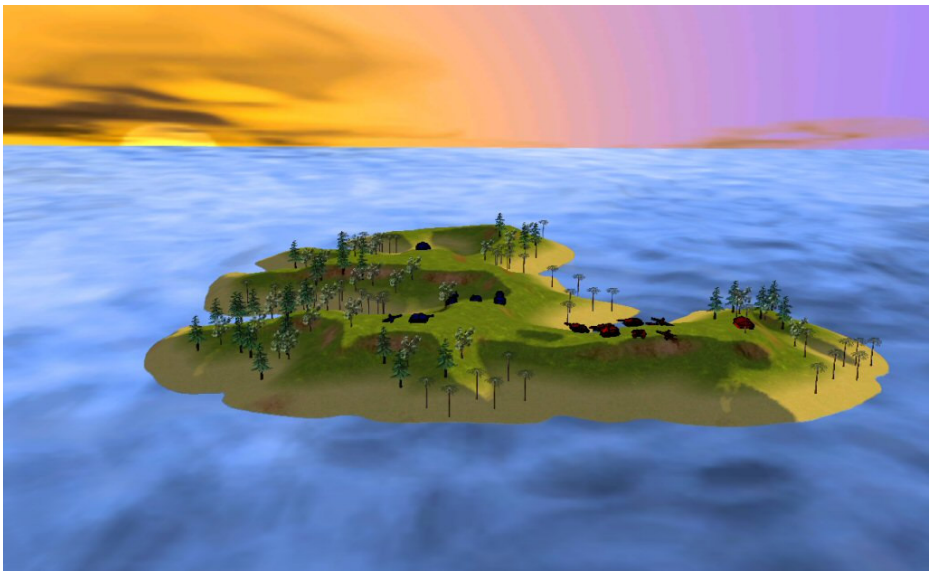
ACK-ACK!

a round based multiplayer strategy game

CG23 Endabgabe



Berger Wolfgang, 532, 0225152, thewulf@gmx.net
Buchetics Matthias, 532, 0225149, mat.buchetics@gmx.net

Zusammenfassung Storyboard

ack-ack [ifml.] .....das Flakfeuer (bes. 2. Weltkrieg)

Ack-Ack ist ein rundenbasiertes Strategiespiel und orientiert sich dabei spielerisch an Klassikern wie Battle Isle oder Advance Wars (GBA).
Das Spiel baut dabei auf einen 2-Spieler Modus auf, beide Spieler ziehen abwechselnd ihre Einheiten.
Wichtig ist uns dabei ein einfach verständliches, schnelles, aber taktisch forderndes Spielsystem. Alle Einheiten werden mit wenigen Parametern beschrieben (Hitpoints, Angriffspunkte, Verteidigungspunkte, Angriffsreichweite), damit soll es leicht möglich sein, neue Einheiten dem Spiel hinzuzufügen. Zusätzlich zu den Fähigkeiten der Einheiten kann der Spieler die Landschaft ausnutzen, um sich einen Vorteil gegenüber seinem Gegner zu erarbeiten. So greifen Einheiten von einer erhöhten Position aus stärker an und auch das Terrain (Wald,...) kann (und soll) taktisch genutzt werden.

Kurzbeschreibung

Ack-Ack lässt zwei Teams abwechselnd gegeneinander antreten. Das Ziel ist es, alle gegnerischen Einheiten oder die Basis des Gegners zu zerstören und dabei die taktischen Vorteile des Terrains bzw. der einzelnen Einheiten auszunuten. Das aktuelle Team zieht all seine Einheiten und führt mit ihnen eventuelle Angriffe durch. Hat man alle Einheiten "aufgebraucht", so ist das andere Team dran - sofern noch was davon übrig ist...

Alle notwendigen Informationen über Spielfeld, Einheitenstatus, Gesamthitpoints usw. liefert das grafische Interface welches sich dynamisch der Spielsituation anpasst.

Es können nur Einheiten ausgewählt werden, die dem aktuellen Team angehören und noch nicht gezogen wurden. Bei der jeweils ausgewählten Einheit werden im "Move"-Modus die befahrbaren Felder eingeblendet. Die eingeblendete Grid wird außerdem farblich codiert, sodass höhere von niedrigeren Ebenen leicht unterschieden werden können.
Der Bewegungsradius ist neben den Fähigkeiten der Einheiten (der Jeep kann sich deutlich weiter bewegen als etwa der Panzer) auch vom Terrain abhängig. Ein Panzer kann sich z.B. nicht auf Sand bewegen, der Jeep hat mit diesem Untergrund keine Probleme. Dasselbe gilt für die Steilheit des Terrains.

