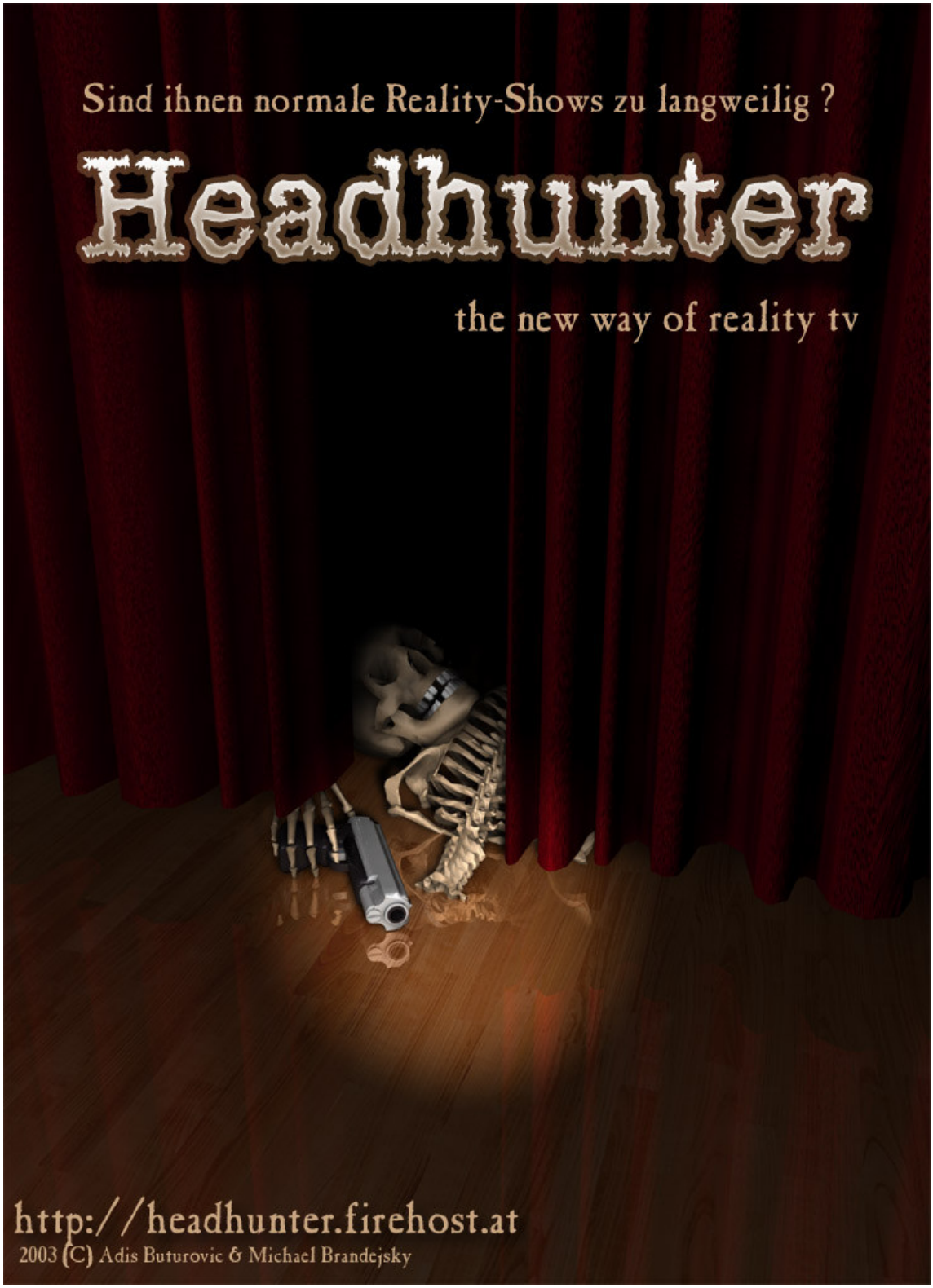


3te und letzte Abgabe CG 2/3 Übungen, SS 2003

Brandejsky, Michael, 532, 0125168 , michael@brandejsky.com
Buturovic, Adis, 532, 0125423 , adisb@chello.at

