Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

Fakultät DMI
Studiendepartment Technik

28.08.2009

# Bachelorarbeit

zur Erlangung des akademischen Grades Bachelor of Science (B. Sc.)

Thema: „Game Design in der Unreal Engine: Die Entstehung eines Mehrspieler-Levels für Unreal Tournament 3“

Erster Gutachter:
Prof. Gunther Rehfeld, HAW Hamburg

Zweiter Gutachter
Prof. Dr. Roland Greule, HAW Hamburg

vorgelegt von
Daniel Thiele

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| BSP | Binary Space Partitioning bezeichnet die grobe Aufteilung des Raumes mit simplen Formen und geometrischen booleschen Operationen. |
| GUI | Das Graphical User Interface (Benutzeroberfläche) umfasst das Menü und die räumliche Einteilung eines Programms in mehrere Fenster, die dem Benutzer helfen, das Programm zu benutzen. |
| UnrealEd | Der Unreal Editor ist der Level Editor der Unreal Engine. |
| UT3 | Unreal Tournament 3 ist ein First Person Shooter, der eine Mehrspieler-Auskopplung der „Unreal”-Reihe darstellt. |
| MSUC | Der Make Something Unreal Contest ist ein Wettbewerb, bei dem Künstler und Programmierer ihre Modifikationen und ihre eigenen Inhalte für Unreal Tournament 3 einschicken können, um Preise im Gesamtwert von 1000000 US-Dollar und eine Lizenz zur kommerziellen Vermarktung eines Spieles auf Basis der Unreal Engine 3 zu gewinnen. |

# Problemstellung und Gang der Untersuchung

Die Unreal Engine ist eine hochentwickelte und weit verbreitete Game Engine. Alle Entwicklertools, mit 20 Stunden Videomaterial zu deren Bedienung, sind bei Unreal Tournament 3 dabei. Unreal Tournament 3 beinhaltet die aktuellste Version des Level Editors, auch wenn seit Erscheinen des Spieles bereits weitere Spiele auf Basis der Unreal Engine veröffentlicht wurden.

Viele Menschen haben das Spiel bereits modifiziert und es gibt hunderte von Anleitungen dazu im Internet. Zum praktischen Game Design in einer Game Engine gibt es keine Monographien mit wissenschaftlichem Anspruch. Die einzigen Bücher zur Unreal Engine 3 erschienen erst kurz vor Ende der Arbeit. 3D-Künstler und Game Designer lernen die Programme heutzutage über Videos kennen, die man entweder als DVD kaufen oder sich kostenlos im Internet ansehen kann. Solche „Tutorials“ kann jeder an seinem Computer erstellen und im Internet hochladen. Professionelle Game Designer und Game Artists bieten ihre Videos zum Verkauf an. In Amerika gibt es Professoren, die Videos zu Lernzwecken für ihre Studenten erstellen.

Da die CryEngine2 an der  Hochschule für angewandte Wissenschaften lizensiert wurde und als großer Konkurrent der Unreal Engine erschien, ist es interessant die Unterschiede beider Engines aufzuzeigen. Bisher wurden nur die optischen Ergebnisse beider Engines verglichen, doch in der Spielentwicklung ist Grafik nicht der wichtigste Faktor. Ein Spiel muss heutzutage auf mehreren Zielplattformen veröffentlicht werden können und die Entwicklertools eine schnelle Einarbeitung, sowie ergebnisorientiertes Arbeiten ermöglichen.

Ziel ist es ein vollständiges Mehrspielerlevel für Unreal Tournament 3 zu erstellen. Zunächst werden die grundlegenden Begriffe genau definiert und eine Einführung in das Genre gegeben. Level Design und Game Art Workflows werden ebenfalls vorgestellt. Anhand dieser Basis werden die Grundbegriffe des Unreal Editors erläutert. Anschließend wird die Unreal Engine 3 mit der CryEngine2 verglichen, um aufzuzeigen, ob die CryEngine2 eine gute Wahl für Spielentwickler ist.

# 1. Einführung

## 1.1 Die Unreal Engine als Game Engine

Eine Game Engine ist eine Software, die speziell für die Entwicklung von Videospielen konzipiert wurde. Die Unreal Engine ist eine der am meist verbreiteten kommerziellen Game Engines weltweit und beinhaltet diverse Technologien zum Echtzeit-Rendering[1] von 3D-Grafik, Sound, Physik und Animationen. Dadurch ist es möglich, das Geschehen als Spieler zu beeinflussen und trotzdem sehr realistisch wirkende Bilder auf dem Fernseher oder am Computer zu produzieren. Texturen, Sounds, 3D-Objekte usw. werden in einer Game Engine zu einem interaktiven Programm verbunden. Die Unreal Engine besteht dabei aus einigen kleineren Engines, die unabhängig voneinander arbeiten und zur Laufzeit miteinander synchronisiert werden. Die Bestandteile sind Grafik-Engine, Physik-Engine und Sound-Engine. Die Berechnung steht dabei immer in Abhängigkeit vom gegebenen Input. Der Input Manager erfasst die Eingaben des Spielers und leitet sie an die Grafik-Engine weiter. Die Grafik-Engine zeigt genau das an was der Spieler sehen soll. Im selben Moment werden die Daten an die Sound Engine weitergeleitet, sodass der entsprechende Sound wiedergegeben wird. Die Physik Engine berechnet während dessen Animationen wie bewegliche Lichter, das Verhalten von Fahrzeugen usw.

Sie zeichnet sich durch gute Grafik bei sehr guter Performance aus, sodass Spiele nicht nur auf den neusten Computern gut aussehen. Die Unreal Engine 3 ist die aktuellste Version auf dem Markt und diverse Spiele wurden bisher damit entwickelt. Durch ein Partnerprogramm hat Epic Games einige führende Middleware

[1] Rendering ist die Berechnung von dreidimensionalen Bildern.

Technologien in seiner Engine vereinigen können, um sie so weiter zu verbessern[2]. Außerdem können Entwickler Spiele für Computer mit DirectX 10, Xbox 360 und PLAYSTATION 3 entwickeln. Die Unreal Engine wird ständig weiter entwickelt und obwohl die vierte Version, welche erst für die nächste Generation von Spielen geplant ist schon in Entwicklung ist, gibt es weiterhin regelmäßige Updates der Unreal Engine 3.

Der Unreal Editor ist fester Bestandteil der Engine und eine sehr komfortable Entwicklungsumgebung, Er wird benutzt, um Levels, Charaktere und Vehikel in das Spiel einzubringen.

Epic Games weiß, dass „Modding"-Support ein wichtiges Element zur Vermarktung von Spielen ist, da die Spieler oft gerne selber neue Inhalte in die bestehende Spielwelt einbringen und das Spiel auch manchmal verbessern oder gänzlich verändern (Total Conversion). Hinzu kommt, dass jedes Entwicklerstudio, das ein Spiel in der Unreal Engine entwickelt hat, den Unreal Editor mit dem Spiel vermarkten kann. Es gibt darum auch einige Käufer, die sich nur das Spiel kaufen um den Editor zu benutzen.

Als Marketingstrategie hat Epic Games für Unreal Tournament 2004 bereits 2007 zum ersten Mal den „Make Something Unreal"-Contest (MSUC) veranstaltet, bei dem Modder für ihre eigenen Kreationen belohnt werden.

Die Unreal Engine wird nicht nur für Spiele verwendet sondern auch für Fernsehserien und Visualisierungen.[3] Sie ist also eine Entwicklungsumgebung für jegliche Art von interaktiver Darstellung in Echtzeit.

---

[2] Infomationen zum Partnerprogramm mit weiterführenden Links findet man unter EPIC Games, Partners, 2008

[3] Eine grobe Übersicht über die Erfolge der Unreal Engine 3 bietet EPIC Games, Succes, 2008

## 1.2 Unreal Tournament 3

Unreal Tournament ist ein First Person Shooter, bei dem es vor allem um wettkampfartige Mehrspielerpartien in futuristischen Environments geht. Unreal Tournament 3 ist aber nicht der dritte Teil der Serie - die Nummerierung bezieht sich lediglich auf die Version der Game Engine.

Die ursprüngliche Geschichte handelt von einem Turnier in einer weit entfernten Zukunft, welches von einem Konzern auf Grund der hohen Kriminalitätsrate gegründet wurde. An diesem Turnier kann jeder, der sich traut, teilnehmen und Preise gewinnen. Es gibt verschiedene Spielmodi sowohl für Einzelkämpfer, als auch für Teams (Clans). Im „Kern“[4] des traditionellen Deathmatch geht es darum, alleine so viele Gegner so schnell wie möglich auszuschalten. Der Spieler mit den meisten „Kills“ gewinnt die Runde. In jedem Level befinden sich „Spawner“, bei denen man wiederbelebt wird, nachdem man eliminiert wurde, weshalb es eigentlich keine Toten bei Unreal Tournament gibt, sondern jeder permanent auf dem Spielfeld ist.

Inzwischen wurde die Geschichte weitergeführt und es herrscht Krieg zwischen verschiedenen Rassen. Auch wenn Epic Games versucht hat, einen Story Modus in das Spiel zu integrieren, richtet es sich vor allem an „Competitive Gamers“[5] und „Professional Players“[6], für die der Story-Modus unwichtig ist. Es gibt weltweit Turniere in denen Teams gegeneinander antreten.

Während die Kämpfe früher in Arenen ausgefochten wurden, geschieht dies heute in großen Außenarealen oder gar ganzen Stadtteilen. Hinzu kommen die verschiedensten Fahrzeuge und Waffen. Mit der Zeit wurden neue Spielmodi entwickelt und alte verbessert. Bei dem neuen „Warfare“-Modus muss man in verschiedenen Environments Knotenpunkte nacheinander einnehmen und

---

4 Vgl. BRATHWAITE, Brenda; SCHREIBER, Ian, *Core, 2009, What is a Game, S. 6ff.*

5 Vgl. BRATHWAITE, Brenda; SCHREIBER, Ian, *Competetive, 2009, What is a Game, S. 111*

6 Vgl. BRATHWAITE, Brenda; SCHREIBER, Ian, *Professional, 2009, What is a Game, S. 110 f.*

verteidigen, um letztendlich den gegnerischen Basisknoten zu zerstören und die Runde für das eigene Team zu entscheiden.

Die Bedienung der Waffen und Fahrzeuge erfordert Geschick und Übung, bzw. Skill[7], sodass nur Profis alle taktischen Möglichkeiten erkennen und nutzen können. Hinzu kommen noch weitere Gegenstände, wie spezielle Waffen, Rüstung, zusätzliche Lebensenergie und andere sogenannte PowerUps, die an bestimmten Punkten im Level verteilt wurden. Erfahrene Spieler wissen, wo diese Gegenstände sind und wie sie diese am effektivsten erreichen. Außerdem ist es von Vorteil, die Umgebung für eigene Zwecke zu nutzen. Es gibt beispielsweise Waffen, deren Kugeln an den Wänden abprallen, sodass man einen Gegner um die Ecke ausschalten kann. Dabei muss man aber geschickt vorgehen und darauf achten, dass man sich nicht selber verletzt.

Außerdem gibt es von Level zu Level unterschiedliche Fallen und weitere Einzelheiten, die ein Spieler benutzen kann, um sich einen Vorteil gegenüber seinen Gegnern zu verschaffen. Wichtig sind für Profis Werkzeuge wie ein kleiner tragbarer Teleporter (Translocator), die Möglichkeit bei einem Sprung in der Luft nochmal zu Springen (Double Jump), ein fliegendes Skateboard (Hoverbeard) usw.

Realismus wird von den Entwicklern als sehr wichtiger Faktor angesehen, da gute Grafik die Verkaufszahlen beeinflusst und zusätzlich auch Werbung mit dem Spiel für besondere Features der Game Engine gemacht wird. Manchmal wirkt es für Spieler so, als ob die Entwickler mit dem Spiel nur Werbung für die Engine machen. Das Spiel sieht dann sehr gut aus, aber die Story und Möglichkeiten für den Spieler bieten wenig Innovation.

Ein wichtiger Grund für die realistische Grafik, Physik, Sound und auch Blut in Computerspielen ist die Immersion. Der Spieler wird bei Unreal Tournament in eine Welt hineingezogen, in der alle Maschinen, Environments, Charaktere und Spezialeffekte sehr plastisch wirken und man als Spieler Spaß daran hat, zuzusehen

---

[7] Vgl. BRATHWAITE, Brenda; SCHREIBER, Ian, *Skill, 2009, What is a Game, S.83 ff.*

wie diese Science Fiction-Welt funktioniert und sie auch zu benutzen bzw. zu zerstören oder diese Welt durch einen Mod neu zu erschaffen und damit zu erweitern. Es ist wie ein interaktiver Film.

## 1.3 Unreal Editor (UnrealEd)

„Der Unreal Editor ist ein "What You See is What You Get"-Level Editor und fester Bestandteil der Unreal Engine. Die neuste, öffentlich zugängliche, Version (Version 3.5) ist bisher ausschließlich Bestandteil von Unreal Tournament 3 (UT3). Im UnrealEd wurden alle Levels für UT3 gestaltet. In diesem Editor befinden sich außerdem über 1000 Objekte, sogenannte Assets, die in externen Anwendungen erschaffen wurden, wie 3D Meshes, Texturen, Sounds und Animationen.

Im UnrealEd lassen sich über Binary Space Partitioning (BSP), mit Hilfe von booleschen Funktionen, geometrische Formen erzeugen, die als Basis für die spätere Umgebung fungieren. Diese Basis wird dann texturiert und mit Static Meshes detailliert. Anschließend werden Partikeleffekte, vorprogrammierte Ereignisse und Zwischensequenzen, hinzugefügt.

UnrealEd ist nicht dazu geeignet, selber komplexe 3D-Objekte, Texturen oder Musik zu schaffen, sondern ist dazu gedacht, diese in der Unreal Engine zu benutzen und für die entsprechenden Zwecke anzupassen. Es gibt für 3ds Max, Maya und Softimage XSI Plugins, mit deren Hilfe Assets in den Editor exportiert werden können." (eigene Übersetzung)[8]

[8] Vgl. BUSBY, Jason; PARRISH, Zack, Intro, 2007,002_ Intro_to_UnrealED, Intro_to_UnrealEd.wmv, ab 0:20min.

### 1.3.1 Bestandteile des Unreal Editors

Der Unreal Editor besteht aus mehreren kleineren Editoren, die jeweils ihren eigenen Namen haben. Die Verwendung dieser einzelnen Werkzeuge zur Erstellung eines Levels soll hier an einem theoretischen Beispiel erklärt werden. Auf die einzelnen Editoren wird später auch im Detail eingegangen, da sie zur Erstellung eines sichtbaren Ergebnisses unverzichtbar sind, sämtliches Grundlagenwissen zum Thema Level Design in der Unreal Engine abdecken und eine Rolle für den späteren Vergleich mit der CryEngine2 spielen.

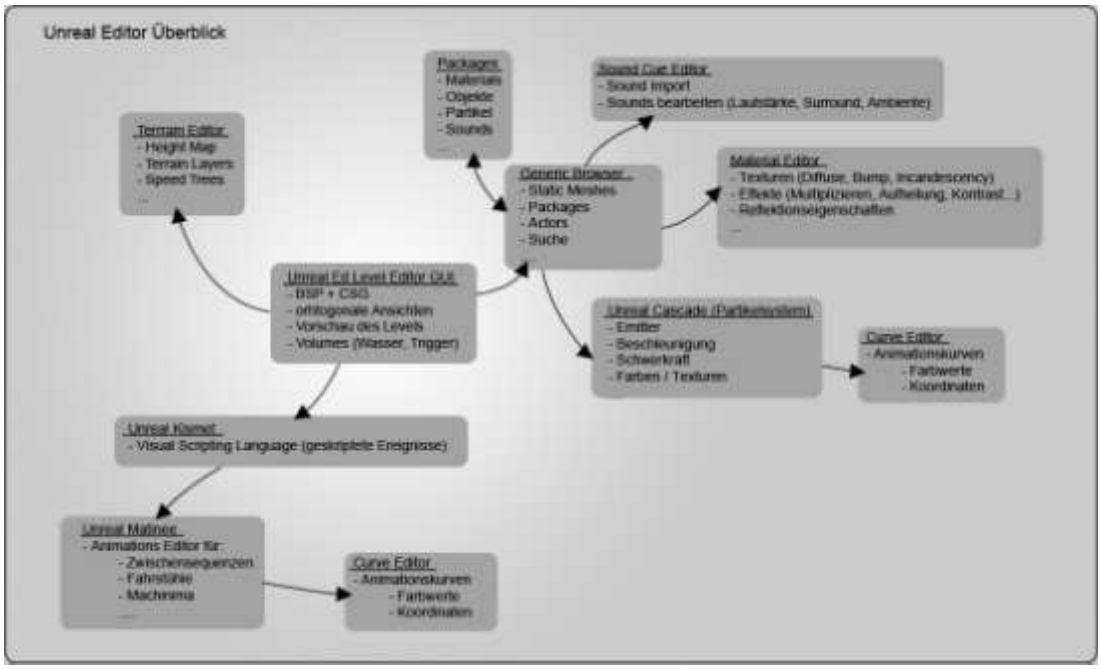



*Abbildung 1.1: Aufbau des Unreal Editors*

Im eigentlichen **Level Editor** wird ein leerer Raum erschaffen und mit BSP ein Grundriss des Levels gestaltet. Dazu gehören ebene Flächen und simple Formen, wie Rechtecke, Zylinder usw. Der Level Designer verwendet nur simple Formen.

Nachdem ein Grundkonzept steht, fertigt das Concept Art Team ein Design an, das am besten zu dem Grundriss passt und fertigt dazu mehrere Zeichnungen an. Die 3D-Designer erstellen Objekte und Texturen, welche daraufhin in den Editor importiert werden.

Sie werden in den **Generic Browser** importiert und in **Packages** gespeichert. Ein Package enthält immer Assets, die zusammengehören. Die Packages werden geöffnet und deren Inhalt kann dazu verwendet werden, das Level auszuschmücken.

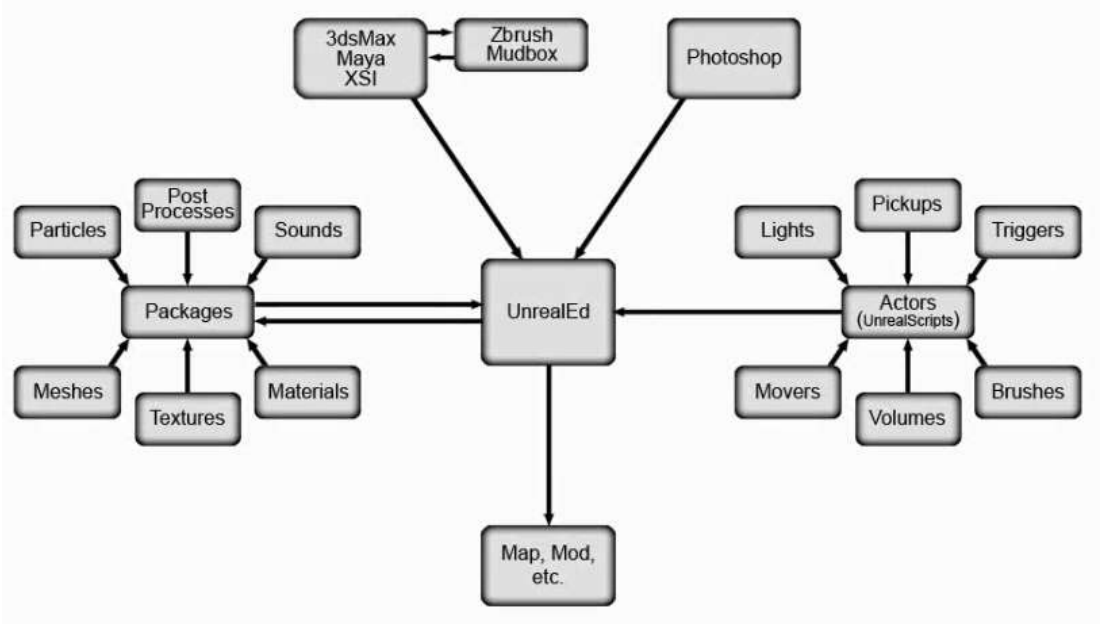



*Abbildung 1.2: Genereller Überblick außerhalb des Editors*[9]

Damit ein Static Mesh auch wieder so aussieht wie in der 3D-Applikation, müssen die zugehörigen Texturen im **Material Editor** zu einem neuen Material zusammengefügt werden. Das Material wird dann im **Static Mesh Editor** auf das Static Mesh gelegt.

Danach kann das Static Mesh aus dem Generic Browser in das Level eingefügt werden. Zusätzlich kann im **Terrain Editor** eine natürliche Umgebung erstellt werden. Der Terrain Editor ist ein Werkzeug im Level Editor und ermöglicht es, mit Brushes Höhenunterschiede und Pflanzen zu malen. Die Bodentexturen kommen, wie auch die Pflanzen und sogenannte Speed Trees, wieder aus dem Generic Browser und werden in externen Anwendungen erstellt.

