Kursunterlagen zum Seminar **Spielräume**

Dipl.-Ing. Mathias Fuchs
(Mathias.Fuchs@uni-ak.ac.at)

Kommunikationstheorie
am Institut für Experimentelles Gestalten und Raumkunst
an der Universität für angewandte Kunst in Wien

Linz, am 28. Februar 2001

Die folgenden Seiten sind als Lehrbehelf und kleines Nachschlagewerk für alle Studentinnen und Studenten gedacht, die sich mit der Geschichte der computergestützten Spiele, mit der ästhetischen Theorie der Spiele und mit der Praxis des Spieledesigns auseinandersetzen möchten. Da die Entwicklung der Software zum Gamedesign rasant voranschreitet, empfiehlt sich der wiederholte Besuch der nachfolgen aufgelisteten Websites.

Die Kursunterlagen stützen sich auf Webtutorials bei:

http://www.unreal.de/

http://www.unrealized.com

http://www.epicgames.com

Weitere Links finden sich auf

http://synreal.netbase.org/links.htm

einer sehr guten Linksammlung,

die Max Moswitzer zusammengestellt hat.

Beispiele zu Unreal-Levels, die Mathias Fuchs und Sylvia Eckermann in den vergangenen Jahren entwickelt haben, finden sich auf:
http://www.t0.or.at/~fuchs-eckermann/projects/

Inhaltsverzeichnis

1. Games People Play - Every Night and Every Day

„Eine Epoche kann durch die ihr eigenen Spiele charakterisiert werden“, behauptet Roger Caillois und fährt fort: „Es erscheint uns nicht unsinnig, die Diagnose einer Zivilisation so zu beginnen, daß wir fragen, welche Spiele in dieser Zivilisation besonders prosperieren.“ [Caillois, 1967]

Welche Spiele prosperieren also in unserer Epoche, unserer Zivilisation?

Ist es Tomb Raider, ist es Super Mario, sind es Fernsehspiele wie Big Brother oder Taxi Orange? Die Einschätzung unseres zivilisatorischen Zustandes müßte davon abgeleitet werden können, daß wir einen Nachmittag im römischen Kolosseum zur Zeit Hadrians vergleichen mit einem viktorianischen Bridgeabend im Brighton des 19. Jahrhunderts und dieses wiederum mit einer Frag-Night[1] im East-End New Yorks. Wer noch tiefer absteigen will in die Gruft der Zivilisationskritik, vergleiche damit die Fernsehspiele, die das österreichische Fernsehen als Abendbelustigung anbietet. Seit dem Zeitpunkt, zu dem Caillois seinen Frage stellte, sind 40 Jahre vergangen und die Spielelandschaft hat sich radikal verändert. Aus den harmlosen Gesellschaftsspielen Halma, Monopoly und Mensch-Ärgere-Dich-Nicht entwickelten sich First Person Shooter und Multiplayer „Shoot‘ em Ups“[2]. Aus ebenso freundlichen wie leicht-seichten Fernsehprogrammen im Stile von Dalli-Dalli wurden exhibitionistische Survival-Tests, die den neo-liberalen Wertekodex ins Wohnzimmer transportieren. Diejenigen Spiele, die als Lernhilfen für autochtone Wohnzimmerlämmer Sozialdarwinismus unterrichten sollen, erfüllen eine Funktion, von der Caillois meinte, „sie unterrichten und richten die Spieler ab, errichten Werte und bestätigen inseitig Gewohnheiten oder Vorlieben.“ In diesem Sinne muß man Wettbewerbe, in denen jeder Fernsehdarsteller gegen seine Teamgefährten und das Publikum gegen alle kämpft, als verschärfte Version der Gladiatorenkämpfe sehen. In letzteren konnte immerhin an cäsarische Milde oder die Barmherzigkeit des Publikums appelliert werden. Was aber bedeutet es, wenn das Prinzip der Belohnung der Spieler (Wünsch Dir was) durch ausschließliche Negativsanktionen (Big Brother) ersetzt wird? Nichts Gutes möchte man meinen.

Andererseits wächst in der Spielelandschaft dieses Jahrzehnts ein Pflänzchen, das am technologischen Boden von Kampfspielen das Primat des Hirnes vor der Faust behauptet: Wissensspiele, Erkenntnisräume, computergestützte Bildungssysteme. In subversiver Verkehrung des Grundsatzes „schneller - simpler - größer“ entstehen Spiele nach dem Motto „intelligenter - komplexer - lustvoller“. In den Game-Designer Labs und Universitäten in London, Tokyo und der amerikanischen Westküste entwickeln Teams von Wissenschaftern, Künstlern und Designern Spiele, deren Spielraum der Wissensraum ist. In diesen Spielen, die als Edutainment bezeichnet werden können, versucht sich Erkenntnis das zurückzuerobern, was Nietzsche als „Fröhliche Wissenschaft“ schon fast verloren geglaubt hat. Das Terrain, das Wissensspiele reklamieren, ist das der Einheit von Erlebnis und Erkenntnis. Die Verbindung zwischen Erfahrungsmoment und begrifflicher Konzeption ist ein Erkenntnisprinzip, das Wissenschaftskonzepten galiläischer, archimedischer oder newtonscher Prägung bekannt war. In der modernen Naturwissenschatt ging diese Verbindung verloren, verkehrte sich sogar in ihr Gegenteil, das dann in einen grundsätzlichen Zweifel am Wert von sinnlicher Erfahrung mündete. Zum Zeitpunkt, zu dem die Einsichtigkeit physikalischer, kosmologischer und psychologischer Zusammenhänge sich aufgrund zunehmender Komplexität verdunkelte, schoß die industrielle Erfahrungssubstitution mittels Film, Radio und Unterhaltungstechniken aus dem Boden. Es entstand etwas, das 100 Jahre später als „Industrial Light & Magic“ bezeichnet werden sollte. Die technologisch aufgelösten Erfahrungssubstrate des 19. Und 20. Jahrhunderts kreierten ihre Inhalte allerdings „neben der wissenschafltichen Realität“ und nicht in ihr. Während es im 18. Jahrhundert noch als chique galt, Vakuumpunpen, prismatische Experimente und mechanische Automaten als gesellschaftliche Belustigung aufzufahren, trennten sich

[1] Frag-Events sind LAN-Parties, zu denen Spieler mit ihren Computern an einen vereinbarten Platz kommen, um die Computer mittels eines lokalen Netzes zusammenzuschließen und daraufhin miteinander zu spielen.

[2] Insbesondere die Spiele Doom (1993), Quake (1996) und Unreal (1998)

die Wege vom Wissen über die Welt und vom Genuß am Verständnis der Welt in den darauffolgenden Jahrhunderten. Wissen und Staunen, Begriff und „Ergriffenheit“ [Wiener, 1995] gingen getrennte Wege: Während erstere in den Hörsälen und Seminaren stattfanden, öffneten sich für letztere Musik- und Kinosäle, Panoramen, Weltausstellungen und Themeparks. Damit einher ging die Vorstellung vom weltfremden Wissenschafter und dem blutleeren Professor, der zwar unglaublich viel wisse, aber keine Ahnung vom Leben habe. Es schien, als wären Erfahrungslust und Wissensgenerierung für immer voneinander getrennt: Getrennt durch räumliche Disposition, getrennt durch personelle Spaltung, getrennt auch durch die Verschiedenartigkeit der jeweiligen Apparaturen.
Spätestens in den 50er Jahren trat der Computer als doppelköpfige Maschine zwischen Erkenntnis- und Genußsuchende, indem er sich als Wissenschaftstool par excellence und als Unterhaltungsmaschine ohne Konkurrenz etablierte. Nachdem das anfängliche Vorurteil vom Computer als „Rechenmaschine“ ausgeräumt war, wurde klar, daß der Computer kein Zahlenmanipulations- sondern ein Symbolmanipulationsinstrument war: Der Computer entwickelte sich zum potentesten Musikinstrument, zum vorherrschenden Bildmanipulationswerkzeug und zum beliebtesten Spielzeug. Gegenwärtig transformiert sich der schlechte Ruf der Computerspiele, die vor allem im protestantischen Europa als „primitiv“, „brutal“, „unmoralisch“ und „arbeitszersetzend“ galten, in eine Wertschätzung, die die östlichen Kulturen längst internalisiert haben, und die bei uns von den Kindern an die Alten weitergeleitet werden muß. Unsere Zivilisation, die Huizingas Postulat vom „Homo Ludens“ [Huizinga, 1955] dergestalt pervertierte, daß sie sich in verantwortungslos spielende Kinder und lustlos arbeitende Alte spalten wollte, beginnt zu realisieren, daß Spielräume der Schlüssel zu Erkenntnisprozessen sein können, die Bildung, Kultur und Entertainment synthetisieren: Edutainment eben.
In virtuellen Wissensräumen versuchen verschiedene Game-Designer, komplexe Erfahrungszusammenhänge und Informationen auf der Basis einer Game-Engine zu implementieren. Es entstehen Wissensräume, in dem die Besucherinnen und Besucher wie Spieleheldinnen im Stile einer Lara Croft auf Entdeckungsreisen gehen können. Dabei gelangen sie in eine Welt, in der es keine Grenzen der Disziplin und Branche gibt. Verknüpfungen zwischen Exponaten, Objekten und Ideen entstehen aufgrund der assoziativen Nähe und nicht aufgrund tradierter Ordnungssysteme. Im Knowledge Space des Projektes „Expositur“ beispielsweise gelangt man von der Prothesensammlung des Technischen Museums zur „Prothesengott“ Theorie Sigmund Freuds, von dieser zum Freudschen Zyklamentraum, von dort zu Blumenornamenten, zu volkskundlichen Blumendarstellungen, zu orientalischen Ornamenten, von dort zu pakistanischen Lastwagenbemalungen und so weiter. Im virtuellen Wissensraum werden die willkürlich gewachsenen Grenzziehungen zwischen den Wissenschaftsbereichen und den musealen Kompetenzen durchlöchert. Unserer Ansicht nach liegt der Gewinn derartig unsystematischer Vorgangsweise darin, Wissen mit den Möglichkeiten auszustatten, die Freud dem Traum zuschreibt: Information verdichten, Information verschieben, Information skalieren.[3] Im virtuellen Wissensraum wird daher das Diktat des Maßstabs gebrochen zugunsten der Angemessenheit der Größe. Das Diktat der chronologischen Linearität wird gebrochen zugunsten der subjektiven Zeitlichkeit. Das Diktat der Vollständigkeit wird gebrochen zugunsten der Suggestionskraft des Partikularen und des Singulären.
Der virtuelle Wissensraum stellt also ein immaterielles „Département diagonal“ dar, das Wissensgebiete verknüpft, das raumlos und zeitreisend rekonstruiert und dekonstruiert, und das ohne Museumspaläste bauen zu müssen, Wissenskathedralen, Flohmärkte der Erfahrung und Fundgruben der Erinnerung errichtet. Der Gang durch diese Märkte, Hallen und Gruben wird ein Erkundungsverhalten erfordern, das dem bedächtigen Schritt des Museumstouristen entgegensteht: Man wird springen müssen wie Super Mario, laufen wie Sonic the Hedgehog und stets beweglich bleiben wie Pacman. Man kann

[3] Freud spricht von der "Verdichtung" und der "Veschiebung" durch den Traum. Den Begriff "Skalierung" verwendet er meines Wissens nach nicht, er weist aber darauf hin, daß im Traum die Größenverhältnisse der Objekte korrigiert werden wie in alten Gemälden die Größenverhältnisse von Hauptpersonen und Nebencharakteren.

daraus ableiten, daß wir uns den Wissenssuchenden nicht als jemanden vorstellen, der still und ausdauernd sammelt, sortiert und kategorial ordnet, sondern als jemanden, der springt, blitzschnell transformiert und subversiv reorganisiert. Der Wissenssuchende gleicht unserer Ansicht nach dem Spieler.[4]

[4] „Spieler“ soll hier natürlich nicht als Soziotypus des verantwortungslosen Parasozialen verstanden werden, sondern als Wissenssuchender, der auf die erkenntnisfördernde Hilfe des Zufalls, der „Coupure“ (asignifikanter Bruch) [Derrida, 1978] und der „Mystory“ [Ulmer, 1985] setzt.

Caillois beschreibt die vier Grundfesten des Spieles, auf die der Spieler vertraut:

1. **Mimikry** (Simulation)
2. **Agón** (Wettbewerb)
3. **Alea** (Zufall)
4. **Ilinx** (Taumel) [Caillois, 1967]

All diese Aspekte spielerischer Erfahrung lösen Computerspiele ein und all diese Aspekte befördern den virtuellen Wissensraum in ein Erkenntnisfeld, das zwischen Wunderkammer, Archäologie, Feldforschung und Spekulation, Kuriositätensammlung Glücksspiel und penibler Recherche liegt. Der Spieler-Forscher trägt im Sinne Caillois die „Physionomie générale“ seiner Gesellschaft, also die Gesichtszüge und Charakteristika seiner Zeit. Diese Gesichtszüge zeigen, was an der Epoche und der Gesellschaft entwicklungsfähig ist und was stagnieren muß. Am Spiel des Spieler -Forschers zeigen sich „les forces d’une société donnée à tel moment de son évolution.“[Caillois, 1967] Der Gesellschaft eine Entwicklung zu attestieren, die sich in den Formen des Spieles wiederspiegelt, heißt aber auch: das Spiel ernstnehmen. Nur durch eine kulturell-zivilisatorische Arbeit am Ensemble der Spiele kann ein zivilisatorischer Prozess deutlich gemacht werden, dem es nicht in erster Linie ums Spielen geht. Gesellschaftliche Transformationen sind kein Spiel - sondern Ernst. Spiele zeigen uns aber den Zustand einer Gesellschaft auf. Sie verweisen auf Bedrohungen und Ängste, sie signalisieren Auswege und Fluchtlinien, sie breiten Hoffnungsfelder aus und skizzieren Modi künftigen Zusammenlebens, künftiger Kommunikation und künftiger Erkenntnisweisen.
(Mathias Fuchs)

Quellen:
Caillois, Roger: Les jeux et les hommes. Édition Gallimard Paris 1967
Derrida, Jacques: Writing and Difference. Chicago University Press 1978
Freud, Sigmund: Die Traumdeutung. (Erstmallig erschienen in Wien 1900) S. Fischer Verlag Frankfurt am Main 1991
Freud, Sigmund: Das Unbehagen in der Natur. (Erstmallig erschienen in Wien 1930) S. Fischer Verlag Frankfurt am Main 1991
Huizinga, J.: Homo Ludens: A Study of the Play Element in Culture. Beacon Press Boston 1955
Rosenberg, Richard S.: The Social Impact of Computers.
Academic Press, Inc. San Diego, London, Boston 1977
Ulmer, Gregory: Applied Grammatology. John Hopkins University Press 1985
Wiener, Oswald; Wozu überhaupt Kunst? - Beiträge zu den Karlsruher Tagen S. J. Schmidts, 1979. In: Springer Hefte für Gegenwartskunst Band I Heft 2-3. Wien 1995

2. Eine kurze Geschichte der Computerspiele

Herbst 1961 Die Digital Equipment Corp. liefert dem MIT in Cambridge, Mass. eine PDP-1, den ersten Computer mit Tastatur und Kathodenstrahlbildschirm (Monitor). Entgegen der Erwartung, daß darauf wissenschaftliche Programme entstehen würden, entwickeln zwei MIT Mitarbeiter zunächst ein Spiel.
1962 Steve Russel und Dan Edwards programmieren für die PDP-1 **SPACEWAR,** ein Shoot-Up Game, das es erlaubt, am Bildschirm Raumschiffe abzuschießen. Spacewar war wahrscheinlich das erste Computerspiel.

In den darauffolgenden Jahren entstehen auf großen Mainframe Maschinen zumeist in BASIC programmierte Spiele:

LUNAR LANDER, ein textbasiertes Simulationsspiel, das Treibstoffverbrauch, Landegeschwindigkeit und Beschleunigung einer Mondlandefähre simuliert. In Echtzeit wurde am Bildschirm die momentane Fallgeschwindigkeit der Raumfähre sowie ihr Treibstoffvorrat ausgegeben.

HAMMURABI (KINGDOM), eine Simulation von ökonomischen Prozessen, die Steuereinnahmen, Essensreserven, Sterberaten und Gewinne der Verwalter errechnet, ein Vorgänger von SIMCITY.

HUNT THE WUMPUS, ein Spiel, das ein Netzwerk von Tunnels und Knoten enthält. Die ersten Implementationen verwendeten angeblich eine dodekaedrische Struktur. Die Spieler konnten sich in den Tunnels vorwärts bewegen und wurden informiert, sobald sie sich dem „Wumpus“ näherten. Die KonsolenmelDung lautete dann beispielsweise: „You are in node 5. I smell a Wumpus. Move or shoot.“ Darüber hinaus gab es Fledermäuse, die einen in einen anderen Knoten versetzen konnten („I smell a bat.“). Hunt the Wumpus stellt einen Vorläufer der Dungeon & Dragon Spiele dar.

1971 Die erste „Coin-op“ Arcade entsteht in den USA.
1972 Nolan Bushnell gründet ATARI. **PONG** wird zum Schlager für die Arcade Games. Das Spiel
PONG besitzt nur 2 Bildschirmmessages:
„Deposit Quarter“ + „Avoid missing Ball for High Score“
1974 PONG für Heimcomputer wird auf den Markt gebracht.
1975 Der Commodore PET erscheint als Heimcomputer mit einem gewöhnlichen Kassettenrecorder als Massenspeicher.
1976 Fairchild bringt den Channel F, das erste Cartridge-Spiel heraus. Angeboten werden **PADDLE**, **HOCKEY**, **TIC TAC TOE** u.a.
1977 Das Spiel **SPACE INVADERS** stellt ein weiteres Weltraum Shoot-up dar, daß die eigene Überlebenszeit in Abhängigkeit von vernichteten Feindschiffen errechnet. Space Invaders ist das erste Spiel mit einblendbarem High-Score Display.
1978 ATARIs VCS 2600 wird als hybride Station aus Fernseher, Videospiel und Computer ein Verkaufserfolg.
1979 Die erste Spielesoftware Firma, „Activision“ wird von 4 ehemaligen ATARI Mitarbeitern gegründet.
1980 Mattel startet die Firma „Intellivision“. Nintendo veröffentlicht **PAC MAN.** Pac Man wird sowohl in den Game Arcades als auch am Heimcomputermarkt ein Langzeiterfolg.
1981 Nintendo und SEGA beginnen den Export von Spielen in die USA.
1982 **DONKEY KONG** von Nintendo Entertainment Systems (NES). Am Weltmarkt zeichnet sich ein großer finanzieller Crash des Spielesektors ab.
1984 Mattel verkauft seinen Elektronikbereich.
1989 Nintendo lanciert den GAME BOY und das Spiel **TETRIS.**
1990 Nintendos **SUPER MARIO BROS**. spielt weltweit 500 Mio. $ ein

1991 SEGA bringt **SONIC THE HEDGEHOG** auf den Markt.
1993 **DOOM** wird zum populärsten „First Person Shoot 'em Up" Spiel
1994 PC CD-ROM Spiele werden ein ernst zunehmendes Medium.
Debut der SONY Playstation
1995 SEGA Saturn
1996 Nintendo 64, **QUAKE** entsteht als Nachfolgespiel von DOOM als brutales Shoot-Up
1998 Epic Megagames stellt **UNREAL1** vor. Das Spiel besitzt einen Level Editor,der es ermöglicht, selbst Spielwelten zu erzeugen.
1999 Die Entwicklung des japanischen Spieles **FINAL FANTASY VIII** beschäftigt 400 Künstler. Erstmalig werden für Spieleentwicklung Budgets eingerichtet, die diejenigen großer Hollywood FilmproDuktionen übersteigen. Epic Megagames präsentieren **UnrealTournament**, eine Weiterentwicklung von Unreal. SEGA stellt den Dreamcast vor, mit speziellen Interfaces für Anglerspiele, Golf u.a.
2000 SONY PS2 (Playstation 2) wird als Konsole für den Markt der über 15-jährigen vorgestellt. **Heavy Metal F.A.K.K. 2** verdoppelt die Polygonzahl der Akteure gegenüber State-of-the-Art Spielen wie Unreal. Microsoft kündigt an, in den Spielemarkt einsteigen zu wollen.
2001 Microsoft Xbox soll auf den Markt gebracht werden. Angekündigte Prozessor Clock Rate 800 MHz, Graphikkarten Rate 600 MHz.

Vorgeschichte der Computerspiele
1890 erfaßt die USA eine „Coin-op Craze", also eine Begeisterung für Münzautomatenspiele, die damals vor allem von Phonographen ausging. Die Phonographen wurden bevorzugt in Lobbies, Saloons, Eisenbahnbahnhöfen und kleinen Innenstadtlokalen aufgestellt und dienten der arbeitenden Bevölkerung als Zerstreuung in den Mittagspausen. Oft fanden sich im selben Raum Kaugummi Automaten, Süßigkeiten Automaten oder Maschinen mit den man seine eigenen Knochen durchleuchten konnte (X-Rays). Mitte der 1890er lösten die Kinematoskope die Phonographen ab. Nun konnte man Aufnahmen von Sehenswürdigkeiten, athletischen Darbietungen oder pornographische Animationen betrachten. Ende der 1890er ersetzte das Mutoskop die Kinetoskope. Mutoskope waren interaktive Kinetoskope, bei denen man über einen Hebel die Abspielgeschwindigkeit, die Abspielrichtung oder die Fixierung auf Standbilder regeln konnte.

3. UNREAL (Epic Games) ein Echtzeit 3D-Environment

„Do something Unreal today!"
(Verkaufsslogan von Epic Games)

Im Jahr 1998 brachte die kleine Softwarefirma Epic Megagames das Programm UNREAL heraus, das vom Programmierer und Firmenchef Tim Sweeney in C++ entwickelt worden war und sich zur Unterstützung der extrem schnellen 3D Raumsimulationen auf die

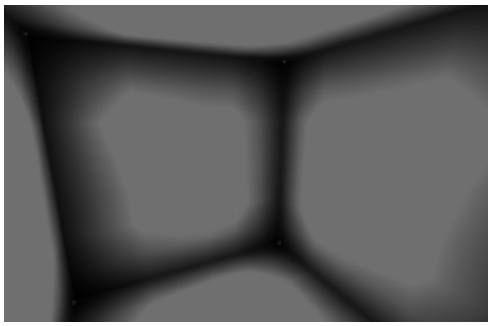
Fähigkeiten der Voodoo Grafikkarten stützte. Das Spiel beinhaltet einen Editor, mit dem es möglich ist, selbst 3D Levels zu erzeugen, die nicht nur die mitgelieferten Texturen und Sounds verwenden, sondern es erlauben, Texturen (im PCX Format), 3D Objekte (im DXF Format) und Soundfiles (im WAV Format) zu importieren. Zusätzlich bietet UNREAL über UNREALSkript, eine Skriptsprache, die verwandt mit Java und C ist, die Möglichkeit Objekte der implementierten Klassen (z.B. Monster, Effekte, Wasser, Nebel usw.) in ihrem Verhalten zu beschreiben. Zur Zeit wird UNREAL für Windows Plattformen ausgeliefert. Der einfache Player ohne den zugehörigen Editor ist auch für LINUX und Macintosh Plattformen erhältlich.
1999 folgte UnrealTournament, das eine überarbeitete und verbesserte Version von UNREAL darstellt. Mit dem Update, resp. Patch 420 wurde der verbesserte Editor UnrealEd 2.0 ausgeliefert.
Im Sommer 2000 kündigte Epic Games an, eine neue Version Unreal2 in Vorbereitung zu haben, von der im Oktober 2000 in London erste Screenshots und technische Details gezeigt wurden. Der Detailierungsgrad der Akteure in Unreal2 wird etwa 5 mal so hoch sein, wie in der derzeitigen Version.

Abbildung: Screenshot aus einer Unreal2 Szene

4. Der erste Raum

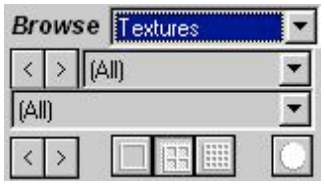
Dieser Abschnitt beschreibt den Bau eines einfachen Raumes in Unreal.

Zuerst wählst Du mit der **rechten Maustaste** (**RM-Taste**) das Icon  aus und danach "**Cube-Properties**...". Hier kannst Du die Eigenschaften wie z.B. Höhe, Breite, Tiefe des Quaders eingeben. 1 World Unit in UNREAL entspricht ca. 2 cm der Realwelt, so ist eine 1,80 m hohe Person 90 UNREAL World Units hoch. In Lightwave ist diese Person 90 m groß. Am besten wählst Du **gerade Zahlen** oder **Potenzen von 2** (z.B. 1024 World Units. Der höchste Raum, den ich jeh gesehen habe, ist 16384 World Units hoch. Das ist im Level "Hall of Giants").
Achtung! Die Eingabe der numerischen Werte muß durch Drücken der **ENTER-Taste** bestätigt werden. Nun klickst Du auf **Build** und die rote Brush hat die von Dir eingegebenen Maße angenommen. Durch Drücken von **STRG-S** (=SUBTRACT) scheidet die rote Werkzeug-Brush (in diesem Fall ein Kubus) unseren ersten Raum zurecht. Dies dürfte dann in etwa so aussehen :

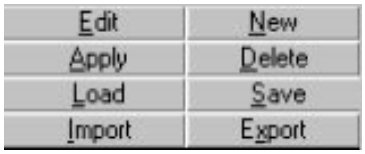


Der Raum ist jetzt mit der Default-Textur überzogen. Dies wollen wir sofort ändern. Also erst einmal mußt Du Dich im 3D-Fenster bewegen können. Dazu drückst Du einfach mit dem Mauszeiger in das 3D-Fenster, hältst die **linke MausTaste** (**LM-Taste**) gedrückt und bewegst die Maus. Um andere Bewegungsmöglichkeiten zu erreichen, drückst Du die **rechte** (**RM-Taste**) **bzw. beide Maustasten**. Gut, nun weißt Du, wie man sich im 3D-Fester bewegt.

Selektieren und Ändern von Texturen: Dazu klickst Du im 3D-Fenster einfach auf die gewünschte Wand (Textur). Um mehrere gleichzeitig auszuwählen, benutzt Du die STRG-Taste, hältst sie gedrückt, und wählst alle Wände (Texturen) nacheinander aus.



Um die Texturen zu ändern, klickst Du einfach auf die gewünschte Textur rechts im Browser.



Um neue Texturen zu laden benutzt Du "**Load**" und lädst eine der Textur-Dateien aus deinem Unreal bzw. Unreal Tournament "Textures" Ordner. Um zwischen den geladenen Texturdateien zu wählen, selektierst Du die gewünschte Textur im Textur-Browser.

So, nun hätten wir einen texturierten Raum. Du möchtest natürlich wissen, wie man ein Objekt, z.B. einen Würfel in diesen Raum einfügt. Gut, dazu gehst Du wie zu Anfang vor, und wählst wieder mit der RM-Taste den Cube oder ein anderes Tool.
Um nun diese Brush zur Welt hinzuzufügen, drückst Du **STRG-A** . Vorher kannst Du allerdings die Brush noch bewegen. Dazu gehst Du in eines der 3 anderen Fenster (über 3D-Fenster = Ansicht von oben / die beiden anderen jeweils die Ansicht von einer Seite) und bewegst die Brush mit gedrückter LM-Taste und STRG an die gewünschte Stelle in Deinem Raum. Danach kannst Du ihn mit STRG-A an der neuen Position zur Welt hinzufügen. Am besten Du experimentierst hier ein bißchen.
Mit **RM-Taste und STRG kannst Du die Brushes drehen**.

Nun wird der "**Playerstart**" in dein Level eingefügt. (Dort startet man im Level) Dazu wählst Du im Browser "**NavigationPoint**" aus und dann "**Playerstart**". Jetzt gehst Du in das 3D-Fenster, klickst mit der RM-Taste an die Stelle wo der "Playerstart" sein soll und wählst **"Add Playerstart here"** (durch Gedrückthalten der RM-Taste öffnet sich ein Popup-Menü) aus. Schon ist er richtig plaziert.

Zum Schluß mußt Du noch "**Rebuilden**", d.h. den Level berechnen lassen und speichern. Dazu drückst Du **F8** und wählst die Karteikarte **BSP**. Hier mußt Du "**Auto Lightning**" deaktivieren, da wir keine Lichter verwendet haben. Wie die funktionieren, findest Du unter "Lichter (farbiges Licht und LensFlare)". Jetzt nur noch zurück auf die Karteikarte "**Geometry**", ein Klick auf "**Rebuild Geometry**" und fertig ist der Level. **Speichern** kannst Du mir **STRG-L** und **spielen** mit **STRG-P**. Unter Unreal Tournament sollte man die **Bots allerdings vorher ausschalten**, da man sonst ständig stirbt.

Jetzt kannst Du deinen ersten Raum betreten und Dich darin bewegen.
Mit der Maus blickt man nach links und rechts, oben und unten und mit den Cursorkeys (Pfeiltasten) geht man vorwärts, rückwärts oder nach rechts und links. Immer wenn Du die **SPACE-Taste** drückst **hüpft** dein Spieler in die Höhe. Wenn Du die **c-Taste** drückst (engl.: crouch) duckt sich der Spieler und bewegt sich solange die Taste gedrückt ist, in der Hocke.

Es gibt eine ganze Reihe von **Spezialtasten**, die bestimmte Funktionen auslösen:

F1: Zeigt die Spieleinformation

F2: Statistische Information

F11: Helligkeit heraufsetzen

„Minus"-Taste: Icons mit Systeminformation schrittweise ausblenden.
o-Taste: nächste Waffe in die Hand nehmen
p-Taste: vorherige Waffe in die Hand nehmen
v-Taste: Du kannst eine akustische Meldung auslösen in Abhängigkeit von weiteren gedrückten Tasten (z.B. „v"-2-4)
ESC-Taste: Spiel abbrechen

4.1. Der Gruppen-Browser / Die Hilfsfenster

Zurück zum Editor. In seiner einfachsten Form sieht der Gruppen-Browser des Editors harmlos und schmächtig aus, genau so wie ihn unsere Abbildung: 01 zeigt, doch mit etwas Geschick, kann er alle anderen Einzel-Browser schlucken und als ein Fenster anzeigen:

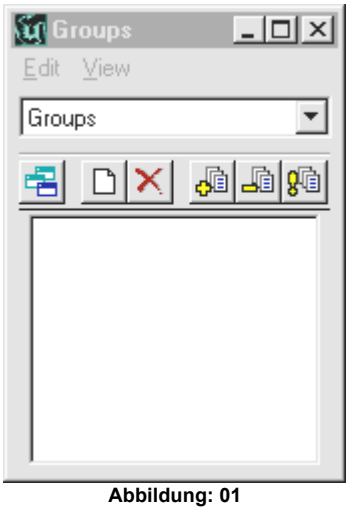


Abbildung: 01

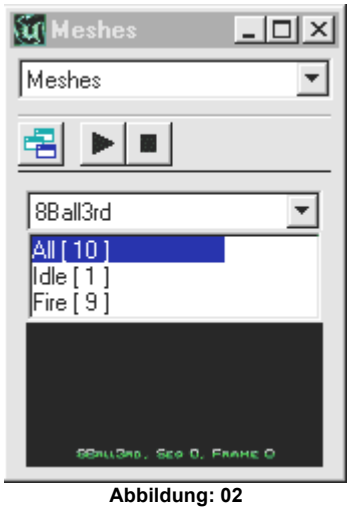


Abbildung: 02

In Abbildung: 02 haben wir den Mesh-Browser als Bestandteil des Gruppen-Browsers abgebildet und in Abbildung: 03 hat sogar der komplette Textur-Browser in der Docking-Station platz.

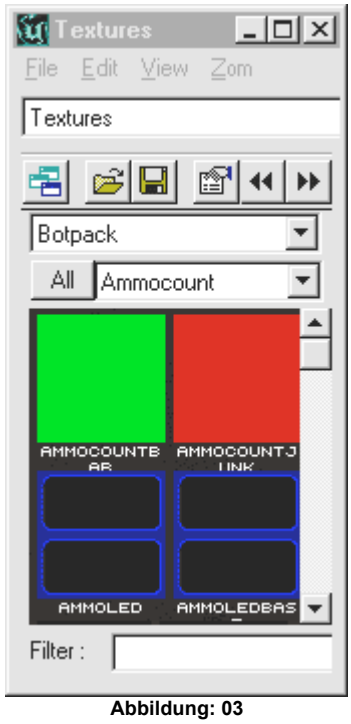


Abbildung: 03

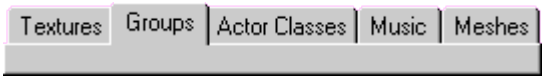
Generell kannst Du alle Fenster, die diesen Button aufweisen in den Gruppen-Browser integrieren:

Toggle Dock Status (ab UnrealED 2.0 Version 430)

Durch einen Klick auf diesen Button wird der Ein/Auslink-Vorgang des jeweiligen Fensters ausgeführt.
Um den Gruppenbrowser zu öffnen mußt Du auf diesen Button klicken:

Group-Browser (ab UnrealED 2.0 Version 430)

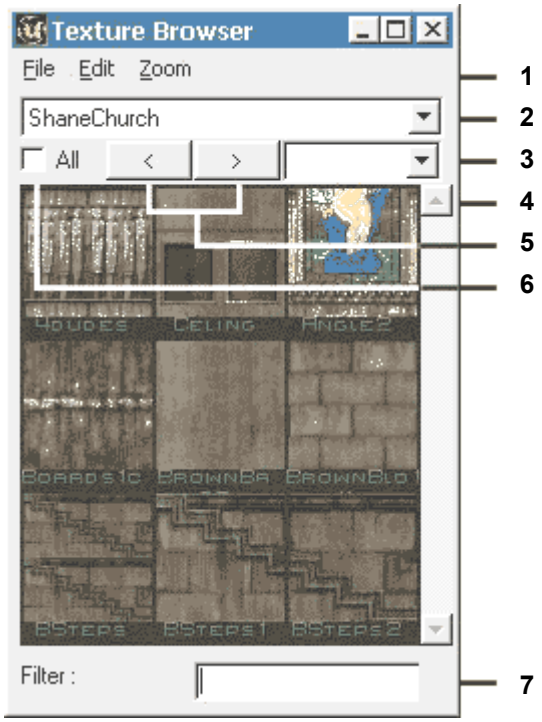
Über dieses Menü kannst Du ganz einfach alle einzelnen angedockten Browser aufrufen:



Im nachfolgenden Teil haben wir nochmals alle Hilfsfenster aufgelistet und zeigen euch, was man damit alles machen kann:

4.2. Der Textur-Browser

Wenn Du die oben stehenden Schritte gemacht hast, ist Dir sicher aufgefallen, daß die bis jetzt von Dir gebauten **Actors** (= alle Gegenstände, die in den Ansichten des UnrealED sichtbar sind) nur eine häßliche Default-Textur besitzen. Über das Menü "**View/Texture Browser**" kannst Du Dir das Fenster mit den Texturen anzeigen lassen:



1. Menüleisten Optionen

2. Name des Textur-Packets

3. Verwendungszweck

4. Texturauswahl

5. Vor/Zurück (steuert Option Nr.3)

6. Zeige alle Texturen an

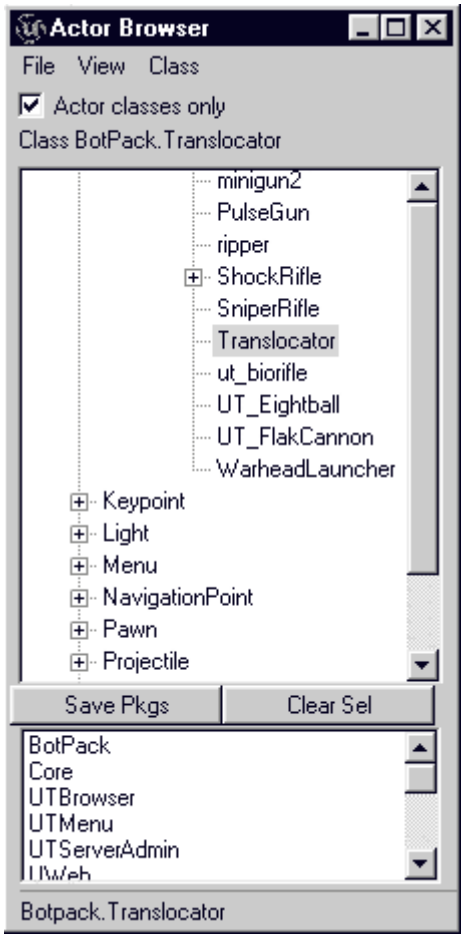
7. Filter

Mit dem Textur-Fenster kannst Du einen Actor mit Texturen ausstatten. Baue erst alle wichtigen Komponenten Deines Levels mit der standardmäßig vorgegebenen Textur. Sie ist sowieso immer geladen, wenn das Spiel oder der Editor gestartet wird und verbraucht somit auch keinen unnötigen Speicherplatz. Wenn jetzt ein Teil des Levels erstellt wurde, dann ist es an der Zeit ein paar schönere Texturen anzubringen. Wähle zuerst ein Textur-Packet an, indem Du auf "File/Open", in der Menüleiste des Textur-Browsers klickst.