Inhalt

Das Spiel:

Im Amerika des Jahres 2025 hat die Fernsehunterhaltung eine neue Dimensionen erreicht. Die Regierung hat die Macht des Fernsehens schon Jahre zuvor erkannt und arbeitet nun eng mit dem Fernsehen zusammen. Das amerikanische Volk wird dabei durch eine Unzahl makaberer Gameshow's unterhalten. So erreicht man gleich zweierlei - Einmal werden die Menschen vor den Bildschirmen gehalten. Ihnen wird das Denken zugunsten niederster Triebe der Gewalt abgenommen und ihre eigenen Aggressionen können sie so durch ihre Begeisterung für perfide Mörderstars abreagieren, anstatt sich dem revolutionären Kampf zuzuwenden, zum Anderen wird die Lüge vom gerechten und allgewaltigen Staat publikumswirksam transportiert. Der Staat ist gerecht, und die Verbrecher bekommen das, was sie verdienen. So kann jeder seine perverse Mordlust ausleben und sich auch noch als guter und gerechter Staatsbürger fühlen.

Die beliebteste der Fernseh-Gameshows ist dabei "Headhunter". Darin kämpfen straffällig gewordene Bürger in speziellen Kampfarena's um ihr Leben.
Sie werden dabei von Kopfgeldjägern gejagt. Schaffen es die Verbrecher durch alle Kampfarena's bis zum Ausgang , so sind sie frei , begnadigt und um einige Millionen Dollar reicher.

Bis jetzt hat es jedoch noch niemand geschafft.....

Headhunter ist ein genre-typischer 3D Shooter. Der Spieler wird dabei in eine künstliche, dreidimensionale Welt (Battlearena) versetzt, in welcher er sich aus der Perspektive einer sich darin bewegenden Spielfigur wahrnimmt.
Zur Steuerung wird eine Kombination aus Maus und Tastatur benutzt (Details siehe unten), die Maus wird für die Kopfbewegung und die Tastatur für diverse Fähigkeiten und für die Bewegung im virtuellen Raum verwendet.
Der Spieler ist nicht allein in seiner Welt. Ihm stellen sich ebenso wie die gesteuerte Spielfigur bewaffnete Gegner (Kopfgeldjäger) in den Weg. Diese Gegner werden als 3D Modele angezeigt und durch eine künstlichen Intelligenz gesteuert. Die künstliche Intelligenz steuert dabei den gesamten Aktionsrahmen der Kopfgeldjäger (Bewegung und Aktionen).

Die Features des Spiels:

- **künstliche Intelligenz bei den Gegnern:** Aufgrund der nicht-geforderten komplexen KI haben wir unsere KI weitgehend bescheiden gehalten. Die von uns gewünschte Fähigkeiten wie Flüchten und dergleichen (deren Anfänge bis zur letzten Abgabe ausgedacht und ausprobiert wurden) wurden auf Grund von ihrer Komplexität und Zeitmangels leider nicht realisiert.

- **animiertes Gegnermodel:** Mit Hilfe des Quake3 MD3 Modelsystems wurde ein animiertes Gegnermodel realisiert. Ober- und Unterkörper können dabei unabhängig voneinander animiert werden. So kann die Figur z.B. gehen während der Oberkörper gleichzeitig die Attackbewegung ausführt. Der Kopf kann leider keine besonderen Animationen besitzen.

- **laden von verschiedene Gegnermodels und Gegnertypen möglich:** Unser System ist so ausgelegt das verschiedene Gegnermodels mit verschiedenen Eigenschaften (Aggressivität, bevorzugter Waffentyp, usw.) geladen werden können.

- **laden von verschiedenen Levels möglich und Lightmaps :** Laden von Quake3 BSP Dateien und Lightmap's ist realisiert worden.

- **Partikelsystem:** Es wurde ein Partikelsystem entwickelt. Dabei gibt es zwei Typen von erzeugten Partikeln: Rauchspuren der Waffen und Explosionen.