---

[9] Vgl. BUSBY, Jason; PARRISH, Zack, UT3 Bonus, 2007, 002_GeneralWorkflow.wmv, ab 0:01min

Wenn der Level Designer einen Aufzug baut, der automatisch hoch und runter fährt, sobald ein Spieler ihn betritt, muss er das Static Mesh des Aufzugs animieren. Er erstellt dafür in dem visuellen Skripting-Editor **Cascade** eine **Matinee**-Sequenz. In der Matinee-Sequenz wird angegeben, wie schnell und wie lange der Aufzug fahren soll. Das optische Ergebnis kann man direkt im Level Editor sehen und verändern. In Cascade wird dafür gesorgt, dass diese Matinee-Sequenz abgespielt wird, sobald der Spieler den Aufzug betritt und dass sie sich wiederholt, wenn der Spieler noch einmal fahren möchte. Das visuelle programmieren hat den gleichen, logischen Ablauf wie das eigentliche Programmieren, jedoch wird es mit optischen Hilfsmitteln vereinfacht. Der Designer muss dementsprechend nicht die Syntax von **Unreal Script** beherrschen, weshalb Unreal Script auch nicht detailliert erklärt wird.

Das Level soll, in unserem Beispiel, dynamische Elemente enthalten. Eine Feuerstelle mit einem knisternden Lagerfeuer und Blättern, die ab und zu - möglichst zufällig - von den Bäumen fallen. Dazu erstellt der Designer in **Kismet** Partikeleffekte. Partikel sind in der Unreal Engine einzelne Objekte, die ein Material haben, sowie unter Umständen auch ein Mesh, und bestimmte physikalische Eigenschaften bekommen. Die einzelnen Partikel werden dann von einem Emitter ausgestrahlt. Dieser Vorgang wird im späteren Verlauf weiter erläutert. Zu dem Feuer verwendet der Level Designer noch einen entsprechenden Sound, der nur in der Nähe seiner Quelle zu hören ist, und dessen Lautstärke abhängig von der Entfernung zu dieser variiert. Der passende Sound befindet sich in einem Package. Er wird im **Sound Cue Editor** bearbeitet und danach im Level Editor platziert. Zum Abschluss werden Waffen und evtl. Vehikel aus dem **Actor Classes** Browser, der sich im Generic Browser-Fenster befindet, eingefügt.

Es gibt weitere Editoren und Werkzeuge für die Unreal Engine, die nicht ausführlich in dieser Arbeit behandelt werden, auch wenn diese relevant sein mögen, da die Zeit fehlt und der Umfang den Rahmen sprengen würde.

**AnimTree Editor** ist dafür gedacht zB. Charaktere zu animieren. Jeder Charakter hat eine Sammlung an Animationen, die im AnimTree Editor bearbeitet werden. Dieser Editor ist hauptsächlich für Charakterdesigner und Vehikel-Designer gedacht, die ihre Assets animieren.

Ein weiteres Tool ist **Phat**, welches sich hauptsächlich mit den Physik- und Kollisionseigenschaften von Assets beschäftigt und in die Kategorie Charakterdesign fällt. Das Thema Physik wird nur am Rande erwähnt, weil zerstörbare Umgebungen im Mehrspieler Modus ungeeignet sind, aus Performance- und aus Balancinggründen.

**Online** ist eine Programmiersprache, die es ermöglicht das Spiel im Netzwerk zu optimieren. Der Spielmodus für das Level, dass hier erstellt wird, ist schon vorhanden und sehr ausgereift.

Der **UI Editor** ist zum Erstellen und Anpassen neuer Interfaces, wenn ein neuer Spielmodus entsteht.

Der **Post-process Editor** kann bestimmte Kameraeffekte, wie beispielsweise Depth of Field und Motion Blur simulieren und wird dementsprechend für Zwischensequenzen genutzt. [10]

## 1.4 Game Development vs. Game Design

Die Begriffe Game Development und Game Design umfassen verschiedene Teilbereiche innerhalb der Production Pipeline von Computerspielen. Game Development ist dabei eher der Begriff für die gesamte Produktion, wobei aber beide oft sehr unterschiedlich verstanden werden. In diesem Abschnitt sollen die zwei Grundbegriffe für das allgemeine Verständnis kurz beschrieben werden. Es geht hierbei ausschließlich um die Entwicklung von Videospielen.

„Ein Spiel ist eine Form von unterhaltsamer Beschäftigung mit Regeln. Der Spieler spielt entweder mit dem Spielsystem, anderen Spielern, Zufall oder Glück.“[11], wie

---

[10] Einen groben Überblick über alle Bestandteile der Unreal Engine findet man unter EPIC Games, Technology 2008.

beim Pokern. „Die meisten Spiele haben Ziele“, wie etwa das Erreichen des höchsten Punktestands. „Oft müssen Entscheidungen getroffen werden“, wie das Entwickeln einer Gesellschaft in Richtung Wissenschaft bei Strategiespielen oder das Umgehen eines Gegners anstelle eines frontalen Angriffs, „aber nicht bei allen Spielen. Spiele die am Computer oder an der Konsole gespielt werden, bezeichnet man als Videospiele. Sie basieren auf den gleichen Grundprinzipien wie analoge Spiele.“ (eigene Übersetzung) [12]

### 1.4.1 Game Development

„Dieser Begriff beschreibt die gesamte Entwicklung eines Computerspiels von der Idee, über die Team-Zusammenstellung bis zum Arbeitsablauf. Wichtig ist dabei für das gesamte Team, dass es nicht nur aus guten Mitarbeitern bestehen, sondern auch effektiv zusammenarbeiten muss. Außerdem ist es wichtig, frühzeitig Verzögerungen zu erkennen und daraus Konsequenzen zu ziehen.“[13] Verzögerungen entstehen oft bei Computerspielen, da beispielsweise die Umsetzung der Ideen zu viel Programmieraufwand erfordert oder die Designertools zu aufwändig sind. „Alpha und Beta Tests sind auch Teil des Game Developments, sowie Web Design und Marketing.“(eigene Übersetzung)[14]

### 1.4.2 Game Design

„Game Design ist ein Teilaspekt des Game Developments und ein Überbegriff für die Gestaltung der Spielwelt und Regeln des Spiels, zu dem beispielsweise folgende Disziplinen gehören: World Design[15], System Design[16], Content Design[17], Interface

---

11 Vgl. BRATHWAITE, Brenda; SCHREIBER, Ian, Game, 2009, What is a Game, S. 5 f.

12 Vgl. BRATHWAITE, Brenda; SCHREIBER, Ian, Game, 2009, What is a Game, S. 5 f.

13 Vgl. BUSBY, Jason; PARRISH, Zack; VAN EENWYK, Joel, Game Developement, 2005, The Process of Game Development, S.19 ff.

14 Vgl. BUSBY, Jason; PARRISH, Zack; VAN EENWYK, Joel, Game Developement, 2005, The Process of Game Development, S.19 ff.

15 Erschaffung von Hintergrundgeschichte und Thema des Spiels.

Design[18], Level Design und Game Writing[19].“[20] Balancing ist ein weiterer Aspekt, dessen Ziel es ist Spielregeln, Items und deren Stärken miteinander zu vergleichen, damit ein Spieler keine Vorteile gegenüber einem Anderen bekommt, sondern die Chancen zu gewinnen gleich verteilt sind. In dieser Arbeit heißt es, Pfade für Nichtspieler-Charaktere anzulegen und Items so zu verteilen, dass sie jeder im Level erreichen kann und somit das ganze Level auch gleichmäßig ausgenutzt wird.

Diese Arbeit soll sich vor allem mit dem Thema Level Design und Game Art für ein Videospiel beschäftigen und neben den zahlreichen technischen Aspekten auch Einblick in die theoretischen Hintergründe geben. Game Designer müssen dabei immer im Rahmen der technischen Gegebenheiten bleiben. Darum müssen sie mit den Technikern eng zusammenarbeiten. Dazu gehört die Wahl der richtigen Game Engine, in Abhängigkeit vom Genre und der Zielplattform. Der Vergleich zwischen CryEngine2 und Unreal Engine 3 soll zeigen, wie sich Game Engines und Entwicklertools unterscheiden können und wofür sie geeignet sind.

Level Design ist die praktische Erarbeitung einer Spielhandlung, die räumlich, in Form von Levels, innerhalb der Game Engine, unterteilt wird. Content und Story werden so im Level angeordnet, dass der Spieler es versteht und damit interagieren kann. Gegenstände der Welt werden im Level platziert, um gewisse Situationen und Stimmungen herbeizuführen, die der Spieler durchleben soll. Dabei soll er die Spielregeln nutzen können, um das Level zu absolvieren. Der Level Designer muss darauf achten, Hindernisse und Aufgaben zu einer lösbaren Herausforderung zu machen. Außerdem muss das Level optisch entsprechend gestaltet werden und die technischen Möglichkeiten sollen optimal genutzt werden. Eine Rennstrecke muss beispielsweise in einer vorgegebenen Zeit auch wirklich schaffbar sein. Ziele und

---

16 Erschaffung der Spielregeln.

17 Erschaffung der für den wichtigen Inhalte, wie Charaktere mit Hintergrundgeschichte, Missionsziele und Lösungsweg oder Items und deren Funktion im Spiel.

18 Gestaltung der Benutzeroberfläche, wie Lebensenergie, Radar und Inventar.

19 Erstellung von Dialogtexten und Handlungen innerhalb der Spielwelt.

20 Vgl. BRATHWAITE, Brenda; SCHREIBER, Ian, *Game Design, 2009, The Basics, S. 2 ff.*

Hindernisse müssen deutlich ersichtlich sein und das Spiel darf bei einem schnellen Rennen nicht anfangen zu ruckeln.

### 1.4.3 Level oder Map

Bei Online-Spielen, wie Unreal Tournament, wird oft der Begriff Map statt Level verwendet und eine nachträglich von Spielern erstellte Map heißt Custom Map. Das Bauen eigener Maps wird auch als Mapping bezeichnet. Level ist also ein Überbegriff.

Der Begriff Map hat seinen Ursprung bei kompetitiven Mehrspielerspielen. Im Einzelspieler von UT (1999) gibt es zu Beginn einfach aufgebaute und kleine Maps mit leichten Gegnern und einfach zu bedienenden Waffen. Im Mehrspielerpart kann man beim Deathmatch jede verfügbare Map auswählen. Wenn man den oder die Gegner besiegt hat, kommt der Spieler ins nächste Level. Die Maps werden dann schwerer und die Gegner besser. Die Waffen sind schwerer zu bedienen und die Items liegen an schwer erreichbaren Stellen. Man spricht hier vom Level als Landkarte oder Spielfeld, auf dem eine Partie gespielt wird. Nachdem die Runde vorbei ist, geht es zur nächsten Map. Das erste Mal ist das Wort Map im Spiel Quake (1996) vorgekommen.

„Maps sind die größten Assets in der Unreal Engine und werden nicht in Packages gespeichert. Ein Level ist die Umgebung, in der das Spiel stattfindet. Es erzeugt die Atmosphäre und Hintergrundstimmung des Spiels, aber es ist auch eine Sammlung aus allen Gegenständen, Skriptereignissen und Spielelementen. Gute Maps erfordern sehr viel Planung und die Ideen sollten immer von den Teamkollegen kritisiert werden.

Eine Map beinhaltet viele weitere Assets, die für den visuellen Effekt sorgen, sodass die Map spielbar wird und Spaß bereitet. Dazu gehören Texturen zum Kolorieren der Oberflächen, Static Meshes für Detailreichtum und Hintergrundsounds (eigene

Übersetzung). Diese Assets liegen in Packages und werden im Level abgebildet. Wird ein Package gelöscht, fehlen auch alle zugehörigen Assets im Level.“[21]

„Ein Level kann außerdem auch narrative Ziele haben. Es stellt bei vielen Spielen ein Kapitel einer größeren Geschichte dar. Im Level kommen Räume, Charaktere und Ereignisse vor, die einen Teil dieser Geschichte erzählen und dazu einen passenden Ort und eine passende Stimmung bieten. Während der Spieler die Details des Level genießt, hat der Level Designer die Kontrolle über den weiteren Verlauf der Geschichte, die sich in den darauf folgenden Levels abspielt (eigene Übersetzung).“[22]

---

21 Vgl. BUSBY, Jason; PARRISH, Zack; VAN EENWYK, Joel, Level Design, 2005, Maps, S.18: “Maps, or Levels, are the one major asset type not stored in packages. They are the environments in which your game will take place. Technically speaking a map is a collection of many game elements as mentioned earlier. Artistically, however, a map is a medium you can use to evoke different feelings or moods in your players and enhance their gaming experiences. There can never be enough planing for a good map. Don't be afraid to share your ideas for a level with other members of the team or the online community. You will definetly benefit from the experience, opinions, and ideas of others who eventually play your level. A map contains many elements, or assets, all of which are necessary to convey the visual effect you want and to make sure your map is both playable and enjoyable. Some of these elements include textures to „color“ the surfaces of your level, static meshes to add the major physical detail, and sound effects, including ambient noises. The map inserts these assets into your level by referencing them from their corresponding packages. For example, if you delete a texture package that a specific map uses, those textures don't appear the next time the map is played.”

22 Vgl. SALEN, Katie; Zimmerman, Eric *Fundamentals, 2004, Narrative Goals, S.286 f.* “Level or mission-based structures in games also provide important narrative goals for players. Completing a level means not only reaching an objective, but also passing through one episode of a larger story. As the player moves through multiple levels, the succession of completed goals creates narrative coherence. Game levels offer players access to specific areas of the narrative world, each level populated by unique events, objects, and characters that create a particular narrative tone and texture. ... Level or mission structures allow players to feel the details of a story while the game designer maintains control of the larger narrative experience. A game's goal, or series of goals, is part of the narrative context that makes up the game. When goals are well-designed to support narrative play, a player's interaction with the game world becomes consistently meaningful. As usual, the discernability and integration of meaningful play is critical. The elaborate multi-step process of going on a Sims date is only meaningful because of the complex system that supports and links player actions. If every date ended the same way no matter what actions the player took, there would be no reason for the player to engage deeply in

Egoshooter wie Half-Life oder Bioshock sind sehr auf die Handlung fokussiert. Der Spieler muss seinen Weg durch große Industrieanlagen und Gebäudekomplexe finden. Dabei erlebt er diverse Ereignisse, die sich über viele Level hinweg zu einer Geschichte zusammenfügen. Bei beiden Spielen beginnt die Geschichte mit einer spannenden Einleitung, in der die Charaktere eingeführt werden und der Spieler die Steuerung und das Benutzerinterface kennenlernt. Danach beginnt das eigentliche Spiel. Die einzelnen Level sind dabei räumliche Aufteilungen, wie Krankenhäuser, Forschungsstationen, Außenareale usw. Innerhalb der Levels gibt es Zwischensequenzen, Minispiele, Rätsel und versteckte Items. Am Ende eines Levels gibt es manchmal einen Endgegner. Am Ende des Spiels hat auch die Geschichte ihr Ende und der Spieler erfährt in Half-Life in einer Zwischensequenz weitere Informationen über die Welt und seine eigene Zukunft, also eine mögliche Fortsetzung. In Bioshock gibt es zur Belohnung eine Rendersequenz, die dem Spieler zeigt, was aufgrund seiner Entscheidungen im Spiel in der Welt passiert.

Ein Level ist aber nicht zwingend als eine räumliche oder erzählerische Aufteilung des Spiels zu betrachten. Bei Tetris wird nur die Spielgeschwindigkeit pro Level erhöht. Ein Level kann außerdem auch die Entwicklungsstufe eines Spielercharakters in einem Rollenspiel sein oder, wie im Englischen, ein Höhenunterschied sein. Hier ist ein Level also als Spielfeld zu verstehen.

Im Zusammenhang mit dieser Arbeit, hat das Level keinen Handlungsstrang den es zu verfolgen gilt. Im (Team) Death Match-Modus gibt es keine Geschichte, die erzählt werden muss. Es geht einfach darum den Gegner zu besiegen, wie beim Schach. Anders als beim Schach, kann die Map jede mögliche Form annehmen. Nur die Regeln für das Gameplay bleiben immer gleich. Der Schauplatz selber kann jedoch eine Geschichte erzählen, um zur Stimmung beizutragen. Ein alter Shaolin-Tempel sollte entsprechend verbraucht und abgenutzt wirken. Bestimmte Gegenstände, Texturen und Sounds können dem Spieler vermitteln, dass in einem

---

> the decision-making process. Because each step of the process plays a role in determining the outcome, the experience of a Sims date provides genuinely meaningful narrative play."

Raum eine Falle ist, oder dass ein anderer Raum einen Dojo darstellt. Es gibt einen kulturellen Hintergrund. Bei einer Custom Map hat man die freie Wahl, was als Austragungsort gewählt wird. Manche Modder bauen Levels aus anderen Spielen nach. Andere bauen Maps nach einem eigenen Thema. Das Level kann durchaus surrealistische Züge annehmen, was manchmal sogar von Vorteil ist, wenn es das Gameplay fördert.

Gameplay und optischer Gesamteindruck sind die beiden wichtigsten Faktoren bei Custom Maps, wobei das Gameplay zu bevorzugen ist, wenn die Map über längere Zeit im Internet gespielt werden soll. Gute Grafik zeigt technisches Know How und ist gut für Bewerbungszwecke. Es gibt viele verschiedene Arten von Custom Maps. Die Warcraft 3 Fun Maps sind beispielsweise kleine Maps, in denen die Spieler immer unterschiedliche Aufgaben erfüllen müssen, die absolut gar nichts mit dem eigentlichen Spiel zu tun haben, außer dass alle Assets aus dem Spiel übernommen wurden. In Counter-Strike gibt es Maps, die optisch sehr schlicht gehalten sind, aber dafür taktisch sehr gut ausbalanciert sind und den Spielern viele Möglichkeiten zum Verstecken, Heranpirschen und Scharfschießen bieten.

### 1.4.4 Modding

Als Modding wird jegliche Art von Veränderung an einem Computerspiel bezeichnet. Das beinhaltet Veränderungen am eigentlichen Spiel gleichermaßen wie neue Spielmodi im Mehrspielerpart.

Eine Modifikation am Spiel kann ein neues Level sein, die Veränderung von Texturen, Game Art, neue Charaktere, Verbesserung der Grafik, neue Features im Spiel und die komplette Veränderung des eigentlichen Spiels zu einem komplett neuen Spielerlebnis.

Eine Modifikation ist niemals ohne das originale Spiel lauffähig und kann nicht eigenständig verkauft werden. Counter-Strike ist die bekannteste Modifikation überhaupt und ist nur im Laden käuflich, weil die Entwickler von Half-Life den Mod und Ihre Entwickler aufgekauft haben.

Modifikationen sind sehr beliebt. Die Anzahl der Einsendungen beim MSUC ist ein Beweis dafür. Websites wie http://www.moddb.com[23] zeigen, dass viele Menschen Mods erstellen und spielen. Ganze Teams von Moddern setzen sich zusammen, um neue Mods zu erfinden. Firmen wie Epic Games empfehlen allen, die einen Einstieg in die Spieleindustrie suchen, so viel zu modifizieren, wie es geht. Durch die Veränderungen der Programme erhält man nämlich tiefe Einblicke in die Workflows und Bedienung der Entwickler Tools, sowie in den Code und die Algorithmen, sofern möglich. Außerdem bekommt man Übung im Umgang mit den Entwickler-Tools und den notwendigen Softwares.

### 1.4.5 Game Art

Game Art ist der künstlerisch praktische Teil der Spielentwicklung. Dazu gehören Concept Art, Texturierung, Modellierung von Charakteren, Fahrzeugen und der Spielumgebung, sowie Items, Machinima. Ein Großteil der Zeit für die Erstellung eines Levels wird mit der Erschaffung von Game Art verbracht.

Bei großen Firmen gibt es für jede Game Art einen Spezialisten:

- Konzept: Concept Artist
- Charakter: Character Designer
- Vehikel: Vehicle Designer
- Spielumgebung: Environmental Artist
- Texturen: Texture Designer
- Animation, Rigging, Skinning: Animator

Die einzelnen Bereiche für die Erstellung von 2D Konzepten und für die 3D-Modellierung können auch wiederum arbeitsteilig geregelt werden, sodass ein Asset im Spiel vorher durch viele verschiedene Hände gehen muss. Die einzelnen

[23] Eine Datenbank in der Modder ihre Arbeit weltweit zum Download bereitstellen.

Spezialisierungen und auch die Berufsbezeichnungen variieren von Unternehmen zu Unternehmen.

Auf 3D-Objekte muss an dieser Stelle gesondert eingegangen werden, denn für jede Game Engine müssen die Texturen und Modelle vorher definierte Formate haben und einen langwierigen Arbeitsablauf hinter sich bringen, um am Ende auch optimal im Spiel zu funktionieren. Fachmänner, die diese Aufgaben übernehmen, nennt man Technical Artists.