Wechselt man in den "Fire"-Modus, wird stattdessen der Feuerradius von der aktuellen Position aus rot markiert. Feindliche Einheiten in diesem Bereich können durch Auswählen angegriffen werden. Den Kampfausgang spiegelt die Hitpointanzeige der Einheit wieder. Ein Angriff ist effektiver, wenn man von einer erhöhten Position aus angreift. Der Attackierte schießt immer zurück. Die Effektivität eines Angriffs wird von den Hitpoints der Einheiten (je mehr Hitpoints eine Einheit hat, desto stärker ist sie), dem Höhenwinkel zum Verteidiger sowie vom Terraintyp (eine Einheit im Wald kann sich besser verteidigen, jedoch können z.B. Panzer nicht in den Wald fahren) beeinflusst.

Der Spieler erhält nicht nur über das Interface, sondern auch direkt in der Spielgrafik Rückmeldungen über den Status der Einheiten. Ist ein Angriff effektiv, d.h. zieht er dem Gegner eine gewisse Anzahl von Hitpoints ab, so wird eine „Splitteranimation“ aufgerufen. Sinkt die Anzahl der Hitpoints einer Einheit unter einen kritischen Wert, so wird die Beschädigung mit Rauch und später mit Feuerflammen dargestellt. Ist eine Einheit zerstört erfolgt eine Explosion und die Einheit verschwindet vollständig vom Spielfeld.

Nach einem Zug wird die eben benutzte Einheit halbtransparent dargestellt und kann erst in der nächsten Runde wieder ausgewählt werden.

Einen besonderen Einheitentyp stellt der „General“ dar. Dieser kann sich nicht nur sehr schnell bewegen, sondern auch eigene Einheiten unterstützen. Maximal einer eigenen Einheit kann er pro Runde mittels des Angriffsbefehls seine Unterstützung zukommen lassen. Wie sich diese Unterstützung auf die jeweilige Einheit auswirkt ist unterschiedlich. In unserer Spielversion erlangen Panzer z.B. einen gesteigerten Angriffswert, Artillerieeinheiten dagegen einen größeren Feuerradius.
Unterstützt der General in der Runde eine Einheit, so kann er keine gegnerische Einheit angreifen. Es ist also immer zu entscheiden, ob man mit dem General aktiv oder passiv in das Kampfgeschehen eingreifen will.

Ebenfalls eine Spezialeinheit ist das Headquarter, welches jedes Team besitzt. Diese „Einheit“ kann weder bewegt werden noch sonst irgendwie in das Spiel aktiv eingreifen. Ist es allerdings zerstört, so ist das Spiel für den Spieler sofort verloren. Es gilt daher neben den Angriffsüberlegungen auch immer das eigene Headquarter im Blick zu behalten.

Programmstart

Das Spiel wird mit Doppelklick auf AckAck.exe gestartet. Im daraufhin erscheinenden Konfigurationsdialog kann im Drop-Down die Auflösung, Farbtiefe und Bildwiederholrate eingestellt werden. Dies ist jedoch nur von Bedeutung, wenn das Häkchen bei Fullscreen aktiviert ist. Läuft das Spiel im Fenstermodus, so wird eine Auflösung von 640x480 gewählt.

Bei den Advanced Einstellungen kann man die Einblendung der Framerate, die Ausgabe von Debuginformationen am Screen, sowie die verwendeten Extensions einstellen.
Letztere Einstellungen sind nur zum Testen der jeweiligen Renderpfade gedacht und sollten immer aktiviert sein. Falls bestimmte Extensions nicht unterstützt sind, werden sie ohnehin im Programm deaktiviert. Außerdem können Musik und Soundeffekte ausgeschalten werden.

Wichtig: Die aufgelisteten Extensions sollten auch aktiviert sein, wenn man im Spiel zwischen den Renderpfaden herumschalten will, sonst ist die Auswahl jener Pfade, die von diesen Erweiterungen abhängig sind, nicht möglich.

Das Spiel kann auch in einem sog. Editor Modus gestartet werden, welcher das Setzen von Landschaftsobjekten und das Verändern von Terraintypen erlaubt. Im Editor Modus ist das Spielen selbst allerdings nicht möglich.

Mit Klick auf "Start" geht es los (hier folgt dann eine kleine Ladezeit).

Steuerung

Einheiten werden mittels linken Mausklick ausgewählt und rechten Mausklick deselektiert (und auf die Ausgangsposition zurückgesetzt). Mit einem weiteren linken Mausklick bewegt sich die Einheit zum Ziel und wechselt dort nach Bestätigung (linke Maustaste) in den Kampfmodus. Befindet sich nun im Angriffsradius eine gegnerische Einheit, so kann diese mit einem Mausklick angegriffen werden. Danach ist die Einheit „aufgebraucht“ und kann in der aktuellen Runde nicht mehr bewegt werden (sie kann sich jedoch noch immer gegen gegnerische Angriffe verteidigen).
Eine Sonderfunktion hat die Generaleinheit. Diese kann nicht nur gegnerische Einheiten sondern auch eigene „angreifen“. Bei einem solchen Angriff wird die eigene Einheit unterstützt (siehe oben).
Mit der rechten Maustaste kann man die jeweilige Aktion abbrechen.

