Quake 4 - Catch the Chicken

Wilde Bauern im Jahre 2523 jagen ihren entlaufenen Hühnern nach und bringen die Hühnerdiebe zur Strecke.

Gliederung:

1 Einleitung

Unsere Modifikation "Catch the Chicken" für Quake 4 ist die Fortsetzung einer Reihe von Mods mit gleichem Namen für Quake 2 und Quake 3. Bisher existiert kein Mod für Quake 4 der das Spielprinzip von Catch the Chicken umsetzt. Im Internet[1] findet man nur einen Mod, dessen Entwicklung aber abgebrochen wurde. So haben wir es uns zur Aufgabe gemacht die Fortsetzung für Quake 4 zu entwickeln.

Das Spielprinzip von Catch the Chicken ist schnell erklärt. Im Kontrast zum normalen Deathmatch Modus gibt es bei Chatch the Chicken einen Spieler von besonderer Bedeutung, den Hühnchenträger. Beim Start des Spiels wird zufällig einem Spieler die Bürde (oder die Ehre) des Hühnchens auferlegt. Der Spieler, welcher im Besitz des Huhnes ist, hat ab sofort keine Waffen mehr und kann lediglich das Huhn als Nahkampfwaffe nutzen. Der Hühnchenträger bekommt alle 30 Sekunden, die er das Huhn trägt, 3 Punkte. Um nun auch die Fairness aufrecht zu erhalten läuft der Spieler mit erhöhter Geschwindigkeit, mittels des "Haste"- PowerUp. Die Aufgabe der anderen Spieler ist es nun den Träger des Hühnchens zu töten. Gelingt es einem Spieler den Träger zu töten so lässt der sterbende Spieler das Huhn fallen und das Huhn bleibt nun an der Position, an der der vorherige Träger starb. Das Huhn kann nun von einem anderen Spieler aufgenommen werden, welcher daraufhin einen Punkt für die Aufnahme des Huhns erhält. Die Punkte die man für die Aufnahme bzw. das Tragen des Huhns erhält werden als Frags gewertet. Die Runde endet also, wenn das Fraglimit bzw. Timelimit erreicht wird und der Gewinner ist derjenige mit den meisten Frags.

2 Aufgaben

Um das oben genannte Gameplay zu realisieren, war es nötig das Problem in einzelne Aufgaben zu zerlegen. Im Folgenden sind diese Teilaufgaben grob aufgeführt:

- Hinzufügen einer Waffe, dem Chicken
- also zunächst Einfügen einer neuen Waffe in das Spiel; mit audio/visuellen Effekten versehen; 3D Modell, Animation und Textur definieren
- Multiplayer- Spiellogik verändern: bei Beginn des Spiels muss das Chicken spawnen, bzw. direkt jemand zugeteilt werden; die Informationen über Besitzer des Huhns im Netzwerk verteilen
- Spielintegrität sicherstellen: verlässt der Huhn- Besitzer das Spiel: sicherstellen, dass Huhn nicht aus Spiel verschwindet; also anderem Spieler zuweisen
- Implementierung einer GUI um über Events, Huhn- Besitzer und Punkte zu informieren; Synchronisation der GUI- Ereignisse über das Netzwerk

[1] http://mods.moddb.com/6101/catch-the-chicken

- ▷ Punktemodell definieren; bei Aufnahme des Huhns erhält der Spieler, bzw das Team, 1 Punkt; hält der Spieler es 30 Sekunden erhält er, bzw. das Team, 3 Punkte
- ▷ Fairness Anpassungen: Huhn als Nahkampfwaffe (Klauen des Huhns können kratzen); "Haste" PowerUp für den Huhnbesitzer (er wird doppelt so schnell, aber leuchtet)

Im Laufe der Entwicklung kamen einige Dinge, die Anfangs nicht geplant waren, hinzu. Beispielsweise haben wir viele visuelle Effekte und Sound Effekte eingebaut, um den Funmod Charakter hervorzuheben. So zum Beispiel "gackern" des Huhns oder (blutige) Federn die das Huhn verliert. Auch haben wir auf die Spielintegrität besonderen Wert gelegt, was Anfangs auch nicht so geplant war. So haben wir bei der Implementierung der Multiplayerlogik darauf geachtet, dass keine ungültigen Zustände auftreten können, beispielsweise wenn ein Huhnträger das Spiel verlässt und somit kein Huhn mehr im Spiel ist oder Spieler in den "Spectate" Modus wechseln.