Alle Texturen liegen im Verzeichnis deines Unreal-Spiels (suche das "Textures"-Verzeichnis). Wir empfehlen für den Anfang ein mittelgroßes Texturpacket zu wählen, wie etwa "Mine.utx" oder "Crypt2.utx" Wenn es geladen wurde, sollte es im Textur-Browser angezeigt werden. Markiere danach jeweils die zu texturierenden Flächen in der 3D-Ansicht des Editors an. Selektiere mehrere Flächen gleichzeitig mit gedrückt gehaltener "STRG"-Taste. Alle selektierten Flächen werden jetzt mit der neuen Textur ausgestattet, wenn Du sie nochmals im Textur-Browser anklickst. Achte darauf, daß die Texturen in Kategorien eingeteilt sind, die ihren Verwendungszweck kennzeichnen (Floor/Ceiling/Wall u.s.w.).

Wenn Du eigene **Texturen importieren** willst, mußt Du sie entweder als **.pcx** oder als **.bmp** vorliegen haben. Ausschließlich diese beiden Formate werden unterstützt. So kann man natürlich auch nur **.pcx oder .bmp Dateien exportieren**.
Für Unreal1 galt: Nur .pcx Dateien mit einer Resolution von 8 Bits/ pixel (d.h. 256 Farben).
Für UnrealTournament sind auch 24 Bit Bitmap Images möglich.

4.3. Der Actor-Browser

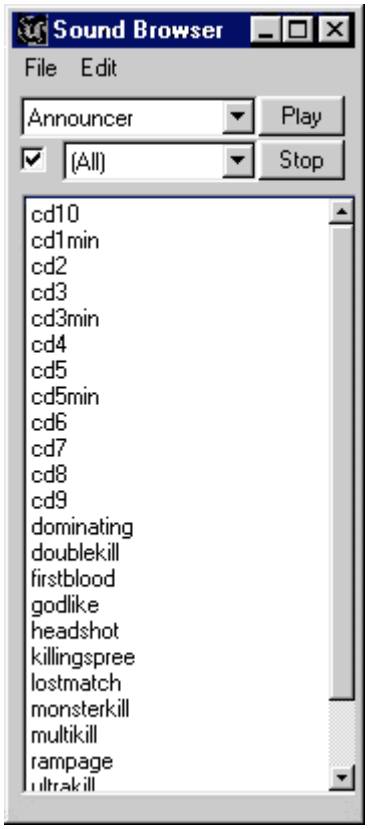


4.4. Der Sound/Music-Browser

Der "Actor-Browser" ist ganz neu in UnrealED2, dieser ersetzt den "Classes-Browser" aus UnrealED1. Hier kannst Du alle Actors finden, die Du früher im Klassen-Browser gefunden hast. Im unteren Feld wählst Du ein entsprechendes Paket aus (hier werden alle verfügbaren Pakets angezeigt), im oberen Feld mußt Du die Klasse selektieren (sie muß farbig unterlegt sein). Dann klickst Du mit der RM-Taste auf die Stelle, an welcher der neue Actor erzeugt werden soll und klickst im Kontextmenü auf "**Add Actor here**". Das kleine Feld mit dem Haken (ganz oben) "**Actors classes only**" sollte immer angewählt bleiben. Es beschränkt die "Classes" für deine Map auf nützliche Arten und blendet viele für Dich nicht brauchbare Klassen aus.

Um ein verändertes Paket abzuspeichern, kannst Du entweder über das Hauptmenü (für dieses Fenster getrennt vom restlichen Editor), oder über die Button-Leiste in der Mitte zwischen den beiden Fenstern benutzen.

Wann Du welche Actors benutzen mußt, und welche Effekte diese Actors dann ausüben, erklären wir ausführlich in den einzelnen Tutorials, die jeweils einen oder mehrere dieser Effekte und deren Anwendung detailliert schildern.



Der "**Sound-Browser**" hat sich kaum verändert im Vergleich zu UnrealED1. Weil er dem Musik-Browser recht ähnlich ist, sparen wir uns dessen Erläuterungen. Um beide Browser zu benutzen mußt Du eigentlich immer wie folgt vorgehen:

Öffne über "**File/ Load Packet**" ein Soundpacket ("Unreal/Sound"-Verzeichnis) deiner Wahl. Der Name des Pakets erscheint dann im oberen hellen Feld des Browsers. Das etwas kleinere darunter zeigt die einzelnen Kategorien des Pakets an, ähnlich wie im Textur-Browser.

Dann mußt Du nur noch einen speziellen Sound selektieren, benutze dazu das große weiße Feld. Über die beiden Tasten "**Play**" und "**Stop**" kannst Du kurz den Sound anhören und ihn, falls er loopen sollte auch wieder stoppen. Um einen Soundeffekt in einer Map anzuwenden, muß immer eine spezielle Klasse im Level gesetzt werden, der dann der Sound im "Properties"-Fenster zugewiesen werden muß. Lese hierzu auch unsere anschließenden Tutorials.

4.5. Das Scripting-Modul

Das "**RichText Modul**" des UnrealED1 hat sich in der neuen Version zu einem ansehnlichen, ausladenden Fullsize-Anwendung gemausert. Das bekannte blaue Fenster wird von einer Browser-Leiste unterstützt, welche die geladenen Klassen anzeigt.

Um eine Klasse in das Modul zu laden, mußt Du im Menü auf den Button mit dem "**Hammer & Zange**"- Icon klicken. Eine geänderte Klasse muß mit den weißen Menü-Knöpfen rekompiliert werden, bevor sie mit dem blauen Button gespeichert werden kann.

Bitte beachte, daß die Buttons mit dem **Fernglas** eine **Suchfunktion** darstellen und nicht wie im großen Editorfenster als "Undo/Redo"-Funktion agieren. Sicherlich nicht gerade ein glücklicher Zustand!

5. Die Clippingtools

5.1. Das Teilen und Zerschneiden von Formen

Diese Tools sind seit Erscheinen des neuen UnrealEditors 2.0 implementiert und erleichtern die Arbeit mit dem Editor erheblich. In ihrer Funktion sind die Clippingtools vergleichbar mit einer Schere, d.h. sie schneiden die gesetzten geometrischen 3D-Gebilde genauso, wie die roten Brushes zurecht. Der Vorteil ist vor allem, daß keine Clippingfehler auftreten.

5.2. Anwendungsbereiche und Funktionen

Anwendung der Clippingtools bezogen auf die rote Cube Brush.

hiermit wird die Brush geschnitten, also Überflüssiges abgetrennt und gelöscht.

auch hiermit wird die Brush geschnitten, also Überflüssiges abgetrennt und gelöscht.

Anwendung der Clippingtools bezogen auf einen Cube aus fester Materie

hiermit wird feste Materie geschnitten und der restliche überflüssige Teil gelöscht

hiermit wird feste Materie geschnitten und gleichzeitig halbiert. (Es bleiben 2 Hälften)

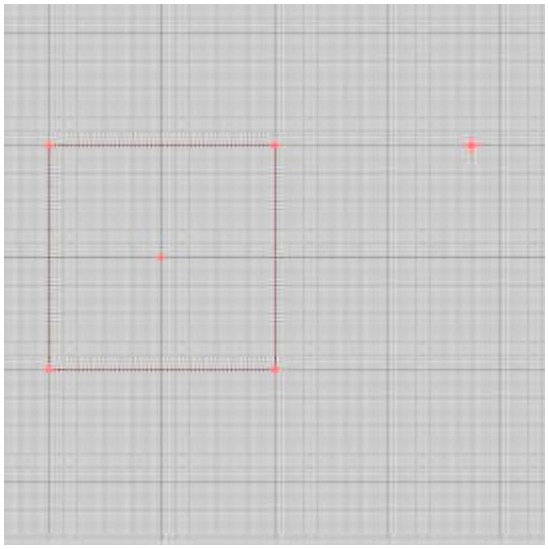
hiermit werden alle gesetzten Marker gelöscht

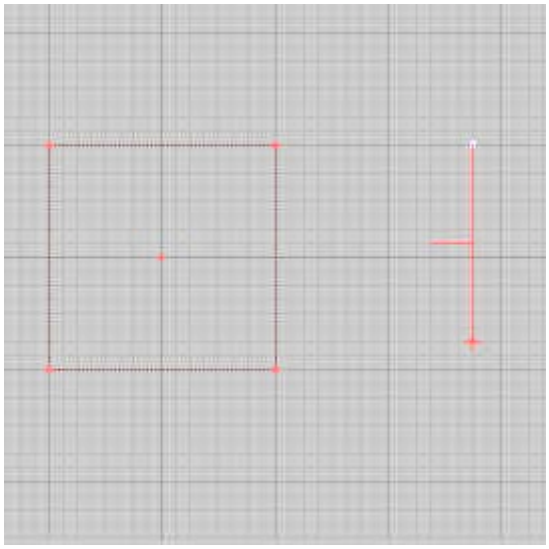
hiermit wird die gesetzte Marker-Linie gedreht/geflippt

Tastenkommandos

STRG + RM-Taste:	Pivot setzen	(STRG halten + RM klicken)
STRG + LM-Taste:	Pivot bewegen	(STRG halten + LM gedrückt halten)

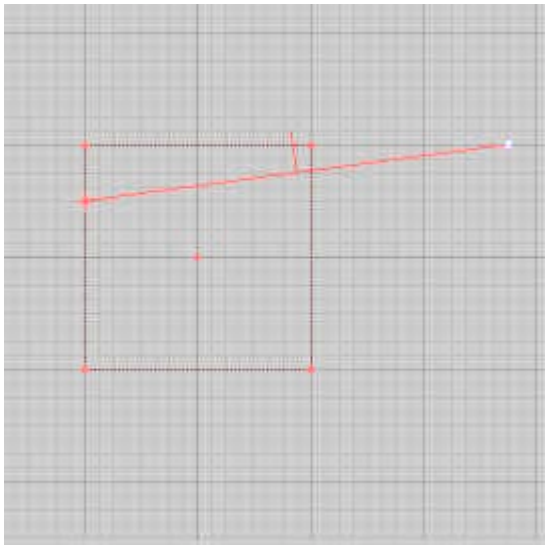
5.3. **Clippingtools /** für eine Brush

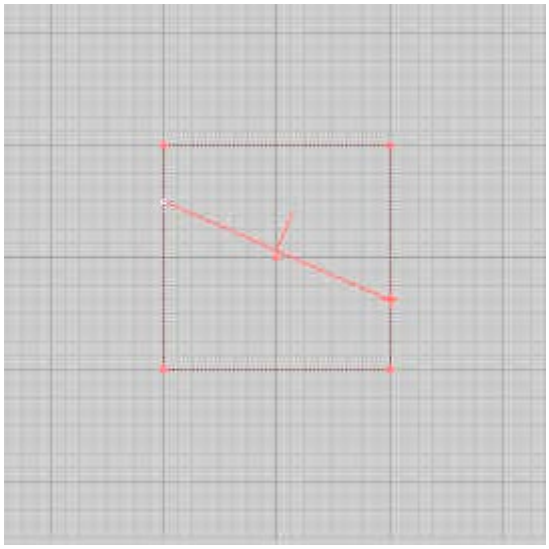
1.  2.



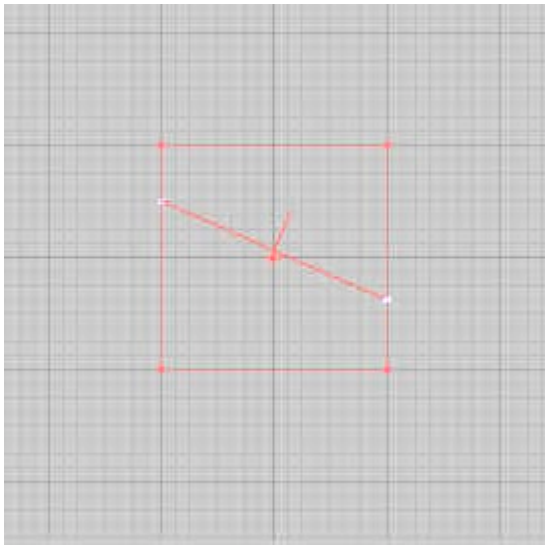
1. Setze eine Cube-Brush (Größe ist egal) und selektiere das folgende Werkzeug (Brush Clipping):

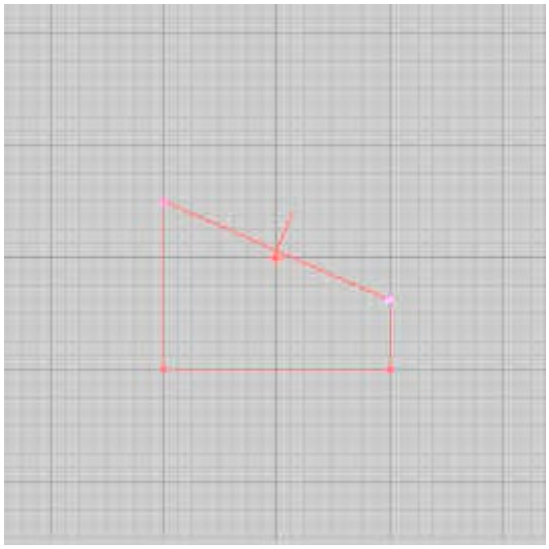
Klicke nun bei gedrückter STRG-Taste mit der rechten Maustaste an irgendeine Stelle in das 2D Fenster, um einen Clipping Point zu erzeugen.
2. Diesen Schritt muß man noch einmal wiederholen. In dem Moment, in dem Du den zweiten Point setzt, werden beide Points mit einer roten Linie verbunden. Diese Line zeigt eine Normale, die die Richtung angibt, in der später weggeschnitten wird.

3.  4.



3. So nun klickst Du den einen Point an, so daß das rote Kreuz genau darauf plaziert ist. Nun STRG- und die LM-Taste gedrückt halten und den Pivot auf einen der Knoten (Vertex) der Brush verschieben und absetzen.
4**.** Nun den zweiten Pivot auf die gleiche Art und Weise wie den ersten verschieben und auf die andere Seite legen. Siehe Beispiel.

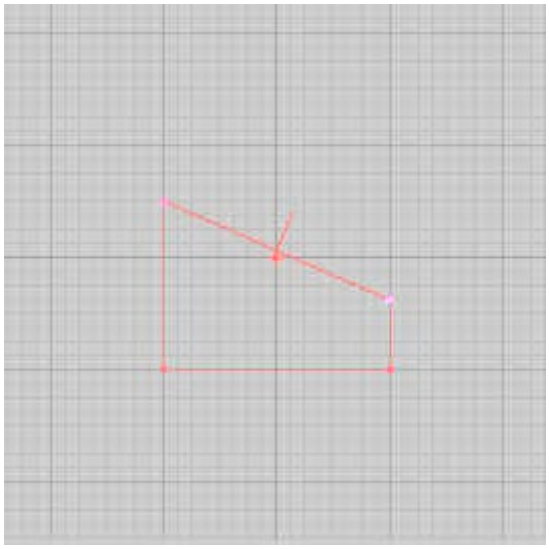
5.  6.



5. Nun kommt das wichtigste überhaupt, klicke nun die rote Cube-Brush an, so daß diese rot hervorgehoben ist. (Siehe Bild)
6. So nun kommt der letzte Teil. Gehe in die Clippingtools und drücke einen der folgenden Button aus der UnrealED2 Werkzeugleiste. Es ist egal welchen, denn beide haben im Bezug auf eine Brush die gleiche Auswirkung.

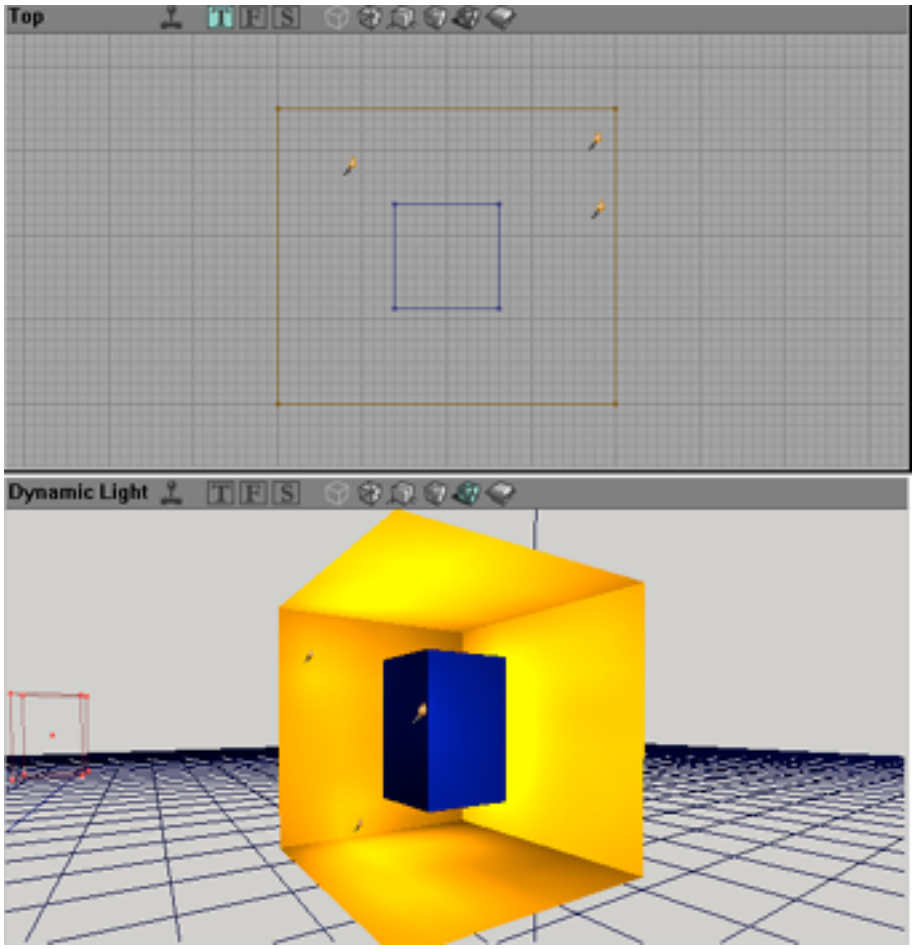
 oder 

Beachte: Die Normale gibt an, welcher Teil weggeschnitten wird!
(Weggeschnitten wird der Teil, auf den die Normale zeigt.)


Nun sollte ein Teil die Brush abgeschnitten sein.

7. Das Erstellen des Eckpunktes geht wieder über die RM-Taste und das Kontextmenü:
"Split Side" oder "CTRL+I"

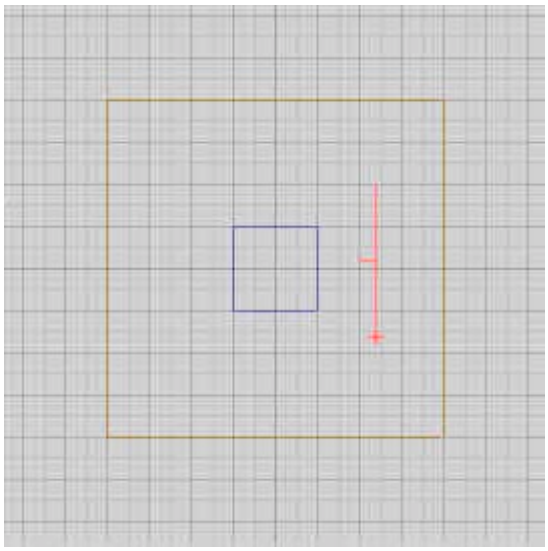
5.4. **Clippingtools /** für feste Materie

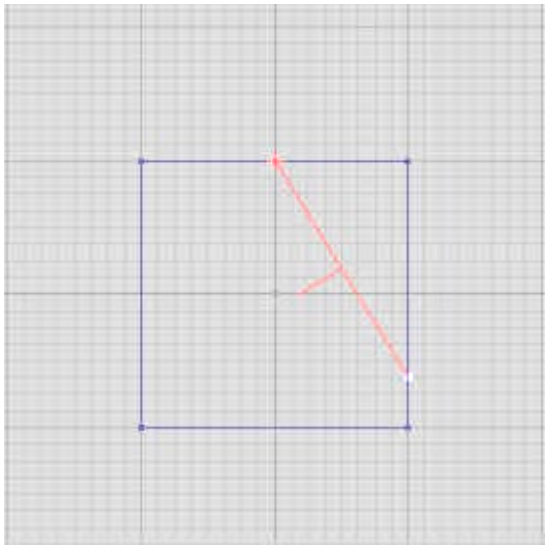
1. 

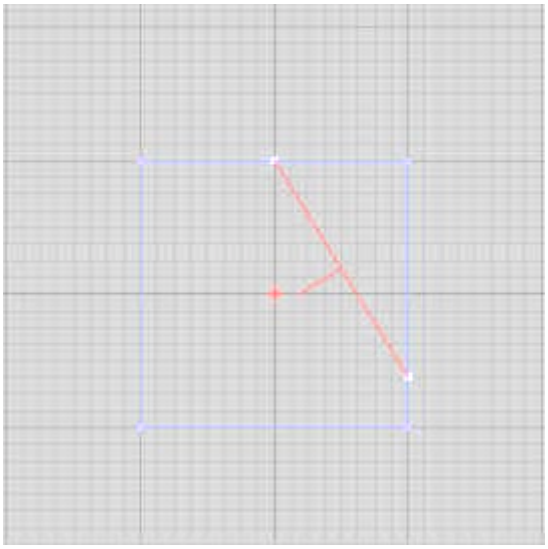
1. Wir arbeiten jetzt mit dem gleichen Schema wie schon oben beschrieben.

Baue einen Raum mit einem Kubus, der mittels "Add" in der Mitte des Raumes gesetzt wird.

2. Im nächsten Schritt setzen wir wie gehabt die Clipping Points.

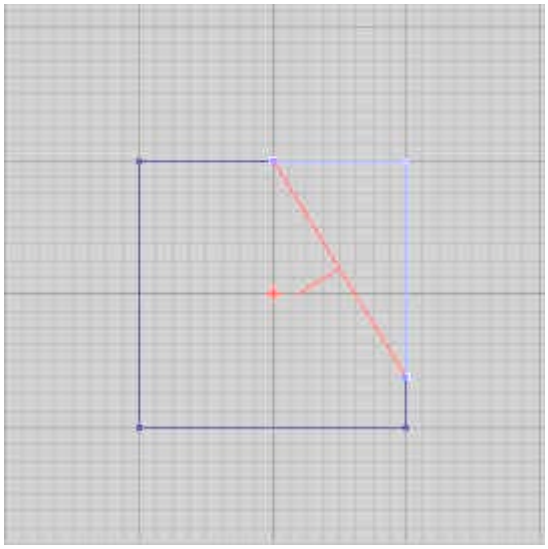
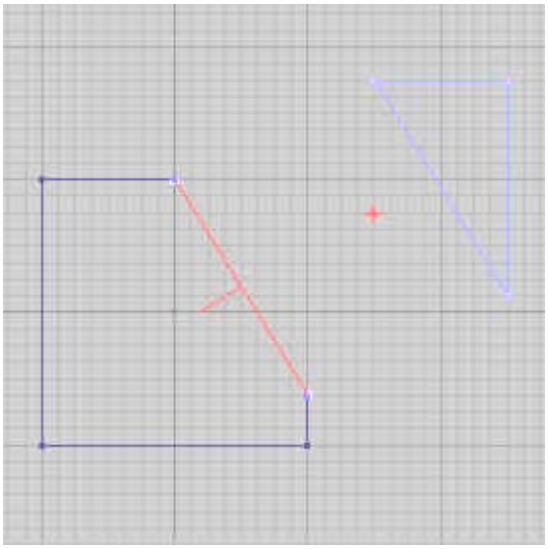
2. 

3. 

4. 

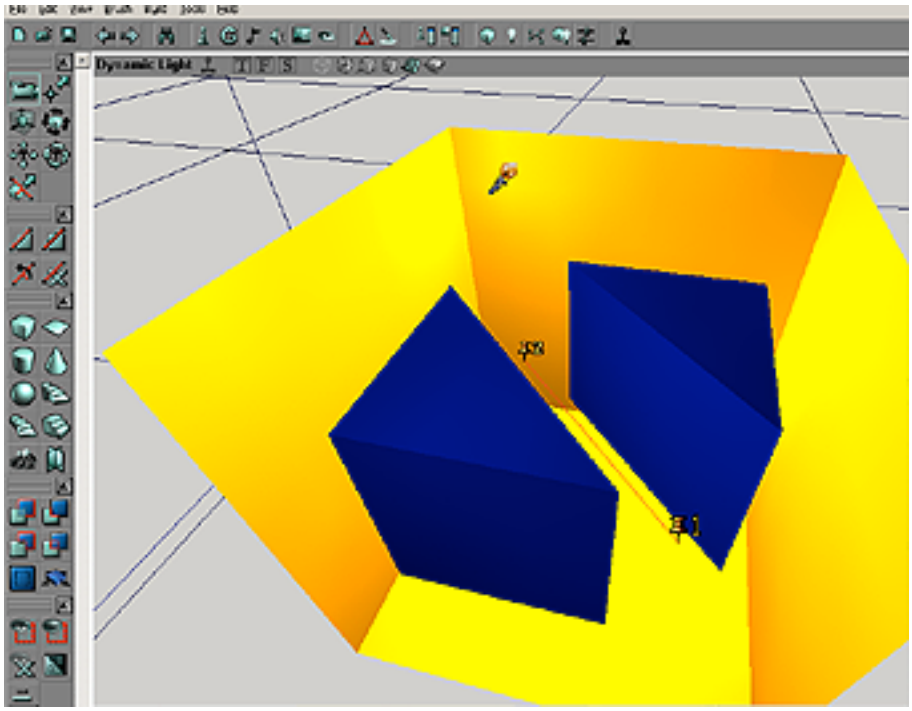
3. Richte beide Pivots nach dem gleichen Schema aus, so daß diese in der Aufsicht (Top View) quer stehen.

4. Die Cube Brush, wie in der Abbildung gezeigt markieren. Also einfach anklicken.

5.  6. 

5. Jetzt den Button „**Split Clipping Brush**“ drücken. 

6. So nun kannst Du den Kubus trennen, so daß Du 2 Hälften erhältst.



7. Das Endresultat nach dem Rebuilden (**F8**) zeigt die Abbildung.

6. Mover / Türen / Plattformen

6.1. Bewegliche Level-Elemente

In den nächsten Arbeitsschritten werde ich erklären, wie man auf einfachste Art und Weise einen sogenannten **Mover** erstellt.

Alle Dinge die sich später einmal in der Unreal Tournament Welt bewegen sollen, sind Mover oder müssen Mover werden. Es gibt unterschiedliche Variationen von Movern. Folgende Dinge stellen immer Mover dar: Türen, Aufzüge, bewegte Wände, Knöpfe, die gedrückt werden müssen, explodierende Wände, zerbrechendes Glas, Fahrzeuge die eine bestimmte Route abfahren etc. Es sind also deinen Phantasien keine Grenzen gesetzt. Im allgemeinen wird der Standard Mover benutzt, selten kommt es vor, daß hier die Mapper entsprechend variieren.

6.2. Wir bauen einen Aufzug

Erstellen einer Plattform, welche sich beim Betreten nach oben bewegt

1. Als erstes erstellen wir einen Raum mit den Maßen 1024 x 1024 x 1024 und fügen Ihn unserer Welt hinzu.
2. Nun bauen wir eine Plattform, welche die Maße 16 x 128 x 128 hat und fügen diese unserem Raum hinzu.
3. Wenn Du die Texturen ändern möchtest, dann mache es jetzt, denn dies ist die letzte Möglichkeit, um Einfluß darauf zu nehmen.
4. Nun erstelle eine neue Brush mit den Maßen 32 x 160 x 160. Die rote Brush nun so um die Plattform setzen, daß sich diese in der Mitte der roten Brush befindet.
5. Den Button „Intersect“ drücken. Nun sollte sich die rote Brush um die Plattform gesetzt haben (wie eine Haut, vgl. Abbildung:01) .

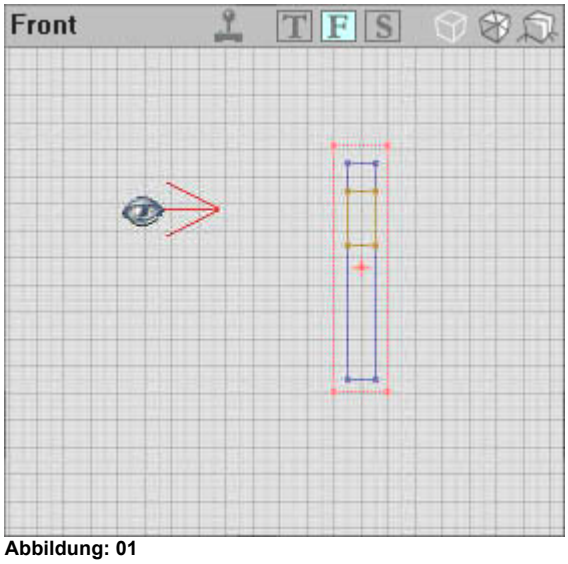


Abbildung: 01

6. Die rote Brush an die Stelle bewegen, wo später Dein Mover sein soll.
7. Jetzt den Button „**Add Mover**“ drücken und schon wird aus der **roten** Brush eine **lila** Brush.

8. Die alte Plattform (**blaue** Brush), kann nun gelöscht werden, da diese nicht mehr benötigt wird.

9. Um dem Mover auch die Bewegungen, die er machen soll zu zeigen, müssen wir ihm entsprechende **Keys** vorgeben. Unter „Keys“ versteht man wie in einem Animationsprogramm Positionen von bewegten Objekten, zwischen denen interpoliert werden kann.

10. Klicke den Mover an, so daß er hell hervorgehoben leuchtet, nun die RM-Taste drücken und unter **Mover Keyframe** den **Key 1** wählen. Nun hat der Mover die Info, daß die Ausgangsposition unten am Boden ist (vgl. Abbildung: 02).

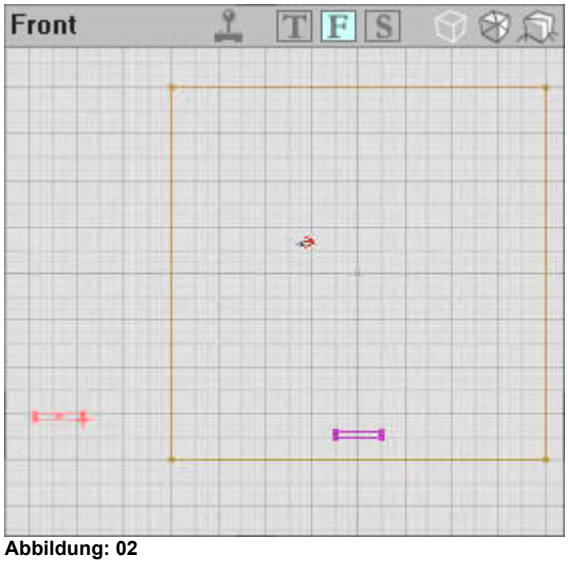


Abbildung: 02

11. Jetzt den Mover (lila) anklicken und bei gedrückter STRG-Taste und LM-Taste im 2D Fenster, den Mover in die Position bringen, in der er sich bewegen soll. In unserem Falle ist es nach oben (vgl. Abbildung: 03).

12. Danach drücke wieder die RM-Taste und gehe auf „**Mover Keyframe**“ und teile ihm **Key 0** zu. (=Base). In dem Moment, wo Du diesen Key vergibst, rutscht der Mover wieder an seine Ausgangsposition, also **Key 1**, am Boden.

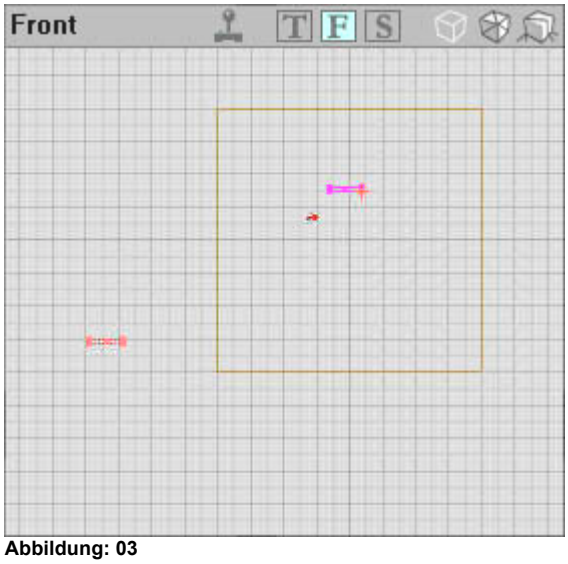


Abbildung: 03

13. Nun ist der Mover soweit korrekt eingestellt und kann getestet werden.

Jetzt nur noch einen **Playerstart** und **Licht** setzen, **Rebuilden** F8 und schon hast Du Deinen ersten funktionierenden Mover, der sich bei Berührung der Plattform nach oben bewegt.

6.3. Wir bauen eine Tür

1. Wir gehen im Grunde genommen nach dem gleichen Schema vor, nur daß unsere Brush, die später die Tür darstellen soll, die Masse 128 x 128 x16 bekommt.
2. Nun schneiden wir in diese Tür noch ein nettes kleines Sichtfenster mit den Maßen 32 x 32 x 16.
3. Jetzt wieder eine neue Brush erstellen, die größer sein soll als unsere Tür (siehe Punkte 1.4 und 1.5).
4. Hier kann nun nach dem gleichen Schema weitergemacht werden wie bei unserer Plattform (siehe 1.6 bis 1.8).
5. Um unserer Tür jetzt auch die richtige Pendelbewegung zu zuteilen, ist es wichtig, auch die richtige Achse, die später die Bewegung beschreiben soll, einzustellen (vgl. Abbildung: 04).
6. Wähle mit der Maus die Bewegungsachse aus und klicke seitlich auf den Pivot Punkt, so daß dann ein rotes Kreuz entsteht (zur Not 2 mal anklicken).

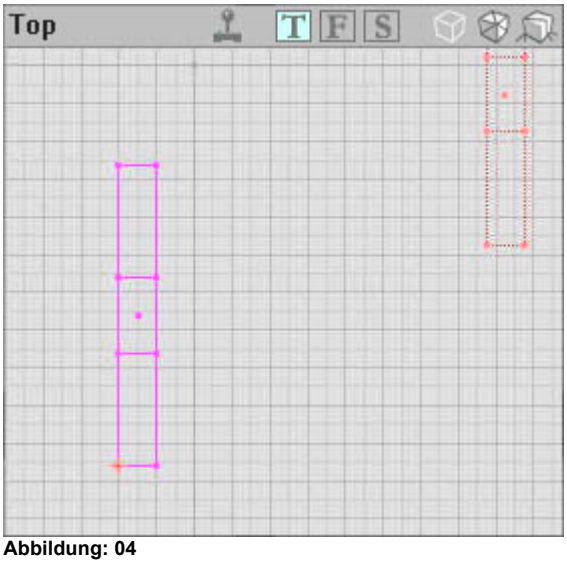


Abbildung: 04

7. Jetzt hast Du die Achse, die später maßgebend für die Pendelbewegung Deiner Tür verantwortlich ist.
8. Jetzt den Mover wieder anklicken, so daß dieser hell hervorgehoben ist, hier unbedingt auf die Achse achten, wenn das Kreuz wieder in der Mitte ist, einfach noch mal den Punkt 2.6 durchführen.
9. Nun mit STRG und RM-Taste den Mover/Tür auf die Achse ausrichten. Man muß auch die Punkte 1.10 bis 1.13 wieder ausführen.
10. Hier ein paar Bilder betreffend die Türachse und wie sie in Position zu bringen ist.

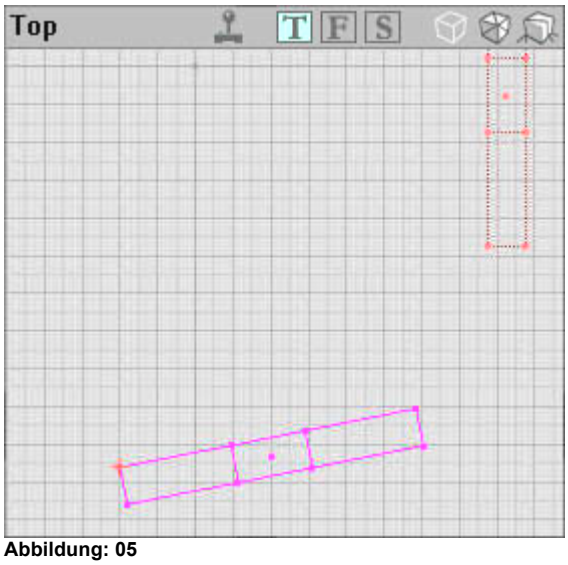


Abbildung: 05

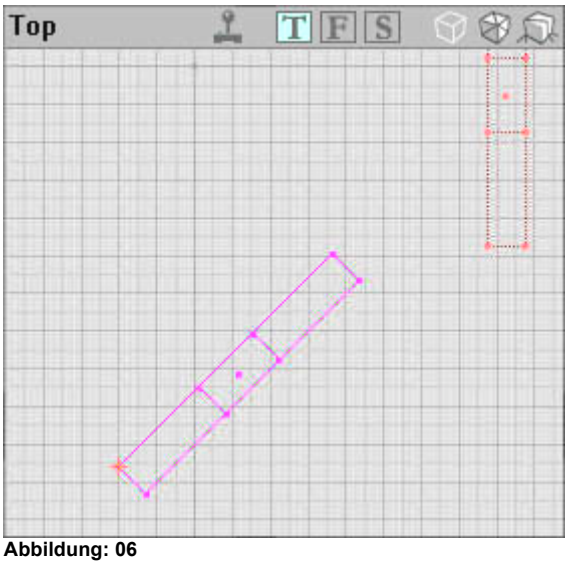


Abbildung: 06

11. Im Moment gehe ich noch nicht auf die weiteren Feineinstellungen ein, diese kommen zum Schluß dieses Tutorials.

6.4. Wir bauen eine automatische Tür (Trigger)

Tür öffnet sich, sobald man sich ihr nähert. (ohne Berührung)

Trigger sind Actor-Classes, die zum Auslösen bestimmter Aktionen benutzt werden. Auch hier gibt es verschiedene Variationen.