Die aktuelle Runde wird mit einem Klick auf „End Turn“ (rechts unten im Interface) beendet. Nun ist der andere Spieler an der Reihe.

Tastenübersicht:

linke Maustaste	Einheit auswählen/bewegen/fixieren, Angriff starten

rechte Maustaste	Einheit deselektieren, vom Angriffsmodus in den Bewegungsmodus wechseln
mittlere Maustaste	Kamera drehen (mit Mausbewegung)
Space	vom Bewegungsmodus in den Angriffsmodus wechseln
Maus am Bildschirmrand	über die Karte scrollen
Mausrad	Zoomen
W	mit Kamera in Blickrichtung vorwärts (zoom in)
S	mit Kamera in Blickrichtung rückwärts (zoom out)
A	Kamera nach links drehen
D	Kamera nach rechts drehen
R	Kamera nach oben kippen (nicht empfohlen - nur zu Debugzwecken)
F	Kamera nach unten kippen (nicht empfohlen - nur zu Debugzwecken)
Pfeil rauf	mit Kamera vorwärts strafen
Pfeil runter	mit Kamera rückwärts strafen
Pfeil links	mit Kamera nach links strafen
Pfeil rechts	mit Kamera nach rechts strafen
F1	CG23 Infos ein-/ausblenden
F2 – F9	CG23 Einstellungen (siehe Infoeinblendung)
F5 – F12	Editorfunktionen (nur im Editor Modus)
ESC	Spiel verlassen

Implementierung

Entwurf der Datenstrukuren:

- Terrain

Die Höhendaten für die Terraindarstellung werden aus einem Graustufen Bitmap geladen und entsprechend skaliert. Das gesamte Terrain wird in sog. Terrain Tiles aufgesplittet. Jedes dieser Teile umfasst einen bestimmten Bereich des Terrains und wird sowohl in einer eigenen Struktur (Vertexdaten, Texturinformationen) als auch direkt in einem Vertexbuffer Object gespeichert. Normalvektoren werden für das Terrain nicht erzeugt, die Beleuchtung findet über vorberechnete Lightmaps statt.
Die Terrain Tiles werden über einen Quadtree verwaltet (siehe "Hierarchisches Visibility Culling für Terrain" bei den Features).

- 3D Modelle

Mithilfe eines 3DS Loaders (siehe verwendete Libraries) werden die Vertexdaten aus dem 3DS File geladen. Dabei kann ein 3DS Modell aus mehreren einzelnen Objekten bestehen, jeweils mit eigenen Texturen.
Um die Verwaltung von 3D Modellen unabhängig von dem zugrunde liegenden Dateiformat zu gestalten, wurde der 3DS Loader von uns entsprechend in ein objektorientiertes System geändert. Die Datenstruktur mit Vertexdaten, Normalvektoren und Texturkoordinaten bleiben dabei für alle Dateiformate gleich. Auch die Darstellung erfolgt auf einheitliche Weise: bei vorhandener Extension mit Vertexbuffer Objects, ansonsten über Standard OpenGL Calls.

Die Normalvektoren werden nicht generiert, sondern müssen in der Datei festgelegt sein. Im Moment ist lediglich ein Loader/Renderer für statische 3DS Modelle vorhanden.

Alle 3D Objekte werden über einen ObjectManager verwaltet, über welchen Objekte geladen sowie deren Eigenschaften (Position, Skalierung, ...) geändert werden kann.
Der ObjectManager liefert dabei nach dem Laden eines Objekts einen Index zurück. Mithilfe dieses Indexes kann dann global auf das Objekt zugegriffen werden. Das Rendering übernimmt der ObjectManager für alle Objekte ebenfalls. Der ObjectManager achtet außerdem darauf, dass gleiche Dateien nur einmal geladen werden. Die Instanz eines Modells wird dann für mehrere Objekte verwendet.

- Partikelsystem

Auch das Partikelsystem wird über einen Manager verwaltet, die einzelnen Partikel werden in einer Liste gespeichert. Ein Partikel besteht dabei aus einem einfachen Quad mit einer Textur. Weitere Informationen zum Partikelsystem finden Sie im Abschnitt „Features“.

Entwurf eines Kameramodells

Die Kamera arbeitet auf Basis der gluLookAt Funktion. Die einzelnen Funktionen der Kamera (Rotation um alle Achsen, Translation) verändert daher nur die Position sowie den LookAt Punkt. Die Rotation wird mit Quaternions erledigt. Die Bewegung der Kamera ist grundsätzlich völlig frei, für den Spieler aber eingeschränkt (Verschieben, Zoomen, Rotation um Y Achse). Die Steuerung ist außerdem zeitabhängig, die Kamera scrollt daher bei jeder Framerate mit derselben Geschwindigkeit.

