MIRAGE
mixed reality: cave.

Impressum

artecLab paper 4

micado
Bericht des studentischen Berichts micado zu Mixed Reality Caves

Laboratory for Art, Work and Technology
Universität Bremen
Enrique-Schmidt-Straße 7 (SFG)
D-28359 Bremen
www.arteclab.uni-bremen.de

Redaktion: Bernd Robben

ISSN 1860-9953
Copyright © artecLab-paper, Bremen
Satz und Herstellung im Eigenverlag

MiCaDo Projektbericht

micado@informatik.uni-bremen.de

15. Dezember 2004

Vorwort

Mira – MiCaDo!

Oder auf bremisch übersetzt:
Kiek an: Mixed Reality Cave – Pack an!

Projekt heißt wörtlich „das nach vorn Geworfene“. Wenn man ein Projekt macht, geht es darum, eine Sache voran zu treiben, etwas und damit sich selbst zu entwerfen. Eine aufregende Sache, das war von Anfang an zu spüren! Schon in der ersten Sitzung wurde der Name des Projekts mit dem Suffix „Do“ vervollständigt. „Tu was!“ hieß die Devise. Projiziert war ein Mixed Reality Cave – eine Höhle der Grenzerfahrung: ein unverschämt hochgestecktes Ziel und gleichzeitig bloß eine vage Idee! Nach sechs Wochen Arbeit war daraus der erste Prototyp geworden: Der technische Rahmen war relativ eng und präzise definiert, der Inhalt dagegen völlig offen.

Daraus ergaben sich zwei Konsequenzen: Erstens die Projektmitglieder brachten sich gegenseitig spezifisches Fachwissen bei, und zweitens planten sie eine Exkursion, um einen eigenen Inhalt zu finden. Die schwierige Phase des Projekts hatte begonnen. Weil alle engagiert bei der Sache waren, erhitzten sich die Gemüter. Zwei Visionen kristallisierten sich heraus: *Balance* im Cave und *Spiel mit Emotionen* im Cave. Und dann kam es, wie es kommen musste: Die Visionäre bekamen Streit, und sie zerstritten sich gründlich. MiCaDo hieß plötzlich nicht mehr: „Tu was – wir bauen das ultimative System der Grenzerfahrung!“, sondern „Wehe jemand berührt eine falsche Stelle – unser Projekt ist ein Ort der Selbsterfahrung!“.

So geht es zu in einem guten selbstorganisierten Projekt. Es muss reifen, es kann nicht alles glatt gehen. Und es hätte hier auch schief gehen können. Das ist das Risiko, das ein *Projekt* immer eingeht.

Aber MiCaDo ist nicht schief gegangen. Ganz im Gegenteil, die Gruppe hat sich hervorragend entwickelt. An die Stelle der Visionen trat ein machbares Ziel: ein *virtueller Dschungel*. Arbeitsgruppen bildeten sich. Nach einem Jahr wurde das Zwischenergebnis präsentiert. Im internen Konkurrenzkampf mit MiCarpet und MiCasa konnte MiCaDo sich sehen lassen. Der studentische Streik im Wintersemester brachte neue Erfahrungen. Plötzlich war schon das Ende sichtbar. Das Ergebnis von vier Semestern Arbeit musste im Fachbereich öffentlich vorgestellt werden. Jetzt hieß es wieder: „Tu was!“

Alle sind dabei geblieben, niemand ist abgesprungen. Alle haben – unterschiedlich nach Interessen, Engagement und Fähigkeiten – zum Gelingen beigetragen. Alle waren – nach anfänglicher Skepsis – Feuer und Flamme, das Projekt auch außerhalb des Elfenbeinturms der Universität zu zeigen. Alle haben auch nach dem offiziellen Ende des Projektes viele Stunden Arbeit auf sich genommen, um die Cave-Installation der Bremer Öffentlichkeit im Lichthaus am Pier 2 zu präsentieren und darüber hinaus ein anspruchvolles Begleitprogramm und ein Symposium in Zusammenarbeit mit dem Kino 46 zu organisieren.

MiCaDo war ein vorbildliches Projekt. Möge es viele Nachahmer finden, die dann wieder alles anders machen!

Bernd Robben
Bremen, 2004

Inhaltsverzeichnis

I Einführung 1

1 Mixed Reality Cave 3

1.1 Was ist ein Cave . . . 3

1.2 Was ist Mixed Reality . . . 4

2 Zielsetzung des Projekts MiCaDo 7

II Unser System 9

3 Die Madness Engine 11

3.1 Das BSP-Format . . . 12

3.2 Das md3-Format . . . 13

3.3 Von den Daten zum Bild . . . 14

3.4 Netzwerkarchitektur . . . 15

4 Cave-Konstruktion **17**
4.1 Der Prototyp . . . 17
4.2 Anforderungen an das finale Cave . . . 18
4.3 Das finale Cave . . . 19
4.4 Cave-Testaufbauten . . . 23

5 Sensorik **27**
5.1 Positionsbestimmung per Ultraschall . . . 28
5.2 Sensorgräben . . . 28
5.2.1 Entfernungsmessung . . . 29
5.2.2 Messstationen entlang der Strecke . . . 29
5.2.3 Reed-Kontakte . . . 29
5.3 Schnittstellen zum Computer . . . 30
5.3.1 Conrad C-Control . . . 31
5.3.2 Der IO-Warrior – die digitale Lösung . . . 32
5.3.3 Der IO-Warrior im Cave . . . 33

