

Unreal Engine 4.7

Eine Anleitung für Project Unik

Elham Nizam

21. August 2015

Inhaltsverzeichnis

1 Vorwort	2
2 Grundlegendes	3
2.1 Überblick der UI	3
2.2 Viewport Navigation	5
2.3 Orthographische Ansicht	6
2.4 View Modes und Show Flags	6
2.5 Objektplatzierung	7
2.6 Objekte bewegen	9
2.7 Objekte rotieren	11
2.8 Objekte skalieren	12
2.9 Bewegen mit der Kamera	12
2.10 Intro zum Content Browser	13
2.10.1 Assets über den Content Browser editieren	13
2.10.2 Navigation im Content Browser	13
3 Blueprints	15
3.1 Einführung einfacher Blueprints	15
3.1.1 Destructible	15
3.1.2 Camerashake	16
3.1.3 Timer	18
3.1.4 Slow Motion Effect	19
3.1.5 Jump Pad	21
3.1.6 Hover-Komponente	22
3.1.7 Simples Conveyor Volumen	22
3.1.8 Komponenten für Game-Behaviour	22



1 Vorwort

Hello alle zusammen,
anlässlich des Projekt Unik und dem Ziel ein marktfähiges Videospiel zu entwickeln, hab ich mich entschlossen, mich umfassend mit der Engine zu befassen und euch diesen Guide zu schreiben. Die meisten Infos und Anleitungen sind direkt den Video-Tutorials der Unreal-Homepage entnommen, einiges auch durch Eigenrecherche ergänzt. In den letzten Tagen kam auch wiederholt die Frage auf, ob Unreal so nützlich ist, wie anfangs gedacht, oder vielleicht doch zu schwierig für ein junges, unerfahrenes Team. Ich muss zugeben, die Frage beschäftigte mich eine ganze Weile. Generell, und das hab ich schon oft verlauten lassen, bin ich ein starkes Befürworter für Ambition. Es gibt jedoch eine feine Grenze zwischen Ambition und Größenwahn, deshalb hab ich während der Gamescom die Gelegenheit ergriffen und mit anderen (teils ebenfalls) jungen Indie-Entwicklern gesprochen und auch oft die Unreal-Frage angesprochen.

Am prägendsten war das Gespräch mit dem Chief Developer von We happy few, einem neuen Indie-Titel, durch Kickstarter finanziert, und nun auf ID@Xbox zu sehen. Compulsion Games, ein Team aus 14 Leuten, hat in We Happy Few ein dystopisches, retrofuturistisches London der 60er geschaffen. Das Gameplay und die Story laufen in einer großen, dynamischen Open-World ab und werden durch eine brillante KI ergänzt. Ich will jetzt keine Werbung für das Spiel machen, allerdings war ich erstaunt, was dieses Team hingekriegt hat. Mir ist bewusst, dass es einen Unterschied zwischen Unik und einem professionellen Indie-Entwickler-Team gibt, dennoch bin ich der Meinung, dass wir mehr als in der Lage sind, ein (halb-)lineares Horrospiel zu kreieren, zumal unsere Herausforderungen in anderen Bereichen liegen werden. Auch die drei anwesenden Entwickler von Compulsion Games rieten mir weiterhin am Ball zu bleiben, und keineswegs wegen möglicher Schwierigkeiten mit der Engine vom Kurs abzuweichen.

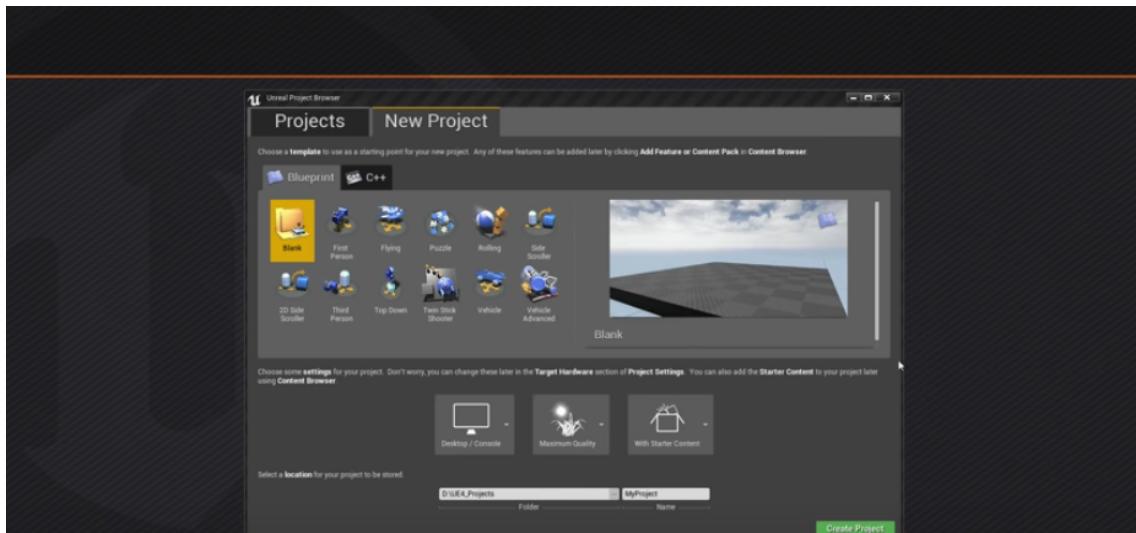
Was ich letztendlich damit sagen mag, wir weichen nicht vom Kurs ab und behalten Unreal weiterhin als Engine. Die Forumbeiträge hab ich natürlich zur Kenntnis genommen, und ich stimme definitiv auch den Problemen zu. Nach umfassender Recherche bin ich aber auch zu dem Schluss gekommen, dass Unreal diese Problemzone durch seine anderen, weitaus größeren Vorteile wieder wett macht.

2 Grundlegendes

Dieses Kapitel befasst sich mit der grundlegenden Steuerung innerhalb des Unreal-Interfaces. Alle Anleitungen sind bezüglich des Standardinterfaces geschrieben und sollen die allgemeine Steuerung innerhalb der Engine näher bringen. Manches mag einem banal vorkommen, dennoch ist eine intuitive Benutzung aller gebotenen Features ausschlaggebend für ein funktionierendes Spiel, daher bitte ich alle darum, aufmerksam die folgenden Seiten zu lesen.

2.1 Überblick der UI

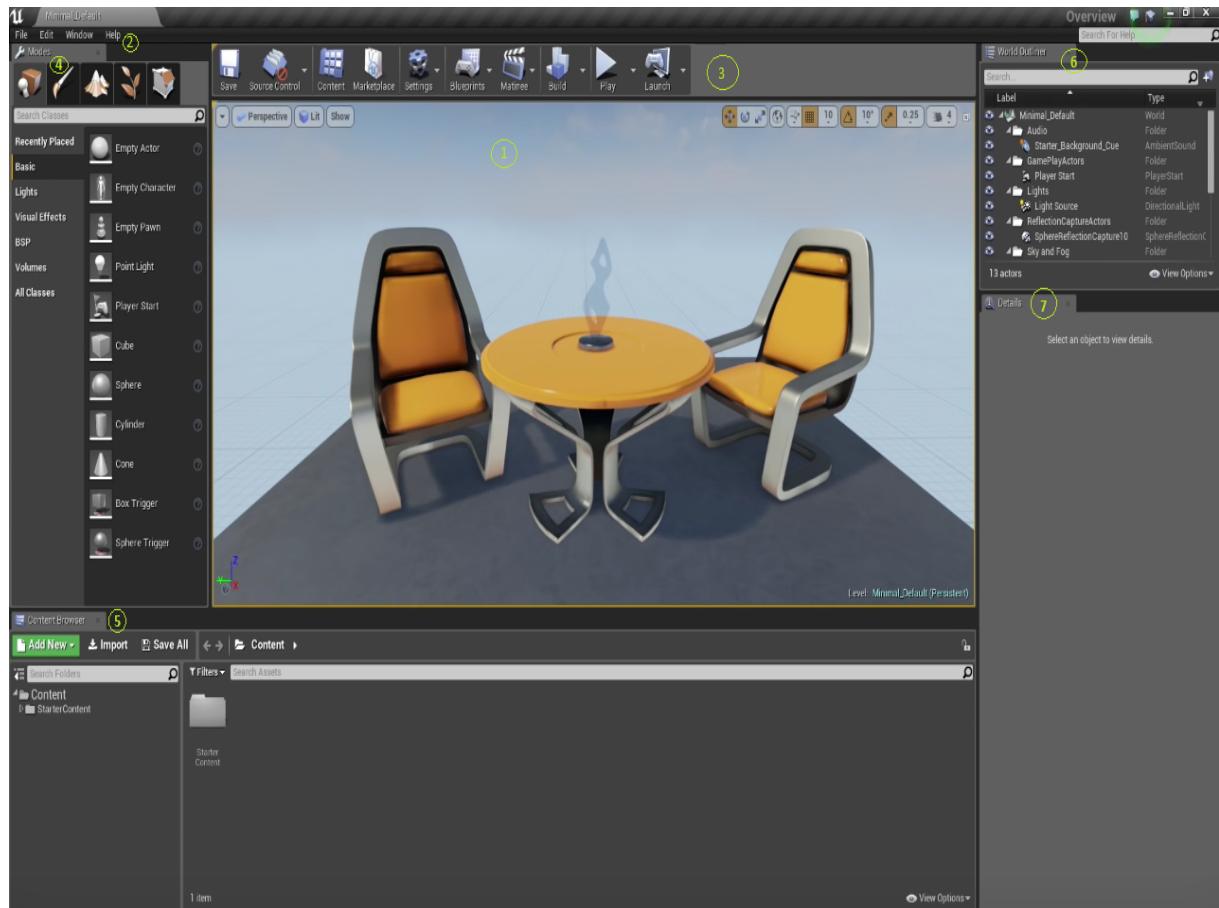
Beim Starten der Engine öffnet sich zunächst der Unreal Project Browser, der uns erlaubt ein neues Projekt zu starten oder ein bereits begonnenes fortzuführen.