## 1.5 Level Design Workflows

In diesem Kapitel werden zwei verschiedene Workflows vorgestellt. Der erste Workflow ist der typische Workflow bei Epic Games, der beschreibt wie die Designer bei der Entwicklung von Unreal Tournament vorgegangen sind. Der zweite Workflow ist der eigene, eher traditionelle Workflow für dieses Level. Außerdem ist es kein komplett neues Level, sondern ein Level das erstellt wird nachdem das Spiel verkauft wurde. Man kann es also als Modifikation des Spiels bezeichnen. Der Workflow von Epic Games ist sehr auf den Spielspaß fokussiert. Die Entwickler Spielen auf der Map von Anfang an. Der zweite Workflow basiert auf Planung und Umsetzung von Geschichten.

Folgende Rahmenbedingungen sind für die Custom Map in UT3 gegeben:

- Genre: Egoshooter
- Spielmodus: Deathmatch und Teamdeathmatch. Ziel ist also die höchste Anzahl an Kills innerhalb von einem vorher bestimmten Zeitraum zu erkämpfen.
- Anzahl Spieler: 2 bis 48 Spieler sind theoretisch möglich.

Spielregeln, Interface, Story und sehr viel Content sind bereits mit dem Spiel mitgeliefert, weshalb es möglich ist eine Custom Map in einem kürzeren Zeitraum zu erschaffen, als eine komplett neue Map. Der selbst erstellte Inhalt dient zur Erweiterung des schon bestehenden Inhalts für bessere Atmosphäre, für besondere Aufgaben/Hindernisse im Level, sodass ein einzigartiges Spielerlebnis entsteht und zum Erlernen neuer Techniken.

### 1.5.1 Level Design bei Epic Games

Das Level auf den folgenden Screenshots ist bereits im ersten Teil der Unreal-Reihe erschienen und war immer beliebt. Es ist übersichtlich, bietet Plätze für Scharfschützen, hat eine spezielle Waffe die versteckt ist und war damals schon gut ausbalanciert. Das heißt, dass nicht nur die Waffen gut verteilt waren, sondern auch dass die räumliche Einteilung gut gelungen ist. Es gibt eine große Halle mit mehreren Rampen. Darum herum sind weitere Flure, in denen der Spieler viele

Gegenstände finden kann. Es gibt auch ein Kellergeschoss, welches hauptsächlich ein gut, aber gefährliches, Säurebad und außerdem auch den Zugang zur Spezialwaffe beinhaltet. Die Säure selbst ist unter den Übergängen platziert, sodass die Spieler dort hineinfallen können. Man muss sich also geschickt anstellen, um nicht selber hineinzufallen oder vom Gegner hinein befördert zu werden.

Der Ort ist im Stil eines Industriegebäudes gehalten, das der Liandri Mining Corporation aus der Unreal Geschichte gehört. Der Anspruch einer realitätsgetreuen Darstellung einer Fabrikanlage wurde zugunsten des Spielspaßes hintan gestellt. Oft kann man auch nicht sagen, ob das Level eine extra erbaute Arena für den Wettkampf sein soll oder ein zufälliger Schauplatz an dem das Turnier stattfindet. Dieses Level ist eher eine Arena.

„Normalerweise startet man mit einer Idee wie z.B. einem Ort, einem Thema, einer Falle in die der Spieler tritt oder einer anderen Inspiration. Von da an wird das Grundgerüst des Levels gebaut (Shelling it out). Nur ebene Flächen, ohne Material und Detail, werden verwendet. Dabei sind bereits Überlegungen anzustellen, wo die Stärken und Schwächen des Levels liegen und wie der Spielspaß maximiert werden kann. Es wird unter anderem geplant, wo Waffen und weitere Gegenstände platziert werden sollen. Durch geschickte Planung entstehen später stellenweise besonders umkämpfte Gebiete auf der Map.

Bei Unreal Tournament 3 kommt es häufig vor, dass gescriptete Ereignisse dem Spieler gewisse Anreize und taktische Vorteile geben. Es gibt eine Map, in der alle paar Minuten ein Raumschiff landet, welches ein Vehikel absetzt. Das Vehikel bietet durch die Panzerung und Feuerkraft große Vorteile gegenüber dem Gegner, weshalb alle Spieler immer auf das Raumschiff warten und der Landeplatz hart umkämpft ist.

„Sobald diese Grundlagen fertig sind, wird das Level getestet.

*Abbildung 1.3: Das Grundgerüst eines Levels*

Danach geht das Level zum Art Team, um das Level auszuschmücken und evtl. zum Concept Art Team, welches Screenshots nimmt und diese übermalt. Ein Concept Artist entwirft eine Umgebung und kann für ein Bild unter Umständen eine ganze Arbeitswoche brauchen.

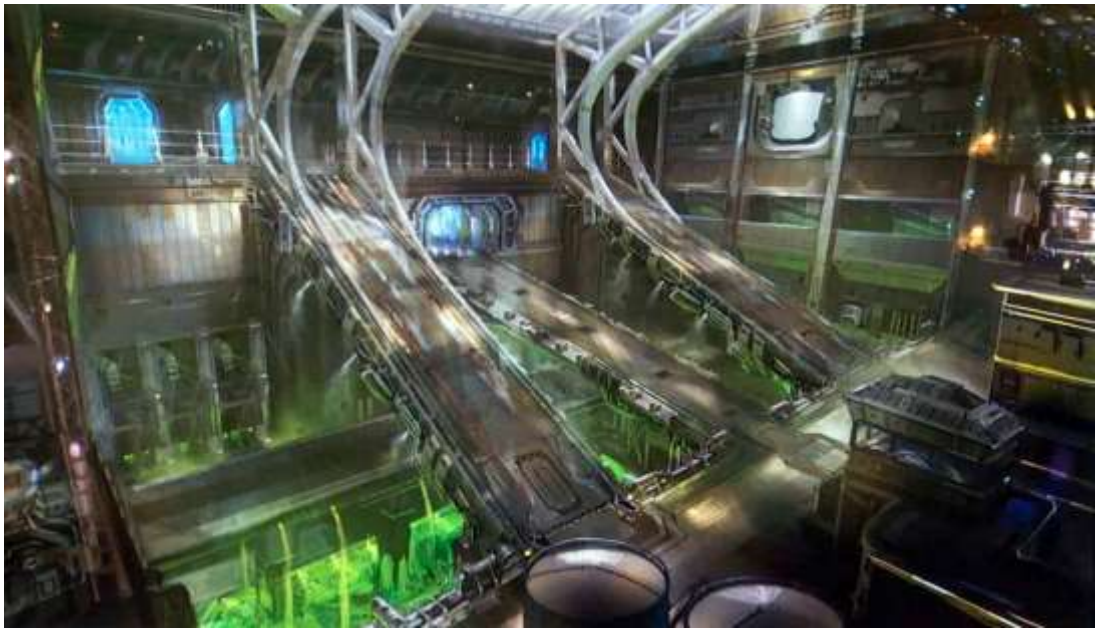


*Abbildung 1.4: Concept Art basierend auf dem obigen Screenshot.*

Anschließend müssen das Art Team und die Environmental Artists die Konzepte umsetzen, was drei Monate Zeit in Anspruch nehmen kann. Unreal Tournament hat

eine sehr hohe Vielfalt an Details und mehrere verschiedene Kulturen an verschiedenen Schauplätzen. Die Erstellung so vieler Objekte, Texturen, Partikeleffekte usw. erfordert eine Menge persönliches Engagement. Dabei ist auch auf technische Aspekte acht zu geben. Das Spiel muss auch flüssig auf möglichst vielen heimischen Computern laufen.“ (eigene Übersetzung)[24]

*Abbildung 1.5: Das fertige Level*

Level Design ist ein sehr organischer Prozess, bei dem sich vom eigentlichen Konzept bis zum fertigen Level viel ändern kann. Wenn man die letzten beiden Bilder miteinander vergleicht, sieht man sofort einige Unterschiede. Die Texturen sind anders als im Konzept und auf der oberen Ebene sind kein Zaun, keine Türen und keine Stahlträger, die bis zur Decke gehen. Anscheinend mussten viele kleine Details weggelassen werden, entweder aus zeitlichen Gründen oder auf Grund der Performance. Das Farbschema ist auch anders. Spieler der Unreal Tournament-Reihe wissen aber, dass das Ergebnis der ursprünglichen Map aus den Vorgängern ähnlicher sieht als das Konzept.

[24] vgl. MORRIS, Jeff, Level Design, 2007, Making_of_UT3_720p.wmv, ab 5:50min

### 1.5.2 Eigener Level Design Workflow

Durch das Spielen von UT3, das Durcharbeiten von Tutorials und die Analyse des Editors, sowie der Levels von UT3, ergab sich die Idee einen Shaolin-Tempel zu bauen.

UT3 enthält bereits zwei Maps, die einen alten asiatischen Stil aufweisen. Viele Lampen, Wände, Statuen und Dächer sind bereits vorhanden, sodass es im Rahmen der Thesis möglich ist, eine Map auf dieser Basis zu bauen.

Auf dieser Grundlage entstehen schnell weitere Ideen für spielerische Details im Level und für neue Assets.

Während die mitgelieferten Tutorials durchgearbeitet werden, wird bereits ein grafisches Konzept erstellt. Der Schauplatz wird visualisiert und in Form von Thumbnails zu Papier gebracht. Einer der Thumbnails wird dann ausgewählt und zu einem fertigen Concept Art ausgearbeitet. Hinzu kommt eine Sammlung von Referenzbildern, die sich durch Recherchen aus dem Internet und aus Fotografien von chinesischen Gärten und traditionell gestalteten Gebäuden ergeben. Die Thumbnails und das fertige Concept Art wurden als Anhang beigefügt.

Durch diese große Sammlung an Material ergibt sich eine klare Vorstellung, wie das Level aussehen kann und wie neue Inhalte, wie japanische Kirschbäume, aussehen müssen. Das Anfertigen der Konzepte wird hier als Erstes abgehandelt, damit nach einer Woche mit der Produktion fortgefahren werden kann. Das fertige Level und die eigentlichen 3D-Modelle haben eine höhere Priorität als ein perfektes Bild. Es ist aber wichtig in den Bildern genug Details zu haben, sodass ein 3D-Designer davon Static Meshes nachbauen kann.

Die Idee des Tempels wurde durch die Tutorials und zahlreiche Filme wie Hero, Last Samurai, Tiger and Dragon usw. beeinflusst:

- Es soll möglich sein, ähnlich wie in Filmen, auf den Dächern und den Bäumen entlang zu laufen, aber es soll Geschick erfordern. So kann ein guter Spieler den taktischen Vorteil nutzen und von oben herunterschießen.

- Eine Drachenstatue soll die beste Rüstung beschützen, indem sie in unregelmäßigen Abständen Feuer spuckt. Der Spieler bekommt so eine Lektion, wenn er ungeschickterweise beim Einsammeln der Rüstung vom Feuer erfasst wird. Er muss daher vorsichtig um das Feuer herum laufen, um zur Rüstung zu gelangen.
- Im ersten Stock des Tempels befindet sich eine Art Dojo, indem eine andere Schwerkraft als in der restlichen Map vorherrscht. Eine weitere Idee, die aus Martial Arts Filmen stammt, in denen die Kämpfer unglaubliche Stunts und Choreografien vorführen.
- Um das Spiel zu beschleunigen und mehr Action herbeizuführen, werden Teleporter und Jumppads (Sprungbretter) eingefügt. Dadurch wird das Spielverhalten auch gleich etwas mehr an Quake 3 Arena erinnern; ein Spiel, welches in der Szene auch sehr bekannt ist.
- Ein besonderer Gegenstand befindet sich auf dem Dach. Das Dach ist nur über Jumpboots erreichbar, die alle paar Minuten auf der Map erscheinen. Es gibt immer Spieler, die alles in einem Spiel entdecken und jeden Winkel erforschen möchten. Jemand der es auf das Dach schafft, wird durch einen besonderen Gegenstand belohnt.
- Da die Map insgesamt eher offen ist, wird das Erdgeschoss etwas verwinkelt, sodass man dort die Flak Canon gezielt einsetzen kann.
- Andere offene Bereiche werden durch Bäume und weitere Objekte verdeckt.

Um mit einer eigenen Map aufzufallen, sollen neue Inhalte erstellt werden.

- Japanische Kirschbäume, weil sie gut aussehen.
- Glühwürmchen, um eine romantische Stimmung zu erzeugen
- Bewegliche Lampen, die einen Schatten werfen. Man hat Sie bisher nur in der Tech Demo der Unreal Engine gesehen, aber nie im Spiel.
- Es soll auch viele verschiedene Farben mit Hilfe von unterschiedlichen Materialien und Texturen geben, um dem Level einen sehr fantasiereichen Look zu geben und eine sehr romantische

und ruhige Hintergrundstimmung zu erzeugen. Der Schauplatz ist schließlich ein religiöser Tempel.

- Hinzu kommen weitere Objekte wie Wooden Dumy-Figuren, und kleine Drachenstatuen oder Schriftrollen.

Sobald das Level spielbar ist, wird es getestet, damit Framerates und Balancing überprüft werden können. Zuerst Alpha Tests, nur mit Bots, und dann Beta Tests mit menschlichen Gegnern. Die Spieler können dann Feedback geben und werden zu den oben genannten Stichpunkten genauer befragt. Dazu gehört auch die Überprüfung jedes neuen Contents. Das bedeutet, dass überprüft wird ob alle Texturen und Meshes zu sehen sind und ob auch alle älteren Computer, etwa solche ohne DirectX10-Grafikkarten, das Level richtig darstellen können.

Danach wird festgelegt wie viele Spieler für die Map empfohlen sind und das Level optisch und für das Gameplay optimiert.

# 2. Level Design im Unreal Editor

In diesem Kapitel werden zuerst die Grundlagen der Benutzeroberfläche kurz beschrieben und später tiefergehend eigene Wege für den Umgang mit der Software beim Level Design beschrieben.

## 2.1 Die Grundlagen des Unreal Editors

### 2.1.1 Einführung in die Benutzeroberfläche

Die grafische Benutzeroberfläche (GUI) des Editors ist auf den ersten Blick vergleichbar mit der von anderen 3D-Anwendungen wie Maya.

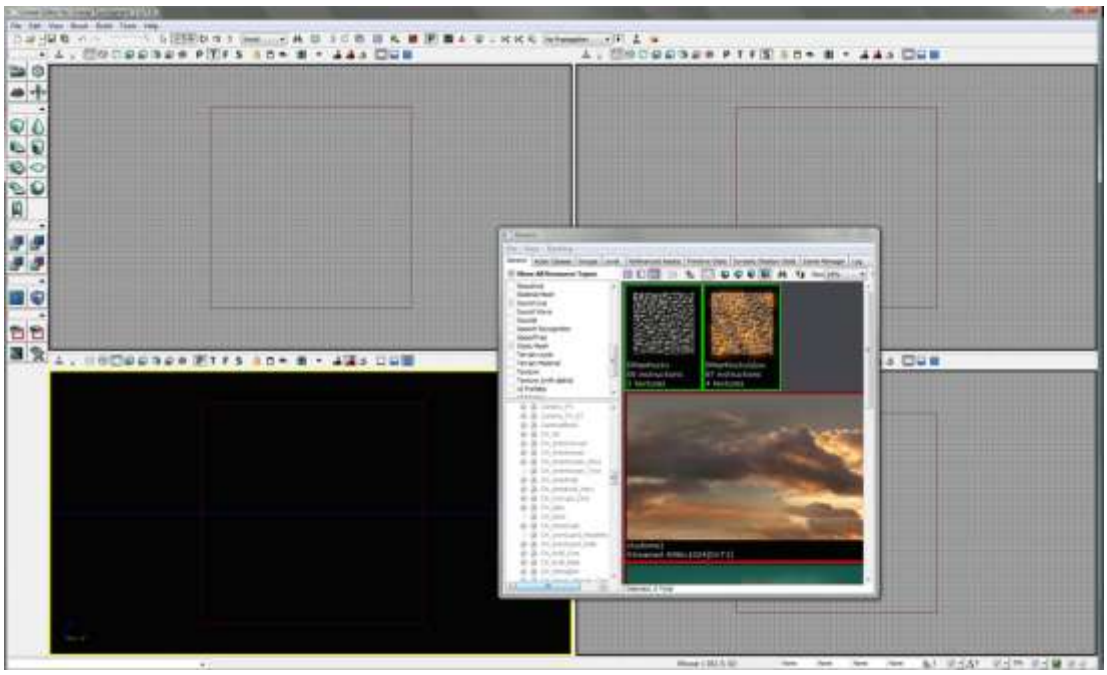


*Abbildung 2.1: GUI des Unreal Editors*

Das Fenster ist aufgeteilt in vier Ansichten: Oben, Unten, Seite und Perspektive. Über jeder Ansicht befinden sich Optionen, um die Darstellung zu ändern. Der Generic Browser, der sich ebenfalls automatisch öffnet, zeigt alle Elemente innerhalb des Editors, die dem Designer zur Verfügung stehen.

Links befindet sich die Werkzeugleiste, die auch wieder in vier Bereiche aufgeteilt ist. Es gibt hier verschiedene Modi zur Erstellung und Bearbeitung der Geometrie innerhalb eines Levels. Die Aufteilung des Levels erfolgt mit Constructive Solid Geometry (CSG) Primitiven und Operationen um mit den Primitiven zu arbeiten.[25]

Oben, unter dem Menü, liegt eine weitere Leiste mit Werkzeugen und Optionen, die auch über das Menü erreicht werden können.

Im File-Menü lässt sich ein neues Level öffnen. Man hat die Wahl zwischen einem additiven und einem subtraktiven Level. Während die additive Methode einen vollkommen leeren Raum erzeugt, in dem neue Spielwelten eingebaut werden, muss man sich bei der subtraktiven Methode einen unendlich großen Raum vorstellen, der mit einer soliden Masse gefüllt ist. Mit Hilfe der Werkzeuge aus der Werkzeugleiste, werden Räume und Korridore von dieser Masse subtrahiert.

Durch Bedienung der Maustasten und des Mausrades wird in den jeweiligen Ansichten navigiert. Um die Kamera nach vorne und hinten zu bewegen, hält man beispielsweise die linke Maustaste gedrückt und bewegt die Maus nach vorne oder hinten.

Unten rechts liegt die Statusleiste, die einem beim Umgang mit CSG und Static Meshes Informationen ausgibt.

## 2.2 BSP

Um einen Raum aufzuteilen, benutzt man sogenannte Brushes und CSG-Operationen. Die einzelnen Buttons dafür findet man in der Werkzeugleiste des Level Editors. In jedem Level existiert von Anfang an der rote Builder Brush, mit dem neue Brushes und Volumen erzeugt werden.

[25] Einen groben Überblick über CSG bietet MEYER, Norman, Game-Engine, 2005, S.69 ff.

Der Brush kann verschiedene Formen und Größen annehmen. Es kann auch eine eigene Form definiert und gespeichert werden. Nachdem der Builder Brush konfiguriert und korrekt im Raum platziert wurde, kann man nun über den CSG: Add-Button Masse in den Raum einfügen. Es gibt außerdem CSG-Subtract, CSG: Intersect und CSG: Deintersect. Intersect bildet die Schnittmenge aus 2 Brushes in einem neuen Brush. Deintersect erzeugt das Gegenteil der Schnittmenge.

*Abbildung 2.2: Die Toolbox*

Der Geometry Modus gehört zu den Werkzeugen in der Werkzeugleiste und ermöglicht es, die einzelnen Polygone und Faces zu manipulieren, genau wie bei der 3D-Modellierung. Das BSP sollte sich, zur schnelleren Lichtberechnung, immer an das Drag Grid halten. Das Drag Grid ist ein Drahtgitter, welches zur Orientierung dient. Bei Transformationen wird das zu bewegende Objekt immer um eine bestimmte Anzahl von Längeneinheiten (Unreal Units) am Gitter entlang bewegt. Die Anzahl der Einheiten lässt sich verstellen und auch deaktivieren. Das Deaktivieren ist ausdrücklich nicht zu empfehlen.

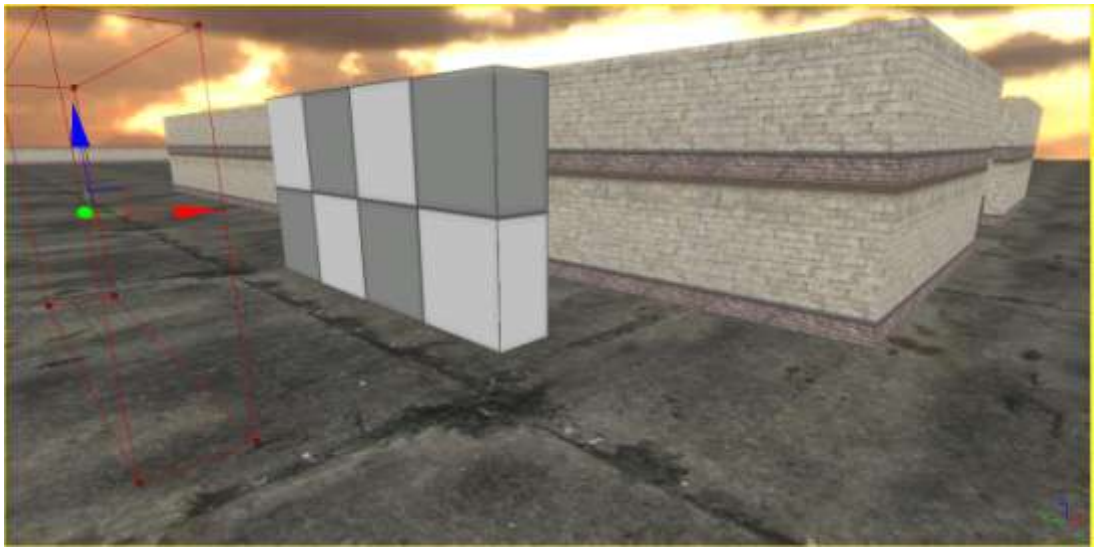


*Abbildung 2.3: Roter Builder Brush, Wand mit No-Material und Wände mit richtigem Material in der Perspektivenansicht*

Neu erstellte Masse hat ein kariertes Muster, das sogenannte No-Material. Jeder Fläche muss noch ein Material hinzugefügt werden. Man kann darauf auch

verzichten, aber es wird eine Warnmeldung produzieren, wenn man das Level so ausliefern will. Dazu später mehr.

