Gothic

E-3: Technical

-

Letzte Änderung: 25.06.99 18:03
Druckdatum: 03.11.98 08:58
Autor: Stefan Nyul

-

Inhaltsverzeichnis

1.

Grafik-Engine

1.1 Figuren-Engine

1.1.1 Struktur

Die Figuren werden aus einzelnen Körperteil-Meshes zusammengesetzt:

- Kopf
- Hals
- Oberkörper
- Unterleib
- 2 Oberarme
- 2 Unterarme
- 2 Hände
- 2 Oberschenkel
- 2 Unterschenkel
- 2 Füße

Jedes dieser 16 Limbs ist abgespeichert als ein festes Drahtgittermodell.

Dazu kommen noch die Waffen, die die Figur bei sich trägt. Diese werden auf dem Rücken der Figur, bzw. an ihrem „Gürtel“ befestigt. Die Anzahl und Art der dargestellten Waffen ist abhängig vom Spielverlauf und muß während des Spiels dynamisch angepaßt werden. Für jede Waffenkategorie gibt es eine festgelegte Position in der Hierarchie, die in jeder Animationsdatei mit abgelegt wird. Da allerdings maximal eine Fern- und eine Nahkampf-Ausstattung gleichzeitig möglich ist, ergeben sich nur zehn Kombinationsmöglichkeiten K0 bis K9.

Waffe	**Trageort**	**Parent-Limb**	**K 0**	**K 1**	**K 2**	**K 3**	**K 4**	**K 5**	**K 6**	**K 7**	**K 8**	**K 9**
1H-Nahkampfwaffe	linke Hüfte	Unterleib		X	X	X	X					
2H-Nahkampfwaffe	Rücken	Oberkörper						X	X			
Schild	Rücken	Oberkörper			X	X				X	X	
Fernkampfwaffe	Rücken	Oberkörper				X	X		X		X	X

Jedes Limb muß nur einmal als Mesh abgespeichert werden und kann für jeden Frame einer Animation benutzt werden. Somit wird eine angestrebte Vielzahl von Bewegungsabläufen und Animationsschritten bei vergleichsweise geringem Speicheraufwand ermöglicht.

1.1.2 Getrennt animierte GAM’s

Die Gelenk-Animations-Modelle (GAM) sind zusätzlich in mehrere Gruppen unterteilt. Das modulare System ist auch hier sinnvoll um vielfältige und flexible Animationen mit geringem Speicherbedarf zu ermöglichen. Wenn die Figur etwas trägt, dann muß nur die Animation der Arme ausgetauscht werden. Dabei muß beachtet werden, daß kombinierbare GAMs die gleiche Anzahl von Frames haben müssen.

Die GAMs sind in 4 GAM-Gruppen unterteilt:

- Beine und Unterkörper
- Oberkörper und Kopf
- linker Arm
- rechter Arm

1.1.3 Echtzeitmodifikation der Animationen

Für einige Animationen ist eine flexible Anpassung an das Spielgeschehen nötig. Zum Beispiel für das Autoaiming der Fernkampfwaffen, die (in gewissen Grenzen) automatisch ein anvisiertes Ziel verfolgen müssen. Dazu müssen die erlaubten Rotationsgrenzen für die einzelnen Gelenke vorher festgelegt werden.

1.1.4 Flexible Texturierung

Jedes Limb-Mesh soll mit mehreren unterschiedlichen Texturen belegt werden um verschiedene Kleidung bzw. Gesichter darzustellen. Die Gesichtstexturen sollten mit höherer Auflösung abgelegt werden als andere Texturen da hier eine grobe Texturierung wesentlich störender ist, als an anderen Körperstellen.

1.1.5 Zerstörungsstufen

Alle zerstörbaren Objekte im Spiel werden in verschiedenen Zerstörungsstufen dargestellt. Für häufig benutzte Gegenstände wie Waffen und Rüstungen gibt es vier Zerstörungsstufen. Drei davon werden durch unterschiedliche Texturierung dargestellt. In der vierten Zerstörungsstufe, sobald das Objekt völlig unbrauchbar geworden ist, wird es durch ein anderes Mesh ersetzt.

Für manche Objekte wie z.B. Möbel können evtl. auch weniger Zerstörungsstufen ausreichen.

1.1.6 Austauschbare Meshes

Einige Rüstungstypen erfordern zusätzlich zur Texturierung eine andere Meshform. Deshalb gibt es für jedes Limb-Mesh mehrere Variationen. Durch die Kombinierbarkeit der Limbs können verschiedene Rüstungstypen, Kopfbedeckungen und Bewaffnungen in jeder Kombination dargestellt werden.

Für die Köpfe stellt sich die Frage, ob es sinnvoll ist, Haare und alle Arten von Kopfbedeckungen als separate Meshes an den Kopf anzuhängen oder ob jedesmal der komplette Kopf ausgewechselt wird, sobald die Figur einen Helm aufsetzt.

Körperteil, Ausrüstung	**Meshvarianten**
Haare, Kopfbedeckungen einzeln???	
Kopf	Visierhelm, Kettenkappe, Kapuze, Hut, lange Haare, kurze Haare, Glatze
Oberkörper	Platemail, Chainmail, Leder
Unterkörper	enge Hose, Pumphose, Röckchen
Oberarm	Platemail, Chainmail, Leder
Unterarm	Platemail, Chainmail, Leder, mit Ärmelaufschlag, nackt
Hand	offen, geschlossen, hält sich fest
Oberschenkel	nackt, Hose
Unterschenkel	nackt, Hosensaum, Stiefel
Fuß	nackt, Schuh (Stiefel)
Waffen	Dolch, 1H-Schwert, 2H-Schwert, 1H-Axt, 2H-Axt, Bogen, Armbrust

Die genaue Anzahl und Art der benötigten Meshvarianten kann erst nach der endgültigen Festlegung des Figurendesigns erfolgen. Es wird angestrebt, die Anzahl der Meshvarianten so gering wie möglich zu halten.

1.1.6.1 Verschiedene Detailstufen (LOD)

Alle benutzten Limbs müssen in mehreren verschiedenen Detailstufen vorliegen. Figuren, die weit entfernt sind werden mit der niedrigsten Detailstufe dargestellt um Rechenzeit zu sparen. Je größer eine Figur zu sehen ist, desto höhere Detailstufen werden vom Programm verwendet. Die Anzahl der LODs sollte nicht für alle Objekte gleich sein, da es nur bei sehr detaillierten Objekten sinnvoll ist, mehrere LODs zu benutzen.

1.1.7 Animationen

Hier nun eine Aufstellung der Animationen die für alle humanoiden NSC's und die SC's benötigt werden. Die Animationen für andere NSC's (Monster) müssen für jede Spezies separat angefertigt werden, sind aber auch bei Weitem nicht so vielfältig.

Dies ist nur eine vorläufige Aufstellung der geplanten Animationen. Hier wird es voraussichtlich noch einige Änderungen geben:

Animation	**m. W.**	**o. W.**	**Frames (ca.)**	**Anmerkung**
Bewegung				
Stehen (automatisch umsehen etc..)	x	x		Leichter Ausfallschritt, Oberkörper bewegt sich etwas
Reden (gestikulieren)				
Gehen	x	x		
Rennen	x	x		
Umdrehen	x	x		
seitlich gehen (Strafe)	x	x		
Rückwärts gehen	x	x		langsamer als vorwärts
Abhang runterrutschen	x	x		wie TombRaider
Springen im Stand	x	x		
Springen nach vorn	x	x		
klettern nach oben		x		an Textur??
klettern seitlich		x		„
an Vorsprung hochziehen		x		
an Vorsprung herunterlassen		x		
Leiter Hoch/Runterklettern				
Treppe steigen	x	x		
Schwimmen		x		schwimmen vorwärts (seitlich + rückenschw.???)
kurz untergehen		x		wenn kein Talent Schwimmen oder zu schwer
Tauchen		x		
In die Hocke gehen	x	x		
aus Hocke auf den Boden legen, aufst.		x		
Setzen / Aufstehen - Schemel , Bett				
Hinlegen / Aufstehen - Bett				
Inventory Objekte				
Aufheben/Ablegen - Boden		x		(geht in die Knie) und greift auf den Boden
Aufheben/Ablegen - Tisch, Podest				
Werfen				
Geben/Nehmen - SC, NSC		x		rechte Hand nach vorne und wieder zurück

		x		
Tragbare Objekte				
Aufheben/Ablegen - Boden		x		geht in die Knie bis Objekt Bodenkontakt hat
Aufheben/Ablegen - Tisch, Podest				
Werfen		x		
Truhe öffnen /knacken				
Große Objekte				
Schieben		x		
Ziehen		x		
Türschloß knacken		x		in die Hocke und in Schloßhöhe „rumfummeln“
Tür öffnen, Tür schließen		x		Hand nach vorne in Griffhöhe
In Faß verstecken		x		
Aussteigen aus Faß		x		
Schmerzen durch Wahnsinn?	x	x		krümmen (und Hände an den Kopf?)
Schmerzen durch Vergiftung?	x	x		krümmen
Nahkampf				
Kampfhaltung				Ausfallschritt, Oberkörper leicht gebeugt
Faustschlag		x		
Wegschubsen	?	x		
1H-Waffe ziehen/wegstecken	x			
2H-Waffe ziehen/wegstecken	x			
Püntenstich Dolch	x			
Meuchelattacke Dolch	x			
	x			
Püntenschlag 1H (Axt, Keule, Sch.)	x			
Standardschlag 1H (Axt, Keule, Sch.)	x			
Schlag nach Links 1H	x			
Schlag nach Rechts 1H	x			
Spezialstich 1H (Schwert)	x			
Spezialschlag 1H ?	?			
Püntenschlag 2H (Axt, Keule, Sch.)	x			
Standardschlag 2H (Axt, Keule, Sch.)	x			
Schlag nach Links 2H	x			
Schlag nach Rechts 2H	x			
Spezialschlag 2H (Berserkerschlag)	x			
Spezialschlag 2H	x			
Parade Schild	x			
Parade 1H Waffen	x			
Parade 2H-Waffen?	?			
getroffen werden (vorne)	x	x		
getroffen werden (hinten)	x	x		
stürzen vorwärts	x	x		
stürzen rückwärts	x	x		
aufstehen vorwärts	x	x		
aufstehen rückwärts	x	x		
Sterben Nahkampf	x	x		
Sterben Fernkampf	x	x		
Krit. Treffer Nahkampf (Kopf ab)	x	x		
Krit. Treffer Fernk. (Pfeil im Auge)	x	x		
offensive Magie	x			
defensive Magie	x			

PSI-Magie ?	?			
Levitation	x	?		
Bogenschießen und Nachladen (Pünte)	x			
Bogenschießen und Nachl. (Talent)	x			
Armbrustschießen und Nachl. (Pünte)	x			
Armbrustschießen und Nachl. (Talent)	x			

1.1.8 Einbindung der Animationen in die MDS-/Skriptsprache

In einer MDS-Datei werden sämtliche zur Verfügung stehenden Animationen aufgeführt, auf „Low-Level-Ebene“ (Start-/Stopframe, Abspielrichtung, Dateiname,...) definiert und mit einem Namen versehen. Um sich nun einen Überblick darüber zu verschaffen, welche Angaben in einem MDS gemacht werden können müssen, und wie genau die Animationen in die Skriptsprache eingebunden werden, kann eine Kategorisierung der Animationen sehr hilfreich sein.