2.1 Zeitplan

Aufgabe	**Woche**											
	1	2	3	4	5	6	7	8	9	10	11	12
existierende Mods erkunden												
Konzept, Aufwandsplanung, Aufgaben												
Recherche SDK und Scripting												
Waffe, Effekten, Sound, Funktionalität												
Waffe, 3D Modell, Texturen, Animation												
Multiplayer-Logik												
Spielintegrität												
GUI, GUI- Events, GUI- Synchronisation												
Punktemodell												
Fairness												
Testen, Bugs beseitigen												
Dokumenation												

Legende			
Im Zeitplan	Vorm Zeitplan A	Vorm Zeitplan E	nicht geplant
Woche überzogen	> Woche überzogen	nicht begonnen	beendet

Wie zu erkennen ist haben wir die Aufgaben, welche geplant waren erfüllen können, haben aber keine große neue Aufgabe hinzugefügt, lediglich einige Teilaufgaben weiter detailliert.

Lediglich das 3D Modell und die Anfangs gewünschten Animationen konnten wir nicht realisieren. Die Gründe dafür sind weiter unten ersichtlich. Wir haben diese Aufgabe aber dahingehend zufriedenstellend gelöst, dass wir ein Huhn haben, was brauchbar aussieht aber keine Animationen enthält.

3 Die Benutzung des Mod

3.1 Voraussetzungen

Da es sich hier um einen Mod für Quake 4 handelt benötigt man selbstverständlich das Spiel Quake 4. Zusätzlich benötigt man den Patch 1.3 von ID Software[2] für Windows oder Linux.

Wir haben den Mod bisher nur mit der englischen Originalversion getestet und entwickelt und können somit nicht garantieren, dass er auch mit der deutschen Version funktioniert.

3.2 Installation

Zusammen mit dieser Dokumentation findet man ein Archiv im ZIP- Format. Dieses Archiv enthält den eigentlichen Mod. Es ist ausreichend dieses Archiv in das Quake Installationsverzeichnis zu entpacken. Anschließend findet man im Quake Verzeichnis ein neues Verzeichnis namens "chicken_mod" mit vier Dateien.

Verzeichnis: chicken_mod

chicken_mod.pk4	Skripte, Artwork, Sounds, Modelle
description.txt	Beschreibung in Textform
game000.pk4	Binäre Spielbibliothek für Windows
game100.pk4	Binäre Spielbibliothek für Linux

3.3 Spielen

Mit dem entpacken des Archivs in das Quake Verzeichnis ist die Installation bereits abgeschlossen. Nun kann man das Spiel starten.

Startet man nun das Spiel kann man den Mod wie üblich über den "Mods" Dialog im Spiel direkt auswählen. Ist der Mod gestartet kann man einen Server starten (oder auf einen Catch The Chicken Server verbinden) und einem Spiel beitreten.

Im Singleplayer- Modus hat der Mod keine Auswirkungen. Startet man also ein neues Singleplayer-Spiel so spielt man das reguläre Quake.

Alternativ besteht die Möglichkeit den Mod auf der Kommandozeile bzw. mittels einer Verknüpfung direkt zu starten. Dazu startet man Quake mit folgenden Optionen:

Mod direkt starten

```
quake4 +set fs_game "chicken_mod"
```

Hinweis:

Wie bei jedem Mod für Quake besitzt jeder Mod eigene Einstellungen für Eingabe, Grafik, Audio und Netzwerk. Unter Umständen muss man also nach dem ersten Start des Mods noch einige Einstellungen in Quakes "Settings" Dialog vornehmen.

[2] http://www.idsoftware.com

4 Mod Internals

In diesem Abschnitt soll es um die Implementierung des Mods gehen. Es werden einige Details erläutert, wie der Mod realisiert wurde. Allerdings können wir hier nur auf einige Dinge eingehen, keineswegs aber auf die Details. An gegebener Stelle wird aber erwähnt wie man im Code unsere Modifikationen ausfindig machen kann.