1. Die bisher beschriebenen Mover werden durch „Berührung" aktiviert also duch Dagegenstoßen.

Dieses ändern wir nun so, daß die Tür sich öffnet, wenn Du Dich an einem bestimmten Ort in Deinem Levels befindest.

2. Gehe in die **Properties** Deines Movers und ändere unter **Objekt / InitialState** BumpOpenTimed auf TriggerOpenTimed (vgl. Abbildung: 07).

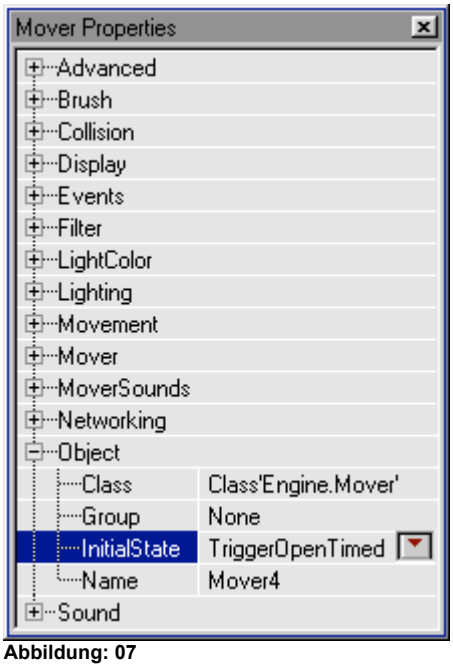


Abbildung: 07

Dadurch wird der Mover deaktiviert und kann per Dagegenstoßen nicht mehr aktiviert werden. Gleichzeitig teilen wir Ihm mit, daß dafür jetzt ein Trigger zuständig ist.

3. Gehe nun in die Classes und wähle TRIGGER, diesen fügst Du nun zu deinem Level an die gewünschte Stelle im Raum hinzu.

4. Doppelklick auf dem Trigger - öffnet die Properties. Hier unter dem Punkt **Events** folgende Einstellung machen. **Event = Dooropen / Tag = Trigger**

5. Nochmals die Properties Deines Movers öffnen und unter **Events die Option "Tag" auf: "Tag = Dooropen"** ändern.

6. So nun ist der **Trigger aktiviert** und Mover und Trigger sind durch **eine rote Linie verbunden**.

7. Immer wenn Du jetzt in die Nähe dieses Triggers kommst, wird sich die Tür automatisch öffnen. Wenn Du den Triggerbereich vergrößern willst, dann wähle noch mal die Optionen deines Triggers und verändere den Trigger **Radius**. (Das ist nützlich bei Türen, die nach innen aufgehen.)

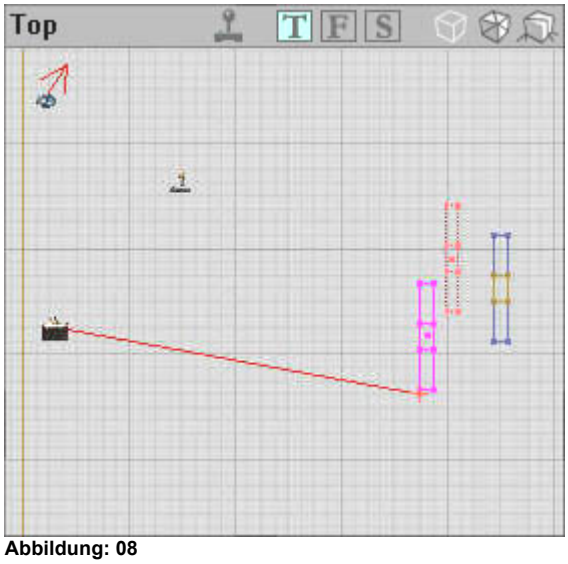


Abbildung: 08

Mit dieser Technik ist es möglich, Türen zu bauen, bzw. zu steuern die ein StarTrek oder StarWars Flair bekommen sollen. (Türen die aus mehreren beweglichen Teilen bestehen. Hierzu brauchst Du dann nur jedem Mover den gleichen Tag zu geben (also n x die Einstellung von Punkt 3.2 und 3.5).

Durch die Verbindung von Triggers - Dispatcher - Movers (mehrere Türen) ist auch eine zeitliche Verzögerungen möglich. (z.B. zuerst öffnet sich Tür1, dann Tür2 und zu letzt Tür3.)

6.5. Ständig bewegte Barrieren im Quake II Stil

So nun bauen wir mal was Schwieriges zum krönenden Abschluß dieses Tutorials.

1. Baue einen Raum mit folgenden Maßen 256 x 768 x 256

2. Setze 2 Lichter an die Decke und 1 Playerstart an die hintere kleine Wand

3. Eine neue Brush mit den Maßen 32 x 256 x 64 bauen. Diese wird unser Mover, der als bewegte Barriere dienen soll.

4. Brush hinzufügen (add) - dann die blaue Brush anklicken und die RM-Taste drücken
- dann **Copy Polygons to Brush** auswählen (Neue Methode!).

5. Die blaue Brush nun löschen und die rote in der 2D Aufsichtsansicht per STRG und LM-Taste so ausrichten, daß diese seitlich zum Raum steht (vgl. Abbildung: 09).

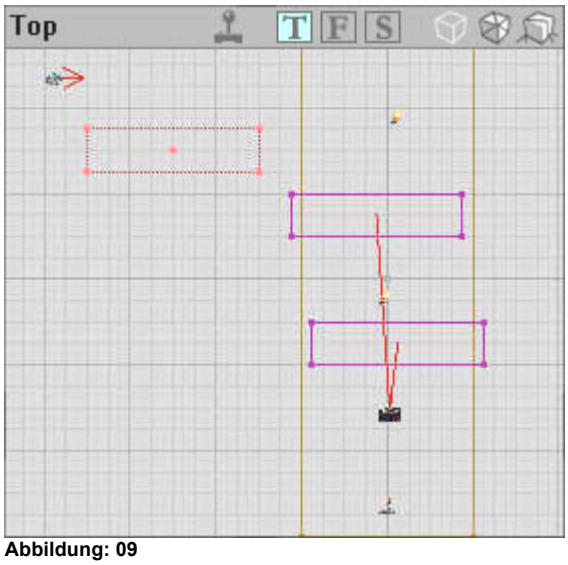


Abbildung: 09

6. Den Mover nun so ausrichten, daß dieser in der Wand verschwindet, hier kann bei der Key Vergabe genauso gearbeitet werden wie bereits vorher erlernt.

7. Am besten baust Du dann gleich noch einen von der Sorte. Klicke auf den Mover und drücke die RM Taste und gehe wieder auf **Copy Polygons to Brush**. Die rote Brush wie gewohnt auch zu einem Mover machen und ausrichten. Allerdings sollte der 2.Mover seitlich versetzt dem 1.Mover gegenüber geaddet werden.

8. So nun in die Classes gehen und unter Trigger einen **TimeTrigger** auswählen.

9. Diesen TimeTrigger Deinem Level "adden" und folgende Einstellungen in den **Properties** vornehmen:

Kapitel

Eintrag

Änderung

Events

Event

stoss

Tag

TimeTrigger

TimeTrigger

bRepeating

setze auf "True"

Trigger

RepeatTriggerTime

setze auf "1.000000"

10. Folgende Einstellungen in den **Mover Properties** vornehmen.

Kapitel

Eintrag

Änderung

Events

Event

none

Tag

stoss

Mover

bDamageTriggered

setze auf "True"

DamageThreshold

1.000000 (wie lange der Mover bei Aufprall steht)

MoveTime

setze auf "0.500000"

StayOpenTime

setze auf "1.000000"

MoverEncorach

"Me_crashWhenEncorach" (damit es auch splattert)"

11. Diese Einstellungen kannst Du bei beiden Movern verschieden machen, damit es abwechslungsreicher wird.

12. Nun noch Rebuilden und Saven und fertig ist der Hindernis Gang (Wenn man nicht aufpasst wird einem hier der Arsch abgerissen.)

Das fertige Gebilde sollte dann etwa so aussehen, wie es unsere nebenstehende Abbildung: 10 zeigt!

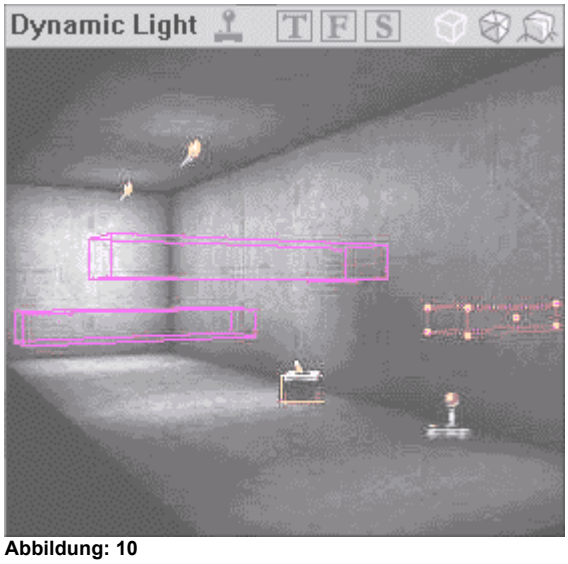


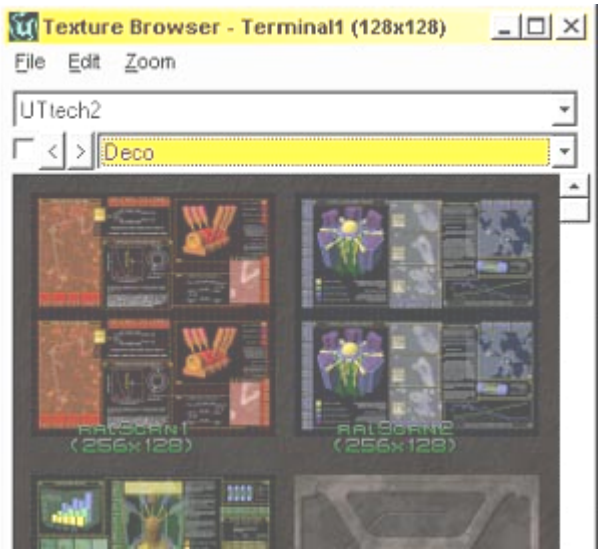
Abbildung: 10

7. **Triggered Textures** (Slideshow)

Diese Tutorial befaßt sich mit Triggered-Textures. Das sind Oberflächen, die durch einen **Schalter** ihre Textur wechseln. Um die einzelnen Schritte nachvollziehen zu können, mußt Du die grundlegenden Funktionen des Editors beherrschen.
Mit folgenden Werkzeugen solltest Du also bereits gearbeitet haben:

 **Würfel**
 **Hinzufügen**
 **Abziehen**
 **Intersect**
 **De-Intersect**

Zunächst mußt Du ein Objekt erstellen, auf das Du die wechselnde Textur legen willst. Ich habe hier eine Art Schautafel erstellt. Die Tafel ist 128 x 256 Einheiten groß, die Tiefe spielt keine Rolle.



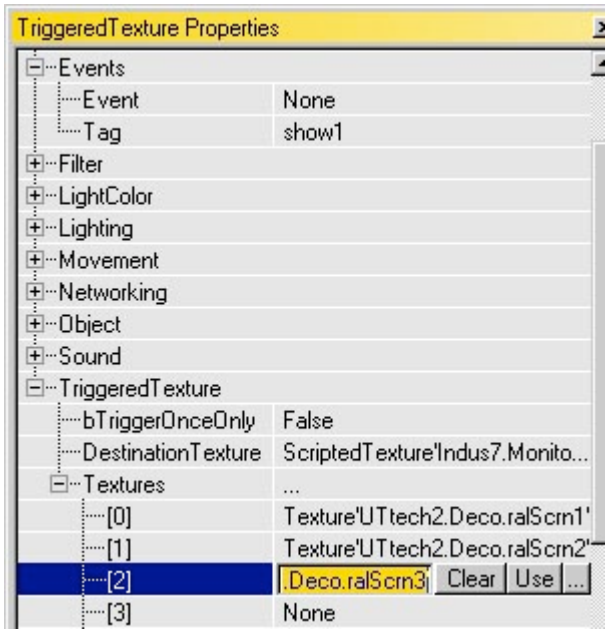
Die 3 Texturen, die wechseln sollen, befinden sich in der Datei **UTtech2.utx** im Package **Deco**. Es sind gleich die drei obersten.

Damit auch nichts schief geht solltest Du eine der Texturen zunächst probeweise auf das Objekt legen und schauen ob sie auch richtig paßt. Du kannst **Scaling** in den **SurfaceProperties** anwenden (und die anderen Parameter ggf. auch ändern).

In unserem Fall mußt Du dort noch den **Flag unlit** aktivieren, damit die Tafel auch richtig leuchtet.

Damit die Texturen gleich wechseln können, mußt Du die entsprechende Oberfläche mit einer sogenannten **ScriptedTexture** belegen. Diese befinden sich in der Datei **InDus7.utx**. Es sind die Texturen **Monitor1** bis **Monitor7.**

Belege die Tafeln jetzt mit der Textur **Monitor1.** Achtung! Diese Textur darf im Level (bzw. in der Zone) nirgends sonst vorkommen, da **alle** Oberflächen die mit dieser Textur belegt sind sonst ihr Erscheinungsbild ändern.

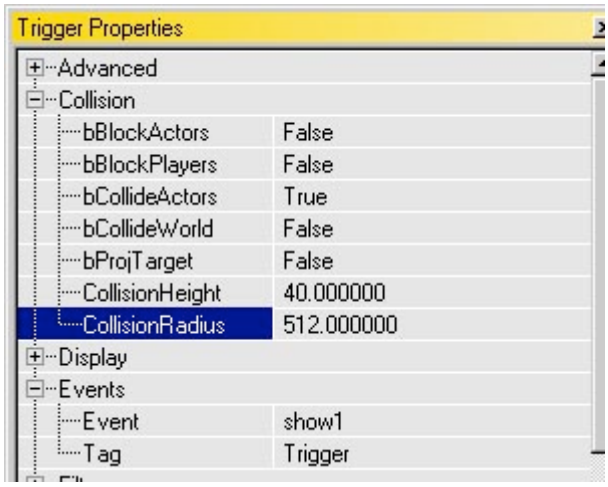


Füge dann aus dem **Actor-Browser** einen Actor unter **Triggers** mit dem Namen **TriggeredTexture** hinzu und setze ihn in die Nähe der Schautafeln.

Stelle für diesen folgende Eigenschaften ein:

- unter **Event** den **Tag** für den Schalter, den Du später hinzufügen mußt.
- aktiviere **bTriggerOnceOnly** wenn die Texturen nur 1x wechseln sollen
- unter **TriggeredTexture** zunächst die **DestinationTexture,** das ist in unserem Fall **Monitor1**. Markiere die Textur im Browser und klicke dann auf den Button Use

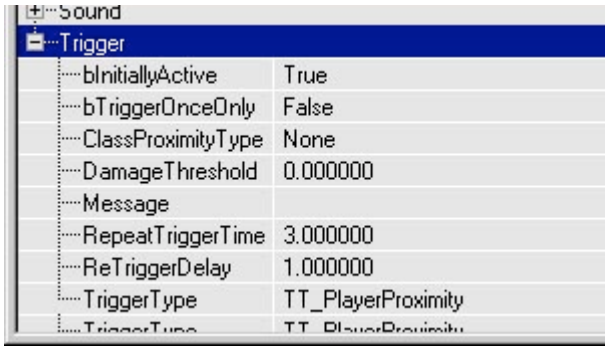
Unter **Textures** kommen die Texturen rein, die wechseln sollen. Mach es genauso wie eben. Die erste Texture [0] ist die Ausgangstextur.



Das war schon fast alles, jetzt brauchst Du nur noch einen Schalter der den Wechsel auslöst. Ich habe einen "normalen" **Trigger** gewählt, der sich auch unter **Triggers** im **Actor-Browser** befindet. Setze ihn vor die Tafel.

Stelle folgende Eigenschaften ein:

- Unter **Collision** den **CollisionRadius** auf 512, damit der Schalter schon ausgelöst wird, wenn man in größerem Abstand vorbei geht.
- Enter **Event** den oben gewählten **Tag** des **TriggeredTexture**



- Unter **Trigger** kannst Du noch die **RepeatTriggerTime** einstellen, das wäre dann die Zeit, die eine der drei Tafeln angezeigt wird, bis der Trigger zum nächsten Bild schaltet.

Ganz wichtig: stelle den **ReTriggerDelay**, also die Zeit wann der Trigger wieder schaltbar wird, auf einen größeren Wert als 0 sonst schaltet der Trigger so schnell hin und her, daß es aussieht, als würde er gar nicht schalten. (Nicht bei uns, weil wir eine RepeatTriggerTime gesetzt haben)

Nach dem **Rebuild** und Starten des Levels, sollte im Spiel die Texture wechseln, sobald Du in den Bereich des Schalters kommst. Ich habe unten zwei verschiedene Schautafeln gebaut. Die DestinationTextures darauf sind unterschiedlich. Es gibt auch zwei TriggeredTexture Objekte und in diesen ist die Reihenfolge der Texturen versetzt, so daß nicht auf beiden Tafeln die gleichen Texturen zur selben Zeit angezeigt werden.

8. Nebel, Spezialeffekte

Wichtig ist für die Konstruktion von Nebeln und speziellen Lichteffekten, daß die Eigenschaft "VolumetricLighting" in der UnrealTournament.ini Datei ermöglicht ist.
Dies kann im Beispiel einer AGTI XPERT 2000 Pro Karte und einem Direct3D Rendering Verfahren so aussehen:

[D3DDrv.D3DRenderDevice]
Translucency=True
VolumetricLighting=True
ShinySurfaces=True
Coronas=True
HighDetailActors=True
UseMipmapping=True
UseTrilinear=False
UseMultitexture=True
UsePageFlipping=True
UsePalettes=True
UseFullscreen=True
UseGammaCorrection=True
DetailTextures=False
Use3dfx=False
UseTripleBuffering=True
UsePrecache=True
Use32BitTextures=False
DescFlags=1
dwDeviceId=20550
dwVendorId=4098
Use32BitZBuffer=False
UseVertexFog=False
UseAGPTextures=False
UseVideoMemoryVB=False
UseVSync=False
Description=XPERT 2000 PRO AGP 4X

Spezial-Effekte
Nicht nur die architektonischen Feinheiten einer Map machen ihre unverwechselbare Optik aus, sondern auch vor allem die perfekte Anwendung der verschiedensten Spezial-Effekte. Dazu zählen unter anderem: Wasser / Feuer / Nebel. Diese Effekte sind relativ einfach zu realisieren. Mit diesem Abschnitt wollen wir Dich in die optischen Highlights des Editors einweihen.

8.1. Erstellen von Nebel-Effekten


Abbildung: 01

Um einen Bereich des Levels mit Nebel zu füllen bedarf es einer vorher erstellten "Zone". Wie du eine solche Zone anlegst, findest du in unserem "Zone/Portal"-Tutorial. Wir gehen hier davon aus, daß du bereits eine funktionsfähige Zone erstellt hast, denn sie markiert die Ausbreitung des Nebel-Effekts innerhalb deines Levels.

Der Effekt selber ist Teil der "ToggleZoneInfo". Um ein solche zu setzen solltest du im "Actor Class Browser" folgende Rubriken öffnen: "Info/ZoneInfo". Dort sollte wie auf unserer Abbildung: 02 sichtbar die Klasse "ToggleZoneInfo" markiert sein.

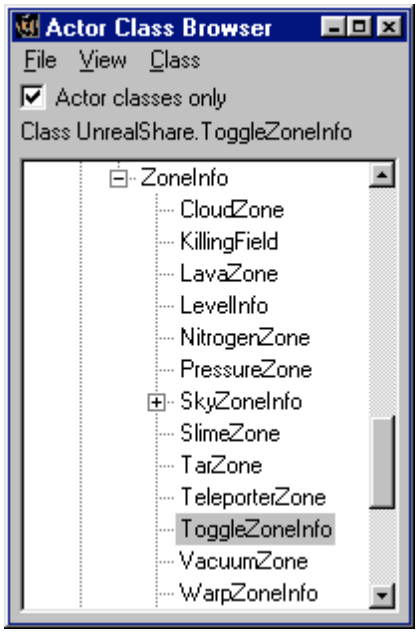


Abbildung: 02

Klicke in der 3D-Ansicht auf die Stelle (muß sich in einer geschlossenen Zone befinden), an der du den Nebel-Effekt erstellen willst. Wähle im Kontextmenü den Eintrag "Add ToggleZoneInfo here". An der betreffenden Stelle sollte jetzt ein Symbol wie in unserer Abbildung: 01 sichtbar sein, dieses mußt du mit der rechten Maustaste anklicken und im Kontextmenü folgenden Eintrag wählen: "ToggleZoneInfo Properties (1 selected)..."

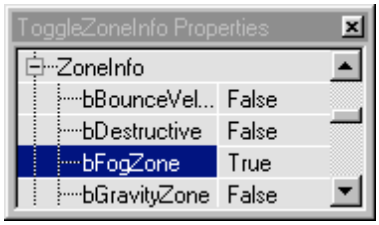


Abbildung: 03

Ändere den Eintrag "bFogZone" wie in unserer Abbildung: 03 ab. Er muß für Nebel auf "True" gesetzt werden, damit die Optionen der weiteren, folgenden Schritten aktiviert werden. Dort kannst du dann die Eigenschaften des Nebels festlegen. Der Nebel ist stets verbunden mit einem Licht. Ohne Lichter gibt es auch keinen Nebel.

Im Nachfolgenden findest Du eine Auflistung der relevanten Parameter:

LightColor	LightBrightness	Dichte des Nebels (0=dunkel / 255=hell)
	LightHue	Farbe des Lichts (respektive des Nebels) (schiebe den Schieberegler ein wenig in beide Richtungen, um die Farbe im Farbkreislauf zu verändern)
	LightSaturation	Farbsättigung des Lichtes (0=volle Farbtiefe / 255=weiß)
Lighting	Volume Brightness	Sichtbarkeit des Nebels (144 bedeutet gute Sichtbarkeit)
	Volume Fog	Dichte des Nebels (0=klare Luft, hohe Werte=neblig) (122 ist ein recht dichter Nebel)
	Volume Radius	Ausbreitungsradius des volumetrischen Nebels (8 ergibt eine relativ kleine Nebelkugel)

Wenn alles geklappt hat, solltest Du jetzt Nebel um die jeweiligen Lampen haben. Du kannst jetzt die Nebelintensität mit VolumeBrightness ändern. Außerdem solltest Du mal die Funktion "VolumeFog" unter die Lupe nehmen.
Wenn Du farbigen Nebel willst, mußt nur wie bei farbigem Licht, die Eigenschaften unter "Lightcolor" einstellen und schon hast Du farbigen Nebel.
Der Effekt Nebel ist im Gegensatz zu normalem Licht der Monster-Ressourcen-Fresser. Geh damit immer schonend und sparsam um. Auch wenn Du einen Athlon 750 hast, deine Freunde oder Leute im Netz haben den eventuell nicht.

8.2. Licht-Effekte

Da oft eine einfache Beleuchtung noch nicht den gewünschten Flair für den Level bietet, kann man einige Veränderungen an der Erstellung von Lichtquellen vornehmen, indem man entweder die "Light"-Klasse mit Features ausstattet oder aber einfache Effekte mit eigenen Polygonkonstruktionen erstellt. Im diesem Kapitel werden wir uns mit der Lichtkegelerstellung und der "Corona/Lens-Flare"-Bildung beschäftigen.

8.2.1. Sichtbaren Lichtkegel erstellen

Setze zuerst eine gewöhnliche Lampe in einen Raum. Diese Lampe sieht bis jetzt zwar schon hell erleuchtet aus und sorgt auch schon für eine Menge Licht in unserem Level, doch bis jetzt kommen noch keine sichtbaren Lichtstrahlen aus der eigentlichen Lampenform. Du solltest deshalb einen Lichtkegel erstellen. Du kannst dies mit Hilfe der Kegel-Bauform erreichen, aber auch einen Würfel benutzen. Prinzipiell kann man aus jeder der nachfolgenden "Primitives" ein volumetrisches Licht erstellen.

Würfel Zylinder Kugel Kegel Abziehen Hinzufügen Spezial

Solid (Punkt setz

Height: 256 Worl

Width: 64 World

Breadth: 64 Wor

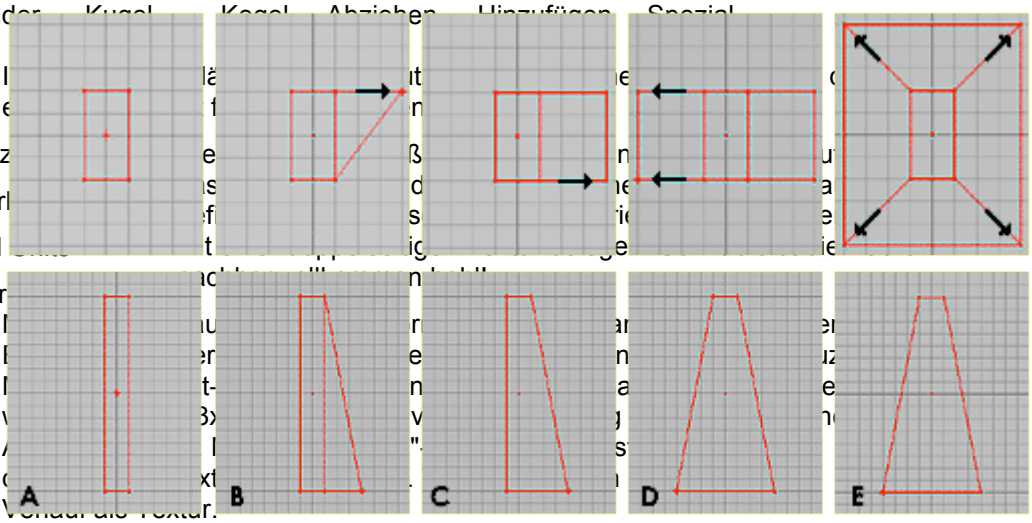
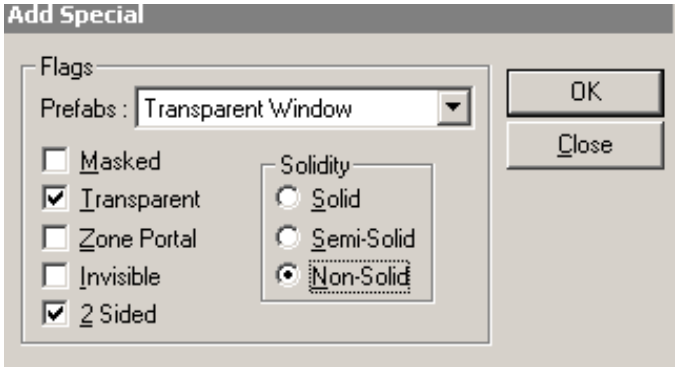


Abbildung: A - B - C - D - E

Setze deine Bauform mit Hilfe des "Special"-Werkzeugs. Beim Klick auf den entsprechenden Menü-Button sollte das "Add Special"-Fenster erscheinen (vgl. Abbildung rechts). Hier solltest Du bei Prefabs auf "Transparent Window" stellen. Du mußt nun bei "Transparent", "2-Sided" und "Non-Solid" die jeweiligen Eigenschaften einschalten.

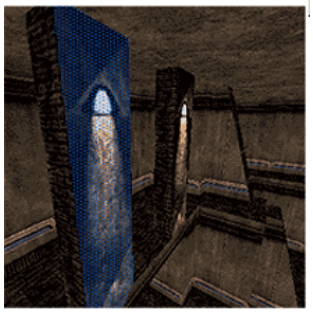


Wenn du die Textur jetzt doppelseitig hinzufügen willst, solltest du eine Textur im Textur-Browser selektiert haben. Nachdem du unseren Lichtkegel hinzugefügt hast, kann es vereinzelt vorkommen, daß er nicht angezeigt wird. Dies wird sich aber ändern, wenn du den Level einmal gerendert hast (F8 Taste drücken). Als letztes solltest du nochmal alle Texturen des Lichtkegels selektieren und "Unlit" anklicken, oder aber "SpecialLit" (muß dann separat mit einer "bSpecialLit"-Lichquelle beleuchtet werden).

So, jetzt ist der Lichtkegel fertig! Deine Lampe mit fertiger Beleuchtung sollte ungefähr so aussehen, wie sie unsere Abbildung (links) zeigt. Der Spieler wird im Spiel in der Lage sein, durch den Lichtkegel der Lampe zu laufen, ohne an ihren Texturen hängenzubleiben. Das ist auch der Grund, warum wir eine doppelseitige Textur verwendet haben, um den Lichtkegel zu erstellen. Wenn der Spieler später genau in der Mitte des Lichtkegels steht, sieht er die entsprechenden Texturen von der Innenseite und dort sehen sie genauso aus, wie auf der Außenseite. Es wirkt so, als ob der Schein des Lichtes aus der Lampe heraus scheint und nur einen bestimmten Bereich des Levels ausleuchtet. Als letztes Feintuning empfiehlt sich eine Anpassung des sichtbaren Lichtbereiches auf dem Boden/Wand, denn nichts wäre schlimmer als ein Lichtkegel, der den entsprechend unter ihm befindlichen Bereich des Bodens/Wand nicht hell miterleuchten würde.

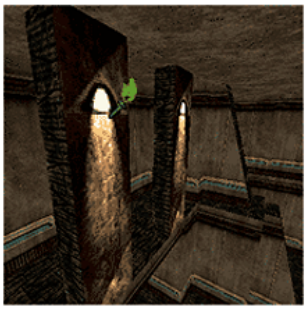
8.2.2. "LenseFlare" oder Corona Blendenflecke

In vielen Leveln hast du bestimmt schon die sog. "Lens Flares" oder "Coronas" gesehen, einen optischen Lichteffekt, der eine Lampe oder Leuchteinheit umgibt wie eine Aura. Diesen werden wir jetzt erstellen. Dazu solltest du bereits über einen Testlevel ferfügen, in dem du jetzt nur noch diesen Effekt einbauen mußt.



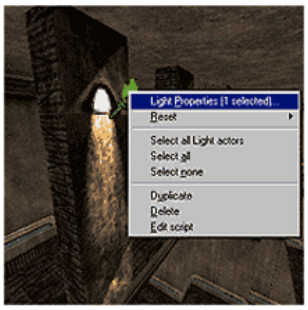
1.) Unser Testlevel besteht aus einem etwas höheren Raum. Wir haben der Einfachheit halber bereits eine Textur gewählt, die eine beleuchtete Lampe darstellt, damit wir nicht noch umständlich eine besondere Form für unsere eigentliche Lampe bauen mußten. Trotzdem kannst du natürlich diesen Effekt auf jede erdenkliche Lampenform in deinem Level adaptieren.

Dieser Effekt ist nicht an die Geometrie deiner Umwelt oder Levels gebunden. Er hängt nur an der beutzten Klasse "Light/TriggerLight".



2.) Als nächstes haben wir eine Klasse "Light" gesetzt und diese bereits an den Platz geschoben, an dem sie später einmal sein soll. Du kannst Deine Beleuchtung wunschgemäß einrichten.

Du kannst also die Farbe des Lichtes und dessen andere Eigenschften frei wählen, jedoch sollten zusätzlich einige Einstellungen vorgenommen werden. Gehe dazu wie folgt vor:



3.) Klicke mit der rechten Maustaste auf die Klasse "Light" und dann im Kontextmenü auf "Light Properties (1 selected)..." Im bereits bekannten Eigenschaftenfenster mußt du jetzt folgende Unterrubriken öffnen:

Display

LightColor

Lighting

Dort sollte man folgende Werte ändern:

Variable	Eintrag	Änderung
Display, DrawScale	1.0	Skalierung des Effekts, z. B. 0.4 (Die Defaultgröße ist nicht brauchbar.)
Lighting, bCorona	false	true
und bLensFlare		true
Display, Multiskins, Skin	...	Textur des Corona-Effekts, z.B. Texture'city.CityFlare'

Lighting	Style	verschiedene alternative Maskierungsmodi (bezieht sich auf Skin)

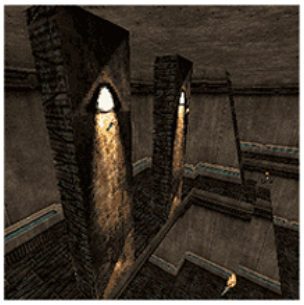
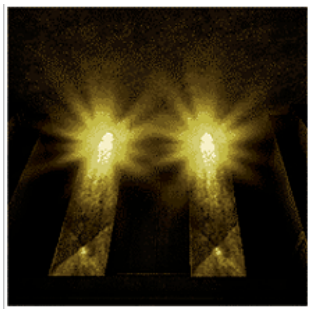


Abbildung: 06

4.) In unserem Beispiel haben wir den Wert bei "DrawScale" auf 0.400000 gesetzt, dabei mußt du beachten, daß sich die Größe der Fackel proportional mitändert (siehe rechts). Bei Skin haben wir dann noch einen Eintrag unserer Textur "GenFX/Flare6" hinterlassen. Die Farbe des Lichtes haben wir entsprechend der "Corona"-Textur etwas gelblich eingefärbt.



5.) Im Editor wirst du vorerst mal keine Veränderung feststellen können, aber wenn du jetzt das Spiel über "Strg+P" testest, wirst du an der Stelle des neu erstellten Lichteffekts, genau selbigen auch sehen können. In unserem Beispiel sieht es aus, als ob unsere beiden Lampen (wir haben einfach noch einen zweiten Effekt daneben erstellt) stark strahlen würden. Natürlich kannst du diesen Effekt auch noch an vielen anderen Stellen in deinem Level einsetzen. So zum Beispiel auch als Sonne, oder Sterne. Jedoch solltest du nicht übertreiben, denn er muß immer mitberechnet werden. Das kostet Rechenleistung.

9. Der 2D Shape Editor

9.1. Wie man komplexe Bauformen erstellt

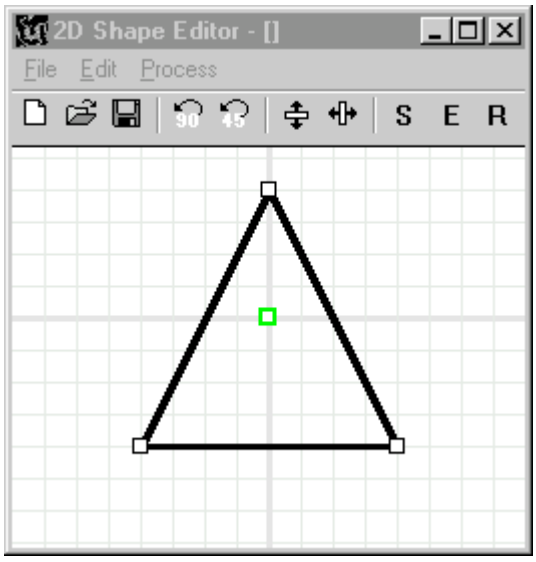
Benutze den 2D-Editor. um zum Beispiel ein gebogenes Rohr, einen unterirdischen Kanal, einen spitzen Torbogen oder ähnliches zu bauen. Es geht zwar auch auf anderen Wegen, doch dies ist die eleganteste: Mit Hilfe dieses Tools kannst Du auf einfache Weise in einer zweidimensionalen Ansicht einen Bauformquerschnitt erstellen und diesen dann per passender Eingabe zu einem 3D-Modell erweitern. Hierbei stehen zur Verfügung:

Sheet: Um eine verwinkelte Fläche mit einer passenden Ebene abzudecken

Extrude: Um einen Torbogen oder Gang zu realisieren

Revolve: Um ein gebogenes Rohr, oder ein gebogenes Objekt zu erzeugen

Um den 2D-Editor zu starten, muß zuerst der **UnrealED 2.0** gestartet werden. Dort kannst Du im Haupt-Menü folgenden Eintrag finden**: "Tools/2D Shape Editor**".



