden, ein kommerzielles Produkt auf dem Markt zu bringen, das du mit der Unreal Engine 4 erstellt hast, musst du 5 % deiner Einnahmen an Epic Games abgeben, aber auch nur, wenn die Einnahmen in einem Quartal mehr als 3000 $ betragen. Solltest du also 2500 $ in einem Quartal verdienen, darfst du alles behalten. Solltest du 3100 $ verdienen, musst du von den 3100 $ an Epic Games 5 % abgeben. Genauere Informationen hierzu findest du auf Englisch un*ter www.unrealengine.com/*faq.

Eventuell schreckt dich nun der Gedanke an die Abgaben ein wenig davon ab, die Unreal Engine 4 zu verwenden, aber bedenke, dass es sich hierbei um eine der besten Engines auf dem Markt handelt, mit der du die gleichen Optionen hast, ein Spiel zu entwickeln wie die großen Spieleentwickler, die ebenfalls die Unreal Engine 4 nutzen. Es gibt keinerlei Restriktionen, und du hast exakt das gleiche Tool wie alle anderen auch. Wenn du also das neue Unreal Tournament von Epic Games ansiehst, denk daran: Das kannst du auch.

Zusätzliche Inhalte, die ich im Beispielprojekt von Kapitel 22 erstellt habe, darfst du für dich nach Belieben in deinen eigenen Projekten nutzen, egal ob kommerziell oder nicht.

## 1.4 Weiterentwicklung der Engine

Es ist üblich, dass sich Engines im Laufe der Zeit weiterentwickeln und neue Versionen erscheinen. Bei der Unreal Engine 4 kommt es alle ein bis drei Monate zu einer neuen Version, welche immer Verbesserungen und neue Features mit sich bringt. Als ich mit dem Buch anfing, war gerade einmal die Version 4.7 veröffentlicht worden, und mittlerweile steht Version 4.17 in den Startlöchern. Bis du diese Zeilen liest, hat sich eventuell sogar noch viel mehr geändert, und neue Features, von denen ich noch gar nichts wusste, sind erschienen, und ich hätte diese sicherlich gerne in diesem Buch beschrieben.

Du darfst dich also nicht wundern, wenn bei meinen Videos und Bildern nicht alles hundertprozentig so aussieht wie bei dir – das lässt sich leider nicht verhindern. In der Regel verändert sich jedoch das Aussehen nur geringfügig, und die jeweiligen Elemente sind oft nur einen kleinen Klick entfernt. Auch wenn ich nun schreibe, dass Feature X/Y noch nicht in der Engine oder in Blueprints verfügbar ist, kann es sein, dass es bei dir mittlerweile ganz anders aussieht und du dir denkst: Was hat der Typ denn für einen Unsinn geschrieben … das ist doch alles hier!

Änderungen sind auf jeden Fall immer von Vorteil und helfen dir bei der Entwicklung, aber es kann auch hin und wieder vorkommen, dass mit einer neuen Engine Version einige deiner Blueprints aufhören zu funktionieren. Dies geschieht in den seltensten aller Fälle, und ich selbst habe es bisher auch nur ein oder zweimal erlebt, aber es kann dennoch passieren. Dann musst du genau nachschauen, an welcher Stelle das Blueprint aufhört zu funktionieren, und dein Blueprint dahingehend anpassen. Dabei kann es sich auch um etwas ganz Banales handeln, wenn z. B. die Verknüpfung zweier Funktionen gekappt wurde. Aber manchmal erfordert das Beheben solcher Probleme ein wenig mehr Vorwissen. Für den Anfang rate ich dir deswegen auch, auf einer Engine-Version zu bleiben, bis du dich mit deinen Fähigkeiten wohl genug fühlst, ehe du dir die neueste Version herunterlädst. Wenn du aus diesem Buch lernst, welches hauptsächlich mit der Version 4.8 – 4.9 erstellt wurde, wirst du dich auch in den Versionen am besten zurechtfinden können. Wie bereits erwähnt, kann ich nicht garantieren, dass sich zukünftige Versionen optisch und funktionell nicht verändern. Mit der zweiten Auflage habe ich jedoch versucht alle Änderungen bis 4.16 zu berücksichtigen und Bilder gegebenenfalls zu aktualisieren.

Aber ich will dir jetzt auch nicht zu viel Angst machen, dass du gar nicht mehr mit der Entwicklung mithalten wirst, wenn du zum ersten Mal damit arbeitest. Aber wenn du etwas nicht auf Anhieb findest, haben sich möglicherweise nur der Ort oder die Bezeichnung ein wenig verändert.

Für alle Fälle werde ich auf meinem YouTube-Kanal immer wieder Updates veröffentlichen und neue Videos herausbringen, um dich auf Änderungen aufmerksam zu machen und dich so weiterhin bei der Entwicklung deines Spiels zu begleiten.

# 2 Erste Schritte

Ich werde nicht nur über die Entwicklung mit der Unreal Engine 4 reden, sondern dir auch ein bisschen von der Philosophie bei der Spieleentwicklung ans Herz legen. Es gibt extrem viel zu beachten, und das alleinige Wissen über Mechaniken wird dich nicht überall hinbringen und garantiert kein erfolgreiches und spaßiges Spiel. Klar muss ich dir jetzt nicht erklären, dass es nicht so toll ist, wenn dein Spiel voller Fehler ist und dein Charakter andauernd durch den Boden fällt. Es geht mir vielmehr darum, dass du immer genau weißt, warum du etwas Bestimmtes machst und welchen Nutzen du und der Spieler daraus erhalten. Das Wichtigste ist, dass du Spaß an der Entwicklung hast und der Gamer Spaß am Spielen, dann sind alle Beteiligten glücklich und somit hoffentlich erfolgreich.

## 2.1 Wie fange ich an?

Es kommt immer darauf an, welches Spiel du machen willst und welche Erfahrung du mitbringst. Der logische Anfang wäre natürlich, dieses Buch zu lesen, denn einfach ohne jedwedes Vorwissen draufloszulegen, wird dir nur Probleme bereiten.

Es lohnt sich nicht, gleich von Anfang an dein Spiel der Träume zu bauen. Setze dir vielmehr zuerst kleine Ziele. Wenn du gerade das erste Mal mit der Engine anfängst und denkst: „Geil, dann mach ich jetzt mein Ultra-MMO mit zerstörbarer Landschaft und einem Minimum von 1000 Stunden Spielzeit“, dann hast du dir ein wenig zu viel vorgenommen. Fang immer klein an, vor allem wenn du noch am Lernen bist. Stelle dir kleine Aufgaben, die du nacheinander erfüllst. Beispielsweise könnten deine ersten Aufgaben darin bestehen, kleine Features wie das Öffnen und Schließen einer Tür oder das Ein- und Ausschalten mehrerer Lichter in einem Raum mit einem Lichtschalter einzubauen. Wie genau du sowas angehst, wirst du im Laufe dieses Buches noch lernen, aber das meine ich mit „klein anfangen“. Wenn du dann irgendwann soweit bist und alle Grundlagen verstanden hast, kannst du anfangen, kleinere Spiele zu bauen.

Neben der Idee brauchst du noch einen gewissen Plan, wie du die Aufgabe angehen willst. Einfach drauflos zu bauen, kann funktionieren, ist aber nicht empfehlenswert. Du musst dir genaue Gedanken machen, wie du deine Aufgabe angehen willst und was du dafür machen

musst. Bleiben wir bei dem simplen Beispiel, einen Lichtschalter für Räume einzubauen. Du benötigst ein Lichtschalter-Objekt, Lichter und die Logik, um diese Lichter ein- und ausschalten zu können. Für die Funktion brauchst du eine Möglichkeit, dass dein Charakter mit dem Lichtschalter kommunizieren kann. Der Lichtschalter muss mit den jeweiligen Lichtern kommunizieren können und wissen, ob er die Lichter an- oder ausschalten soll usw.

Liegen all diese Fragen auf dem Tisch, liegt es an Dir, das Puzzle zusammenzufügen und herauszufinden, wie du die einzelnen Elemente am besten verbinden kannst. Dies kann vor allem bei komplexen Aufgaben und Funktionen helfen.

## 2.2 Motivation

Motivation ist ein wichtiger Faktor bei der Spieleentwicklung. Ich habe schon oft erlebt, wie mangelnde Motivation ein Projekt zum Scheitern gebracht hat. Es gibt ein paar Tricks, wie du deine Motivation aufrechterhalten kannst.

Zum einen ist Abwechslung ein guter Faktor. Wenn du mehrere Wochen an einem neuen und innovativen Feature sitzt, es aber einfach nicht so funktionieren will, wie du gerne hättest, dann lass es erstmal. Such dir eine andere Aufgabe, bring dich auf andere Gedanken und mach deinen Kopf frei. Wenn man sich zu sehr auf etwas versteift, gibt es nur Probleme. Kommst du nach einer kleinen Auszeit wieder an diese Aufgabe zurück, hast du eine vollkommen andere Sicht auf das Geschehen. Der Tunnelblick ist verschwunden, und du hast neue Ideen, dein Problem zu lösen.

Finde Teammitglieder, die genauso motiviert sind wie du! Alleine zu arbeiten kann auch Unmengen an Spaß machen – vor allem, weil man sein eigener Herr ist, aber auf Dauer ist es doch schön, andere Teammitglieder zu haben, als alles alleine zu erledigen. Aber auch hier musst du vorsichtig sein, denn Teammitglieder können auch eine große Motivationsbremse sein. Wenn Aufgaben nicht erledigt werden oder an dir hängen bleiben, keine Kommunikation herrscht und jeder macht, was er will, dann ist das mehr als nervig. Versuche also, deine Teammitglieder gut auszuwählen, und nicht nur nach deren Talent, sondern eher anhand ihrer Motivation. Jeder hat die Möglichkeiten, sein Talent zu verbessern, aber nicht jeder kann seine Motivation für längere Zeit halten.

Nimm dir nicht zu viel vor! Wenn du dir selbst direkt einen Berg an Aufgaben gibst, kannst du schnell die Übersicht und Motivation verlieren. Kleine Schritte führen zum Erfolg!

## 2.3 Planung

Wenn du in einem Team arbeitest, ist es wichtig, dass du und deine Teammitglieder einen Plan haben. Nicht nur, was für Aufgaben momentan erledigt werden sollen, sondern auch, welche Aufgaben schon erledigt sind und wer welche Aufgabe zugewiesen hat. Somit hast

du immer einen genauen Überblick über den momentanen Entwicklungsstand und kannst sehen, an welcher Stelle es hakt und wo es gut läuft. Dafür steht dir auch eine große Bandbreite an Hilfsmitteln zur Verfügung, um deine Planung übersichtlich darzustellen.

Zum einen gibt es das kostenlose Web-Tool namens **Trello**. Auf **Trello** können neue Karten in Kategorien erzeugt und bei Beendigung in eine andere Kategorie verschoben werden. Von der Kategorie 3D-Models kann die Aufgabe Charakter erstellen in die Kategorie Erledigt bugsiert werden. Wie genau deine Kategorien heißen und wie du diese managst, bleibt dir natürlich komplett offen.

Für eine professionellere, nicht kostenlose, aber relativ günstige Möglichkeit gibt es noch **Unfuddle**. Ich persönlich benutze **Unfuddle** ebenfalls. Hier hat man die Möglichkeit, übersichtliche Tickets zu erstellen und diese Personen im Team zuteilen zu können. Sobald ein Teammitglied einen Bug findet, kann dieser dazu ein neues Ticket erstellen und es einer Person zuweisen. Dann erhalte ich auch direkt eine Mail mit dem Ticket und kann sehen, dass es einen Fehler beim Inventarsystem gibt, durch den man Gegenstände nicht mehr ablegen kann. Dann kann ich das Ticket akzeptieren, das Problem lösen und auf z. B. Testing stellen. Die anderen Mitglieder können das natürlich auch sehen, testen und bestätigen, dass der Fehler erfolgreich behoben wurde. Dann wird das Ticket archiviert, und man macht wie gewohnt weiter. Das nur als kleines Beispiel wie man die ganze Geschichte handhaben kann. Zusätzlich kann man dort noch Milestones einstellen (Zeitangaben, wann bestimmte Features fertig sein sollen), damit man ein Ziel hat, auf das hingearbeitet wird.

Es gibt natürlich unzählig verschiedene Seiten und Programme, die du benutzen kannst, und es ist egal, ob du das mit einem tollen Programm erstellst oder alles in einer eigenen Excel-Tabelle verwaltest. Die Hauptsache ist jedoch, dass du einen Überblick und richtige Planung über die Entwicklung hast.