Unreal bietet uns hier eine Reihe von vorgefertigten Templates an, wir benutzen jedoch ein blankes Projekt (Starter Content enabled).

Linksklick Blank → Starter Content enabled → Projekt benennen → Create Project

Das Interface sollte wie folgt aussehen:



- 1 Viewport: Level Editing Fenster, erlaubt den Blick in die 3D Welt
- 2 Menu Bar: Enthält gängige Optionen und Menüs, analog zu anderen Computer Programmen
- 3 Tool Bar: Häufig genutzte Tasks, gängige Shortcuts (Save, Play, etc.)
- 4 Modes Tab: Erlaubt Zugriff auf die verschiedenen Modi des Editors
- 5 Content Browser: Zugriff auf Content, extern und intern (Props, Meshes, Textures)
- 6 World Outliner: Liste aller Elemente in der Szene, außerdem Parenting hier möglich
- 7 Detail Panel: Erlaubt es bestimmte Attribute des gewählten Objekts zu lesen/ändern/scripten etc.

2.2 Viewport Navigation

Navigation mit der Maus:

LMT + Maus vor/zurück bewegen → Bewegung vor/zurück

LMT +Maus rechts/links bewegen → Drehung nach rechts/links

RMT + Maus bewegen → Kamera bleibt stationär, dreht in Richtung der Mausbewegung

MMT + Maus vor/zurück bewegen → Bewegung nach oben/unten

MMT + Maus rechts/links bewegen → Bewegung nach rechts/links

Falls keine mittlere Maustaste vorhanden, kann man auch LMT und RMT gleichzeitig drücken.
Die Geschwindigkeit der Kamera lässt sich in der rechten oberen Ecke des Viewports anpassen
(Camera Speed, Standard bei Wert = 4)

Navigation per Tastatur (WASD):

RMT + W/S bzw. A/D → Bewegung vor/zurück bzw. links/rechts

RMT + E/Q → Bewegung nach oben/unten

RMT + C/Z → Zoom in / Zoom out (bei Loslassen von RMT wieder Ausgangsposition)

Mausrad während Bewegung → Geschwindigkeit höher oder niedriger stellen (temporär)

Maya-Navigation:

ALT + LMT + Maus bewegen → Schwenken der Kamera um ein beliebiges Pivotobjekt

ALT + RMT +Maus vor/zurück → Einem Pivotobjekt entfernen bzw. nähern

ALT + MMT +Maus bewegen → Kamera in Bewegungsrichtung der Maus bewegen

2.3 Orthographische Ansicht

Die orthographische Kartenansicht unterstützt bei genauen Objektplatzierungen und ermöglicht eine genaue Betrachtung der Szene. In der rechten oberen Ecke des Viewports findet man den Minimalize/Maximize-Button. Durch Anklicken wird das aktuelle Viewport auf ein Viertel seiner Größe minimiert und durch die drei orthographischen Perspektiven ergänzt.

Das Fenster in der linken oberen Ecke ist die Seitenansicht, das in der rechten oberen Ecke die Frontsicht und in der rechten unteren Ecke befindet sich die Vogelperspektive.

Standardmäßig sind alle drei in der Wireframe-Ansicht.

In der orthographischen Ansicht wird die Kamera natürlich nicht frei beweglich sein. Durch RMT+Maus bewegen können wir die Kamera nur schieben (ähnlich wie bei einem Strategie-Spiel aus der Vogelperspektive), außerdem kann mit dem Mausrad gezoomt werden (auf die aktuelle Position der Maus). In der orthographischen Ansicht können wir mehrere Objekte auf einmal auswählen in dem wir die linke Maustaste gedrückt halten und alle Objekte einrahmen, die markiert werden sollen.

In der orthographischen Ansicht steht uns mehr als nur die Wireframe-Ansicht zur Verfügung. Durch Klick auf die Leiste 'Wireframe' öffnet sich ein Drop-Down-Menü, in dem wir die uns verfügbaren Ansichten auswählen können.

2.4 View Modes und Show Flags

Wie in 2.3 bereits erwähnt, stehen uns viele Ansichten zur Verfügung. Die Ansichten lassen sich in der linken oberen Ecke unserer Viewports ändern. Standardmäßig ist im Viewport links oben der Modus auf 'Perspective' und die Ansicht auf 'Lit'. Wir betrachten die gängigen Ansichten etwas genauer:

Lit: Enthält alle durch Effekte, Lichtquellen, Meshes etc. erzeugten Verhältnisse und gibt die Szene in ihrer realen Form wieder.

Unlit: Entfernt alle Lichtquellen und gibt die Szene nur mit den Farbinformationen der Meshes wieder.

Wireframe: Ähnelt dem Blueprint eines Gebäudes, zeigt die Struktur an, keine Beleuchtungsinformationen enthalten.

Detail Lighting: Zeigt nur die Beleuchtung, aber auch das Ergebnis aller Normal Maps, d.h. die Oberflächenstrukturen sind in dieser Sicht erkennbar

Reflection: Lässt die Reflexion verschiedener Oberflächen einsehen.

Es gibt noch einige andere Ansichten, die aber sehr komplex sind und uns momentan nicht interessieren. Allerdings findet man im selben Drop-Down-Menü in der letzten Zeile 'Exposure'. Exposure bezeichnet die Anpassung des Auges beim Wechsel von stark beleuchteten Räumen zu dunklen Umgebungen. Da die bereits ausgeählte Option 'Automatic' eine realistische Darstellung bietet, muss hier nicht weiter darauf eingegangen werden, jedoch könnte es nützlich sein, den Effekt hin und wieder mal auszuschalten indem man den Wert auf 'Fixed Log 0' setzt.

Show Flags, die Leiste direkt neben der Auswahl der Ansicht, sind eigentlich nur eine Ansammlung von Checkboxen, die nach Belieben an- und ausgeschaltet werden können. Hier kann beispielweise das Grid, Anti-Aliasing oder auch Partikel ausgeschaltet werden.

2.5 Objektplatzierung

Um Objekte in Unreal zu platzieren gibt es mehrere Möglichkeiten, wir sprechen kurz alle durch. Zu Beginn benutzen wir wieder ein blankes Projekt mit Starter Content. Um die bereits platzierten Objekte ebenfalls zu entfernen, klicken wir unter dem Reiter 'File' auf 'New Level' und anschließend 'Default'. Jetzt haben wir ein Level mit nur einem Player-Startpunkt und einem kleinen Stück Boden.

Die simpelste Objektplatzierung ist in Unreal per Drag&Drop aus dem Modes Panel möglich (solange man sich im Placement Mode befindet, siehe oben links im Modes Panel). Die durch das Starter-Content bereitgestellten Objekte sind praktischerweise bereits in Rubriken (Basis, Lights, Visual Effects etc.) geordnet.

Außerdem ist es möglich Objekte über den Content Browser einzufügen. Analog zum Mode Panel fügt man hier ebenfalls per Drag&Drop Objekte ein.