Um eine Fläche mit einem Material aus dem Generic Browser zu versehen, werden alle Flächen angewählt und das entsprechende Material im Generic Browser angeklickt. Man muss immer Materialien verwenden, denn Texturen sind nur Bestandteil eines Materials und sollen deshalb nicht einzeln genutzt werden. Position und Größe der Texturen lassen sich über die Taste F5 in den Surface Properties verändern. Dort lässt sich das Material horizontal und Vertikal auf der Oberfläche verschieben, die Größe ändern und die Auflösung der vorkalkulierten Schatten lässt sich verändern.

## 2.3 Static Meshes

Nachdem der Raum aufgeteilt und mit Materialien versehen wurde, kann er mit Static Meshes detailliert werden. Static Meshes sind 3D-Objekte mit eigenen Materialien und werden in externen Anwendungen erstellt. Die Anzahl der Polygone ist eigentlich unbegrenzt, solange die Computer genug Rechenleistung bieten. Sie sollte schätzungsweise nicht über den niedrigen Zehntausenderbereich hinausgehen, es kommt dabei aber auf die Zielplattform an. Das Selbe gilt für Texturgrößen. Die Unreal Engine unterstützt derzeit Texturgrößen von 8192x8192 Pixeln, aber nur weil die aktuellen Grafikkarten nicht mehr verarbeiten können.

Static Meshes lassen sich, ebenso wie Brushes auch, rotieren, transformieren und skalieren. Unten rechts in der Leiste des Level Editors, findet man die Skalierungs-Einstellung für das angewählte Objekt. Sie ist aufgeteilt in vier Bereiche: gesamtes Objekt, x-, y- und z-Achse. So kann man ein Objekt entweder komplett oder nur an einer Achse skalieren oder spiegeln. Die Statusanzeige links neben der Skalierungs-Einstellung zeigt an, wie stark und an welcher Achse ein Objekt manipuliert wird.

Die Positionierung von Static Meshes nimmt sehr viel Zeit beim Level Design in Anspruch. Das Dach eines Hauses ist beispielsweise nicht fertig, sondern muss aus mehreren kleinen Teilen zusammengebaut werden, wie der ganze Rest des Gebäudes auch. Dadurch ergibt sich die Möglichkeit einzelne Objekte sehr oft zu kopieren und

nach eigenem Wunsch zu positionieren. Es entstehen aus den gleichen Objekten einzigartige Umgebungen, ohne dabei die Ressourcen zu sehr in Anspruch zu nehmen. Es ist dabei teilweise sehr wichtig auf das Drag Grid zu achten, da sonst evtl. Lücken entstehen.

### 2.3.1 Die Entstehung eines Hauses

Die Wände und Decken werden mit BSP Brushes erstellt. Löcher für Türen oder Flure erzeugt man mit der CSG: Subtract-Funktion. Der fertige Umriss wird mit Static Meshes ausgestattet, welche manchmal sehr oft kopiert werden müssen. Eines dieser Stücke musste im aktuellen Level schätzungsweise 136 Mal kopiert werden.

Das ist einer der Gründe warum Level Design ein wesentlich aufwändigerer Beruf ist, als sich viele denken mögen, aber die Kopien haben dafür weniger Auswirkung auf die Performance als ein gigantisches Static Mesh, dass aussieht wie ein Haus. Nachdem das Dach fertig ist, werden weitere Details, wie Säulen und Fenster eingefügt.

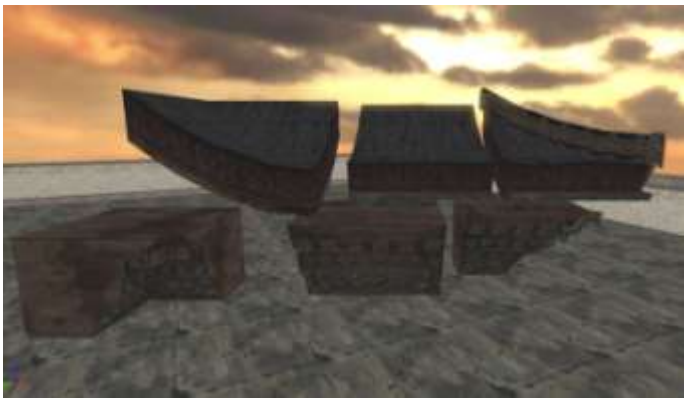


*Abbildung 2.4: Ein realistisch wirkendes Dach wird aus mindestens 6 verschiedenen Teilen*

*Abbildung 2.5: Das weit fortgeschrittene Gebäude*

## 2.4 Generic Browser

Das Generic Browser-Fenster ist in mehrere Karteireiter aufgeteilt. Hier sollen nur die Bestandteile erklärt werden, die für das eigentliche Level verwendet werden.

### 2.4.1 Generic Browser

Der Generic Browser zeigt alle Assets an, die im Unreal Editor zur Verfügung stehen. Ein Asset wird im Editor als Actor bezeichnet. Auf der linken oberen Seite wird ausgewählt, welche Actors angezeigt werden sollen. Darunter werden die Packages dargestellt. Neue Packages können über das Kontextmenü erzeugt werden. Weitere Pakete lassen sich im File-Menü öffnen.

*AbbilAbbildung 2.5: Generic Browser GUI*

### 2.4.2 Actor Classes Browser

Dieser Browser enthält sogenannte Actor Klassen, wie dynamische Lichtquellen, Teleporter oder Kameras. Ein Actor aus diesem Browser ist eine Mischung aus einem Actor und einem Unreal Script. Ein Teleporter ist ein Gegenstand, den man nicht nur sehen und anfassen kann, sondern er enthält ein Skript, welches den Spieler von Punkt a zu Punkt b befördert, wenn er ihn berührt.

Ein Actor wird üblicherweise angewählt und dann über das Kontextmenü im Level platziert. Static Meshes sind auch Actors. Mit F4 öffnen sich die spezifischen Eigenschaften des Actors der gerade angewählt ist.

### 2.4.3 Groups

Mehrere Objekte im Level lassen sich hier gruppieren und können so wie ein einzelnes Objekt behandelt werden.

### 2.4.4 Referenced Assets

Hier werden alle zugehörigen Assets und Packages, die zum gerade im Level angewählten Objekt gehören, angezeigt. Um als Modder ein anderes Level besser zu verstehen, macht man oft Gebrauch von diesem Browser.

### 2.4.5 Primitive Stats und Dynamic Shadow Stats

Unter diesen zwei Karteireitern findet man Tabellen mit Statistiken über Polygonzahlen und kann damit auf den Ressourcenverbrauch schließen. Um herauszufinden wie oft ein Objekt im Level vorkommt und wie komplex dessen Geometrie ist, sucht man im Primitive Stats-Fenster.

### 2.4.6 Log

Das Log-Fenster gibt Statusmeldungen zu jeder durchgeführten Aktion im Editor aus. Jede Aktion wird hier also vermerkt.

# 3 Der Material Editor

## 3.1 Einführung in Materialien mit dem Material Editor

Ein Material ist eine Zusammenstellung von Anweisungen, welche Farbe, Glanz und Unebenheiten von Objekten bestimmen, die vom Spieler gesehen werden. Um ein neues Material zu erstellen, wird mit der rechten Maustaste im Generic Browser das Kontextmenü geöffnet und mit New Material der Material Editor geöffnet. Auf diese Weise wird der Designer gleich aufgefordert das Material zu benennen und in einem Package abzulegen.

Materialien bestehen immer aus Netzwerken von Knoten, die miteinander verbunden sind. Jedes Material hat mindestens einen zentralen Knoten mit festen Kanälen, die alle ihre eigene Funktion haben. Diese Kanäle müssen nicht immer alle verwendet werden, sondern bieten nur Möglichkeiten an, bestimmte Effekte zu erzeugen.

In den Knoten werden sogenannte Material Expressions eingebunden, welche selbst auch wiederum mit weiteren Material Expressions verbunden sein können. Material Expressions können sein: Texturen, Vektoren, Konstanten und Funktionen die es auch in Bildbearbeitungsprogrammen wie Photoshop gibt.

Das folgende Beispiel zeigt einen selbstentwickelten Weg für ein komplexes Material, welches, aus den in 1.5.2 genannten Gründen, in der Map verwendet wird.

## 3.2 Die Kanäle im zentralen Knoten an einem Beispiel erklärt

### 3.2.1 Diffuse

Der Diffuse-Kanal stellt die Eigentlichen Farbwerte eines Materials dar. Es ist möglich einen dreidimensionalen Vektor mit verschiedenen Werten von 0-1 für Rot, Grün und Blau einzuspeisen, oder eine Textur. Das Material bekommt hier eine Steintextur.

*Abbildung 2.6: Diffuse Map*

Das Vorschaubild wird sofort angepasst und das Material auf eine Kugel, einen Würfel oder einen Zylinder gelegt.

### 3.2.1 Emissive

Um dem Material einen Leuchteffekt zu vermitteln, gibt es einen Emissive-Kanal. Es ist möglich einen Vektor mit RGB-Werten zu nehmen, die über den Bereich von 1 hinausgehen. Je höher der Wert desto stärker der Leuchteffekt.

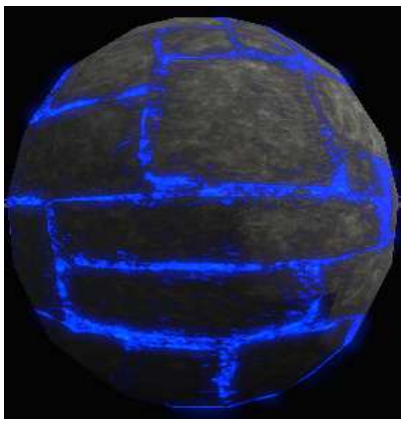


*Abbildung 2.7: Diffuse und Emissive*

Es gibt noch Funktionen, um den Leuchteffekt für bestimmte Bereiche zu verwenden. Der grüne Farbkanal der Textur wird hier dazu verwendet, eine Maske zu erstellen. Der Kontrast der Maske wird erhöht und die Maske am Ende mit der Leuchtfarbe multipliziert. Das Ergebnis wird in den Emissive Kanal geleitet und es entsteht ein interessanter Effekt. Eine Textur kann sogar auch animiert werden, sodass sie sich konstant in eine Richtung bewegt, rotiert oder ein- und ausgeblendet wird. Ein Material kann mit der Zeit sehr komplex werden, jedoch entstehen sehr viele verschiedene Ergebnisse.

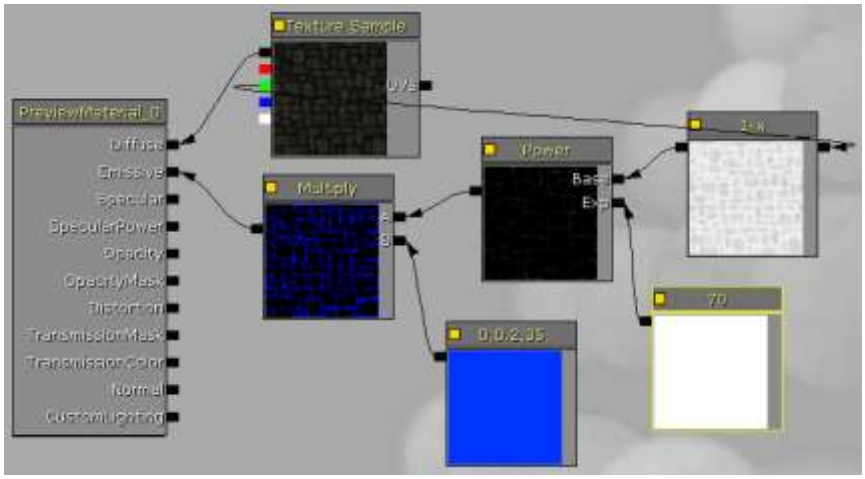



*Abbildung 2.8: Maskierung im Material Editor*

### 3.2.2 Specular und Specular Power

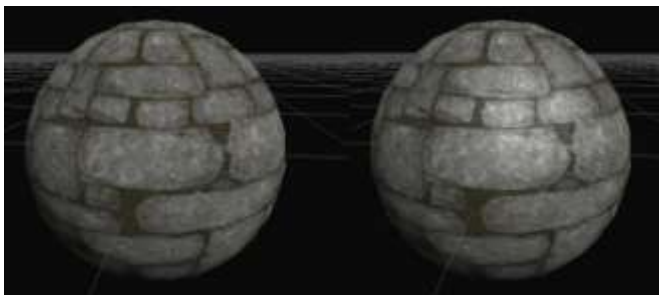


*Abbildung 2.9: Links ohne und rechts mit Speccular Highlight.*

Je nach Beschaffenheit des Materials verändert sich dessen Farbe, wenn Licht darauf scheint. Um diesen Effekt zu simulieren, gibt es den Specular-Kanal, in den eine entsprechende Textur passend zu unserer gehört. Es gibt verschiedene Möglichkeiten so eine Textur zu erstellen, wobei es auf das eigene Augenmaß ankommt. In diesem Beispiel wurde der Grün-Kanal des Materials genommen und der Kontrast erhöht. Zu Anschauungszwecken wurde die Emissionsquelle vorrübergehend wieder entfernt. Der Specular Power-Kanal gibt an, wie stark die Oberfläche glänzen soll. Das Licht im Material Editor erzeugt einen Glanzpunkt, sobald der Kanal benutzt wird. Für dieses Material wird eine Konstante verwendet. Je höher der Wert ist, desto kleiner der Glanzpunkt. Da das Material hier hauptsächlich Stein darstellen soll wird ein großer Glanzpunkt gewählt.

### 3.2.3 Normal

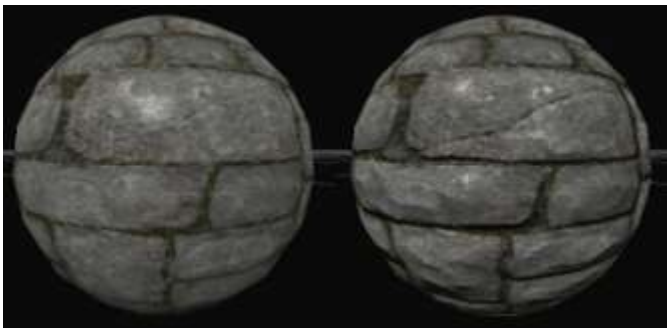


*Abbildung 2.10: Links ohne und rechts mit Normal Map.*

Normal Maps sind heutzutage gängige Methoden, um einem Objekt Unebenheiten und Details zu geben. Dieses Feature ist eine der wichtigsten Neuerung in der Unreal Engine 3. Die Unebenheiten existieren nicht wirklich in Form von Polygonen, sondern werden dem Spieler nur durch Schatten suggeriert. Das Mesh wird also nicht verformt, sondern Schatten in Abhängigkeit vom Standpunkt des Spielers zum Objekt dargestellt.

Mit Normal Maps lassen sich sehr viele Details darstellen, ohne die Rechenleistung zu stark zu beanspruchen. Während die älteren Bump Maps Graustufenbilder sind und nur Abstufungen in der Höhe darstellen, sind Normal Maps Bilder, deren Farben Rot, Grün und Blau den Verschiebungsgrad auf den 3 Raumachsen darstellen. Die Stärke der Verschiebung reicht dabei aber nur für Details wie Muskeln oder Fugen in Steinmauern.

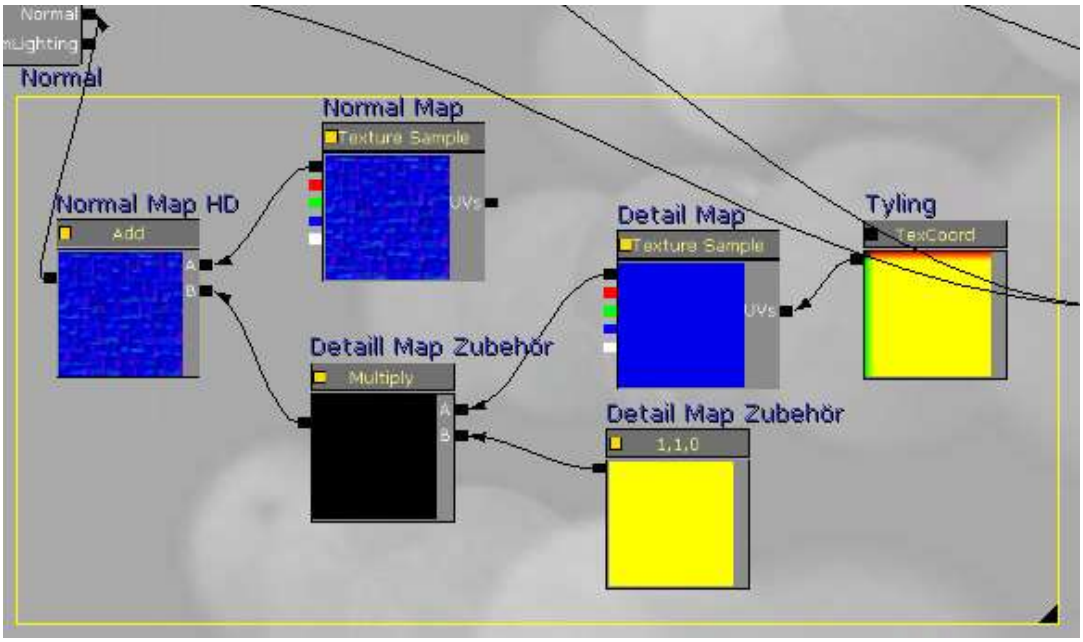



*Abbildung 2.11: High Detail Normal Maps.*

Mit Material Expressions können nochmal mehr Details eingefügt werden, indem man eine kleine Normal Map in gekachelter Form mit der eigentlichen Normal Map multipliziert und die Ergebnisse auf Werte zwischen 0 und 1 normalisiert, wobei 0 der minimale und 1 der maximale Verschiebungsgrad ist. Die daraus entstehende High Detail Normal Map, erzeugt auf dem Material weitere kleine Kratzer, Risse und Unebenheiten, die der Spieler erst sieht, wenn er sehr nah rangeht.

Ein weiterer Effekt der Normal Map ist die Reaktion des Specular Highlights auf die einzelnen Details, was das Material sehr realistisch wirken lässt und erst in bewegten Bildern gut zu sehen ist.

## 3.3 Weitere Kanäle

### 3.3.1 Opacity und Opacity Mask

Um bestimmte Flächen oder Objekte transparent zu machen, gibt es den Opacity-Kanal und einen entsprechenden Kanal um Bereiche für diesen Zweck zu maskieren.

### 3.3.2 Transmission Mask und Transmission Color

Diese Kanäle sind eine weitere große Neuerung in der Unreal Engine 3. Sie simulieren das Sub Surface Scattering. Sub Surface Scattering entsteht, wenn Licht auf ein Material trifft, das lichtdurchlässig ist. Einzelne Photonen dringen ein und prallen ein- oder mehrmals an Molekülen ab (Scattering), um dann wieder aus dem Material auszustrahlen. Die abprallenden Lichtstrahlen können auch wieder auf der Eintrittsseite austreten. Ein gutes Beispiel ist Sonnenlicht, das durch die Blätter eines Baumes strahlt oder ein Mensch der eine Taschenlampe vor seine Hand hält. Um diesen Effekt in Echtzeit vorzutäuschen, verwendet ein Designer diese Kanäle.

Der rote Farbton des Blutes sorgt für ein rotes Leuchten, wenn ein Mensch eine Taschenlampe direkt vor seine Hand, Nase oder an die Ohren hält. Dementsprechend erstellt man ein Graustufenbild, das diese Bereiche maskiert und eine weitere Textur die den Farbton darstellt.

Da dieser Effekt bei sehr vielen Materialien entsteht und nicht nur bei menschlicher Haut, ist er auch sehr wichtig, um ein realistische Ergebnis zu erzeugen.

### 3.3.3 Distortion

In Unreal Tournament 3 gibt es einige Beispiele, die zeigen worum es sich beim Distortion-Effekt handelt. Man kann beispielsweise einen Effekt erzeugen, der an heißen Sommertagen auf Asphalt zu sehen ist. Es ist möglich eine Normal Map zu nehmen und in den Distortion Kanal zu führen. Wenn das Objekt transparent ist, sieht man eine Verzerrung. Die Rot- und Grün- Werte verschieben die eigentlichen Farbpixel horizontal und vertikal auf der Oberfläche.