## 2.4 Sicherheitskopien

Es kann immer zu Problemen kommen, die dein komplettes Projekt lahmlegen können. Sei es, dass du dein eigenes Projekt nach einem Absturz nicht mehr öffnen kannst, oder deine Festplatte den Geist aufgibt: Möglichkeiten, dein Projekt zu zerschießen, gibt es viele. Deswegen ist es immens wichtig, dass du Sicherheitskopien anlegst – am besten auf einen separaten Server.

Mithilfe von Tools wie **SVN** oder **Git** kannst du dein Projekt gesichert auf deinem eigenen Server lagern und anderen Personen Zugriff auf eben dieses Projekt geben. Jeder kann sich somit das Projekt herunterladen, Änderungen durchführen und diese auf den Server hochladen. Sollte also dein PC in die Luft fliegen, kannst du unbesorgt sein (außer natürlich du sitzt mitten in der Explosion), da deine Daten sicher sind und nichts verloren gegangen ist. Sollte es aber auch passieren, das du aus Versehen etwas hochlädst, was dein Projekt unbrauchbar machst, hast du mit **SVN** und **Git** die tolle Möglichkeit, ein paar Versionen zurückzugehen, oder nur bestimmte Dateien auf einen früheren Stand zurückzusetzen.

Im optimalen Fall musst du natürlich nicht darauf zurückgreifen, und alles läuft von Anfang an reibungslos, aber es ist immer sehr hilfreich zu wissen, dass deine Daten sicher sind.

Falls dann doch irgendwann mal was passieren sollte, kannst du ruhig bleiben und musst dich nicht unnötig ärgern, deine ganze Arbeit verloren zu haben.

## 2.5 Learning by Doing

Eines der wichtigsten Aspekte ist das eigenständige Lernen und Ausprobieren. Ich kann dir nicht alles genau vorkauen und beibringen, dafür ist die Engine viel zu groß und vielseitig, um alles innerhalb eines Buches durchgehen zu können. Als ich anfing, mich mit der Engine auseinanderzusetzen, gab es wenige bis gar keine Hilfen und Tutorials, und ich musste mir mein Wissen selbst aneignen, indem ich einfach diverse Sachen ausprobierte und herumexperimentierte. Für mich ist das eine der effektivsten Formen des Lernens, wenn man einfach mal versucht, eigenständig ohne Hilfen Aufgaben zu erledigen. Dies ist definitiv nicht der angenehmste und schnellste Weg, aber im Endeffekt hast du mehr davon, da du die einzelnen Versuche direkt verinnerlichst. Durch das eigene Scheitern lernst du aus deinen Fehlern und bist somit besser gewappnet für die Zukunft, als wenn du alles einfach nur kopierst, ohne dir eigenständig Gedanken zu machen.

Klar, ein bisschen Hilfe kann man immer gebrauchen, vor allem am Anfang. Mir geht es aber darum, dass du nicht sofort aufgibst, sobald du vor einer Hürde stehst, und in einem Forum nach einer schnellen Lösung suchst, sondern zuerst für dich selbst versuchst, eine Lösung zu finden. Auch wenn du dann mehrere Tage an einem an sich einfachen Problem sitzt, macht gerade das Lösen eines Problems am meisten Spaß, wenn du dies eigenständig geleistet hast.

Und zur Not, wenn es gar nicht weitergeht, kann man sich immer noch von anderen helfen lassen.

# 3 Grundlagen

In diesem Kapitel werde ich dir eine kleine Übersicht über die Engine geben – von der Installation bis zur Oberfläche und Bedienung. Viele in der Übersicht vorgestellten Features werden in späteren Kapiteln näher vorgestellt. Wundere dich also nicht, wenn du nicht alles auf Anhieb verstehst; im Laufe des Buches wird dir alles nach und nach beigebracht.

## 3.1 Installation

Zuallererst installieren wir natürlich die Engine selbst. Zum Herunterladen gehst du auf *www.unrealengine.com* und klickst oben rechts auf den Button **Get Unreal**.


**Bild 3.1**
Die Schaltfläche zur Registrierung

In dem Fenster, das sich nun öffnet, gibst du deine Daten für die Registrierung ein.

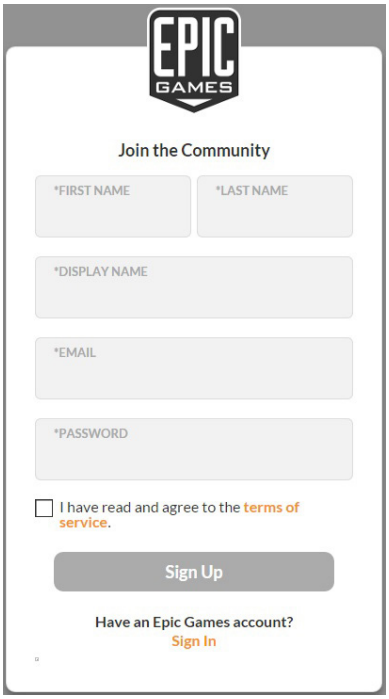


**Bild 3.2**
Hier musst du deine Daten eingeben

Du brauchst nur deinen Vor- und Nachnamen anzugeben, den du mit einem *Display Name* verknüpfst. In der Engine selbst wird dir dieser Name dann angezeigt, und damit kannst du von anderen gefunden werden und im Engine Launcher mit anderen chatten. Dieses komplett optionale Feature brauchst du nicht unbedingt, aber es kann ganz nützlich werden, wenn du neue Leute aus dem Unreal Engine-Forum hinzufügst. Mich kannst du gerne unter dem Namen *Nobody* hinzufügen.

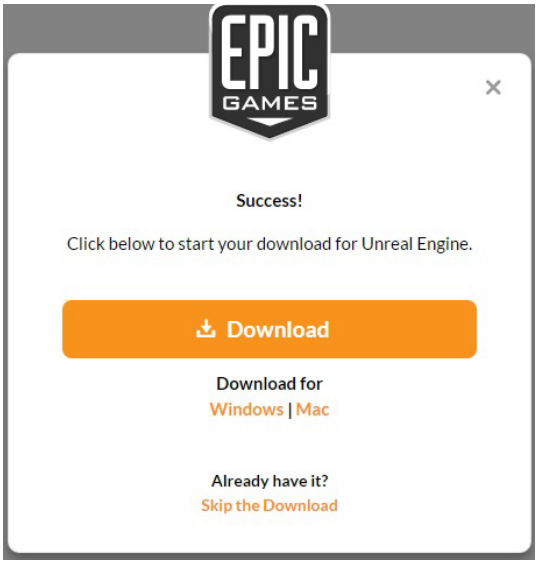


**Bild 3.3**
Einfach auf **Windows** klicken oder auf **Mac**, wenn du einen Mac hast

Für die weitere Registrierung musst du natürlich noch deine E-Mail-Adresse sowie ein dazugehöriges Passwort angeben, womit du dich dann auf *www.unrealengine.com* sowie der Engine selbst anmeldest. Sobald du dich erfolgreich registriert hast, wirst du auch schon direkt zur Download-Seite weitergeleitet.

Ist alles fertig heruntergeladen, muss die Engine nur noch installiert werden, die Installation ist ziemlich selbsterklärend.

Sobald die Installation abgeschlossen ist, öffnen wir den **Epic Games Launcher**, der für die Benutzung der Engine immer zuerst geöffnet werden muss. Dort einfach mit den vorher angelegten Daten anmelden, und schon bist du erfolgreich im Launcher angemeldet. Herzlichen Glückwunsch!

## 3.2 Epic Games Launcher

Der Epic Games Launcher ist die Schnittstelle für das Verwalten deiner Engine-Versionen und des Unreal Marketplace.

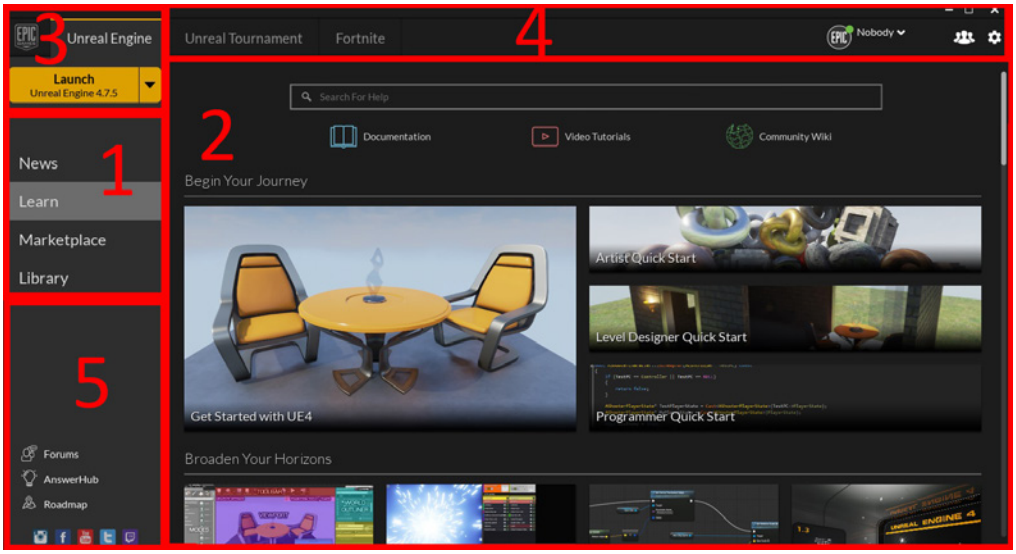


**Bild 3.4** Die Oberfläche des Launchers

Hier ist eine grobe Aufteilung der Oberfläche des Epic Games Launchers:

1. **Menüpunkte:** Hier geht es zu aktuellen News sowie zu vielen Lerninhalten von Epic mit einigen Beispielprojekten zum Herunterladen und Ausprobieren.
2. **Hauptansicht:** In der Hauptansicht siehst du alle Inhalte des jeweiligen ausgewählten Menüpunkts.
3. **Engine-Auswahl:** Du kannst verschiedene Projekte mit verschiedenen Engine-Versionen am Laufen haben und die jeweilige Engine-Version hier öffnen.

4. **Extras:** Hier kannst du dir die Projekte von Epic Games anschauen und herunterladen, z. B. *Unreal Tournament* oder *Fortnite*. Weiter rechts hast du deine Optionen und Freundesliste, wo du mit anderen Personen, die die Unreal Engine benutzen, kommunizieren kannst.
5. **Social Media:** In diesem Bereich bekommst du Zugang zu allen Social Media-Seiten für die Unreal Engine. Insbesondere das Forum ist ziemlich interessant und gewährt dir Einblicke in die aktuellen Vorkommnisse der Engine.

Der Launcher ist ein wichtiger Bestandteil der Engine. Unter dem **Learn**-Bereich gibt es viele kostenlose Beispielprojekte von Epic, die du herunterladen und anschauen kannst. Falls du also etwas siehst, was dich interessiert, wie z. B. *Inventory UI in UMG*, so kannst du das Projekt herunterladen, Schritt für Schritt nachschauen, wie alle Inhalte implementiert wurden, und für das eigene Projekt nachbauen oder abändern.

Im **Marketplace**-Bereich gibt es viele qualitativ hochwertige Produkte für dein Projekt: Texturen, 3D-Modelle, 2D-Modelle, Sounds, Blueprints, Animationen und Effekte. Im Gegensatz zu anderen sogenannten *Asset Stores* werden hier alle Inhalte gründlich getestet, und nur die qualitativ hochwertigsten Produkte schaffen es überhaupt in den Marketplace.

In der **Library**-Sektion kannst du die neuesten Engine-Versionen downloaden, alte Engine-Versionen löschen, alle deine Projekte und alle gekauften Marketplace-Produkte verwalten. Hier sollte immer auf die jeweilige Engine-Version geachtet werden. In den meisten Fällen, insbesondere bei Texturen oder 3D-Modellen, sollte es zu keinen Problemen kommen, aber bei Blueprints kann es immer passieren, dass von einer zu einer anderen Engine-Version Elemente abgeändert wurden, sodass die Blueprints nicht mehr richtig funktionieren. Das passiert aber nur in den seltensten Fällen.

## 3.3 Erstellung eines Projekts

Nachdem wir uns nun um den ganzen organisatorischen Teil gekümmert haben, wird es Zeit, sich die eigentliche Engine anzuschauen. Dafür müssen wir zuallererst ein eigenes Projekt erstellen. Sobald du dir bei **Library** eine Engine-Version deiner Wahl ausgesucht und sie gestartet hast, kannst du direkt dein erstes Projekt erstellen.