ALLE Animationen werden teilen sich in vier logische Bereiche auf, die jeweils leicht unterschiedlich gehandhabt werden müssen:

- Default-Animationen
- Random-Animationen
- Action-Animationen
- Skript-Animationen

1.1.8.1 Default-Animationen

Immer wenn KEINE der anderen drei Animationsarten für ein SC/NSC/Monster abgespielt wird, wird die Default-Anim benutzt. Dies hat den Zweck, nie eine Figur wie zur Salzsäule erstarrt dastehen zu lassen. Selbst bei Verwendung von Zufallsanims, die vielleicht alle 5 oder 10 Sekunden gestartet werden, muß der Zeitraum dawischen mit Bewegung überbrückt werden. Dies hat ganz nebenbei bemerkt den technischen Effekt, daß stets **ALLE** sichtbaren SCs/NSCs/Monster **IMMER** animiert werden müssen.

Eine Default-Anim wird ständig geloopt, daß heißt sie muß extrem dezent sein und darf nicht rythmisch erscheinen. Typische Default-Anims sind:

- Brustkorb durch Atmung leicht heben und senken (Arythmik z.B. durch kurz gehoben und länger gesunken)
- Minimale Links-/Rechtsdrehung des Oberkörpers
- Minimales Baumeln lassen der Arme

Es gibt verschiedene Zustände einer humanoiden Figur, die jeweils eine unterschiedliche Default-Anim benötigt (allerdings immer nur EINE pro Zustand). Diese Zustände sind:

Ruhezustand	Vorschlag für Namen im MDS	Beispiel
Stehen	DEFAULT_STAND	leichte Atmung, leichte Oberkörperdrehung, leichtes Armbaumeln
Kampfhaltung	DEFAULT_FIST	schwerere Atmung, boxertypisches Hin- und Herwiegen des Oberkörpers
	DEFAULT_1H_UNSKILLED	schwerere Atmung, Waffe minimal bewegen

	DEFAULT_1H_SKILLED	
	DEFAULT_2H_UNSKILLED	
	DEFAULT_2H_SKILLED	
	DEFAULT_BOW_UNSKILLED	
	DEFAULT_BOW_SKILLED	
	DEFAULT_CROSSBOW_UNSKILLED	
	DEFAULT_CROSSBOW_SKILLED	
Klettern	DEFAULT_CLIMB	
Schwimmen	DEFAULT_SWIM	Schwimmbewegungen auf der Stelle
Tauchen	DEFAULT_DIVE	korrigierende Ruderbewegung unter Wasser
Sitzen	DEFAULT_SIT	
Schlafen	DEFAULT_SLEEP	

Die Namen im MDS beginnen mit „DEFAULT_“ und werden gefolgt von Zustand. Sollte das Programm einen erwarteten Namen nicht finden (falsch getippt, noch nicht eingebunden) spielt es keine Default-Anim ab und gibt im Debug-Modus eine Warnung aus. Dieses Verhalten sollte für das Suchen und Nicht-Finden für ALLE Animationen gelten.

1.1.8.2 Random-Animationen

Um die Spielwelt mit mehr Leben zu erfüllen, werden die Figuren ab und zu Bewegungen durchführen, die keinen speziellen Zweck haben. Typisch:

- Sich kurz umsehen
- Sich am Kopf/Hintern kratzen
- Gewichtsverlagerung von einem Bein aufs andere und zurück
- Kreisende Bewegung mit dem Schwert
- Faust in Handinnenfläche schlagen
- Flache Schwertklinge in Handinnenfläche schlagen

Für Random-Anims gibt es einige Dinge zu beachten:

Trigger: Random-Anims werden nur abgespielt, wenn keine Action- und keine Skript-Anims abgespielt werden. Wenn man so will sind sie, ebenso wie die Default-Anims, nur ein Pausenfüller. Allerdings werden sie nicht kontinuierlich geloopt, sonder ab und zu, nach einem Zufallsmuster eingestreut. Sie überlagern dann die Default-Anim.

Ruhezustand: Ebenso wie Default-Anims muß für Random-Anims die Art des „die Spielfigur macht gerade nichts“, also der Ruhezustand, unterschieden werden (siehe Tabellenspalte Ruhezustand). Steht die Figur normal herum, wird sie sich am Hinter kratzen, während sie nur mit gezogener Zweihandwaffe eine Wirbelnde Schwertbewegung macht. Beides geht im jeweils anderen Ruhezustand nicht.

Vielfalt: Für jeden Ruhezustand muß es die Möglichkeit (aber nicht die Notwendigkeit) geben, mehrere Randam-Anims zu unterstützen. So Wird im Ruhezustand „Normal Stehen“ zufällig zwischen Kopf-Kratzen, Umsehen und Standbeinwechseln ausgewählt. (Oh je, der Animationsname im MDS wird zusehends länger, so müßte er z.B. für die erste Zufallsanim bei der talentierten Bogengrundstellung lauten: RANDOM_BOW_SKILLED_0)

Wahrscheinlichkeit: Für das Feintuning der Random-Anims ist nötig, Einfluß auf die Zufallswerte der einzelnen Anims zu haben. Wie genau die zufällige Ermittlung der Anims programmintern auch immer gehandhabt wird, müssen zwei Dinge gesteuert werden können:

- Wahrscheinlichkeit pro Sekunde Ruhezustand, daß überhaupt irgend eine Random-Anim abgespielt wird. (→ Anwendung: Paranoiker, der sich ständig umsieht oder „Kretze-Kurt“, der sich ständig irgendwo kratzt)
- Falls eine Random-Anim abgespielt werden soll: Wie groß ist die Wahrscheinlichkeit für welche der X Random-Anims pro Ruhezustand. (→ am Hintern kratzen wird nicht so oft vorkommen, wie Standbein wechseln oder sich mal kurz umsehen)

MDS: Obwohl, die Angaben von solchen Zufallswerten zur Zeit im MDS noch nicht gemacht werden können, ist es doch wichtig, daß sie genau dort vorgenommen werden können. Die Aufteilung der Arbeitsbereiche in Animationen und Skripte (unterschiedliche Leute) sprechen gegen ein Aufnehmen der Zufallswerte in die Skripte.

1.1.8.3 Action-Animationen

Diese Animationen werden getriggert durch Aktionen von Figuren. Für SCs sind dies sämtliche Handlungen (Gehen, Rennen, Springen, Kämpfen,...), die durch Tastendruck ausgelöst werden. Alle zum jetzigen Zeitpunkt (Ende Februar 98) eingebundenen Animationen sind demnach Action-Anims.

Auch für Action-Anims gibt es im MDS feststehende Namen, die vom Programm erwartet werden.

1.1.8.4 Skript-Animationen

Last but not least gibt es eine Reihe von Animationen, die auf „direkten Befehl“ der Skriptsprache abgespielt werden können müssen. Diese Skript-Anims werden zwar auch in den MDS-Dateien aufgeführt und mit einem Namen versehen, aber dieser MDS-Name wird nicht von der Programmlogik direkt, sondernd durch ein Skriptbefehl wie z.B.: PlayAnim(npc, WAVE_HAND) aufgerufen.

Anwendungsfälle finden sich vor allem bei Gesten während Dialogen, aber auch für Mini-Cutscenes oder Event-Skripts:

- Dialog: Arme mit den Worten „Mit mir nicht“ verschränken
- Dialog: NSC kniet und hebt seine Hände schützend über seinen Kopf mit den Worten „Bitte verschon mich.“
- Dialog: „Du, komm her.“ (NSC zeigt mit Arm auf SC)
- Event: SC betritt Höhle worauf schlafender Drache aufsteht, die Flügel streckt und in Grundhaltung geht.

1.1.8.5 Basis- und Individual-Animationen

Dies sind **keine** weiteren Animationsarten. Vielmehr geht es um folgendes Problem:

Die meisten humanoiden weisen, bis auf wenige Ausnahmen, identische Bewegungsmuster auf, unterscheiden sich aber doch in einigen wenigen Animationen. Um nun zu vermeiden, jedem humanoiden eine unterschiedliche MDS-Datei zu verpassen, die aber größtenteils identisch ist, schlage ich ein System ähnlich dem Ableiten von Klassen in der objektorientierten Programmierung vor:

Für jeden NSC wird in der Skriptklasse eine **Basis-MDS-Datei** und eine **Individual-MDS-Datei** angegeben. Die Basis-MDS ist z.B. grundsätzlich HUMAN.MDS und enthält alle „normalen“ humanoiden Animationen während die Individual-MDS meist nur eine Hand voll Einträge mit den Abweichungen enthält. So könnte es z.B. eine HUMPEL.MDS geben, die lediglich eine Animationsangabe enthält. Diese eine Angabe ersetzt das „normal“ Gehen aus der Basis-MDS

durch die humpelnde Animation. Da in deser HUMPEL.MDS dagegen z.B. keine Schwimmanimation angegeben ist, wird die aus der Basis-MDS verwendet.