Die dritte Möglichkeit Objekte einzufügen befindet sich direkt im Viewport. Rechtsklick im Viewport ermöglicht es Objekte direkt zu platzieren. Es ist möglich über 'Place Actor' alle zur Verfügung stehenden Objekte zu finden, es kann allerdings auch eine Schnellwahl festgelegt werden, die es ermöglicht ein bestimmtes Objekt schnell mehrfach einzufügen. Auch die 'recently placed' Objekte sind hier zu finden und somit schnell einsetzbar.

Fortgeschrittene können unter dem Reiter 'Window' auf 'Developer Tools' mit dem Class Viewer Objekte hinzufügen. Im Class Viewer werden alle benutzten Klassen angezeigt und alle blau markierten Objekte können hinzugefügt werden, ebenfalls per Drag&Drop (im Class Viewer sind die Blueprints ebenfalls enthalten, mehr dazu später).

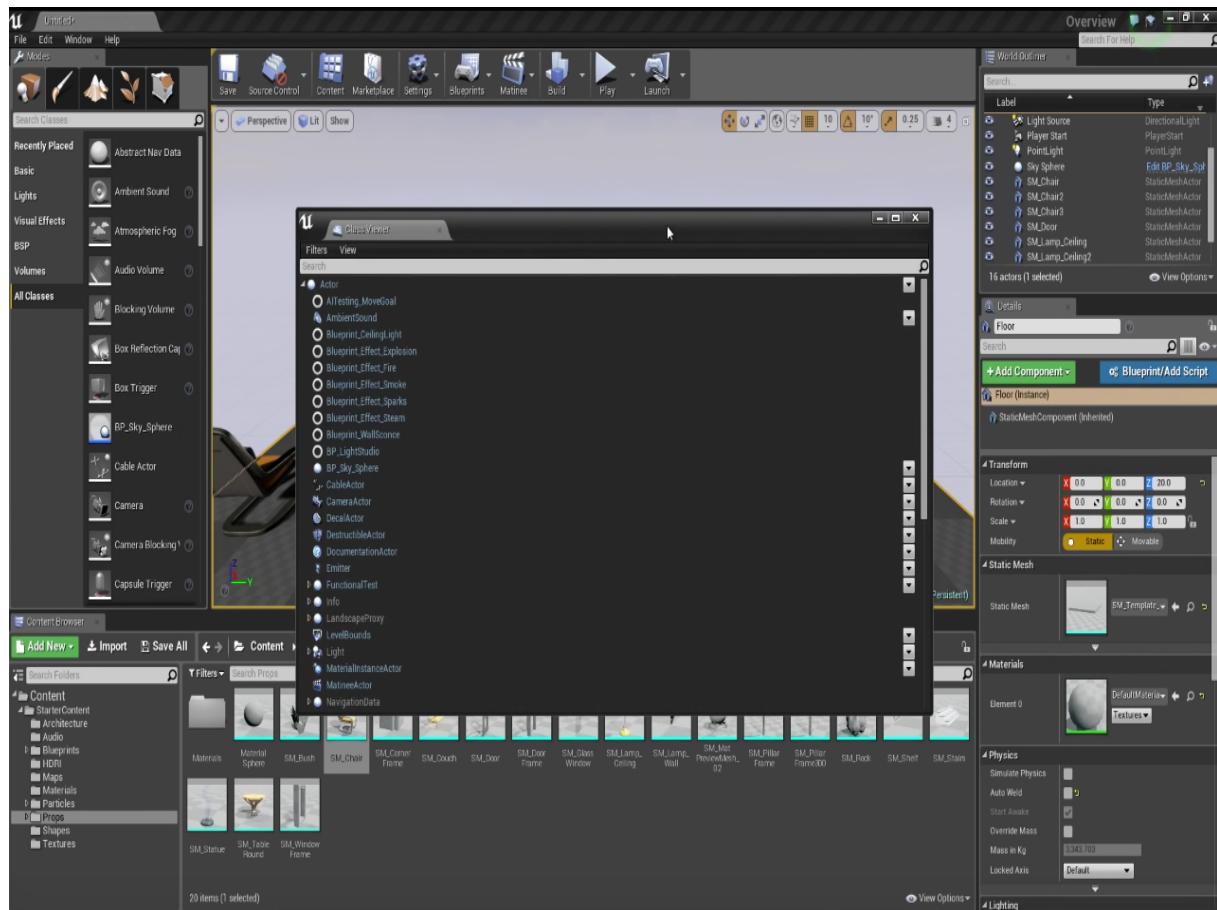


Abbildung 1: Class Viewer

2.6 Objekte bewegen

In der Standardansicht werden bei jedem ausgewähltem Objekt die 3 Achsen angezeigt. Die rot/blau/grüne Achse entspricht dabei jeweils der x/y/z-Achse. Per Klick auf eine der Achsen ist es möglich ein Objekt nur in die Richtung der gewählten Achse zu verschieben (keine versehentliche Verschiebung in Richtung der beiden anderen Achsen).

Zoomt man etwas genauer in das gewählte Objekt hinein, sieht man jeweils eine Verbindung zwischen zwei Achsen, die es ermöglicht, beide Achsen gleichzeitig auszuwählen und in die Richtung beider Achsen zu schieben (ohne Verschiebung in Richtung der dritten Achse).

Die weiße Kugel im Ursprung ermöglicht eine Bewegung in Richtung aller drei Achsen.

Hotkey-Tipp: Drücken der End-Taste bei ausgewähltem Objekt setzt dieses direkt zurück auf den Boden.

World Space: Die bisherigen Bewegungen haben alle im World Space stattgefunden, d.h. die Bewegung in x/y/z-Richtung erfolgt in x/y/z-Richtung der Welt (immer gleich).

Local Space: Local Space bezeichnet die x/y/z-Richtung des Körpers selbst. Im Normalfall sind diese identisch, ist ein Objekt aber jedoch rotiert, dann entsprechen die Richtungen nicht mehr einander.

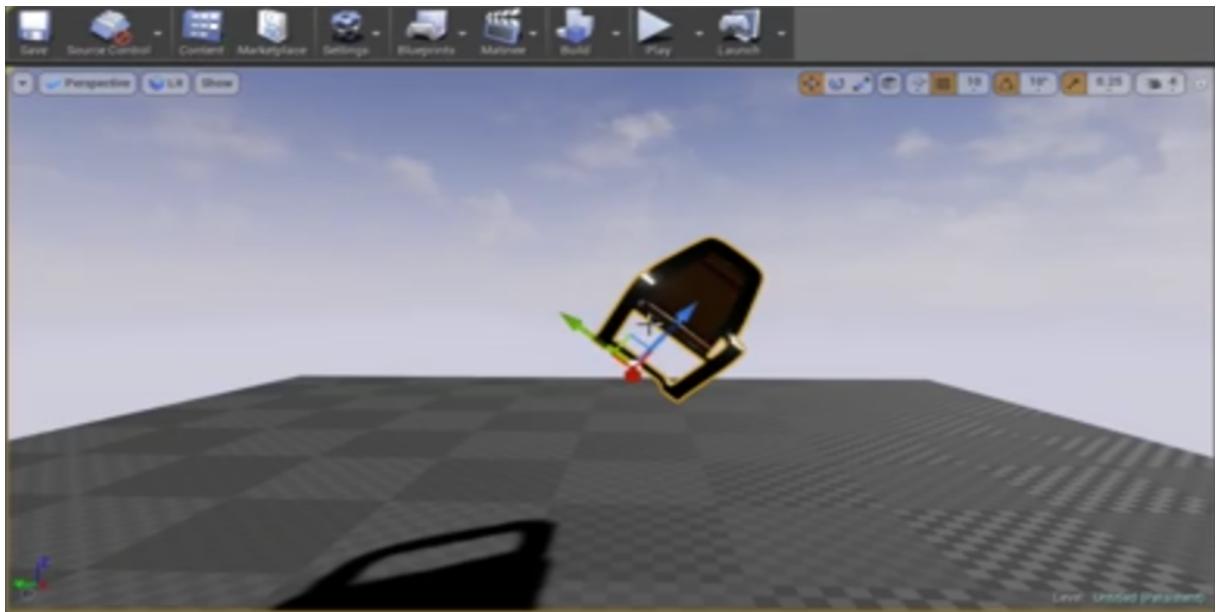


Abbildung 2: Achsen in Local Space

In der rechten oberen Ecke des Viewports ist ein Globus-Symbol sichtbar, per Klick wechseln wir hier auf den Local-Space. Die Achsen werden nun entsprechend des gewählten Objekts positioniert (siehe Bild).