Du kannst auch diesen Button aus der oberen UnrealED Button-Leiste benutzen:

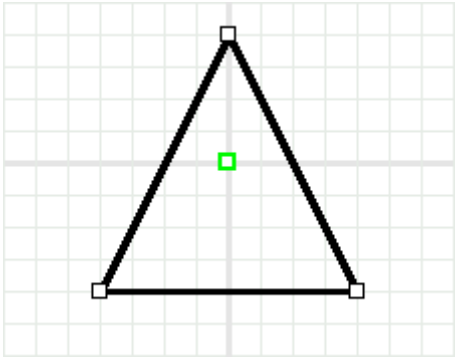
Öffnet den 2D-Editor (Version 420 - 428)

2D Shape Editor - (ab Version 430)

Es sollte sich danach das "2D Shape Editor"-Fenster öffnen, vergleiche dazu die nebenstehende Abbildung. Sie zeigt den 2D-Editor in dem Zustand, wie er erscheint, wenn man ihn zum ersten Mal aufruft. Man sieht den Mittelpunkt (grün) und eine aus drei Seiten bestehenden, einfachen Bauform im Querschnitt.

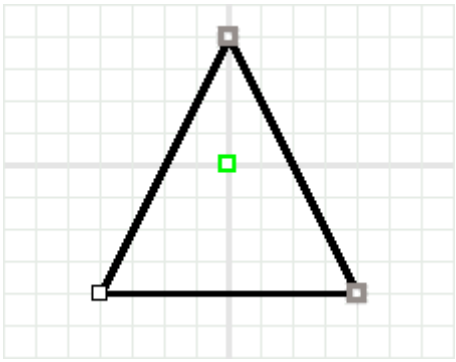
Dieses Tutorial wurde auf Basis eines UnrealED 2.0 Version 425 erstellt. In neueren Versionen des Editors hat sich die Optik, Stabilität und Handhabung geringfügig geändert, jedoch sind die einzelnen Handgriffe immer noch die selben, auch wenn unsere Bilder noch recht klobig wirken.

5.2 2D-Brush anfertigen



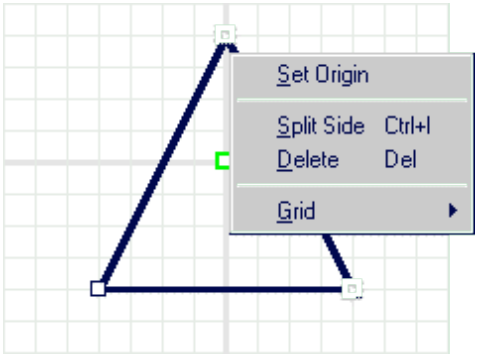
1. Nach dem ersten Start des 2D Shape Editors sieht man eine ganz einfache 2D-Bauform: ein Dreieck mit einem grünen Viereck als Mittelpunkt.

Alle Punkte (ob Eck- oder Mittelpunkt) kannst Du mit einem einfachen Mausklick selektieren (der Punkt wird dann rot). Um mehrere Punkte gleichzeitig zu selektieren benutze die RM-Taste in Kombination mit der STRG-Taste.



2. Um einen Eckpunkt an beliebiger Stelle einzufügen, bedarf es etwas Vorbereitung. Die gewünschte Seite, an der Du den neuen Eckpunkt einfügen willst, muß zuerst selektiert werden. Nutze hier: **"STRG" und "LM-Taste".**

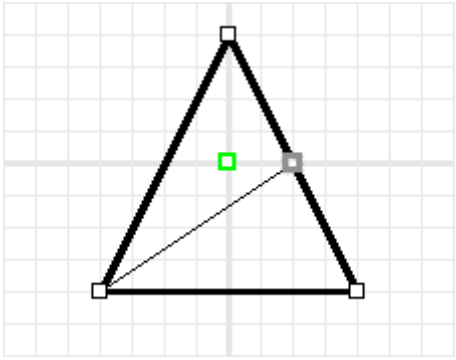
In unserem Beispiel haben wir den oberen Punkt und den Punkt unten-rechts selektiert (leider im Bild grau, sonst im Editor rot!)



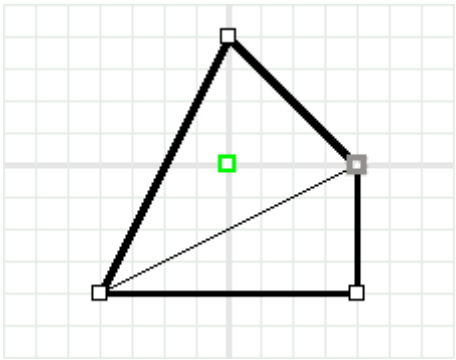
3. Den weiteren Eckpunkt, der in unsere Bauform einfügt werden soll, erstellen wir wie folgt:

Klicke dazu mit der "**RM-Taste**" irgendwo auf eine Seite der Form und wähle im Kontextmenü den Eintrag:

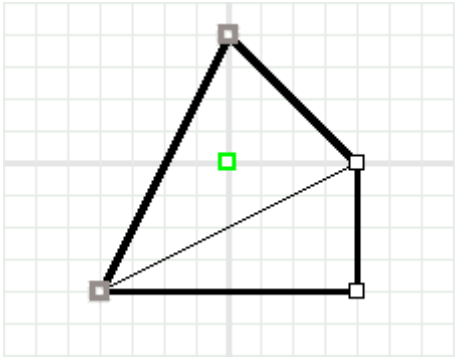
"**Split Side**" oder "**CTRL+I**"



4. Ein neuer Eckpunkt ist entstanden, er liegt irgendwo (meistens in der Mitte) auf der Linie, die wir benutzt haben um ihn per Kontextmenü zu erzeugen

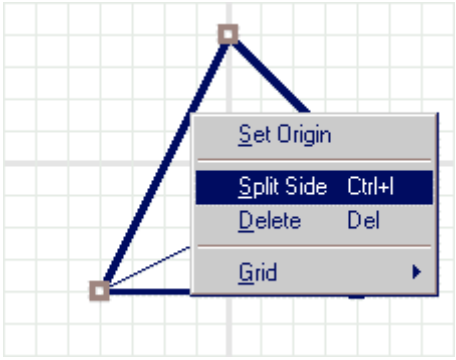


5. Jetzt schieben wir (wie oben erklärt) den Eckpunkt an die Stelle, wo wir ihn später haben wollen. In unserem Beispiel haben wir den neuen Eckpunkt etwas an den rechten Rand verschoben. Wir wollen uns einen Torbogen bauen.



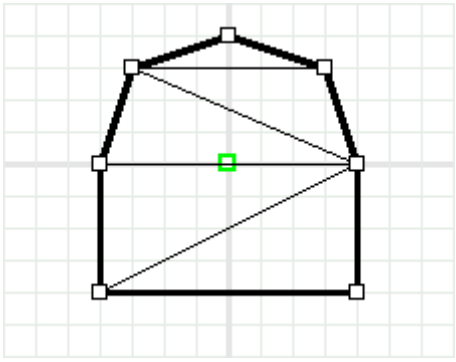
6. Bei Bedarf kannst Du auch die anderen Eckpunkte deines Gebildes verschieben.

Um weitere Eckpunkte, an beliebigen Seiten einzufügen, kannst Du den oberen Vorgang wiederholen, bis genügend Eckpunkte vorhanden sind. In unserem Beispiel haben wir die linke Seite des Gebildes selektiert, um dort einen weiteren Eckpunkt einzufügen.



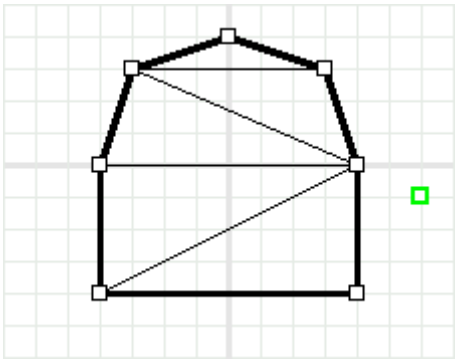
7. Das Erstellen des Eckpunktes geht wieder über die **RM-Taste** und das Kontextmenü:

"Split Side" oder "CTRL+I"



8. Wir haben den oberen Vorgang solange wiederholt, bis eine 2D-Bauform entstanden ist, wie Du sie nebenstehend erkennen kannst.

Jetzt ist deutlich, was wir erstellt haben, eine halbrunde Form, mit einer Bodenfläche. Die Form ist ähnlich einem gotischen Torbogen, es entsteht eine gerundete Decke. Wir könnten es als Gang einsetzen.



9. Damit wir später die Möglichkeit haben, diese Form auch als gebogene Röhre zu verwenden, muß **der Mittelpunkt verschoben** werden. Um diesen Mittelpunkt kann später diese Form gebogen werden. Deshalb muß der Mittelpunkt außerhalb des Objektes liegen.

In unserem Beispiel haben wir ihn um 32 Welteinheiten nach rechts verschoben.

9.2. Umwandlung von 2D- in 3D-Bauformen

In diesem Kapitel erklären wir die Umwandlung von 2D- in 3D-Bauformen. Das **sogenannte Shape-to-Brush** Verfahren erlaubt es, die im 2D-Editor erstellten Flächen als 3D-Formen im UnrealED einzusetzen. Dabei stehen folgende grundlegende Optionen zur Verfügung:

 "**Create a Sheet**" - Ebene erstellen (z.B. Wasser)

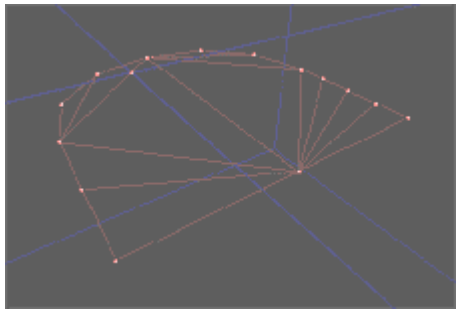
 "**Create a Revolved Shape**" - runde Form erstellen (z.B. Säulen)

 "**Create an Extruded Shape**" - extruierte Form erstellen (z.B. Fenster)

"**Extrude to Point**" - Form zu einem Punkt extrudieren (z.B. Pyramiden)

"**Extrude to Bevel**" - Form zu einer Schräge extrudieren (z.B. Podeste/Rampen)

Über die 2D-Menü Funktion "**Process**" kannst Du deine im "2D Shape Editor" erstellte zweidimensionale Form zu einer dreidimensionalen Form ausbauen:

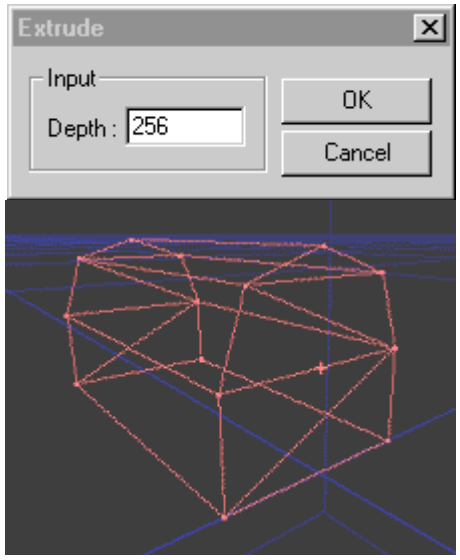


Create a Sheet:

Entweder über diesen Button: 

oder aber ...

Über die Menüfunktion "**Process/Create a Sheet**" erstellst Du eine einfache Ebene (zum Gebrauch für Wasserflächen / ZonePortals u.v.m.). Ein Klick auf den Button reicht aus, es gibt hierbei keine weiteren Optionen.



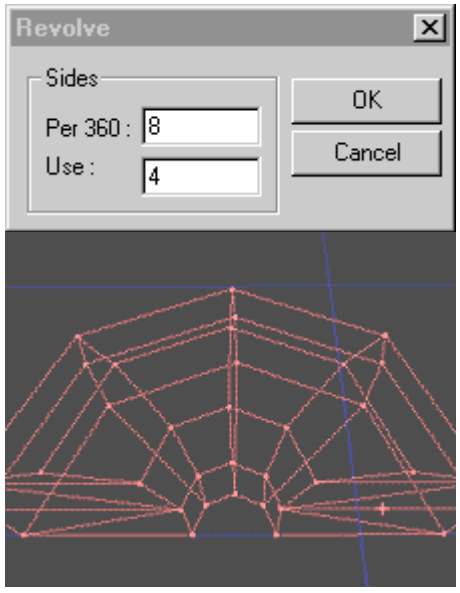
Create an Extruded Shape:

Entweder über diesen Button: 

oder aber ...

Über die Menüfunktion "**Process/Extrude**" kannst Du eine Tiefe für deine 2D-Bauform festlegen. Wenn diese Funktion angewählt wird, erscheint nebenstehendes Fenster mit der einfachen Option die Tiefe einzutragen.

Es sollte dann eine Brush entstehen, wie in unserer Abbildung links.



Create a Revolved Shape:

Entweder über diesen Button: 

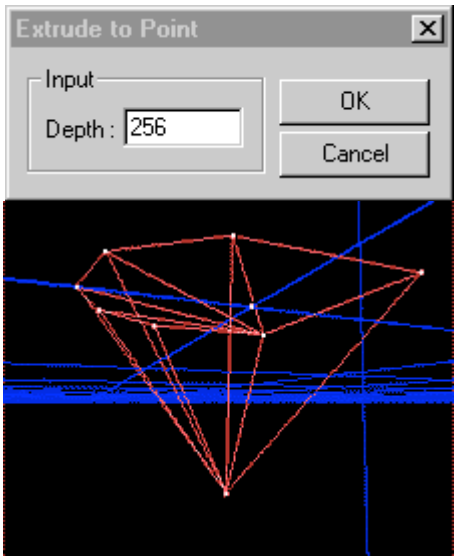
oder aber ...

Über die Menüfunktion "**Process/Revolve**" kannst Du eine Drehung der 2D-Bauform um die eigene Achse festlegen. Wenn Du diese Funktion auswählst, wird nebenstehendes Fenster angezeigt. Dabei gilt folgende Vorgehensweise

Per 360: Gesamtanzahl der Segmente bei 360° Drehung

Use: Wieviele Segmente davon benutzt werden sollen

In unserem Beispiel haben wir bei voller Drehung auf 8 Segmente eingestellt, 4 verwenden wir für die 3D-Form. So entsteht ein Halbkreis, wie ihn unsere nebenstehende Abbildung zeigt. Damit kannst Du z.B. ein gekrümmtes Gangsystem bauen.



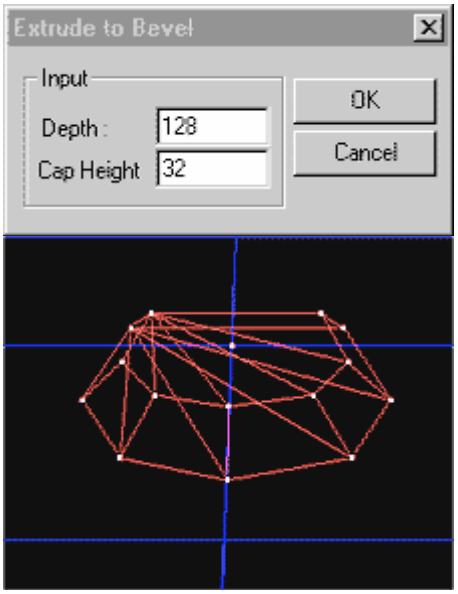
Extrude to Point:

Entweder über diesen Button: 

oder aber ...

Über die Menüfunktion "**Process/Extrude to Point**" kannst Du deine Form in einen Fluchtpunkt auslaufen lassen.

Der Fluchtpunkt liegt immer auf dem grünen Mittelpunkt, den Du versetzen kannst, um ein anderes Ergebnis zu erreichen.



Extrude to Bevel:

Entweder über diesen Button: 

oder aber ...

Über die Menüfunktion "**Process/Extrude to Bevel**" kannst Du die Kanten abschrägen.

Depth: Versatz zur Grundform

Cap Height: Höhe des Gebildes

9.3. Wie man einen Torbogen erstellt

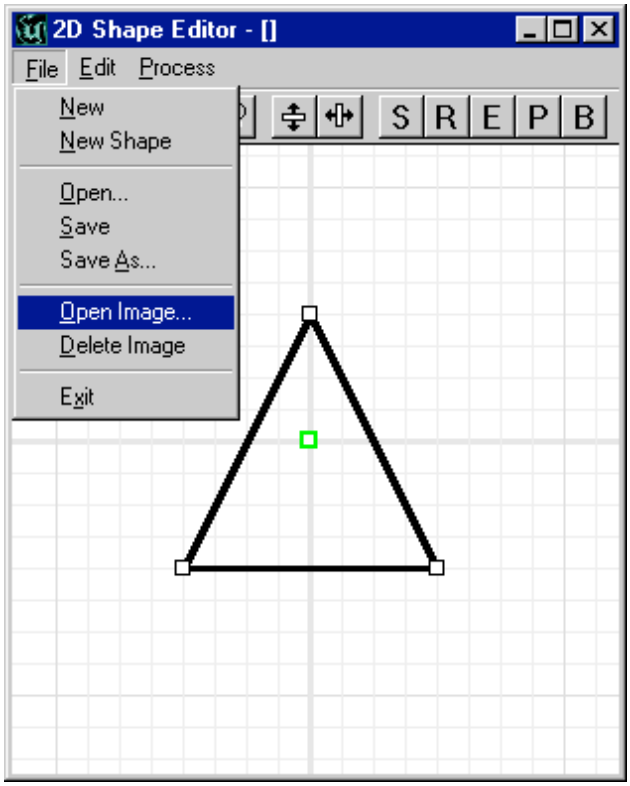
Ziel der Übung ist es einen perfekt gestalteten Torbogen zu erstellen, der genau auf eine Torbogen-Textur (in unserem Beispiel: ShaneChurch/BArchesL) abgestimmt ist. Die Form soll genau in die schwarze Fläche der Textur passen. Mit Hilfe einer solchen Form ist es möglich die Wand genau auszuschneiden. Um es uns so einfach wie möglich zu machen, nutzen wir den 2D-Editor.

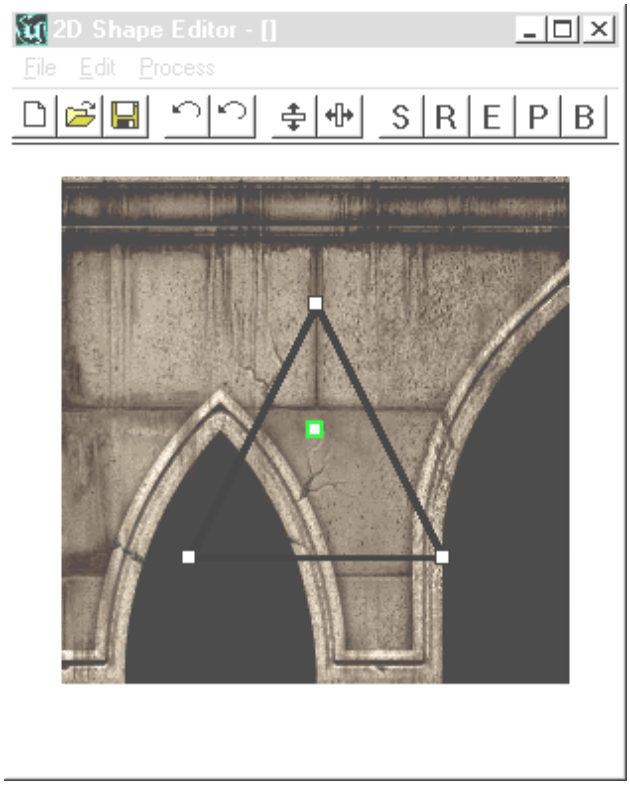
1. 2D-Editor starten

(alter Button)


(neuer Button)

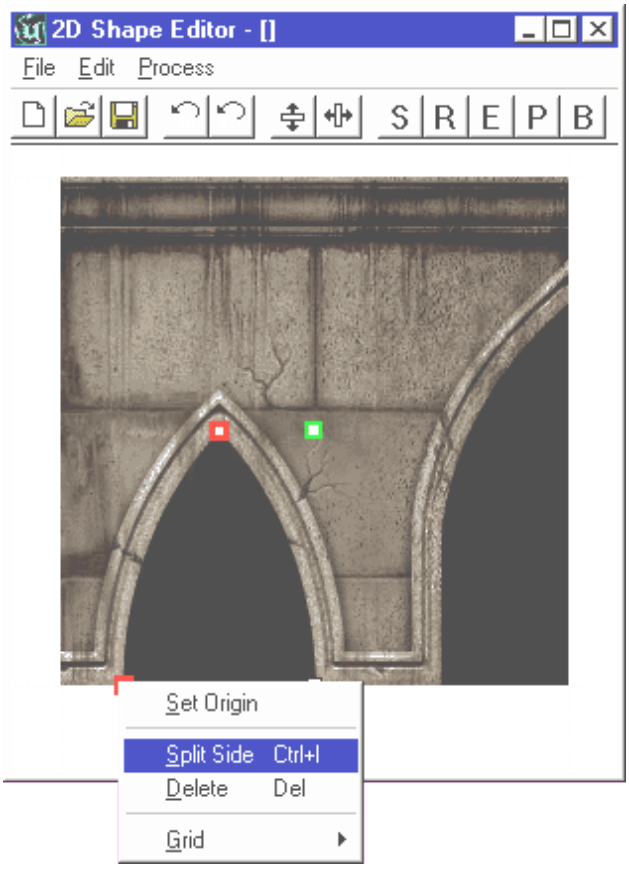
Danach sollte sich das "2D Shape Editor"-Fenster öffnen, vergleiche dazu die nebenstehende Abbildung. Sie zeigt den 2D-Editor in dem Zustand, wie er erscheint, wenn man ihn zum ersten Mal aufruft. Man sieht den Mittelpunkt (grün) und eine aus drei Seiten bestehende, einfache 2D-Form.
Wähle, wie auf unserer Abbildung, den Menüpunkt "**File/Open Image**" an.





2. Dort kannst Du dann eine eigene Bilddatei laden. In unserem Beispiel haben wir die Textur: ShaneChurch/BArchesL genutzt. Um sie zu importieren, mußten wir sie erst aus der Texturpalette exportieren und abspeichern (als .bmp-Datei). Danach kann man die Texturen in den 2D-Editor importieren, wie in unserer Abbildung ersichtlich.

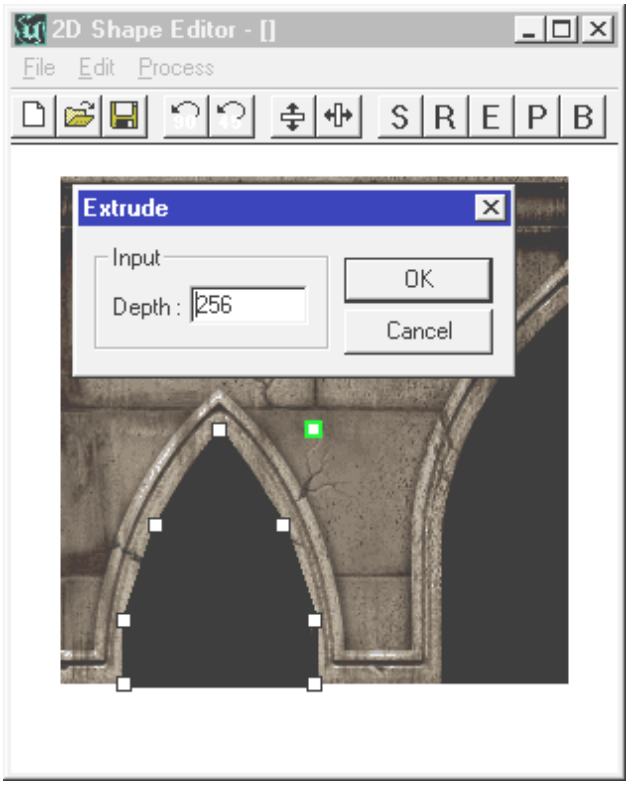
Das Bild liegt im Hintergrund. Weiterhin gut sichtbar und vor allem weiterhin nutzbar bleibt die 2D-Form.
Uns bleibt es vorbehalten, jetzt nur noch die Anzahl und Position der Eckpunkte zu bestimmen, um die spätere 3D-Form genau zu gestalten.



3. Wenn Dir das vorgegebene 16er-Raster zu grob ist, kannst Du dies jetzt noch manuell abändern. Nutze dazu, **die Option "Grid/1,2,4,8,16,32,64**". Achtung: Je feiner das Raster, und je mehr Eckpunkte Du verteilst, desto aufwendiger wird die Berechnung der späteren Form (Performanceverluste drohen in der späteren Map.)

Zuerst müssen wir die Eckpunkte in etwa an die Form annähern (so gut es eben geht). In unserem nebenstehenden Beispiel war es recht einfach, die 3 Eckpunkte zu verteilen. Jedoch reicht natürlich die Anzahl nicht aus, um die Außenlinie genau einzurichten.

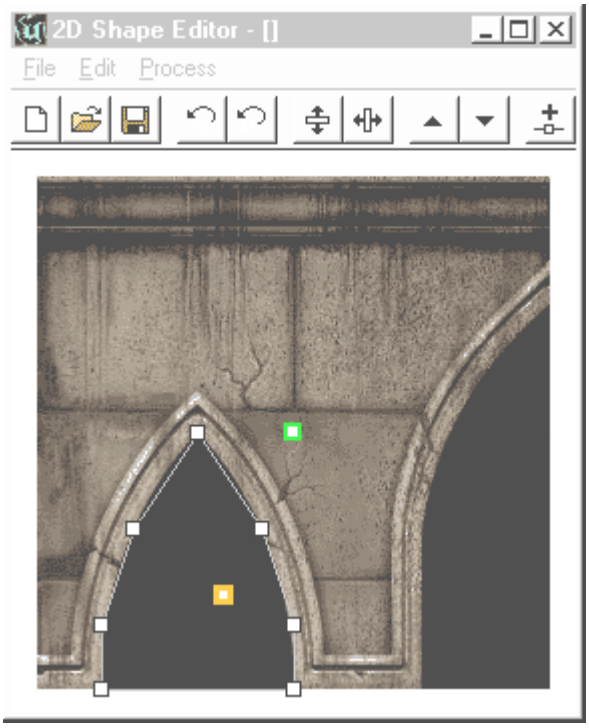
Dazu müssen erst zwei nebeneinander liegende Punkte (rot) markiert werden, dann kann man mit einem Klick auf einen markierten Eckpunkt oder die entsprechende Linie mit der **RM-Taste** - durch "**Split Side**" oder "**CTRL+L**" einen neuen Eckpunkt erzeugen.



4. Genau in der Mitte (der beiden markierten Eckpunkte) sollte jetzt ein neuer Eckpunkt entstehen, der nur noch an eine passende Stelle verschoben werden muß. In unserem Beispiel haben wir vier weitere Eckpunkte eingefügt und an die passende Stelle geschoben. **Vermeide dabei unbedingt das Auftreten von roten Verbindungslinien, diese produzieren in der späteren 3D-Bauform einen schweren Grafikbug**. An dieser Stelle passen die Texturen nicht mehr auf die Form!

Eigentlich sind wir fast fertig. Um die Wandstärke zu bestimmen, klicke einfach auf den **"E"-Button** in der 2D-Editor Menüleiste und gib bei "**Depth**" die Einheiten ein (z.B. 64 Welteinheiten). Nachdem Du auf den "OK"-Button geklickt hast, steht die passende Bauform im UnrealED zur Verfügung. Mit Hilfe der normalen Bauarten (Hinzufügen/Entfernen) kannst Du diese Bauform jetzt an der passenden Stelle im Level einbauen.

Neuere Versionen vom 2D Editor



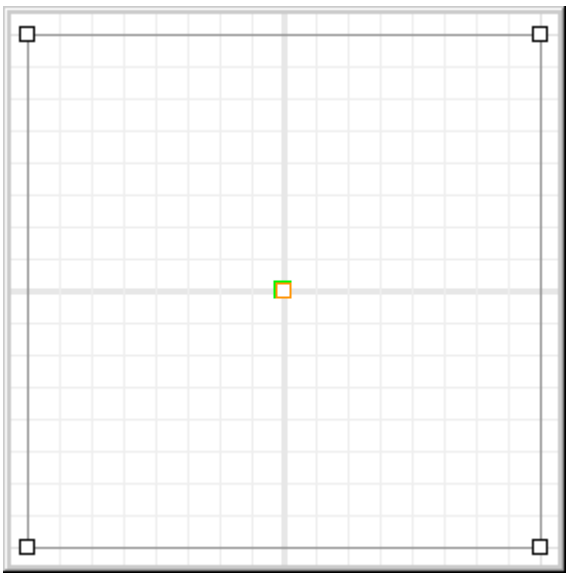
1. Da gerade im 2D-Editor-Bereich immer wieder Verbesserungen einfließen, kann eine neuere Version des 2D-Shape-Editor durchaus etwas anders aussehen, als bei unseren oberen Abbildungen, die auf der Version 425 basieren.

In Versionen über 430, gibt es einige Änderungen: Die Vertex-Bearbeitung ist wesentlich vereinfacht worden, im Menü haben sich die grafischen Buttons geändert.

Generell aber ändert sich an der allgemeinen Benutzung nichts, wenn auch weitere Optionen hinzugekommen sind!

9.4. Bezier-Segmente

Wie man abgerundete 2D-Shapes erstellt:
Bezier-Segmente sind virtuelle Hilfssegmente (später nicht sichtbar). Mit Hilfe dieser Segmente kann man eine Gerade beliebig krümmen, ohne umständlich weitere Zwischenpunkte in den Shape (die 2D-Form) setzen zu müssen. (Diese Technik fand erstmals im ID-Titel Quake3 Arena Verwendung, hat sich mittlerweile aber auch in UT (EPIC) eingefunden!



1.  öffnet den 2D-Editor (neuer Button)

Nebenstehendes Fenster sollte sich öffnen, oder ein zumindest vergleichbares Fenster.

Wir haben die Menüleiste aus Platzgründen in unseren Bildern eingespart und nutzen **UnrealED 2.0 in Version 432** für dieses Tutorial, da die hier beschriebenen Features erst ab Unreal Version 430 implementiert wurden!

2. Um es uns einfacher zu machen laden wir eine Textur in den 2D-Shape Editor (siehe Torbogen-Tutorial), die uns als Schablone, für unsere Form die wir erzeugen wollen, dient.

Als erstes solltest Du die maximale Fläche ausfüllen in unserem Beispiel ist das immerhin über 60% der gesamten Fensterfläche, die wir ausschneiden wollen.

Zuletzt müssen nur noch die Eckpunkte markiert werden. Danach wollen wir die noch geraden Verbindungslinien krümmen.

... auf den entsprechenden Button im 2D-Shape Editor Menü klicken:

 ein Bezier-Segment wird hinzu gefügt.

Wie in unserer nebenstehenden Abbildung ersichtlich, erscheinen an den entsprechend markierten Punkten seltsame blaue Hilfslinien, mit deren Hilfe man jetzt der dicken, grauen original Verbindungslinien krümmen kann.

In unserem Beispiel wollen wir sie nach oben hin, dem Fensterrahmen angepaßt, krümmen. Dazu geht man dann wie folgt vor:

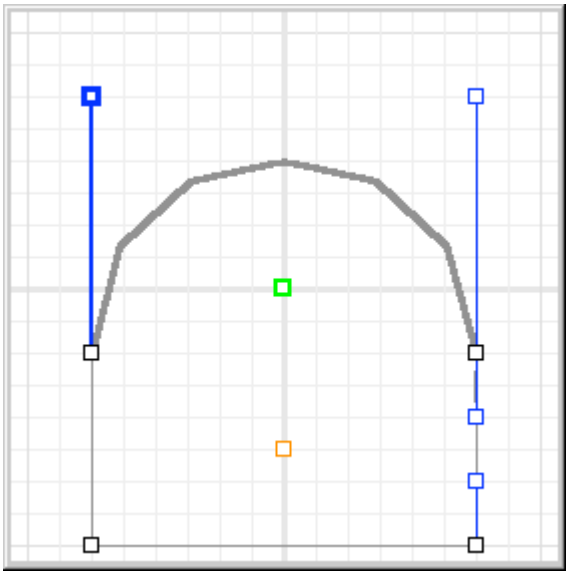
4. Wie man auch die herkömmlichen Eckpunkte verschieben kann, so kann man dies auch mit den **neuen blauen Hilfspunkten** machen. Diese einfach mit der **LM-Taste** anklicken und mit gedrückter Taste in die gewünschte Richtung verschieben.

Wir nutzen jeweils den innen liegenden Punkt und verschieben ihn 90° nach oben, etwa doppelt so weit entfernt vom Gelenkpunkt (fester Eckpunkt), wie früher, da wir einen Halbkreis formen wollen.
Achtung: Erst wenn auch der zweite Punkt verschoben ist, kannst Du den genauen Endpunkt des Bezier-Segmentes festlegen, da sich noch etwas ändern kann, wenn auch er bewegt wird. Das wollen wir dann auch machen:

5. Genau spiegelverkehrt, bewegen wir auch den zweiten, innen liegenden Bezier-Endpunkt 90° nach oben! Um eine gleichmäßige Rundung zu erzeugen, setzen wir ihn auf gleicher Höhe ab. So und das war's!

Der sogenannte 2D-Shape ist erstellt.

Um bei großen Shapes und verwirrenden Textur-Vorlagen nicht den Kopf zu verlieren, kannst Du zur Überprüfung der 2D-Form einfach sicherheitshalber die Textur zum Schluß ausblenden und nochmals Schwarz auf weiß dein neues Meisterwerk betrachten:



6. Die in unseren Bildern noch sichtbaren blauen Bezier-Hilfslinien werden beim Exportieren der Bauform in den Unreal Haupt-Editor verschwinden. Übrig bleibt dann eine simple 2D/3D-Form, die wir in unserem Beispiel nutzen, um ein Fenster aus der Wand zu schneiden, oder einen Mauervorsprung zu simulieren.
Je mehr Eckpunkte zwischen den Bezier-Segmenten angebracht werden, um so besser kann man eine Fläche runden, oder Rund erscheinen lassen. Gerade in Verbindung mit einem RevolvedShape lassen sich großartige Effekte erzeugen.

10. Player, Bots und Skins

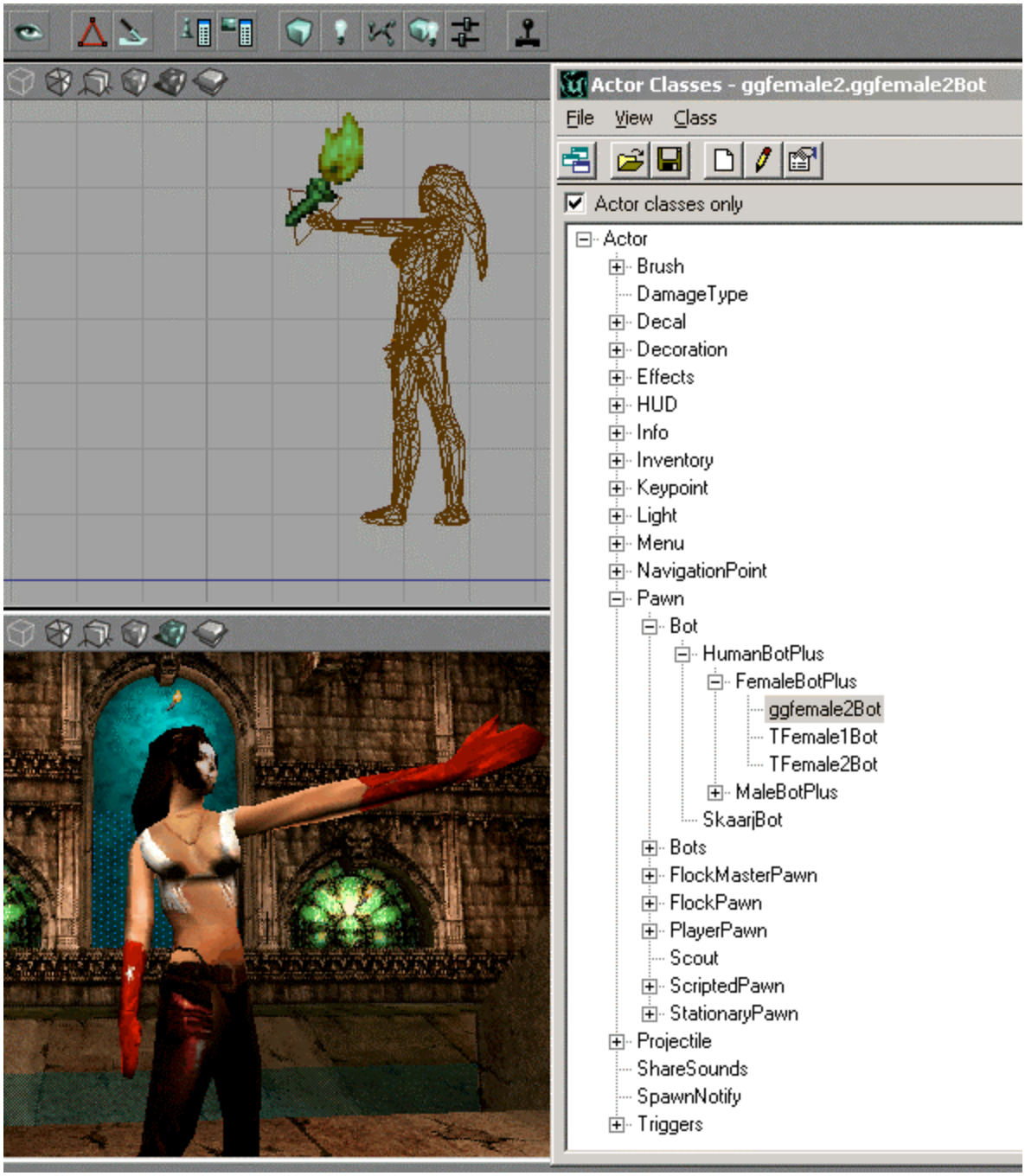
Die Player, die UnrealTournament als Spielerfiguren anbietet, sind stereotype amerikanische Heldenfiguren mit gepolsterten Schulter und dummem Gesichtsausdruck. Viele Spieleentwickler fragen sich, wie man diese Figuren durch individuelle Gestalten ersetzen kann. Die Frage lautet: Wie erstelle ich neue Player Figuren?
Falls man nicht ganz von vorne anfangen will, gibt es zwei Methoden. Beide greifen auf die 3D Modelle zurück, die Gratuiousgames als natürliche Figuren mit der Bezeichnung GGfemale und GGmale am Netz anbietet (http://www.gratuitousgames.com/UTmodel1.htm):

1. Wenn ich alle Texturen, .uc Scripts und die _d.3d und _a.3d Files habe, so kann ich die Texturen in Photoshop editieren, ggf. das Verhalten der Figuren in einem Texteditor modifizieren oder die Geometrien modifizieren und danach alles mittels „ucc make“ rekompilieren.