Animierte Objekte

Animierte Objekte beschränken sich auf die Bewegung der Spielfiguren. Die Einheiten bewegen sich realistisch über die Landschaft, passen sich dabei z.B. der aktuellen Steigung an und drehen sich in die Richtung ihres Angriffsziels. Die Bewegung ist ebenfalls zeitunabhängig.

Zu den animierten Objekten kann man ev. auch das Partikelsystem zählen, dieses wird an späterer Stelle noch genauer beschrieben.

Implementierung von Texture Mapping

Der TextureManager funktioniert ähnlich wie auch der ObjectManager. Er kapselt sämtliche Laderoutinen und liefert nach dem Laden einer Textur einen Index zurück.
Auch hier werden gleiche Texturen maximal einmal in den Grafikkartenspeicher geladen.
Als Texturformate stehen TGA und BMP zur Verfügung.

Das Texture Mapping wird vom Terrain bzw. der 3D Modellklasse übernommen. Wie schon oben bei den Datenstrukturen beschrieben werden dort die Texturkoordinaten gespeichert. Beim Terrain werden diese beim Programmstart erzeugt, bei den 3D Modellen werden sie aus der entsprechenden Datei geladen.

Beleuchtung/Materialien

Die Beleuchtung des Terrains geschieht über eine, in die Textur hineingerechnete, Lightmap. Die Objekte dagegen werden über ihre Normalvektoren mit Gourad Shading korrekt beleuchtet. Das Setzen der Lichtinformationen übernimmt der ObjectManager für alle Objekte (keine doppelten Aufrufe). Die Normalvektoren werden, wie oben erwähnt, aus den 3DS Dateien gelesen. Um eine uniforme Skalierung der Objekte zu ermöglichen wird GL_RESCALE_NORMAL gesetzt.

Die Einfärbung der Einheiten in rot und blau wird über Materialien realisiert. Dazu setzen wir die ambienten und diffusen Materialwerte entsprechend. Nachdem eine Einheit bewegt wurde, wird sie transparent dargestellt. Auch das geschieht über ein einfaches Material.

Als Effekt haben wir einen Cel Shader implementiert. Ist dieser aktiv (je nach Fähigkeit der Grafikkarte sowie den gewählten Einstellungen) führen wir die Beleuchtung im Vertexshader selbst durch. Mehr zum Cel Shader im Abschnitt „Effekte“.

Spieldesign

Grundsätzlich haben wir Wert darauf gelegt, möglichst alle Gameplayinformationen in XML Dateien zu speichern, sodass diese schnell und einfach geändert werden können.
Es ist daher ohne Probleme möglich neue Einheitentypen in das Spiel zu bringen. Auch alle sonstigen Parameter sind ohne Neukompilierung einstellbar.

Die Spielsteuerung und -logik wird von einer eigenen Klasse übernommen (CGamelevel), die Einheiten werden über einen UnitManager verwaltet (auch hier ist die Funktionsweise ähnlich zu den anderen Managern). Der UnitManager übernimmt auch die Berechnung des Kampfausganges anhand der Eigenschaften der Einheiten sowie dem Terrain.
CGamelevel dagegen kümmert sich um die Korrektheit von Zügen und gibt dem Spieler entsprechende Unterstützung (Einblendung von Bewegungsradius usw.).

Features

Error-based Level of Detail für Terrain:

Um die Landschaftsdarstellung zu beschleunigen, werden verschiedene vorberechnete Detailstufen einzelner Abschnitte verwendet. Das verwendete Detaillevel wird jedoch nicht nur abhängig von der Entfernung zur Kamera bestimmt, sondern berücksichtigt auch den begangenen Fehler bei der Verwendung einer niedrigeren Detailstufe. In der Nähe der Kamera ist der erlaubte Fehler niedriger, während er mit zunehmendem Abstand immer höher wird.

Der Vorteil dieser Methode im Vergleich zum herkömmlichen abstandsbasierten LOD besteht darin, dass bei sehr weichen (nicht stark zerklüfteten) Landschaften niedrigere Detailstufen gewählt werden, obwohl die betroffenen Teile sehr nah bei der Kamera liegen. Dies liegt daran, dass der begangene Fehler aufgrund wenig "verschluckter" Unebenheiten sehr niedrig ist. Außerdem werden stark unregelmäßige Landschaftsabschnitte länger in höheren Detaillevels gehalten und so stark sichtbares Umschalten der Detailstufen reduziert.

Um "Risse" in der Landschaft resultierend aus verschiedenen aufeinandertreffenden Detaillevels zu vermeiden, gibt es zwischen den einzelnen Terrainabschnitten sogenannte Seams, welche zwischen den Detailstufen mappen. Auch diese sind komplett vorberechnet.

Als Darstellungsmethode für das Terrain wurden für Vertices und Indizes Vertex Buffer Objects verwendet.