Alle Dateien, die wesentlich für den Mod sind, befinden sich im ZIP- Archiv `chicken_mod.pk4`. Dort befinden sich alle Artworks, Sounds, Definitionsskripte und Models. In dieser Dokumentation werden wir auf die wichtigen dieser Dateien eingehen und erläutern in welchem Zusammenhang sie wichtig werden. Des weiteren befinden sich die Änderungen an der SDK in binärer Form in den Spielbibliotheken.

Zu erwähnen ist noch, dass der Mod entsprechend dem Quake Patch mit der Version 1.3 der Quake 4 SDK entwickelt wurde.

Hinweis:

Die Veränderungen im Code wurden alle mit einem umschließenden Kommentar gekennzeichnet:

```
//>>> CTC
                ...
//<<<
```

4.1 Die Waffe

Verantwortlicher: Sebastian Eichelbaum – Tools: VIM

In unserem Mod wurde das Huhn als Waffe im Spiel realisiert. Dazu musste zunächst in der SDK im Verzeichnis "`game/weapon/`" eine neue Datei erstellt werden, welche die neue Klasse beherbergt. Diese Klasse wurde zunächst von der bereits vorhandenen Klasse "`rvWeaponGauntlet`" abgeleitet. Die neue Klasse "`rvWeaponChicken `" in der Datei "`WeaponChicken.cpp`" wurde dabei um einige Funktionen erweitert:

`rvWeapinChicken`

- `rvWeaponChicken::Raise()`
  - Aufgerufen wenn die Waffe aufgenommen wird
  - Flag im Besitzer ("idPlayer"- Instanz) gesetzt
- `rvWeaponChicken::PutAway()`
  - Aufgerufen wenn die Waffe gewechselt wird
  - modifiziert um Wechsel zu verhindern
- `rvWeaponChicken::OwnerDied()`
  - Aufgerufen wenn Besitzer stirbt
  - Flag im Besitzer ("idPlayer"- Instanz) zurückgesetzt

Waffen sind in Quake wie eine State Machine aufgebaut. Der Wechsel zwischen den Zuständen kann dabei manipuliert werden. So haben wir verhindert, dass der Hühnchenträger sein Huhn ablegen oder zu einer anderen Waffe wechseln kann. Außerdem wurden vorhandene Funktionen

die für die Feuer und Idle Zustände verantwortlich sind erweitert. Sie wurden so manipuliert, dass die Huhnwaffe permanent "gacker" Geräusche von sich gibt oder ständig Federn verliert. Auf die Effekte der Partikelengine soll später noch eingegangen werden.

Korrespondierend zu dieser neuen Klasse in der SDK muss diese Waffe noch in den Spielbeschreibenden Skripten eingebaut werden. An dieser Stelle kommen die Dateien "def/Player.def" und "def/weapons/chicken.def" ins Spiel.

So muss in "def/Player.def" für den Spieler ein Slot im Inventar erstellt werden, indem die Waffe zukünftig liegen soll:

Auszug aus "def/Player.def"

```
entityDef player_marine
{
  ...
  "def_weapon11" "weapon_chicken"
  "ammo_chicken"          "-1"
  ...
}
```

Allgemein ist die Aufgabe von "def/Player.def" die Definition von verschiedenen Eigenschaften eines Spielers im Singleplayer und Multiplayer- Modus. So zum Beispiel Healthwerte, Waffenslots, Ausgabetexte und verschiedene Spieleranimationen und Sounds.

Man erkennt, dass die Waffe im Skriptsystem "weapon_chicken" heißt. Damit das Skriptsystem nun diesen Namen zuordnen kann muss eine neuen Waffen Entität erstellt werden, welche die Klasse in der SDK, die die Funktionalität implementiert, mit einem Namen im Spiel verbindet. Dazu wurde eine Definitionsdatei für die neue Waffe erstellt:

Auszug aus "def/weapons/chicken.def"

```
entityDef weapon_chicken
{
  ...
  "weaponclass"       "rvWeaponChicken"    // neue Klasse in der SDK
  "inv_weapon"        "weapon_chicken"     // Name des Inventarslot
  ...
}
```

Des weiteren wurde in dieser Datei eingestellt, welche Munition die Waffe nutzt (keine Munition), welche Damagewerte sie hat, welche Waffenanimationen genutzt werden sollen, welche FX Dateien für die Partikeleffekte genutzt werden sollen und so weiter.