Mit diesem System können auf einfache Art und Weise viele „animations-optisch“ deutlich unterscheidbaren NSCs gebaut werden. Der Hauptvorteil ist folgender:

Nehmen wir an wir hätten 30 MDS-Dateien, die jeweils alle Anims enthalten sich aber immer nur in einigen wenigen unterscheiden. Wollte man nun z.B. die „Low-Level-Werte“ (Start-/Stopframe, Dateiname,...) der normalen Geh-Animation ändern, so müßte man durch alle 30 MDSs durchgehen, überprüfen ob in dieser Datei die normale WALK benutzt wird und dann die zu ändernden Parameter in jeder Datei ändern. Die Folge wäre der Verlust der Übersicht und das Einschleichen vieler kleiner Fehler.

1.1.9 Einbindung der Figuren-Darstellung in die Skriptsprache

Die Darstellung der Figuren, also die Meshes, die Texturen und die Animationsfiles sollte folgendermaßen in die Skripte integriert werden:

Eine Figur besteht aus dem „ungerüstet Mesh“ und dem Kopfmesh. Während im ungerüsteten Mesh die Texturen fix und unveränderlich enthalten sind, ist dies beim Kopfmesh nicht der Fall. Die Texturierung des Kopfmeshes wird unterteilt in Frisurtextur und Gesichtstextur. Beide Texturen zusammen bedecken das Kopfmesh zu 100% und müssen separat angegeben werden.

Dies hat seinen Grund im Aufsetzen und Abziehen eines Helmes. Will der Spieler z.B. einen Helm aufsetzen, so passiert folgendes:

- Austauschen des Kopfmeshes durch das „Gerüstet-Mesh“ des Helmes (geschieht durch das Proggie automatisch, das Helm-Mesh ist in C_Item des betreffenden Helmes angegeben).
- Texturieren des Helm-Meshes mit der Gesichtstextur der Figur (auch automatisch durch das Programm). Dabei ist zu beachten, daß das Gerüstet-Mesh des Helmes die Texturkoordinaten für die Gesichtstextur bereits enthalten hat.

Wird nun der Helm wieder abgezogen, so passiert folgendes:

- Austauschen des Helmmeshes durch das Kopfmesh der Figur.
- Texturieren des Kopfmeshes mit der Frisurtextur und der Gesichtstextur. Für beide (in C_NPC separat angegebenen) Texturen sind die Texturkoordinaten bereits im Kopfmesh enthalten.

Hier zwei Tabellen um das gesagte übersichtlich zusammenzufassen.

1.1.9.1 Figur

In der Instanz einer Figur (Klasse C_NPC) sind folgende Angaben vorhanden:

Element	enthält	Anmerkung	Dateiformat
Ungerüstetes Mesh	Mesh & Texturen		Verweis auf ASC-Datei (3DSmax ASCII-Format)
Kopfmesh	Mesh		Verweis auf ASC-Datei
Frisurtextur	Textur		TGA-Datei
Gesichtstextur	Textur	Gesichtsausdrücke (siehe 1.1.9.4)	TGA-Datei mit Nomenklatur (siehe 1.1.9.4 Gesichtsausdrücke)

1.1.9.2 Rüstungen und Helme

Für Rüstungen und Helme muß folgendes in der Skriptklasse C_Item aufgeführt werden:

Element	enthält	Anmerkung	Dateiformat
Gerüstetes Mesh	Mesh & Texturen	Mesh komplett oder partiell texturiert	Verweis auf ASC-Datei
Item-Mesh	Mesh & Texturen	für die Darstellung am Boden, in Kisten und im Inventory	Verweis auf 3DS-Datei

1.1.9.3 Animationen

Sollen Texturen auf Items, MOBs, NPCs oder Level animiert werden, so entspricht der Texturname einer bestimmten Nomenklatur:

< Texturname >_A<zweistellige Zahl>.TGA

Zur Zeit wird das „A“ noch weggelassen. Um jedoch eine Unterscheidung zu den Gesichtsausdrücken bzw. den Schadensstufen (siehe unten) durchführen zu können, ist die Kennzeichnung mit dem „A“ wichtig.

Hier einige Beispiele:

Beispiel-Dateiname	Erklärung
WATER_A0.TGA	Wasser → Animationsphase 0
FIREBLADE_A1_D2.TGA	Klinge eines Flammenschwertes → Animationsphase 1, Schadensstufe 2

1.1.9.4 Gesichtsausdrücke

Um den Figuren ein lebendiges Gesicht zu geben, ist folgender Automatismus geeignet:

Bei der Angabe der TGA-Datei für die Gesichtstextur in der Instanz von C_NPC wird eine Nomenklatur verwendet. Diese lautet:

<Texturname >.TGA

In der Skriptdatei wird also nur der „Texturname“ angegeben. Um später unterschiedliche Gesichtsausdrücke zur Verfügung zu haben, muß das Programm allerdings automatisch Gesichtstexturen der Nomenklatur

< Texturname >_G<zweistellige Zahl>.TGA

im selben Verzeichnis einlesen können. Die zweistellige Zahl ist eine fest vereinbarte und hardgecodete „Gesichtsausdruck-Nummer“ (GAN), die in bestimmten Situationen (halt hardgecodet) der Figur automatisch einen anderen Gesichtsausdruck (durch auswechseln der Gesichtstextur) gibt.

Hier einige Vorschläge für die Gesichtsausdruck-Nummer:

GAN	Gesichtsausdruck	Verwendung	Beispiel-Dateiname
0	normal	immer, wenn kein anderer Gesichtsausdruck gezeigt wird	SCARFACE.TGA
1	grinsen	Attitüde steigt, NSC wird beschenkt	SCARFACE_G01.TGA
2	grimmig	Attitüde sinkt, Figur schlägt zu	SCARFACE_G02.TGA
3	schmerzverzerrt	wenn die Figur getroffen wird	SCARFACE_G03.TGA

Ermittelt das Programm nun die Spielbedingungen für einen anderen Gesichtsausdruck, so wird automatisch versucht die entsprechende SCARFACE_XX.TGA zu laden bzw. zu aktivieren. Ist diese nicht vorhanden, so bleibt es beim normalen Gesichtsausdruck und damit bei der normalen Gesichtstextur. Ist die Grafik aber vorhanden, wird die Gesichtstextur für einen kurzen Moment (vielleicht 1 Sekunde) ausgewechselt und dann wieder zurückgewechselt.

Der große Vorteil dieses Verfahrens ist, daß immer nur die normale TGA-Datei also oben SCARFACE.TGA für die Gesichtstextur angegeben werden muß. Außerdem muß so nicht für jeden NSC jeder Gesichtsausdruck gepixelt werden.

1.1.10 Einbindung der Darstellung von Items in die Skriptsprache

Für Items (außer Rüstungen und Helmen, die ja beim „equippen“ das Mesh der Figur verändern) wird lediglich eine texturiertes 3DS-File als Visualisierung in der jeweiligen Instanz (C_Item bzw. C_Mob) angegeben.

1.1.10.1 Schadensstufen Item

Die Meshes von Items (bei Rüstungen und Helmen gilt dies sowohl für das „Item-Mesh“ als auch für das „Gerüstet-Mesh“) enthalten Verweise auf Texturen im TGA-Format. Um nun ein einfaches und effektives System für Schadensstufen zur Verfügung zu haben, entspricht (wie ja von Dir, Dieter schon gefordert) der TGA-Dateiname einer Nomenklatur. Diese ist

<Texturname >.TGA

falls keine Schadensstufen angezeigt werden sollen, oder aber

< Texturname >_D<einstellige Zahl>.TGA

falls Schäden am Gegenstand sich in der Textur wiederspiegeln sollen. Die <einstellige Zahl> ist eine feste Schadensstufe, welcher ein fester prozentualer Bereich der Strukturpunkte des Items zugeordnet wird (was ein Satz :-)

#	Schadensstufe	Strukturpunkte	Beispiel-Dateiname
0	ganz	100% - 81%	BLADE_D0.TGA
1	angeschlagen	80% - 31%	BLADE_D1.TGA
2	fast kaputt	30% - 1%	BLADE_D2.TGA
(3)	(kaputt)	(kaputt eben: Der Gegenstand verschwindet hinter einem Grafikeffekt im Nichts und wird aus dem Spiel genommen)	-

1.1.11 Einbindung der Darstellung von MOBs in die Skriptsprache

Bei MOBs verhält es sich ähnlich wie bei Items. Es existiert lediglich eine 3DS-Datei zur Visualisierung, die sowohl Mesh als auch Texturverweise/-koordinaten enthält.

Da ein zerstörtes MOB sich aber nicht in „Nichts“ auflösen soll (man stelle sich einen 3 Meter langen Tisch vor, welchen der Spieler mit einer Axt bearbeitet und der sich dann in einem „Puff“ auflöst) muß zur normalen Visualisierung auch eine „Kaputt-Visualisierung“ angegeben werden können (z.B.: C_Mob.visual_destroyed). Diese Visualisierung ist ebenfalls eine 3DS-Datei.

Visualisierung	Variablenname	Verwendung
normal	C_Mob.visual	Schadensstufe 0 bis 2
zerstört	C_Mob.visual_destroyed	Schadensstufe 3 (siehe oben)

1.1.11.1 Schadensstufen MOB

Die Schadensstufen bei MOBs werden, wie auch bei den Items, über die Texturen geregelt. Die einzige Abweichung besteht darin, daß ein zerstörtes MOB nicht aus dem Spiel entfernt wird, sondern durch ein Kaputt-Mesh ersetzt wird. Dieses zerstörte MOB muß dann auf OBJ_INDESTRUCTABLE gesetzt werden.

Wichtig ist, daß für das zerstörte MOB nicht wieder extra eine Skriptinstanz geschrieben werden muß (unnötiger Skriptaufwand).

1.2 Kamera-System

Das Kamerasystem verwendet im wesentlichen die 3rd-Person-Perspektive, wobei der Aufenthaltsort sowie die Blickrichtung des „Kamerasatelliten“ in Abhängigkeit von der Spielsituation „smart“ vorgenommen werden muß.

Ein erster Ansatz ist, das Kameraverhalten aufgrund unterschiedlicher Spielsituationen in verschiedene Modi einzuteilen.