Auch die eingeblendete Höhengrid verwendet dieses System, wobei die Abschnitte kleiner sind und die Detailstufe vom überblendeten Terrainblock übernommen wird. Hierfür mussten natürlich eigene Vertex- und Indexbuffer berechnet werden, damit es keine Probleme gibt, wenn Seams (teilweise) überdeckt werden.

Zusammen mit dem unten beschriebenen hierachischen Visibility Culling kann auf diese Weise eine Landschaft statt mit über 120000 Polygonen mit ca. 5000 Dreiecken ohne sichtbaren Unterschied dargestellt werden.

Überprüft kann diese Arbeitsweise werden, indem man bei der Landschaftsdarstellung in den Wireframe-Modus wechselt und mit 'W' soweit wie möglich in die Landschaft zoomt. Da die mitgelieferte Heightmap eine sanft geschwungene Landschaft beschreibt, schaltet das System sehr spät auf ein höheres Detaillevel um. Dann sind die höhere Polygonzahl sowie die Seams zu erkennen. Bei einer stark unregelmäßigen Heightmap wäre die Detailstufe schon bei größeren Abständen viel höher.

Hierarchisches Visibility Culling für Terrain

In Hinblick auf eventuell größere Landschaften haben wir uns für hierarchisches Visibility Culling beim Terrain entschieden und verwenden hierfür einen Quadtree. Dieser wird solange traversiert, bis eine eindeutige Entscheidung über die Sichtbarkeit einzelner, immer kleiner werdenden Terrainabschnitte vorliegt, oder ein Blattknoten erreicht ist.

Damit diese Vorgehensweise für kleinere Terrains nicht unnötigen Rechenaufwand aufgrund rekursiver Funktionsaufrufe und Verwaltung von Knoten am Heap hervorruft, wird der gesamte Baum in einem Feld gespeichert und die Rekursion aufgelöst.
Aufgrund dieser Lösung wird die Traversierung erheblich beschleunigt.

Visibility Culling für Objekte

Für jedes Objekt wird nach dem Laden eine Bounding Sphere berechnet. Der Test der Bounding Sphere gegen das View Frustum ist sehr schnell und aufgrund der Skalierbarkeit der Objekte ist eine Sphere in diesem Fall optimal. Gerendert werden am Ende nur vollständig und teilweise sichtbare Objekte.

Partikelsystem

Das Partikelsystem besteht aus zwei großen Komponenten: den Partikeln selbst und den Partikelemittern. Jedes Partikel kann über verschiedene Parameter initialisiert werden, z.B. Lebensdauer, Blending Effekte, Textur usw. Diese Einstellungen verwaltet es selbst und rendert sich (als Quad) ebenfalls selbstständig.

Für die Erzeugung der diversen Effekte (siehe Abschnitt „Effekte“) wird die ParticleManager Klasse verwendet. Über diese können neue Partikelemitter angelegt bzw. vorhandene gesteuert werden. Aus einer XML Datei werden alle nötigen Einstellungen geladen. Für einige Parameter können Bereiche angegeben werden, aus denen dann zufällig ein Wert bestimmt wird (notwendig, damit nicht jeder Effekt immer gleich aussieht).

Das Partikelsystem ist völlig unabhängig vom Rest der Engine, der UnitManager besitzt allerdings „Attach“ Funktionen. Über diese kann ein Partikelsystem an eine Einheit „angehängt“ werden. Bei einer Bewegung der Einheit bewegt sich dann auch das Partikelsystem mit.

Sound/Musik

Für alle Soundeffekte und die Musik verwenden wir die FMOD Library. Für jede Einheit können (wieder in der XML Datei) eigene Soundeffekte verwendet werden. Für die Hintergrundmusik gibt es ein eingeschränkt dynamisches Soundsystem, welches je nach Spielsituation möglichst das passende Musikstück abspielt. Um dies zu erreichen, gibt es ein Stück für den Kampfbeginn, eines für dessen Ende, eine Playlist für Variationen des Main Themes, sowie eine „Bridge“-Thema. Dieses wird verwendet, um für einen nahtlosen Übergang zwischen den Hauptthemen sowie zwischen Haupt- und Endthema zu sorgen.

Großer Dank geht hierbei an Rold und Flo, welche den gesamten Ack-Ack Soundtrack komponiert, geschnitten und dabei wirklich Großartiges geleistet haben. Weitere Informationen hierzu sind im Abschnitt „Zusätzliche Bemerkungen“ nachzulesen.

Interface

Die Textinhalte und –positionen sowie die angezeigten Boxen können komplett extern via XML konfiguriert werden. Auch der verwendete Zeichensatz sowie die gesamte Farbliche Gestaltung lassen sich auf diesem Wege verändern. Der Interfacehintergrund (die Boxen) werden dabei immer farblich an das aktuelle Team angepasst, sodass für den Benutzer immer klar ersichtlich ist, wer an der Reihe ist. Weiters werden alle relevanten Infos für das markierte Terrainstück, die ausgewählte Einheit und die Einheit, über der sich Mauszeiger befindet angezeigt. Um den Überblick besser behalten zu können, gibt es auch noch einen Lebensbalken, welcher die verbleibenden Gesamthitpoints des Teams anzeigen.