Für die GGfemale Figuren sehe ich jetzt (wenn GGfemale2skin2.int nicht im System Folder liegt) eine einzige neue Figur im Player Setup des Tournament Players. Zusätzliche Charaktere oder Gesichter sind nicht selektierbar.

Wenn man die GGfemale Figuren im MeshBrowser oder im Texturenbrowser sehen können, oder sie aus dem Actor Browser heraus als Bots einsetzen können möchte, so muß man im File UnrealTournament.ini die Eintragung
EditPackages=ggfemale2
tätigen.

Nachdem man dies getan hat, findet man die Figuren in ggfemale2bot, einer Unterklasse von Bot/ HumanBotPlus/ FemalBotPlus



2 .Falls Du jedoch nicht an die Scripts, Textures oder 3D-Geometrien herankommst, (von manchen Klassen geben die Programmierer/ Designer keine Geometrien und Textures her, und liefern nur das kompilierte Package aus) so sind Deine Bearbeitungsmöglichkeiten etwas eingeschränkt. Du kannst dann im Texturenbrowser das .utx Package mittels „open“ öffnen, danach die zu modifizierenen Texturen mittels „export“ als pcx Files exportieren, im Photoshop editieren, und schließlich wieder in den Texturen Browser mittels „import“ importieren. Das modifizierte Texturenpackage muß ich schließlich mittels „save“ wieder abspeichern an den Ort, wo es herkam.
Im System Folder von Tournament muß das File GGfemale2skin2.int die einzelen Players der GGfemale Familie, deren Typen und Gesichter verzeichnen:

[public]
Object=(Name=GGFemale2Skin2.jenn1,Class=Texture,Description="LoveGoddess")
Object=(Name=GGFemale2Skin2.jenn1jenn,Class=Texture,Description="Love")
Object=(Name=GGFemale2Skin2.jenn2,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn2t_0,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn2t_1,Class=Texture)

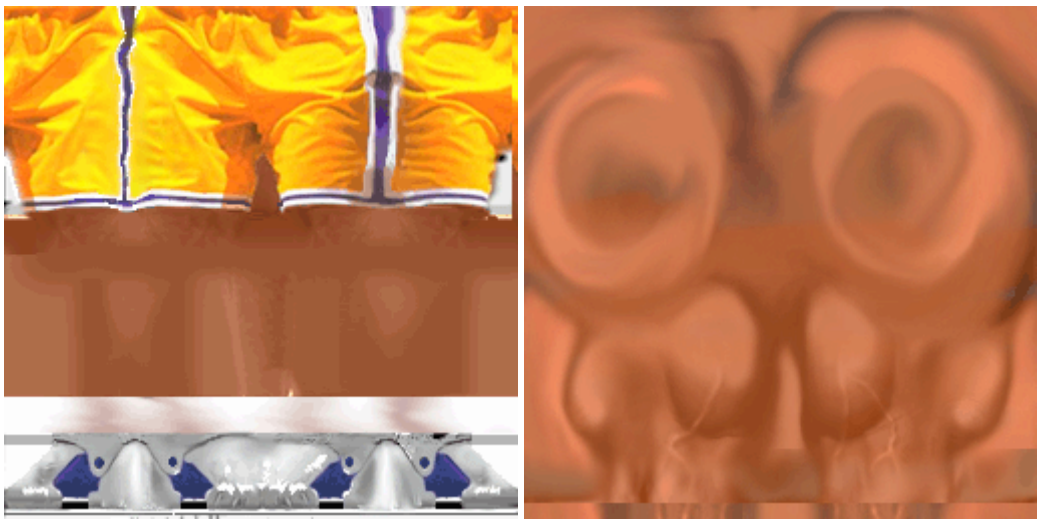
Object=(Name=GGFemale2Skin2.jenn2t_2,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn2t_3,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn3,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn3t_0,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn3t_1,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn3t_2,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn3t_3,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn4,Class=Texture)
Object=(Name=GGFemale2Skin2.jenn5jenn,Class=Texture)

Eine Figur, auf die die Texturen aufgebracht sind, sieht beispielsweise wie der links abgebildete Basketballspieler aus.

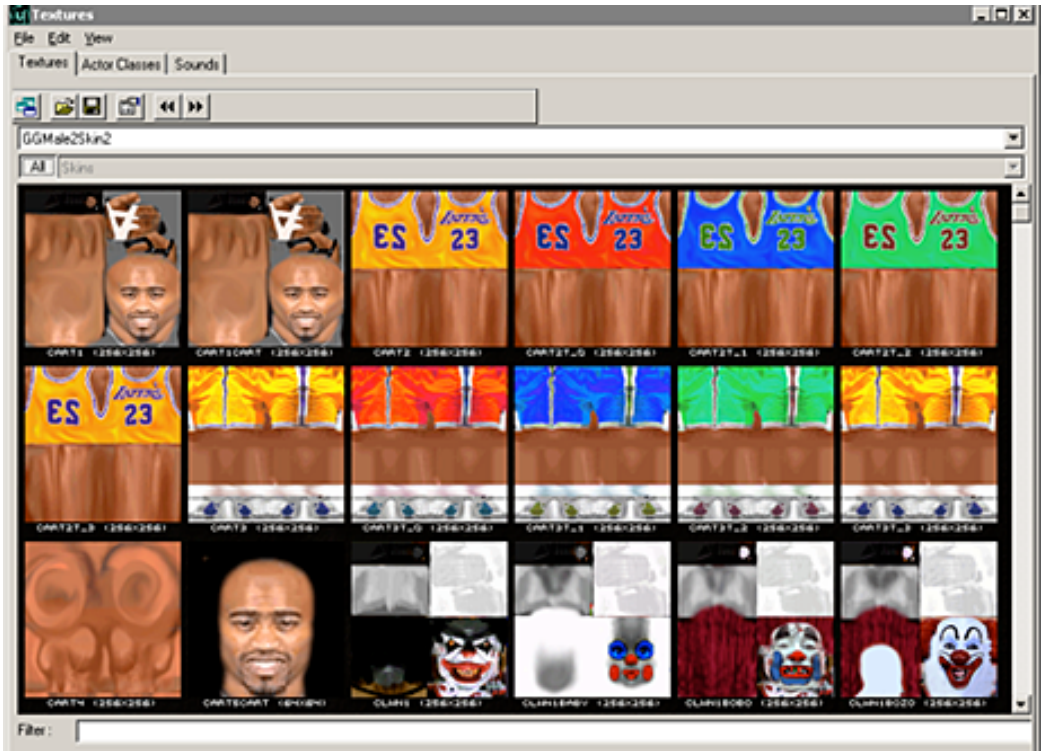
Alle Player und Bots sind in 4 Textur Bitmaps zerlegt, die auf das 3D Model gewrapped werden. Der Mapping Mechanismus ist ein UV-Texture Mapping. Dabei werden Bitmaps der Größe 256 x 256 verwendet. Diese Bitmaps enthalten die Texturen für den Kopf, Brust und Rücken, Unterleib und Arme. Im Falle des Basketball Players sehen die "Skins" wie nachfolgend gezeigt aus:

Kopf (Gesicht und Hinterkopf unten links),
Hände, Brust und Rücken (unten rechts):

Unterleib und Arme:



Will man diese Skins durch einfaches Übermalen oder eine andere Manipulation in Photoshop verändern, so muß man achtgeben, welche Bitmaps man exportiert und wieder in UNNREAL zurückholt. Im Texture Browser siehst Du in nachfolgender Abbildung zwei mal ein 256 x 256 Pixel großes Bitmap des Gesichts, dann 5 Oberkörper mit T-Shirt. Danach 5 Hosen mit Oberschenkel usw. Für eine Modifikation des Gesichtes der Player Figur muß das zweite Image modifiziert werden. Das erst (Texturenname cart1) hat keine Wirkung auf die Erscheinung des Players. Das zweite (Texturenname cart1cart) wirkt sich im Spiel aus. Exportiere dieses Image als .pcx File, manipuliere es in Photoshop oder einem ähnlichen Programm und reimportiere das abgespeicherte Bild. (Das reimportierte Bild muß Namensgleich mit dem ursprünglichen sein. Ggf. "rename" anwenden)



Die 5 Ober- und Unterkörper entsprechen den Teams "None", "Red", "Blue", "Green", "Gold".

Nach der Veränderung der Skin des Basketballspielers könnte beispielsweise fogende Transformation erfolgt sein:

Bei Beibehaltung der Geometrie  wurde nur die Textur geändert.

Wenn man nun auch noch das Verhalten der Figuren verändern will, so muß man die Skripts, die das Verhalten der Player steuern und bestimmten Situationen entsprechend Animationssequenzen zuordnen, modifizieren. Entweder man tut dies im Script-Browser des UnrealEditors und exportiert und rekompiliert die veränderten Skripts dann, oder man editiert ein Skript wie ggmale2.uc bzw. ggfemale2.uc in einem beliebigen Texteditor, löscht dann die korrespondierende .u Datei und rekompiliert mittels „ucc make“.

Nachfolgend wurden die Bewegungen CockGun, CockGunL und andere durch die Animationssequenz Victory1 ersetzt. Achtung! Die Animationssequenzen müssen unter den alten Namen CockGun, CockGunL usw. im Skript bleiiben. Lediglich die zugeordeten Startframes und Numframes also der erste Kader und die Anzahl der Kader in der Animation wurden substituiert.

```
//=============================================================================
// ggfemale2 - silly movements replaced / Mathias Fuchs, 2001
// The animation sequences StilFrRp, StillLgFr, StillSmFr
// have been replaced with to Breath1
// The animation sequences CockGun, CockGunL, Taunt1 and Thrust
// have been replaced with Victory1
//=============================================================================
class ggfemale2 extends TournamentFemale;

#exec MESH IMPORT MESH=ggfemale2 ANIVFILE=MODELS\ggfemale2_a.3d
DATAFILE=MODELS\ggfemale2_d.3d X=0 Y=0 Z=0
#exec MESH ORIGIN MESH=ggfemale2 X=0 Y=0 Z=-70

#exec MESH SEQUENCE MESH=ggfemale2 SEQ=All                STARTFRAME=115 NUMFRAMES=1
        Group=TakeHit
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=AimDnLg            STARTFRAME=0 NUMFRAMES=1
        Group=Waiting
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=AimDnSm            STARTFRAME=0 NUMFRAMES=1
        Group=Waiting
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=AimUpLg            STARTFRAME=1 NUMFRAMES=1
        Group=Waiting
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=AimUpSm            STARTFRAME=1 NUMFRAMES=1
        Group=Waiting
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=Breath1            STARTFRAME=3 NUMFRAMES=20
        Group=Waiting
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=Breath2            STARTFRAME=3 NUMFRAMES=20
        Group=Waiting

// The animation sequences Taunt1 and Thrust
// have been changed to Victory1
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=Taunt1 STARTFRAME=301 NUMFRAMES=38 RATE=60
        Group=Gesture
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=Victory1 STARTFRAME=301 NUMFRAMES=38
RATE=40Group=Gesture
```

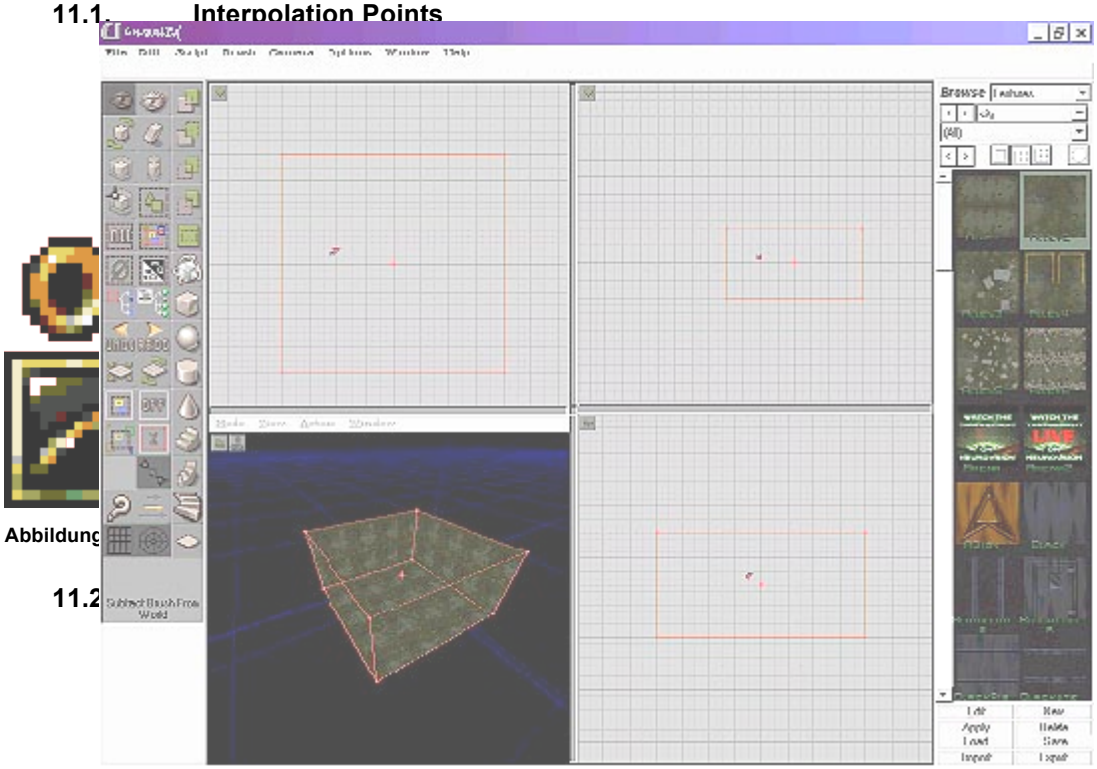
```
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=Thrust  STARTFRAME=301 NUMFRAMES=38 RATE=30
        Group=Gesture
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=Wave   STARTFRAME=354 NUMFRAMES=40 RATE=30
        Group=Gesture

// The animation sequences CockGun, CockGunL, StilFrRp, StillLgFr, StillSmFr
// have been changed to Breath1
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=CockGun          STARTFRAME=301 NUMFRAMES=3
        Group=Gesture
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=CockGunL         STARTFRAME=301 NUMFRAMES=38
        Group=Gesture
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=StillFrRp        STARTFRAME=3 NUMFRAMES=20
        Group=Waiting
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=StillLgFr        STARTFRAME=3 NUMFRAMES=20
        Group=Waiting
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=StillSmFr        STARTFRAME=3 NUMFRAMES=20
        Group=Waiting

#exec MESH SEQUENCE MESH=ggfemale2 SEQ=DuckWlkL         STARTFRAME=90 NUMFRAMES=15
        Group=Ducking
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=DuckWlkS         STARTFRAME=90 NUMFRAMES=15
        Group=Ducking
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=HeadHit          STARTFRAME=116 NUMFRAMES=1
        Group=TakeHit
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=JumpLgFr         STARTFRAME=117 NUMFRAMES=1
        Group=Jumping
#exec MESH SEQUENCE MESH=ggfemale2 SEQ=JumpSmFr         STARTFRAME=117 NUMFRAMES=1
        Group=Jumping
```

11. Erstellen von Movies / Intros

11.1. Interpolation Points



01). Man

Abbildung

11.2

Abbildung: 02

und setzen da jetzt noch einen Block mit den Maßen 512x512x512 mit einer Textur aus City.utx hinein.

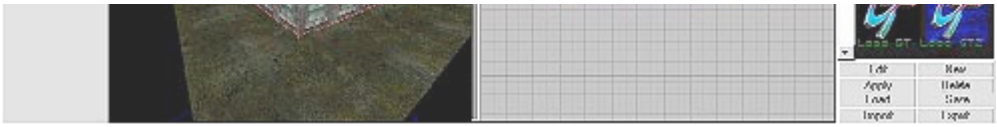


Abbildung: 03

Jetzt haben wir ein Haus, um das wir gleich fliegen werden. Einen Playerstart und Lichter setzten, mit **F8** Rebuilden und den Level testen (kein Problem, wenn mehrere Bots mitspielen, im Laufe des Tutorials verschwinden die!!!). Jetzt kommt ein wichtiger Teil des Arbeitens mit InterpolationPoints: Das richtige setzen aller kommenden Actors! Wir wählen nun den InterpolationPoint Actor (s.o.) aus, gehen im ED auf halbe Höhe des Levels und setzen pro Wand 2 Actors (nicht an die Häuserwände, sondern an den gegenüber liegenden Wänden). GANZ WICHTIG: Bei einem InterpolationPoint noch einen zusätzlichen daneben setzen – (wird später erklärt).

Nun wählen wir einen InterpolationPoint aus und öffnen mit einem Doppelklick auf die LM-Taste die Eigenschaften des InterpolationPoints. Wir durchstöbern nun ein bißchen die verschiedenen Einträge. Im Moment ist nur der Eintrag "InterpolationPoints" wichtig. Einmal auf das „+“ klicken um das Unterverzeichnis zu öffnen.
Das Aktivieren der InterpolationPoints. Wir haben jetzt alle 9 InterpolationPoints gesetzt, müssen sie aber noch dazu bringen, den Spieler auf diesem Weg um das Haus fliegen zu lassen. Wir wählen jetzt den rechten der 2 InterpolationPoints aus, die Du zu Anfang nebeneinander setzen solltest und gehst mit einem **Doppelklick auf die LM-Taste** auf **InterpolationPoint/ Position**. Du gehst jetzt immer einen InterpolationPoint weiter nach links und schreibst jedesmal eine Zahl unter **Position**, immer mit 1 addiert ([Anfang] 0; 1; 2....7; 8 [ENDE]). Jetzt ist jedem InterpolationPoint eine Zahl zugeordnet. Markiere jetzt alle InterpolationPoints (RM-Taste / Select all InterpolationPoint actors) und trage unter **Tag** einen Xbeliebigen Namen ein, z.B. Hausflug.

Jetzt kommt zum Schluß des 2.Teils noch der Befehl, die InterpolationPoint's zu benutzen. Geh unter **Classes/ Triggers/ SpecialEvent** und "adde" mit Klick auf RM-Taste ein **SpecialEvent** in den Level (Position ist egal) und stelle folgendes ein: **Tag: interpolate, Event=Hausflug** (damit steuert das SpecialEvent alle InterpolationPoints an) und unter dem **Verzeichnis**: **Object/ InitialState PlayerPath** (damit das SpecialEvent weiß, daß es die InterpolationPoints benutzen).

Abbildung: 04

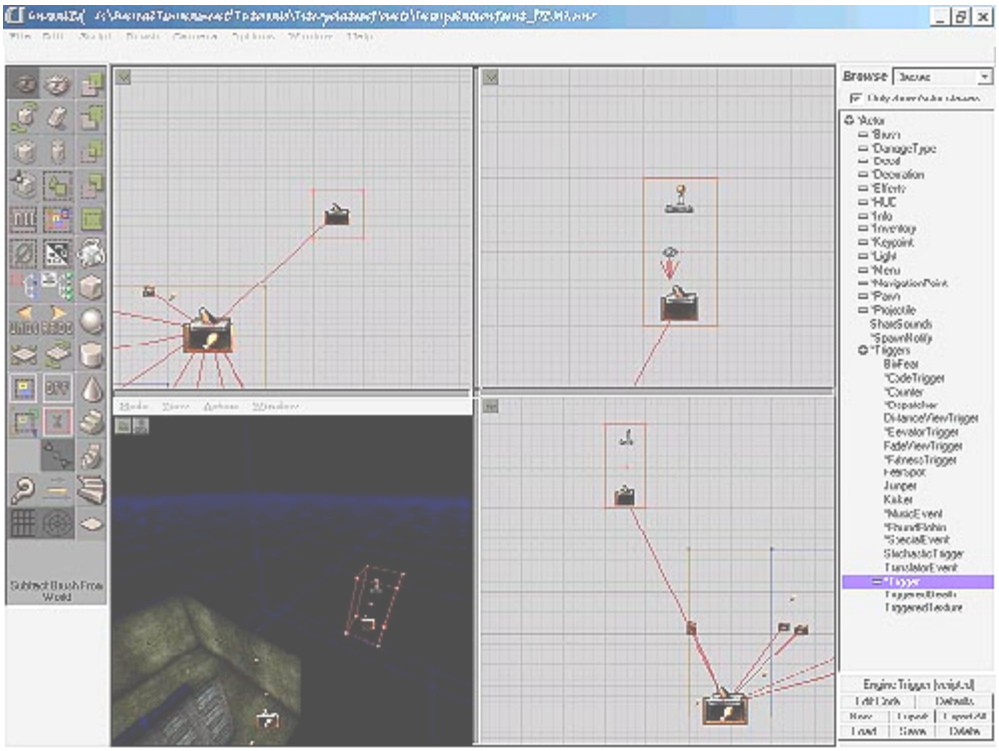


Abbildung: 05

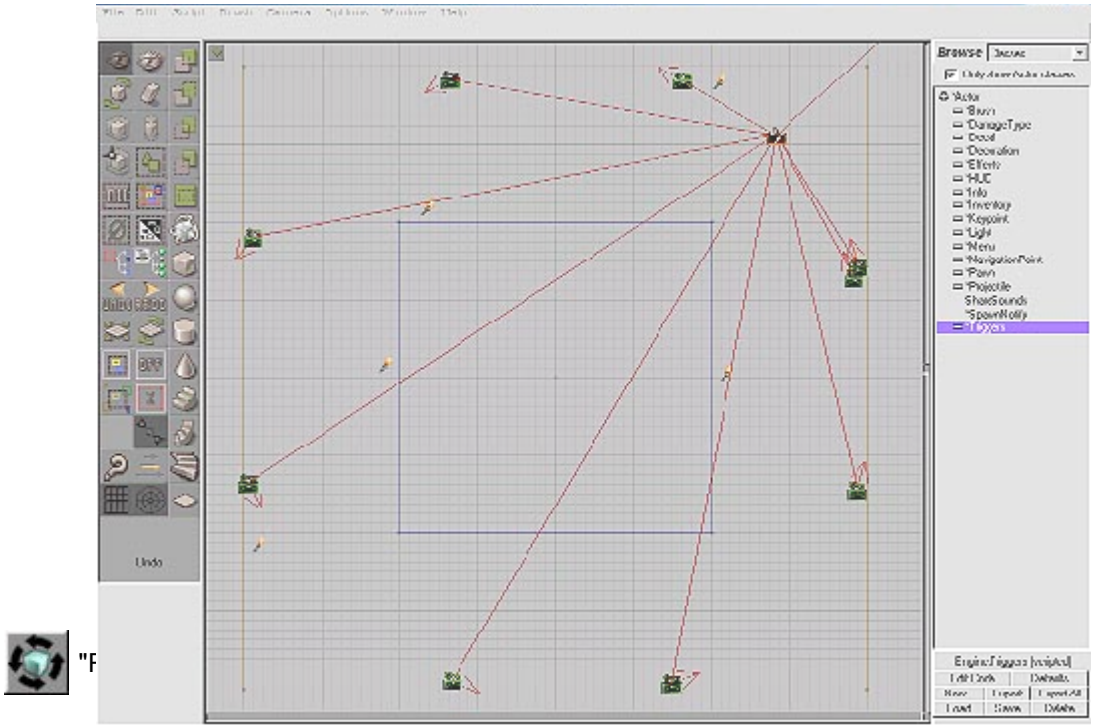


Abbildung: 06

Jetzt nur noch Rebuilden, abspeichern und dann testen!!! Wenn alles so gemacht wurde wie beschrieben, müßtest Du jetzt um das Haus fliegen.

Ein Problem bleibt immer noch. Man fliegt nicht direkt jeden InterpolationPoint ab, sondern nähert sich ihm nur soweit, wie es der nachfolgende InterpolationPoint erlaubt. Wenn Du den Level testest, wirst Du das merken.
Wenn Du den Level so gebaut hast, wie beschrieben, wird Dir auffallen, daß der Flug an einer Stelle plötzlich ruckt. Der Fehler ist, daß der Flug DIREKT vom 1. InterpolationPoint aus startet, Du aber am letzten InterpolationPoint nur annähernd vorbei fliegst. Achte deshalb bei Rundflügen darauf, daß der 1. und der 2. InterpolationPoint Direkt übereinander liegen, dann kommt eine flüssige Bewegung dabei raus.

11.3. Weitere Anwendungsbereiche

- Wenn man z.B. einen Level hat und sofort von einem InterpolationPoint zum nächsten, gibt man beim vorhergehenden InterpolationPoint unter **RateModifier** eine 0 ein – oder versucht: **bSkipNextPart=True**.

- Wenn Du ein **Zeit/Dimensionenportal** baust, kannst Du auf die **FovModifier Funktion** zurückgreifen. Je kleiner der Wert, desto weiter zoomt das Bild, bis man den nächsten InterpolationPoint erreicht. Ist der Wert größer als 1, so geschieht das Gegenteil.
- Eine besondere Eigenschaft der InterpolationPoints ist, daß man mit ihnen **auch Trigger auslösen** kann. Probiert mal folgendes aus: Baut eine lange Röhre und baut innen drin dann die InterpolationPoints ein und auf halber Strecke einen Trigger, der einen Sound abgibt, wenn er berührt wird. Vervollständige den Level und lasse Dich dann durch den Trigger fliegen. Du hörst im Flug auf einmal ein Geräusch!
- Mit der Option **GameSpeedModifier** kannst Du Gewehrhülsen langsamer zu Boden fallen lassen oder gescriptete Bots eine Funktion in **Zeitlupe** ausführen lassen. Mit einer ähnlichen Technik wurde sogar der Film MATRIX hergestellt. Man gibt einem Bot, mit Hilfe eines Triggers, hochzuspringen. Wenn der InterpolationPoint den Trigger erreicht, stelle den Wert des GameSpeedModifiers der nachfolgenden InterpolationPoints so klein ein, daß der letzte InterpolationPoint gerade die Szene erreicht, wo der Bot oben angekommen ist. Man läßt alle InterpolationPoint eine Halbkurve machen, so entsteht ein genialer Effekt (wer den Film gesehen hat, weiß wovon er redet!). Muß man halt etwas trainieren und gut einschätzen wie lange der Bot braucht, um vom Absprung bis zum höchstmöglichen Punkt braucht.
- Die Sache mit den Teleportern: Es gibt, wie bei den Triggern, die Möglichkeit, einen vordefinierten Flug durch einen **Teleporter** zu unterbrechen. Ein schönes Beispiel ist das UT Intro. Dort fliegt man am Ende auf die beschriftete Wand zu, und wird dann durch eine Rechtskurve auf einen Teleporter gelenkt. Der lädt dann den Level und die Musik.

12. Sounds, Voices, Music

Es gibt in Unreal verschiedene Möglichkeiten, Klangmaterial einzusetzen: Standardmäßig sind dies die Formate SoundEffects und Music, die einfach gesagt .wav Dateien und .mod Dateien oder .iit Dateien darstellen. Als .wav Dateien kann man derzeit nur Mono-Soundfiles verwenden, diese allerdings mit 44100 Hz Samplingrate und 16 Bit Wortbreite. Die einzelnen Klänge sind in Gruppen zusammengefaßt, die als Packages im .uax Format gespeichert sind.

Package	**Specific**
Activates	Beeps, Clicks and Machinery
AmbAncient	Monk, Lava, Fire, Glass, Explosion, Ghost and Creak
AmbModern	Alarm, Fan, Electric, Hum, Machine, Pipes, Teleport and Scream
AmbOutside	Animals, Waterfall, Swamp, Waves, Lightning, Quake and Thunder
Announcer	Announcer Voice
BossVoice	Boss Voice
BotPack	Weapon and Item Sound
DoorsAnc	Thumps, Chain, Stone and Wood door sounds
DoorsMod	Modern Door and Lift sounds
Female1Voice	Female 1 Voice
Female2Voice	Female 2 Voice
FemaleSounds	Female sound effects
Male1Voice	Male 1 Voice
Male2Voice	Male 2 Voice
MaleSounds	Male sound effects
Pan1	Helecopter, Train and some Voice
Rain	Rain and Thunder
Vrikers	Voice sounds

"Music" im Sinne von UNREAL sind .mod oder .it Files, die man mittels Trackerprogrammen herstellen kann. Dies sind Musikformate, die Samples und Samplestrukturen enthalten. Diese Formate sind bei Spieledesignern populär, in der musikalischen Welt dagegen nicht sehr gebräuchlich. Es gibt jedoch verschieden Konvertierungsutilities, die helfen, gängige Formate wie .mp3 in .umx umzuwandeln.

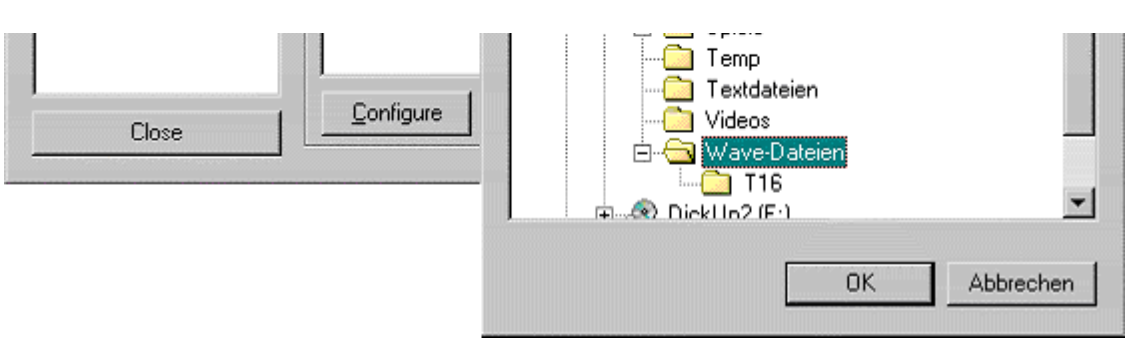
12.1. MP3 für Unreal (.mp3 Files in .umx Files umwandeln)

MP3-Format zu UMX-Formater: Wie konvertiert man das?
Viele Gamedesigner fragen sich, ob und wie es geht, eine MP3-Datei so zu konvertieren, daß man sie in UT als Levelmusik einsetzen kann - nun, hier ist die Anwort in 4 Schritten. Schnappt euch euer Lieblingslied und auf geht's! Dieses Tutorial behandelt nicht den UnrealED, sondern Dritt-Anbieter-Programme, die euch hier erklärt werden:

Hier die Programme, die Ihr für den Vorgang benötigen werdet:
- Winamp (Freeware http://www.winampcom)
- CoolEdit Pro (Shareware) oder SoundForge http://www.syntrillium.com
- ModPlug Player (Freeware http://www.modplug.com/)
- ModPlug Tracker (Freeware wie oben)

1.Schritt - Erzeugen einer .WAV-Datei

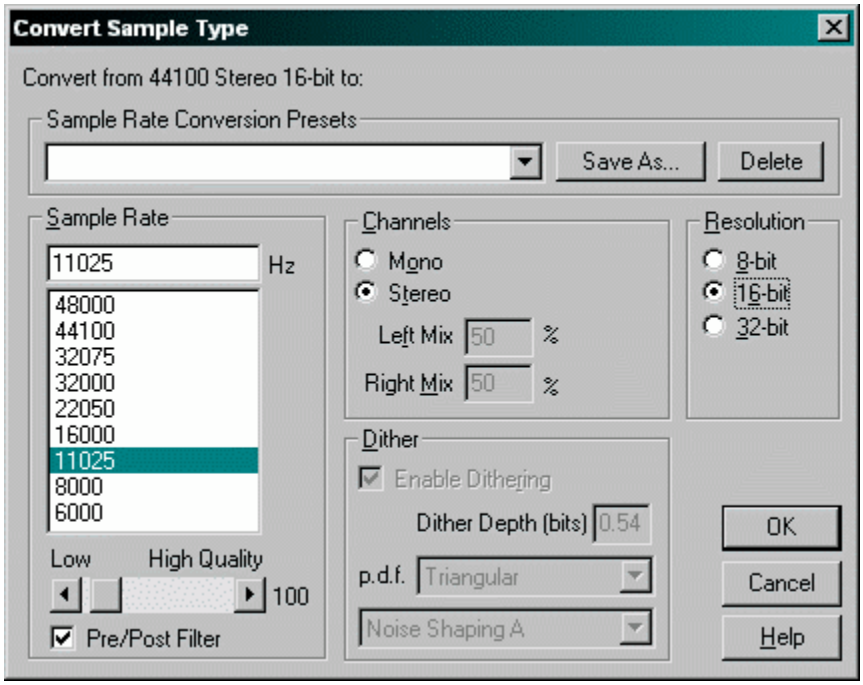
Zuallererst muß man die MP3-Datei in eine Wave-Datei exportieren. Dazu könnt Ihr allerlei Programme benutzen, die es unterstützen, aber ich persönlich benutze



Wenn Alles fertig konfiguriert ist, drückt Ihr nur noch Play und wartet, bis die Datei zu Ende "gespielt" wurde (Achtet darauf, daß in Winamp die "Repeat"-Funktion abgeschaltet ist.) Ihr findet die Wave-Datei dann in dem Verzeichnis, das ihr angegeben habt.
Schließt jetzt Winamp - es wird nicht mehr benötigt.

2. Schritt - Samplerate-Umwandlung

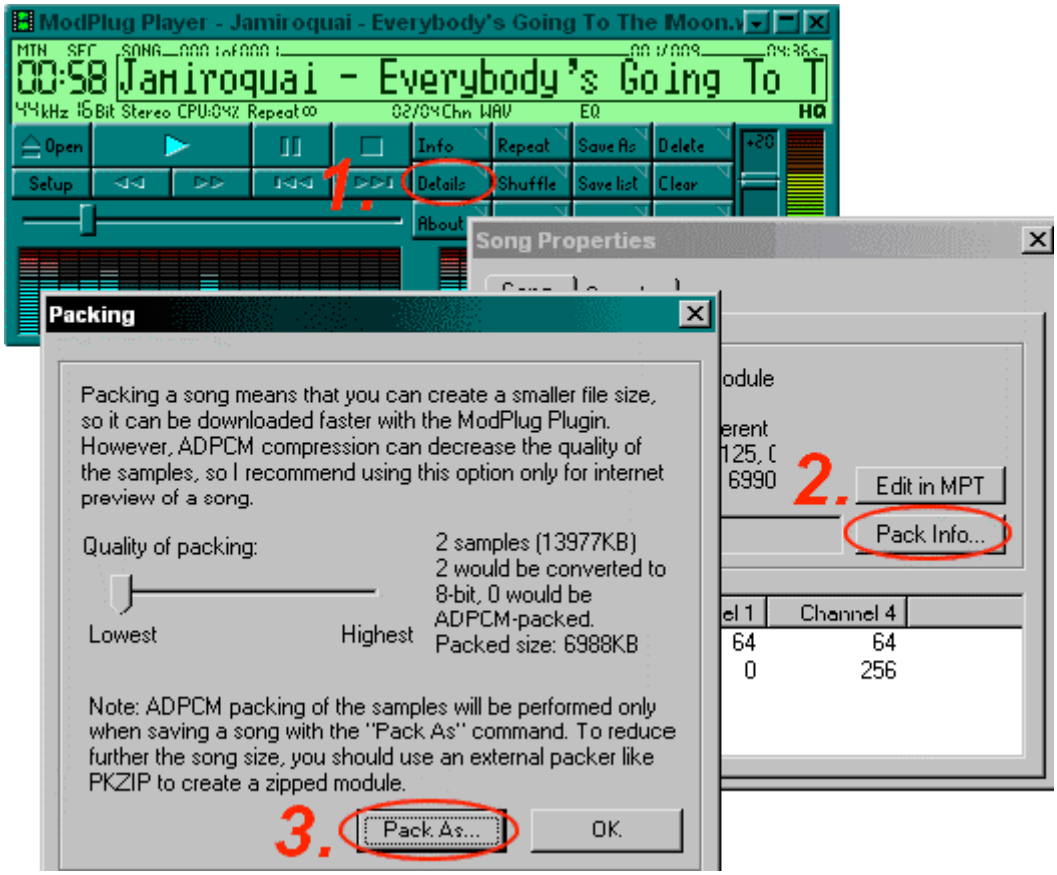
Für diesen Schritt könnt Ihr wieder ein beliebiges Programm benutzen, das diese Funktion unterstützt - ich benutze SoundForge, es geht aber auch mit dem Shaewareprogramm CoolEdit Pro.
Ihr ladet die Wave-Datei rein und wählt aus dem Menü "Edit/Convert Sample Type" oder drückt F11.
Hier stellt Ihr die gewünschte Qualität ein und drückt OK.
Ich empfehle die folgenden Einstellungen, weil sie Dateien mit dem besten Verhältnis zwischen Qualität und Dateigröße erzeugt.
Wer auch mit Mono-Musik zufrieden ist, der kann dafür die Sample Rate auf 22050 Hz stellen - der Ton klingt dann klarer, die Datei ist allerdings genauso groß wie bei diesen Einstellungen hier.