### 3.3.4 Custom Lighting

Das eigentliche Beleuchtungs-Modell in der Unreal Engine ist das sogenannte Phong-Modell, welches heutzutage in Game-Engines weit verbreitet ist. Der Custom Lighting Kanal bietet die Möglichkeit ein anderes Modell oder gar keins zu verwenden und über eine Textur eine Light Map hinzuzufügen. Diese Light Map täuscht die Beleuchtung des Materials vor. In der Unreal Engine 3 wird bis jetzt das statische Licht nach Fertigstellung des Levels einmalig berechnet und in mehreren Texturen gespeichert, sodass das Licht von allen statischen Lichtquellen nicht mehr in Echtzeit berechnet werden muss. Dazu aber später mehr (Kapitel 6.4.3 S.59) . Diese Texturen können auch aus externen Anwendungen stammen und müssen in diesen Kanal eingespeist werden.

## 3.4 Fallback Materials

Einige Materialien werden so komplex, dass sie nur von den modernsten Grafikkarten dargestellt werden können. Der Material Editor gibt eine Warnmeldung aus, welche Kanäle zu komplexe Material Expressions haben und somit bei alten Grafikkarten wegfallen würden. Für ältere Grafikkarten muss deshalb ein Fallback

Material bereitgestellt werden, welches dann anstelle des eigentlichen Materials verwendet wird.

# 4 Partikelsysteme in Unreal Cascade

„Partikelsysteme bestehen aus Partikeln und Emittern. Ein Partikel ist in Unreal ein Punkt im Raum, der animiert werden kann. Partikel sind immer mit Sprites behaftet, damit man sie sehen kann. Ein Sprite ist hierbei eine Fläche, manchmal ein Mesh, die immer zur Kamera ausgerichtet ist, sodass der Spieler sie sehen kann. Die Fläche ist mit einem Material behaftet, das auch wieder animiert werden kann.

Die Partikel werden von einem Emitter ausgestrahlt, wobei ein Emitter verschiedene Modulatoren hat, die das Verhalten der einzelnen Partikel beeinflussen. Ein Modulator steht beispielsweise für Beschleunigung, Farbe, Flugbahn usw..

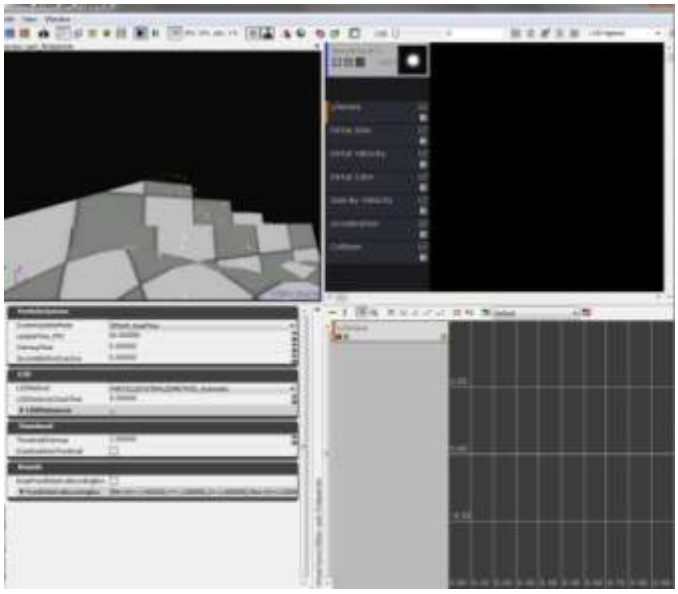
Eine Sammlung von verschiedenen Emittern bildet ein Partikelsystem. Ein Partikelsystem kann für diverse Effekte benutzt werden, wie Feuer, Rauch, Regen und ähnliches.“[26]

## 4.1 Erstellung eines Partikelsystems am Beispiel

Wie beim Material Editor wird Cascade über das Kontextmenü des Generic Browsers geöffnet. Cascade ist in vier Bereiche aufgeteilt. Oben links befindet sich das Vorschaufenster, oben rechts die einzelnen Emitter und Module. Unten links erscheinen immer die jeweiligen Eigenschaften der einzelnen Module oder der ausgewählten Emitter und unten rechts ist das Fenster mit dem Curve Editor, der immer dann verwendet werden kann, wenn in den einzelnen Modulen mit Farben oder Koordinaten gearbeitet wird. Es soll nun als Beispiel ein Schwarm Glühwürmchen kreiert werden, der bereits in 1.4.2 erwähnt wurde.

[26] vgl. BUSBY, Jason; PARRISH, Zack; VAN EENWYK, Joel, Partikel, 2007, 001_Cascade_Overview.wmv, ab 01:15min

*Abbildung 4.1: Cascade Benutzeroberfläche*



Der erste Schritt ist die Erzeugung eines neuen Emitters mit einem Namen und einem Material für die Partikel. Das Material wird aus dem Generic Browser gewählt und in den Eigenschaften hinzugefügt. Die Glühwürmchen selbst sind kleine Punkte, die am Rand nach außen hin in die Transparenz verlaufen. Weitere wichtige Grundeinstellung des Emitters sind Lebensdauer und Anzahl der Partikel, die ausgestrahlt werden. Es sollen immer 10 Glühwürmchen innerhalb von 5 bis 10 Sekunden erscheinen.

Jetzt hat man die Aufgabe, diese Punkte wie umherschwirrende Insekten wirken zu lassen. Dazu gibt es eine Vielzahl von Modulatoren, die einem Emitter zugefügt werden können. Jeder Partikel bekommt eine Lebenszeit von 10 bis 20 Sekunden, eine Größe und eine Geschwindigkeit. Zu jeder dieser Eigenschaften gibt es einen Minimal- und einen Maximalwert, sodass viel Variation entsteht.

Der Emitter bekommt weitere Modulatoren, die dafür sorgen, dass die Partikel innerhalb von einem festgelegten Raum erscheinen und auch innerhalb dieses Bereiches bleiben. In dem Bereich können sie sich frei bewegen oder sich an einer der Raumachsen orientieren und in Kreisen fliegen.

Weitere Modulatoren dienen dazu, noch realistischeres Verhalten und ein noch schöneres Ergebnis zu erzielen. Farbe und Größe lassen sich über die Lebensspanne eines Partikels hinweg anpassen. Für Farbe und Größe werden Vektoren mit beliebig vielen Punkten erzeugt und von Punkt zu Punkt über die Lebensspanne interpoliert. Die einzelnen Punkte und die Interpolation lassen sich als Kurven im Curve Editor anzeigen und anpassen. Ein Modulator sorgt dafür, dass sich die Partikel gegenseitig anziehen, wenn sie in einem bestimmten Radius aneinander vorbeifliegen. Dadurch

hat man am Ende ein Partikelsystem, dessen Partikel sehr realistische Flugbahnen aufweisen. Die Glühwürmchen wirken wie aus einem Fantasy-Film.

## 4.2 Ein Modulator im Detail

Der Modulator Color Over Live verändert die Farbe eines Partikels über die Lebenszeit. Dabei kann man so viele Farben verwenden, wie man möchte. Die einzelnen Farben werden als Koordinaten in einem Vektor gespeichert, wobei die einzelnen Farbanteile Rot, Grün und Blau als Werte als X, Y und Z gespeichert werden.

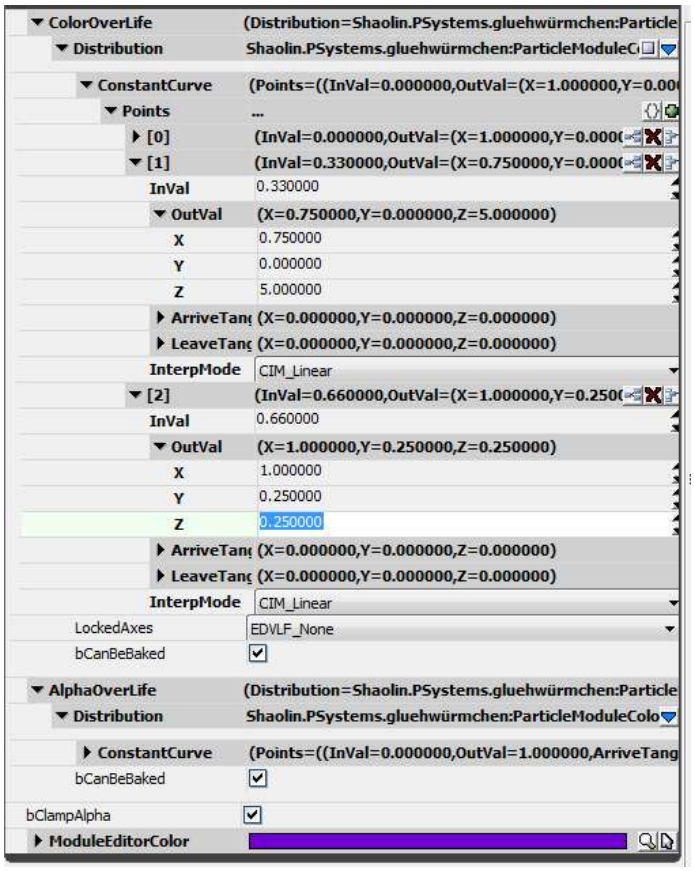



*Abbildung 4.3: Die Einstellungen des ColorOverLife-Modulators*

Zwischen den einzelnen Farben wird interpoliert, wenn man es möchte und wie man es möchte. Die Lebensphase wird in Werte zwischen 0 und 1 aufgeteilt, sodass jeder Farbwert in diesem Wertebereich positioniert werden muss.

Jeder Modulator hat seine speziellen Eigenschaften, aber sie sind im Aufbau ähnlich.

Distribution steht für die Art der Funktion, die mit den Punkten durchgeführt wird. In diesem Fall ist es VectorConstantCurve und das bedeutet, dass die einzelnen Punkte zu einer Kurve verbunden werden. Diese Kurve wird in der Lebenszeit eines Partikels durchlaufen, wobei jede Koordinate eine Farbe darstellt. In diesem Beispiel wurden drei Punkte eingetragen, von denen jeder einen Eingangswert InVal und einen Ausgangswert OutVal hat. Der OutVal ist die eigentliche Farbe. Der Inval ist der Zeitpunkt.

Eigentlich geht der Wertebereich im RGB- Farbraum jeweils von 0-255, aber auch hier wird der Bereich auf Werte zwischen 0 und 1 normiert.

Mit den Werten Arrive Tangent und Leave Tangent kann die Kurve vor und nach dem jeweiligen Punkt weiter angepasst werden.

Die Option InterpMode bietet die Möglichkeit, die Art der Interpolation zu verändern, und mit LockedAxes kann man einzelne Achsen deaktivieren, sodass beispielsweise nur die Y- und Z-Werte verwendet werden.

Ein weiterer Distributions-Typ, der noch unbedingt erwähnt werden muss, ist der Typ VectorParticleParameter, der in Unreal Caskade beeinflusst werden kann. So kann später über gescriptete Ereignisse das Partikelsystem zusätzlich verändert werden.

## 4.3 Curve Editor

Der Curve Editor kommt mehrfach im Unreal Editor vor. Er ist kein eigenständiger Editor und hat nur die Funktion Kurven darzustellen und zu bearbeiten. Da er aber oft verwendet wird, wird er kurz am Beispiel beschrieben.

Die drei Punkte, die im Modulator ColorOverLife eingestellt wurden, können im Curve Editor grafisch dargestellt werden.

Die einzelnen Werte für X, Y und Z im Modulator können ein- und ausgeblendet werden. Die x-Achse im Editor repräsentiert die Zeit und die y-Achse den Wert, den der Vektor annimmt.

Die Kurven nehmen sogar die jeweiligen Farbwerte an. Hier kann man nun die einzelnen Kurven und Punkte auf intuitive Weise mit der Maus anpassen.

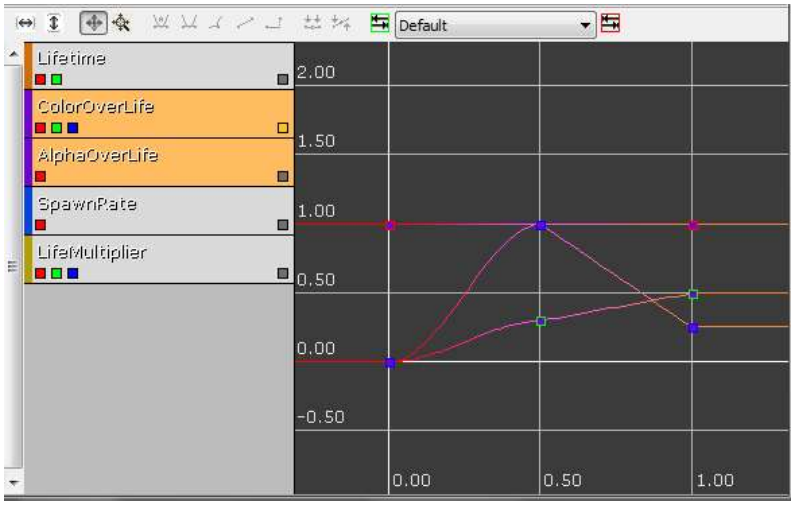



*Abbildung 4.4: Die GUI des Curve Editors*

# 5 Visual Scripting in Unreal Kismet

Jeder der schon einmal eine Script- oder Programmiersprache verwendet hat, wird sich sehr leicht in Kismet zurecht finden. Kismet ist der Visual Scripting Editor im Unreal Editor. Visual Scripting ist das Selbe wie normales Scripting nur ohne Syntax, dafür mit dem Aufbau eines Flussdiagramms. Der logische Aufbau mit Algorithmen und Methoden bleibt weiterhin bestehen.

Objekte aus dem Level Editor können in Kismet genutzt werden, um Ereignisse vorherzubestimmen, wobei man fast alles machen kann, was sich ein Game Designer vorstellen kann. Für das Level, welches im Rahmen dieser Arbeit erstellt wurde, wurde eine feuerspeiende Drachenstatue programmiert. Sie wurde bereits in Kapitel 1.5.2 beschrieben. Das Feuer wird alle 5 bis 15 Sekunden zufällig aktiviert und nach 5 Sekunden wieder deaktiviert. Hinzu kommen ein paar Post Processing Effekte, die auftauchen, wenn man dem Feuer zu nahe kommt. Außerdem kann sich der Spieler an dem Feuer verbrennen, d.h. er bekommt Lebenspunkte abgezogen, wenn er zu nah ran geht. Dieses Ereignis wird mit dem Level gestartet und wiederholt sich in einer Endlosschleife.


*Abbildung 5.1: Der Drache beschützt die Rüstung, die vor ihm liegt, indem er alle 5 bis 15 Sekunden Feuer speit.*

Kismet wird vom Level Editor aus geöffnet, indem man auf einen Button in der oberen Leiste klickt. Das Fenster für Kismet ist sehr übersichtlich und scheint auf den ersten Blick leichter verständlich als die vorangegangenen Editoren, da alle Variablen und Funktionen und Objekte über das Kontextmenü geöffnet werden.

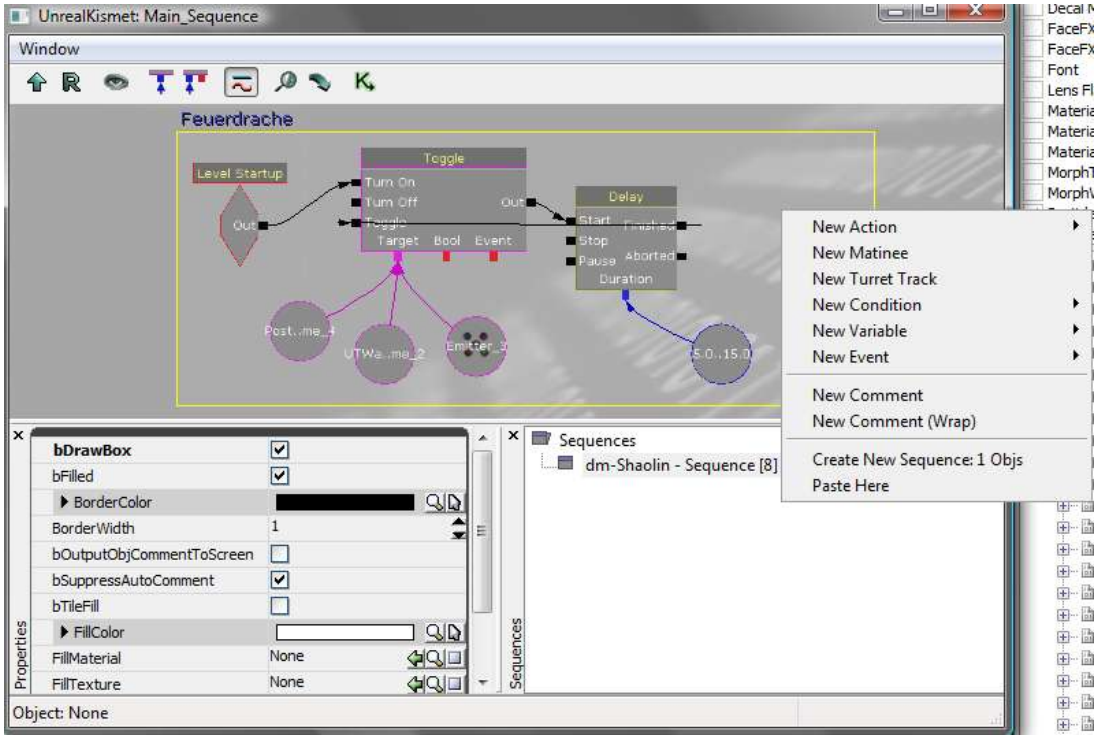



*Abbildung 5.2: Die Benutzeroberfläche von Kismet*

## 5.1 Der Aufbau einer Sequenz

Zunächst muss man sich in chronologischer Reihenfolge den Ablauf der Sequenz vorstellen. Als erstes muss die Sequenz gestartet werden. Es gibt verschiedene Auslöser, wie Trigger (Knöpfe), Trigger Volumes (das Betreten eines Raumes) und hier ist es einfach ein sogenanntes Event namens Level Startup, das, wie der Name schon sagt, die Sequenz startet, sobald das Level startet..

Nachdem das Level gestartet wurde, soll das Feuer aktiviert werden. Das Feuer ist eine Objektvariable, die man über das Kontextmenü einfügen kann, während man das Partikelsystem im Level Editor angewählt hat. Das Feuer muss mit einer Action aktiviert werden.

Die Toggle Action ermöglicht es, das Objekt an- und auszuschalten, je nachdem in welchem Zustand es sich gerade befindet. Der Ausgang des Level Startup-Events wird, da das Feuer noch aus ist, in den Turn On- Kanal der Toggle-Action geführt und als Target (Ziel) der Toggle-Action das Feuer eingesetzt.

Nachdem das Feuer aktiviert wurde, soll es für eine Zeit von 5 bis 15 Sekunden brennen. Dafür verwendet man einen Delay (Verzögerung). Der Delay hat einen Kanal für die Dauer der Verzögerung, in den eine Float Variable eingebunden werden muss.

Die Einstellungen der Float Variable erscheinen unten links in Kismet, sobald die Variable angewählt ist. Innerhalb dieser Einstellungen kann ein Minimal- und ein Maximalwert gewählt werden, sodass ein Zufallswert innerhalb des Bereiches gewählt wird. Außerdem kann man auch eine Abstufung von x Sekunden einstellen, sodass beispielsweise immer ein Vielfaches von 5 Sekunden als Verzögerung genommen wird.

Nachdem die Verzögerungszeit abgelaufen ist, kann der Ausgang vom Delay wieder mit dem Toggle-Eingang der Toggle-Aktion verbunden werden.

## 5.2 Matinee

Eine Matinee-Sequenz wird in der Map für die animierten Holzfiguren verwendet, die den Zugang zu einem PowerUp erschweren sollen. Dabei wird die Matinee-Sequenz vor und zurück abgespielt. Die Skript-Sequenz, in der sich die Matinee-Sequenz befindet, sorgt dafür dass der Spieler Schadenspunkte erhält, wenn er eine rotierende Figur berührt.

Matinee ist ein Editor für Animationen im Spiel. Gegenstände, wie Türen und Aufzüge, können beispielsweise über Keyframes bewegt werden. Matinee-Sequenzen sind immer ein Teil von Kismet-Sequenzen. Es gibt aber auch weitere Funktionen, wie die Veränderungen von Materialien.

Die Veränderungen finden auf verschiedenen Spuren statt. In einem Film ließe sich eine Spur pro Kamera verwenden. Auch der Curve Editor steht wieder zu Verfügung. Die Veränderung wird im Level Editor vorgenommen und als Keyrame festgehalten. Das Ergebnis ist sofort sichtbar.

Das zu animierende Objekt wird in Kismet als Objektvariable eingefügt und mit einer Matinee-Sequenz verbunden, welche auch vorher in Kismet angelegt wird.