Spielsituation	Kameramodus	Anmerkung
Normalmodus – Stehen/Gehen	CamModNormal	
Normalmodus – Rennen	CamModRun	
Dialogmodus	CamModDialog	solange ein NSC-Gesprächspartner im Dialogmodus zum SC ist
Inventory	CamModInventory	solange das Inv geöffnet ist
Nahkampf	CamModMelee	Während eine Nahkampfwaffe gezogen (auch Fäuste) UND Gegner im Fokus ist. Erfaßt der Fokus einen Gegner, während der SC läuft, so bleibt der alte Kameramodus solange bis der SC stehen bleibt (falls er den Gegner dann noch im Fokus hat)
Fernkampf	CamModRanged	Nur, wenn der Spieler eine Fernkampfwaffe gezogen UND gleichzeitig die ACTION-Taste gedrückt hält.
Magie	CamModMagic	sobald ein Zauberspruch ausgewählt wurde
Schwimmen	CamModSwim	auf der Stelle oder vorwärts/seitwärts
Tauchen	CamModDive	solange der SC untergetaucht ist
Springen nach vorne	CamModJump	solange der SC mit dem Vorwärts-Sprung in der Luft ist
Springen nach oben	CamModJumpUp	solange der SC mit dem Aufwärts-Sprung in der Luft ist (hält er sich an der Kante fest → CamModClimb, fällt er wieder
Klettern	CamModClimb	solange der SC an Kanten hängt oder sich hochzieht/runterläßt
Fallen	CamModFall	sobald einen Abgrund herunterfällt, oder sich von einer Kante losläßt; NICHT wenn er in der Abwärtsphase des Vorwärtssprungs ist.
Tod	CamModDeath	wenn der SC tot auf dem Boden (oder sonstwo) liegt
...	CamMod...	*(eventuell sind weitere Modi nötig oder einige der oberen unnötig, man wird sehen :-)*

Um nun nicht verschiedene Kamerasysteme programmieren zu müssen, wird im folgenden ein System vorgestellt, welches einen flexiblen Parametersatz verwendet. Die Umschaltung zwischen den Kameramodi wird dann lediglich durch Austauschen des Parametersatzes (unter Beibehaltung des Kamerasystems) vorgenommen.

Das Kamerasystem besteht aus folgenden Elementen:

- Der Kamerasatellit (CAM)
- Das Ziel (TARGET)
- Gültige Zone
- TARGET bewegt sich
- CAM-Kollision
- Durchbrechen der Line-Of-Sight
- Dynamik der Kamerabewegung
- Kamerafahrt zwischen Ist- und Soll-Koordinate
- Fixe Kameras

1.2.1 Kamerasatellit

Die CAM selbst ist ein VOB, daß, wie auch alle anderen VOBs im Spiel, nicht durch andere VOBs hindurchtauchen darf. Es muß also eine **Kollisionsabfrage** der CAM mit der statischen Welt sowie mit allen anderen VOBs erfolgen.

1.2.2 Ziel

Egal wo die CAM sich auch immer befindet, sie muß immer so ausgerichtet sein, daß sie direkt auf das Ziel (TARGET) „schaut“. Das TARGET wird im Parametersatz angegeben und kann eines der folgenden Dinge sein:

Ziel
eigener SC
anderer SC
NSC
MOB
Item

Um noch etwas mehr Flexibilität beim Feintuning der Kamera zur Verfügung zu haben, sollte noch ein **Offsetwinkel** für alle drei Drehachsen angegeben werden können. Soll zum Beispiel der Blick von der CAM nicht direkt zum Ziel (z.B. Hüfte des SCs) führen, sondern immer etwas darüber (zum Beispiel durch oder etwas über den Kopf der eigenen Spielfigur), so muß ein Winkel-Offset für das Drehen um die X-Achse angegeben werden können.

Ein weiterer Punkt: **Line-Of-Sight**. Der Weg zwischen CAM und Target darf weder von der Statischen Welt, noch von VOBs irgendeiner Art blockiert werden.

1.2.3 Gültige Zone

Für die Position der Kamera in Bezug zum Ziel muß ein erlaubter/gültiger Aufenthaltsbereich angegeben werden. Diese VALID ZONE (VZ) wird durch 3x3 Parameter (Kugelkoordinatensystems mit Hysterese) beschrieben:

- radialer Abstand CAM-TARGET → RANGE
  - minimal
  - optimal
  - maximal
- Seitenwinkel (Rotation der CAM um die Y-Achse des TARGETs) → Aus Sicht des Targets

ist dies der Azimuth (AZI) zur CAM

- minimal
- optimal
- maximal

- Höhenwinkel (Rotation der CAM um die X-Achse des TARGETs) → Elevation (ELEV)
  - minimal
  - optimal
  - maximal

Diese Kugelkoordinaten beziehen sich immer auf die **Position** des TARGETs. Die **Ausrichtung** des TARGETs wird dabei NICHT berücksichtigt. Angaben von ELEV und AZI sind also unabhängig davon ob der TARGET-SC steht, liegt oder seitlich umfällt.

Soweit so gut.

Die Kamera versucht nun stets in allen drei Kugelkoordinaten auf dem optimalen Wert zu sein. Aber wie genau wird dies bewerkstelligt?

1.2.4 TARGET bewegt sich

Für die Bewegung eines TARGETs wird zwischen **Translation** (SC geht, rennt, springt, „strafed“) und **Rotation** (SC dreht sich) unterschieden. Hierzu ein paar einfache Regeln.

Art der Bewegung	Auswirkung auf Kamera
während Z-Translation (vorwärts/rückwärts)	CAM dreht auf optimalen ELEV und AZI
Stop der Translation	CAM bewegt sich zur optimalen RANGE
während Y-Translation (hoch/runter)	CAM bleibt stehen, um ein ruhiges Bild zu bieten
während X-Translation (hoch/runter)	CAM bleibt stehen
während Rotation	CAM bleibt stehen

Hier einige Beispiele:

- Die CAM hinter dem SC (TARGET) hat die RANGE 2/3/4 (min/opt/max), die Kamera beginnt also bei 3m Abstand. Der SC bewegt sich nun kontinuierlich vorwärts während die CAM solange stehen bleibt, bis sie 4m Abstand zum SC hat. Dann folgt sie ihm in diesem Abstand, bis er wieder zum Stillstand kommt. Sobald dies der Fall ist (Stop der Translation), holt die CAM wieder zum SC auf bis sie wieder die optimale RANGE von 3m hat.
- Fall wie eben, aber der SC dreht sich um 90° nach rechts (AZI: +180°/0°/-180° also Rundumsicht erlaubt). Die CAM sieht den SC jetzt also von seiner rechten Seite und hat sich durch die Rotation des SCs nicht bewegt. Jetzt beginnt der SC kontinuierlich zu rennen. Dadurch („während der Translation“) wird der Azimuth der Kamera auf 0° (optimal) gedreht. Die RANGE selbst wird nicht korrigiert, aber alleine durch die Vorwärtsbewegung des SCs automatisch erhöht. Der Effekt müsste sein, daß sich die CAM von 3m/90° in einer Kurve auf 4m/0° bewegt.
- Der SC von oben steht in einem engen Gang und will einen Schalter in der Wand betätigen. Die Kamera steht auf 3m/0° und der SC schaut den Gang entlang. Nun dreht er sich nach links zum Schalter (CAM bewegt sich nicht, da nur Rotation), drückt diesen und dreht sich wieder zurück (CAM bewegt sich noch immer nicht). Erst wenn der weiter den Gang entlangläuft folgt ihm die CAM wieder. → Vorteil: ein sehr ruhiges Kameraverhalten, ohne den zwanghaften Bedarf, sich beim Drehen des SCs zum Schalter irgendwie zwischen den

SC und die dahinterliegende Wand zu quetschen.

- Würde der SC im engen Gang nach der Drehung zum Schalter sich um weitere 90° drehen und nun direkt in die Kamera schauen bliebe die Kamera immer noch stehen. Erst wenn er einen Schritt auf die CAM geht (Start der Translation) schwenkt der Azimuth der CAM wieder auf 0° (wenn auch hier in einer sehr engen Kurve nah am SC vorbei, da ja die Wand im Wege ist).
- Der SC (TARGET) hat wieder die RANGE 2/3/4m und diesmal AZI: +90°/0°/-90°. Der Spieler steht in einer großen Halle und dreht sich um 90° nach rechts. Nun ist er wieder von seiner rechten Seite zu sehen (CAM bewegt sich nicht). Dreht sich der Spieler nun noch weiter nach rechts, so bleibt die CAM an der AZI-Begrenzung +90° hängen und dreht sich in diesem AZI um den SC herum. Bewegt der sich wieder vorwärts, schwenkt die CAM schließlich wieder hinter ihn.

1.2.5 CAM-Kollision

Der Punkt „TARGET bewegt sich“ beschäftigte sich ausschließlich damit, was passiert, wenn sich die CAM beliebig in der gültigen Zone bewegen kann. Was aber passiert, wenn die CAM durch die Bewegung des TARGETs auf dem Weg durch die VZ an einer Wand oder einem VOB hängen bleiben oder besser gesagt, der angepeilte Aufenthaltsort für die CAM in einem VOB bzw. hinter einer statischen Wand und somit unerreichbar ist?

Hierfür muß für jede „blockierte Kugelkoordinate“ eine „Ausweichkoordinatenreihenfolge“ im Parametersatz angegeben werden können. Dies wird an einem Beispiel deutlicher:

Der CAM auf den SC hat RANGE 2/3/4m, AZI +90°/0°/-90° und ELEV 0°/30°/90°. Nun geht er rückwärts an eine ebene Wand. Die CAM kollidiert im Abstand von 2 Metern mit der Wand. Der SC tut einen weiteren Schritt nach hinten (sagen wir er sei jetzt 1 m von der Wand entfernt). Die CAM kann ihren Minimalabstand von 2m (direkt hinter der Figur) nicht mehr einhalten und muß nun entweder nach oben/unten oder zur Seite ausweichen. Sagen wir die CAM versucht zuerst nach oben/unten auszuweichen. Sollte dies nicht möglich sein (z.B. extrem niedrige Decke), muß sie nach rechts/links ausweichen.