Detaillierte Informationen findet man entsprechend in den Definitionsdateien "def/weapons/chicken.def" sowie "def/player.def" in "chicken_mod.pk4" (zu öffnen mit einem ZIP Programm) und den Code Dateien der SDK.

4.1.1 Waffeneffekte

Wir haben die Waffe mit einigen speziellen Effekten ausgestattet. Hier seien speziell die Partikeleffekte hervorgehoben. So gibt das Huhn während es in der Hand des Spieler ist ständig Federn ab und gackert. Benutzt der Spieler das Huhn als Nahkampfwaffe wird ein weiterer

Partikeleffekt abgespielt der blutige Federn fallen lässt. Diese Effekte sind mittels Quake's Partikelengine realisiert. Diese Engine lässt sich über Konfigurationsdateien steuern, welche in der Datei "chickenmod.pk4" im Verzeichnis "effects/weapons/chicken" zu finden sind. Dabei gibt es verschiedene Dateien für die Verschiedenen Zustände der Waffe (also Idle, Feuer usw). Auf die Funktionsweise sei hier nur kurz eingegangen. Zunächst ein Ausschnitt aus einer solchen Datei:

Auszug aus "effects/weapons/chicken/impact.fx"

```
effect effects/weapons/chicken/impact
{
  ...
  emitter "federvieh"
  {
    sprite
    {
      ...
      material  "gfx/effects/particles_shapes/feather.tga"
      start
      {
        velocity { box 10,-5,-5,60,5,5 }
        acceleration { point -20,0,0 }
        ...
      }
      ...}...}...}
```

Man sieht wie hier ein Partikel definiert wird. Es wird die Textur für den Partikel festgelegt, die initiale Geschwindigkeit, Beschleunigung, Anzahl an Partikeln, Bewegungsrichtung und so weiter. Es ist möglich diverse `emitter` zu kombinieren und so komplexe Effekte auszulösen. Diese Effekte müssen allerdings an gegebener Stelle in der Spiellogik gestartet werden. Dazu ein Auszug aus dem SDK Code:

Auszug aus "game/weapons/WeaponChicken.cpp"

```
if (!impactEffect)
{
  impactMaterial = tr.c.materialType ? tr.c.materialType->Index() : -1;
  impactEffect = gameLocal.PlayEffect ( gameLocal.GetEffect ( spawnArgs,
                 "fx_impact", tr.c.materialType ), tr.endpos,
                 tr.c.normal.ToMat3(), true );
}
else
{
  impactEffect->SetOrigin ( tr.endpos );
  impactEffect->SetAxis ( tr.c.normal.ToMat3() );
}
```

Dieser Code zeigt, wie ein Federeffekt ausgelöst wird, insofern er noch nicht ausgelöst wurde. Wurde er bereits ausgelöst wird lediglich die Position und Ausrichtung aktualisiert. Das Auslösen eines solchen Effekts ist gebunden an die Angabe einer Position und einer Normalen. Diese

Normale und der Punkt wird aus einem `TracePoint` ermittelt, welcher für beliebige Objekte im Spiel eine Gerade ermittelt, die sie verbindet.

Für Details sei auch hier wieder auf die Entsprechenden Dateien im "`chicken_mod.pk4`" Archiv und der SDK verwiesen.

4.2 Spiellogik und Architektur

Verantwortlicher: Lars Kolb – Tools: MS Visual C++ .Net und Notepad

Beim Spielstart, `rvGameState::NewState(GAMEON)`, spawnt der Server das Chicken zufällig bei einem Spieler, so dass dieser es sofort aufnehmen kann. Alle Spieler werden per Gui über alle "Chicken-Events" (Pickup, Punkte, Drop) informiert (siehe "Gui Moding.").