# 6 Erweiterte Techniken

## 6.1 Natürliche Umgebungen zeichnen

Environments lassen sich im Unreal Editor wesentlich schneller erstellen als Architektur und Städte. Man kann sie mit verschiedenen Tools in 3D so erstellen, als ob man malen würde.

Das Environment wird in dieser Map für eine Hintergrundlandschaft und den chinesischen Garten verwendet. Die Anhänge drei und vier zeigen, um welche Landschaften es sich handelt.

Berge und Täler werden mit einer Height Map erzeugt. Das Environment ist zunächst nur eine Flache Ebene ohne Textur. Die Ebene wird auf die gewünschte Größe skaliert und dann so oft aufgeteilt, dass man die einzelnen Höhen- und Tiefenunterschiede einzeichnen kann. Das Environment wird dabei immer nur nach unten oder nach oben bewegt und nicht zur Seite.

Das Environment kann anschließend auf mehreren Ebenen texturiert und mit Static Meshes detailliert werden. Beides wird auch wieder gemalt. Dabei muss der Level-Designer auf die Anzahl der Polygone und die der Texturebenen achten. Ein Static Mesh, welches aus tausend Polygonen besteht, sollte nicht zu oft verteilt werden. Je dichter die Meshes aneinander liegen, desto stärker wird der Rechner später im Spiel beansprucht. Deshalb gibt es Objekte, die nur aus bemalten Flächen bestehen. Eine Reihe von Grashalmen kann beispielsweise auf eine Ebene gemalt und die freie Fläche als transparent maskiert werden. Diese Grashalme sind ein leistungsschonendes Mittel, um ein grünes Environment zu bewalden.

Das Environment wird mit Materials texturiert. Für jedes Material muss man eine neue Ebene anlegen. Das Material kann auf jedes Stück der Ebene gemalt werden und Verläufe zwischen verschiedenen Materialen sind möglich. Um die Grafikkarte nicht zu sehr zu belasten, sollten nicht zu viele verschiedene Ebenen verwendet werden.

Der Shaolin-Tempel soll auf einem kleinen Berg liegen. An den Rand des Levels grenzt auf einer Seite eine typische Berglandschaft Chinas. Die Berge sind größtenteils mit grünem Grass bewachsen, an manchen Stellen ist Stein zu sehen. Beides wird mit Materialebenen erzeugt, ohne Static Meshes. Die Hintergrundlandschaft soll kaum Rechenleistung in Anspruch nehmen. Um die Performance zu verbessern, wird nur der chinesische Garten besonders detailliert. Dort sind auch mehrere Static Meshes mit höheren Polygonzahlen zu sehen. Um sie speziell anzuordnen, werden sie aber nicht unbedingt gemalt, sondern eventuell wie andere Static Meshes normal platziert.

### 6.1.1 SpeedTrees

SpeedTrees sind ein besonderer Bestandteil von Environments. Sie können nur als Teil eines Environments gemalt werden und sind keine Static Meshes. Sie bewegen sich im imaginären Wind und verändern ihr Alter in Abhängigkeit der Größe, in der man sie malt. Es gibt verschiedene Baumarten zur Auswahl. Die einzelnen Texturen für Rinde und Blätter können unabhängig voneinander hinzugefügt werden.

### 6.1.2 Foliage Volumes

Eine weitere Möglichkeit ein Environment zu bewalden, sind Foliage Volumes. Ein Volumen wird mit BSP Geometrie erstellt und die Option für Volumen sind in der Werkzeugleiste des Level Editors zu finden. In den Einstellungen des Volumens lässt sich ein Static Mesh sowie dessen Größe und die Dichte für die Verteilung innerhalb des Volumens festlegen. Man muss keine neuen Deco Layers festlegen und kann gezielt Bereiche eines Environments schmücken.

## 6.2 Wasser

Das Level hat mehrere Stellen an denen Wasser vorkommt. Es ist nicht nur typisch für einen chinesischen Garten, es ist auch interessant zu untersuchen wie es umgesetzt wird, um es später mit der CryEngine2 zu vergleichen. Realistisches

Wasser, das aus Partikeln besteht, gibt es nicht in Unreal Tournament. Es würde zu viele Ressourcen verbrauchen. Wasser ist eine Mischung aus Materialien, Volumen und Partikeleffekten.

Die Oberfläche eines Flusses ist entweder eine Ebene aus BSP-Geometrie oder ein simples Static Mesh, in der Form eines Wasserfalls, mit einem animierten Material. Die Spritzer und die durch die Luft fliegenden Wassertropfen entstehen mit Hilfe von Partikeleffekten. Um dem Spieler das Gefühl von Wasser zu vermitteln, wenn er sich darin befindet, nimmt man „Volumes“. Das sogenannte WaterVolume hat einen Wiederstand, den der Spieler spürt und es wird ein Sound abgespielt, sobald er sich im Wasser bewegt. Ein PostProcessVolume sorgt mit Tiefenschärfe und Bewegungsunschärfe dafür, dass sich die Sicht unter Wasser verändert.

## 6.3 Nebel

Der Nebel sorgt für Atmosphäre und ist typisch für die chinesischen Landschaften. Der „Height Fog“ ist außerdem sehr gut dazu geeignet, Objekte auszublenden (Occluding). So werden sie nicht gerendert und Rechenleistung wird gespart. Zusätzlich kann man den Rand des Levels damit verdecken. Bei einem additiven Level entstehen zufällige Bilder im leeren Raum. Den Height Fog findet man im Actor Classes Browser und er hat einige Einstellungen für Dichte, Farbe usw.

## 6.4 Beleuchtung

Die Beleuchtung eines Levels ist eine der aufwendigsten Aufgaben im Level Editor. In dieser Map soll eine harmonische Sonnenuntergangsstimmung herrschen, wobei es eher auf eine romantische Stimmung als auf Realismus ankommt, um den Spieler an die religiöse Hintergrundstimmung des Levels zu erinnern.

Realistisches Licht in der Unreal Engine ist sehr schwer umzusetzen, weil es kein realistisches Licht in dieser Engine gibt. Es gibt nur wenige Lichttypen und keine

realistischen Vergleichswerte wie Lumen oder Candela. Stattdessen gibt es bei jedem Lichttyp zumindest folgende Einstellungen:

Color: Farbe

Brightness: Helligkeit

Radius: Radius der über die Reichweite des Leuchtmittels bestimmt.

FalloffExponent: Beschreibt wie stark der Helligkeitsunterschied zwischen Zentrum und Rand des Radius sein soll.

Außerdem gibt es kein echtes Raytracing-Verfahren oder etwas vergleichbares, denn das Licht prallt nicht von Oberflächen ab (Bounce Light), sodass eine einzelne Lichtquelle oft nicht reicht, um einen Raum realistisch zu beleuchten. Es gibt in der Version von Unreal Tournament 3 auch keine Global Illumination oder Radiosity Features. Stattdessen müssen diverse einzelne Lichter im Level verteilt werden, mit denen das apprallende Licht simuliert wird.

### 6.4.1 Lichttypen:

Point Light: Vergleichbar mit einer Glühbirne, die gleichmäßig in alle Richtungen strahlt.

Sky Light: Beleuchtet die gesamte Szene gleichmäßig. Es gibt keine Schatten. Es gibt aber die neuen Einstellungen LowerBrightness und LowerColor, die Licht von unten suggeriert. Das Licht von oben vermischt sich dann mit dem Licht von unten in der Mitte eines Raumes. Die Mitte ist abhängig von der Platzierung der Quelle im Raum.

Directional Light: Vergleichbar mit der Sonne. Die komplette Szene wird aus einer Richtung beleuchtet und es entstehen entsprechende Schatten. Für ein Außenareal muss die Cast Shadows-Option in den Einstellungen des Skydomes deaktiviert werden, sonst hüllt der Skydome das gesamte Level in Dunkelheit.

Spot Light: Leuchtet kegelförmig wie ein Autoscheinwerfer in eine Richung. Für dieses Licht gibt es die zusätzlichen Einstellungen InnerConeAngle und

OuterConeAgel, mit denen man innerhalb des Kegels einen Verlauf erzeugen kann. Es wird für Laternen innerhalb des Tempels mehrfach verwendet.

Directional und Spot Light sind dynamische Lichtarten, die entweder beweglich oder an- und ausschaltbar sind. Leider darf man nicht allzu viele dynamische Lichtquellen in einem Level benutzen, damit die Performance auch auf schwächeren Rechnern akzeptabel bleibt.

Die Grundstimmung wird mit einem Sky Light erzeugt. So wird alles gleichmäßig etwas erhellt und es gibt keine komplett schwarzen Stellen im Level. Ein paar Directional Lights erzeugen das Licht vom Himmel und besonders viele Point Lights sorgen für den Rest.

Um nur bestimmte Static Meshes oder Bereiche des Levels zu beleuchten, werden Light Channels und Light Volumes verwendet. Jedes Objekt hat außerdem Optionen für dynamische und statische Schatten, sowie für dynamische und statische Lichter. Es können immer beide Optionen getrennt voneinander gewählt werden.

### 6.4.2 Weitere Beleuchtungsmöglichkeiten

Post Process Volumes können dazu verwendet werden, das gesamte Level in verschiedene Farben zu tauchen. Sonnenstrahlen (God Rays) sind eigentlich Static Meshes, die ein transparentes Material bekommen. Lens Flares sind Ebenen, die ein Lens Flare-Material bekommen und so gerendert werden, dass sie vor allem erscheinen, sobald der Ursprungsort, also das Zentrum, der Lens Flares gesehen wird. Objekte sehen durch den Emissionskanal so aus, als würden sie Leuchten, aber sie strahlen kein Licht aus. In Wirklichkeit muss vor jedem Static Mesh, das aussieht wie eine Lampe, mindestens ein Point Light platziert werden. Meistens werden aber mehrere Point Lights um das Static Mesh herum platziert.

### 6.4.3 Light Maps

Nachdem alle Lichtquellen in der Map platziert sind, werden über den Build Lighting-Button mehrere Light Maps für das Level generiert. Das Licht und die Schatten aller statischen Quellen werden in Texturen gespeichert. Je nachdem, wie

hoch die Auflösung pro Oberfläche und Static Mesh eingestellt ist, variieren die Details für Licht und Schatten. Diese Details können optisch einen realistischen Eindruck machen, aber je höher die Details sind, desto mehr Speicher wird für die Texturen verbraucht. Dadurch, dass alle dynamischen Schatten in Texturen gespeichert wurden, sind sie unbeweglich.

Für BSP-Geometrie sind die Einstellungen der Light Maps, pro Oberfläche auf der f5-Taste zu finden. Je höher die Zahl, desto geringer ist die Auflösung. Eine Auflösung von 8 bedeutet, es gibt ein LightMap Pixel für alle 8 Unreal Units. Für Static Meshes sind die Einstellungen anders. Normalerweise nutzen Static Meshes Vertex Lighting. Die Lichtwerte werden über die Vertices der Geometrie gespeichert und über die Flächen ineinander geblendet. Wenn in den Einstellungen im Static Mesh Editor für das jeweilige Static Mesh der Wert für LightMapCoordinateIndex 1 ist, dann ist ein UV-Set für Light Maps vorhanden. In den Static Mesh-Einstellungen (f4-Taste), lässt sich die Light Map unter bOverrideLightMapResolution aktivieren und unter OverriddenLightMapResolution wird die Auflösung eingestellt. Hier enstpricht sie der eigentlichen Angabe in Pixeln. Also ergibt eine Eingabe des Wertes 4 eine Textur der Größe 4*4 Pixel. Die UV Koordinaten für die Light Map müssen in einem extra UV-Kanal liegen, normalerweise mit dem Index 1, da das Array für die Koordinaten mit 0 beginnt.

Die Koordinaten können entweder im Unreal Editor automatisch generiert werden, oder man erstellt sie in einer 3D-Applikation. Die einzigen Voraussetzungen für eine funktionierende Light Map sind: Die UV-Koordinaten müssen im Wertebereich zwischen 0 und 1 liegen und sie dürfen sich nicht überlappen.

Je größer die Light Maps sind, desto länger dauert es sie zu generieren. Dieser Vorgang kann, abhängig von der Anzahl der Lichter, mehrere Minuten dauern. Der Speicherplatzbedarf der Map wird durch Light Maps größer, das heißt, es dauert länger sie herunterzuladen. Außerdem wird mehr Arbeitsspeicher verbraucht, was zur Folge haben kann, dass die Map auf älteren PC's oder der PlayStation 3 nicht funktioniert. Es ist deshalb empfehlenswert die Einstellungen für Light Maps bei BSP und Static Mesh auf Standard zu lassen, wenn die Optik des Levels nicht darunter leidet.

Leider gibt es Probleme beim Generieren der Light Maps. Es kommt des Öfteren vor, dass eine Map während des Generierens des statischen Lichts abstürzt. Es gibt dabei keine erkennbare Fehlermeldung und Forenthreads zu dem Thema führen ins Leere. Ein Directional Light in der aktuellen Map funktioniert nur, solange man das Licht nicht löscht, bewegt, oder Einstellungen verändert. Es ist eventuell möglich das Problem zu umgehen, indem man alles außer dem fehlerhaften Licht exportiert.

### 6.4.4 Bewegliche Lichter

Bewegliche Lichter die Schatten werfen, waren nur undeutlich in der Tech Demo zu sehen. Mich hat es deswegen interessiert herauszufinden, ob es in Unreal Tournament wirklich möglich ist, diese Lichter herzustellen.

Um ein Static Mesh beweglich zu machen, wird es als Rigid Body hinzugefügt. Nun reagiert es auf Kollisionen mit Projektilen, Explosionen und Fahrzeugen. Damit die Lampe bei Berührung pendelt, wird über den Actor Classes Browser ein Joint Constraint hinzugefügt. Constraint und Static Mesh verbindet man über die Actor Properties (f4).

Solange der Constraint aktiv ist, erscheinen die Eigenschaften unter „RB_ConstraintActor“. Sobald diese angezeigt werden, muss man auf das Schloss oben links klicken, um den Constraint zu verriegeln. Jetzt kann das Static Mesh angewählt werden, ohne dass sich das Actor Properties-Fenster ändert. Wenn man bei „ConstraintActor1“ auf den grünen Pfeil klickt, wird das Static Mesh hinzugefügt und beide sind verbunden.

Ein bewegliches Licht wird aus dem Actor Classes Browser hinzugefügt. In dessen Actor Properties liegt die Option „Attachement“ → „Base“, die es ermöglicht, das Static Mesh als Basis mit der Lichtquelle zu verbinden.

Rigid Bodys erzeugen automatisch dynamische Schatten. Diese Schatten sind aber sehr rechenintensiv und außerdem nur sichtbar, wenn der Spieler sehr nah rangeht. Sie werden im Spiel normalerweise nicht verwendet, außer bei ganz besonderen Situationen und in ganz kleinen Räumen, in denen das Problem mit der Distanz nicht auffällt. Diese dynamischen Lichter kann man nicht häufig einsetzen, da sie auf PS3,

Xbox 360 und älteren Computern den Spielfluss zu stark verlangsamen. Darum werden sie meistens deaktiviert.

Um die Schatten einer Laterne zumindest nachzuahmen, lassen sich sogenannte Light Functions in der Unreal Engine verwenden. Dabei wird über ein Spotlight ein Graustufenbild gelegt und eine Silhouette erzeugt. Je nach Helligkeit entsteht ein Schatten. Mit dieser Methode kann man Dinge wie Schatten von Feuer, Wolken oder Lichtkegel für Taschenlampen und Autoscheinwerfer erzeugen.

## 6.5 Botpfade, PickUps und PlayerStartUps

Bots, PickUps und Waffen, sind ein notwendiger Bestandteil jeder Map in Unreal Tournament. Ohne Waffen gibt es keinen Wettkampf und ohne PickUps keine taktischen Vor- und Nachteile. Ohne Startpunkte können die Spieler nicht wiederbelebt werden. Falls der Spieler im Einzelspieler-Modus spielt oder auf dem Server zu wenig Spieler sind, werden die echten Spieler durch Bots ersetzt. Solange nicht mindestens 16 Startpunkte verteilt sind, wird der Level Designer auch vom UnrealEd darauf hingewiesen. Die Einstellungen dafür kann der Host selber vornehmen. Bots sind Gegner, die vom Computer gesteuert werden. Damit die Bots funktionieren, müssen Laufpfade für sie angelegt werden. Dadurch können sie sich im Level bewegen.

Jedes Level braucht PlayerStartUp-Punkte. Das sind Stellen an denen der Spieler im Level startet, nachdem er gestorben ist. Diese Punkte werden möglichst an etwas verdeckten Stellen platziert, damit ein Spieler nicht gleich wieder abgeschossen wird, in der Sekunde nachdem er wiederbelebt wurde. Außerdem kann man die Blickrichtung des Spielers beim „Spawning“ bestimmen.

Die Waffen und zugehörige Munition sind ebenfalls wichtig für die Navigation der Bots. Das gleiche gilt für Rüstung, Lebensenergie und PowerUps. Alles ist im Actor Classes-Browser unter dem Punkt NavigationPoints zu finden. JumpPads und Teleporter sind auch Punkte zur Navigation, aber gesondert im Browser aufgeführt.

Im Build-Menü werden die Pfade schließlich generiert. Punkte, die sich nicht verbinden lassen, weil Objekte im Weg sind, werden über PathNodes verbunden, damit die Bots alle Gegenstände finden können.

Die Bots laufen nicht gerade an den Pfaden entlang, sondern folgen nur der Richtung in die diese Pfade führen. Je nach Schwierigkeitsgrad ist das Verhalten unterschiedlich. Durch das Laufen und Springen können die Bots an Gegenständen, wie SpeedTrees, im Level hängen bleiben, oder in einen Abgrund fallen. BotBlockingVolumes halten sie davon ab. Weitere Actor Classes, wie Verteidigungspunkte, lassen sich ebenfalls platzieren, damit eine wichtige Stelle im Level beschützt werden kann.

### 6. 5. 1 Portale und JumpPads

Portale und JumpPads sind besondere Navigationspunkte. Dadurch kann man andere Stellen im Level erreichen, die sonst für den Spieler unerreichbar oder auf Umwegen erreichbar sind. Das Level für diese Arbeit hat sechs Portale. Vier davon dienen der Beschleunigung des Gameplays. Zwei von ihnen befinden sich am Anfang und am Ende der Map, sodass ein Spieler nicht einmal den ganzen Weg von vorn bis nach hinten laufen muss, sondern durch das Portal gehen kann um einen Gegenstand zu erreichen oder einen Gegner zu finden. Die Spieler können sich schneller finden und es kommt schneller zum Kampf. Zwei Portale sind als Aufgang zum ersten Stock geplant. Statt eines Fahrstuhls oder einer Treppe ist der Aufgang zuerst nur über das Portal geplant gewesen. Der Anreiz zum Betreten des ersten Stockwerks war der Raketenwerfer, der die stärkste Feuerkraft im Level hat. Das Problem, dass dadurch entstand, war, dass ein Spieler oben beim Portal mit dem Raketenwerfer warten konnte, sobald er ihn hatte, um dann die nächsten Spieler, die durch das Portal kommen, in einen Hinterhalt zu locken.

Damit die Spieler auch von anderen Seiten Zugang zum ersten Stock haben, wurden zwei JumpPads außerhalb des Gebäudes platziert. Die Spieler werden nach oben, in Richtung eines angegebenen PathNodes, katapultiert. Sie können jedoch auch die Richtung mit den Bewegungstasten beeinflussen. So wird das oberste Stockwerk zu

einer schnell erreichbaren und hart umkämpften Zone, was den Beta-Testern sehr gut gefallen hat.

JumpPads und Portale sind im ActorClasses-Browser zu finden. In den Einstellungen müssen die Ziele angegeben werden. Für das Level Design war es notwendig einige der Portale zu verändern, sodass sie weniger Performance beanspruchen. Normale Portale zeigen nämlich Vorschaubilder von dem was sich auf der anderen Seite befindet. Da diese Vorschau jeweils mit 15 fps gerendert wird, wird die Grafikkarte zusätzlich zum normalen Spielverlauf beansprucht. Um die Performance zu optimieren, wurde bei den meisten Portalen ein Material gewählt, welches nur einen Screenshot vom Zielort anzeigt. Die Portale innerhalb des Gebäudes haben weiterhin eine Echtzeitvorschau. Da innerhalb des kleinen Raumes weniger Polygone dargestellt werden und dort auch keine dynamischen Lichter in unmittelbarer Nähe sind, ist das Gameplay nicht gefährdet.

# 7 Erkenntnisse und Schlussfolgerungen aus dem Beta Test

Der erste Betatest ergab bereits ein positives Feedback. Kleine Schwachstellen konnten sofort ausfindig gemacht werden. Hier soll ein Überblick über die Entwicklungen und Schlussfolgerungen nach dem ersten Test gegeben werden.