In diesem Beispiel war die „blockierte Kugelkoordinate“ die RANGE und die „Ausweichkoordinatenreihenfolge“ ELEV vor AZI.

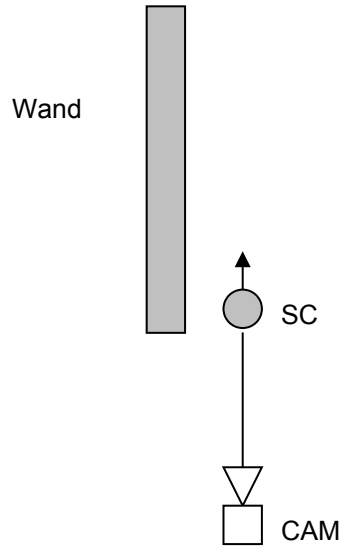
Sollte trotz Ausweichen in beiden „Ausweichkoordinaten“ die CAM keinen einzigen Punkt in der VZ finden, sollte die CAM in das Target fahren (First Person). Dies ist dann allerdings Dungeonfehldesign (oder vielleicht auch noch ein Fehler im Kamerasystem) und sollte uns zeigen, daß es noch Probleme gibt, die behoben werden müssen.

1.2.5.1 Durchlässige Materialien

Bei einigen statischen Polys macht es durchaus Sinn, die CAM nicht Kollidieren zu lassen. Zum Beispiel Wasser, Glas, Rauch, Feuer muß für CAMs durchfahrbar sein. Daher muß für Materialien angegeben werden können, daß sie für die Kamera durchlässig sind.

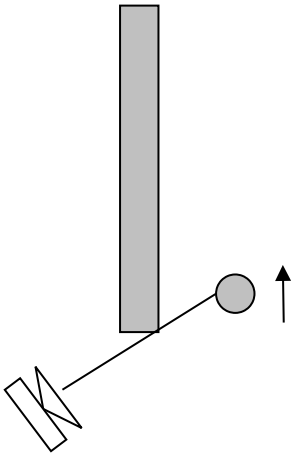
1.2.6 Durchbrechen der Line-Of-Sight

Wie Eingangs schon erwähnt, muß die CAM stets Sichtkontakt mit dem TARGET haben. Folgendes Szene aus der Vogelperspektive:



Dreht sich der SC nun 90° nach rechts bleibt die CAM zunächst wo sie ist (falls AZI +90°/0°/-90°). Macht der SC dann noch einen kleinen Schritt nach vorne, versucht die CAM auf den optimalen Seitenwinkel also auf 0° zu schwenken. Bevor sie diesen optimalen Punkt jedoch erreichen kann wird die Sichtlinie durch die Wand unterbrochen.

In diesem Fall muß die CAM genauso reagieren, als ob sie selbst mit einer Wand Kollision hätte. Sie bleibt also auf einem suboptimalen Seitenwinkel (natürlich immer in Bezug zur Blickrichtung des TARGETs) stehen.



Sollte dieser erzwungene suboptimale Seitenwinkel außerhalb der gültige Zone liegen, so muß daß bei 1.2.5 “CAM-Kollision” beschriebene Verfahren der „Ausweichkoordinatenreihenfolge“ angewendet werden.

Sinngemäß wird auch mit den anderen blockierten Kugelkoordinaten verfahren. Hier einige typischen Problemfälle:

- Der SC geht/rennt Kreise in einer Halle mit vielen Säulen.
- SC geht eine sehr steile Treppe runter (Treppenstufen zwingen evtl. die ELEV auf einen höheren Wert)
- SC geht durch einen niedrigen Torbogen. (Unterkante des Torbogens zwingt die ELEV auf einen niedrigeren Wert)

1.2.7 Dynamik der Kamerabewegung

Wann immer sich eine CAM bewegt, sollte die Bewegung nicht „abgehackt“ sein. Das heißt es muß ein dynamisches Beschleunigungs-/Abbremsverhalten geben. Dieses Verhalten muß im Parametersatz editierbar sein. Sinn könnte es machen, die physikalischen Werte:

- $\mathbf{v}_{max}$ (Maximalgeschwindigkeit in m/s bzw. °/s)
- $\mathbf{a}_{+}$ (positive Beschleunigung in m/s² bzw. °/s²)
- $\mathbf{a}_{-}$ (Abbremsbeschleunigung ebenfalls in m/s² bzw. °/s²)

für jede der Kugelkoordinaten anzugeben. Eventuell sind durch die programmtechnische Umsetzung des Beschleunigungsverhaltens auch andere Parameter sinnvoll, sollten aber einen ähnlichen Einfluß auf die Art der Dynamik zulassen.

1.2.8 Kamerafahrt zwischen Ist- und Soll-Koordinate

Kamerafahrten treten in folgenden Situationen auf:

- TARGET **beginnt** sich vorwärts/rückwärts zu bewegen und die CAM ist in diesem Moment nicht auf dem idealen ELEV/AZI (z.B.: erst umdrehen, dann wieder loslaufen).

- Umschalten von der Third-Person- auf eine Fixe Kamera und wieder zurück (ausschaltbar)
- Wechsel des Kameramodus (Soll-Koordinaten der CAM wird schlagartig verändert)

Generell sind dies Situationen, in welchen schlagartig der Ist- und der Soll-Wert der CAM-Position sehr weit auseinanderrücken. Da die CAM nun in kurzer Zeit eine große Strecke überbrücken muß, sollte man an die Kamerafahrt (im Gegensatz zur normalen Kamerabewegung) andere Anforderungen stellen:

- Der Zielpunkt muß vorausberechnet werden, bevor die Kamerafahrt beginnt.
- Der Weg zum Zielpunkt sollte keine Zacken haben. Eine einzige Kurve wäre ideal.
- Die Berücksichtigung der VZ (während der Fahrt) kann außer acht gelassen werden, da die Fahrt erstens sehr schnell und zweitens eventuell die VZ unbedingt verlassen muß. Natürlich muß der Zielpunkt wieder Line of Sight haben.
- Line of Sight-Check nur für den Zielpunkt (während der Fahrt muß keine LoS vorhanden sein)

Was trotzdem gilt:

- 1.2.7 „Dynamik der Kamerabewegung“
- 1.2.5 „CAM-Kollision“ (die Fahrt darf nicht in die statische Welt hineinführen, logisch)

1.2.9 Fixe Kameras

An einigen Stellen wird es Sinn machen, fixe Kameras zu installieren. Es sind zwei Arten von fixen Kameras nötig:

- Skriptgesteuerte Kamera
- Auto-Kamera

1.2.9.1 Skript-Kamera

Der Wechsel vom Kamera-Satelliten auf die Skript-Kamera wird durch ein Skript-Kommando durchgeführt. Ebenso der Wechsel zurück. Dies wird durch ein Event-VOB realisiert.

Diese Art der fixen Kamera gibt es prinzipiell bereits. Allerdings müssen für jede Skript-Kamera folgende Eigenschaften editierbar sein:

- Flag für automatische Zielverfolgung (nur Rotation)
- Flag, ob von und zur Skript-Kamera eine Kamerafahrt durchgeführt, oder direkt umgeschaltet werden soll.
- Flag, ob der SC gesteuert werden kann, während auf die fixe Kamera umgeschaltet ist

1.2.9.2 Auto-Kamera

Sobald das TARGET sich in Reichweite der Auto-Kamera befindet wird überprüft, ob sich das TARGET im Frustrum der Kamera befindet und Line-Of-Sight gegeben ist. Ist dies der Fall, so wird automatisch auf die Auto-Kamera umgeschaltet. Sind diese Kriterien nicht mehr gegeben, wird automatisch auf den Kamera-Satelliten zurückgeschaltet.

Für diese Art der Kamera sind folgende editierbare Eigenschaften nötig:

- Flag für automatische Zielverfolgung (siehe oben)
- Flag für Kamerafahrt (siehe oben)

- Flag für SC-Steuerung (siehe oben)
- Erfassungsreichweite (Ab wann wird die Überprüfung auf CAM-Umschaltung überhaupt erst durchgeführt)
- Flag, ob zusätzlich zur Überprüfung „TARGET in CAM-Frustrum“ auch ein LineOfSight-Check durchgeführt werden soll.

1.2.10 Parametersatz

Hier nun nochmal alle Parameter für einen Kameramodus im Überblick. Außerdem sind hier für die verschiedenen Kameramodi geschätzte Startwerte angegeben, die natürlich später feingetuned werden müssen.

Parameter	Sub-Parameter	Normal	Dialog	Inventory	Nahkampf	Fernkampf	Schlafen	Tod
Zielart	evtl. Skriptinstanz	SC	SC	SC				
Zielwinkel-Offset	Rotation um X-Achse	15°	15°	15°				
	Rotation um Y-Achse	0°	10°	20°				
	Rotation um Z-Achse	0°	0°	0°				
Radius CAM-TARGET	minimal	2	2	1,5				
	optimal	3	3	2				
	maximal	4	4	3				
Seitenwinkel CAM-TARGET	minimal	-90°	-70°	-250°				
	optimal	0°	20°	-160°				
	maximal	90°	110°	-70°				
Höhenwinkel CAM-TARGET	minimal	0°	0°	0°				
	optimal	30°	30°	30°				
	maximal	90°	90°	90°				
Ausweichkoord.-reihenfolge	für RANGE-Blockierung	1. ELEV	1. ELEV	1. ELEV				
	für ELEV-Blockierung	1. RNG	1. RNG	1. RNG				
	für AZI-Blockierung	1. RNG	1. RNG	1. RNG				
Dynamik – v_{max}	für RANGE	10 m/s	10 m/s	10 m/s				
	für ELEV	90°/s	90°/s	90°/s				
	für AZI	90°/s	90°/s	90°/s				
Dynamik – a+	für RANGE	30 m/s²	30 m/s²	30 m/s²				
	für ELEV	270°/s²	270°/s²	270°/s²				
	für AZI	270°/s²	270°/s²	270°/s²				
Dynamik – a-	für RANGE	-20 m/s²	-20 m/s²	-20 m/s²				
	für ELEV	-180°/s²	-180°/s²	-180°/s²				
	für AZI	-180°/s²	-180°/s²	-180°/s²				

WICHTIG: Editierbarkeit aller Kameraparameter WÄHREND des Spiels. Wahlweise Unterstützung eines „Kamera-Feintuning-Modus“ im Spiel, der zuschaltbar ist.