Während des Spiels testet der Server periodisch in `idMultiplayerGame::CommonRun()`, wer der momentane Besitzer des Huhnes ist, und reagiert dementsprechend. Hat es einen neuen Besitzer, wird dessen `ClientId` und die Zeit der Aufnahme gespeichert sowie seine Punktzahl erhöht. Weiterhin bekommt er ein das Powerup "Haste", welches ihn doppelt so schnell macht und leuchten läßt. Ist eine bestimmte Zeitspanne, während der selbe Spieler das Huhn besitzt, abgelaufen, werden ihm ebenfalls Punkte gutgeschrieben. Ist der gespeicherte Client nicht mehr im Besitz des Huhnes, so wird seine `Id` auf dem Server zurückgesetzt und ein neues Huhn an seiner letzten Position gespawnt.

Zum Ende eines Spieles werden vom Server sämtliche sämtliche Spielerflags auf `false` gesetzt um diese Informationen in einem folgenden Spiel nicht zu verwenden.

Um Disconnects von Spielern, die im Besitz des Huhns sind zu behandeln, testet der Server in `idMultiplayerGame::CheckAbortGame()`, was nach einem Disconnect, einem Timeout und dem Wechseln eines Spielers zum Spectator aufgerufen wird, ob noch ein Spieler das Huhn besitzt. Ist dies nicht der Fall, spawnt er es an der letzten bekannten Position des ursprünglichen Besitzers.

Hier wird nur Auszugsweise auf einige Details eingegangen. Für den kompletten Code muss man nur einen Blick in "`game/MultiplayerGame.cpp`" der SDK werfen.

Im Folgenden sind die Änderungen an der Methode `idMultiplayerGame::CommonRun()`, in welcher der Schwerpunkt der Modlogik liegt, angegeben:

Auszug aus "game/MultiplayerGame.cpp"

```
...
gameLocal.setChickenOwner(player);
gameLocal.setChickenDroped(false);
gameLocal.setTimeDummy(gameLocal.time);
gameLocal.setChickenOwnerYaw(player->viewAngles.yaw);
gameLocal.setChickenOwnerOrigin(player->GetPhysics()->GetOrigin());

AddPlayerScore(player, gameLocal.PICKUP_POINTS);
if(gameLocal.gameType == GAME_TDM)
  AddTeamScore(player->team, gameLocal.PICKUP_POINTS);

player->GivePowerUp(POWERUP_HASTE, -1, false);
...
```

Hier sieht man wie Serverseitig einem Spieler das Huhn zugewiesen wird, wenn noch kein Spieler im Besitz des Huhnes ist und wie entsprechende Flags in `gameLocal` gesetzt werden. Diese Flags werden später bei der Abfrage auf Serverseite wieder wichtig. Außerdem werden die Punkte verteilt und das `HASTE` Powerup.

Im Folgenden Abschnitt sei noch kurz ein Ausschnitt aus der Nachrichtenverteilung aufgezeigt. Man erkennt wie `RELIABLE_MESSAGES` über Ereignisse mit dem Besitzer des Huhns verteilt werden, damit auf den Clients zum Beispiel die GUI über den besitzer informiert wird.

Auszug aus "game/MultiplayerGame.cpp"

```
...
idBitMsg       outMsg;
...
outMsg.WriteByte( GAME_RELIABLE_MESSAGE_CHICKEN_EVENT );
outMsg.WriteByte( gameLocal.CHICKEN_PICKUP );
outMsg.WriteString(chickenOwnerName);
networkSystem->ServerSendReliableMessage( -1, outMsg );
...
break;
```

Die nun folgenden Codeausschnitte beziehen sich auf den `else` Block von

`if(gameLocal.getChickenOwner() == NULL)`

und stellen die Logik bereit, die die Punkteverteilung übernimmt und die Behandlung eines "`DROP`" Ereignisses, wenn also das Huhn gedroppt wurde.

Auszug aus "game/MultiplayerGame.cpp"

```
if(player->getHoldsChicken())
{
  ...
  if( MS2SEC(gameLocal.time-gameLocal.getTimeDummy())
      >=gameLocal.HOLD_SECS)
  {
    /* POINTS */
    gameLocal.setTimeDummy(gameLocal.time);
    AddPlayerScore(player, gameLocal.HOLD_POINTS);
    if(gameLocal.gameType == GAME_TDM)
      AddTeamScore(player->team, gameLocal.HOLD_POINTS);
    ...
  } //endif time
} //endif holdsChicken
```

Zu sehen ist die Verteilung von Punkten nach einem gewissen Zeitintervall. Nicht zu sehen ist, dass an dieser Stelle wieder eine `GAME_RELIABLE_MESSAGE_CHICKEN_EVENT` Nachricht verschickt wird, um die Clients über den neuen Punktestand zu informieren.