**HINWEIS:** Vor Erscheinen einer neuen Engine-Version gibt es immer mehrere *Preview*-Versionen. Es ist nicht zu empfehlen, diese Versionen für die aktive Entwicklung zu verwenden. Die Möglichkeit besteht, dass dein Projekt dabei beschädigt werden kann. Deswegen warte lieber auf ein *Full Release,* bevor du mit deinem Projekt zu einer neueren Engine-Version wechselst.

Wie du sehen kannst, hast du die Option zwischen zwei Arten von Projekten.

**C++-Projekte** kommen mit einigen vordefinierten C++-Dateien für dein Projekt. Ein neuer Ordner namens **Source** wird angelegt, wo du alle deine C++-Dateien findest. In deinem Projektordner findest du die üblichen Solution-Dateien, die mit Microsoft Visual Studio

geöffnet werden müssen. **Visual Studio Community** kannst du kostenlos herunterladen: *https://www.visualstudio.com/de-de/downloads/download-visual-studio-vs#DownloadFamilies_2*. Die Engine selbst ändert sich aber nicht sonderlich beim Öffnen eines C++-Projekts. In dieser Art Projekt kannst du jederzeit neuen C++-Code hinzufügen und abändern, aber auch ganz normal mit Blueprints arbeiten, ohne jemals C++-Code hinzufügen zu müssen.

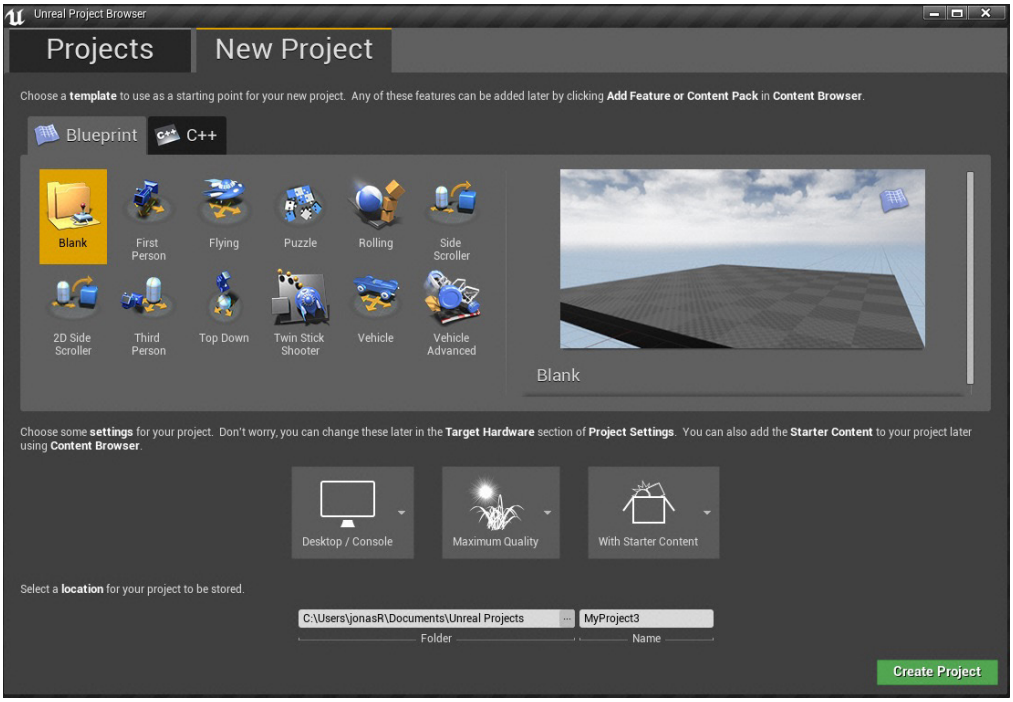


**Bild 3.5** Bei der Projekterstellung hast du eine Auswahl an Basisfunktionen

**Blueprint-Projekte** kommen komplett ohne eine Zeile C++-Code aus, und hier wird auch nirgendwo C++-Code angelegt. Da wir uns in diesem Buch hauptsächlich mit Blueprints auseinandersetzen, empfehle ich insbesondere für Anfänger, ein *Blueprint-Projekt* anzulegen.

Nachdem du dich für eine Art entschieden hast, kannst du wählen, ob du ein *Blank*-Projekt anlegen oder mit einigen der vordefinierten Optionen anfangen willst, die du aber jederzeit auch später in dein Projekt einfügen kannst. Bei diesen Optionen geht es hauptsächlich um die Steuerung von vordefinierten Charakteren. Du kannst also z. B. einen *Third Person*-Charakter (Schulter-Ansicht) in ein *Vehicle* einsteigen lassen, womit du von einer Steuerung auf die nächste wechseln könntest. Alternativ kannst du natürlich immer alles komplett selbst erstellen, aber insbesondere für den Anfang ist es immer gut, sich anzuschauen, wie es funktionieren *kann*. Wie in der Spieleentwicklung üblich, gibt es immer mehrere Wege zum Ziel.

Hast du dich entschieden, müssen auch noch die **Zielplattform** und **Qualität** ausgewählt werden. Auch hast du die Option, **Starter Content** hinzuzufügen, womit dir einige Texturen und Materials sowie einige Basis-**Assets** *(3D-Modelle)* zur Verfügung gestellt werden, aber

es bleibt dir selbst überlassen, die vorgefertigten Assets oder lieber deine eigenen zu benutzen.

Zu guter Letzt musst du nur noch deinen Projektnamen ändern, und schon ist dein Projekt fertig erstellt, und die Engine-Oberfläche wird mit deinen Einstellungen geladen. Dieser Schritt sollte nicht lange dauern.

## 3.4 Oberfläche

Die Oberfläche der Unreal Engine 4 ist vollkommen frei anpassbar und kann nach deinem Belieben verändert werden, so wie du dich damit am wohlsten fühlst. Es *kann* vorkommen, dass mit einer neueren Engine-Version auch die Oberfläche verändert wird und nicht alles exakt an der Stelle zu finden ist, wie hier im Buch dargestellt, aber die Veränderungen sind in der Regel eher klein und leicht zu entdecken.

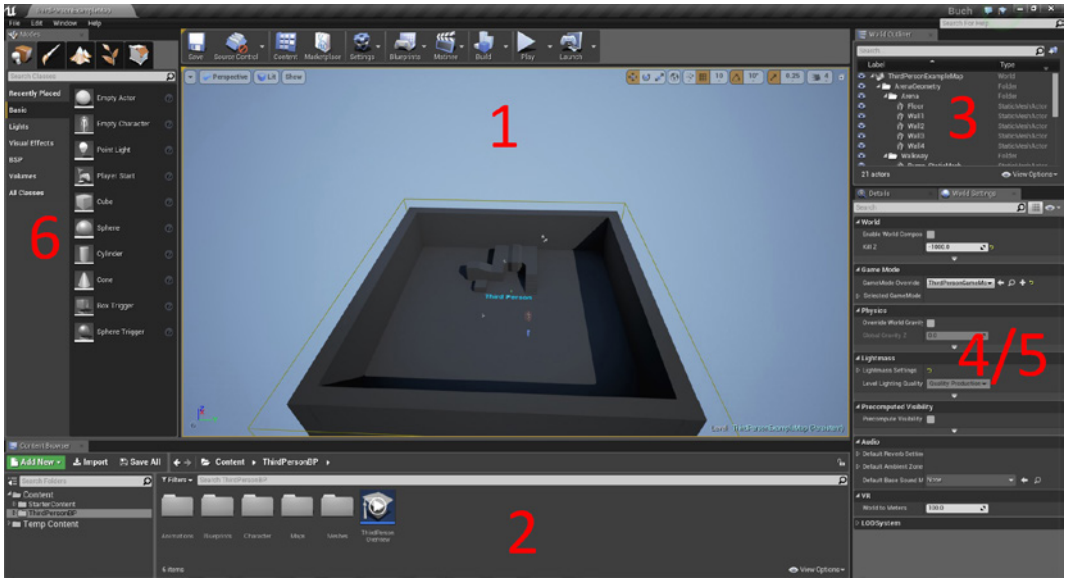


**Bild 3.6** Die Default-Ansicht der Oberfläche mit Third-Person-Template

Beim ersten Start der Engine sollte die Oberfläche so wie in Bild 3.6 aussehen.

1. **Game/Editor View:** In dieser Ansicht bewegst du dich durch dein Level und platzierst Gegenstände in der Welt. Dabei gibt es noch einige wichtige und nützliche Hilfen, die ich aber später erklären werde.
2. **Content Browser:** Hier werden alle deine Assets/Blueprints angezeigt und verwaltet. Erstellst du neue Blueprints oder willst du etwas Neues hinzufügen, wird der Content Browser benutzt.

3. **World Outliner:** Alle Assets, die in dem momentan geladenen Level vorkommen, werden in diesem Bereich aufgelistet. Mithilfe von Ordnern bewahrst du die Übersicht.
4. **Details:** Hast du ein Objekt im Editor View oder World Outliner ausgewählt, hast du die Möglichkeit, einige Optionen an dem jeweiligen Objekt in diesem Bereich einzusehen und zu verändern.
5. **World Settings:** Hier kannst du einige Optionen spezifisch für die jeweilige Map verändern.
6. **Modes:** Hier finden sich einige hilfreiche Level-spezifische Sachen: von Licht, Volumes bis hin zu Level-Design-Tools wie *BSP*. Was genau *BSP* ist, werde ich dir in *Kapitel 9* erklären.

Wie anfangs in diesem Abschnitt erwähnt, ist die gesamte Oberfläche frei anpassbar. Willst du beispielsweise einen größeren *Game/Editor View* haben, könntest du den *Content Browser* unter den *Modes* anheften.

## 3.4.1 Game/Editor View

Der Game/Editor View ist einer der wichtigsten Bereiche, die du für dein Projekt brauchst. Du siehst nicht nur genau, wie dein Level aufgebaut ist, sondern bekommst einige enorm hilfreiche Optionen, die dir das Bauen und Betrachten deines Levels erleichtern.

### 3.4.1.1 Viewport

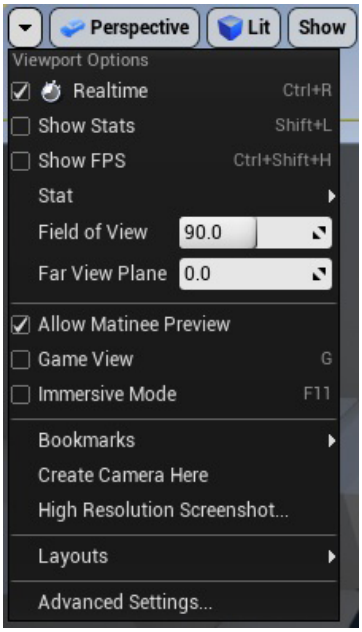


**Bild 3.7**
Die *Viewport*-Optionen

Wenn du auf den kleinen Pfeil auf der oberen linken Seite des Game/Editor Views gehst, bekommst du einige Optionen, mit denen du mehr Informationen über deinen momentanen

Level bekommst. Zum Beispiel kannst du mit **Show FPS** die sogenannten *Frames Per Seconds* anzeigen lassen, die dir einen kleinen Einblick geben, wie stark das momentan Gezeigte an der Performance zerrt.

**HINWEIS:** FPS sind die Anzahl an Bildern, die pro Sekunde auf dem Bildschirm angezeigt werden können. Bei niedriger FPS beginnt das Spiel zu ruckeln. Deswegen ist es ratsam, das eigene Spiel so zu optimieren, dass mindestens 30, bestenfalls 60 FPS herauskommen (bei einem PC-/Konsolenspiel). Die FPS-Zahl im Editor spiegelt aber nur eine ungefähre Zahl wider, die in deinem fertigen Spiel durchaus höher ausfallen kann. Deswegen immer schön testen!