1.2.11 Skriptkommandos

Per Skript müssen folgende Kommandos zur Verfügung stehen, damit sie in Events eingebaut werden können:

- Switch to Camera
- Switch back
- Change Target

1.2.12 Weitere (nicht ganz klare) Punkte

- Ist es möglich, die Ausmaße des Frustrums zu ändern (Fischauge, Tele-, Weitwinkel,...) ?
- Eventuell ist es sinnvoll bei speziellen Kameras Farbfilter zu ermöglichen.
- Sind wie bei Hexen II dynamische Verzerrungseffekte unter Wasser möglich ?

1.3 Licht

Hier einige Anforderungen an die Technik des Lichtsystems:

- Dynamisches Licht: Geht von Fackeln, brennenden/magischen Geschossen, Magie und Leucht-Monstern aus.
- Statisches Licht (mit Schatten): von Lagerfeuern, großen(!) Fackeln, Tageslicht, Leuchtkristallen im Alten Tempel.
- Alles Licht in 2 Wirkungsradien mit (relativ) hartem Übergang. 1) Heller Lichtkern: Jeder wird gesehen; 2) Halbdunkel: Alle werden auf ihrer dem Licht zugewandten Seite hell beleuchtet, der Boden liegt im Halbdunkel, Talent „Hide in Shadows“ (s. dort) wird nicht hell beleuchtet und nicht gesehen (komplett halbdunkel texturiert); 3) totale Dunkelheit, keiner wird gesehen. Bezieht sich auch auf Gegenstände.
- Gegenstände und Monster sollten auch dargestellt werden, wenn sie in der totalen Dunkelheit stehen, damit sie sich als schwarze Silhouette vor einem beleuchteten Hintergrund abzeichnen können.

1.3.1 Dynamisches Licht

Dynamisches Licht bedeutet zum einen, die Lichtquelle ist mobil und nimmt ihr Licht mit, zum anderen, die Lichtparameter können animiert werden.

Egal in welcher Art die Parameter animiert werden, der Wechsel der Farbe in Abhängigkeit der Zeit findet auf einem frei definierbaren Farbenspektrum von 26 Farben statt. Die erste Farbe trägt den Kennbuchstaben „A“, die letzte „Z“.

Für dynamisches Licht könnten folgende Parameter angegeben werde.

- Startfarbe (Kennbuchstabe „A“) als RGB-Wert.
- Endfarbe (Kennbuchstabe „Z“) als RGB-Wert.
- 0 bis 24 (vielleicht auch weniger) Zwischenfarben (Kennbuchstaben „B“ bis „Y“) als RGB-Werte. Für jeden dieser Zwischenbuchstaben sollte eine eine eigenständige Farbe definiert werden können (muß aber nicht). Alle nicht belegten Buchstaben werden RGB-interpoliert. Das Entstehen dieses 26-Farbenspektrums ist identisch zur Bearbeitung einer Farbpalette im guten alten Deluxe Paint (unter Verwendung der „Spread“-Funktion).

- Art der Farbanimation
  - CONSTANT Nur Farbe „A“ wird angezeigt, die sich aber nie ändert
  - SWELLING (ruhiges, pingpongartiges „Hin- und Hergleiten“ auf dem 26-Farbspektrum)
  - CALM TORCH (ruhiges Flackern einer bewegungslosen Fackel ohne Luftzug)
  - WILD FLICKER (wildes Flackern einer Fackel in starkem Luftzug oder während der Spieler rennt)
  - BURN OUT (Flackern einer Fackel kurz bevor sie ausgeht)
  - <weitere> (vielleicht gibt es eine sinnvolle Parametrisierung so daß neue Arten generiert werden können)
- 1. Reichweite: Heller Lichtkreis
- 2. Reichweite: Halbdunkel-Kreis

2. Testmodus/Konsole

Das Programm sollte mit einer (nicht leicht nachvollziehbaren) Tastenkombination zwischen *Normalmodus* und *Testmodus* umschaltbar sein (z.B. SHIFT+STRG+M). Im *Normalmodus* sind nur Tastatureingaben möglich, die auch es auch in der Releaseversion geben wird, im *Testmodus* sind alle zusätzlichen „Developer-Keys“ sowie eine Kommandokonsole à la Quake und Konsorten möglich.

Bei Programmstart ohne Kommandozeilenparameter ist der Normalmodus aktiv. Für die Entwickler kann das Programm auch mit dem **Kommandozeilenparameter** `-TESTMODE` aufgerufen werden.

2.1 Developer-Keys

Während die Developer-Keys völlig frei sind und sich während der Entwicklung auch laufend ändern können (dann aber in KEYS.TXT dokumentieren !!!), wird für die Konsole nun ein erster Satz an Befehlen vorgeschlagen, der dafür gedacht ist, dem „Heer der Datenauffüller“ optimale Testmöglichkeiten zu geben.

2.2 Konsole

- Sobald ENTER (am besten nur auf dem NUM-Pad) gedrückt wird, erscheint die Konsole. Sie ist eine **transparente Box**, die das obere Drittel des Screens abdunkelt (aber transparent eben).
- In der untersten Zeile ist ein **Texteingabecursor** zu sehen.
- Nun kann ein **Kommandowort** eventuell mit **Parametern** eingegeben werden. RETURN bestätigt die Eingabe.
- Nach jedem Buchstaben wird gesucht, ob es noch ein anderes Befehlswort gibt, daß mit dem gleichen Buchstaben anfängt. Wenn nicht, wird sofort das entsprechende Befehlswort (z.B.: „insert“ nach Eingabe von „i“, da kein anderes Befehlswort mit „i“ anfängt“) ausgeschrieben → **Auto-Complement**. Gibt es mehrere Worte mit dem gleichen Buchstaben, wiederholt sich das Ganze beim zweiten Buchstaben solange, bis die Eingabe eindeutig ein Befehlswort beschreibt.

- Viele Befehle haben ein **2. Befehlswort** (toggle, edit, cheat,...) . Nachdem, das erste erfolgreich eingegeben wurde, wird mit dem zweiten genauso verfahren.
- Die Eingabe des **Parameters** (oft irgendein Instanzname) sollte, ebenso wie die Befehlswörter, die laufende Eingabe automatisch ergänzen.
- Jede Eingabe sollte entweder mit einer Erfolgsmeldung (z.B. „NSC *Fighter* inserted“) oder aber mit einer Fehlermeldung quittiert werden:
  - „unknown command <input>“
  - „unknown parameter <wrong paramter>“
  - „parameter missing“
- Die normale „Spieltastaturabfrage“ sollte komplett erhalten bleiben, damit der Held mit gezückter Waffe (ich meine natürlich „...der Developer mit geöffneter Konsole“) herumlaufen kann. Weder die 26 Buchstaben, noch die 10 Ziffern, noch die RETURN-Taste wird für das UI benötigt, so daß es zu keinen Kollisionen kommen sollte.
- Ein erneutes ENTER schließt die Konsole wieder.

2.2.1 Konsolen-Befehle

Hier nun die Referenz der Konsolenbefehle:

Befehlswort	**2. Befehlswort**	**Parameter**	**Anmerkung**
insert	-	Instanzname	Einfügen von Npcs, Items, MOBs
goto	waypoint	Instanzname des Waypoints	aktuelle Figur wird zum angegebenen Waypoint gebeamt
toggle	desktop	-	Toggle; blendet alles außer Viewport ein/aus
toggle	debug	-	Toggle; zeigt Debug-Meldungen an/nicht an
toggle	spy	-	Toggle; schaltet Meldungen an den zSpy ein/aus
toggle	text	-	Toggle; schaltet das Textfenster ein/aus
toggle	waynet	-	Toggle; schaltet die Anzeige des Wegnetzes ein/aus
toggle	frame	-	Toggle; schaltet die Frame-/Polyanzeige ein/aus
toggle	screen	-	Toggle; schaltet zwischen Fullscreen und Windowed um
edit	camera	-	Kamerakonsole öffnet sich
edit	combat	-	Kampfkonsole der kontrollierten Figur öffnet sich
edit	species	-	Konsole für speziesabhängige Werte der kontrollierten Figur öffnet sich
edit	abilities	-	Konsole für die Attribute, Talente und Zauber der kontrollierten Figur öffnet sich
edit	render	-	Konsole zur Einstellung von renderspezifischen Dingen wie wireframe/gouraud/textured, farclipping-/fog-/light-range
edit	world		Konsole zur Einstellung von Spielzeit (Stunden/Tage vor-/zurückstellen)
make	bsp	Genauigkeit von 0.1 bis 1	berechnet einen neuen BSP des aktuellen Levels
save	bsp	-	speichert den aktuellen BSP mit Fileselector (Defaultname erlaubt einfaches Bestätigen)
load	bsp	-	lädt einen BSP mit Fileselector (Defaultname wie bei savebsp
make	vertex	-	berechnet eine Vertex-Lightmap für statische Lichter
save	vertex	-	speichert die aktuelle Vertex-Lightmap mit Fileselector und Defaultname ab
load	vertex	-	lädt eine Vertex-Lightmap mit Fileselector und Defaultname
load	mds	-	lädt eine MDS-Datei für die kontrollierte Figur (Fileselector)
camera	mode	Instanzname des Kameramodes	Schaltet auf den angegebenen Kameramodus um.

camera	autoswitch	-	Toggle; schaltet die automatische Umschaltung der Kameramodi durch Spielsituationen ein/aus
cheat	godmode	-	Unverwundbar
cheat	full	-	Füllt Lebens-, Magie- und Willenspunkte wieder auf das Maximum auf.
save	position	-	merkt sich die Position der aktuellen Figur
load	position	-	setzt die aktuelle Figur auf die mit „save position“ abgespeicherte Position

3. NSC-Editor

- Dialogskript eingeben/ändern
- Charakterwerte eingeben

4. Ausgabe-Einheiten

Sämtlicher Output der Charaktere im Spiel (SCs sowie NSCs) wird über sogenannte „Ausgabe-Einheiten“ (Output-Units = OU‘s) abgewickelt. Dadurch werden folgende Problematiken gelöst:

- Synchronisation zwischen Text, Sprache und Animation der Figuren
- Trennung zwischen den Arbeitsschritten „Skripten“ und „Audio-Visualisieren“ mit einer vom Arbeitsfluß her sinnvollen Schnittstelle.
- Dialogliste für die Sprachaufnahmen im Studio (mit Kommentaren)
- Dialogliste für die Lokalisierung der Texte/Sprachaufnahmen (mit Kommentaren)

4.1 Editor

Während die eigentlichen Skripte in einem externen Texteditor erstellt und gewartet werden, benötigt die Bearbeitung und das Handling der OUs einen graphischen Editor. Dazu bietet sich der Spacer an.