Auszug aus "game/MultiplayerGame.cpp"

```
/* CHICKEN DROPED */
gameLocal.setTimeDummy(-1);
gameLocal.setChickenOwner(NULL);
gameLocal.setChickenDroped(true);
...
networkSystem->ServerSendReliableMessage( -1, outMsg );
...
/* spawn new chicken at the former owners last position */
gameLocal.spawnChicken();
```

Der letzte Codeausschnitt zeigt nun abschließend noch den Fall eines gedropten Huhnes. Es werden die entsprechenden Flags, welche oben in "gameLocal" gesetzt wurden, zurückgesetzt und wieder eine Nachricht an die Clients geschickt. Anschließend wird ein neues Huhn an der letzten Position gespawnt.

4.2.1 GUI

Wie bereits erwähnt, werden alle Spieler über die relevanten Events per Gui informiert. Dazu wurde die Datei `<quakeDir>chicken_mod/chicken_mod.pk4/guis/mphud.gui`, ursprünglich zu finden in `<gameDir>/q4base/pak001.pk4` erweitert.

Das verändern der Gui des Spielers erfolgt auf der Seite des Clients. Dabei werden Hauptsächlich drei Strings gesetzt: Der aktuelle Besitzer des Huhns, die vergangen Zeitspanne seit er es besitzt und die Art des Events. Bei der nächsten Aktualisierung der Gui malt die Engine die neuen Werte. Das setzen der Strings wird durch den Server mit Hilfe von `idGameLocal::ServerSendReliableMessage(...)` in Game_network.cpp initiiert und auf der Client Seite von `idGameLocal::ClientProcessReliableMessage(...)` ausgewertet.

4.3 Huhn Modell und Animation

Verantwortlicher: Patrick Oesterling – Tools: Milk Shape 3D, Paint Shop Pro, Dev C++

Da es weder im Interesse der Gruppe noch für die Umsetzung unseres eigenen Mods notwendig war, haben wir uns entschieden, kein eigenes Chicken-Model zu designen, sondern ein bereits vorhandes Model zu verwenden. Nach vergeblicher Suche nach brauchbaren Models, blieb als letzte Alternative das Chicken-Model aus dem "Catch the Chicken"-Mod von Quake3. Um dies zu extrahieren, benutzen wir das Modellierungsprogramm Milk Shape 3D, welches in der Lage ist, die MD3-Dateien von Quake3 zu lesen und sie im proprietären Format MilkShape 3D ASCII zu exportieren, welches widerrum die Grundlage für ein kleines selbstprogrammiertes Konvertiertool bildet, um die für das Model erforderlichen md5anim- und md5mesh Dateien zu erstellen.

Da diese MD5-Dateien auch im ASCII-Format geschrieben wurden, beschränkt sich diese Konvertierung darauf, die erforderlichen Daten wie Vertices, Dreiecke, Knoten plus Gewichte und Textur-Koordinaten aus der exportieren Datei zu filtern und MD5-spezifische Angaben wie Quaternionen, Boundingboxen und verschiedene Bone- und Framewerte standardmäßig zu definieren.

Weiterhin verlangt die Quake4 Engine ein Model im LWO-Format, welches angezeigt wird, wenn das entsprechende Item nicht vom Spieler benutzt wird, sondern als dropable Item irgendwo

auf der Map angezeigt werden soll. Diese LWO-Datei kann auch mit Milk Shape 3D exportiert werden und wird dann in der `def/weapons/chicken.def` deklariert.