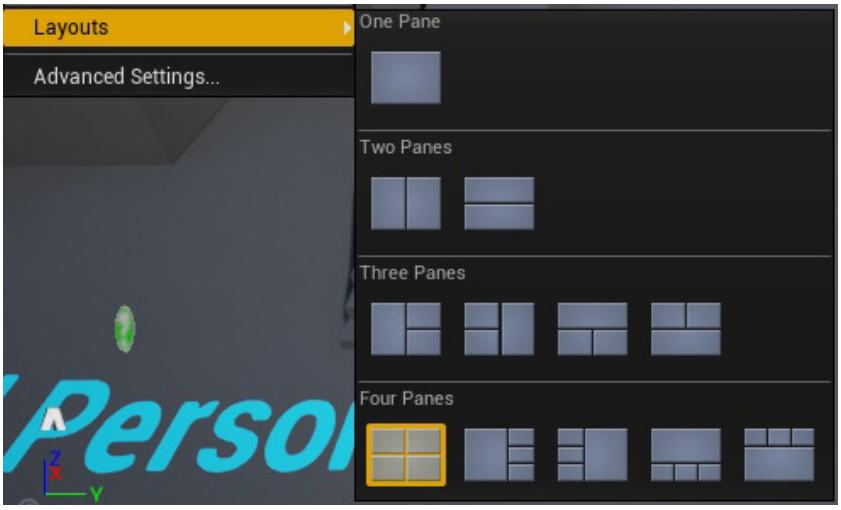


**Bild 3.8** Einige vorgefertigte Layouts

Unter **Layouts** gibt es einige vorgefertigte Ansichten für deinen Editor. Ganz nach deinen Präferenzen kannst du dich durch die verschiedenen Layouts klicken, bis du das Layout gefunden hast, was dir am meisten zusagt.

Auf die anderen Viewport-Optionen werde ich nicht weiter eingehen. Es lohnt sich aber definitiv, sich mal die Zeit zu nehmen und sie eigenständig durchzuklicken. Viele der Optionen (insbesondere unter *Stat*) sind Hardware-fokussiert und bringen dir tiefe Einblicke in dein Spiel, die du sonst nicht sehen würdest. Im späteren Verlauf der Entwicklung können dir diese Statistiken Aufschluss über momentane Geschehnisse auf dem Monitor und in deinem Level geben. Vor allem, wenn du Performance-Probleme bekommst und nicht genau weißt, woran das liegt, erhältst du einen Einblick, wie lange jeder Prozess braucht.

Für den Anfang brauchst du dir keine Sorgen zu machen, aber behalte einfach mal im Hinterkopf, dass es diese Option gibt. Ob du sie jemals nutzen wirst, liegt dann ganz an dir und wie dein Projekt läuft.

Auf der Registerkarte **Perspective** kannst du deine Sicht auf das Geschehen ändern, um z. B. genau von oben auf dein Level zu schauen. In der Regel ist die Standardperspektive jedoch vollkommen ausreichend.

### 3.4.1.2 View Mode


**Bild 3.9**
Alle View-Mode-Optionen

Im **View Mode** kannst du die Darstellungsweise des Game/Editor Views verändern, was dir einige wichtige und interessante Informationen bietet und dir auch bei der Gestaltung deines Levels behilflich sein kann. Arbeitest du an einer dunklen Höhle mit wenig Licht, wäre es sinnvoll, während des Bearbeitens auf *Unlit* zu stellen. Damit wird alles gleichmäßig hell, was für die Arbeit an dunklen Abschnitten unabdingbar ist.

Weiterhin hilfreich ist die Ansicht **Player Collision/Visibility Collision**. Um ein angenehmes Spielgefühl zu gewährleisten, sollte es nicht vorkommen, dass Kollisionen (wo der Spieler nicht hindurch kann) im Weg stehen oder gar nicht vorhanden sind. Wenn du also merkst, dass dein Charakter immer durch eine Brücke fällt, könnte es auch einfach daran liegen, dass die Brücke noch keine Kollision hat.

**PRAXISTIPP:** Es hilft sehr, sich einige der Optionen anzuschauen, damit du ungefähr weißt, wie diese aussehen. Klicke dich durch die einzelnen Views und schau dir an, was sich verändert. Bei manchen Optionen wirst du anfangs noch nichts erkennen können, die gehören dann sicher zu den fortgeschrittenen Ansichten.

Unter dem Bereich **Show** bekommst du weitere Optionen für die Darstellung deines Levels in Bezug auf die vorhandenen Objekte in deinem Level. Du kannst also z. B. bei Bedarf alle *Static Meshes* deaktivieren lassen, um deine Performance zu verbessern, während du an einer anderen Stelle an der Landschaft arbeitest. Es gibt unzählige verschiedene Szenarien, wo die eine oder andere Option hilfreich sein kann.

### 3.4.1.3 Snapping


**Bild 3.10** Optionen für Snapping und Selecting

Auch auf der oberen rechten Seite des Game/Editor Views sind einige sehr hilfreiche Optionen.

Die ersten drei Bilder zeigen die Möglichkeiten auf, die du hast, um deine ausgewählten Objekte zu **bewegen**, **rotieren** und **skalieren** *(W, E, R)*. Mit dem Drücken der *Leertaste* kannst du zwischen den Optionen wechseln.

Die anderen Optionen sind sogenannte Snapping-Optionen. Damit kannst du ein Objekt immer um eine gewisse Anzahl Stufen verschieben, rotieren oder skalieren. Dazu mehr in *Kapitel 6*.

### 3.4.1.4 Navigation

Kommen wir nun zum wesentlichen Bestandteil der Fortbewegung im Game/Editor View.

| Navigation | |
|---|---|
| **Perspective Viewport** | |
| Move forward / backward | LMB + Drag up / down |
| Rotate left / right | LMB + Drag left / right |
| Free Rotate | RMB + Drag |
| Move up / down | LMB + RMB + Drag |
| Zoom in / out | Mouse Scroll Wheel |
| **Top / Front / Side Viewport** | |
| Pan | RMB + Drag |
| Zoom in / out | LMB + RMB + Drag<br>or<br>Mouse Scroll Wheel |
| **Selection** | |
| Select | LMB on Actor |
| Toggle selection | Ctrl + LMB on Actor |
| Marquee Selection | LMB + Drag |
| Clear Selection | Esc |
| **Focusing** | |
| Focus selected object | F |
| LMB = Left Mouse Button<br>RMB = Right Mouse Button | |

**Bild 3.11**
Übersicht der Möglichkeiten zur *Navigation*

Wie auf der Übersicht zu sehen, gibt es einige Fortbewegungsarten, die du zum Navigieren verwenden kannst. Im Folgenden siehst du die gleiche Übersicht wie in Bild 3.11, nur ins Deutsche übersetzt:

Im **Perspective**-View (Standardansicht *ALT* + *G*):

- **Vorwärts/Rückwärts bewegen:** Linke Maustaste gedrückt halten *(LMT)* und nach oben und unten bewegen.
- **Rechts und links rotieren:** *LMT* gedrückt halten und nach rechts und links bewegen.
- **Freies Rotieren:** Rechte Maustaste gedrückt halten *(RMT)* und in gewünschte Richtung bewegen.
- **Hoch und runter bewegen:** *LMT* und *RMT* gedrückt halten und bewegen.
- **Hinein- und Hinauszoomen:** *Mausrad* benutzen.

**Alternativ:** *RMT* gedrückt halten und mit den **WASD**- oder **Pfeiltasten** bewegen.

In der **Seitenansicht:**

- **Bewegen:** *RMT* gedrückt halten und bewegen.
- **Herein- und Herauszoomen:** *Mausrad* benutzen oder *RMT* + *LMT* gedrückt halten und bewegen.

**Selektion:**

- **Auswählen:** *LMT* auf Objekt klicken.
- **Mehrfachauswahl:** *LMT* + *STRG* auf Objekte klicken.
- **Auswahl aufheben:** *Esc* Taste drücken.
- **Objekt fokussieren:** *F* beim ausgewählten Objekt drücken.

Das ist die grundlegende Navigation, die du kennen solltest. Solltest du dich noch fragen, wie du jetzt genau ein selektiertes Objekt bewegst, dann versuch einfach mal, ein Objekt anzuklicken und mithilfe von **W/E/R** und der LMT auf die angezeigten Symbole das Objekt zu verändern. Spiele ein bisschen damit herum. Falls es noch nicht ganz so flüssig klappt: keine Sorge, in *Kapitel 6* werde ich nochmal genauer auf das Thema zurückkommen.

### 3.4.2 Content Browser

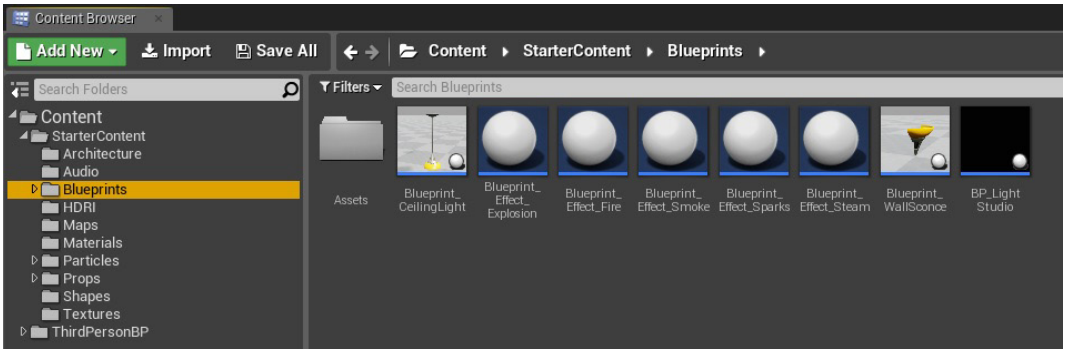


**Bild 3.12** Der Content Browser für all deine Assets

Der **Content Browser** ist der Ort, an dem alle deine Assets für das Spiel aufbewahrt werden. Die Struktur ist gleichzusetzen mit der eigentlichen Ordnerstruktur deines Betriebssystems. Jedoch ist es nicht möglich, Assets, wie z. B. 3D-Modelle, einfach so in deinen Windows-Ordner zu packen, damit diese in deinem Projekt erscheinen. Du musst also beim Einfügen neuer Assets immer auf *Import* drücken oder diese mithilfe von *Drag & Drop* direkt in den **Content Browser** verschieben.

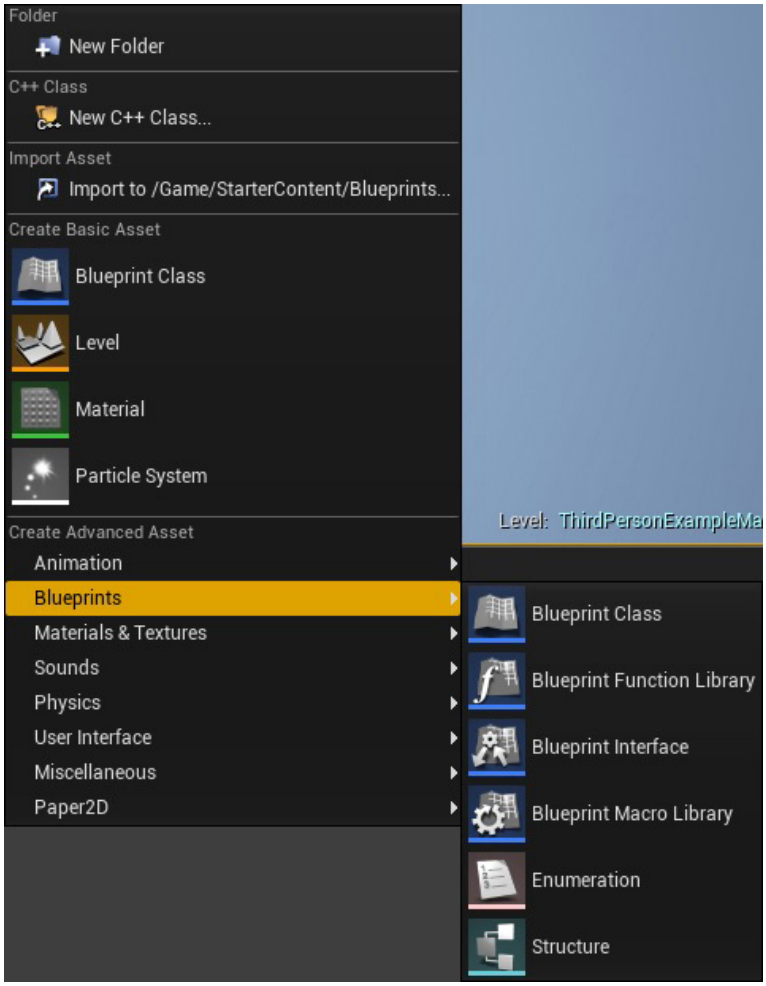


**Bild 3.13** Alle Optionen für das Erstellen verschiedener Assets

Mit einem einfachen Klick auf **Add New** oder **Rechtsklick** im Content Browser erhältst du eine Liste an Optionen für das Erstellen neuer Inhalte, aufgeteilt in einige Unterbereiche, wie du in Bild 3.13 sehen kannst. Das wirst du während der Entwicklung ziemlich oft benutzen müssen, um neue Assets/Blueprints erstellen zu können. Auf die einzelnen Bereiche werde ich später noch einmal zurückkommen.