Das gesamte Interface wurde selbst programmiert und verwendet für die Standardausgaben gluOrtho2D, um die jeweiligen Elemente am Screen zu positionieren. Weiters gibt es noch spezielle Einblendeffekte für Kommentare zum Spiel. Diese werden im 3D-Raum mittels eigener Perspektive eingeblendet und liefern Infos zum vergangenen Zug bzw. meldet, wenn ein Zug zu Ende ist.

Wegfindung

Um herauszufinden, welche Felder innerhalb des Bewegungsradius einer Einheit erreichbar sind, wurde der Flood-Fill Algorithmus angewandt (4-connected, vertical span). Nicht verwendbare Felder sind solche, die von der Einheit aufgrund des Terraintyps, der Steigung, einer anderen dort platzierten Einheit oder einer allgemeinen Markierung „not useable“ nicht

befahren werden können. Es kann natürlich auch sein, dass dadurch andere – normalerweise verwendbare – Bereiche abgeschnitten werden. Daher eignet sich Flood-Fill ausgehend von der eigenen Einheit ausgezeichnet für diese Berechnung.

Nachdem nun der wahre Bewegungsradius berechnet und ein Ziel für die Einheit ausgewählt wurde, wird der exakte Weg mithilfe des A* Algorithmus berechnet. Der Algorithmus wird dabei auf das Regular Grid der Spielkarte angewandt. Auch wenn es spielerisch keine Auswirkung hat, so können die Kosten der einzelnen Terraintypen angegeben werden, d.h. eine Einheit wird sich eher über eine Wiese als über Sand bewegen.

Offenes Spielsystem

Wie schon mehrmals erwähnt, sind sämtliche Spielinhalte von Ack-Ack nicht hart codiert sondern über XML Dateien steuerbar. Es ist ohne weiteres möglich, komplett neue Einheiten mit neuen Fähigkeiten in das Spiel zu integrieren. Dasselbe gilt auch für Landschaftsobjekte, Terraintypen, Kampfinformationen usw. Selbst das Partikelsystem ist vollständig von außen steuerbar.

Verwendete Extensions:

* GL_ARB_vertex_buffer_object
* GL_SGIS_generate_mipmap
* GL_ARB_texture_compression
* GL_ARB_vertex_program
* GL_ARB_multitexture

Effekte

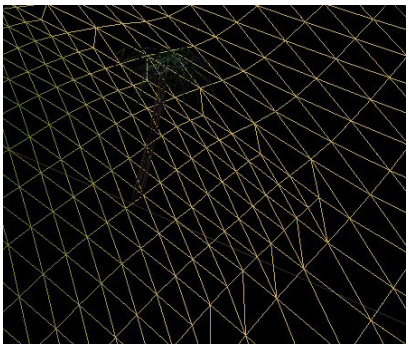
Bewegter Rauch, Rauchspuren, Feuer

Diese vier unterschiedlichen Effekte wurden alle über das Partikelsystem erzeugt. Um die Besonderheiten von Feuer und Rauch verwirklichen zu können, gibt es zwei unterschiedliche Blending Modi. Beim „FIRE“ Modus werden die Partikel mit GL_ONE geblendet, es können damit sehr schöne helle Effekte erzeugt werden. Für Effekte wie Rauch oder Explosionen wird der „SMOKE“ Modus verwendet, dort wird nur der Alphakanal der Textur berücksichtigt.

Wo zu sehen: bei der Bewegung von Einheiten, beim Kampf, am Ende des Spiels.

Literatur: einfache Partikelsysteme auf www.ultimategameprogramming.com, nehe.gamedev.com, allerdings deutlich erweitert (Managerfunktionen, Blending Modi, zusätzliche Parameter bei den Partikeln usw.), vor allem durch viel Experimentieren entstanden

Fehlerbasiertes Level of Detail



Siehe auch „Error-based Level of Detail für Terrain” bei den Features.

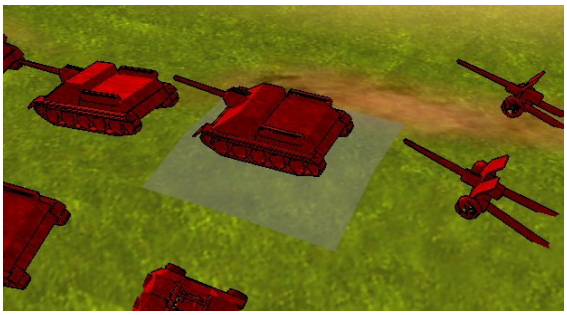
Wir haben das unten angeführte Paper nicht eins zu eins in unser Programm übernommen, sondern unseren Anforderungen angepasst und somit einige Berechnungen einsparen können. So wurde der Fehler nicht in Pixel Error umgerechnet, sondern der vertexbasierte Fehler bestimmt und mit der Distanz eine „Magic Number“ berechnet. Da die Kamera in der Regel einen fixen Blickwinkel hat wurde dieser Wert aufgrund von Erfahrungen ausgewertet und der entsprechende Detaillevel gewählt. Dieser abgewandelte Algorithmus funktioniert allerdings auch bei flacheren Blickwinkeln sehr gut und sorgt (bei statisch ausgeleuchtetem Terrain) für praktisch nicht sichtbare LOD-Switches.