Um das Model mit einer Textur auszustatten, muss diese einerseits in der LWO-Datei mit Pfadangabe bzgl. der Mod-Paketstruktur als auch in den md5mesh-Dateien (sowohl für world- als auch ego-Perspektive) als shader-Attribut deklariert werden. Dafür verlangt die Quake4 Engine eine Material-Datei (*.mtr), welche im `materials` -Verzeichnis der Paketstruktur anzulegen ist. Diese Datei verweist dann auf die eigenliche Textur ("diffuse map") und hat mindestens folgenden Umfang:

Auszug aus "materials/chicken.mtr"

```
models/weapons/chicken/world
{
        noSelfShadow
        qer_editorimage          models/weapons/chicken/world/chicken_d.tga
        diffusemap               models/weapons/chicken/world/chicken_d.tga
}
```

Die eigentliche Texturdatei `chicken_d.tga` kann dabei nach belieben mit einem Bildbearbeitungsprogramm verändert werden. Bei den üblichen Waffen und Items liegt die Textur zur Verbesserungs der Detailgenauigkeit weiterhin als `bumpmap`, `heightmap`, `normalmap` und `specularmap` vor. Wir haben uns allerdings nur auf die `diffusemap` beschränkt.

Desweiteren verlangt die Doom3-Engine sämtliche Angaben von Rotationen und Orientierungen mittels Quaternionen, weil diese kompakter, effizienter und stabiler als herkömmliche orthogonale 3x3 Matrizen sind und weil durch die Repräsentation durch Quaternionen u.a. der in `yaw-pitch-roll-Winkelsystemen` möglicherweise auftretende `"gimbal lock"` vermieden wird, bei dem sich bestimmte hintereinander ausgeführte Rotationen gegenseitig beeinflussen,was zu falschen Ergebnissen führt. Da wir auf eine Einarbeitung in existierende extrem umfangreiche 3D Modellierungsprogramme verzichtet haben, welche alle Rotationen und Orientierungen in Quaternionenangabe expotieren können, mussten wir diese Quaternionen manuell in die md5anim-Dateien einfügen, da diese dort u.a. dafür benötigt werden, um Veränderungen (z.B. Drehungen) in den Animationsframes bzgl. eines Baseframes zu repräsentieren.

Dafür haben wir uns ein kleines Programm geschrieben, welches aus der Angabe der drei Rotationswinkel um die entsprechenden Achsen ein Einheitsquaternion (mit der Norm 1) berechnet, dessen drei komplexe Anteile dann direkt in die MD5-Dateien geschrieben werden. Anzumerken sei hier, dass es sich bei der Doom3-Engine implizit um Einheitsquaternionen handelt, da nur diese eine Drehung repräsentieren (deshalb wird der reelle Anteil auch nicht mit abgespeichert, weil sich dieser aus der impliziten Norm von 1 zurückrechnen lässt).

Auszugsweise hier zunächst die Definition eines Quaternion in `C++`:

Definition des Quaternion

```
struct Quaternion {
  double x,y,z,w;

  Quaternion(double _w, double _x, double _y, double _z)
  {
    x = _x; y = _y; z = _z; w = _w;
  }

  void mult(Quaternion q)
  {
    float wTmp, xTmp, yTmp, zTmp;
    wTmp = w*q.w - x*q.x - y*q.y - z*q.z;
    xTmp = w*q.x + x*q.w + y*q.z - z*q.y;
    yTmp = w*q.y - x*q.z + y*q.w + z*q.x;
    zTmp = w*q.z + x*q.y - y*q.x + z*q.w;

    w = wTmp;
    x = xTmp;
    y = yTmp;
    z = zTmp;
  }
```

Dieses `struct` wurde nun benutzt um die Berechnungen entsprechend durchzuführen. Dazu hier die Berechnungen in verkürzter Form:

Berechnung der Quaternionen

```
  ...
  double xAngle = 0.0 * 3.141592654 / 180.0; // input here
  double yAngle = 0.0 * 3.141592654 / 180.0; // input here
  double zAngle = 0.0 * 3.141592654 / 180.0; // input here

  Quaternion qXRotation( cos(xAngle / 2.0), sin(xAngle / 2.0), 0, 0);
  Quaternion qYRotation( cos(yAngle / 2.0), 0, sin(yAngle / 2.0), 0);
  Quaternion qZRotation( cos(zAngle / 2.0), 0, 0, sin(zAngle / 2.0));

  Quaternion qRotationResult(1.0, 0.0, 0.0, 0.0);
  qRotationResult.mult(qXRotation);
  qRotationResult.mult(qYRotation);
  qRotationResult.mult(qZRotation);
  ...
```

Durch diese manuellen Angaben ist es leider auch nur mit erheblichen Aufwand möglich Animationssequenzen zu realisieren, welche man aber sinngemäß nur unter Angabe von verschiedenen Winkeln und Positionen interpolieren bräuchte, um z.B. das Senken oder Heben des Chickens zu realisieren. Es sei nochmals darauf verwiesen, dass es dafür spezielle Programme gibt (z.B. Maya) mit welchen solche Animationen problemlos realisiert werden können und welche die mathematischen Details verbergen.