- **Verschiedenen Waffenarten:** Zur Zeit sind 2 verschiedene Waffen im System eingebaut. Das eine ist ein Lasermaschinengewehr , das andere ein Raketenwerfer. Beide Waffen schießen Projektile ab, welche eine Partikelspur hinterlassen.

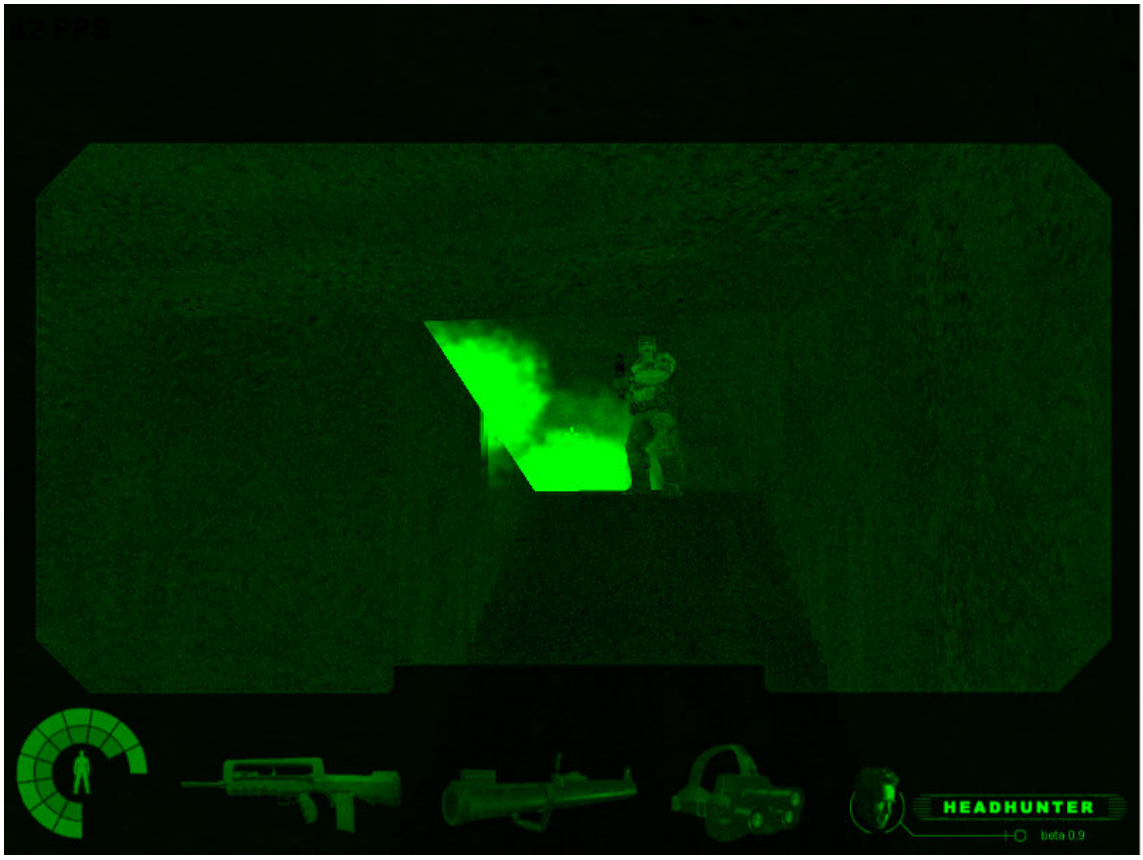
- **Bewegter Rauch und Explosionen:** Die Projektile der beiden Waffentypen stoßen während des Fluges kleinere Rauchwolken (abhängig vom Waffentyp) aus. Beim Auftreffen auf ein Hindernis explodieren die Waffenprojektile mit einer vom Waffentyp abhängigen Explosion.

- **Collision Detection:** Die Collision - Detection Level <-> Model und Level <-> Partikel ist mit Hilfe des Brushsystems von Quake3 realisiert worden. Ein weiteres Collision – Detection - System für die Waffen <-> Model wurde unter Hilfename von BoundingBoxen realisiert.

- **Gravitationssystem, Stiegensteigen, Sliden an der Wand :** Ein Gravitationssystem zieht die Spielfiguren nach unten. Das Stiegensteigen und das entlangsliden an der Wand wurde mit Hilfe der Collision Detection realisiert.