Dazu sollte es im Spacer vier generelle Modi geben:

- Texturierungs-Modus
- Objektierungs-Modus
- Ausgabe-Einheiten-Modus
- Cutscene-Modus

Wird der Ausgabe-Einheiten-Modus aktiviert, so stehen folgende Funktionen zur Verfügung:

4.1.1 Button „Reload Scripts“

Hiermit wird das aktuelle SRC-File neu geparsed und die OU-Liste (siehe unten) aktualisiert. Die OU-Liste wird gebildet, indem alle Skriptbefehle

```
AI_Output(C_NPC npc, int id);              // Kommentar
```

gesammelt werden und für jeden dieser Befehle ein OU-Eintrag angelegt wird. „npc“ ist hierbei die Figur, auf die sich der Befehl bezieht, „id“ ist eine Konstante, die eine spätere Zuordnung von OU-Listen-Eintrag und Skriptzeile ermöglicht und als „Kommentar“ kann der vorgeschlagene Text angegeben werden, der bei der Neuerstellung eines OU-Listen-Eintrages als Textausgabe-String eingesetzt wird.

Eine Skriptzeile wie

```
AI_Output(self, NEW);                      // „Hau ab!“
```

würde also für den aktuell handelnden NSC („self“), einen neuen OU-Listen-Eintrag (Konstante „NEW“) mit dem Initial-String „Hau ab!“ erzeugen. Zugleich wird im Skript das Wort „NEW“ durch eine vom Editor ermittelte, eindeutige id ersetzt.

WICHTIG: Besteht schon eine OU-Liste, so behält ein „Reload Scripts“ alle Werte der OU-Liste bei und ergänzt diese nur durch neue Einträge. Es werden **keine** bestehenden OU-Listen-Einträge gelöscht (eventuell werden nur Teil-SRCs geparsed!).

4.1.2 OU-Liste

Die OU-Liste enthält alle Ausgabeeinheiten (OU’s) aller im SRC-File angegebenen Skripte. Jeder Eintrag dieser Liste enthält:

- ID (für Zuordunung zu Skripten)
- Instanzname der Figur
- Timing Sprachsamples (wann wird welches WAV-File gestartet?)
- Timing Animationen (wann wird welche Animation gestartet?)
- Timing Text (wann wird der Text aus dem Skript angezeigt?)
- Kommentar (manuell eingeben)

WICHTIG: Da der normale Lebenszyklus einer OU damit beginnt, daß ein Text aus den Skripten übernommen wird, aber weder Animations-, Sprach- oder sonstige Daten vorhanden sind, sollte beim Anlegen des OU-Listen-Eintrags per Default eine Standardtiming der gesamten OU vergeben werden (z.B. anhand der Textlänge: 0.1sec/character).

WICHTIG: Wird dann der „Nur-Text-OU“ ein WAV-File zugewiesen, so wird die Gesamtlänge der OU automatisch auf dieses WAV-File angepaßt. Das heißt, daß ein Text der auf 2s Länge geschätzt wurde (20 Zeichen) wird nach dem Zuweisen eines 2,5s WAV-Files auf die neue Länge angepaßt.

AUCH WICHTIG: Eine einzige Skriptzeile „AI_Output(...)“ kann zu mehreren Einträgen in die OU-Liste führen, wenn diese Zeile von mehreren Figuren benutzt wird. Benutzen also zum Beispiel 10 Söldner die Zeile

```
AI_Output(self, NEW);                      // „Hau ab!“
```

so werden 10 Einträge in der OU-Liste angelegt, die dann möglicherweise auf die selbe WAV-Datei verweisen aber unter Umständen andere Animatonen/Texte haben.

4.1.2.1 Sprachen-Auswahl

Da das Timing jeder Konversationssprache völlig unterschiedlich ist, ist es notwendig, für jede Sprache eine eigene OU-Liste anzulegen.

Im Editor muß daher eine Auswahl für die Sprache vorgenommen werden können. Vorläufig genügen folgende Konversationssprachen:

- Deutsch (Default)
- Englisch
- Französich
- Spanisch

Wird also z.B. „Englisch“ ausgewählt, wird die deutsche OU-Liste ausgeblendet und die englische dafür angezeigt.

4.1.2.2 Abspeichern

Beim Abspeichern der OU-Liste sollte eine Datei mit der Bezeichnung

```
\DATA\CUTSCENES\OUL\OU_GER.POL
```

angelegt werden. Für englisch/französich/spanisch wird ein anderes Kürzel z.B. „`_ENG/_FRE/_SPA`“ verwendet.

4.1.2.3 Filter/Sortierung

Da sich im Laufe der Datenerstellung sehr viele OUs ansammeln werden (mehrere tausend), sind Werkzeuge zur Erhöhung der Übersichtlichkeit nötig:

- Filter: Zeigt nur die OUs mit bestimmten Eigenschaften an
  - bestimmte Figur (Figureninstanz)
  - bestimmte Gilde (Variable in der Figureninstanz)
  - bestimmtes Storykapitel (bestimmtes Skriptunterverzeichnis)
- Sortierung alphabetisch
  - nach OU-Name
  - nach Stimme
  - nach Figur
  - nach Gilde
  - nach Cutscenes
  - nach Storykapitel

4.1.2.4 Editieren eines Eintrags

Wird ein Eintrag der OU-Liste angewählt, so öffnet sich ein sequencer-artiges Timingfenster mit folgenden Möglichkeiten:

- Zuweisen eines WAV-Files (suche ab \DATA\SOUND\SPEECH\<Sprachkürzel>\ → auf diese Weise können alle Sprachen die selben WAV-Namen in unterschiedlichen Verzeichnissen haben. Dies spart beim Anpassen des Timings der fremdsprachlichen Samples eine Menge Zuweisungsarbeit.)
- Startzeit des WAVs mit Schiebebalken ändern

- Zuweisen mehrerer Animationen
- Startzeiten der Animationen ändern
- Start- und Endzeit der Textdarstellung ändern

4.1.3 Textlisten

Sowohl für Sprachaufnahmen im Studio als auch für Übersetzer in andere Sprachen werden Textlisten benötigt. Für beide Zwecke genügt eine gemeinsame Liste denn der Bedarf ist sehr ähnlich.

Es wird eine Funktion benötigt, die aus der aktiven OU-Liste alle dargestellten (Filter!) OU-Texte in ein gemeinsames Textfile schreibt. Jeder Eintrag enthält:

- ID für Reimport
- WAV-Name (automatisch ermitteln, vielleicht aus ID?)
- Textstring
- Auto-Kommentar
  - Stimmcharakter ("//"-Kommentar hinter dem Eintrag „`voice =`" im NSC-Skript)
  - Kommentar aus der OU-Liste

4.1.3.1 Export Textlist

Mit dieser Funktion wird die Textliste erstellt und in einer Datei mit dem Namen

```
\data\cutscenes\textlists\<name>_<sprachenkürzel>.txt
```

abgelegt. Dabei kann der Name frei eingegeben werden und das Sprachkürzel wird von oben übernommen. Nun kann mit der Textliste ins Studio gegangen werden oder sie wird an einen Übersetzer weitergereicht.

4.1.3.2 Import Textlist

Sollte die Textliste an einen Übersetzer gegangen sein, so ersetzt er die deutschen Texte durch die jeweiligen fremdsprachlichen und liefert eine Datei im Format

```
\data\cutscenes\textlists\<gleicher name>_<neues sprachenkürzel>.txt
```

an. Diese lokalisierte Textliste wird nun eingelesen und daraufhin die entsprechende OU-Liste der Zielsprache aktualisiert.

4.2 OUs im Hauptprogramm

Damit nun das Hauptprogramm bei Auftreten einer Ausgabe-Einheit auf die richtige OU-Liste zugreifen kann, muß beim Start oder im Installer ein Sprachparameter angegeben werden.

5. Das Cutscene-System

Eine Cutscene ist eine Szene im Spiel, bei denen auf mehreren Spielern synchronisiert AICommands abgespielt werden können. AICommands werden zu dabei zu Blöcken zusammengestellt, die nacheinander abgespielt werden. Die Blöcke sollen so gewählt werden, daß

sie kleine logische Einheiten bilden. Zwischen diesen Blöcken kann die Szene vom Spieler unterbrochen werden.

5.1 Cutscene (was ist das)

Eine CS ist eine logische Sychronisations-Einheit (lSE). Eine lSE ist ein Block, der mehrere Spuren enthalten kann, die wiederum Blöcke enthalten können.

5.1.1 Spur

Eine Spur ist eine Aneinanderreihung von Blöcken. Sie dient zur Zuweisung einer Quelle, eines Akteurs, oder eines Elementes, auf die sich enthaltenen Blöcke beziehen. Eine Spur ist immer der Focus (siehe unten).

5.1.2 Spurtypen

Rollen-Spur alle Blöcke dieser Spur beziehen sich auf ein NSC, der diese (bestimmte) Rolle übernommen hat.

- Command-Spur enthält nur AICommand-Blöcke.
- BewegungA-Spur enthält nur BewegungA-Blöcke.
- Ambient-Spur enthält Ambient-Blöcke oder Kamera-blöcke.

5.1.3 OutputUnit

Eine Outputunit ist eine Cutscene mit nur einer Rolle
Sie ist damit als Teil eines Cutscene verwendbar.