Shortcut-Tipp: Strg + L ermöglicht schnelles Wechseln zwischen Local und World Space

Wie bereits bekannt sein dürfte, lässt sich die Position natürlich auch rechts im Details Panel ändern. Hierbei ist zu beachten, dass standardmäßig eine Unreal-Einheit genau 1cm entspricht. Im Detail Panel lässt sich im DropDown-Menü auch die Bewegung zwischen 'World' und 'Relative' umschalten (Relative bezeichnet Position gegenüber einem Parent-Objekt).

Grid Snapping:

Grid Snapping eignet sich für das genaue Anpassen zusammengehöriger Objekte, wie zum Beispiel Wände oder auch Räume. Der Grid Snap Value ist ebenfalls in der rechten oberen Ecke des Viewports zu sehen und ist standardmäßig auf 10, d.h. wir bewegen ein Objekt immer um 10 Einheiten in die jeweilige Richtung. Der Wert kann nach Belieben angepasst werden, oder gar komplett ausgeschaltet werden.

2.7 Objekte rotieren

Wie im vorigen Kapitel gesehen, wird bei Auswahl eines Objekts zunächst die drei Koordinatenachsen angezeigt. Ist das Objekt gewählt, wechselt man durch drücken der E-Taste in den Rotationsmodus. Statt den Achsen wird nun ein Rotationsgizmo angezeigt (Alternativ lässt sich dies auch per Klick auf das Move Tool im Viewport ändern, die W-Taste wechselt zurück zu den Achsen).

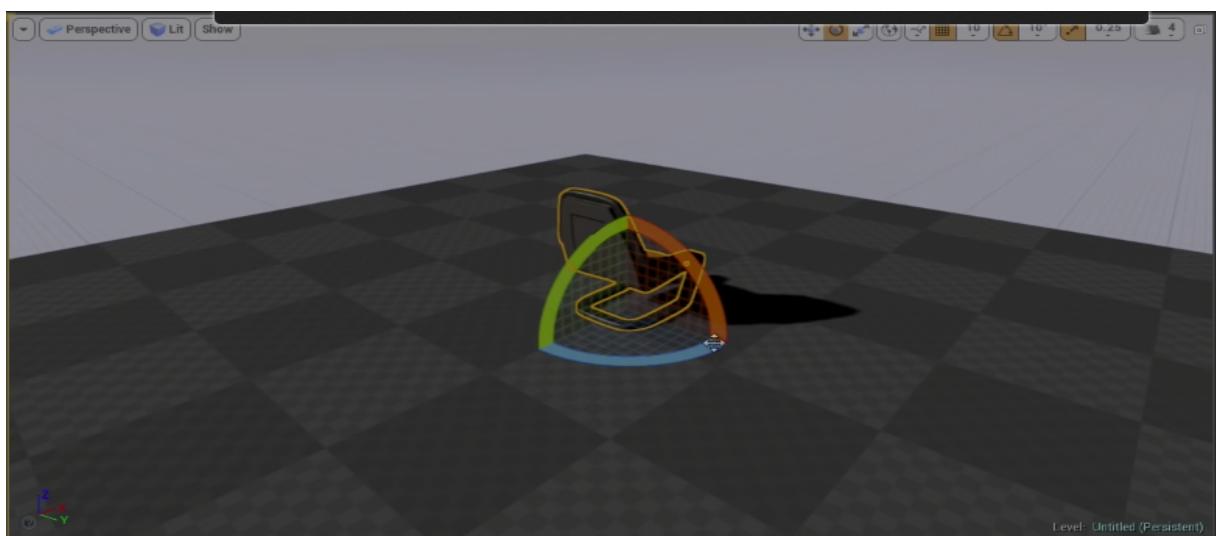


Abbildung 3: Rotationsbögen

Die drei farbigen Bögen repräsentieren Rotationsachsen. Analog zum Bewegen entlang einer Achse lässt sich hier ein Bogen auswählen und anschließend das Objekt in diese Richtung rotiert werden. Nach Klicken und Halten eines Bogens wechselt die Sicht in den Kreiskompass. Hier lässt sich beim Drehen genau ablesen, welchen Gradwert die aktuelle Rotation besitzt. Analog zum Grid Snapping wird hier standardmäßig in 10° Schritten rotiert, auch dies kann wahlweise angepasst werden. Rotationen sind ebenfalls abhängig von Local und World Space und auch wie gewohnt im Detail Panel vertreten (Merken für Blueprints: Yaw = Z-Achse (blau), Pitch = Y-Achse (grün), Roll = X-Achse (rot))

2.8 Objekte skalieren

Um in den Skalierungsmodus zu wechseln, wird bei ausgewähltem Objekt die R-Taste gedrückt. Statt den ursprünglichen Koordinatenachsen erhalten wir nun die Skalierungsachsen, die ebenfalls in x-y-z-Richtung zeigen (beim Skalieren immer in Local Space). Analog zum Bewegen eines Objektes können hier ein oder mehrere Achsen gewählt werden und durch Ziehen mit der Maus in die jeweilige Richtung skaliert werden.

Natürlich lässt sich auch im Detail Panel skalieren. Hierbei ist die eingetragene Nummer entsprechend ein Vielfaches des Standardwertes '1' (Der Eintrag '3' bei z würde das Objekt in z-Richtung strecken, bis es die dreifache Länge in dieser Richtung besitzt).

Wie bei den beiden Transformationen ist bei der Skalierung ebenfalls ein Snap-Value eingetragen, der nach Belieben angepasst oder ausgeschaltet werden kann.

2.9 Bewegen mit der Kamera

Angenommen man möchte ein Objekt bewegen, der gewünschte Zielort befindet sich aber außerhalb des Sichtbereichs. Mit den bisherigen Kenntnissen würde man das Objekt bis an den Rand bewegen (per Drag&Drop), die Kamera bewegen und anschließend erneut das Objekt bewegen. Eine Möglichkeit diesen Vorgang zu vereinfachen, ist das Objekt zu bewegen bei gedrückter SHIFT-Taste. In diesem Fall bewegt sich die Kamera mit dem Objekt mit.

Eine weitere wichtige Funktion ist das 'Locking' der Kamera an bestimmte Objekte, beispielsweise einer Lichtquelle. In der linken oberen Ecke befindet sich ein Pfeilsymbol, das ein Drop-Down-Menü öffnet. Unter der Option 'Lock Viewport to Actor' lässt sich das aktuell ausgewählte Objekt anklicken. Die Kamera springt dann automatisch zu besagtem Objekt und man 'sieht' in Richtung des Objektes (beispielsweise in Richtung des Lichtstrahls, falls eine Lichtquelle gewählt wurde).

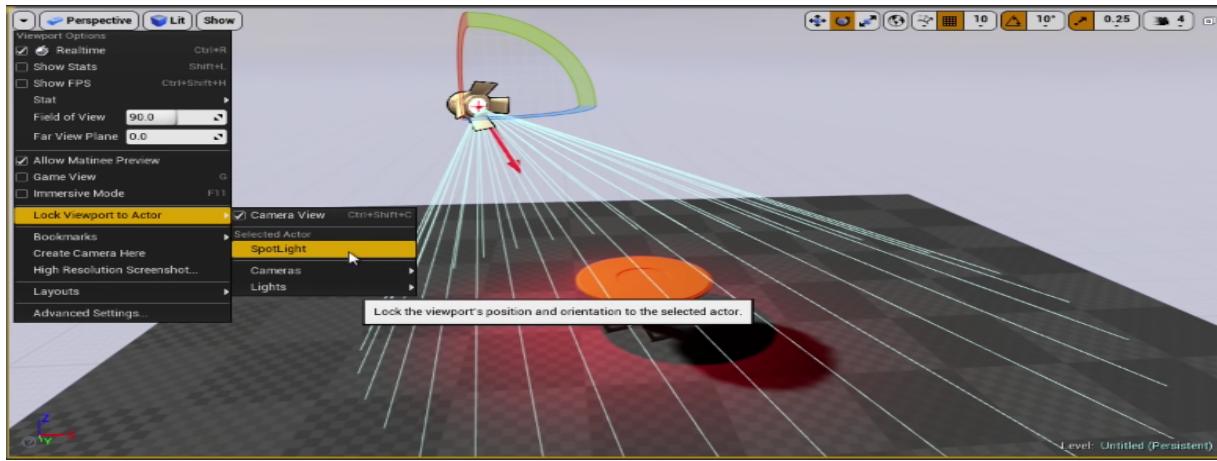


Abbildung 4: Vorteil beim Locking: Bewegen wir die Kamera, bewegt sich das gewählte Objekt mit. Im selben Drop-Down-Menü lässt sich über 'Unlock' die Kamera wieder lösen.