Der Spielspaß bei solchen Spielen wird schnell gemindert, wenn der Spieler an Gegenständen wie Büschen und Stufen hängen bleibt. Bei einem schnellen Ego-Shooter wie UT3 ist ein Spieler immer nur am Laufen, Springen, Ausweichen und Schießen. Sobald er hängen bleibt, kann das seinen Tot bedeuten. Laufen ist deshalb der Grundzustand eines Spielers. Es sei denn, er hat sich irgendwo verschanzt. Dadurch entsteht ein Flow. Sobald der Lauf durch ein Hindernis unterbrochen wird und sich dieses Ereignis auch noch wiederholt, empfindet der Spieler es als sehr störend.

Spawn-Punkte für einen Spieler müssen auch immer dort platziert werden, wo der Spieler etwas geschützt ist, damit er nicht gleich wieder stirbt und diese Punkte müssen auch so platziert werden, dass der Spieler gleich weiter laufen kann. Selbst wenn der Spieler es nicht soll, empfinden unerfahrene Spieler sowas als schlechtes Level Design. Das Spiel wird oft sehr schnell, und Neulinge haben noch nicht so gute Reflexe. Außerdem kennen sie das Level nicht, was die rechtzeitige Reaktion zusätzlich erschwert.

Die selbstgestalteten Bäume und Glühwürmchen, sowie der feuerspeiende Drache wurden sehr positiv bewertet. Die beweglichen Lampen, die im Tempel an der Decke hängen, sind hingegen kaum bis gar nicht aufgefallen! Sie wurden als normal hingenommen und keinem ist aufgefallen, dass die Schatten nicht echt sind. Während sie in einem externen Modder-Forum als „freaking awesome“ bezeichnet wurden, denkt der Spieler sie zählen zu den Mindestanforderungen eines Levels. Da sie kaum auffallen und gleichzeitig viele Ressourcen verbrauchen, sind sie vermutlich auch kaum im eigentlichen Spiel zu sehen.

Während einige Bereiche des Levels besonders gut ausgeschmückt wurden, gab es noch einen Bereich der etwas zu leer war. In der Realität hätte der leere Platz vor dem Tempel einen Sinn ergeben, weil die Mönche dort Übungen absolvieren, aber

aus der Perspektive eines Spielers wirkt es leer und langweilig. Außerdem wurden fast alle Static Meshes und Materials, die im Editor enthalten sind, bereits verwendet. Es musste etwas Neues her, das zeitlich machbar war, zu einem Shaolin Tempel passt, möglichst gut aussieht und vielleicht sogar den Spieler mit einbezieht.

Bereits am selben Tag gab es neue Ideen und alte wurden wieder hervorgeholt. Aus vielen Martial Arts-Filmen kennt man die verschiedenen Übungen und Lehren, die ein Meister seinen Schülern beibringt. Nicht nur Balanceübungen oder die sich bewegenden Holzfiguren aus einem alten Film mit Jackie Chan wurden in Betracht gezogen. Ebenso kam auch die Idee eines Shaolin-Mönchs auf, der mindestens ein Mal pro Runde auftaucht und gegen die Spieler kämpft; der zusätzliche Entwicklungsaufwand, den die Gestaltung eines Charakters erfordern würde, überschreitet jedoch den Zeitrahmen, sodass diese Idee in den Hintergrund gestellt wurde. Es kamen aber genug Ideen zusammen für die Vervollständigung des Levels.

# 8 Game Art

## 8.1 Erstellung eines Modells

Ein grobes Modell wird zuerst in Maya 8.5 oder 3Ds Max 2008 modelliert. Danach wird es in Unreal getestet.

Dabei wird auf fehlerhafte Polygone geschaut, ob das Modell wie gewünscht aussieht und die Kollision zufriedenstellend funktioniert.

Im Garten des Tempels gibt es zwei kleine Brücken. Sie wurden darauf getestet, ob man herüberlaufen kann, ohne dass man dabei behindert wird. Die Bäume mussten mehrfach importiert werden, weil sie optisch nicht in die Umgebung passten - die Äste waren zu lang und der Stamm zu kurz. Auch wenn sie realistisch wirkten, passten sie jedoch nicht in den Garten. Es stellte sich dabei auch heraus, dass Polygone, die zu nah aneinander liegen, in der Unreal Engine transparent sind. Um in ZBrush harte Kanten zu erzwingen, müssen mehrere Edge Loops sehr nah aneinander gelegt werden. Deshalb muss ein Objekt oft hin und her exportiert werden, damit Fehler so früh wie möglich entdeckt werden.

Zwischen 3ds Max, Maya und ZBrush wird das obj.-Format verwendet. Es wird von allen Programmen unterstützt. Für die Unreal Engine wird in Maya der Collada Exporter installiert. Der Collada Exporter nutzt das dae.-Format, welches frei im Internet zur Verfügung steht.

Eine weitere Art der Modellierung ist das "ZSphere Modelling" in ZBrush. Es eignet sich besonders gut zur Erstellung von organischen Modellen. Ein Baum bekommt dadurch schneller seine Grundform als durch gängige Methoden. Er muss trotzdem in Maya exportiert werden, damit die UV-Koordinaten angefertigt und Optimierungen vorgenommen werden können.

Nachdem das Grundgerüst fertig ist, können die UV-Koordinaten angelegt werden. Dadurch wird festgelegt, wie genau die Texturen auf das Modell zu legen sind. Die

Ränder oder "Seams" sind das größte Problem und müssen möglichst unauffällig platziert werden, weil sie ansonsten später sichtbar sein können.

Die Koordinaten lassen sich in 3ds Max, Maya und ZBrush auch automatisch anlegen, aber die präziseste Möglichkeit, bei komplexen Modellen, ist das manuelle Anlegen der Koordinaten mit dem RoadKill-Plugin, welches frei im Internet zur Verfügung steht.

Nachdem die Koordinaten angelegt wurden, wird das Objekt in ZBrush detailliert. Brushes werden in Photoshop angefertigt und in ZBrush als Alphas importiert. Alphas sind Graustufen-Bilder, welche in ZBrush als Höhen und Tiefen interpretiert werden.

In ZBrush wird ein hochauflösendes Mesh erstellt und detailliert. Diesen Prozess nennt man Sculpting. Nach dem Sculpting wird das Mesh via Polypainting koloriert. Die Objekte müssen in die Umgebung passen, sowie auch zur Architektur der schon bestehenden Bauelemente. Darum werden Farben von den Texturen der ursprünglichen Assets zum Kolorieren der neuen Assets verwendet, indem man Screenshots vom Level macht und die Farben in Photoshop analysiert, um dann mit ihnen in ZBrush zu malen. Die Farben sehen in ZBrush eventuell dunkler aus als in Photoshop, weshalb die Farben nachträglich wieder korrigiert werden müssen. Es kommt dabei auf den Workflow an.

Beim Sculpting werden mehrere Subdivision Levels zum Mesh hinzugefügt, sodass es mehrere Millionen Polygone hat. Durch die Bearbeitung des Meshes werden auch alle anderen Levels verändert. Die UV-Koordinaten bleiben jedoch unverändert, weshalb die Textur verwaschen wirken kann, wenn sie auf das untere Level, das ursprüngliche Mesh, gelegt wird. Darum muss das unterste Level, bevor die Texturen angefertigt werden, noch einmal exportiert werden und mit der Unfold-Option im UV-Editor die UV-Koordinaten neu ausgerichtet werden. Das Mesh wird dann wieder in ZBrush reimportiert. Die Geometrie darf dabei nicht verändert werden. ZBrush akzeptiert keine neuen Polygone.

Es hat sich herausgestellt, dass die Export-Option „Mrg.“ in ZBrush aktiviert werden muss, damit die Koordinaten nicht pro Polygon in einzelne Stücke zerlegt werden. In Maya reicht es nicht, die obj.-Datei einfach zu importieren. Vorher wird die Import-

Option geöffnet und Create Multiple Objects auf "False" gestellt. Die UV-Koordinaten können in ZBrush immer mit der Funktion UV Check auf Richtigkeit überprüft werden.

Ein Objekt in der Game Engine genau so aussehen zu lassen wie in ZBrush, ist ein sehr komplizierter und langwieriger Workflow.

Die Farbtextur kann auf verschiedene Arten erstellt werden. Die reinen Farbinformationen lassen sich leicht als Textur im psd.-Format exportieren. Diese Textur hat aber keine Schatten und Materialinformationen, auch nicht die eigentliche Materialfarbe.

Um die Materialfarbe und die Schattierung in die Textur einzubinden, kann das ZApplink-Plugin, welches frei im Internet zur Verfügung steht, verwendet werden. Damit werden Screenshots vom Viewport in ZBRush erstellt und dann in Verbindung mit Photoshop automatisch auf das Objekt gelegt.

Das Problem ist, dass dabei Ränder und Schatten entstehen, die nachträglich entfernt werden müssen. Manche Objekte müssen auch zerlegt werden und der Ablauf für jedes Teil einzeln durchgeführt werden.

Eine weitere Möglichkeit ist die Erstellung einer Displacement Map, welche dann in ZBrush mit einem Material belegt wird, sodass die Schatten eingebettet werden. Dieses Bild wird exportiert und dann in Photoshop über das Farbbild gelegt. Diese Methode liefert sehr schöne Ergebnisse, aber sie sind nicht immer präzise genug.

Die dritte Möglichkeit für dieses Projekt besteht darin, eine Cavity Map in ZMapper zu erstellen und sie als Ambient Occlusion Map zu benutzen. Mit ZBrush kann man keine Ambient Occlusion Maps erstellen, Cavity Maps sind diesen jedoch sehr ähnlich. Sie können entweder als Alpha Channel Bestandteil einer Normal Map sein oder sie werden in Photoshop mit der Color Map zusammengelegt.

Die letzte Möglichkeit ist ähnlich der zweiten, liefert aber bessere Ergebnisse. Hierbei wird wieder eine Displacement Map generiert. Mit dem Image Plane-Plugin, welches für ZBrush kostenlos zum Download bereitsteht, wird die Textur als Dokument geöffnet und mit dem gewünschten Material versehen. Die Intensität der

Displacement Map wird angepasst, sodass die Textur aussieht wie im Pixol-Modus. Anschließend kann das Dokument als psd.-Datei gespeichert werden. Auch hier kann es vorkommen, dass die Displacement Map für Schatten sorgt, die nachträglich in Photoshop geändert werden müssen.

Die verschiedenen Texturen werden in Photoshop mit Ebenenmodi zusammengelegt. Hier können zusätzlich weitere Anpassungen der Farben vorgenommen, sowie weitere Details eingefügt werden. Die Textur kann außerdem geschärft werden, um den Detailgrad zu erhöhen. Die Textur wird im tga.-Format in den Unreal Editor importiert und kann dort zusätzlich im Material Editor angepasst werden. Ein wichtiger Teil der Optimierung ist das Reparieren der Ränder der Texturen an den UV-Koordinaten. Die sichtbaren Ränder bezeichnet man als Seams. Sie werden hauptsächlich mit dem Clone Stamp-Tool in Photoshop entfernt. Ein ähnliches Werkzeug gibt es auch im Projection Master von ZBrush.

Die Normal Map soll die hochauflösenden Details suggerieren, die in der Game Engine nicht als Mesh dargestellt werden können. Um eine möglichst gute Normal Map zu erhalten, wird eine 16-Bit Displacement Map in ZBrush erstellt und diese als Alpha behalten, während in ZMapper eine Tangent Space Normal Map kreiert wird. Normal und Displacement Map können unter Umständen nicht auf der untersten SubDivision erstellt werden, weil sie zu viele Details darstellen müssen, was nicht gut aussieht. Eckige Kannten, die leicht abgerundet sind, werden durch eine Normal Map schattiert dargestellt, weil die Normal Map durch die Schatten Tiefe suggerieren will. Es ist besser von einer höheren SubDivision die Maps zu generieren und auch diese SubDivision zu exportieren. Die Normal Map wird dann in Photoshop optimiert und in den Editor importiert.

Das Mesh wird im dae.-Format exportiert und im Static Mesh Editor vom Unreal Editor geöffnet. Hier wird dem Mesh ein Kollisionsmodell, ein Material und ein "Physical Material" hinzugefügt, damit der Spieler damit interagieren kann. Ein Kollisionsmodell kann entweder außerhalb des Editors angefertigt oder im Unreal Editor auf Wunsch automatisch erzeugt werden.

Die Texturen werden in einem neuen Material zusammengelegt. Hier lassen sich weitere Anpassungen für Helligkeit und Kontrast vornehmen. Außerdem werden

Detail-Texturen für den Specular und den Normal-Kanal hinzugefügt. Für den Specular-Kanal wäre eine richtige Textur besser, aber um weniger Speicher für Texturen zu verbrauchen, wird die Farbtextur im Material Editor über Material Expressions verändert und verwendet. Da man die Texturen möglichst klein halten sollte, um wenig Speicher zu verbrauchen, kann man sie auch nachträglich im Editor verkleinern und gleichzeitig die Ergebnisse sehen.

Alle selbsterstellten Objekte werden in einem Package gespeichert. Damit die Objekte auch bei anderen Spielern nach dem Export der Map sichtbar sind, wird das Package genau wie die Map benannt, damit alle enthaltenen Objekte auch in der Map gespeichert werden. Leider reicht es nicht, dass alle Static Meshes im Static Mesh Editor schon ihr Material zugewiesen bekommen. Es ist auch notwendig, ihnen das Material später im Level noch einmal zuzuweisen. Alle nicht im Level gespeicherten Assets werden beim Export aus dem Map Package gelöscht. Es hat sich außerdem herausgestellt, dass Fehler beim Import in den Unreal Editor auftauchen können und der Editor beim Export ohne Fehlermeldung abstürzt. Es ist also notwendig die Map regelmäßig zu exportieren, um zu prüfen, ob alles fehlerfrei ist.

## 8.2 Texturen und Materialien

Außer Texturen für Static Meshes gibt es noch Texturen für die Oberflächen der BSP-Geometrie. Sie werden entweder in Photoshop generiert oder basieren auf einem Photo, welches dann in Photoshop bearbeitet wird.

Um ein Foto für eine Textur zu schießen, empfiehlt sich ein bewölkter Nachmittagshimmel, sodass möglichst diffuses und neutrales Licht auf die Textur fällt.

Das Foto wird anschließend in Photoshop zurecht geschnitten und danach mit dem Offset-Filter sowie dem Kopierstempel bearbeitet. Die Textur wird später im Spiel, je nach Größe der Oberfläche, mehrfach aneinander gereiht. Durch die Bearbeitung soll die Wiederholung unauffällig und die Sichtbarkeit der Übergänge auf ein Minimum reduziert werden.

Nachdem das Bild fertig ist, wird mit Crazy Bump eine Normal Map generiert. Die Demo-Version ist kostenlos im Internet verfügbar. Auch Displacement und Specular Maps lassen sich generieren. Nachdem die Normal Map fertig ist, wird der Grün-Kanal in Photoshop umgekehrt, um die richtigen Ergebnisse in der Unreal Engine zu erzielen.

Die einzelnen Texturen werden im .tga-Format exportiert und danach ein Material im UnrealEd erstellt. Damit sich die Texturen nicht so oft wiederholen, kann man im Materialeditor noch andere Bilder über die Color Map legen. Dasselbe gilt für Normal und Specular Maps.

# 9 Vergleich der Unreal Engine 3 mit der CryEngine2

## 9.1 Entwicklungsgeschichte

Unreal Tournament 3 und Crysis erschienen beide etwa gleichzeitig und wurden zu ihrer Zeit als die beiden grafisch anspruchsvollsten Titel ihrer Zeit angesehen. Während das amerikanische Unternehmen Epic Games schon seit Langem erfolgreich mit der Vermarktung der Unreal Engine ist, musste sich die deutsche Firma Crytek mit der CryEngine2[27] erst profilieren.

Zu einer Game Engine gehört auch ein entsprechender Editor[28], damit die Designer ihre Vision in die Tat umsetzen können. In diesem Kapitel sollen die technischen Möglichkeiten zur Umsetzung von Spielen im Editor und die optischen Ergebnisse innerhalb der Game Engines miteinander verglichen werden. Anschließend sind die Vor- und Nachteile beider Engines und ihrer Editoren zu vergleichen. Die Vergleiche der einzelnen Features basieren hierbei lediglich auf Erfahrungen, die in Einzelarbeit und in Teamarbeit, mit beiden Engines, gemacht wurden.

Beide Engines wurden in den letzten Jahren weiterentwickelt, wobei viele neue Features jedoch nur in Tech Demos und Videos zu sehen sind, aber nicht in fertigen Spielen.

[27] Einen groben Überblick über die CryEngine2 bietet, BÖGE, Torben; MEYER, Stefan; TSANG, Chor-Hung, CryEngine2 2008

[28] Einen groben Überblick zum Level Editor von Crysis bietet AROUS, Ibrahim; THIELE, Daniel; TRENNT, Hanno, Projekt B, 2008

## 9.2 Licht und Rendering

Crysis hat die realistischere Lichtberechnung. Es ist sogar möglich, die Software zu Visualisierungszwecken mit echtem Raytracing zu verwenden. Crysis unterstützt keine Global Illumination Features, aber Ambient Occlusion, dynamische, weiche Schatten und echte Sonnenstrahlen. Diese Features hatte die Unreal Engine 3 damals nicht. Da das dynamische Licht bei UT3 in Texturen gespeichert wird, gibt es keine dynamischen Schatten in diesem Umfang. Es gibt dynamische Lichter, die Schatten werfen können, aber keine Soft Shadows über eine weite Distanz. Die Schatten sind hart und sie verschwinden ab einer deutlich sichtbaren Entfernung. Die UE3 ist für Konsolen optimiert. Mit dynamischen Lichtquellen wird deshalb sparsam umgegangen.

## 9.3 Performance

Die Performance von Crysis ist befriedigend. Es kommt stark auf die Einstellungen und die Auflösung an. Auf einem Rechner mit Q6700, 2*8800 GTX SLI, 4GB Ram und einer Auflösung von 1920*1200 Pixeln läuft Crysis nicht schnell genug für einen Ego-Shooter. Sogar mit deaktiviertem Anti-Aliasing waren die „Very High“-Einstellungen auf Vista (x64) nicht optimal spielbar. In Kampfsituationen brach die Performance unter die 30 fps-Marke ein. Mit der Zeit kamen neue Patches und Treiberupdates, sodass sich die Performance etwas verbesserte, aber nie ein Optimum erreichte.

Bei Unreal Tournament ist die Performance deutlich besser. Das Spiel ist für Xbox 360 und PlayStation 3 optimiert. Es läuft auf dem gleichen Rechner im Durchschnitt mit ca. 50 fps und mehr. Die Levels sind so gebaut, dass die Polygonzahl immer auch auf älteren Rechnern darstellbar ist, aber das statische Licht in der Unreal Engine zahlt sich hierbei aus. Die CryEngine2 war exklusiv für den PC und dort haben sich viele Spieler über die Performance aufgeregt.

## 9.4 Environments

In Crysis sehen Pflanzen sehr plastisch und natürlich aus. Sie reagieren teilweise auf Berührung und lassen sich auch teilweise zerstören. Das Wasser sieht realistisch aus und verändert sich bei Berührung. Der Tag-/Nachtrhythmus ist dynamisch. Es gibt aber sehr wenig Abwechslung im eigentlichen Spiel. Die CryEngine2 ist für ein Spielszenario im Dschungel optimiert. Die Blechhütten im Spiel sind teilweise zerstörbar und das Wetter ist dynamisch. In Crysis wirken die Umgebungen insgesamt realistischer.

In Unreal Tournament sehen Pflanzen nicht so realistisch aus, wie in Crysis. Sie sind teilweise sehr einfach gehalten, um die Performance zu verbessern und sie sind unzerstörbar. Zerstörbare Static Meshes sind jedoch in der Unreal Engine möglich. In Crysis sind zerstörbare Objekte, wie Blechhütten, aus Modulen zusammengebaut. Im UnrealEd werden Bruchstücke vorher modelliert oder vorberechnet. Die Speedtrees in Unreal sehen zwar nicht so realistisch aus wie die Palmen in Crysis, aber sie können dynamisch und entsprechend ihrem Alter angepasst werden. Dadurch sehen kleine Bäume aus wie Setzlinge. In Crysis werden nur Textur und Größe geändert. Es gibt besonders viel Architektur im UnrealEd und Materialien lassen sich besonders abwechslungsreich gestalten. Die beiden Spiele sind also für verschiedene Situationen optimiert.

## 9.5 Editoren

Sowohl Sandbox 2 (Crysis) als auch UnrealEd (Unreal Tournament) sind sehr fortschrittliche Leveleditoren und weisen diverse Features auf. Deshalb sei hier nur kurz auf die größten Unterschiede und Gemeinsamkeiten hingewiesen.

In beiden Entwicklungsumgebungen gibt es die Möglichkeit Environments, Materialien, Partikeleffekte, Skriptsequenzen und Animationen zu erstellen. Die Vorgehensweisen sind im Einzelnen sehr unterschiedlich, aber die Idee ist bei beiden Editoren dieselbe.