### 3.4.3 World Outliner

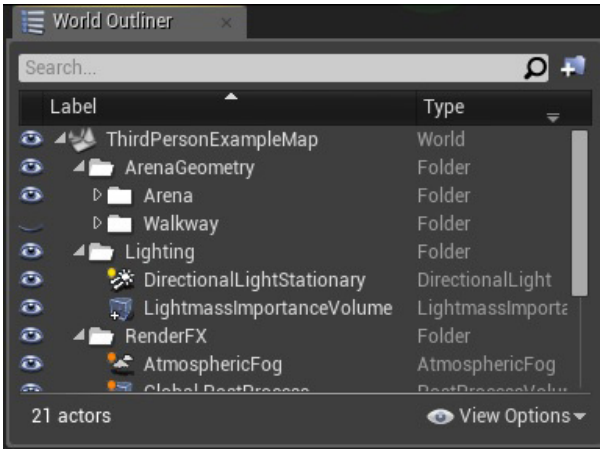


**Bild 3.14**
Der World Outliner

Zum World Outliner gibt es nicht viel zu sagen. Hier wird alles aufgelistet, was nur irgendwie in deinem Level vorkommt. Je nachdem, wie gut du dich um eine Ordnung darin gekümmert hast, hast du auch einige Ordner für bestimmte *Assets/Lichter/Areale*. Du kannst jederzeit einzelne Assets oder ganze Ordner mit einem einfachen Klick auf das Auge deaktivieren. Es ist dir möglich, durch diese Deaktivierung beim Erstellen deines Levels Performance einzusparen und ohne FPS-Probleme effektiver zu arbeiten.

Ordner erstellst du mit einem Klick auf den kleinen Ordner + Bildchen rechts neben der Suchfunktion.

### 3.4.4 Details

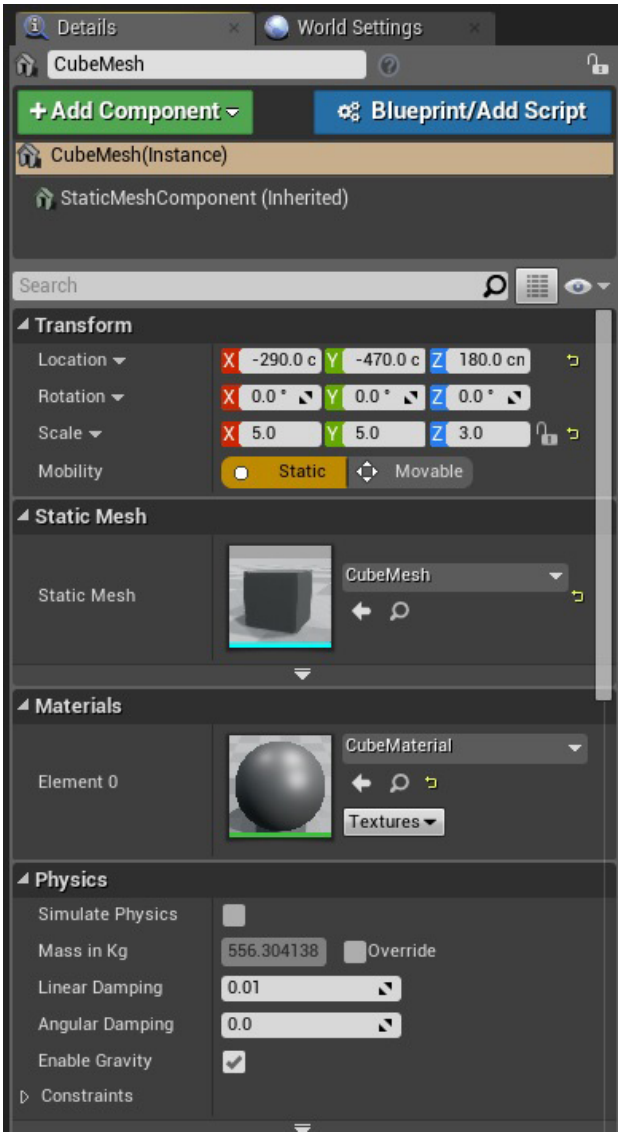


**Bild 3.15**
Der Details-Bereich für einen *„Static Mesh Actor“*

Im Details-Bereich gibt es alle Informationen und Einstellungsmöglichkeiten zu deinem ausgewählten Asset. Dieser Bereich ist für jedes Asset anders und bietet dir andere Möglichkeiten. Im Beispiel in Bild 3.15 geht es um ein **Static Mesh**. Ein *Static Mesh* ist nichts anderes als ein statisches 3D-Objekt, das du in deinem Level platzieren kannst (dazu wie über vieles andere später mehr).

Hier hast du nun u. a. die Möglichkeit, das Aussehen deines Objekts zu verändern, sei es das *Model* an sich oder die *Textur/Grafik* auf dem Objekt. Auch kannst du hier deinem Objekt direkt physikalische Eigenschaften geben, um damit zu interagieren, und vieles mehr.

### 3.4.5 World Settings

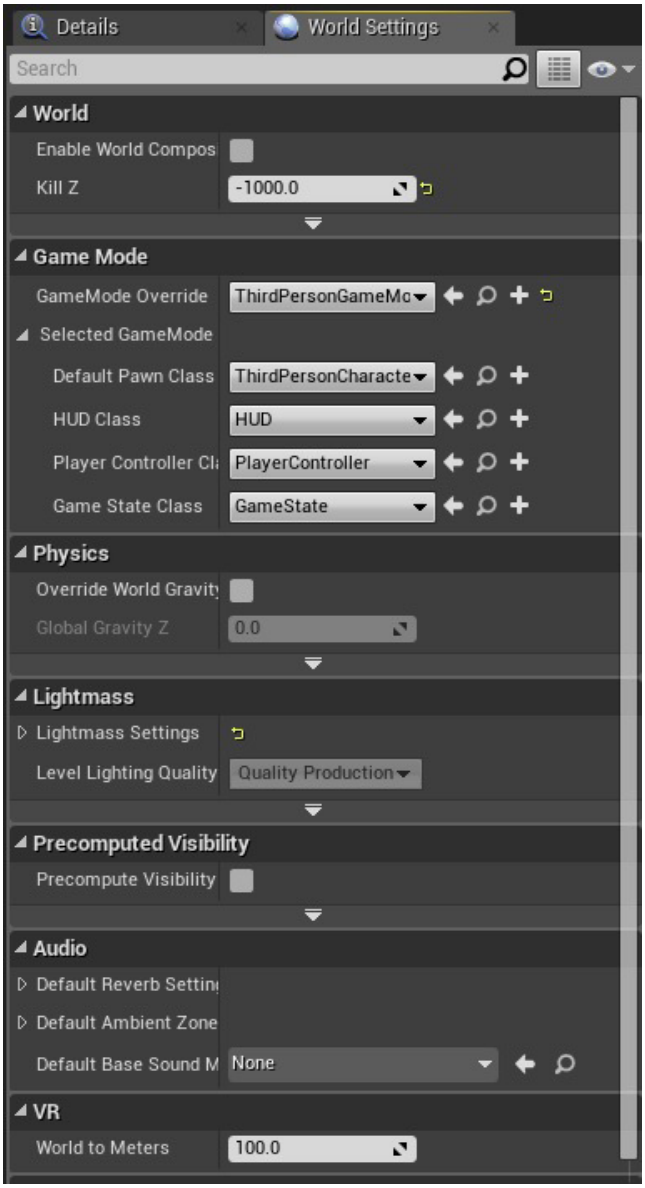


**Bild 3.16**
Ein paar Einstellungen, die für dein ganzes Level gelten

In World Settings werden wichtige Eigenschaften für dein Level festgelegt. Solltest du diesen Bereich *nicht* sehen, so kannst du ganz oben links auf **Window** und dort auf **World Settings** klicken, und schon ist das Fenster da.

Zu sehen gibt es einige interessante Aspekte für dein Level. **Kill Z** ist beispielsweise die *Tiefe*, in der jedes Objekt sofort zerstört wird. Fällt aus irgendwelchen Gründen ein physikalischer Gegenstand aus deinem Level, fliegt dieser also nicht für immer in die Tiefe, sondern wird aus Performance-Gründen gelöscht.

**Game Mode** ist der für den Anfang wichtigste Bereich und besteht aus fünf sogenannten *Blueprints*. Was genau ein *Blueprint* ist, wird im nachfolgenden Kapitel ausführlich erklärt, aber ich gebe dir erst einmal eine grobe Einsicht in die *Game Modes*.

- **GameMode Override:** Nichts anderes als eine Datei, die die vier untergeordneten Einstellungen beinhaltet. Veränderst du eine der untergeordneten Blueprints, wird das automatisch in der Game-Mode-Datei gespeichert und gilt für alle Level.
- **Default Pawn Class:** Der Charakter, mit dem der Spieler anfängt.
- **Hud Class:** Das **HUD** (Head Up Display) ist ein Anzeigesystem, womit Grafiken auf deinen Bildschirm projiziert werden, die beispielsweise das Fadenkreuz oder die Lebensanzeige deines Spielers darstellen.
- **Player Controller Class:** Der Player Controller ist eine Datei, die deinen Spieler „kontrolliert". Mehr dazu später ☺.
- **Game State Class:** Hier gibt es hilfreiche Optionen und Informationen für dein Spiel. Es werden hier beispielsweise alle Multiplayer-Spieler gespeichert und können abgerufen werden.

### 3.4.6 Modes

**Modes** ist ein sehr umfangreicher Bereich und unterteilt in die fünf Abschnitte **Place**, **Paint**, **Landscape**, **Foliage** und **Geometry Editing**. Ich werde nur recht kurz auf einige dieser Modes eingehen, da diese Teil eines späteren Tutorials sind, aber du kannst dir jetzt dennoch eine kleine Übersicht dazu ansehen, damit du ungefähr weißt, was dich erwartet.

### 3.4.6.1 Place

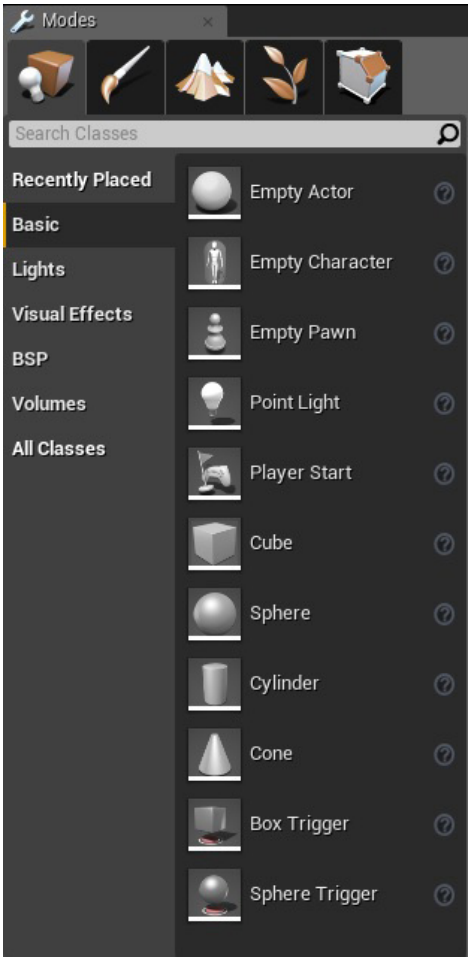


**Bild 3.17**
Eine Übersicht der Place-Optionen

**Place** steht schlicht und einfach für *Platzieren*. Eingeteilt in mehrere Kategorien hast du hier die Möglichkeit, alle Objekte zu platzieren, die du *nicht* unbedingt im *Content Browser* findest. Wenn du einen **Player Start**-Punkt platzieren willst (die Position, wo der Spieler anfängt), reicht ein einfaches Ziehen und Ablegen in dein Level, und schon ist ein Startpunkt platziert.

- **Basic:** Einfache Basisobjekte, die häufig benutzt werden.
- **Lights:** Die komplette Auswahl an zur Verfügung stehenden Lichtern.
- **Visual Effects:** Bestimmte visuelle Effekte wie z. B. Nebel.
- **BSP:** *(Geometry Brush Actors)* Einfache geometrische Formen, die zur schnellen Levelgestaltung genutzt werden können.