- **HUD (head - up display):** Das transparente HUD zeigt alle wichtigen Informationen für den Spieler wie z.B.: Waffenstatus an.

- **Nightvision - Modus:** Der Nightvision - Modus simuliert ein Nachtsichtgerät welches man bei dunklen Stellen einsetzen kannst. Der Schirm wird halbtransparent - grün und der Spieler ist fähig, Sachen zu erkennen ,welche man mit freiem Auge nicht so leicht erkennen kann.



- **Bullettime-Modus:** Als erster 3D First-Person-Shooter hat das HeadHunter Team den berühmten Bullettimemodus integriert. Später soll dieser Modus per Bonusgegenstand benutzbar sein. Die Bullettime erlaubt es dem Spieler per Knopfdruck die Geschwindigkeit zu verlangsamen. Alles läuft dann wie in Zeitlupe ab. Man sieht einzelne Kugeln fliegen und kann ihnen auch ausweichen.

- **Sound und Musiksystem:** Bei verschiedenen Events wie das Abfeuern einer Kugel oder der Aufprall einer Kugel löst einen Sound aus. Weiters gibt es eine Hintergrundmusik welche automatisch abgespielt wird. Der Sound und die Musik wurden mit Hilfe von FMOD realisiert.

Die Steuerung des Spiels im Detail :

Pfeiltaste links ←	Bewegung des Spielers nach links (Strafe left)
Pfeiltaste rechts →	Bewegung des Spielers nach rechts (Strafe right)
Pfeiltaste oben ↑	Bewegung des Spielers in Richtung des mit der Maus anvisierten Ziels (move forward)
Pfeiltaste unten ↓	Bewegung des Spielers in Gegenrichtung des mit der Maus anvisierten Ziels (move backward)
Pfeiltaste links a	Bewegung des Spielers nach links (Strafe left)
Pfeiltaste rechts d	Bewegung des Spielers nach rechts (Strafe right)
Pfeiltaste oben w	Bewegung des Spielers in Richtung des mit der Maus anvisierten Ziels (move forward)
Pfeiltaste unten s	Bewegung des Spielers in Gegenrichtung des mit der Maus anvisierten Ziels (move backward)
Mausbewegung	Steuerung der Sicht (durch Bewegung der Maus nach oben, unten, links und rechts)
Taste Esc	Spiel beenden
Taste n	Nachtsichtgerät wird aktiviert/deaktiviert. Siehe Featurebeschreibung oben!
Taste 1	Waffe 1 „Lasermaschinengewehr“ wird benutzt
Taste 2	Waffe 2 „Rocket Launcher“ wird benutzt
Maustaste links	Abschuss eines Schusses mit der gewählten Waffe
Maustaste rechts	Sprung des Spielers, noch nicht vollständig implementiert.
Space	Sprung des Spielers, noch nicht vollständig implementiert.

Cheats und Debug Infos :

Taste Pos1 (Home)	Bewegung des Spielers nach oben (wenn Gravitation ausgeschalten)
Taste Ende (Ende)	Bewegung des Spielers nach unten (wenn Gravitation ausgeschalten)
Taste Einfg (Ins)	Botgangart wechselt zwischen Gehen und Laufen
Taste Entf (Del)	Bot kommt unverzüglich zum Spieler nächstgelegenen Waypoint
Taste F2	Wireframemodus wird aktiviert, alle Objekte und der Level wird nur mit Linien gezeichnet
Taste F3	Blending wird aktiviert, nun kann man durch alle Objekte hindurchsehen (wenn das HUD ausgeschalten ist)
Taste F4	Debuginformationen werden angezeigt. Von links nach rechts und oben nach unten : FPS, Koordinaten des Bots (X,Y,Z) , Letzter besuchter Waypoint des Bots (L_WP), Nächstes Waypoint-Ziel des Bots (T_WP), Hauptwaypoint-Ziel des Bots (T_M) , Bot sieht dich oder nicht (noch nicht vollständig implementiert), Winkel des Bots, derzeitiger Status des Bots, Koordinaten des Spielers (XYZ), nächstgelegender Waypoint des Spielers und diverse andere Informationen :)
Taste F5	Waypoints des Bots werden als Pyramiden angezeigt (die weiße Linie dazwischen bedeutet ,dass sich der Bot zwischen den beiden Waypoints bewegen kann)
Taste F6	Lightmaps werden angezeigt oder deaktiviert
Taste F7	Gravitation des Players wird ein bzw ausgeschalten
Taste F11	Bullettimemodus wird aktiviert, alles außer dem Player bewegt sich um ein vielfaches langsamer. Siehe Featurebeschreibung oben!
Taste F12	Maus wird invertiert bzw nicht-invertiert
Taste Enter (Return)	HUD ein bzw. ausschalten
Taste f	Nebel ein/ausschalten
Taste +	Nebelsichtweite wird verringert
Taste –	Nebelsichtweite wird erhöht
Mittlere Maustaste	Collision-Detection für den Player ein/aus, bei ausgeschaltener Collision-Detection kann sich der Player durch die Wände bewegen.