So, jetzt könnt Ihr Euch erstmal zurücklehnen und 'nen Tee schlürfen, denn das Konvertieren ist äußerst zeitraubend (ca. eine Viertelstunde für eine fünfminütige Datei auf einem K6-2 500).
Nachdem das ganze abgeschlossen ist, speichert die Datei am Besten unter einem neuen Namen.
Und auch CoolEdit Pro darf nun geschlossen werden.

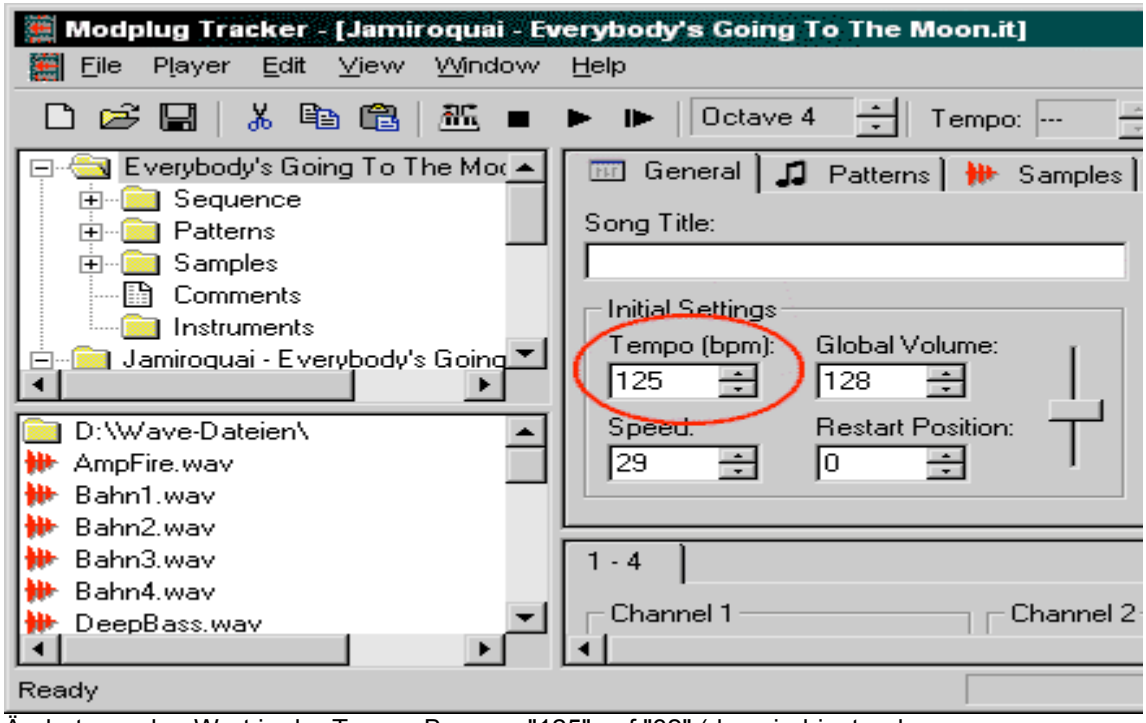
3. Schritt - Wave-Datei in Tracker konvertieren

So, jetzt beginnt der lustigste Teil - dazu benötigt ihr den ModPlug Player (MPP) und den ModPlug Tracker (MPT). Öffnet nun im MPP die konvertierte Wave-Datei (In der Öffnen-Box unten "All Formats" auswählen, dann erscheint auch die Wave-Datei) und klickt auf "Details", dann auf "Pack Info..." und dann auf "Pack as...".



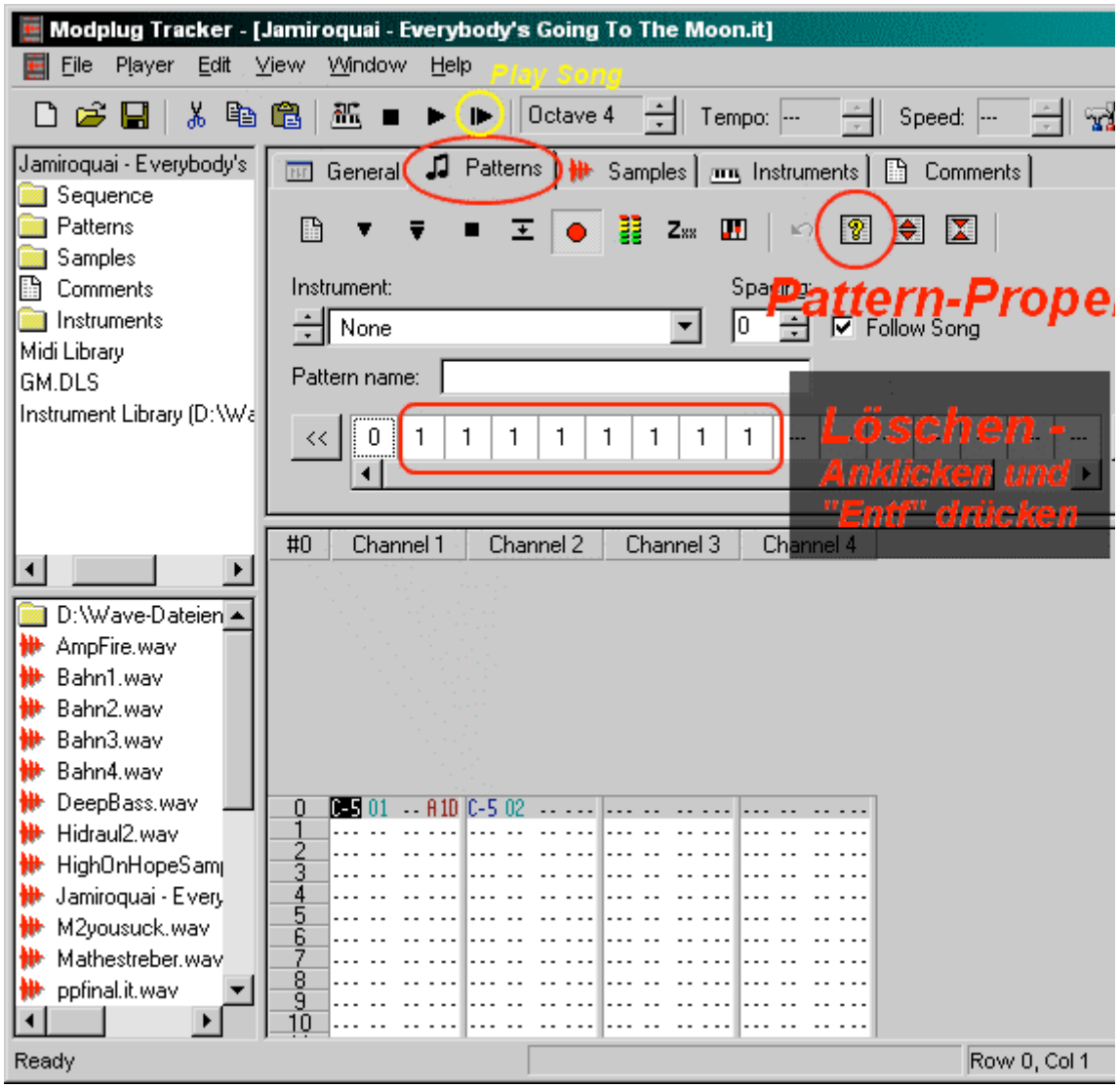
Hinweis: Der Button "Edit in MPT" auf der zweiten Dialogbox erscheint erst, nachdem man den ModPlug Tracker das erste Mal gestartet hat - dieser Button wird gleich noch benötigt! Achtet darauf, dass auf der dritten Dialogbox rechts neben den Schieberegler steht: "2 would be converted to 8-bit, 0 would be ADPCM-packed."
Nun speichert ihr das Ganze - am Besten als Impulse Tracker (*.IT).
Nach dem Speichern schliesst Ihr alle Dialogboxen ausser dem MPP.
Öffnet jetzt die IT-Datei im MPP, die Ihr grade gespeichert habt, geht nocheinmal ins Details-Menü und klickt nun auf den "Edit in MPT"-Button (siehe Hinweis).

Es müßte sich Euch nun ein ähnliches Bild zeigen:



Ändert nun den Wert in der Tempo-Box von "125" auf "32" (das niedrigste, das geht).

Geht nun auf die Patterns-Tab:



Hier müßt Ihr zuerst in den Pattern-Properties den höchsten Wert einstellen, den es gibt (256).
Dann könnt ihr die überflüssigen Patterns löschen, indem Ihr die ganzen Einser (die Null NICHT!) anklickt und die ENTF-Taste auf der Tastatur drückt.
Jetzt klickt Ihr auf den Play-Button (Dreieck Icon) und hört Euch euren Song an, bis er zu Ende ist - (Merkt euch die Song Position!)

Geht jetzt nocheinmal in die Pattern-Properties-Box und stellt jetzt dort den Wert, den Ihr Euch gemerkt habt, PLUS 1 ein -
z.B. der Song ist 145 Takte lang - Ihr stellt aber 146 ein, weil der nullte Takt mitgezählt wird. Alles klar? ;-)
Speichert das Ganze - jetzt seid Ihr FAST fertig!
Der MPP und der MPT können jetzt geschlossen werden.

4. Schritt - Importieren in den UnrealEd

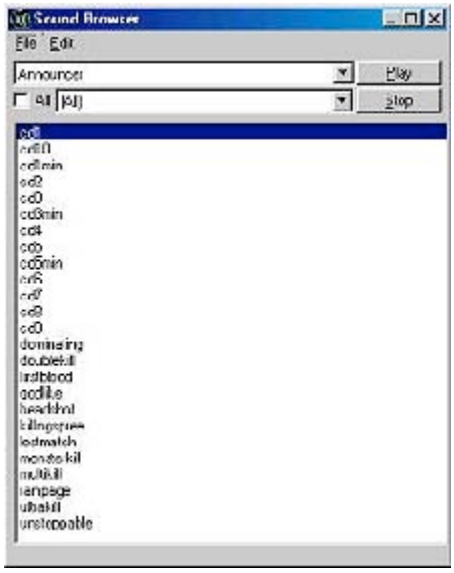
Um die so mühevoll konvertierte und editierte Datei jetzt noch benutzen zu können, öffnet Ihr den UnrealEd, geht dann in den Music-Browser, wählt den Menü-Punkt "Import...", sucht die gespeicherte IT-Datei, gebt einen Song-Namen ein und speichert abschliessend die Datei als UMX-Musik ab. Wie man einen fertigen UMX-Track im UnrealED einsetzen kann, zeigt euch das Sound und Musik Tutorial (http://unrealed.gamesweb.com/editor/2-sound.php), dort werden Sounds in den Editor eingebunden. Fertig! Ihr könnt die Musik jetzt in Euren Maps benutzen.

12.2. Voicepacks

12.2.1. Erstellen der .uax Datei

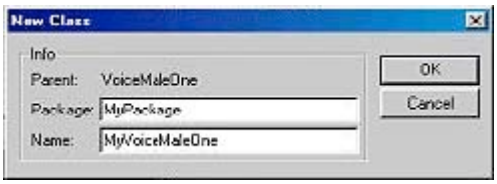
Zu einem VoicePack gehört eine .uax Datei, welche die einzelnen Sounds (.wav) enthält, sowie eine .u und eine .int Datei, die Informationen über das VoicePack enthalten.

Gehe im Editor auf **View/Sound Browser**.... Dort importierst Du deine Samples in dem Du auf **File** und dann auf **Import from .wav**... klickst danach auf OK klicken (Nie mehr als 10 auf einmal importieren, das macht der Editor nicht mit!). In das Feld das jetzt erscheint, gibst Du den Namen des Packages, z.B. MyVoices, ein und die Gruppe zu der das Sample gehören soll, z.B. Taunts - OK drücken – das erstellte Package im Browser auswählen und dann auf File/Save gehen, und unter dem Namen z.B. MyVoices abspeichern.

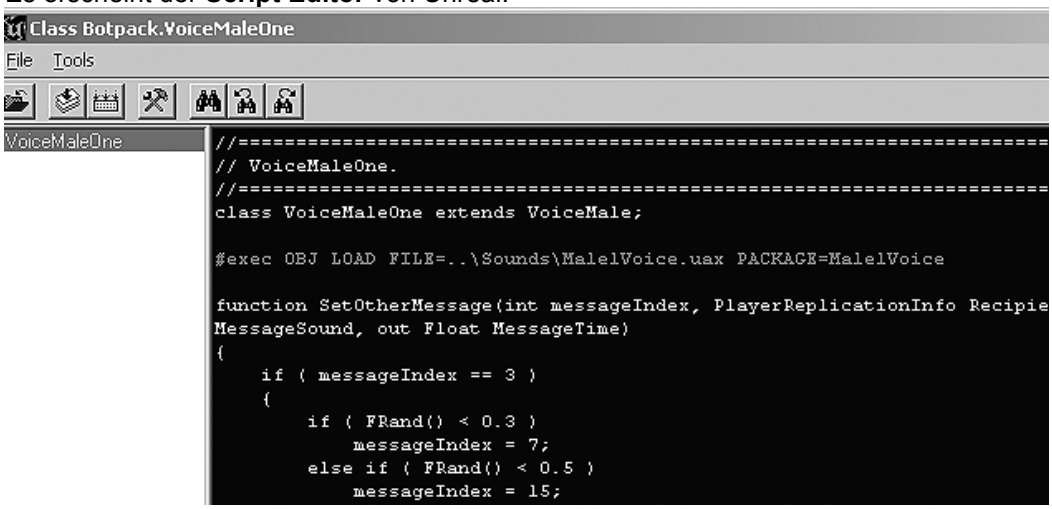


12.2.2. Erstellen und Bearbeiten der .u Datei

Öffne den ActorClass Browser und wähle unter Info/VoicePack/ChallengeVoicePack VoiceMale aus. Klicke es mit der RM-Taste an und wähle New.... Im nächsten Feld gibst Du den Namen der .uax Datei und den Namen, den die .u Datei haben soll an (nimm am Besten die gleichen Namen).



Es erscheint der **Script Editor** von Unreal.

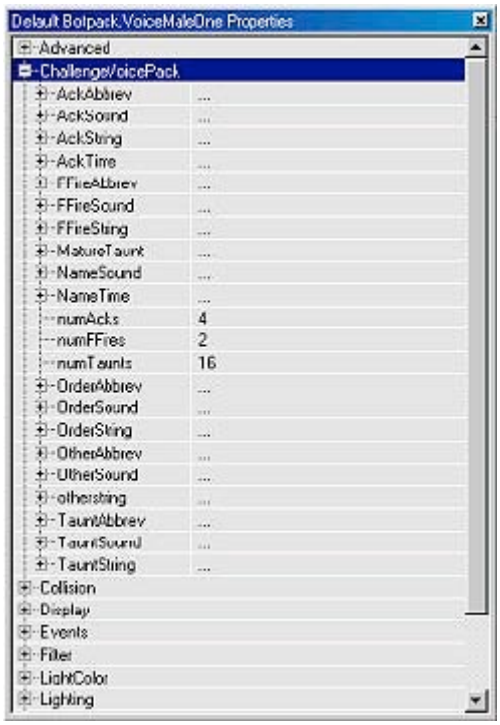


Nun öffnest Du noch das **Script von VoiceMaleOne**, in dem Du es im Actor Class Browser, unter VoiceMale, mit der RM-Taste auswählst und auf EditScript... klickst. Wähle nun den gesamten Text, angefangen bei #exec OBJ , aus. Drücke STRG+C , um den gewählten Text zu kopieren. Öffne das Script von deinem Package (es wird am Rand des Script Editors angezeigt) und drücke STRG+V, um den Text einzufügen. Nun mußt Du noch folgende Zeile abändern: #exec OBJ LOAD FILE=..\Sounds\Male1Voice.uax PACKAGE=Male1Voice. Gebe statt

Male1Voice.uax und Male1Voice den Namen von deiner .uax und .u Datei ein. Gehe dann auf Tools und **Compile Changed**. Jetzt noch schnell das Package im ActorClassBrowser unter File speichern.
In verschiedenen Tutorials wird fälschlicherweise behauptet, daß man damit ein spielfähiges Package und .u File erzeugt habe. Mit den beschriebenen Aktionen wird nur ein Folder mit dem Namen des Packages angelegt, in dem „Classes“, „Textures“, „Sounds“ Subfolder zu finden sind.
Um tatsächlich ein Package mit der Extension .u herzustellen, muß man von der Windows 2000 Konsole aus den Befehl „ucc make“ eingeben.

12.2.3. Bearbeiten der .u-Datei

Wir befinden uns immer noch im Actor Class Browser, aber sind unserem Ziel schon um einiges näher gekommen, da wir unsere .uax und .u Datei bereits haben. Jetzt müssen wir die .u Datei noch bearbeiten, damit man das VoicePack auch in UT verwenden kann. Klicke mit der RM-Taste auf Dein Package und wähle Default Properties. In diesem Fenster kannst Du die einzelnen Samples zu Taunts, Acknowledge, Order oder Other zuordnen. Dabei mußt Du aber einiges beachten! Bei Orders und Others kannst Du nicht einfach irgendwelche Samples hinein stellen, sondern Du mußt Dich an eine ganz bestimmte Reihenfolge halten:



Orders

0 Defend

1- Hold this Position

2- Attack

3- Cover me

4- Freelance

Others

0- Base uncovered

1- Feind hat die Flagge

2- Ich hab die Flagge

3- Got your Back

4- Im Hit

5- Man Down

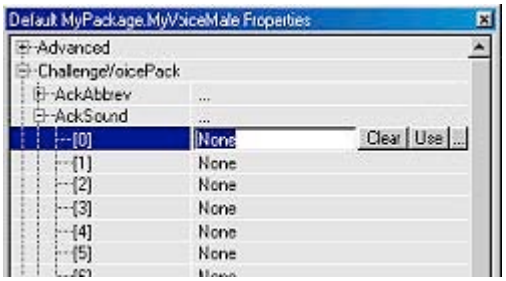
6- Under Attack

7- You got Point

E fehlt noch die .int Datei. Öffne im WindowsExplorer unter UnrealTournament\System eine neue .txt Datei und schreibe folgendes hinein:

[Public]
Object=(Name=MyVoices.MyVoices,Class=Class,MetaClass=Botpack.VoiceMale,
Description="MyVoices")

Wobei Du MyVoices gegen den Namen von deinem Package austauschst. Jetzt das ganze unter dem gewünschten Namen speichern. Aus dem .txt mußt Du noch ein .int machen! Nun noch den Namen des Packages in die UnrealTournament.ini zu den ServerPackages schreiben (SeverPackages=MyVoices) und fertig ist das VoicePack!


Default MyPackage.MyVoiceMale Properties
Advanced
ChallengeVoicePack
AckAbbrev
...
AckSound
...
[0]
None
Clear
Use
...
[1]
None
[2]
None
[3]
None
[4]
None
[5]
None

13. Capture the Flag (CTF) Maps erstellen

13.1. Einführung

Dieser Teil des Tuorials wendet sich an alle Mapper, die zum ersten Mal einen CTF Level bauen wollen. Um die einzelnen Schritte nachvollziehen zu können, müßtet ihr die grundlegenden Funktionen des Editors beherrschen.
Dieses Tutorial basiert auf der Version des Editors, die mit dem Patch 425 mitgeliefert wurde. Es läßt sich aber leicht auch auf die Version des ersten UEd2.0 übertragen. Auch mit dem älteren UEd1.0 wird man keine unüberwindlichen Probleme bekommen.
Das Tutorial ist in vier Teile unterteilt. Die ersten beiden Teile befassen sich mit der Erstellung der Geometrie des Levels. Die letzten beiden Teile befaßen sich mit dem Verhalten der Bots.

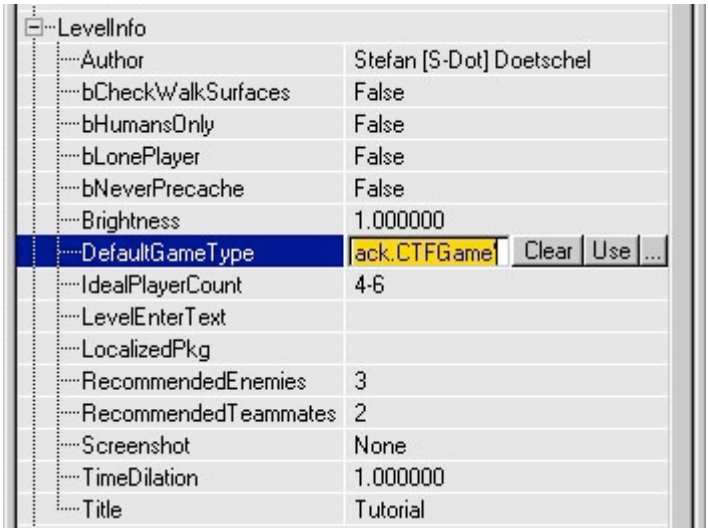
13.2. Was ist CTF?

Kurz gesagt ist **CTF** ein Team Game in dem jede Mannschaft eine Basis hat. In dieser Basis steht eine **Flag** (Fahne). Der Spot auf dem sie steht ist die **FlagBase.** Diese Fahne muß bewacht werden. Gleichzeitig muß die gegnerischen Fahne gestohlen werden.
Bringt man nun die gegnerische Fahne in die eigene Basis zurück und läuft mit ihr über die FlagBase, so bekommt das eigene Team einen Punkt, wenn die eigene Fahne zu Hause ist. Das Team, das als erstes 5 Punkte hat, hat gewonnen.

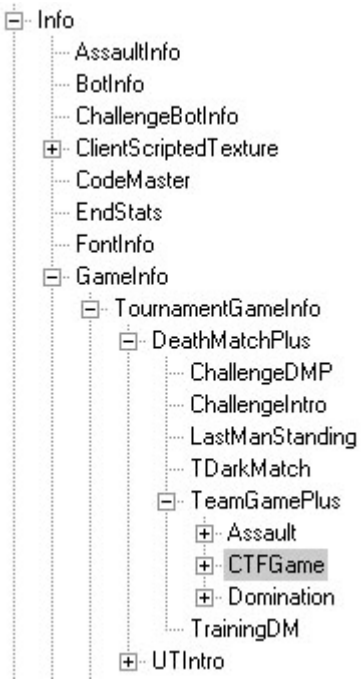
13.3. Vorbereitung

Zunächst müssen in der Engine einstellen, daß wir ein **Capture The Flag** Level bauen wollen.

Öffne durch das Drücken der **Taste F6** die **LevelProperties**.



Wie immer sind eine Fülle von Einstellungen möglich, von denen aber nur wenige notwendig sind und überhaupt Wirkung zeigen. Öffne den Abschnitt **LevelInfo**
Erforderlich sind für uns nur: **Author**
DefaultGameType (dazu kommen wir gleich
IdealPlayerCount (optimale Anzahl der Spieler
Title



Alle anderen Felder bleiben unverändert.

Neben dem Eintrag **DefaultGameType** seht ihr drei kleine

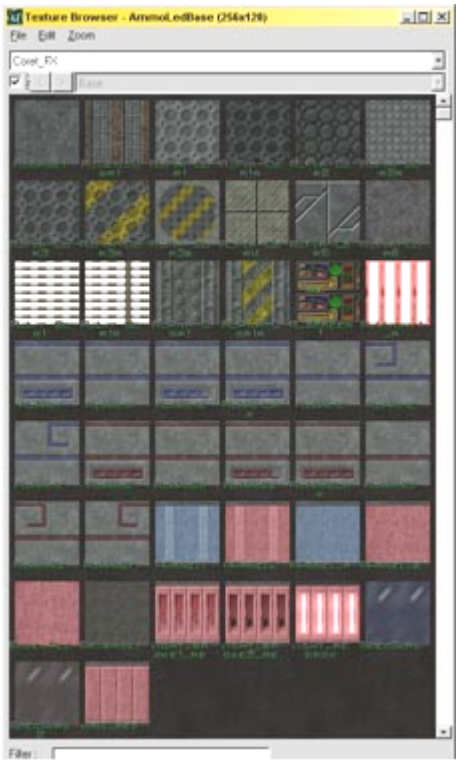
Schaltflächen. 

Drückt auf die ganz Rechte davon, die mit den Punkten. Es öffnet sich der ActorBrowser.

Erweitert dort die Einträge unter **Info** so lange, bis ihr zu **CTFGame** kommt. Dies laßt ihr so markiert, wie es links zu sehen ist und drückt dann in den LevelProperties den mittleren Button "Use". Jetzt wird der GameType eingetragen und ihr könnt die LevelProperties wieder schließen.

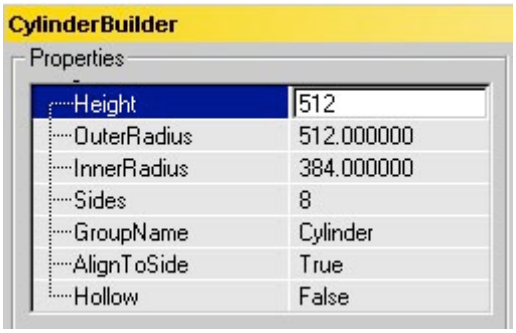
So, es wird Zeit die Map zu speichern. Geht auf **File > Save as ...** und nennt sie z.B. "CTF-UEdTutorial.unr". Damit ist der erste Schritt schon abgeschlossen.

13.4. Geometrie



Damit das hier nicht zu lange dauert, werden wir die Geometrie auf das nötigste vereinfachen.
Der Einfachheit halber benutzen wir nur Textures aus der Datei **Coret_FX.utx** und für die Flaggensymbole die Datei **CTF.utx.** Diese kannst Du mit dem TextureBrowser schon mal laden.

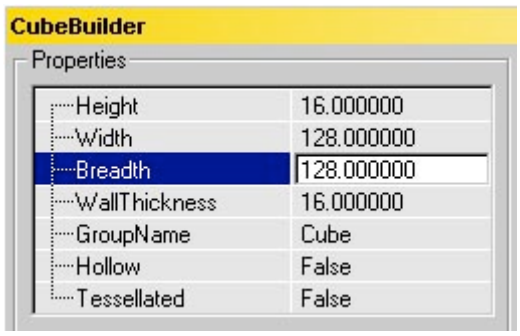
13.4.1. Die rote Basis



Als erstes wollen wir die rote Basis bauen. Öffne die Coret_FX Textures und markiere die Texture **FL-Wal20** (mit einem roten Streifen). **Subtract** dann einen zylindrischen Raum mit den Einstellungen links.

Belege den Boden mit der Texture **FL_Flr_Cor3** (Noppen) und die Decke mit **FL_Base1** (grau marmoriert).

Tip: Wenn Du eine Texture einer Brush markierst kannst Du mit SHIFT+B alle anderen Texturen dieser Brushes markieren. Wenn Du die STRG-Taste festhältst kannst Du danach einzelne Texturen wieder abziehen bzw. hinzufügen

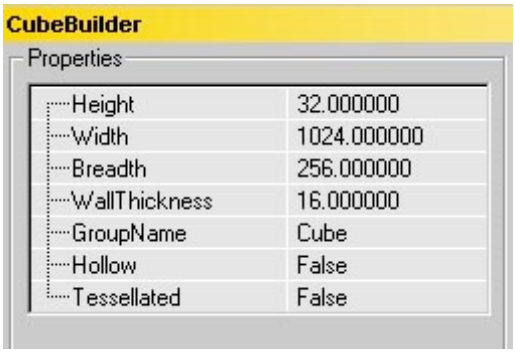


Als nächstes kommt die Basis. Markiere die Texture **Br_Base1** (marmoriert) und erstelle einen Cube mit den Abmessungen links.

Positioniere ihn Direkt auf den Boden, dazu mußt Du unten im Editor das Raster auf 8 Einheiten stellen.

Die Oberseite, auf der die Fahne nachher stehen wird, ändern wir in die Texture **W100_Red** (mit dem roten Symbol). Diesmal aus dem **CTF.utx** Paket.

Die Texture ist zu groß. Deshalb öffnen wir mit F5 die Surface Properties **und skalieren sie auf 0,5. Drehen müssen wir sie auch noch um 90 Grad. Markiere außerdem den Flag** Special Lit**, denn die Basis soll später pulsieren.**



Dann bauen wir noch eine Ebene in der Mitte der Flagbase ein. Die Abmessungen stehen wie immer links. Die Texture heißt diesmal A_Flr_Cor6 (Metallboden).

Schiebe die rote Brush auf die linke Seite und **Deintersect**, so daß er in den Zylinder paßt. 

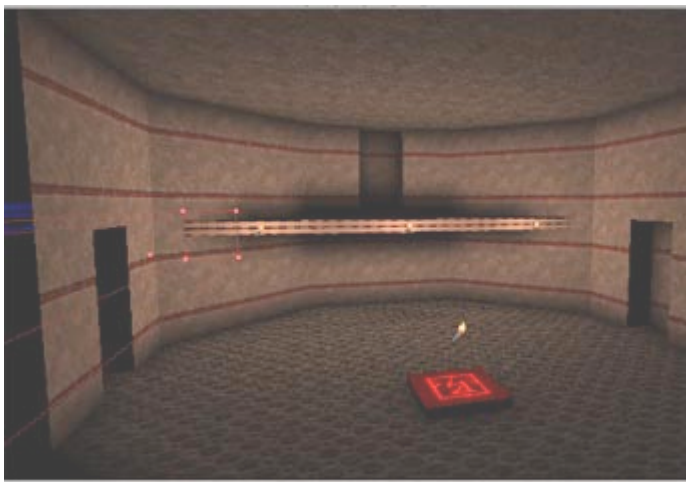
Jetzt sieht das ganze fast schon so aus:

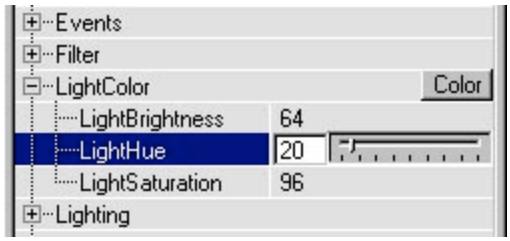
Ihr seht hier, daß ich die Kante der Galerie mit einer Lampentextur belegt habe.
Sie heißt **A_Lit_Cor1.**

Setze für sie die beiden Flags:

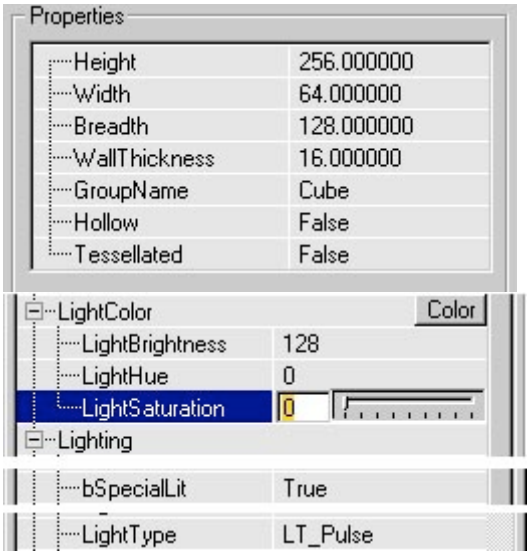
HighShadowDetails

BrightCorners





Vor diese Kante hab ich auch gleich noch drei **Lichter** gesetzt und deren Properties eingestellt, damit die Basis in einem rötlichen Licht erscheint



Die pulsierende Flagbase erreicht man durch setzen eines **Lichts** mit diesen Einstellungen.

Lighting -> bSpecialLit True

und -> LightType LT_Pulse

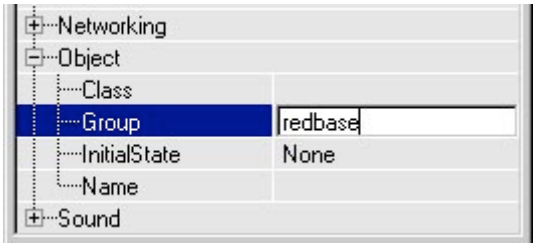
Dazu hast Du vorhin die Texture auf **SpecialLit** gesetzt. So pulsiert nun das rote Symbol, während der Rest der Base ganz normal beleuchtet bleiben.

Um diesen Effekt im Editor zu sehen, muß **Realtime Preview** aktiviert sein (Joystick-Symbol über dem jeweiligen Fenster).

Wie ihr seht hab ich auch noch zwei Türen rechts und links eingesetzt und eine oben auf der Empore. Diese bekommen wieder die selbe Texture **FL-Wal20** wie die Wände.

Rebuild nicht vergessen.

Tip: zum Kopieren von Textures geht man wie folgt vor:
1. Markiere die zu übertragende Texture und klicke mit Alt + rechte Maustaste auf sie
2. Markiere die Texture die genauso tapeziert werden soll und klicke mit Alt + LM-Taste auf sie.



Wer schon einmal mit CAD-Programmen gearbeitet hat, weiß es zu schätzen, daß der Editor auch die Möglichkeit bietet, Gruppen zu bilden.

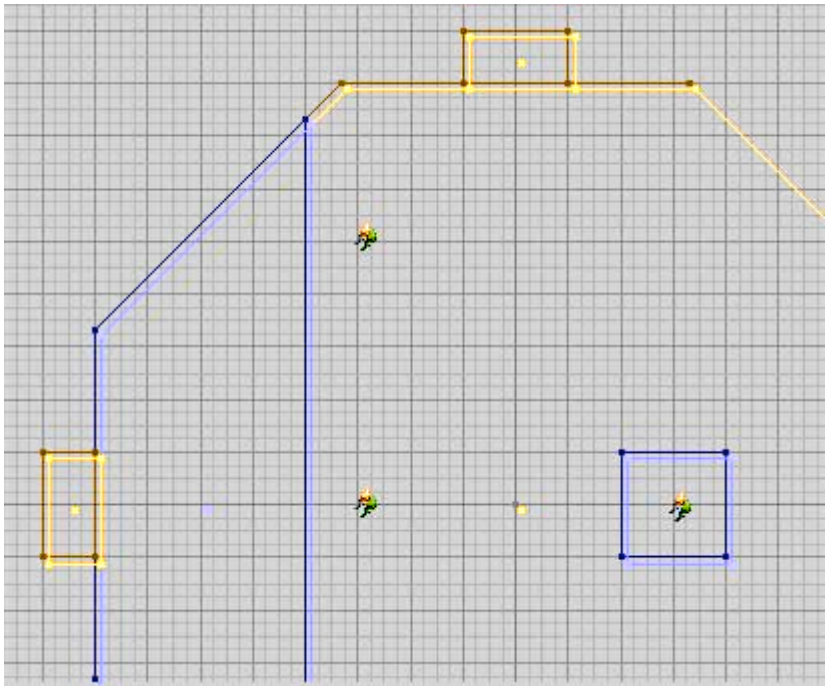
Markiere alle Elemente indem Du auf eines mit der RM-Taste klickst und aus dem Kontextmenü **Select All** auswählst.

Dann klicke mit der RM-Taste auf eines der Elemente und wähle den obersten Eintrag **Actor Properties (10 Selected).**

In dem Property Fenster trägst Du unter **Object -> Group** einen Namen z.B. "redbase" ein**.**

13.4.2. Die blaue Basis

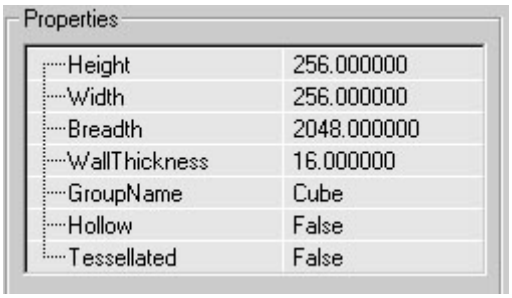
Die 10 Actors sind immer noch markiert. Klicke wieder mit der RM-Taste auf eines der Elemente und wähle diesmal den Eintrag **Duplicate.**
Jetzt hast Du alle Elemente kopiert und diese sind markiert. Klicke vorsichtig mit der RM-Taste auf eines der neuen Elemente und wähle wieder **Actor Properties (10 Selected).**
Dies wird die blaue Basis, also nenne sie unter **Object -> Group** "bluebase". Schiebe sie danach aus der roten Basis heraus auf eine freie Stelle im Level.
Die kopierten Elemente sieht man in der 3D-Ansicht erst nach dem **Rebuild**



Tip: Wenn Du die Auswahl verlierst, indem Du neben die Brush klickst, dann drücke einfach die **Undo** Funktion und die Auswahl ist wieder markiert.