- **Volumes:** Volumen, die bestimmte Eigenschaften verändern können, sobald der Spieler diese betritt. So kann z. B. ein *PhysikVolume* für einen Raum erstellt werden, das die Gravitation verändert, sodass sich der Spieler darin schwerelos bewegen kann.
- **All Classes:** In diesem Bereich gibt es *alles*: alle Inhalte des *Content Browsers* und auch sonst alle zur Verfügung stehenden Objekte.

Ich könnte jetzt weitere hundert Beispiele nennen, für die verschiedensten Anwendungsgebiete der einzelnen Objekte, aber im Laufe des Buches wirst du dir ein immer besseres Bild machen können.

### 3.4.6.2 Paint

Wie der Name andeutet, geht es hier ums Malen (Painten). Was erst einmal relativ simpel klingt, ist in Wirklichkeit aber ganz schön kompliziert.

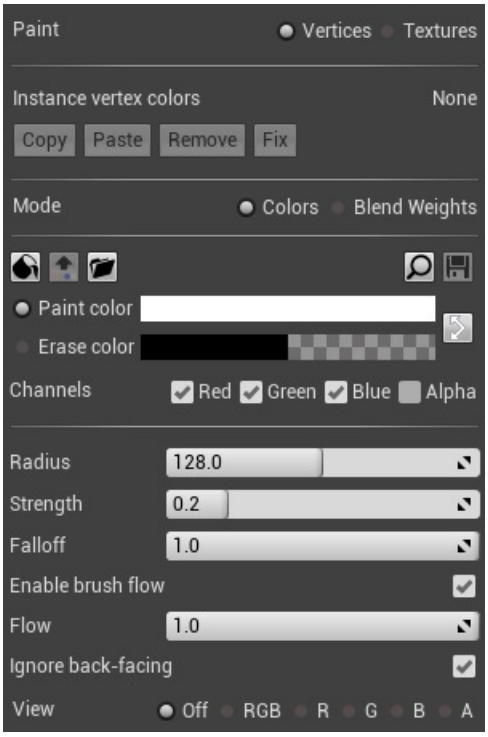


**Bild 3.18**
Sieht einfach aus? Ist es aber nicht.

Dieser Bereich ist eher für fortgeschrittene Benutzer, und ich werde nicht wirklich näher darauf eingehen. Im Download-Bereich kannst du dir dazu ein Video von mir anschauen, in dem ich die Nutzung demonstriere. Du solltest dir aber erst das Video anschauen, nachdem du *Kapitel 6* gelesen hast und dich etwas mehr mit **Materials** auskennst.

### 3.4.6.3 Landscape

Eine Landschaft gehört in fast jedes Spiel – seien es blumige Wiesen, prächtige Berge oder dunkle Wälder.

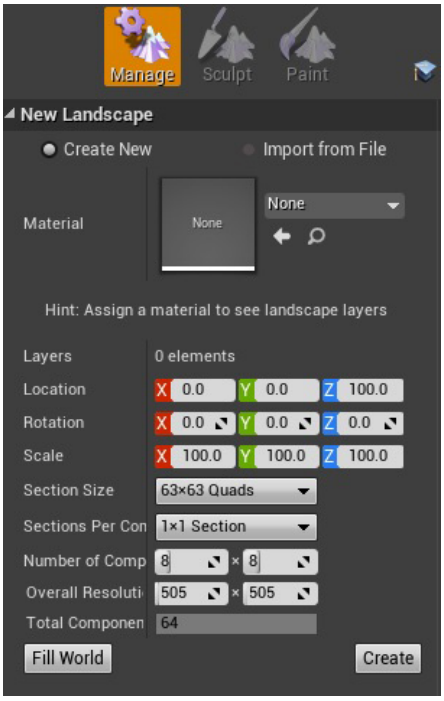


**Bild 3.19**
Die ersten Landscape-Einstellungen

Landscapes brauchen ein eigenes Kapitel, weswegen du mehr davon in *Kapitel 10* erfahren wirst. Du darfst dich schon einmal auf einige schöne Sachen dort freuen!

### 3.4.6.4 Foliage

In Foliage kannst du alle deine Bäume und Pflanzen in die Welt malen und organisieren. Seit Unreal Engine 4.8 ist der Foliage Mode deutlich einfacher und besser zu benutzen. Intensiver werden wir uns mit dem Tool in *Kapitel 10* befassen.

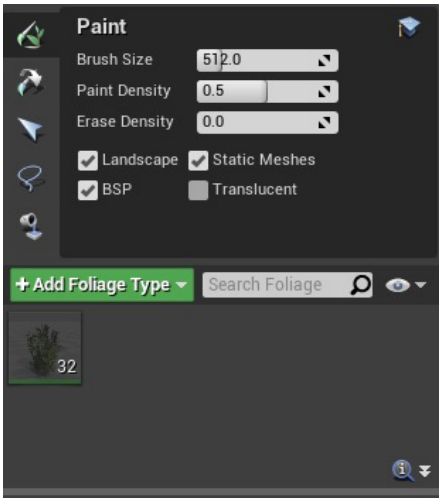


**Bild 3.20**
Foliage Mode mit einem Bush als Beispiel

### 3.4.6.5 Geometry Editing

Geometry Editing hat mit den Unreal-internen geometrischen Formen zu tun. Diese kannst du verändern und bearbeiten, wie du es eventuell von einem 3D-Programm kennst.

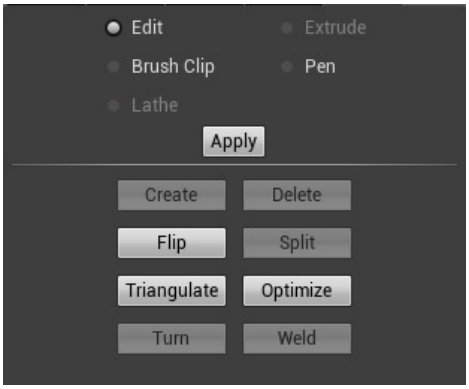


**Bild 3.21**
Einige der Optionen zum Verändern der Geometrie

In *Kapitel 9* werde ich intensiver auf das Thema eingehen und dir mehr zu diesem Tool erzählen. Prinzipiell kannst du es dir wie einfachste Geometriemanipulation vorstellen, wo du eine Wand größer/kleiner machen oder eine Ecke entfernen kannst. Dies funktioniert aber nur mit den *BSP*.

## 3.4.7 Play

Mit einem einfachen Klick auf den *Play*-Knopf kommst du direkt ins Spielgeschehen. Im Regelfall öffnet sich das Spiel im Game/Editor View, dies kannst du jedoch mit dem Pfeil neben dem *Play*-Knopf ändern.

- **Selected Viewport:** Spiel wird im *Game/Editor View* ausgeführt.
- **Mobile Preview:** Öffnet ein *neues Fenster*, in dem die Handy-Steuerung benutzt werden kann.
- **New Editor Window:** Ein *neues Fenster* öffnet sich, in dem dein Spiel ausgeführt wird.
- **VR Preview:** Hast du ein virtuelles Headset, wie z. B. *Oculus Rift*, angeschlossen, öffnet sich das Spiel direkt auf dessen Display, ohne dass deine anderen Monitore davon betroffen sind.
- **Standalone Game:** Hierbei öffnet sich auch ein neues Fenster, aber im Gegensatz zum *New Editor Window* ist diese Sicht näher am späteren fertigen Spiel.
- **Simulate:** Im *Game/Editor View* wird das Spiel ausgeführt, aber in diesem Fall bist du kein interaktiver Bestandteil, sondern schaust nur zu, wie sich alles abspielt.

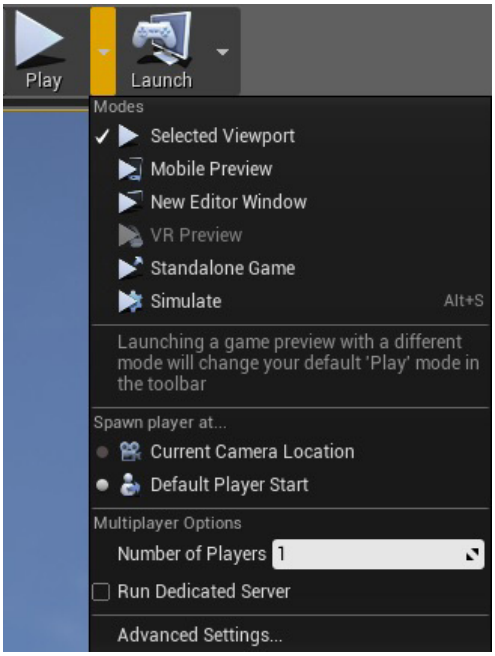


**Bild 3.22**
Die Optionen zum Spielen

Weiter unten erhältst du noch die Option, mehrere Spieler einzustellen, um eventuell deinen **Multiplayer**-Modus testen zu können. Sinnvoll dabei wäre, *New Editor Window* aktiviert zu haben, damit sich für jeden Spieler ein neues Fenster öffnet.

## 3.5 Ausprobieren

Kommen wir nun zur ersten kleinen „Aufgabe“.

Diese Aufgabe ist eher *optional,* aber sehr zu empfehlen, wenn du dich noch nie mit einer Engine auseinandergesetzt hast.

**ÜBUNG:** Nachdem du nun ein bisschen über die Oberfläche gelernt hast, wird es an der Zeit, dich damit mehr zu befassen, damit du dich im Umgang mit der Engine wohler fühlst und weißt, wo was zu finden ist. Ich will, dass du einfach mal ein bisschen herumspielst und neue Objekte erstellst, in dein Level platzierst, dich herumbewegst, Licht platzierst und sonst alles machst, was du einfach mal gerne testen würdest.

Es geht noch nicht darum, dass du genau weißt, was du da machst, das kommt alles später.

**HINWEIS:** Falls du jemals einen Bereich oder ein Fenster geschlossen hast oder nicht mehr findest, kannst du dieses jederzeit wiederherstellen, indem du oberhalb des Editors oder Blueprints auf **window** klickst. Dort hast du eine Auswahl aller zur Verfügung stehenden Fenster, die du anzeigen oder verstecken kannst

Sollte es also passieren, dass du beispielsweise den Details-Bereich nicht mehr siehst, kannst du auf **Window** → **Details** klicken, um das jeweilige Fenster wiederherzustellen.

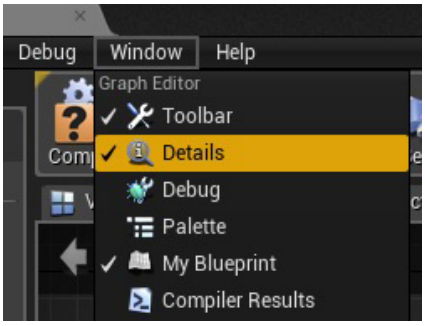


**Bild 3.23**
Window-Bereich zur Anzeige von Fenstern

Jeder gezeigte Bereich kann mit einem „×“ geschlossen oder verschoben werden, pass deshalb immer genau auf, worauf du klickst. Sollte jemals ein Fenster verschwinden, denke immer daran, dass du Zugriff auf jedes Fenster im Window-Bereich hast.

# 4 Blueprints

In den Kapiteln und bei deinen ersten eigenen Versuchen ist dir der Name *Blueprint* schon oft begegnet. In diesem Kapitel werden ich dir die Grundlagen und das Verständnis, was genau Blueprints sind, erklären. Aber dies ist nicht das einzige Kapitel über Blueprints, es ist nur der Anfang. Fast jedes Objekt ist auch ein Blueprint: dein Spieler, eine Tür, die man öffnen kann, oder die Steuerung deiner Animationen – alles wird mit Blueprints erledigt.

## 4.1 Was sind Blueprints?

Blueprints sind im Grunde nichts anderes als C++-Code-Klassen, nur dass man hierbei mit sogenannten **Nodes** arbeitet – anstatt sich durch viele Zeilen Code zu quälen, was besonders für Anfänger schnell unübersichtlich wird. Nodes kann man mit Bausteinen vergleichen. Man baut sich die gesamte Logik aus vielen verschiedenen Bausteinen zusammen. Will man einen Lichtschalter mit Licht einbauen, kann man sich die Logik so vorstellen:

Schalter wird betätigt → Ist das Licht an? → Wenn ja, ausschalten/Wenn nein, einschalten

Programmieren und Blueprints gehen nach genau diesem Prinzip vor: Einer Aktion folgt eine Reaktion. Aber im Gegensatz zum normalen Programmieren mit viel C/C++-Code setzt du die Blueprint-Bausteine ein, genau wie es die Logik vorgibt, mit einfachen Verbindungen zueinander. Ein Baustein im obigen Beispiel ist, dass der Schalter betätigt wird. Darauf folgt ein Baustein, der fragt, ob das Licht an oder aus ist, gefolgt von einem letzten Baustein, der das Licht dementsprechend verändert.