Verwendete Sourcefiles :

Init.cpp	Initalisierungsroutinen
Main.cpp	Hauptprogramm
main.h	Hauptdefinitionen
gamedefinitions.h	Spieledefinitionen (für Bot,...)
Camera.cpp	Kamerafunktionen, Steuerung
Camera.h	Definitionen zur Kamera
Bot.cpp.h	Steuerung eines Bots
Bot.h	Definitionen zur Steuerung
level.cpp.h	Levelinformationen (Waypoints,...)
level.h	Definitionen zum Level
objekt.cpp.h	Objekte im Spiel
objekt.h	Definitionen zu den Objekten
collision.cpp.h	Collisionsabfrage von beiden Collisionstypen (Boundingbox & Brushes)
collision.h	Definition dazu
partikel.cpp.h	Partikelsystem: Emitter(zur Zeit ungenutzt) und Partikel
partikel.h	Definition der Emitter und Partikel
sound.cpp.h	Soundsystem
sound.h	Definitionen dazu
config.h	Configuration speichern/laden (noch nicht implemetiert)
Image.cpp	Imageloader (BMP, TGA, JPG)
Image.h	Definitionen zum Imageloader
jpeglib.h	JPEG Libary (http://www.ijg.org/)
Md3.cpp	Quake 3 MD3 - Modelloader
Md3.h	Definitionen zum Modelloader
Frustum.cpp	Anzeige der durch die Kamera sichtbaren Szene
Frustum.h	Definitionen zur Frustum View
Quake3Bsp.cpp	Quake 3 BSP - Levelloader
Quake3Bsp.h	Definitionen zum Levelloader

Detailangaben zu den Sourcefiles:

Init.cpp : Ist für die Initialisierung des Fensters, der OpenGL Funktionen zuständig. Weiters schließt diese Funktion am Ende des Programms das Fenster.

Main.cpp, main.h : Main.cpp ist das Hauptprogramm, es erzeugt die am Ende vorhandene Headhunter.exe.
Das Programm ladet alle Unterfunktionen, initialisiert mit Hilfe von Init.cpp das Fenster, ladet den Level, die vorhandenen Models, die Levelstruktur und Leveldefinitionen.
Danach wird der Bot initialisiert und das Spiel gestartet.
Nachdem nun das Spiel gestartet wurde, ist das Programm für dessen gesamten Ablauf zuständig.

gamedefinitions.h : Hier befinden sich alle Spieldefinitionen sowie oft verwendete Vektorfunktionen. Die verwendeten Spieldefinitionen sind die Parameter für die Bot-KI , die Definition der Waypoints und die Definition eines Bots (Stärken, mögliche Aktionen,...).

Camera.cpp, Camera.h : Mit dieser Klasse wird die Kamera realisiert. Die Kamera wird mit Hilfe von 3 Vektoren erstellt und bewegt.
Der Position, der View und der Up Vektor.
Der Position ist für die Position der Kamera zuständig.
Der View Vektor zeigt an in welche Richtung die Kamera schaut.
Der Up Vektor zeigt an welche Richtung oben ist.
Mit Hilfe von verschiedenen Unterfunktionen wird die Kamera bewegt und rotiert.