Die Berechnung der Übergänge zwischen den einzelnen Terrainstücken wurde dahingehend erweitert, dass nicht immer die Indices des aktuellen Stückes verändert werden, sondern eigene Seams von jedem Detaillevel auf jedes (vor-)berechnet werden. Allgemein werden sämtliche Indices (Seams, Rest des Feldes) beim Laden des Spieles basierend auf der mitgegebenen Heightmap berechnet, und somit wird während des Spiels nur Rechenaufwand für das Feststellen des Detaillevels bei den sichtbaren Terrainabschnitten benötigt.

Wo zu sehen: Wireframe Darstellung (F3), Kamera bewegen und zoomen (Mausrad), abhängig von der gewählten Heightmap (bei flachem Terrain nicht so gut sichtbar)

Literatur: „Fast Terrain Rendering Using Geometrical MipMapping", Willem H. de Boer, 2000. http://www.flipcode.com/tutorials/geomipmaps.pdf

Cel Shading



Um den Comiclook unseres Spieles zu verstärken, werden die Einheiten mittels Cel Shading beleuchtet. Hierfür wird beim Laden eine eindimensionale Luminance Map mit 16 Einträgen erstellt, bei der die Helligkeitswerte nicht gleichverteilt sind, sondern die hellen Werte anteilsmäßig überwiegen und insgesamt nur vier Helligkeitstufen existieren. Dadurch werden harte Beleuchtungsübergänge erreicht. In einem Vertex Programm werden nun die entsprechenden Texturkoordinaten für die einzelnen Vertices basierend auf den zugehörigen Normalen sowie dem inversen Lichtvektor berechnet. Dies wird durch ein einfaches Punktprodukt zwischen den beiden erreicht, welches in den Bereich zwischen null und eins mapped. Dieser Wert wird dann in die Texturkoordinate des Vertex eingetragen. Wir haben einen derartigen Shader sowohl für (einfach) texturierte als auch für nicht texturierte Objekte erstellt.

Zusätzlich wird noch eine schwarze Outline gerendert, indem das Objekt ein zweites Mal im Wireframe-Modus gezeichnet wird und Culling für Frontfaces aktiviert wird.

Bei den eingeblendeten Spielkommentaren wird für die Outline eine andere Methode verwendet, da die generierte 3D-Schrift einen Aufbau hat, der mit dem oben beschriebenen Prozess Darstellungsfehler produziert. Die Alternative verwendet Stencilbuffering, was wiederum nicht auf die Spielobjekte anwendbar wäre, da sie wirklich nur die Silhouette findet und keine „Ränder" innerhalb des Objektes entstehen. Der Vorgang wurde wie folgt realisiert: Wenn man ein Objekt rendert, schreibt man einen konstanten Wert in den Stencilbuffer. Danach wird das Objekt wieder als Wireframe (Frontfaces) dort ge-

zeichnet, wo der Stencilbuffer nicht gesetzt wurde. Dadurch, dass dickere Linien verwendet werden bleibt eine Outline mit der Hälfte der angegebenen Dicke zurück.

Wo zu sehen: Cel Shading + Outline: auf den Einheiten. Da das Vertex Programm nur beim Einsatz von Vertex Buffer Objects aktiv wird, kann man den Beleuchtungsunterschied einfach feststellen, indem man mittels „F6“ auf einen anderen Renderpfad wechselt.
Outline mittels Stencilbuffering: bei den eingeblendeten Spielkommentaren

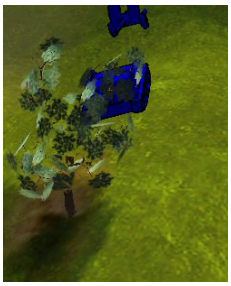
Literatur: „Cel-Shading“, Sami “MENTAL” Hamalaoui, 2001.
http://www.gamedev.net/reference/articles/article1438.asp
www.ultimategameprogramming.com, jedoch erweitert, um mit Vertex Buffer Objects und sich ändernder Kamera zusammenzuarbeiten (Berechnung des Lichtvektors)

Wasser

Wasser wird mittels Multitexturing (2 Texturen) realisiert. Die erste Texturestage wird ohne Animation gerendert, während die zweite durch Transformation der Texturkoordinaten darüber zu „fließen“ scheint.