2.10 Intro zum Content Browser

Wie bereits erwähnt enthält der Content Browser alle Materials, Blueprints, Partikeleffekte und Meshes, sowie alle externen Assets. Sollte man versehentlich den Content Browser entfernt haben, lässt es sich jederzeit in der Toolbar über dem Viewport öffnen (Alternativ Strg+Shift+F). Praktischweise ist der Content Browser links in mehrere Ordner unterteilt und mit einer Suchfunktion ausgestattet, die das Finden von Assets erleichtern.

Externe Assets können jederzeit über den Import-Button importiert werden. Unreal unterstützt dabei eine Vielzahl von Formaten, die Liste ist unter 'Documentation' zu finden. Außerdem können Assets auch direkt in den Ordner C/.../UE4_Projects/'Projektnam'/Content hinzugefügt werden.

2.10.1 Assets über den Content Browser editieren

Eine der Hauptfeatures des Content Browsers ist die Möglichkeit Assets zu editieren. Per Doppelklick auf ein Asset öffnet sich der entsprechende Editor, um das gewählte Objekt zu bearbeiten.

Static Meshes:

Nach dem Doppelklick öffnet sich der Static Mesh Editor. Hier können Einstellungen für Collisions vorgenommen, Materials hinzugefügt und Standardeinstellungen verändert werden.

Blueprints:

Der Blueprint Editor ermöglicht es, die Abläufe der Blueprints zu modifizieren. Mehr zu Blueprints gibt es im entsprechenden Kapitel.

Material Editor:

Ebenfalls in Unreal enthaltener Sub-Editor, der die Bearbeitung von Materials ermöglicht.

2.10.2 Navigation im Content Browser

Dank den enthaltenen Sub-Editors ist es möglich, in Unreal neue Assets direkt selbst zu erstellen. Über den Button 'Add New' in der linken oberen Ecke lässt sich jederzeit ein neues Objekt mit den bereits erwähnten Editors erstellen (Unter 'Add New' befinden sich auch einige nützliche Templates, unter anderem auch Blueprint Templates die direkt als Blueprints oder auch als C++ Version enthalten sind). Des Weiteren ist es möglich jederzeit eigene Ordner in der links beigefügten Hierarchie anzulegen oder C++ Klassen einzufügen, ebenfalls über den 'Add New' Button (Achtung: Änderungen und Hinzufügen mit dem 'Save all' Button speichern).

Im Ordnerverzeichnis an der linken Seite stehen uns auch einige Optionen (Rechtsklick auf leeren Bereich) zur Verfügung, allerdings sind diese größtenteils selbsterklärend. Wichtig ist allerdings die Option 'Connect to Source Control'. Hier kann man, falls ein großes Team Source Control benutzt, sich jederzeit mit dieser verbinden bzw. die Verbindung trennen.

Ein weiteres nützliches Feature ist die Lock-Funktion auf der rechten Seite des Content Browsers. Man nehme folgende Situation: Aus dem Ordner Static Meshes sollen Körper hinzugefügt werden, anschließend dann verschiedene Materials eingefügt werden. Ist ein Körper ausgewählt, kann man per Klick auf das Lupen-Icon neben Material im Detail Panel ein Material hinzufügen. Nach Klick auf die Lupe öffnet sich der entsprechende Ordner im Content Browser. Möchte man nun laufend neue Körper und Materials hinzufügen, muss man zwischen diesen beiden Ordnern hin und her springen. Betätigt man aber vorher das Lock Icon im Content Browser, bleibt dieser auf den aktuell ausgewählten Ordner. Dies ermöglicht es, viele Objekte aus verschiedenen Ordnern parallel einzufügen, da nun nach dem Betätigen des Lupen-Icons ein neues Content Browser Fenster sich öffnet. Es können bis zu vier Content Browser gleichzeitig aktiv sein.

Direkt links neben der 'Search Content'-Leiste befindet sich eine Filter-Option, die es ebenfalls einfacher macht, gesuchte Assets zu finden. Die Checkboxen können nach Belieben aktiviert/deaktiviert werden, die Ergebnisse werden unabhängig von ihren jeweiligen Ordner angezeigt.

In der rechten unteren Ecke lassen sich außerdem die 'View Options' bearbeiten, sollte man die Anzeige des Content Browsers verändern wollen (Die 'Columns'-Ansicht enthält zu jedem Objekt weitere Informationen, die in der Tiles oder List-Ansicht nicht angezeigt werden). Hier lassen sich auch die Größe der Tiles skalieren und deren Thumbnails mit dem 'Thumbnail Edit Mode' verändern. Im aktivierte 'Thumbnail Edit Mode' lässt sich auch die Form des Materials ändern, zum Beispiel von einer Sphäre zu einem Würfel, per Klick auf die angezeigte Form direkt auf dem Thumbnail.

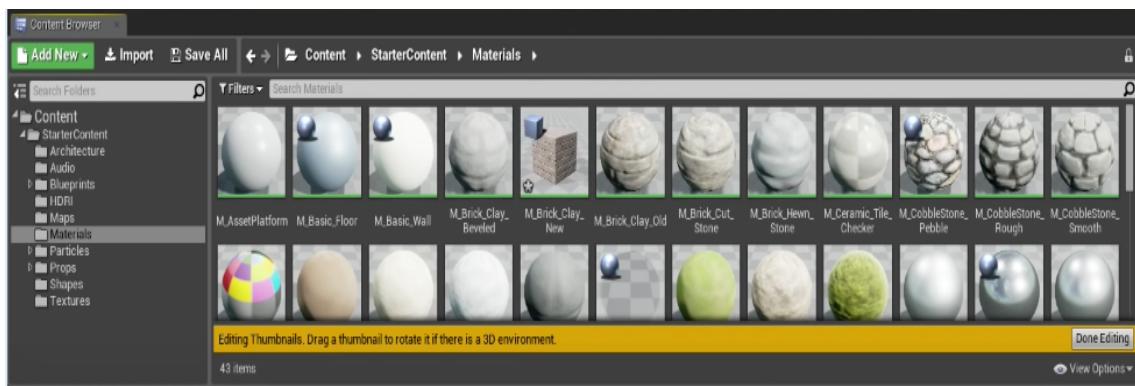


Abbildung 5: Content Browser im Thumbnail Edit Mode, in der linken oberen Ecke jedes Thumbnails lässt sich die Form ändern

3 Blueprints

Blueprints sind die wichtigsten Hilfsmittel beim Entwickeln eines Spiels, gerade wenn Zeit eingespart werden soll oder auch die C++ Kenntnisse noch nicht ausreichend sind. Sie ermöglichen die schnelle Implementierung häufig verwendeter Abläufe und sind gerade für gescriptete Events optimal einzusetzen. Unreal Engine kommt mit vorgefertigten Blueprints, außerdem ist es jederzeit möglich eigene Blueprints zu erstellen. Das Blueprint-System ist ausschlaggebend für das Design eines Videospiels, sollte dennoch trotzdem als Kernfeature unserer Engine von allen beherrscht werden. Da ein Großteil unserer Animationen ebenfalls per Blueprint festgelegt werden, bitte ich alle darum, sich umfassend mit diesem Kapitel zu beschäftigen.

3.1 Einführung einfacher Blueprints

Bevor man sich komplett in das Blueprint-System vertieft, sollte man zunächst die bereits vorhandenen Möglichkeiten innerhalb der Unreal Engine kennen. Unreal birgt bereits eine Vielzahl von Objekteigenschaften, die gerade für Blueprints später notwendig.

3.1.1 Destructible

Die realistische Zerstörung von Objekten ist meist ein schwieriges Unterfangen. Für Static Meshes hat Unreal bereits ein physikgetreues Modell, dass die einfache Erzeugung von Destructibles ermöglicht. Im Content Browser per Rechtsklick das jeweilige Mesh auswählen und im Pop-Up-Menü die Option 'Create Destructible-Mesh' auswählen.