Bot.cpp.h, Bot.h : Hier ist die künstliche Intelligenz des Bots realisiert. Sie initialisiert zuerst die vorhandenen Bots und steuert dann ihr Verhalten.
Mit Hilfe eines Zustandssystems wird der jeweilige Aktionszustand realisiert.
So schaltet der Bot automatisch wenn er den Spieler sieht in den Angriffsmodus.
Ist der Spieler verschwunden so geht der Bot auf die Jagd und geht zu dem ihm bekannten letzten Aufenthaltsort des Spielers.
Ansonsten geht der Bot auf Spielersuche, er geht dabei zur Zeit von Waypoint zu Waypoint.
Der Bot hat dabei 2 Ziel-Waypoints: das nächste Waypoint-Ziel des Bots und das Hauptwaypoint-Ziel des Bots.
Das Hauptwaypoint-Ziel steht für das eigentliche Ziel aus dem mit Hilfe der Dijkstramatrix der schnellste Weg zu diesem Ziel gesucht wird.
Die Waypointziele geben dabei die jeweiligen Zwischenstationen an.
Die Bewegung des Bots wird ähnlich der Kamera mit Hilfe von 2 Vektormatrizen realisiert. Die Vektormatrizen pos und dir stehen dabei für Position und Richtung.

level.cpp.h, level.h : Hier werden die Detailinformationen zu dem Level initialisiert und berechnet. Das sind die Waypoints, die dazugehörige Adjazenzmatrix und eine sich-selbst-erstellende Dijkstramatrix.
Die Waypoints sind Wegepunkte mit deren Hilfe sich die Bots im Level bewegen können. Diese werden später über eine Datei hineingeladen.
Die dazugehörige Adjazenzmatrix beinhaltet Informationen damit der Bot weiß, welche Waypoints verbunden sind und welche nicht.
Die Dijkstramatrix wird am Start des Spiels berechnet und besteht aus Informationen wie der Bot sich am schnellsten und intelligentesten von seinem Startwaypoint zum Endwaypoint bewegen kann.

objekt.cpp, objekt.h : Hier werden später dann , die im Spiel vorhandenen Elemente (Schüsse, Leben, usw.) erstellt, ihr Verhalten gesteuert. Dieser Teil ist noch nicht implementiert und folgt in der nächsten Abgabe.

collision.cpp.h, collision.h : Collisionsabfrage von beiden Collisionsarten, (Boundingbox & Brushes). Liefert zurück ob es eine Collision gab oder nicht.

partikel.cpp.h, partikel.h : Partikelsystem zur Erzeugung von Rauch und Explosionen. Die Klasse ist für das gesamte Partikelverhalten zuständig (erzeugt die Partikel, bewegt , gibt sie aus und löscht die Partikel).

sound.cpp.h, sound.h : Das FMOD Soundsystem – Initialisierungsroutine und die Routinen zum Laden und Abspielen von WAV und MP3 Files. (loadmp3, playmp3)

config.h: Konfiguration speichern/laden (noch nicht implementiert)

Image.cpp, Image.h, jpeglib.h : Hier werden verschiedene Grafiktypen (BMP, TGA und JPEG) geladen und der dazugehörige Speicher freigemacht. Diese Routinen werden für die Texturen benutzt.

Md3.cpp, Md3.h : Ist zuständig für das Laden eines Quake 3 Models.
Ein Quake3 Model besteht aus 3 Segmenten: head.md3 - Kopf, upper.md3 - Oberkörper , lower.md3 – Beine
Bei den Md3 Models können Ober- und Unterkörper animiert werden. Es gibt viele verschiedene Animationen. Unter anderem laufen, gehen, stehen, schwimmen, sterben, schießen und andere.
Ober- und Unterkörper können dabei unabhängig voneinander animiert werden. So kann die Figur z.B. gehen während der Oberkörper gleichzeitig die Angriffbewegung ausführt. Der Kopf kann leider keine Animation haben. Nach dem Laden werden die einzelnen Teile zusammengefügt und das Model am Bildschirm dargestellt.