5.1.4 Block

Ein Block kann eine konstante, oder eine erst zur Laufzeit festgelegte (zeitliche) Länge, oder gar keine Länge (fixer Zeitpunkt) haben. Er kann durch SC-Einwirkung, im folgenden Events genannt, unterbrochen oder abgebrochen werden. Erst, wenn der Block beendet wurde, egal ob regulär oder durch ein Event, kann der nächste Block auf einer Spur beginnen

5.1.5 atomare Blöcke

Text-Block Ein Wave-File wird gespielt und Text dazu ausgegeben; Spur=Quelle; konstante Länge;
Anim-Block Eine Animation wird gezeigt; Spur=Akteur; konstante Länge;
BewegungA-Bl. Der Akteur bewegt sich zu Punkt, sofern er ein Vob besitzt; Spur=Akteur; variable Länge;
AICommand-Block Der Akteur führt ein anderes AICommand aus; Spur=Akteur; variable Länge;

5.2 Was kann eine Cutscene

5.2.1 Die Rollen

An einer Cutscene können alle NSCs, eigendlich sogar alle Vobs teilnehmen. Sie melden sich über einen Namen an der Cutscene an, und warten, bis alle Startbedingungen erfüllt sind. Dabei können auch zwei verschiedene NSCs versuchen unter der Rolle A an der Szene teilzunehmen, die zweite wird gegebenenfalls abgelehnt. Weiter ist es auch möglich den Spieler an einer Cutscene teilnehmen zu lassen, wenn dieser vollständig, oder nur teilweise die Kontrolle Über seine Figur verliert. z.B. bei einem Intro oder einer längeren, vorgegebenen Unter- haltung mit einem NSC.

5.2.2 Was kann genau abgespielt werden

Es können grundsätzlich alle AICommands abgespielt werden. Dies erfordert nur eine spezielle Implementation des Kommandos für das CSSystem. Als Parameter dürfen die AICommands außer „self“ ein weiter NSC/Vob, der an der Cutscene teilnimmt, oder einen Wegpunkt haben. Ein Gegenstand, den eine Person nehmen soll, oder ein Stuhl, auf dem eine Person Platz nehmen soll, muß als Rolle in die Cutscene eingefügt werden. Das Vorhandensein des Stuhls ist dann eine Startbedingung, das verrücken des Stuhls ein eingreifen auf die Szene (s.u.). Soll ein anderer Gegenstand Ziel einer Aktion sein, z.B. ein von den NSC mitgebrachter Gegenstand, so muß durch den Ablauf der Szene verhindert werden, daß dieser vom Spieler manipuliert wird. Die Cutscene hat keine Kontrolle über diesen Gegenstand, und kann nicht entsprechend reagieren, und auch nicht ihre Existenz nachprüfen. Solche Vobs können aber über ihren Namen als Parameter für AICommands verwendet werden.

5.2.3 Wie startet eine Cutscene

Im Tagesablauf der NSCs kann das Teilnehmen an einer Cutscene gescriptet werden. Die Teilnehmer sollten sich zunächst an den Ort des Geschehens begeben, um sich dort anzumelden. Sind alle Startbedingungen erfüllt, ist die Cutscene bereit. Nun kann die Cutscene über einen Trigger, z.B. Boundingbox, die von einem Spieler betreten wird, zum Spielen gebracht werden. Bis zu diesen Zeitpunkt behalten alle NSCs ihre normale AI. Sie setzen sich also zur Wehr, wenn sie vom Spieler angegriffen werden. Ist die Szene im Lauf, reagiert die Cutscene auf Aktionen der Spieler.

5.2.4 Unterbrecher

Die Cutscene kann jederzeit durch die Benutzerführung, z.B. Menu-Zugriff, unterbrochen werden, und ihr Stand als Spielstand abgespeichert werden. Der zuletzt begonnene Block wird beim Wiederaufnehmen der Szene wiederholt. Dies ist z.B. nötig, wenn der Block zu wenig gespielt wurde, um vom Spieler wahrgenommen zu werden.

5.2.5 Unterbrecher durch Aktionen des Spielers

Die Cutscene, sogar jeder SynchronisationsBlock, kann Eigenschaften haben, ob und wie sie auf Aktionen des Spielers reagieren soll. Dabei sind alle Aktionen möglich, die auch eine Reaktion von normalen NSCs hervorrufen. Als Reaktion ist es möglich die Szene nur für einen kleinen Einschub zu Unterbrechen, oder die Szene vollständig zu beenden, und allen Teilnehmern ihre normale AI zurückzugeben. Nach einem Einschub, vielleicht wieder eine Cutscene, würde dann der letzte Block wiederholt werden (s.o.).

Grundsätzlich ist eine Cutscene für die Dauer des Spielens als ein Individuum zu sehen, daß von allen Spielern gesehen werden kann, und von allen Spielern beeinflußt werden kann.

5.3 Beispiele für Szenen

5.3.1 Gegenstand wird übergeben
Rolle A: "Hi, gib her!", Rolle B: "Hier hast du es!", Beide (Übergabe); Rolle A: "Danke!"

5.3.2 Unterhaltung
Mehrfacher Wortwechsel mit Kamerawechsel und Kamerafahrt.

5.3.3 Unterhaltung mit Besonderheit
Ein Gesprächspartner läuft zu fixem Punkt, nimmt dort Gegenstand auf und kehrt zurück (variable Länge).

5.3.4 Ein Gegenstand (z.B. magisch) aktiviert sich von selbst.
Das Wetter verändert sich durch eine Zaubergeste einer Rolle.

5.3.5 Sprechchor
Mehrere Rollen sprechen im Chor und Gestikulieren individuell. (ein wave und mehrere Gesten, oder viele waves?)

5.3.6 Scharade
Eine Rolle vollführt Gesten, während andere Rollen Sprechen und Gestikulieren.

5.3.7 Zerstörung
Eine Höhe bebt und fällt ein, während Personen sich (gestikulierend und rufend) retten (dramatisch Musik und Kamerafahrt)

5.3.8 Würfelspiel
Unterhaltung und Aktionen, die wiederholt werden.

6. Netzwerk/Multiplayer

6.1 Übersicht Multiplayer-Modus
Der Multiplayer-Ansatz ist einfach und genial:

In GOTHIC gibt es den **Hauptheld**en und vier **Nebenhelden**. Im Solospiel wird immer der Hauptheld gespielt und die Nebenhelden werden als besonders herausgearbeitete NSCs dem Spieler nahegebracht. Dabei werden die Nebenhelden so designed, daß sie dem Spieler ebenso ans Herz wachsen wie sein Hauptheld.

Das Speichern des Solospiels beinhaltet somit **einen** konsistenten Spielstand mit allen fünf Helden, der auch für Multiplayer-Spiele benutzt werden kann.

In einem Multiplayerspiel übernimmt immer ein Spieler den Hauptheld und damit die gesamte dazugehörige, konsistente Spielwelt. Die Mitspieler übernehmen in dieser Welt die (Gast-)Rolle eines Nebenhelden, dessen Charakter sie aus dem eigenen Solospiel sehr gut kennen.

Vorteile:

- Erster wirklicher RPG-Multiplayer-Mode **mit kompletter Story** und allen dazugehörigen Aufträgen und Rätseln.
- Ein-/Aussteigen in ein Solospiel möglich. Dabei werden Nebenhelden nach dem Start im Multiplayer-Mode von NSCs zu SCs, bei späterem Fortsetzen als Solospiel wieder zu NSCs. → Keine Trennung von Solo- und Multiplayerspiel
- Kein Verändern von Solospielwelten durch ein früher geplantes „Mergen von verschiedenen Solospielwelten".
- Optimale Berücksichtigung der häufigsten RPG-Multiplayer-Situation: Spieler spielen 90% der Zeit alleine und wollen auch mal für einen Nachmittag zusammen mit dem Kumpel spielen.
- Trotzdem möglich: Komplettes durchspielen der gesamten Story gemeinsam mit Freunden → „Gastrolle" als Nebenheld wird dabei zur vollwertigen Hauptrolle, denn die Nebenhelden werden dann zu jeder Zeit von den Mitspielern geführt.

6.2 Design

Hier einige ungeordnete Designpunkte zum Multiplayer-Modus

- Haupthelden sowie Nebenhelden werden nach dem Tod auf dem Altar des Steinkreises automatisch wiederbelebt.

6.3 Technik

Hier nun Technikpunkte:

- Begriffsbestimmung: Der Rechner, der ein Multiplayer-Spiel startet (entweder „neues Spiel" oder „Spiel laden") wird im Konzept als **MASTER** bezeichnet. Alle Rechner, die im Multiplayer-Menü mittels des Befehls „Mitspielen" in der Welt des Masters mitspielen werden als **SLAVE**s bezeichnet
- Durch die getrennten Spielmodi „Solo" und „Multiplayer" ist kein „Anklopfen" von Mitspielern während einer laufenden Solorunde nötig. Will der Solospieler andere Mitspieler reinholen, so speichert er, geht zurück ins Hauptmenü, startet ein Multiplayer-Modus und wird so zum Master. Beim Start lädt er seinen Solo-Spielstand und holt dort auch die Mitspieler als Slaves rein.
- Wird ein Slave während einer laufenden Multiplayer-Runde ordnungsgemäß runtergefahren, so können der Master und die eventuell anderen Slaves weiterspielen. Der Nebenheld des runtergefahrenen Slaves wird zum NSC und von der AI übernommen.
- Stürzt ein Slave ab, so sollten ebenfalls alle anderen Spieler weiterhin spielen können.
- Wird ein Master ordentlich runtergefahren oder stürzt er ab, so wird für alle Slaves das Spiel automatisch beendet und ins Hauptmenü zurückgekehrt.

- Auf keinen Fall darf ein Absturz eines Rechners, auch auf anderen Rechnern zum Absturz führen.
- Hat ein Rechner kein TCP/IP-Protokoll installiert, so sollte das Programm dies automatisch entdecken und eine entsprechende Fehlermeldung ausgeben. Noch besser wäre, wenn automatisch das entsprechende Dialogfenster von Win95/98/NT aufgerufen wird.
- Optional: Die „Chatline“ im Spiel könnte durch eine Sprachübertragung ergänzt werden, falls der Rechner über ein Mikrophon verfügt.