Falls Multitexturing nicht zur Verfügung steht, wird dies durch zwei übereinanderliegende Quads aufgelöst. Eine Textur wird wiederum ohne Transparenz und ohne Animation gerendert, die andere befindet sich über dieser fixen Textur und wird mit 50%iger Transparenz über die Texturkoordinaten animiert.

Transparenz mit vereinfachter Tiefensortierung



Die Baumobjekte in unserem Spiel enthalten Texturen mit Alphakanal, um die Blätter besser darstellen zu können. Damit Einheiten, die unter Bäumen stehen, trotzdem gesehen werden, werden diese von unserem Objectmanager vor der Vegetation gerendert. Damit haben wir eine Tiefensortierung erreicht, die auf Annahmen bezüglich des Spielinhaltes basiert (Einheiten sind immer näher am Boden als Bäume etc.).

Bei der einblendbaren Grid wurde keine Tiefensortierung vorgenommen, sondern lediglich der Depth Buffer vor dem Rendern dieser Objekte gesperrt.

Wo zu sehen: eine Einheit unter einen Baum bewegen

Objekte

Beleuchtung

Alle 3D Objekte mit Ausnahme des Terrains selbst werden mit einer direktionalen weißen Lichtquelle beleuchtet (von schräg oben). Für das Terrain wird mit einem externen Programm (siehe zusätzliche Anmerkungen) eine Lightmap erzeugt, welche direkt in die Landschaftstextur hineingerechnet wird. Die Partikel werden nicht beleuchtet.

Texturierung

Landschaft, Bäume, Einheiten und Partikel (also alle Objekte im Spiel) sind texturiert. Die Textur des Terrains wird mit einem externen Programm (ebenfalls von uns) anhand der Höhe und der Steilheit generiert. Für die anderen Objekte wurden Texturen gezeichnet und in Milkshape den Modellen zugewiesen.

Tools

Für die 3DS Objekte haben wir hauptsächlich Milkshape verwendet. Die Bäume waren noch aus einem älteren Projekt vorhanden, die sonstigen Objekte (Einheiten, Basis) stammen aus dem Internet und wurden von uns teilweise in der Polygonanzahl reduziert.
Die Heightmap der Landschaft wurde mit einem externen Programm (siehe zusätzliche Anmerkungen) erzeugt.

Verwendete Libraries

GLUT

- zur Erstellung des Fensters
- zur Ausgabe der Debuginfos am Screen
- zum Teil zur Abfrage der Eingabe
- Hinweis: es wurde die von einem anderen Teilnehmer der Übung modifizierte GLUT-Version eingesetzt, um das Mausrad benützen zu können

http://www.ultimategameprogramming.com/

- Laden von Bitmap- und TGA-Texturen
- Vector Klasse
- Matrix Klasse

http://www.gametutorials.com

- 3DS-Loader

http://www.dplate.de/

- XML Klasse

FMOD

- für Soundeffekte
- zum Abspielen der Hintergrundmusik

Zusätzliche Anmerkungen

Um den Sourcecode unter Visual Studio .NET 2003 erfolgreich kompilieren zu können, sind zusätzliche Inlcude-Files und Libraries von Visual C++ 6 notwendig. Sie befinden sich in den Verzeichnissen "\doc\vc6\include" sowie "\doc\vc6\lib" und sollten in das "include"- und "lib"-Verzeichnis von Visual C++ 7 kopiert werden.

Weiters sind auch noch die Includes und Libs für GLUT und FMOD notwendig, welche in den Verzeichnissen „\doc\glut“ bzw. „\doc\fmod“ zu finden sind.

Das mehrfach erwähnte externe Programm, mit dessen wir die Heightmap, die Lightmap sowie die Texturen der Landschaft erzeugt haben, nennt sich Scapemaker und wurde von uns mitentwickelt. Es ist unter http://www.scapemaker.de.vu downloadbar und kann zur Erzeugung von Landschaften für Ack-Ack sehr gut verwendet werden. Scapemaker verwendet allerdings DirectX 9 als Plattform. Sourcecode ist im Installationspackage zwar nicht vorhanden, kann aber gerne angefordert werden.

Soundtrack

Wie bereits erwähnt, dürfen wir uns als eines der (höchstwahrscheinlich) wenigen Teams der Übung rühmen, einen eigens für das Spiel erstellen Soundtrack anbieten zu können. Möglich gemacht wurde das durch zwei Mitbewohner im Studentenheim. Durch deren ausgezeichnete Arbeit gewinnt das Spiel an Atmosphäre und damit auch Spielspass. Noch einmal möchten wir uns an dieser Stelle herzlich für ihren Beitrag bedanken. Hier ein paar Infos zu den beiden:

Project Soundmühle

Homepage: http://www.soundmue.org
e-Mail: webmaster@soundmue.org