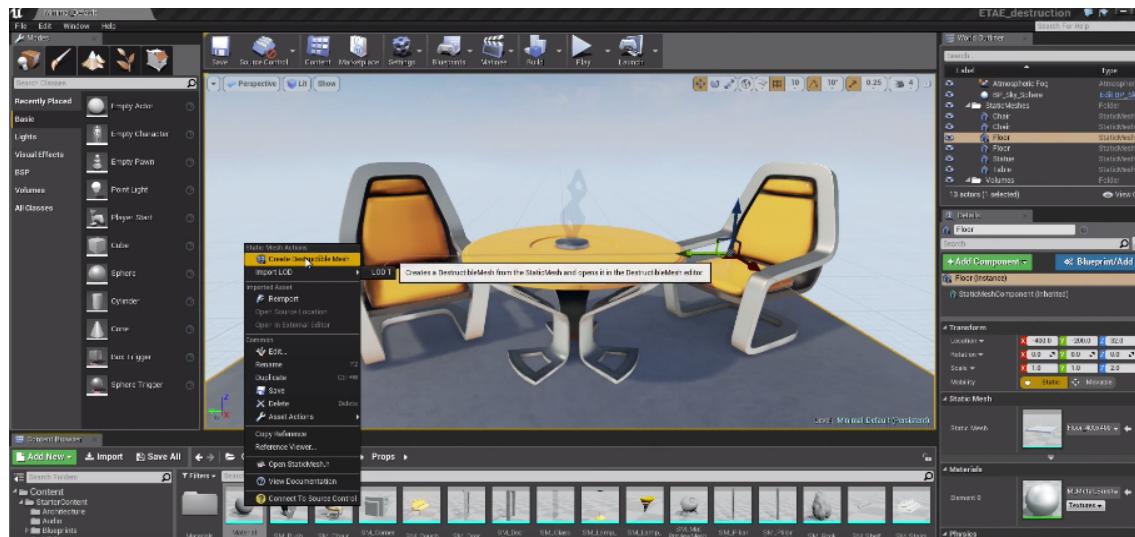


Abbildung 6: Erzeugung eines Destructibles

Das gewählte Mesh wird dupliziert und mit einer kleinen rosa Leiste im Content Browser angezeigt. Alle Objekte mit dieser rosa Leiste sind Destructibles, per Doppelklick lässt sich der Destructible Editor öffnen.

Hier lassen sich auf der rechten Seite Einstellungen für das Destructible vornehmen. Uns interessiert hier zunächst mal das Feld 'Cell Site Count'. Hier lässt sich die Anzahl der Zellen festlegen, und somit die Feinheit des Zerfalls (Achtung: Größere Zahl hier bedeutet mehr Berechnungen → Perfomanz). Über den Button 'Fracture Mesh' in der oberen linken Ecke wird anschließend das Mesh als zerstörtes Mesh angezeigt. Über den Schieberegler 'Explode Amount' kann man den Zerfall beobachten.

Ein paar weitere Einstellungen sind hier wichtig:

- Damage Threshold: Hier lässt sich festlegen, bei welchem Schaden, dass Objekt zerfällt
- Enable Impact Damage: Wenn aktiviert, nimmt das Objekt bei physikalischer Aktion Schaden.
- Impact Damage: Physikalischer Schaden das Objekt wahrnehmen kann, bevor es zerbricht.

3.1.2 Camerashake

Ein Camera-Shake-Effekt ermöglicht es auf einfache Weise große Vibrationen darzustellen, um so zum Beispiel einer Explosion Nachdruck zu verleihen. Damit wir diesen Effekt einmal selber implementieren, laden wir das First-Person-Template als Projekt.

Wir beginnen damit, dass wir per Rechtsklick in den Content Browser die Funktion 'Blueprint Class' auswählen, um ein neues Blueprint zu erstellen. Über Suche lässt sich die Camera-Shake-Klasse schnell finden, auswählen und dann öffnen. Nun kann man das Template nach belieben benennen und anschließend im Blueprint Editor öffnen. Wir beachten wieder die Attribute auf der rechten Seite.

Oscillation Duration: Dauer des Camera-Shakes

Rotation Oscillation: Gibt dem Shake auch eine gewisse Rotation (Pitch, Yaw, Roll)

Location Oscillation: Die Oszillation in x/y/z-Richtung

Nach Eingabe der Werte erst kompilieren ('Compile') und anschließend speichern.

Wir öffnen nun den Blueprint FirstPersonCharakter im Blueprint-Editor. Das untere Rechteck beinhaltet alle Blueprints, die beim abfeuern der Waffe benötigt werden. Ganz am Ende der Blueprint-Kette 'Play Sound at Location' wollen wir nun den Camera-Shake hinzufügen. Mit der Maus können wir an dem nicht-gefülltem Pfeilsymbol eine neue Linie ziehen und somit ein neues Node platzieren. Nachdem man die Linie mit der Maus gezogen hat, öffnet sich ein Suchfenster. Durch Eingabe von 'shake' finden wir das Node 'Play World Camera Shake' und wählen dieses aus. Es wird damit direkt ein neues Node platziert.

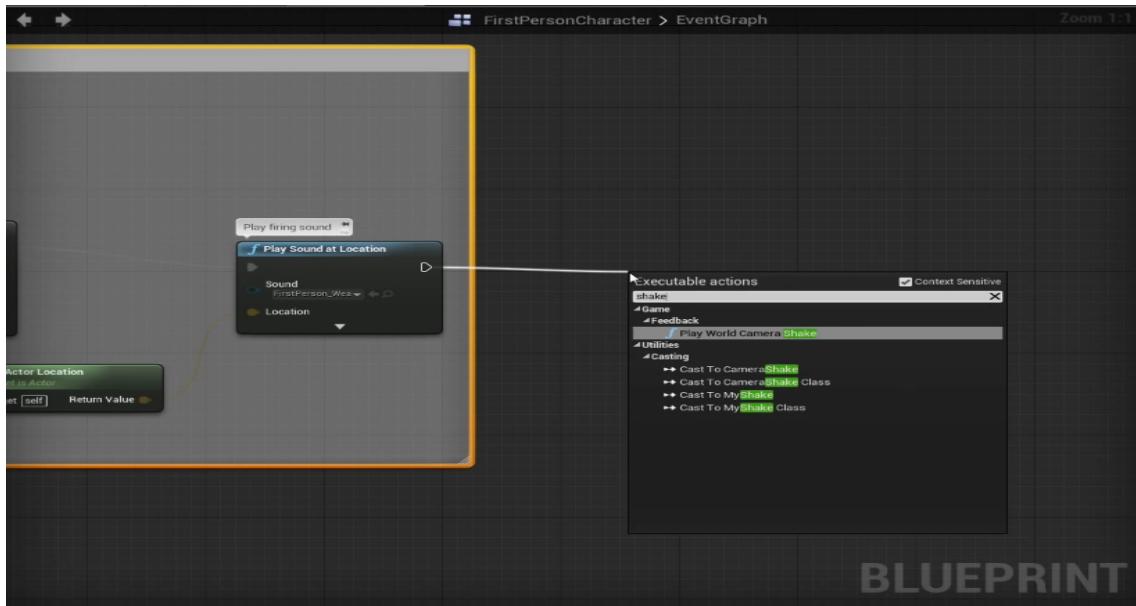


Abbildung 7: Neuen Node platzieren

Der platzierte Node besitzt eine Menge von Eigenschaften, darunter 'Shake'. Hier definieren wir die benutzte Klasse für den Shake-Effekt. Die von uns neu erstellte Shake-Klasse kann dann hier ausgewählt werden. Nun brauchen wir noch eine Location, an der besagter Shake ablaufen wird (Epicenter). Im bereits implementierten FirstPersonCharacter-Blueprint befindet sich das Node 'Get Actor Location'. Wir verbinden diesen Node mit unserem Node, damit der Shake die jeweilige Position erhält. Die anderen Werte sind erstmal zweitrangig.

Wenn wir das ganze nun abspeichern und zurück zu unserem Spiel wechseln, wird jetzt bei jedem Schuss eine Vibration der Kamera stattfinden.

3.1.3 Timer

Timer können in vielen Situationen hilfreich sein. Sie ermöglichen das Ablaufen bestimmter Ereignisse in einem definierten Zeitintervall ein- oder mehrmals. Um den Timer zu implementieren, benutzen wir hier eine Lichtquelle, dessen Farbe wir in bestimmten Zeitabständen verändern.

Ist ein Objekt ausgewählt, wird wie gewohnt rechts das Detail Panel angezeigt. Für jedes Objekt haben wir hier die Möglichkeit einen Komponenten hinzuzufügen ('Add Component') oder eine Blueprint-Klasse zu erzeugen, die dieses Objekt enthält ('Blueprint/Add Script'). Wir wählen also unsere Lichtquelle und erstellen eine Blueprint-Klasse über den blauen Blueprint-Button im Detail-Panel.