13.4.3. Verbindungen

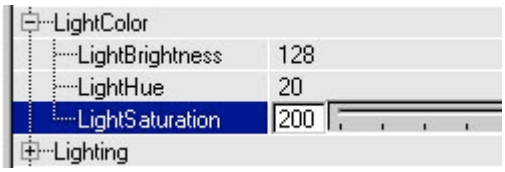
Jetzt kommen wir zu den Verbindungen zwischen den Basen. Du solltest mindestens zwei davon haben plus eine, die nicht auf Anhieb zu finden ist. Außerdem müssen alle diese Wege ungefähr gleich lang sein. Wenn einer der Wege dreimal so lang ist wie die anderen, dann werden die Bots diesen Weg so gut wie nie benutzen. Warum das so ist erfährst Du später bei den **AlternatePaths.**



Wir bauen rechts und links der Basis zwei Gänge mit diesen Abmessungen.

Diesmal nehmen wir für die Wände die Texture **A_Wal2** mit den langen Rechtecken zusätzlich zu den Streifen, um die Gänge etwas interessanter zu machen.

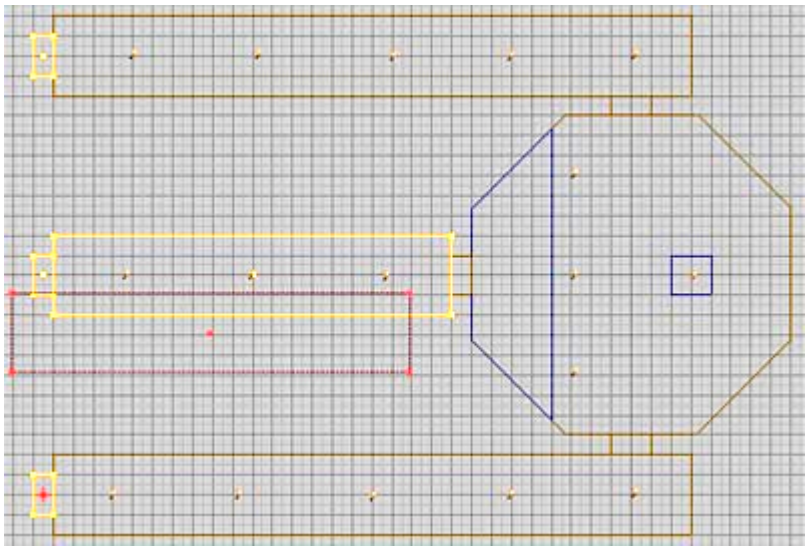
Fußboden und Decke belegen wir genauso wie die Basis.



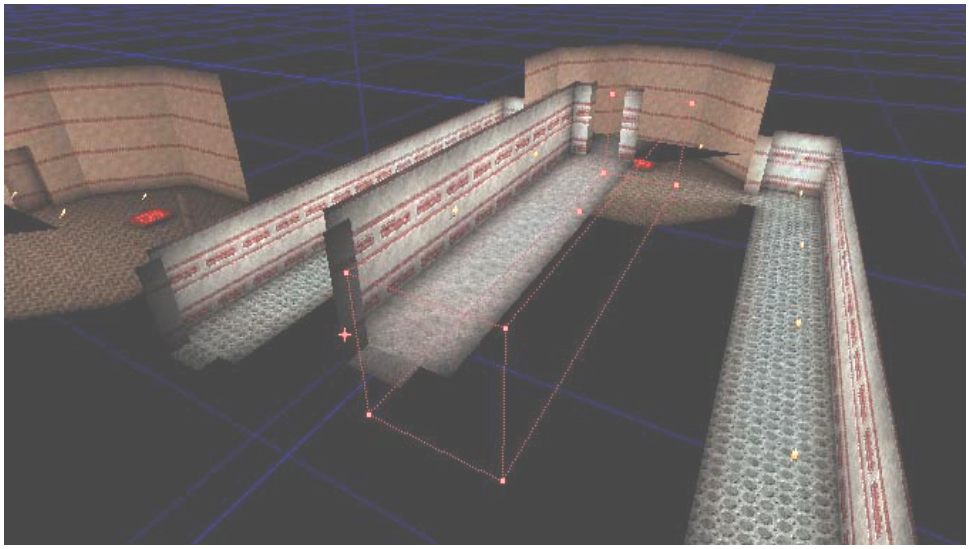
Verteile 5 Lights gleichmäßig im Gang, um ihn auszuleuchten. Eigentlich sollte man jetzt noch Beleuchtungskörper erstellen, von denen das Licht abstrahlt, aber das lassen wir weg.

LightHue (Farbe) ist wie in den Basen. Durch Erhöhung der **LightSaturation** (Sättigung) wird das Licht weniger Rot.

Erstelle am besten ein Light und kopiere es dann 4 mal.

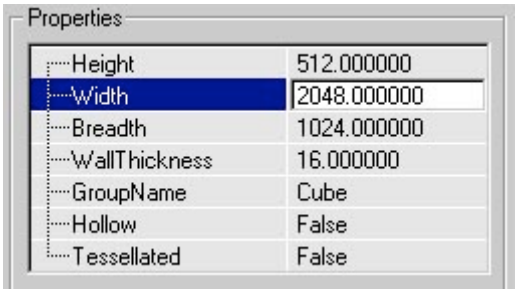


Markiere den Gang und die 5 Lights und bezeichne sie unter **Object > Group** mit "redhw". Kopiere den Gang für die andere Seite (kann die selbe Gruppe bleiben).
Erstelle noch einen Gang oben mit den gleichen Abmessungen und der Länge 1280 und positioniere diesen vor die obere Tür.
Nehme drei der Lights aus dem langen Gang, kopiere sie und schiebe sie in den neuen Gang. Auf die richtige Höhe achten!
Dann habe ich noch die schon vorhandenen Türen kopiert und sie ans Ende des Ganges geschoben und richtig gedreht.
Außerdem habe ich die zuletzt erstellten Türen und den oberen Gang der gleichen Group "redhw" zugeordnet. (Ist im Screenshot noch markiert)
Man kann sich also viel Arbeit ersparen, wenn man Elemente richtig fertig stellt und dann kopiert. Das sieht nicht langweilig aus, wenn man es nicht übertreibt. Es bringt sogar Ruhe in die Architektur des Levels. Ziel ist es nicht, möglichst viele verschiedene Texturen zu verwenden, sondern die Texturen richtig einzusetzen und auszurichten.
Tip: Elemente dreht man durch drücken der Taste STRG + RM-Taste und bewegen der Maus. Manchmal muß man den **Pivot** (roter dickerer Punkt) mit der RM-Taste auf einen anderen Schnittpunkt setzen.



So sollte das ganze jetzt aussehen. Die Gänge sollten auf einer Linie abschließen. Links sieht man die kopierte zweite Basis. Spätestens jetzt solltest Du den Level sichern, damit im Falle eines Absturzes nicht alles verloren geht. Speicher häufiger mal unter neuen Namen wie z.B. "CTF-UEdTutorial1.unr" usw. Dann hast Du den letzten Stand, falls Du einen schweren Fehler machst oder der Level sich nicht mehr laden läßt.

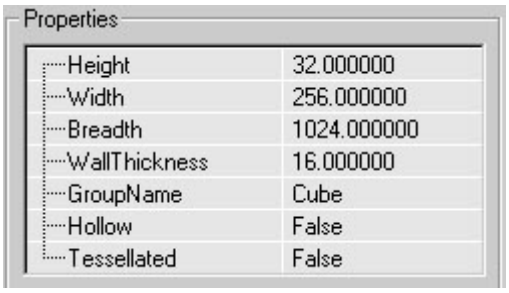
13.4.4. Level Center



Jetzt kommt der mittlere Raum. Er ist recht groß. Richte ihn genau mittig zu den Gängen aus. für die Wand nimm die Texture A_Flr_Cor4. Stelle für sie in den Surface Properties das Scaling auf 2.0, damit sie nicht in den Augen flirrt.

Jetzt markiere einen der vorhin erstellten Gänge und wähle mit der RM-Taste **Select All.** Alle Elemente mit der Gruppenbezeichnung "redhw" werden markiert. Kopiere sie und gebe den neue entstandenen Elementen die Gruppenbezeichnung "bluehw". Solange sie noch markiert sind, kannst Du sie auch gleich an die richtige Stelle drehen und verschieben.

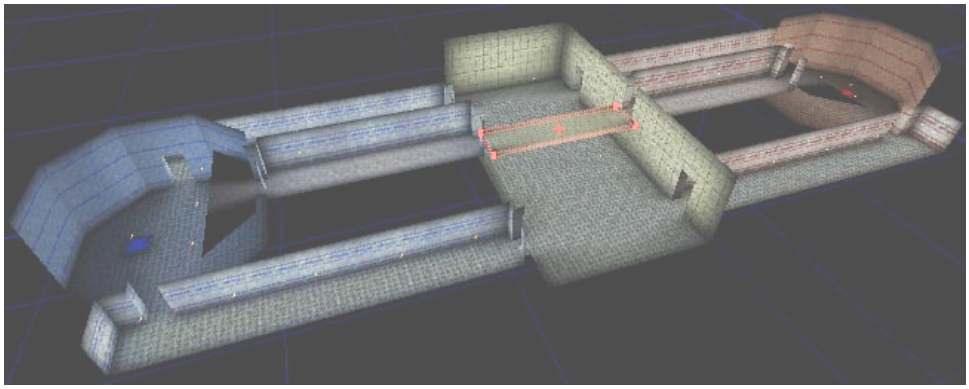
Genauso verfährst Du mit der vorhin kopierten zweiten Basis. Wähle Select All und drehe und verschiebe sie so, daß ein symmetrischer Level mit zwei sich gegenüber liegenden Basen entsteht.



Beleuchte den mittleren Raum. Ich habe zwei Lichter genommen, für die ich die Brightness auf 255 und die Saturation auf 222 gestellt habe. Das ist ein neutrales Licht.

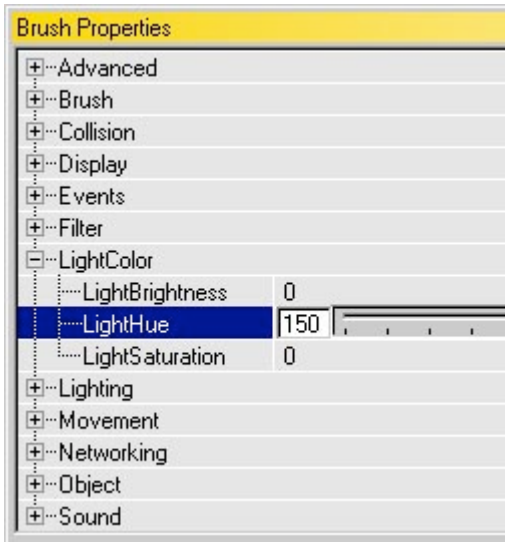
Außerdem habe ich noch einen Steg zur VerbinDung der oberen Gänge eingebaut. Die Texture ist die gleich der Wände aber ohne Scaling

Nachdem **Rebuild** ist die zweite Basis auch in der 3D-Ansicht sichtbar:



Die Basen sind jetzt beide rot. Das müssen wir ändern. Ich habe alle gleichen roten Texturen durch gedrückt halten der STRG Taste markiert und jeweils durch die entsprechende blaue Texture ersetzt. Vielleicht geht das auch einfacher.

Decke und Boden sind neutral. Markiere eine der Boden Texturen, drücke SHIFT+T und wähle mit der RM-Taste **Align Selected - Align as Floor / Ceiling.** Damit sind alle Texturen für unsere Zwecke ausreichend ausgerichtet.



Jetzt müssen noch die Lichter in der blauen Basis blau beleuchtet werden.

Um alle Lichter auf einmal zu ändern, wenden wir einen kleinen Trick an:

Markiere eine Brush der blauen Basis und eine der blauen Gänge.

Wähle **Select All** (rechte Maustaste)

Stelle in den Properties **LightColor -> LightHue** auf 150 (blau). Lasse alles andere wie es ist.

Klicke irgendwo ins Leere, um die Markierung aufzuheben.

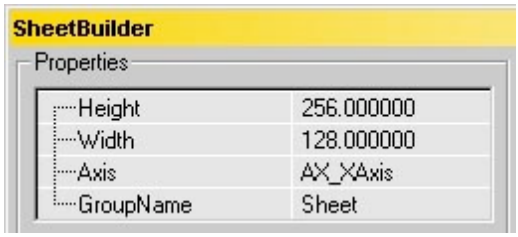
Klicke auf irgendeine Brush und wähle **Select All Brush** (RM-Taste)

Stelle den **LightHue** für alle Brushes wieder auf 0 zurück

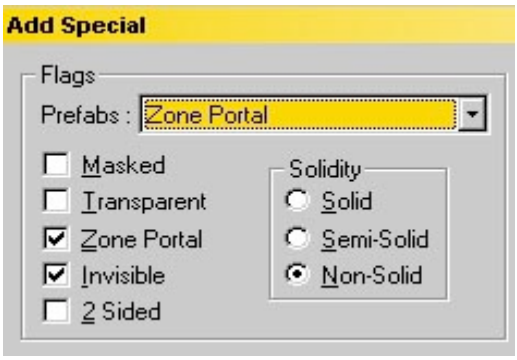
13.5. Botpathing

13.5.1. Zonen

Jetzt setzen wir in alle Türen **Zone Portals**. Der Level ist zwar nicht groß und die Performance wäre sicher in Ordnung. Wir brauchen die Zonen aber, um im Spiel zu wissen, wo sich unsere Mitspieler befinden.



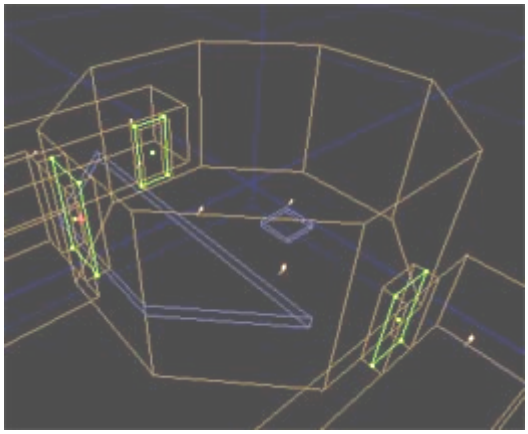
Baue ein Sheet mit den folgenden Abmessungen.



Dann wähle **Add Special Brush** (das Dunkelblaue Rechteck) und wähle das **Prefab Zone Portal** aus der Liste aus. Lasse die Einstellungen wie sie vorgegeben sind.

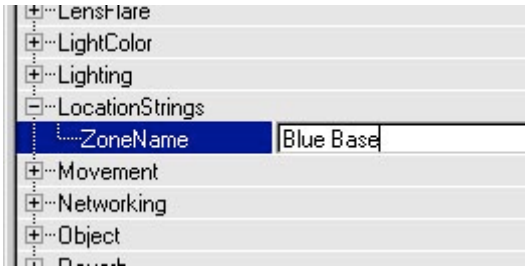
Du kannst Dir die Arbeit erleichtern, indem Du ein oder mehrere Portale so oft kopierst, bis alle Türen versiegelt sind.

Im Bild rechts seht ihr, wie drei Portale genau in der Türöffnung sitzen.



Rebuild und wähle **Portal View** (Symbol ganz rechts über dem 3D Fenster). Alle Zonen müssen farblich voneinander getrennt sein.

Öffne den **Actor Class Browser** und markiere unter **Info -> ZoneInfo** (das mit dem Pluszeichen, keine Unterklasse davon).



Setze in jede Zone ein **ZoneInfo**

In den ZoneInfo Properties findest Du einen Eintrag **LocationString.** Bei einer Nachricht eines eigenen Spielers, wird dieser in Klammern hinzugefügt. So kann man immer sehen, wo sich die eigenen Mitspieler befinden.

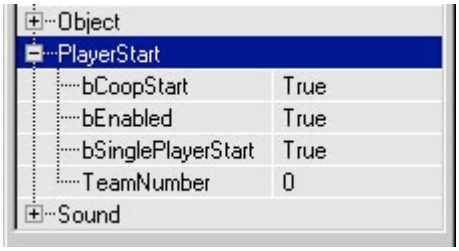
Nehme kurze Namen wie "Blue Base", "Center", "Upper Passage (blue)" etc.

13.5.2. PlayerStart

Die CTF-spezifischen NavigationPoints

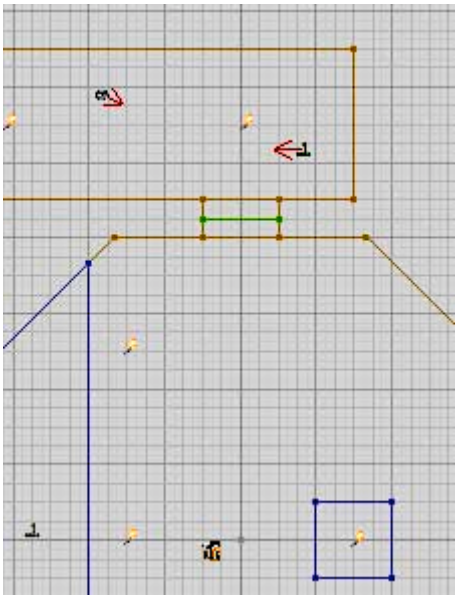
Alle **NavigationPoints**, die für CTF Level wichtig sind befinden sich im Actor Browser unter - ja richtig - **NavigationPoints**.
Als erstes setzt Du die **Playerstart**. Standardmäßig sind maximal 16 Spieler möglich, deshalb würde ich grundsätzlich auf jeder Seite 8 **Playerstart** einsetzen, auch wenn die Map wie diese hier ziemlich klein ist. So vermeidest Du Telefragging, wenn Du aus Versehen mit mehr als der empfohlenen Anzahl Spieler spielst.
Ein paar allgemeine Regeln: Setze die **Playerstart** der eigenen Mannschaft in die Nähe der eigenen Basis. Setze die **Playerstarts** nicht in der gegnerischen Hälfte. Setze aber nicht zu viele allzu Dicht an die eigene Basis. Sonst müßte man jeden Gegner 3 mal hintereinander besiegen, wenn man die gegnerische Fahne geklaut hat.

Setze sie auch nicht mitten in die Gänge, sonst könnte ebenfalls ein Spieler der gerade vorbeiläuft telefragged werden.



Setze den ersten **Playerstart** in die rote Basis, gegenüber der **FlagBase**.

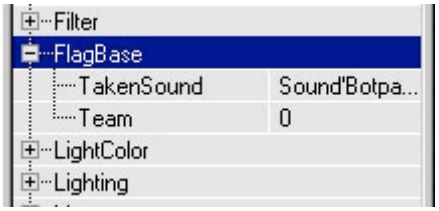
Die einzige Einstellung die Du vornehmen mußt ist die Zuweisung des Teams, also welcher Mannschaft dieser Start gehört. Sie befindet sich unter den **Properties** unter **Playerstart -> TeamNumber** > Team Red = 0 Team Blue = 1



Setze je einen **Playerstart** in die Nischen neben den unteren Ausgängen aus der Basis. Wenn Du die **Playerstarts** markierst, bekommt er einen kleinen roten Pfeil.
Diesen kannst Du durch drücken von STRG + RM-Taste drehen. Der Pfeil zeigt die Richtung an, in den der Spieler beim erscheinen blickt. In unserem Fall also den Gang entlang.
Setze noch zwei **Playerstarts** in die Ecken des großen Raumes in der Mitte, in der Nähe der Ausgänge der unteren roten Gänge und richte sie aus.
Das wären dann 5 und das soll erstmals genügen. Setze die **Playerstarts** nicht zu Dicht an die Wände, sondern mit einem Abstand von ca. 64 Einheiten.

Jetzt mußt Du noch die 5 **Playerstarts** des blauen Teams entsprechend setzen. Ich würde die 5 roten **Playerstarts** markieren und Kopieren. Dann würde ich die Kopierten in die blaue Basis verschieben und drehen. Zuletzt würde ich, so lange sie noch alle 5 markiert sind, ihre **Properties** öffnen und ihnen unter **Playerstart -> TeamNumber** den Wert "1" geben.

13.5.3. FlagBase



Du findest sie im Actor Browser unter **NavigationPoints > FlagBase**

Auch für sie mußt Du die Team-Zugehörigkeit einstellen. Es gilt genauso wie für alle **NavigationPoints**

Team rot = 0 Team blau = 1

Setze die beiden Flaggen auf die pulsierenden Podeste und drehe sie im Grundriß richtig.

Im Editor werden beide Flaggen blau dargestellt, das ist OK.

Da Du jetzt **Playerstarts** im Level hast, kannst Du sie eigentlich auch mal testen. Vorher mußt Du aber ein **Rebuild** ausführen. Ich mach das immer in folgender Reihenfolge

.

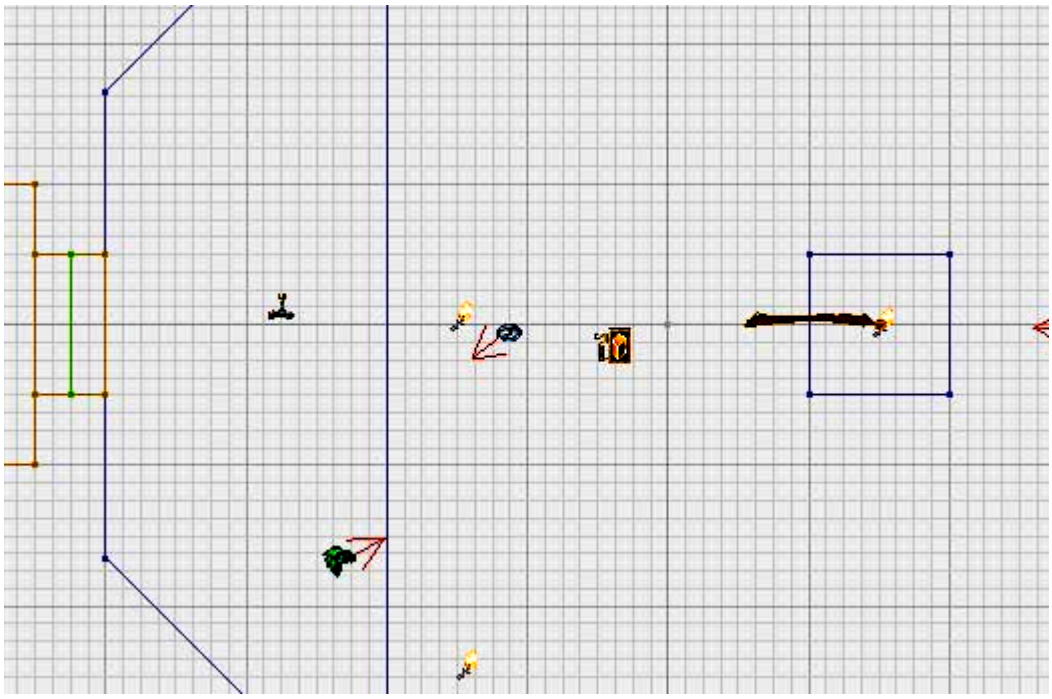
Vielleicht ist das übertrieben, es erspart aber merkwürdige Abstürze und ähnliches.

Speichere ihn danach ab und klicke auf das Joystick Symbol in der oberen Menüleiste neben den Rebuild Symbolen und UT startet den Level.

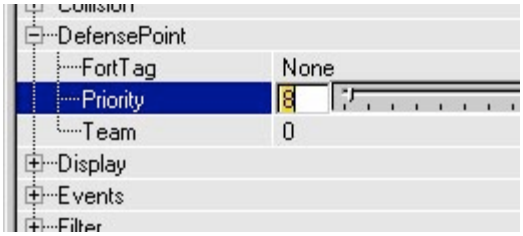
Wirst Du gleich beim Spielstart telefragged, so mußt Du die Taste ESC drücken und unter **Start Practice Session** die Bot Anzahl auf 6 stellen. Danach mußt Du den Level aus dem Spiel heraus neu starten. Da noch keine Pfade gesetzt sind, reagieren die Bots erst, wenn sie Dich sehen. Das wird jetzt geändert:

13.5.4. DefensePoint

Ein **DefensePoint** gibt an, von wo ein defensiv eingestellter Bot die eigene Basis verteidigen soll.
DefensePoints findest Du ebenfalls unter **NavigationPoint.** Es ist eine Unterklasse von **AmbushPoint**, Du mußt also im Actor Browser auf das Pluszeichen vor **AmbushPoint** klicken, damit er erscheint.



Setze je einen **DefensePoint** direkt hinter die Flagge in der Mitte zwischen Podest und Wand. Stelle in den **Properties** das **Team** ein. Außerdem mußt Du wie gehabt die Blickrichtung richtig drehen.
Wenn Du mehrere **DefensePoints** je Basis hast (und das solltest Du), dann kannst Du den Wert **Priority** einstellen. Ein **DefensePoint** mit höherer Priority wird eher besetzt als einer mit niedrigerer.



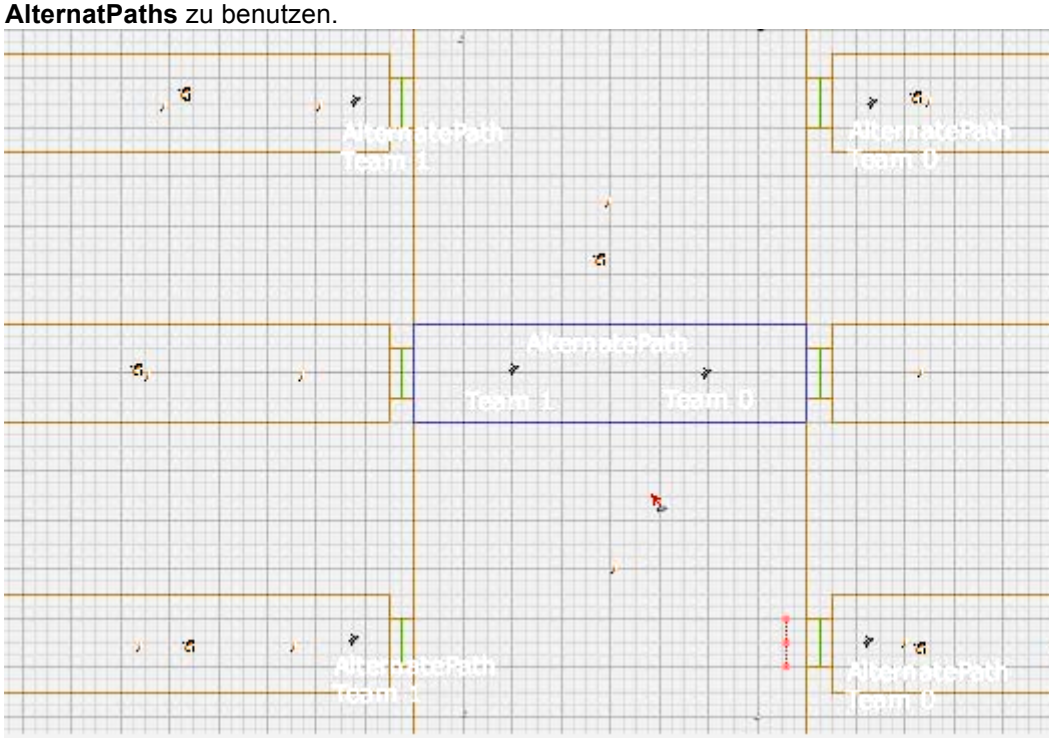
Setze also je einen zweiten **DefensePoint** seitlich oben auf die Galerie, so das der Gegner einen verteidigenden Bot nicht sofort sieht, wenn er den oberen Gang entlang kommt.

Setze die **Priority** für den unteren **DefensePoint** auf "8", damit dieser zuerst besetzt wird.

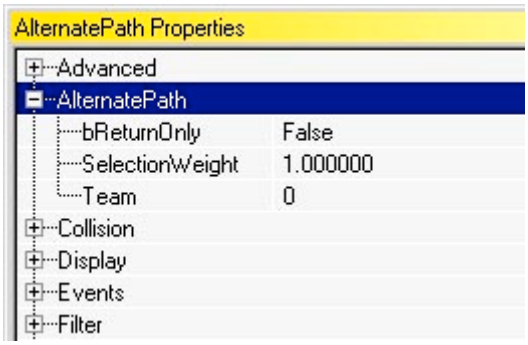
13.5.5. AlternatePath

Kommen wir zu den **AlternatePaths**. Grundsätzlich brauchst Du keine **AlternatePaths** in deinem Level. Die Bots finden auch so ihren Weg.
Wie funktionieren **AlternatePaths**?
Wenn der **Bot respawned**, dann sucht er sich einen Weg zu einem **AlternatePath** seiner Wahl und von dort aus den schnellsten Weg zur gegnerischen **FlagBase**. Du kannst so die Richtung vorgeben, in die ein Bot zunächst gehen soll. Er wird immer erst zu seinem **AlternatePath** gehen, auch wenn der Weg direkt an der gegnerischen **FlagBase** vorbei geht (Was eine sehr dumme Sache wäre). Hat er den **AlternatePath** erreicht, dann sucht er sich von dort den schnellsten Direkten Weg zur gegnerischen **FlagBase**. Er geht also nicht von **AlternatePath** zu **AlternatePath**.
Ich habe bisher in jeder Epic Map **AlternatePaths** gefunden. Man kann durch sie die Bots auf Wege locken, die sie sonst nicht benutzen würden, weil sie z.B. zu lang wären. Allerdings muß man aufpassen, daß der

kürzeste Weg nicht wieder zurück zur Basis führt. Sonst geht der Bot zum **AlternatePath**, dreht dann um und geht wieder zurück.
Setze also die **AlternatePaths** der jeweiligen Mannschaft immer in die eigene Hälfte, für den Rückweg scheinen die Bots die gegnerischen **AlternatPaths** zu benutzen.



Setze je zwei **AlternatePaths** ans Ende der jeweiligen unteren Gänge (nahe dem Zentrum) und je einen oben auf den Steg, der die oberen Gänge verbindet.



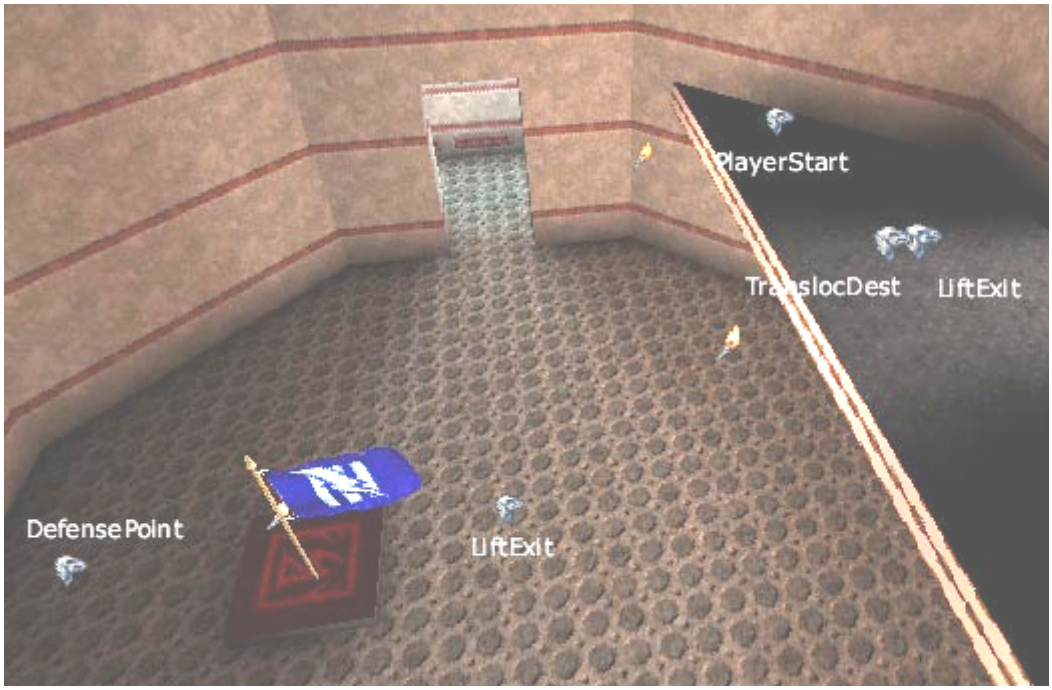
bReturnOnly gibt an, daß der Bot den Path benutzen soll, wenn er die Fahne hat, also für den Rückweg. Leider habe ich keine guten Erfahrungen mit dieser Einstellungen gemacht. Ich empfehle also, **AlternatePaths** nicht für den Rückweg zu setzen.

SelectionWeight: hier gilt das gleiche. Es scheint relativ egal zu sein, was man hier einstellt, also laßt es wie es ist.

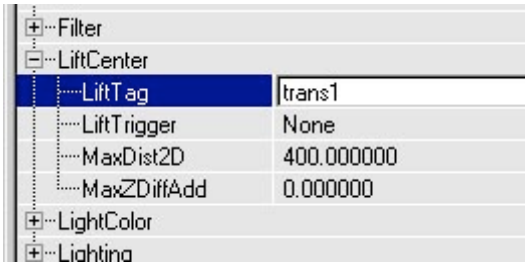
Die Einstellung **Team** ist wohl inzwischen klar.
Setze alle **NavigationPoints** für beide Teams, führe ein volles **Rebuild** aus, wie ich es oben beschreiben habe, speichere und teste den Level.
Es ist noch genauso wie beim ersten Test, die Bots stehen nur herum, es sei denn sie sehen einen oder man schießt auf sie. Das wird sich gleich ändern, denn wir machen den Rest des Pathings. Die nachfolgenden **NavigationPoints** sind unabhängig vom Team, beide Teams benutzen die selben.

13.5.6. TranslocDest

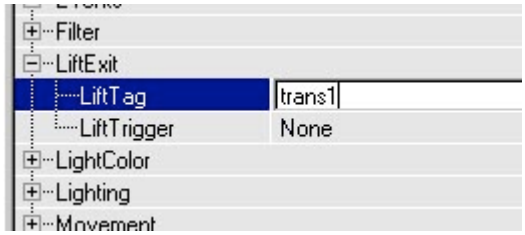
Auf den oberen Gang kommt man mit dem **Translocator**. Damit ihn die Bots benutzen, mußt Du **TranslocDests** setzen. Er ist eine Unterklasse von **LiftCenter**, befindet sich also "hinter" dem **LiftCenter** im Actor Browser.



Setze einen **TranslocDest** dort, wohin der Bot seinen **Translocator** schießen soll. In unserem Fall also oben auf die Galerie. Setze also einen davon in die Mitte der Galerie in der Basis.
Zu jedem **TranslocDest** gehören (mindestens) zwei **LiftExit.** Setze einen davon dort, von wo der Bot seinen **Translocator** abschießen soll. Also unten, vor der **FlagBase.** Setze den zweiten Liftexit sozusagen als Ausgang oben, direkt hinter den **TranslocDest.**



Jetzt mußt Du noch den **LiftTag** des **TranslocDest** auf einen eigenen Namen setzen. Also z.B. "trans1". Dieser Name darf nur von diesem einen **TranslocDest** verwendet werden.

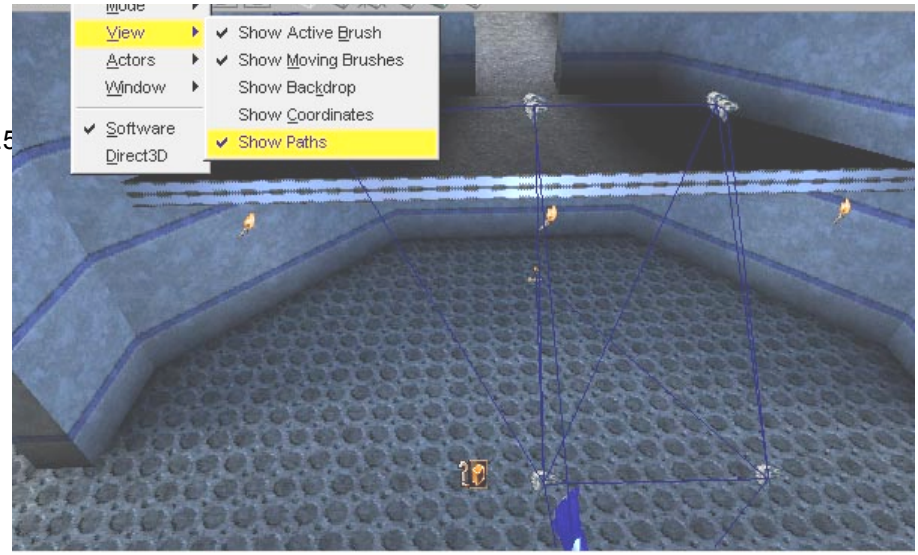


Die beiden **LiftExit** bekommen den selben **LiftTag.**

Baue einen zweiten **TranslocDest** mit den beiden **LiftExits** in die andere Basis ein und gib ihm den **LiftTag** "trans2"

Du kannst auch noch einen dritten **TranslocDest** genau in die Mitte auf den Steg setzen, der durch den zentralen Raum führt. Diesmal kannst Du einen dritten **LiftExit** auf den Boden setzen, so daß man von beiden Seiten auf den Steg springen kann.
Tip: Der **TranslocStart** ist eine Unterklasse vom **LiftExit**. Er wird an Stelle des **LiftExits** benutzt, wenn der Bot diesen **LiftExit** nur erreichen kann, indem er wiederum den **Translocator** benutzt.