Klingt logisch, oder? Gehen wir noch einen Schritt zurück, und ich zeige dir ein konkretes Beispiel, wie man eine einfache Textausgabe macht. Dabei werde ich Blueprints mit einfachem C-Code vergleichen.

Wenn du dich schon ein wenig mit C auskennst, wird dir der kommende Code-Ausschnitt bekannt vorkommen. Wenn nicht, ist das aber nicht schlimm.

**Listing 4.1** Einfaches C-Beispiel

```
#include <stdio.h>

int main(void) {

printf(„Hallo Welt");

}
```

Hierbei handelt es sich um eine einfache Textausgabe. Auf dem Bildschirm wird der Text *Hallo Welt* angezeigt, und sonst passiert nichts. In der Unreal Engine 4 und Blueprints passiert das ganz ähnlich, aber hierbei braucht man, wie anfangs erwähnt, keinen Code.

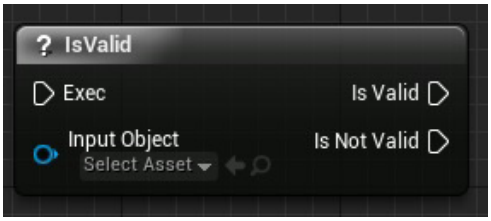


**Bild 4.1**
So sieht das Ganze in Blueprints aus

Wie man schnell sehen kann, gibt es ein sogenanntes *Event* namens *BeginPlay*. Existiert das jeweilige Blueprint in deinem Level, so wird anfangs immer das *Event* aufgerufen, ähnlich wie bei `int main` im C-Beispiel. Daraufhin wird mit *Print String* ein Text auf den Bildschirm ausgegeben, und genau wie im C-Beispiel wird die Welt mit „Hallo“ begrüßt.

Im direkten Vergleich kannst du sehen, wie man die Bausteine in Blueprints benutzt. Die Logik bleibt die gleiche. Mithilfe der Bausteine kannst du nahezu alles mit Blueprints programmieren, ohne dabei eine einzige Zeile Code zu schreiben. Aus eigener Erfahrung kann ich sagen, dass mir das Arbeiten mit Blueprints mehr Spaß macht als Code zu schreiben, und ich kann auch deutlich schneller arbeiten. Das ist natürlich alles Geschmackssache, aber ich hoffe, dass ich dich im Verlauf des Buches von den Vorteilen von Blueprints überzeugen kann.

## 4.2 Das Actor-Blueprint

Bevor wir uns tiefer in die Logik von Blueprints und der Programmierung stürzen, schauen wir uns erst einmal an, wie so ein Blueprint standardmäßig aussieht.

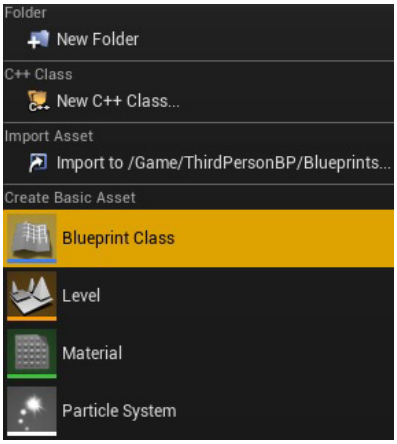


**Bild 4.2**
Der erste Schritt zum Erstellen eines Blueprints

Mit einem Rechtsklick im Content Browser in einem Ordner deiner Wahl kannst du eine Blueprint-Klasse erstellen: einfach auf *Blueprint Class* klicken, und schon öffnet sich ein neues Fenster.

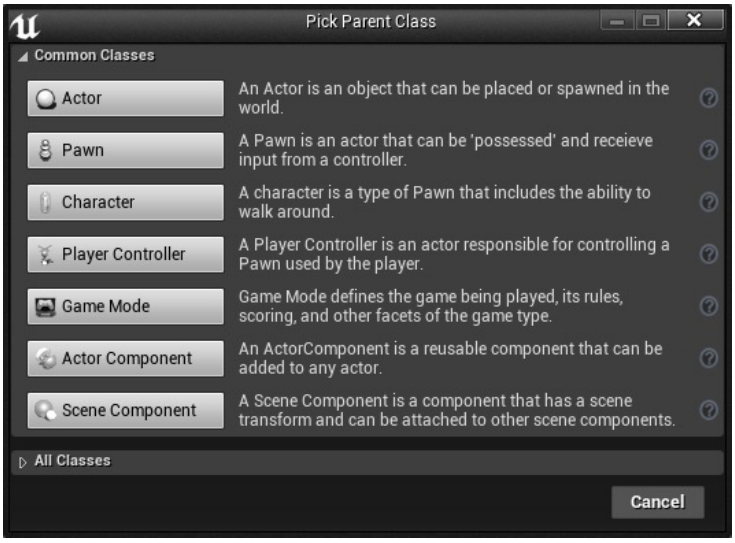


**Bild 4.3** Die gängigsten Blueprints-Klassen

Es gibt unzählige verschiedene Blueprint-Klassen mit verschiedenen einmaligen Funktionen. Wir kümmern uns erst einmal nur um das **Actor**-Blueprint. Diese Variante wirst du am meisten nutzen, um deinem Spiel Leben einzuhauchen. Deswegen erstellen wir auch erst einmal einen Actor.

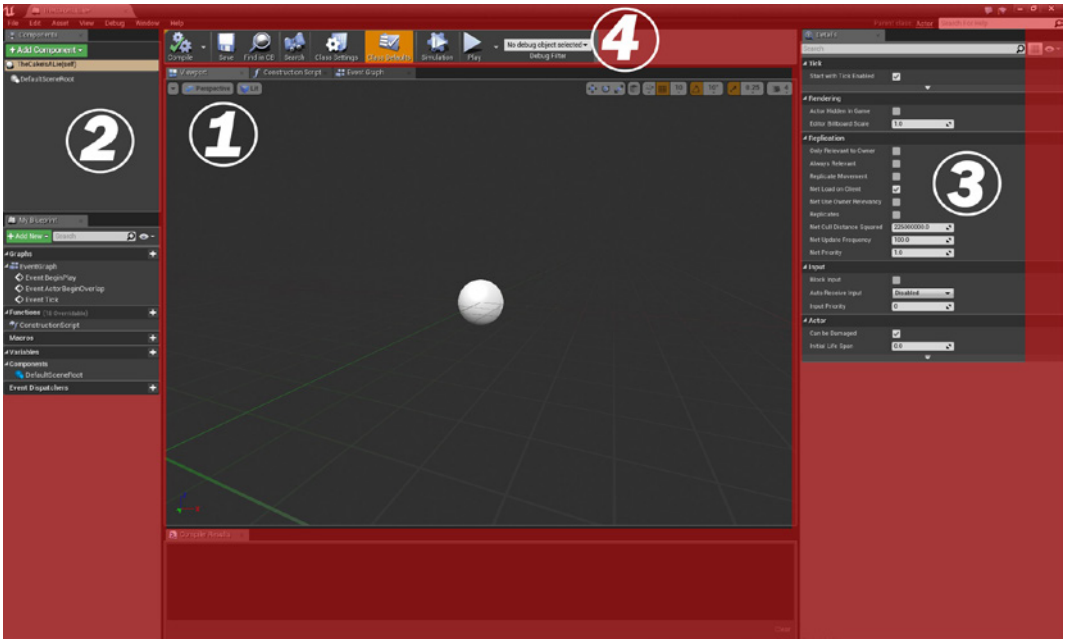


**Bild 4.4** Übersicht eines Actor-Blueprints

- **Bereich 1: Viewport/ConstructionScript/EventGraph:** In diesem Bereich wirst du dich die meiste Zeit aufhalten.
- **Bereich 2: Components:** Hier werden alle Objekte, Variablen, Funktionen und vieles mehr aufgeführt, die sich momentan in deinem Blueprint befinden. Auf Add Component können auch neue Inhalte deinem Blueprint hinzugefügt werden.
- **Bereich 3: Details:** Wie gewohnt, bekommt man im *Details-Bereich* alle Informationen zum momentan ausgewählten Objekt bzw. Variable oder, falls ausgewählt, einige Standardeigenschaften des Blueprints.
- **Bereich 4: Debug:** Einige Optionen zum sogenannten Debuggen, zur Suche und zum direkten Start des Spiels aus dem Blueprint heraus.

### 4.2.1 Der Hauptbereich

Da wir uns die meiste Zeit im Hauptbereich (vgl. Bild 4.4, Bereich 1) aufhalten werden, schauen wir uns diesen Bereich als Erstes an.


**Bild 4.5** Die drei wichtigsten Bereiche

Es gibt drei Unterbereiche, die alle sehr wichtig für den täglichen Gebrauch sind. Wir werden uns alle nacheinander grob anschauen:

- **Viewport:** Besteht unser Blueprint aus ein oder mehreren Objekten, kannst du sie hier verändern und so positionieren, wie du es für richtig hältst.
- **Construction Script:** Hier wird es ganz interessant. *Construction Script* ist eine Funktion, die immer ausgeführt wird, sobald sich das Blueprint in deinem Level verändert, während du im Editor bist. Beim Spielen selbst wird das Script nicht ausgeführt.
- **Event Graph:** Alle anderen Funktionen und der Kern der Blueprint-Programmierung befinden sich im *Event Graph*.

Nach dieser kleinen Übersicht kehren wir nun zum Blueprint-Programmieren zurück.

### 4.2.1.1 Viewport

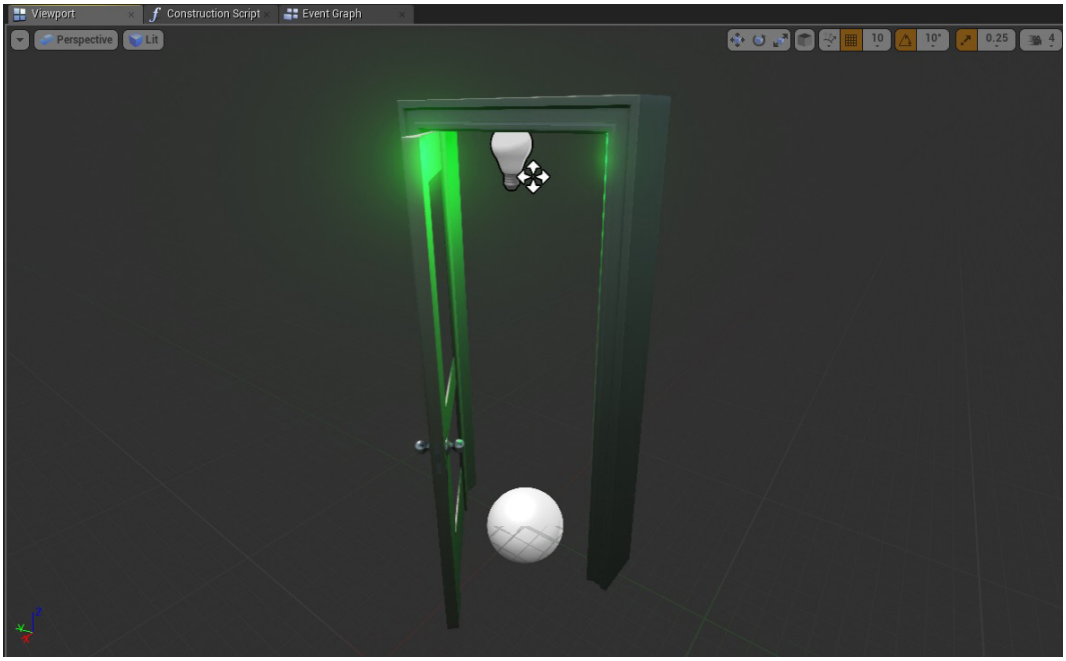


**Bild 4.6** Beispiel-Viewport eines Blueprints

Im **Viewport** kannst du dir alle Inhalte des Blueprints ansehen und nach Belieben ausrichten. Hierbei handelt es sich natürlich nur um Inhalte, die wir auch sehen können und die im Components-Bereich hinzugefügt wurden. Dazu später mehr.