Die erstellte Blueprint-Klasse ist nun im Content Browser zu finden. Zum Bearbeiten öffnen wir die Klasse im Blueprint-Editor. Im Viewport wird uns die gewählte Lampe angezeigt, zum Bearbeiten der Abläufe benötigen wir den Event Graph, welcher einer der Tabs über dem Viewport ist. Hier sehen wir die vetaute Form des Blueprint-Editors mit den Nodes, noch ohne Verbindungen.

Um den Ablauf bei Beginn zu starten, ziehen wir erstmal eine Linie vom Node 'Event Begin Play'. Im folgenden Menü suchen wir nach dem Blueprint 'SetLightColor(PointLightComponent)' bzw. SetLightColor(benutzte Lichtquelle). Im neuen Node ist nun eine neue Farbe nach dem Wechsel einzugeben. Wir möchten diese Farbe zufällig wählen, dazu öffnen wir per Rechtsklick erneut das Nodemenü und wählen den Node 'Random Vector' (Farbe ist ein Vektor RGB). Um den zufälligen Vektor nun noch in einen gültigen Farbvektor zu formen, erstellen wir das Node 'ToLinearColor'. Jetzt können wir Node 'Random Vector' mit Node 'ToLinearColor' verbinden und dieses anschließend an das Attribut 'New Light Color' unseres Blueprints 'SetLightColor' anknüpfen.

Hinweis: Der Timer ist zwar noch nicht implementiert, aber wenn wir jetzt zurück im Editor auf 'Play' klicken, hat unsere Lichtquelle einen zufälligen Farbwert.

Um den Timer zu implementieren, ziehen wir erneut eine Linie vom Node 'Event Begin Play' und fügen das Blueprint 'SetTimer' an. Die Attribute sollten selbsterklärend sein, es ist darauf zu achten, dass Looping aktiviert ist, damit die Farbe sich wiederholend ändert. 'Function Name' bezeichnet welche Funktion aufgerufen wird. Wir erstellen per Rechtsklick ein 'Custom Event' und benennen es beispielsweise RandomCol. Wir verbinden RandomCol mit unserer vorher erstellten SetLightColor und können dann beim Timer den FunctionName 'RandomCol' benutzen(Kompilieren und Speichern nicht vergessen).

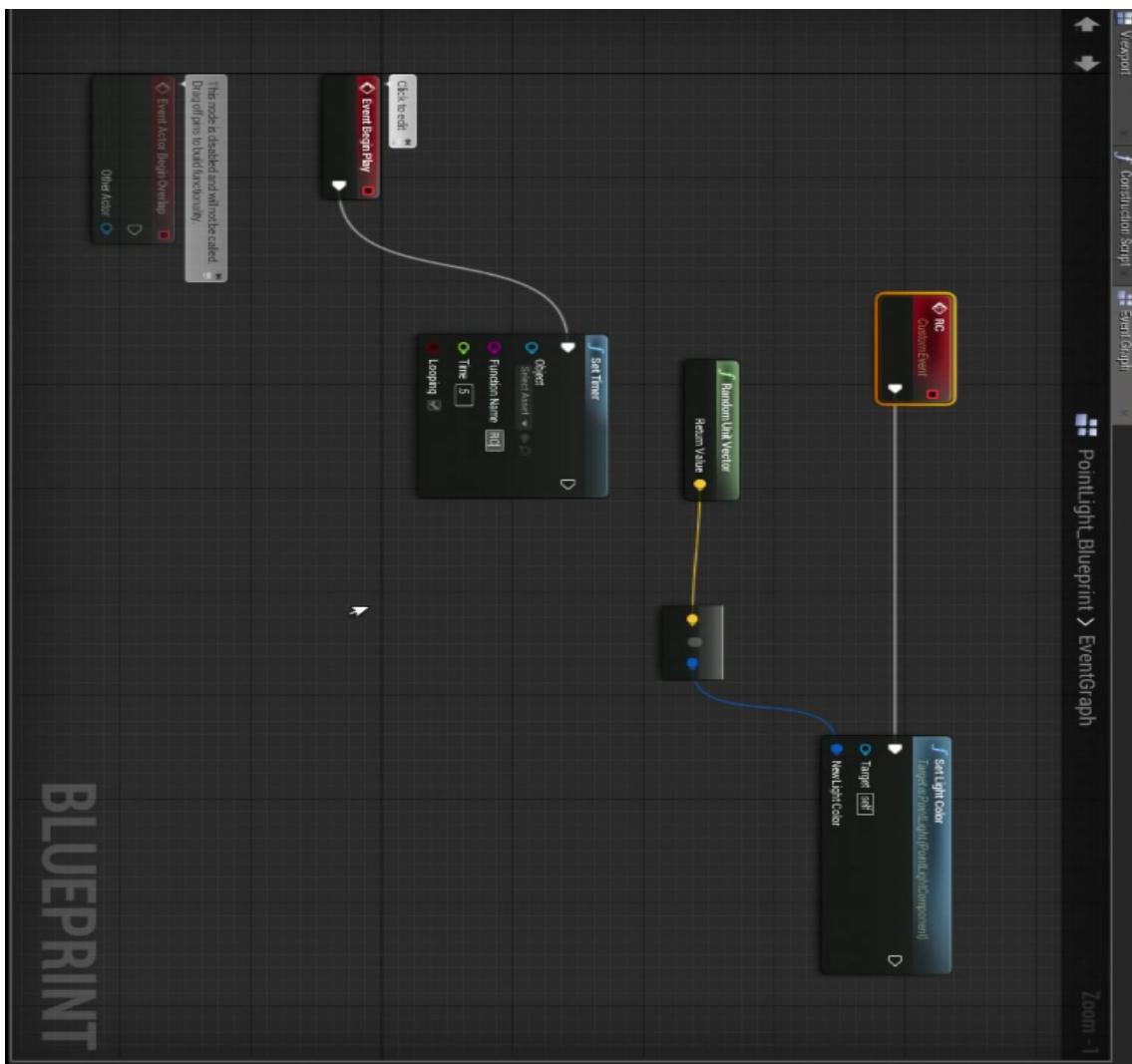


Abbildung 8: Nodes für Lichtwechsel in bestimmtem Zeitabstand

3.1.4 Slow Motion Effect

Wir starten erneut mit einem neuen Projekt und laden das First Person Template. Um den Slow-Motion Effekt einzufügen, öffnen wir FirstPersonCharacter im Blueprint-Editor und erstellen ein neues Node 'Event Tick' (wie gewohnt per Rechtsklick und anschließend über die Suchleiste). Event Tick ist ein Node, dass für jeden Frame aufgerufen wird.

Wir erstellen ein zweites Node 'Execute Console Command', welche wir benötigen werden, um den Slow-Motion-Effekt an unseren Charakter anzupassen . Außerdem benötigen wir ein drittes Node 'Build String(float)', welches über Return Value an das Command-Attribut des Execute Console Command verknüpft ist (das Prefix slomo ist ein Shortcut zum Console Command für die Zeitlupe).