Quake3Bsp.cpp, Quake3Bsp.h : Ist zuständig für das Laden des Levels.
Dabei wird das von Quake3-bekannte BSP-Format verwendet. Wir benutzen zum erstellen des Levels den Editor GtkRadiant von ID-Software. Die mit diesem Editor erstellten .map-Dateien werden am Ende mit dem BSP-Compiler, der mit GtkRadiant mitgeliefert wird, kompiliert. Die dabei erzeugten BSP Levels bestehen grundsätzlich aus brushes und entities, wobei brushes die Geometrie bilden und entities für weitere Informationen an die Spieleengine verwendet werden. Sie werden im Editor als kleine Icons angezeigt, die im Raum plaziert werden können.
Sie können verschiedene Parameter annehmen und Werte speichern. Somit kann mit entities etwa ein Licht gesetzt werden, dessen Farbe, Intensität, Radius, etc. über solche entity-Parameter angegeben werden können.
Wir benutzen zur Zeit noch keine solche Entities für Informationen für unsere Gameengine.
Der Hauptvorteil von BSP aber ist, die Optimierung der Sichtbarkeit. Da die BSP Dateien aus BSP Bäumen bestehen ist dies sehr einfach. BSP steht für „Binary Space Partitioning – Baum“ und bedeutet, dass wir unsere simulierte 3D Welt damit in einzelne Partitionen unterteilen. Diese Partitionen werden in binären Bäumen gespeichert. Sinn so eines Baumes ist es, die Daten eines Levels so zu strukturieren, dass wir wesentlich schneller entscheiden können welche Teile des Levels im potentiellen Sichtbereich des Spielers liegen (vor der Kamera). Diese Berechnung erfolgt dabei mit Hilfe von Frustum.cpp. Da zur selben Zeit am selben Ort nie die gesamte Welt sichtbar ist, muss sie auch nicht komplett gezeichnet werden.

Es wird deshalb immer nur die Geometrie dargestellt, die vom Spieler aus gerade sichtbar ist.

Frustum.cpp, Frustum.h : Trennt die Teile , welche innerhalb des Sichtfelds der Kamera sind , von denen , welche außerhalb liegen. Die Teile , welche innerhalb des Sichtfelds liegen , werden angezeigt.

Verwendete Verzeichnissstruktur :

/bin	Das Hauptprogramm
/fidget	Das Gegnermodel des Bots 0
/jack	Das Gegnermodel des Bots 1
/sound	Soundlibrary, Sounds und Musik
/maps	Verwendete Maps des Levels
/textures	Verwendete Texturen des Levels
/weapons	Waffenmodels

Verwendete Software :

Photoshop 7.0e
Visual C++ .Net
GtkRadiant 1.2.1.3-update
Milkshape 3d 1.6.6a
BPM Studio

Quellenangaben :

Es wurden Tutorials von gametutorials.com als Grundlage für einige die Level und Model Loader verwenden und abgeändert.

Weiters wurde das Forum von gametutorials.com bei schwierigeren Problem in Anspruch genommen.

Hilfe von verschiedenen Artikeln:

Camera System:
http://nehe.gamedev.net/data/articles/article.asp?article=08

Partikel System:
http://nehe.gamedev.net/data/articles/article.asp?article=06

Billboarding:
http://nate.scuzzy.net/docs/billboard/

Erzeugung eines Wasserfalls durch ein Partikelsystem:
http://www.gup.uni-linz.ac.at/thesis/practical/stephan_gsoellpointner/waterfall.pdf

Das Soundsystem:
http://www.fmod.org

Schlussbemerkung :

Wir fanden die Übung sehr unterhaltsam und moderner im Vergleich zu anderen Lehrveranstaltungen. Da wir beide jedoch das Bakkalaureat Medieninformatik-Design studieren, das Fach deshalb nur als Freifach belegen und andere Pflichtübungen machen musste, konnten wir nicht so viel Zeit wie wir erhofft hatten dafür verwenden. Deshalb konnten wir vor allem gegen Ende des Semesters verschiedene Spielabläufe, Features , Effekte und Vorschläge nicht mehr realisieren.