13.5



13.5.8. Show Paths

Wenn Du mit der RM-Taste auf die Fensterleiste klickst und im nachfolgenden Menü **View > Show Paths** aktivierst, dann kannst Du nach dem Rebuild die Pfade überprüfen.
Es herrscht die Meinung, das blaue Linien für das blaue Team und rote Linien für das rote Team sind. **Das ist falsch**. Die blauen Linien zeigen bessere Wege an als die roten.

13.6. Der Rest des Pathings

13.6.1. Waffen & Munition

Zuletzt setzen wir die **Pickups** und die Waffen sowie die Munition. Ein paar allgemeine Tips für Level normaler Größe:
Setze die Waffen nur in der Nähe der Basen und Direkt ins Zentrum. Sei nicht verschwenderisch mit Waffen. Setze von jedem Typ maximal eine für jedes Team. Bedenke, daß jeder Bot oder Spieler die Waffe im Falle seines Ablebens einfach liegen läßt.
Setze neben jede Waffe zwei Einheiten Munition. Setze zusätzliche Munition mit Bedacht. maximal 6-8 Einheiten pro Waffe.
Setze die "normalen" Waffen, dazu zähle ich **MiniGun**, **PulseGun**, **Ripper**, **ShockRifle** und **BioRifle** so, daß man sie schnell findet und die "mächtigeren" Waffen **FlakCannon** und **RocketLauncher** etwas außerhalb des Weges, das man nicht sofort über sie stolpert. Lege niemals Waffen Direkt neben den **Playerstart**
Studiere mal einen Level wie DM-Tempest oder CTF-Coret im Editor um einen Eindruck zu bekommen, welche Menge von Items man wo setzen sollte. Du wirst überrascht sein, wie wenige das sind.

Unser Level ist sehr klein. Setze also nur ein paar Waffen.

- **PulseGun** und **MiniGun** jeweils unter die Galerie in der Basis
- Nur eine **ShockRifle** und einen **RocketLauncher** in den zentralen Raum gegenüberliegend unter den Steg.
- je eine **FlakCannon** oben in den Gang, damit sich der Sprung lohnt

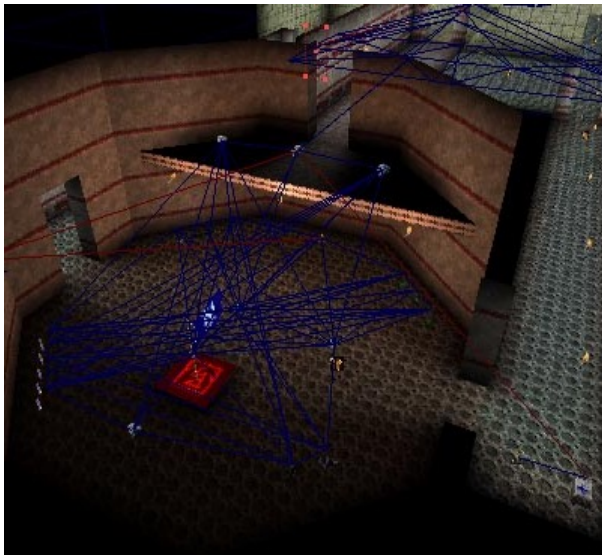
Sniper, **Redeemer** und die anderen lassen wir weg.

13.6.2. Health & Armor

Für unsere Zwecke legen wir folgende **Pickups** jeweils in die Basis

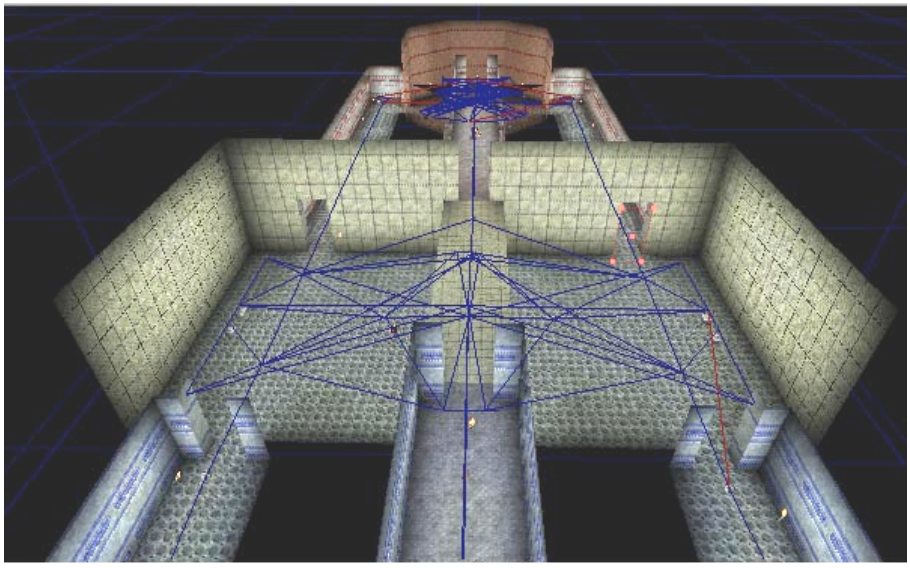
- **JumpBoots** und **ThighPads** neben die FlagBase an die Wand
- 5 **HealthVials** auf die andere Seite neben die **FlagBase**
- je eine **MedBox** in jeden Gang neben den **Playerstart**.
- je zwei **MedBoxes** rechts und links in den zentralen Raum und in der Mitte die Armor

So sieht nach dem **Rebuild** die rote Basis aus. Dies ist auch der Beweis, daß das Gerücht von den roten Pfaden für das rote Team falsch ist.



13.6.3. PathNodes

Zuletzt müssen wir den Rest des Levels mit **PathNodes** auffüllen. Zwar gibt's es bei den **PathNodes** auch **Properties**. Ich habe aber noch keinen nennenswerten Effekt im CTF-Game bemerkt. Deshalb laß einfach die Einstellungen wie sie sind.
Setze zumindest auf beiden Seiten der Türen je einen **PathNodes** und in die Mitte der Gänge welche im Abstand von etwa 500 Einheiten. Setze noch jeweils im zentralen Raum einige **PathNodes** in die Mitte. Setze keine **PathNodes** da, wo schon andere **NavigationPoints** oder **Pickups** sind. Du kannst einen "vollen" **Rebuild** machen und den Level speichern. Schau Dir noch mal die Pfade an und korrigiere notfalls einige **PathNodes** und **NavigationPoints**.
Info: Bots müssen **PathNodes** sehen, um zu ihnen zu kommen. Haben sie einen erreicht, verwenden sie die in ihm gespeicherte Tabelle in dem steht welchen Punkt sie als nächstes erreichen können.



13.7. Test

Wenn Du gleich den Level startest, dann sollten die Bots etwas lebhafter sein als vorhin. Probier es aus und komm dann noch mal hierher zurück.
Und? - Bei mir gab's ein paar Probleme. Erst haben die Bots die beiden Außengänge aus der Basis ignoriert. Also habe ich einige **PathNodes** dazugefügt. Danach wollten sie nicht mehr so gerne auf die Galerie springen. Deshalb solltest Du das **PulseGun** und die **ThighPads** von unten auf die Galerie befördern. - **Rebuild** nicht vergessen.
Lege außerdem neben die **AlternatePaths** in den unteren Gängen etwas Munition. Jeweils zweimal für das **PulseGun** und das **MiniGun**. Die Bots, die im Zentrum geboren werden, laufen in den Gang zum **AlternatePath** und dann sofort zurück. Jetzt haben sie einen Grund, dies zu tun.
Zuletzt haben die Bots Probleme auf den Steg im Mittelraum zu gelangen. Sie schießen den **TranIocator** gegen den Steg. Setze deshalb die unteren **LiftExits** etwas weiter vom Steg weg.
In der Basis hingegen ist der **LiftExit** für den **Jumpspot** zu weit weg von der Galerie. Die Bots treffen die Galerie nicht. Deshalb schiebe ich diesen **LiftExit** weiter an die Galerie. Sollte das nichts nützen, benutze den **bAlwaysAccel** Tag.

13.7.1. Spectator Modus

Im **Spectator** MoDus lassen sich die Bots am besten beobachten. Diesen aktivierst Du im Hauptmenü des Spiels unter **Options > Player Setup**. Da ist unten ein Feld zum abhaken.

Einer der Gegner ist über den oberen Steg gekommen und springt zur Unterstützung des Fahnenträgers in meine Basis. Man sieht: die Fahne ist gerade unterwegs.

Die Bots benutzen alle drei Wege.

Da die **Priority** des **DefensePoints** auf 8 gesetzt wurde wird dieser Punkt fast immer besetzt.

Du solltest den Level mit verschiedenen Bots und in verschiedenen Skill-Stufen testen. Fange zum Beispiel mit 4 (2 macht nicht so viel Sinn) auf **Novice** eingestellten Bots an und gehe bis 10 Bots in **Godlike**, was wahrscheinlich ein ganz schönes Gemetzel gibt.

Du solltest den Level mit verschiedenen Bots und in verschiedenen Skill-Stufen testen. Fange zum Beispiel mit 4 (2 macht nicht so viel Sinn) auf **Novice** eingestellten Bots an und gehe bis 10 Bots in **Godlike**, was wahrscheinlich ein ganz schönes Gemetzel gibt.

Du kannst die einzelnen Bots mit der Maustaste "durchschalten". Ich beobachte immer jeden bis er 2 oder 3 mal gestorben ist und gehe dann zum nächsten bis alle durch sind.

13.7.2. Orders

Wenn Du während des Spiels die Taste "V" drückst erscheint ein Menü in dem Du den Bots Orders, also Aufträge erteilen kannst.

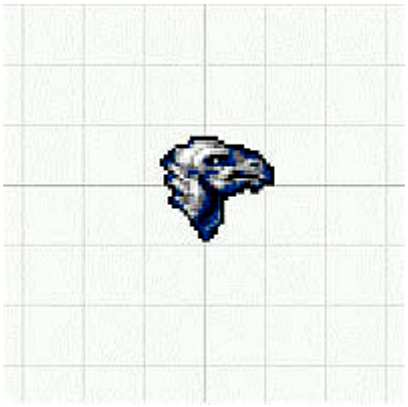
Schicke mal alle Bots die Fahne holen, bleibe selbst in der Base und warte darauf von wo die Gegner kommen. Probiere einfach etwas mit den Befehlen und teste möglichst viele Spielsituation. Schließlich soll der spätere User dieser Map keine unangenehmen Überraschungen erleben.

14. RocketArenaUT (RA) Maps erstellen

Dieses Tutorial soll Dir Aufschluß geben, wie man eine RA Map mit ihren verschiedenen Eigenarten und Informationen, erstellen kann. Sinnvoll ist es natürlich, wenn Du bereits über diverse fundierte Kenntnisse des Editors verfügt.

14.1. Info / Arena Info

Als erstes solltest Du Dir vorhandene RA Maps anschauen um zu lernen wie man die Geometrie am besten umsetzt. Da der Mod Rocket **Jumps** bietet ist es Sinnvoll mehrere Ebenen auf unterschiedlicher Höhe zu bauen. Des weiteren werden bei RA die gesamten Arenen in einer Map untergebracht, was ich Persönlich als ein Manko bezeichne. Du solltest Dir im Voraus schon Gedanken machen, wieviel Arenen und in welcher Größenordnung Du baust.

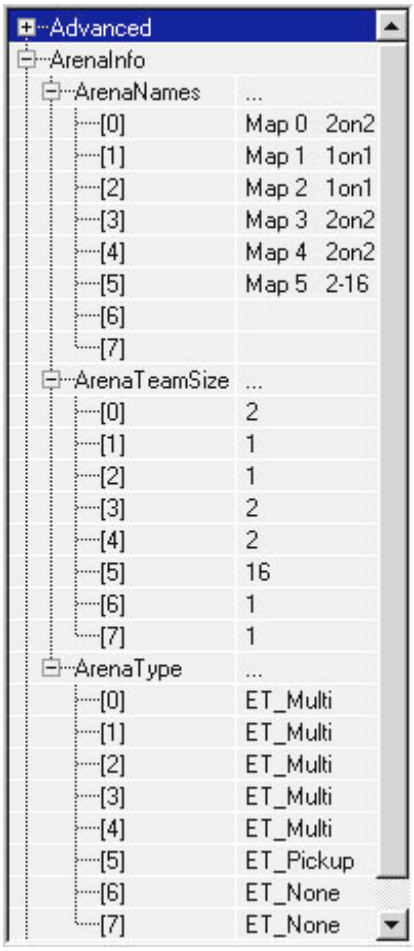


Kommen wir nun zu den eigentlichen Dingen die Du beim RA Mapping beachten mußt.

Gehe in den ActorClassBrowser und Lade die RocketArena Systemerweiterung „RocketArena.u“

Setzte eine Rocket Arena Info irgendwo in deine Map (ich geh mal davon aus das die Arenen schon gebaut sind), Du findest sie unter ActorClassBrowser

Öffne die Rocket Arena Info (siehe oben stehende Abbildung) und editiere



Öffne die Rocket Arena Info (siehe nebenstehende Abbildung) und editiere sie. Folgende Einstellungen sind vorzunehmen:

ArenaNames

Trage hier die Namen der einzelnen Arenen ein

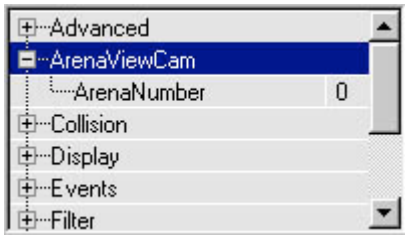
ArenaTeamSize

Trage hier die PlayerAnzahl ein
(1 steht für 1on1)
(2 steht für 2on2)
(16 steht für bis zu 16 Player)

ArenaType

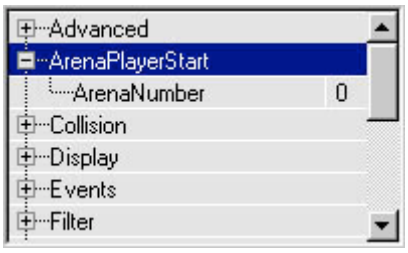
Trage hier den Gametype ET_Multi steht für 1on1 oder 2on2
ET_Pickup steht für die Große 16 PlayerMap

Jetzt mußt Du in jede Arena eine **ArenaCam** setzten
(zu finden unter ActorClassBrowser/Info/ArenaViewCam)

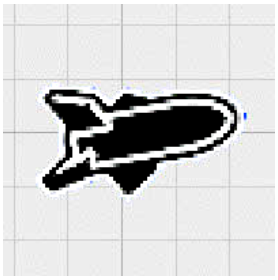


14.2. BotAI und PlayerStarts

14.2. BotAI und PlayerStarts



Du mußt jeder ArenaCam eine Nummer geben, welche Du in der ArenaInfo angegeben hast z.B.. 0 für dieArena 0=Map0 2on2



Playerstarts müssen - je nachdem in welcher Arena Du gerade eine setzt - ebenfalls eine Nummer bekommen und auch hier wieder die, die Du in der ArenaInfo angegeben hast also die gleiche Nummer wie die ArenaViewcam

Da bei RA die Waffen nicht eingesammelt werden müssen sondern man sie schon hat, müssen wir also die BOTS überlisten und sie sozusagen Austricksen.

Dafür sind die Tasty Pellets(zu finden unter "ActorClassBrowser / Inventory / Pickup / Ammo / TournamentAmmo / Tasty Pellet"). Setzte sie nur Gering und an Strategischen Stellen in Deiner Map.

15. Leseliste

Caillois, R: Man, Play and Games. Thames and Hudson London 1962

Darley, Andrew: Visual Digital Culture – Surface Play and Spectacle in New Media Genres.
Routledge London and New York 2000

Herz, J.C.: Joystick Nation. Little, Brown and Company Boston, New York, Toronto 1997

Huhtamo, Erkki: Encapsuled Bodies in Motion. In: Penny, S.:
Critical Issues in Electronic Media. New York State University Press New York 1996

Huizinga, J.: Homo Ludens: A Study of the Play Element in Culture.
Beacon Press Boston 1955

Rosenberg, Richard S.: The Social Impact of Computers.
Academic Press, Inc. San Diego, London, Boston 1977
(Chapter 6: Computers and Education)

Die Bücher von Huizinga und Caillois sind die klassischen Werke zur philosophischen Theorie der Spiele. Herz's Werk ist eine leicht zu lesende unterhaltsame Geschichte der elektronischen Spiele. Huhtamo und Darley konzentrieren sich auf die jüngsten elektronischen Entwicklungen - vor allem auch im Hinblick auf ihre mediale Vermittlung und Formierung.
Eine empfehlenswerte Zeitschrift stellt das EDGE Magizin aus London dar.

16. Glossar

Actor
Primär 'Akteur' wie etwa Mitspieler oder Monster, der Begriff wird aber oft für alle in der Szene animierbaren Objekte gebraucht, da diese technisch sehr unterschiedlich zur weitgehend unbeweglichen Szenearchitektur gehandhabt werden.

Avatar, Player
3D-Spielfigur eines am Spiel beteiligten Menschen. Selbst bei nicht vernetzten Spielen wird ein Avatar gebraucht, sobald es Spiegel im Spielfeld gibt und der Spieler sich sozusagen selbst sieht.

Binary Space Partitioning, Octree Partitioning
Das Grundproblem von Echtzeit-3D ist die Beschränkung der Berechnungen auf in der Szene sichtbare Details. Dabei eignet sich 'Binary Space Partitioning' (BSP) für Szenen in geschlossenen Räumen sehr gut. Dieses Verfahren nutzt vorhandene Decken und

Wände oder künstlich ermittelte Schnittebenen zur rekursiven Halbierung der gesamten Geometrie, um so eine Baumstruktur aufzubauen. Die Kameraposition läßt sich so einem Knoten dieses Baums zuordnen und alle benachbarten Flächen entsprechen benachbarten Kanten des Baums. Doom-Engines benutzen BSP nur für Linien in zweidimensionalen 'Bauplänen' der Szene, da es keine schrägen Wände gibt. Quake-Engines nutzen BSP mit Schnittebenen im Raum.
Für Außenszenen eignet sich eher 'Octree Partitioning', wo ein Baum der achten Ordnung entsteht, wobei eine Schnittebene für jede der drei Dimensionen zur Raumaufteilung von 'Räumen ohne Wände' genutzt wird.

Bone, Muscle & Skin
3D-Modelle von Akteuren setzen sich aus einer Hierarchie von Unterobjekten, etwa Torso, Oberarm, Unterarm, Hand und Finger, zusammen, die über Gelenke miteinander verbunden sind. Die Animation läßt sich wesentlich leichter berechnen, wenn man die sichtbare Hülle (Skin) eines jeden Unterobjekts um einen 'Knochen' (Bone) ergänzt, der Orientierung und Drehpunkt des Unterobjekts bestimmt. Mathematisch gesehen bilden Bones ein lokales Koordinatensystem, ausgedrückt durch eine 4 x 4-Transformationsmatrix zur affinen Abbildung. Die Weltkoordinaten eines 'Fingers' erhält man dann durch Matrizenmultiplikation aller Matrizen vom Torso bis zum Finger. 'Bessere' Animationsprogramme bilden auch noch anatomische Details wie das Anschwellen von Muskeln (Muscles) nach.

Brush
Brush ist der Name für eine Grundform der Szenearchitektur bei einer BSP-basierten Engine. Eine Brush muß konvex sein, kann aber sonst aus einer Vielzahl von Wänden bestehen, die unterschiedlich texturiert sein können. Raum ausfüllende 'Solid Brushes' können mit 'Cut Brushes' ausgehöhlt werden (siehe CSG). 'Hint Brushes' und 'Clip Brushes' erzeugen nicht sichtbare Hilfsflächen, welche dem BSP-Compiler die optimale Raumaufteilung erleichtern.
Darüber hinaus gibt es viele weitere spezialisierte Brush-Typen etwa für Fenster, Wasser und unpassierbare Grenzlinien.

Constructive Solid Geometrie (CSG)
Verfahren, das komplexe Geometrien durch 'Boolesche' Operationen wie Vereinigen, Schneiden und Subtraktion einfacher, konvexer Grundkörper wie Quader, Pyramiden, Kegeln oder Kugeln aufbaut siehe [5]. Dieses Verfahren erleichtert die Durchführung der BSP-Kompilierung, so daß die Architektur der Szene mittels 'Solid Brushes' und von ihnen abgezogenen 'Cut Brushes' aufgebaut und in einer MAP-Datei dargestellt wird. Editorprogramme für die Maps machen sich dieses Prinzip ebenfalls zunutze, da sich recht komfortabel damit arbeiten läßt und Manipulation einzelner Punkte oder Linien nur selten gebraucht wird.

Deathmatch
Zweikampf im Cyberspace. Eine Kopie des Spiels oder ein dediziertes Serverprogramm bildet die Szene, in der sich mehrere Spieler im Netz treffen und gegeneinander kämpfen. Neuerdings gibt es auch 'Arenen' und 'Tribünen' für weitgehend passive Zuschauer und für die meisten FPS-Spiele Ligen und Meisterschaften, Wetten werden abgeschlossen und wie bei Tennis oder Fußball heiße Diskussionen über die Vorzüge einzelner Spieler, Teams und deren Ausrüstung in Newsgroups und Chat Rooms geführt.

Entity
Für Programmierer und Level-Designer gleichermaßen zugängliche Datenstruktur. Vom Programmcode gesteuerte Objekte wie Waffen, Monster oder Effekte (SFX) können so vom Level-Designer mit den gewünschten Eigenschaften 'in Szene gesetzt' werden.

First Person Shooter (FPS)
Im Gegensatz zu Third Person Shootern (TPS) wie Tomb Raider sieht der Spieler seinen Avatar allenfalls im Spiegel oder mit Hilfe einer Überwachungskamera, weil die Hauptkamera den Augen des Spielers entspricht. Beide Kategorien versetzen den Spieler in eine virtuelle 3D-Welt, in der 'Lebensenergie', Waffen, Munition und bösartige Gegner irgendeiner Form existieren. Das Spiel ist eine Aneinanderreihung von 'Level' genannten Szenen mit steigendem Schwierigkeitsgrad. Das Spielziel ist, durch

Vernichten von oder Flucht vor möglichst vielen Gegnern einen möglichst hohen Level zu erreichen oder sich im 'Deathmatch' mit menschlichen Gegnern zu messen.

Filterung, MIP-Map-Texturen
Betrachtet man sich Texturen aus der Nähe, wird ihre Pixelstruktur sichtbar. Interpolation löst beim 'bilinearem Filtern' die Pixelstruktur durch Unschärfe auf. Ein besseres Ergebnis liefern MIP-Map-Texturen, die je nach Kamera-Abstand in unterschiedlichen Auflösungen benutzt werden. Erzeugt man noch feiner abgestufte Texturversionen durch Interpolation zwischen benachbarte MIP-Map-Stufen, spricht man von trilinearem Filtern. Aktuelle Grafikkarten unterstützen all diese Verfahren hardwareseitig.

Frame, Key Frame
Ein Frame ist ein einzelnes Vollbild. Die Leistung einer Engine wird mit 'Frames pro Sekunde' (fps) angegeben, also der Anzahl der einzelnen Vollbilder, die sie pro Sekunde zeichnen kann. Bei Animationen sind Key Frames' Bewegungsphasen zu definierten Zeitpunkten. Zwischenstände dieser Bewegungen für die konkret gerenderten Frames interpoliert die Engine.

Game Engine
Eine Game Engine ist eine Funktionen-Bibliothek, ein Framework oder ein komplettes Toolkit für Spiele einer bestimmten Kategorie wie etwa FPS. Damit ist eine Game Engine viel spezifischer und umfassender als etwa eine nur auf 3D-Grafik beschränkte Graphics Engine oder gar nur eine gänzlich unspezifische 3D-Render-Engine wie OpenGL oder Direct3D.

Geometrie Setup
Moderne Grafikkarten berechnen bei 3D-Grafik die endgültigen Farbwerte der Bildschirmpunkte innerhalb vorzugebender Polygone - meist Dreiecke - und erkennen dabei Bildteile, die aufgrund von Verdeckung nicht sichtbar sind. Der dafür verantwortliche Rasterizer benötigte bei den ersten Grafikchips noch umfangreiche Parametersätze, die der PC-Prozessor berechnen mußte. Die letzten beiden Chipgenerationen besitzen dagegen ein eigenes Geometrie-Setup und können die Parameter für den Rasterizer selbst berechnen. Sie verstehen daher das von der Game Engine und von Direct3D benutzte Gleitkommaformat direkt und können unmittelbar mit deren 3D-Koordinaten arbeiten.

Konvexe Polygone
Polygone oder Polyhedrons sind konvex, wenn keine zwischen beliebigen Punkten ihres Randes oder ihrer Hülle gezogene Linie das Objekt verläßt - wenn also keine Aushöhlungen oder 'Extremitäten' existieren. Grundformen wie Quader, Kegel oder Kugel sind konvex, ein offener Behälter wie eine Tasse oder ein Gebilde mit 'Zacken' wie ein Seestern ist es nicht. Brushes in BSP-Szenen respektive MAP-Dateien müssen konvex sein.

Level
Ein anderes Wort für Szene oder Welt; ältere Spiele behalten beim Verlassen eines Levels keine Informationen über dessen Zustand und verbieten so eine Rückkehr. Neuere Spiele verbinden mehrere Levels, zwischen denen man beliebig wechseln kann, zu einem 'Hub'.

Level Of Detail (LOD)
Bei Szenen mit 'Überblick' kann der Rechenaufwand für weit entfernte, aber detaillierte Szenedetails die Performance zunichte machen. Als Abhilfe kann man unterschiedlich detailgetreue Modelle für unterschiedliche Abstände benutzen, die von der Engine je nach Kameraabstand ausgewählt werden. Klassische BSP-Renderer unterstützen dieses Verfahren nicht, weshalb sie für Außenszenen kaum geeignet sind.

Lightmap
Vorausberechnung der Wirkung von Lichtquellen auf sichtbare Flächen (beziehungsweise Texturen) erlaubt die Nutzung sehr rechenintensiver aber realitätsnaher und stimmungsvolle Beleuchtungsmodelle wie Radiosity. Die bei dieser Berechnung vom 'LIGHTS'-Compiler erstellten Tabellen heißen Lightmaps und enthalten

für jedes Polygon die Wirkung benachbarter Lichtquellen auf 16 x 16 Pixel große Bereiche der MIP-Map-Textur.

Maps und MAP-Dateien
Bei älteren Spielen bis zu den Doom-Engines waren Maps schlicht zweidimensionale 'Baupläne' der Level-Architektur. Seit Quake sind Maps geschlossene, mittels CSG-Operationen aus 'Brushes' zusammengesetzte 3D-Modelle, in denen darüber hinaus 'Entities' mit definierten Eigenschaften plaziert werden. Die so erstellten standardisierten MAP-Dateien werden in drei Schritten mit einem BSP-Compiler, einem VIS-Compiler und einem LIGHTS-Compiler übersetzt, wobei eine durch die Engine direkt ausführbare BSP-Datei entsteht.

Polyhedron
Ein Polyhedron ist ein aus Polygonen zusammengesetzter Körper. Da Polygone wiederum aus geraden Linien bestehen, ist dies ein eckiger beziehungsweise kantiger Körper. Diese störenden Kanten kann man entweder mit einer enorm hohen Polygonzahl oder interpolierten Spline-Kurven bekämpfen. Um die Datenmenge klein zu halten, wird bei neueren Echtzeit-3D-Spielen mit nur durch wenige Kontrollpunkte bestimmte, aber durch Interpolation wunderbar abgerundete Spline-Kurven gearbeitet.

Potentially Visible Set (PVS)
Der BSP-Baum liefert zwar für eine gegebene Kameraposition schnell die umgebenden Wände, aber noch keine Information was an einer Position konkret sichtbar ist und was nicht. Deshalb berechnet ein spezieller Compiler nach Ermittlung des BSP-Baums für jedes Blatt eine Tabelle, die das 'Potentially Visible Set' enthält. Konkret enthält diese Tabelle für jedes Blatt ein Bitset mit einem Flag für jedes andere Blatt der Szene, das dessen Sichtbarkeit von dieser Stelle aus kennzeichnet.

Prefab
Komplexere, im MAP-Dateiformat vorliegende Szeneobjekte, wie etwa vorgefertigtes Mobiliar, Bäume, Brücken und Portale

Quaternion
Ein aus vier Elementen bestehender Vektor, der die Drehung um einen definierten Winkel einer definierten Achse beschreibt und so eine Platz und Rechenzeit sparende Alternative zu einer 4x4-Transformationsmatrix darstellt. In Key Frames wird Rotation meist mittels Quaternions definiert.

Radiosity
Beleuchtungsmodell, das neben reiner Strahlenoptik auch berücksichtigt, daß alle beleuchteten Flächen Lichtquellen sind und die diffuse Helligkeitsverteilung in Räumen nachbildet. Es ist so viel realistischer als 'Raytracing', das nur den Weg der Lichtstrahlen von der Quelle zum Auge berücksichtigt, ohne etwa deren Absorption zu beachten. Raytracing erzeugt beispielsweise unrealistische, hart abgegrenzte Schatten, während der Schattenwurf bei Radiosity natürlich wirkt. Der Nachteil von Radiosity ist ein extrem viel höherer Rechenaufwand als schon bei Raytracing. Echtzeit 3D benutzt daher den 'Trick' voraus berechneter Lightmaps und Interpolation zur Echtzeit um Radiosity nachzuahmen.

Raycasting
Beim Raycasting werden durch jedes Pixel des Bildschirmfensters verlaufende 'Sehstrahlen' in die Szene geschickt, bis das nächste, durch nichts verdeckte Objekt getroffen wird. Aus dem Abstand ergibt sich der Abbildungsmaßstab für die Textur der getroffenen Fläche, und Interpolation bestimmt letztlich die benötigten Texturpixel beziehungsweise die darzustellende Pixelfarbe. Dies ist das älteste Echtzeit-3D-Verfahren der ersten Wolfenstein-Engines (inklusive des Originals), die es ausschließlich für orthogonale Wände nutzten.

SFX
Abkürzung für 'Special Effects'; Spezialeffekte wie im Kino: Feuer, Blitze, Wasserfluten, in Levels von 3D-Spielen mittels Entities realisiert.

Skybox, Backdrop

Ein Backdrop ist ein Hintergrund-Bitmap der Szene. In 3D-Welten sind Skyboxes die sechs Flächen eines die gesamte Szene ‘im Unendlichen’ umgebenden Würfels. Geeignete Projektion erlaubt ein realistisches Rundumpanorama.
Die Animation der Skybox gibt die Illusion vorbeiziehender Wolken.

Sprite
2D-Bitmap, in 3D-Szenen auch manchmal ‘Billboard’ genannt. Besitzt meist transparente Teile und hat entweder verschiedene Ansichten oder dreht sich automatisch in Richtung Kamera. Animierte Sprites haben Bitmaps für eine Folge von Key Frames.

Terrain Rendering
Offenes Gelände wird meist durch Höhenliniendateien beschrieben. Das kann auch eine Bitmap sein, deren Pixelhelligkeit in Höhe umgesetzt wird. Große Terrains können nicht ‘auf einen Schlag’ geladen werden, weil sie extrem viel Speicherplatz erfordern. Mechanismen wie ‘Level of Detail’ müssen zu detailreiche Darstellung weit entfernter Objekte verhindern. BSP ist für Terrain Rendering total unbrauchbar, Octree-Partitioning ist besser geeignet. Alles in allem wird ein total anderer Grafik-Engine-Typ als für Innenräume benötigt.

Viewing Frustrum
Um den Rechenaufwand zu reduzieren, werden nur Objekte gerendert, welche sich mit einem Pyramidenstumpf schneiden, dessen Oberseite dem ‘Viewport’ also der als ‘Fenster in die Szene’ sichtbaren Fläche entspricht und deren Unterseite die Sichtweite begrenzt. Dieser Pyramidenstumpf heißt ‘Viewing Frustrum’. Bei Bewegung in die Szene gelangende Objekte tauchen leider aus dem ‘Nichts’ auf. Nebel oder die Aussicht blockierende Gegenstände müssen diesen Effekt verhindern.

VRML
Abkürzung für ‘Virtual Reality Modelling Language’; Programmiersprache, die ursprünglich für das Internet-Umfeld entwickelt wurde, um so beim ‘Surfen’ den Eindruck einer dreidimensionalen Welt zu erzeugen.

17. UNREAL Editor Shortcuts

Der Unreal Editor kann nicht nur über das sichtbare Menue dirigiert werden, sondern kann auch über Tastaturkürzel aufgefordert werden bestimmte Dinge zu tun. Diese Shortcuts sind nach einem simplen Shema aufgebaut, wenn auch nicht gerade immer vertraut. Was alles möglich ist seht ihr auf dieser Seite. Zuerst mal die einfache Tastaturbelegung:

Tastenbelegung und Tastaturkürzel

Taste:	Del	Lösche alle selektierten Aktoren
	F1	Hilfefunktion (funktioniert nur mit UnrealED_HelpFix)
	F4	zeige "Actor Properties"-Fenster an (z.Bsp für Aufzüge & Türen)
	F5	zeige "Texture Properties"-Fenster an (z.Bsp zum Texturpitch)
	F6	zeige "Level Properties"-Fenster an (z.Bsp. für's Previewpic)
	F7	rekompiliere alle editierten UnrealScripts (Achtung: Backup!)
	F8	zeige "Rebuilder"-Fenster an (zum Rendern des Levels)
	B	schalte zwischen "Brush"-Ansichten hin und her
	H	schalte zwischen Aktoren-Ansichten hin und her
	P	schaltet zwischen Animations-Ansichten hin und her
	1 2 3	3D-Bewegungsgeschwindigkeit (1=langsam / 3=schnell)

Allgemeine Shortcuts

Ctrl	A	Füge die "Brush" zum Level hinzu
	B	Lade eine gespeicherte "Brush"
	C	Kopiere den markierten Bereich / markiertes Element
	D	Entintersektiere die "Brush"
	E	aktuellen Level speichern unter "..."
	L	Speichere den aktuellen Level
	O	Lade einen Level
	P	Spiele den aktuellen Level
	N	Intersektiere die "Brush"
	R	Letzten Schritt rückgängig machen
	S	ziehe "Brush" von Welt ab
	V	Einfügen aus dem Zwischenspeicher
	W	Dupliziert den markierten Bereich
	X	Ausschneiden und im Zwischenspeicher halten
	Z	Letzten Schritt wieder herstellen

Selektionierungs Shortcuts

Shift	A	Selektiere alle Aktoren

B	Selektiere alle ähnlichen Texturen (Referenz ist die markierte "Textur")
C	Selektiere bündige anliegende "Brushes" (Referenz ist die markierte "Brush")
D	Dupliziere gewählte Aktoren
F	Selektiere sich berührende Flächen (Referenz ist die markierte "Fläche")
G	Selektiere alle ähnlichen Flächen (Referenz ist die markierte "Fläche")
I	Selektiere alle ähnlichen Gegenstände (Referenz ist der markierte "Gegenstand")
J	Selektiere alle sich plan berührende Flächen (hiermit selektiert man Bodenflächen)
L	gewählte Texturen horizontal ausrichten
M	Speichert die aktuelle Selektierung
N	aktuelle Selektierung aufheben
O	Selektiere alle geladenen Flächen (Referenz: Alle im Speicher befindliche Flächen)
P	Selektiere alle Flächen
N	Selektierung aufheben (Selbe Funktion wie Shift & Z)
R	Fügt die gespeicherte Selektierung ein (vgl. Shift & M)
T	Selektiere alle aktivierten Flächen (Referenz ist die aktivierte "Textur" im rechten ED-Fenster)
U	-+- keine Ahnung / ist ehh... Bullshit! -+-
W	Selektier sich stufig berührende Texturen (hiermit selektiert man Wandflächen)
X	Selektiere alle Flächen, wie in Vorauswahl. (Referenz ist die vorher gespeicherte Selektierung / vgl. Shift & M)
Y	Selektiere sich berührende schräge Flächen (hiermit selektiert man gekippte Flächen wie Dächer & Felswände)
Z	Selektierung aufheben (Selbe Funktion wie Shift & N)