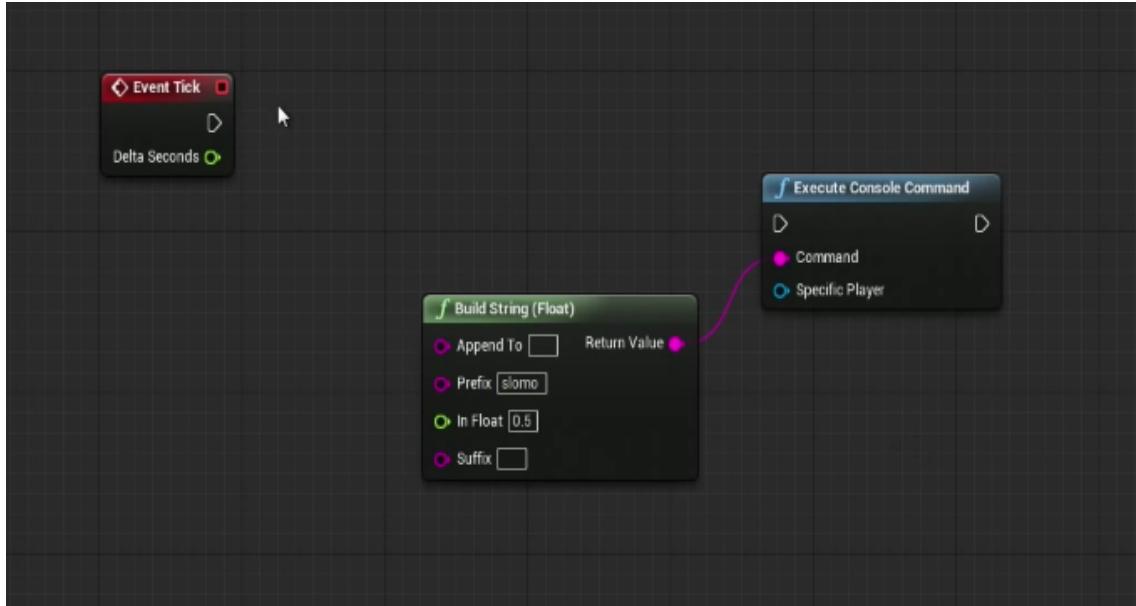


Abbildung 9: Prefix 'slomo' , float-Wert soll vom Charakter abhängig sein

Um den float-Wert an den Charakter anzupassen, erstellen wir ein neues Node 'Get Velocity'. Der return-value von Velocity ist ein Vektor, den wir nicht als float benutzen können, allerdings können wir wiederum das Node 'Vector length' aufrufen, der die Länge des Vektors berechnet. Diesen können wir allerdings noch nicht an den float-Wert knüpfen, da sich die Länge des Vektors nicht zwischen 0 und 1 befindet. Links im Reiter Components finden wir die Komponente Character-Movement, hier können wir einsehen, dass die minimale Geschwindigkeit unseres Charakters 0 und die maximale Geschwindigkeit 600 ist. Diese Werte benötigen wir für den Node 'Map Range', welchen wir jetzt erstellen. Als In Range geben wir 0 für A und 600 für B ein, als Out Range 0 und 1. Dies bedeutet, dass unsere Vector Length auf Werte zwischen 0 und 1 gemappt wird, solange die Input-Werte zwischen 0 und 600 liegen.

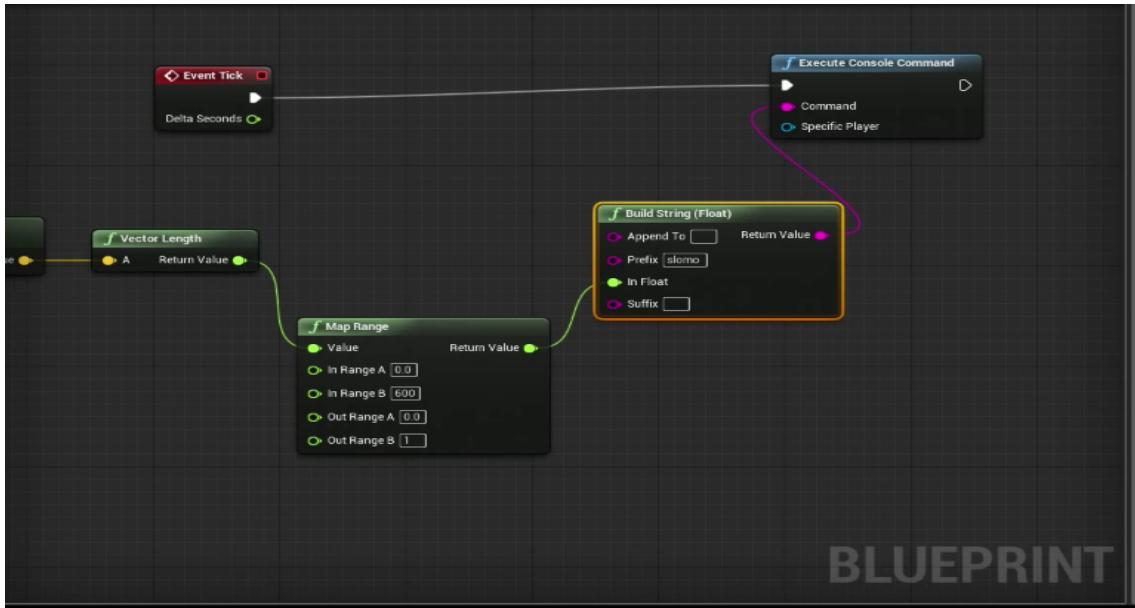


Abbildung 10: Slow-Motion abhängig von Geschwindigkeit des Charakters

In diesem Beispiel ist das Spiel sehr langsam, wenn der Spieler sich kaum/nicht bewegt, aber kaum auffallend wenn der Spieler sich schnell bewegt. Analog zu vorigen Beispielen kann der Slow-Motion-Effekt unabhängig von der Spielergeschwindigkeit sein und entweder als Funktion aufgerufen werden (siehe CameraShake) oder auch an einen Timer gekoppelt werden.

3.1.5 Jump Pad

Zu Beginn wählen wir das Third-Person-Template und erzeugen wie bereits bekannt über den Content Browser eine neue Blueprint-Class vom Typ 'Actor'. Anschließend wird die neue Klasse benannt und im Blueprint-Editor geöffnet.

Zu Beginn braucht unser Jump Pad ein Mesh, welches das Pad repräsentieren soll. Über 'Add Component' fügen wir der Einfachheit halber das Basic Component Cylinder. Per Scaling im Panel rechts passen wir die Dimensionen entsprechend unserer Vorstellung an (Traditionelle Jump Pads sind Zylinder mit sehr kleiner Höhe). Ebenfalls im rechten Panel unter dem Reiter lässt sich die Einstellung 'Collision Presets'. Standardmäßig ist diese Option auf 'BlockAllDynamic', was natürlich für ein Jump Pad eher kontraproduktiv ist. Beim Jump Pad wollen wir die Möglichkeit haben, es mit dem Spieler zu überlappen aber trotzdem das Event zu erzeugen. Die gewünschte Einstellung ist also 'OverlapAllDynamic', aber die Option 'Generate Overlap Events' sollte aktiviert bleiben(Kompilieren und Speichern nicht vergessen).

Den neuen Actor können wir jetzt in unsere Szene positionieren. Noch wird nichts passieren, da wir ja kein Event für den Overlap definiert haben. Dazu wechseln wir wieder in den Blueprint-Editor und wählen den Event-Graphen. Wir erstellen das Node 'Event Actor Begin Overlap'. Dieser Node wird jedes mal aufgerufen, wenn ein anderer Actor sich mit dem jetzigen überlappt. Über das Attribut 'Other Actor' können wir festlegen, welche anderen Actors das Event auslösen können. In diesem Beispiel setzen wir an dieses Attribut nur unseren ThirdPersonCharacter (neues Node-Cast to ThirdPersonCharacter, verbinden). Natürlich können auch andere Objekte als Eventauslöser festgelegt werden, zum Beispiel Kisten oder ähnliches. An den Charakter können wir das Blueprint 'Launch Character' andocken. In dem Node 'Launch Character' können wir die Geschwindigkeit des Sprungs in x,y,z-Richtung definieren (Ein Sprung entspricht nur einem z-Wert → vertikal). Einen beliebigen Wert eingeben, kompilieren und speichern. Beim Starten wird unser Charakter nun jedes mal, wenn er das Jump Pad berührt, in die Luft geschleudert.

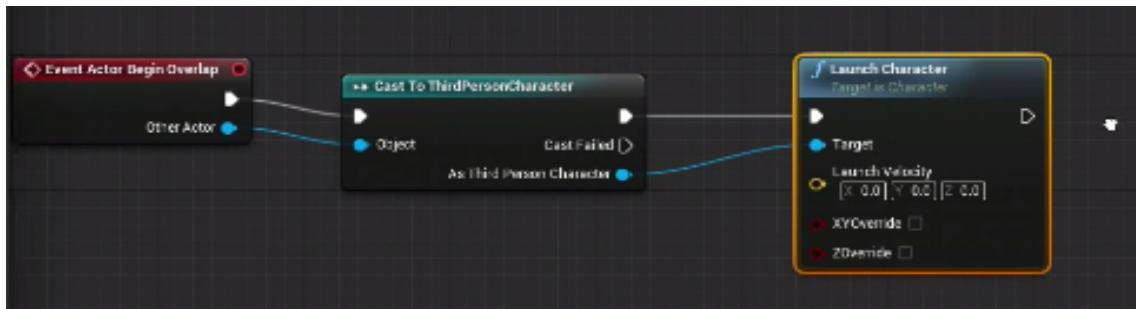


Abbildung 11: Blueprint für das Jump Pad

3.1.6 Hover-Komponente

3.1.7 Simples Conveyor Volumen

3.1.8 Komponenten für Game-Behaviour