Letzten Endes werden alle benötigen Dateien für drei Models verwendet die in der `chicken_mod.pk4` unter `models/weapons/chicken` zu finden sind:

- ein LWO-Model, wenn das Chicken gedroppt wurde
- ein worldView-Model, was man sieht, wenn ein anderer Spieler das Chicken trägt
- ein view-Model, was man sieht, wenn man selbst das Chicken trägt (ego-Perspektive)

5 Probleme

Die Hauptschwierigkeit bei der Entwicklung des Mods bestand in der absolut mangelhaften Dokumentation. Dies trifft sowohl auf den Code als auch auf die Definitionsdateien zu. Aus diesem Grund waren wir 60% der Entwicklungszeit damit beschäftigt, diverse Foren (hauptsächlich für Doom 3) zu durchsuchen und Werte auf ihre Auswirkungen zu testen. Bezeichnend für dieses Problem ist die offiziele Quake 4 SDK Dokumentationsseite, auf welcher eine Vielzahl von Seiten "under construction" ist und zum Code gar keine Informationen vorhanden sind. Nicht einmal Kommentare im Code sind vorhanden. Gut dokumentiert sind lediglich Themen rund ums Mapping und Leveldesign, was für uns jedoch irrelevant war.

Weiterhin kam erschwerend hinzu, dass alle verfügbaren Mods, welche die Spiellogik verändern, als Binaries vertrieben werden, so das für uns keine Möglichkeit bestand von den Erfahrungen der Moder zu profitieren.

Sehr zeitraubend war das Testen der Modifikationen, da ein Start des Spiels rund eine Minute dauert. In dem Zusammenhang kommt auch folgendes Problem zum tragen: Quake stürzt bei falschen Variablennamen oder Einstellungen in Definitonsdateien, FX Dateien, Materials usw. ohne jegliche Fehlermeldung ab.

Diese Probleme konnten wir nur durch intensives nachlesen des Quake SDK Codes, durch endloses Probieren und diverse Internetforen[3] lösen.

Wie im Abschnitt "Huhn Modell und Animation" bereits zu erkennen war, war auch der Mangel an Spezialsoftware ein Problem. Man kann davon ausgehen, dass Models und deren Animationen nur mit professioneller und teurer 3D Modellierungssoftware vernünftig realisierbar sind. Aber selbst wenn diese Software zur Verfügung stände wär die Einarbeitungszeit zu hoch.

6 Bekannte Probleme

Zur Zwischenpräsentation waren noch einige Bugs und Probleme vorhanden. Diese konnten wir in den letzten Wochen lösen und beseitigen. Ein solches Problem war zum Beispiel, dass beim Huhnträger nach dem Spawnen der `Health` Wert permanent bis 0 dekrementiert wird. Dieses Problem konnten wir beseitigen.

Wenn der Hünchenträger stirbt und ein anderer Spieler das Huhn aufnimmt, so wurde manchmal das Huhn nicht als Waffe automatisch übernommen. Man musste die Waffe wechseln und das Huhn auswählen. Dieser Fehler ist beseitigt. Allerdings tritt er noch immer dann auf, wenn man innerhalb der Zeitspanne eines Snapshots, welchen Quake zur Synchronisation überträgt, zwei Waffen (also das Huhn und eine andere Waffe) aufnimmt. Dieses Problem tritt allerdings nur sehr selten auf (da nur sehr selten das Huhn und eine Waffe so eng nebeneinander liegen) und beeinflusst das Spielerlebnis nicht. Die Lösung des Problems wär eine komplette Überarbeitung des Snapshot- Mechanismus der Engine, was aber mit extremen Aufwand verbunden ist.

[3] http://www.doom3world.org/phpbb2