### 9.5.1 Environments

Das Malen von Environments ist in beiden Editoren sehr ähnlich, aber während das Platzieren von Straßen und Flüssen in Crysis wesentlich einfacher ist, treten Bugs bei den Materialebenen auf. Wie im UnrealEd gibt es pro Material eine Ebene und im Sandbox-Editor müssen die Texturen der Oberflächen generiert werden. Das Wetter ist in Crysis vollkommen dynamisch veränderbar. Das Wasser lässt sich mit der Windrichtung anpassen. In UT3 ist das Wasser meist eine Textur auf einer BSP-Oberfläche oder ein Static Mesh mit animiertem Material. Der Himmel ist ebenfalls ein animiertes Material. Volumetrische Wolken gibt es nur in der CryEngine2.

### 9.5.2 Materialien

Der Material Editor im UnrealEd ist mächtiger als sein Gegenstück im SandBox2-Editor. Im UnrealEd besteht die Möglichkeit die Texturen zu bewegen und zu animieren. Die einzelnen Texturen können mehrfach übereinander gelegt und wie in Photoshop bearbeitet werden, wodurch sehr komplexe Netzwerke erstellt und Materialien sehr vielseitig und detailliert eingesetzt werden können.

Beide Engines haben die Möglichkeit Sub Surface Scattering zu simulieren. Der Einsatz von Normal Maps und Displacement Maps ist ebenfalls bei beiden möglich.

### 9.5.3 Partikeleffekte

Beide Editoren haben Partikeleffekte und die Engines scheinen sie sehr realistisch darzustellen, ohne viel Performance zu verbrauchen. Die Partikeleffekte im Editor vom UnrealEd sind skalierbarer und flexibler als die des SandBox2-Editors. Der modulare Aufbau ermöglicht ein Feuer, mit Flammen, Funken und Rauch, als ein Partikelsystem. Im SandBox2-Editor wären es 3 einzelne Emitter.

### 9.5.4 Skriptereignisse und In Game-Animationen

Visual Scripting in Kismet und Flow Graphs in Crysis sind von der Grundidee identisch(Siehe Kapitel 5.1). Im SandBox2-Editor gibt es mehr Funktionen als im

Editor für Unreal Tournament 3. Dies liegt an den einzelnen Einzelspieler-Modi der beiden Spiele. Beim Editor für Gears of War gibt es andere Funktionen, als für Unreal Tournament. Der eigentliche Vorteil im SandBox2-Editor ist der logische Aufbau und die Optionen der einzelnen Funktionen. Eine Skriptsequenz sieht sehr ordentlich aus und ist wie ein Flussdiagram, dass von links nach rechts geht, aufgebaut. Im Unreal Editor ist eine Sequenz mit einer Mind Map zu vergleichen.

Fast alle Funktionen sind bei Crysis aber nur im Einzelspieler-Modus verfügbar. Ein Coop-Modus ist somit für Crysis ausgeschlossen. Im Unreal Editor sind alle Funktionen auch im Mehrspielermodus verwendbar. Unreal Tournament ist im Coop-Modus spielbar.

Außerdem ist das Animieren von Objekten über Keyframes nur im UnrealEd möglich. Es gibt einen extra Animationseditor für die Bewegung von Gegenständen und Kamerafahrten. Diese Bewegungen können im Spiel ausgeführt werden. Die Erstellung von Zwischensequenzen und Machinima ist deshalb mit weniger Aufwand im UnrealEd zu erstellen. Im SandBox2-Editor kann man Gegenstände nur über Zahlen im Menü ändern. Die Bewegungen können, in beiden Engines, auch als Teil einer Skriptsequenz existieren und im Spiel über Trigger ausgelöst werden.

Beide Engines haben Kameraeffekte und können Animationen aus externen Anwendungen importieren.

### 9.5.6 Stabilität und Modding Support

Der SandBox2-Editor stürzt leider bei den verschiedensten Aufgaben ab. Es gibt oftmals keine hilfreichen Fehlermeldungen und eine Suche in Internetforen ergab oft keine Lösung. Es gibt weniger Tutorials im Vergleich zur Unreal Engine und das Modifizieren wird auf diese Art und Weise erschwert. Die im Unreal Editor auftretenden Probleme lassen sich dagegen meist leicht eingrenzen. Viele Foren haben aktive und hilfreiche Mitglieder. Es gibt dennoch öfter unerklärliche Probleme bei der Generierung von Light Maps. Sobald ein mitgeliefertes Package verändert wird, ist eine Neuinstallation des Spiels erforderlich. Es gibt besonders viele Tutorials für Modder und Designer, sodass ein schneller Einstieg möglich ist. Für

Unreal Tournament 3 kamen drei Bücher auf den Markt, die alle Grundlagen für die Entwicklung von Spielen mit dem UnrealEd erklären.

Hinzukommt das Epic Games UT3 lange weiterentwickelt hat. Für Crysis kamen schon früh keine Updates mehr, obwohl die Modding-Community es erwartet hatte. Stattdessen wurde am käuflichen, eigenständigen Add On „Crysis: Warhead“ entwickelt. Für UT3 erschien ein kostenloses Add On.

## 9.6 Fazit und Ausblick

Beide Engines haben Stärken und Schwächen. Der Vorteil der CryEngine2 ist dass Rendering von Licht und Schatten, sowie die realistische Umgebung. Nachteile sind Mangel an Support für Modder, keine Konsolenunterstützung, und weniger Flexibilität, vor allem bei Materials und Animationen.

Für ein Spiel mit viel Grün, wie ein Rollenspiel im Mittelalter-Setting oder eine Neuauflage von Battlefield Vietnam, mit zerstörbaren Environments, ist die CryEngine2, aus optischer Hinsicht, klar im Vorteil. Die Unreal Engine ist hingegen für Konsolen und Vielseitigkeit optimiert.

In beiden Engines ist es möglich Geschichten zu erzählen und sie cineastisch umzusetzen. Crysis und Crysis Warhead sind noch immer die besten Beispiele für die CryEngine2. Für die Unreal Engine 3 sind Bioshock und Gears of War sehr gute Beispiele. In Gears of War ist die Grafik sogar besser als in UT3, obwohl es älter ist. Grund ist die Optimierung im Level Design. UT3 hat teilweise große Maps mit weitem Sichtfeld, auf denen bis zu 40 Spieler gleichzeitig spielen. Gears of War ist für Einzelspieler und Geschichtenerzählung ausgelegt.

Beide Engines werden weiterentwickelt. Crytek arbeitet an der CryEngine3 mit Konsolensupport und die Unreal Engine 4, die bereits seit Jahren in Entwicklung ist, wird erst mit der nächsten Konsolengeneration veröffentlicht. Die Unreal Engine 3

wird aber trotzdem stark weiterentwickelt und es kommen weiterhin Spiele damit auf den Markt.[29]

[29] Videos von Präsentationen der neusten Features beider Engines sind auf DVD im Anhang enthalten.

# 10 Vergleich der Level Design Workflows

Hier sollen die beiden Workflows aus Kapitel 1.5 miteinander verglichen werden. Wie sich herausstellte, haben beide Vor- und Nachteile.

## 10.1 Workflow Epic Games

### 10.1.1 Vorteile

Der Fokus liegt eindeutig auf dem Spielspaß. Das Level wird von Anfang an besser auf Spielbarkeit überprüft.

Die Concept Artists haben eine Grundlage, mit der sie arbeiten können. Sie können evtl. sogar Screenshots von dem Level machen und diese übermalen.

### 10.1.2 Nachteile

Der Levelaufbau wirkt dadurch oft eher inkonsistent und unrealistisch, weil Dinge, wie Fahrstühle oder Säurebäder, ungeschützt und offen im Level platziert werden. Dadurch kann man solche Dinge besser als Spieler nutzen, aber sie wirken unrealistisch und unlogisch.

Diese Vorgehensweise ist für ein kompetitives Onlinespiel besonders geeignet, weil Konsistenz und Handlung zweitrangig sind. Besonders wichtig sind Gameplay und Performance. Das Gameplay wird von vornherein getestet und die Performance wird durch die Assets beeinflusst, die erst nachträglich eingebaut werden.

## 10.2 Eigener Workflow

### 10.2.1 Vorteile

Die Schauplätze wirken aus architektonischer Hinsicht sehr glaubwürdig und die Atmosphäre wird dadurch verbessert.

Die Konzept Art Phase wird schnell abgeschlossen und man spart viel Zeit für die Produktion, sodass das fertige Produkt besser aussehen kann als das Konzept.

Gut, wenn man sehr viel planen will und geeigneter für Einzelspieler Missionen wie in Bioschock oder Half-Life, da man dafür richtige Pläne erstellen kann, um eine Geschichte zu erzählen.

### 10.2.2 Nachteile

Der Spielspaß geht verloren, wenn der Realismus das Wichtigste ist. Das Level sieht zwar aus, wie man es sich vorstellt, aber es macht keinen Spaß darin zu spielen, weil es eventuell weniger taktische Möglichkeiten bietet. Durch die Details wird die Performance negativ beeinflusst.

## 10.3 Persönliches Fazit

Wir haben den Workflow von Epic Games bisher bei keiner anderen Firma gesehen. Die Idee, den Spielspaß so in den Vordergrund zu stellen, ist sehr gut.

Aus zeitlichen Gründen wurde die Planungsphase schnell beendet, um Zeit für die Produktion zu sparen, denn zu dem Zeitpunkt waren auch noch nicht alle technischen Möglichkeiten des Editors bekannt, um, wie bei Epic Games, eine spielbare Alphaversion der Map zu erstellen. Wenn man das Konzept mit dem eigentlichen Level vergleicht, war dieser Workflow der richtige, denn das Level sieht letztendlich besser aus als das Konzept und das ist was zählt. Gameplay-Mechanismen konnten so auch in das realistischere Environment eingefügt werden. Hinzukommt, dass die Betatester auch nachträglich für weitere Inspiration und Motivation gesorgt haben.

# Anhang 1: Literaturverzeichnis

AROUS, Ibrahim; THIELE, Daniel; TRENNT, Hanno, [Projekt B, 2008], Projektarbeit an der HAW: CryEngine2 Crysis – Counter-Strike Modifikation, Hamburg, Deutschland, 2008

BÖGE, Torben; MEYER, Stefan; TSANG, Chor-Hung, [CryEngine2, 2008], Projektarbeit an der HAW: Documentation of CryEngine2, Hamburg, Deutschland, 2008

BRATHWAITE, Brenda; SCHREIBER, Ian, [Competetive, 2009], Comepetetive Gamers“ in Challenges for Game Designers, Boston, MA, USA, 2009

BRATHWAITE, Brenda; SCHREIBER, Ian, [Core, 2009], The Core of a Game “ in Challenges for Game Designers, Boston, MA, USA, 2009

BRATHWAITE, Brenda; SCHREIBER, Ian, [Game, 2009], “Challenges for Game Designers”, Boston, MA, USA, 2009

BRATHWAITE, Brenda; SCHREIBER, Ian, [Game Design, 2009], “Challenges for Game Designers”, Boston, MA, USA, 2009

BRATHWAITE, Brenda; SCHREIBER, Ian, [Professional, 2009], Professional Players“ in: Challenges for Game Designers, Boston, MA, USA, 2009

BRATHWAITE, Brenda; SCHREIBER, Ian, [Skill, 2009], Elements of ‘Strategic’ Skill “ in: Challenges for Game Designers, Boston, MA, USA, 2009

BUSBY, Jason; PARRISH, Zack, [Intro, 2007], "002_Intro_to_UnrealEd", in: Unreal Tournament 3 (Special Edition) Bonus DVD Content, 3D Buzz, Inc., 2007

BUSBY, Jason; PARRISH, Zack; Van Eenwyk, Joel, [Level Design, 2005],"Mastering Unreal Technology – The Art of Level Design, Indianapolis, Indiana, USA 2005

EPIG GAMES (Hrsg.), [Technology, 2008], "Current technology – Unreal Engine 3", in: Unreal Technology 2005-2008, (http://www.unrealtechnology.com/features.php?ref=editor) [21.5.2009]

EPIG GAMES (Hrsg.), [Success, 2008], "Success Stories", in:

Unreal Technology 2005-2008, (http://www.unrealtechnology.com/success-stories.php) [23.6.2009]

EPIG GAMES (Hrsg.), [Partners, 2008], "Partners Program", in:

Unreal Technology 2005-2008, (http://www.unrealtechnology.com/partner-program.php) [23.6.2009]

MEYER, Norman [Game-Engine, 2005], "Constructive Solid Geometry", in Diplomarbeit: „Game-Engine: Das Spiel mit der Maschine als Lebenswelt", 2005

MORRIS, Jeff,[ Level Design, 2007]," Behind the Scenes of Unreal Tournament 3" in: Making_of_UT3_720p.wmv, ab 5:50min

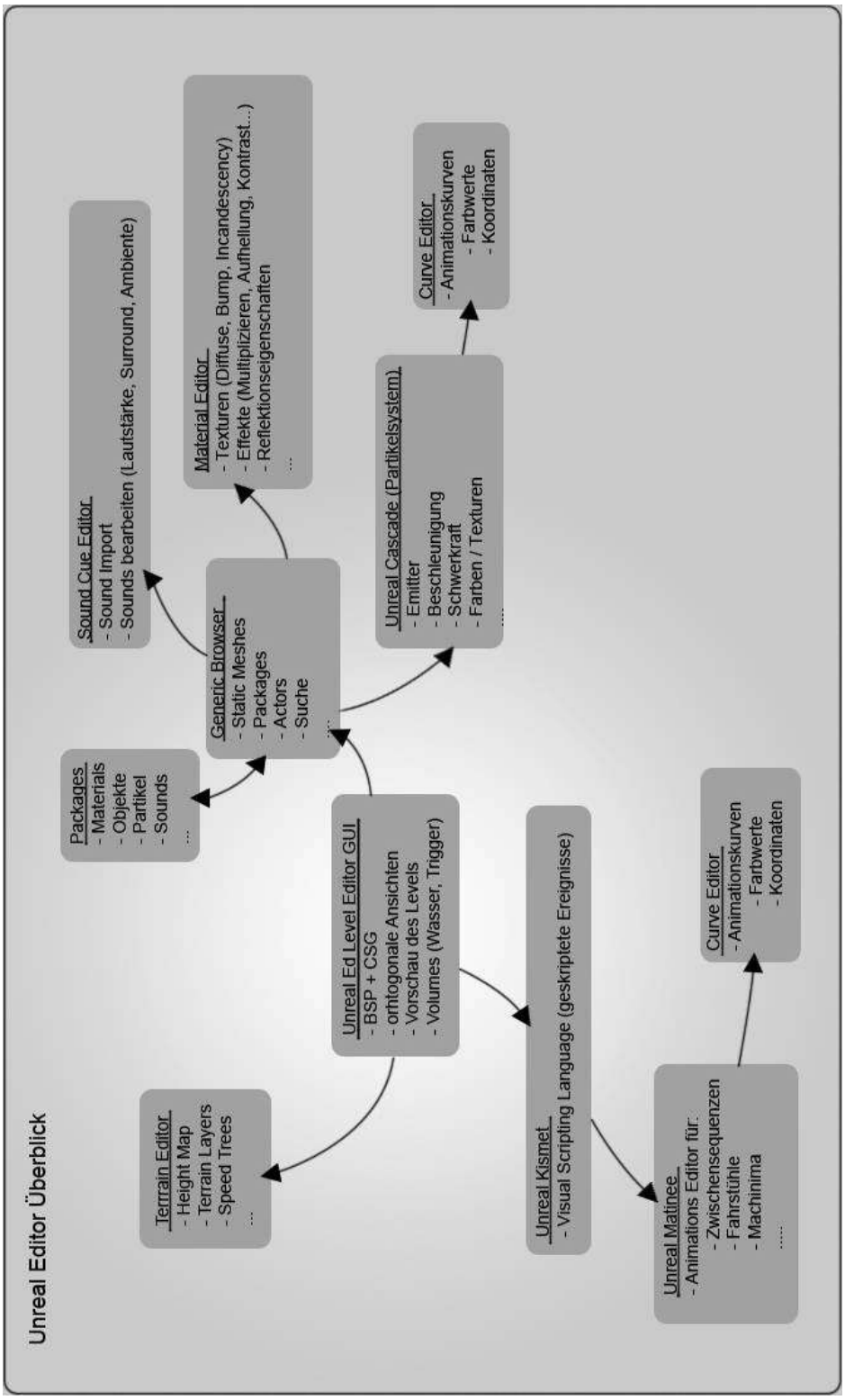
SALEN, Katie; ZIMMERMAN, Eric, *[Fundamentals, 2004]*, Rules of Play - Game Design Fundamentals, Massachusetts London, England 2004

WIKIPEDIA (Hrsg.) [Level, 2009]: „Level (video games)“, in: Wikipedia - Die freie Enzyklopädie, 11. August 2009, (http://en.wikipedia.org/wiki/Level_(video_games)) [11. August 2009]

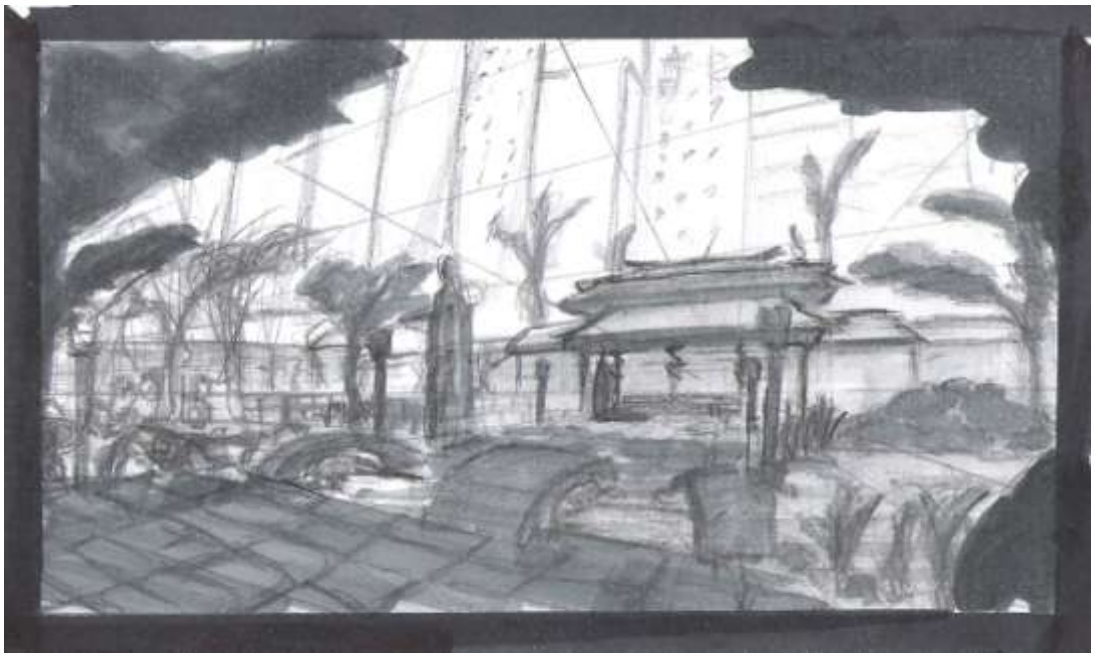
NACHNAME, Vorn, [velweise, jahr], Überschrift, name des Buches bzw. der Arbeit

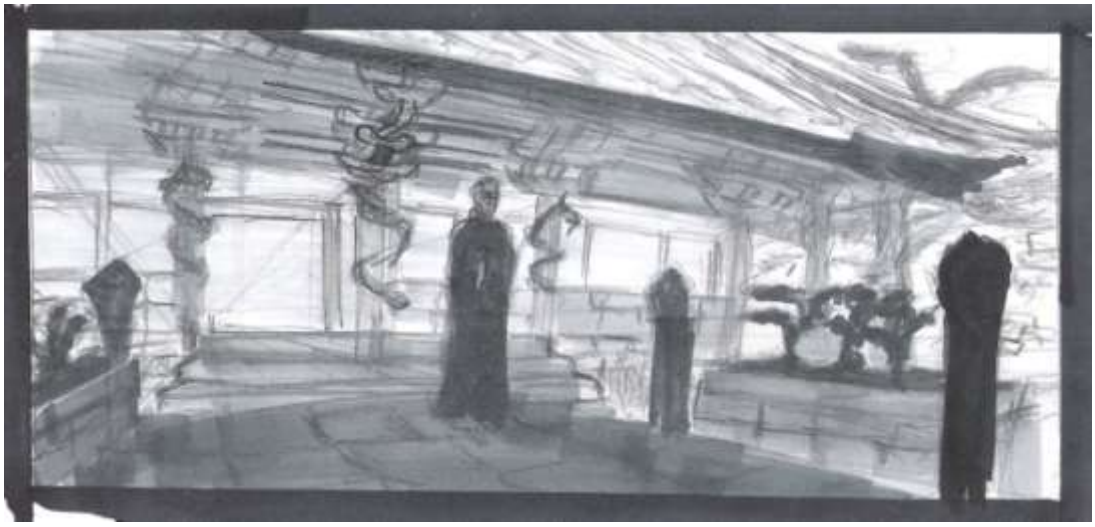
Siehe SCHERFGEN, David, Programmierung, 2003, S. 88f.

# Anhang 2: UnrealEd Überblick



# Anhang 3: Thumbnails





# Anhang 4: Concept Art