Wie du sehen kannst, besteht dieses Viewport-Beispiel aus vier Komponenten bzw. Inhalten:

1. Scene Root
2. Türrahmen
3. Tür
4. Licht

Der **Scene Root** ist die große weiße Kugel und symbolisiert in der Regel den absoluten Null-Punkt im Blueprint, von dem der Rest der Komponenten ausgeht. Dieser existiert standardmäßig in jedem Actor-Blueprint, kann aber auch mit anderen Objekten ersetzt werden.

Türrahmen und Tür sind beides einfache statische Objekte, die ich im Blueprint platziert habe. Mit diesem Blueprint könnte ich jetzt die Funktion einbauen, diese Tür auf- und zumachen zu können. Das Tolle an Blueprints ist, dass du die Logik nur einmal herstellen musst und das Blueprint dann immer wieder verwenden kannst. Somit müsstest du die Tür nur einmal bauen und kannst sie verwenden, so oft du willst.

Das grüne Licht habe ich hinzugefügt, um zu zeigen, dass man wirklich alles in Blueprints einfügen kann. Wenn du möchtest, kannst du auch andere Blueprints hinzufügen oder Partikeleffekte, Sounds und vieles mehr.

#### 4.2.1.2 Construction Script

Das **Construction Script** ist im Prinzip ähnlich wie der Event Graph, hat jedoch ein paar Restriktionen. Dieser Bereich wird immer ausgeführt, wenn sich das Blueprint im Editor verändert, sei es die Umstellung einer Variablen oder gar das Bewegen des Blueprints in deinem Level. Wichtig zu wissen ist, dass dieser Bereich nicht beim Spielen selbst ausgeführt wird, sondern nur beim Erstellen, sprich beim Konstruieren des Levels. Es gibt einige nützliche Anwendungsbereiche für das **Construction Script**, aber dazu werden wir später in einem anderen Kapitel kommen.

#### 4.2.1.3 Event Graph – Was ist ein Event?

Beim Aufruf des Event Graphs werden dir standardmäßig drei sogenannte Events vorgegeben. Ein **Event** könnte man aus der Sicht eines Programmierers auch als eine Funktion ohne Rückgabewert betrachten. Für Nicht-Programmierer ist ein Event ein Punkt, von dem aus die Logik anfängt. Diese kann entweder durch ein bestimmtes Ereignis aufgerufen werden, oder man ruft das Event selbst auf.

Stellen wir uns ein Event als Knopf vor, so wird das Event mit allem, was dahintersteht, ausgeführt, wenn der Knopf gedrückt wird. Eine Funktion ist so ähnlich, wobei man im Gegensatz zu Events keinen automatischen Rückgabewert geben kann. Wenn ich also eine Funktion habe, in der ich überprüfen will, ob das Licht an oder aus ist, kann ich das Ergebnis als sogenannten Rückgabewert ausgeben. So könnte man in einem anderen Blueprint diese Funktion aufrufen und dort dann den Rückgabewert überprüfen, um zu wissen, ob das Licht an oder aus ist. Ein Event geht aber grundsätzlich nur in eine Richtung und kommuniziert nicht zurück von dem Ort, an dem es aufgerufen wurde. Zum Glück gibt es da aber dennoch Möglichkeiten, zurück zu kommunizieren. Das ist aber ein wenig komplizierter. Mehr zu diesen Funktionen später, kommen wir nun erst einmal wieder auf die Events zu sprechen.

### Die Standard-Events

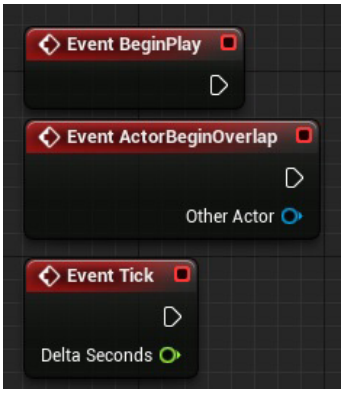


**Bild 4.7**
Die Standard-Events

- **Event BeginPlay:** Wird ausgeführt, sobald das Spiel mit dem Blueprint startet oder du das Blueprint während der Laufzeit erstellst.
- **Event ActorBeginOverlap:** Befindet sich in deinem Blueprint ein Objekt mit Kollision, wird das Event ausgelöst, sobald es ein anderes Objekt mit Kollision überlappt. Zu dem Event gehört, wie in Bild 4.7 zu sehen, der **Other Actor**. **Other Actor** bezieht sich auf das Blueprint, welches das Event ausgelöst hat, also sprich: wer das aktuelle Blueprint überlappt hat.
- **Event-Tick:** Wird bei jedem **Tick** ausgeführt. In der Unreal Engine 4 bezieht sich ein **Tick** auf die momentane *FPS*. Je performanter dein Spiel läuft, desto öfter wird das Event ausgeführt. Zu dem Event gehört, wie in Bild 4.7 zu sehen, die **Delta Seconds**. Delta Seconds bezieht sich auf die Zeit von einem zum anderen **Tick**.

**HINWEIS:** Event-Tick kann sehr nützlich sein, aber man muss sehr aufpassen, wann und wie man das Event benutzt. Ein paar dieser Events sind gleichzeitig verkraftbar, aber wenn du es mit den Event-Ticks übertreibst, kann die Performance in deinem Spiel schnell in den Keller wandern. Also immer schön aufpassen und wenn möglich, etwas anderes benutzen!

Es gibt noch eine Vielzahl an unterschiedlichen Events mit den unterschiedlichsten Funktionen, dies wäre aber definitiv zu viel für den Anfang. Wir werden uns in den kommenden Kapiteln nach und nach weitere Events anschauen, aber für den Anfang sollte das erst einmal reichen.

Aber anstelle nur von diesem Pool an Events abhängig zu sein, kannst du dir auch ein eigenes Event erstellen.

### Ein eigenes Event erstellen

Mit Rechtsklick im Event Graph öffnen wir das sogenannte *Context-Menü,* in dem alle verfügbaren Optionen von Nodes und Events, die man platzieren kann, dargestellt werden.

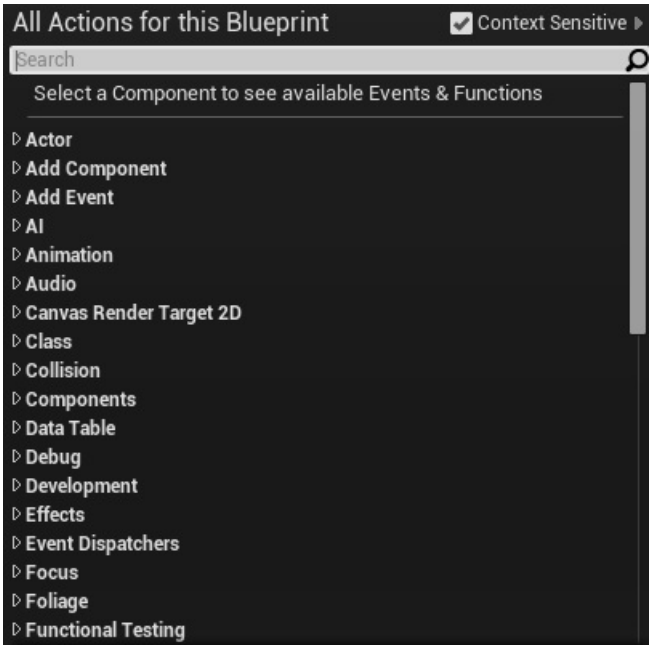


**Bild 4.8**
Das Context-Menü im Event Graph

Mit einer Suche nach *Add Custom Event* und anschließendem **Enter** oder mit einem Mausklick erstellst du dein eigenes *Custom Event,* dem du auch prompt einen Namen geben kannst.

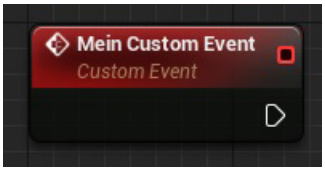


**Bild 4.9**
Ein wunderschönes Event

Mit unserem erstellten Event kannst du ein *Print String* hinzufügen, wie im eingangs erwähnten Beispiel in Bild 4.1 Einfach mit einem Mausklick auf den Pfeil in deinem Custom Event klicken, ziehen und loslassen. Damit kannst du eine neue **Node** erstellen, und beide werden sich automatisch nach der Erstellung verbinden.

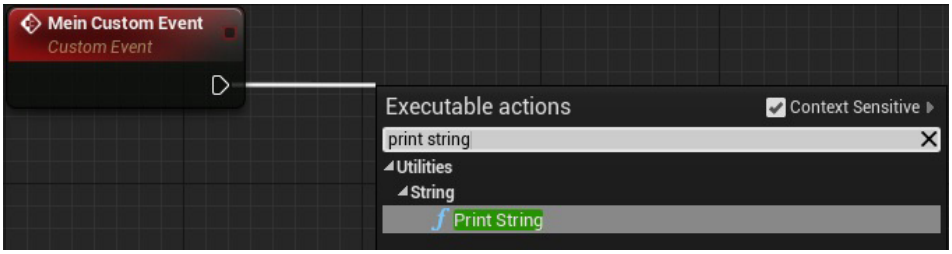


**Bild 4.10** So sollte es aussehen

Nachdem du nach *Print String* gesucht und es hinzugefügt hast, sieht der ganze Spaß dann wie folgt aus:

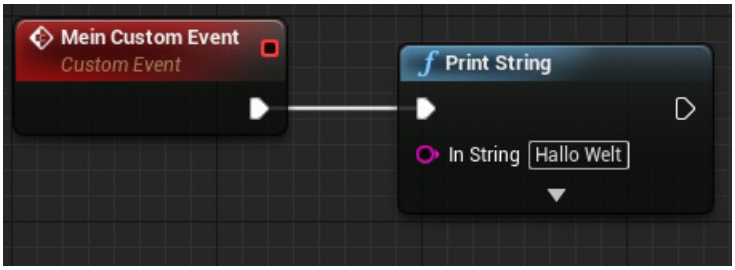


**Bild 4.11** Das erste funktionsfähige Custom Event

Sehr gut, nun haben wir ein eigenes Event, das beim Aufruf den Text *Hallo Welt* ausgibt. Aber das ist nun genau die Frage: Wie rufe ich mein eigenes Event auf? Ganz einfach, wir nehmen oder erstellen das **Event BeginPlay** und rufen unser Event in einer neuen **Node** auf. Dafür müssen wir von **Event BeginPlay** aus nach den Namen unseren erstellten Events suchen. Nur noch per Klick bestätigen, und schon haben wir ein Event, das ein anderes Event aufruft.

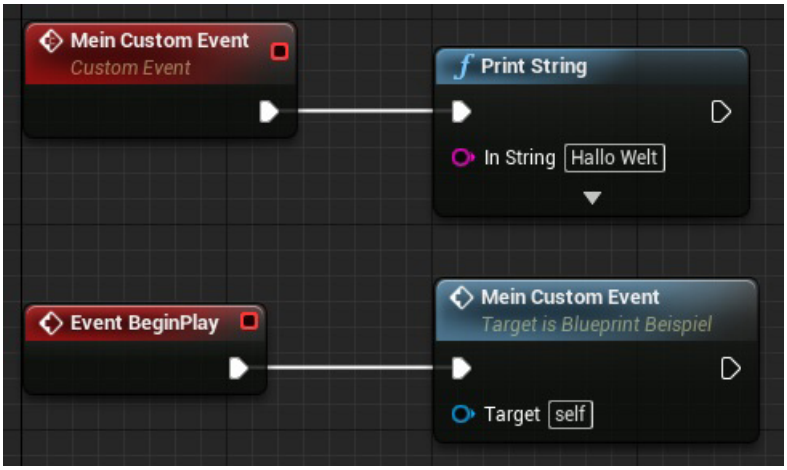


**Bild 4.12** Mein Custom Event wird beim Start ausgeführt

Als Erstes wird **Event BeginPlay** ausgeführt und geht direkt zur nächsten Node namens *Mein Custom Event*. Das **Target** (Ziel) ist, wie man sehen kann, das Blueprint selbst, zu erkennen am *self*. Man kann dementsprechend Events in anderen Blueprints ausführen, wenn einem das Ziel bekannt ist. In *Mein Custom Event* wird direkt **Print String** ausgeführt, das den Wert *Hallo Welt* ausgibt.

Platzieren wir unser Blueprint in das Spiel und drücken auf *Play,* wird also auf dem Bildschirm *Hallo Welt* ausgegeben. Vergleichbar wäre das in etwa mit dem C-Code in Listing 4.2.