6 Interaktionen mit den Kreaturen **39**
6.1 Grundsätzlicher Interaktionsablauf . . . 39
6.2 Realisierung der Interaktion . . . 40

7 Pathfinding **45**
7.1 Struktur eines Wegfindealgorithmus . . . 46
7.2 Möglichkeiten der Navigationsdatenerzeugung . . . 47
7.2.1 Rasterung . . . 47
7.2.2 Wegpunktesystem . . . 48
7.2.3 Navigationsgitter . . . 49

7.3 Routenberechnung mit dem `pathtool` . . . 50
7.3.1 Erstellung des Wegpunktgraphen . . . 51
7.3.2 Eigenschaften der Wegpunkte . . . 54
7.3.3 Die resultierende Pfaddatei . . . 56
7.3.4 Ein Beispielszenario . . . 57

8 Kreatur-KI **61**
8.1 Motivationslevel und Events . . . 62
8.2 Aktionen . . . 64
8.3 Kreaturspezifische Konfigurationsdatei . . . 66
8.4 Kreaturbewegung . . . 68
8.4.1 Anderen Kreaturen ausweichen . . . 68
8.4.2 Zum Wegpunktesystem zurückfinden . . . 74

9 Sound **77**
9.1 Unser Ausgangspunkt . . . 78
9.2 Spatial Hearing: Die Soundmodelle . . . 79
9.3 Das Interface zur Kreatur-KI: Die Soundmanager . . . 84
9.4 Sounddesign . . . 85

10 Modellierung **87**
10.1 Levelmodellierung . . . 87
10.1.1 Arbeitsablauf . . . 88
10.1.2 Probleme beim Modellieren des Dschungel-Levels . . . 93
10.2 Entwicklung eines Charakters . . . 94
10.2.1 Modellierung . . . 94
10.2.2 Animation . . . 96

10.2.3 Texturierung . . . 97
10.2.4 Export . . . 98

11 Konfigurations- und Netzwerktools **107**
11.1 Die Weltkonfiguration . . . 107
11.2 Das `remad`-Fernsteuerungsprogramm . . . 109
11.3 Netzwerk Receiver-Thread . . . 110

III Projektverlauf **113**

12 Historischer Abriss über den Verlauf des Projekts **115**

13 Projektmanagement **125**

14 Sponsoring **127**

15 Der Projekttag **131**
15.1 Vorbereitung . . . 132
15.2 Ablauf . . . 132

16 mira im Kino 46 **135**
16.1 Planung . . . 136
16.2 Ablauf im Lichthaus . . . 137

17 Nachwort **141**

18 Projektteilnehmer **143**

Anhang **145**

Anhang Sponsoring 147

Anhang Modellierung 149

Quake III konforme *animation.cfg* 149

MiCaDo-Erweiterung der *animation.cfg* 150

Anhang Pathfinding 152

Translation File *demo.trans* 152

Anhang Konfigurations- und Netzwerktools 154

Beispiel-Weltkonfiguration *dschungel.world* 154

Umgebungsvariablen für die Madness Engine 158

Abbildungsverzeichnis **161**

Literaturverzeichnis **164**

Glossar **167**

— TEIL I —

Einführung

— KAPITEL 1 —

Mixed Reality Cave

Das Informatik-Projekt MiCa, Mixed Reality Cave, das unter der Leitung von Prof. Dr.-Ing. Friedrich-Wilhelm Bruns zum Wintersemester 2002 entstand, wurde wegen der vielen interessierten Studenten in drei kleine autonome Projekte unterteilt. Es entstanden die Projektgruppen MiCasa, MiCarpet und MiCaDo mit jeweils eigenen Projektleitern und Inhalten. Unser Teilprojekt, von dem wir in diesem Text berichten möchten, hat sich den Namen „MiCaDo“ gegeben, wobei „Do“ für unser Vorhaben und Schaffen steht. Es bestand aus siebzehn Studenten und unserem Betreuer, dem wissenschaftlichen Mitarbeiter des Institut artec Bernd Robben. Die Aufgabe unseres viersemestrigen Projektes war es, einen Cave zu bauen und diesen mit einer Anwendung zu versehen. Dazu gehörte die Entwicklung der Software zur Rechnersteuerung, die Erstellung einer dreidimensionalen virtuellen Welt, die Entwicklung eines oder mehrerer Eingabegeräte zur Interaktion und die dazugehörigen Schnittstellen.

1.1 Was ist ein Cave

Cave bedeutet *Computer Animated Virtual Environment* und ist eine Konstruktion, die es ermöglicht, den Benutzer in eine virtuelle, dreidimensionale Illusionswelt zu versetzen. Durch eine beispielsweise aus vier oder mehr Leinwänden bestehende Konstruktion erhält man einen begehbaren Raum, der den Benutzer komplett umgibt und in die Welt integriert. Die grafischen Daten, die auf die

Leinwände projiziert werden, können von dem Benutzer interaktiv kontrolliert und beeinflusst werden. Der Benutzer befindet sich dadurch nicht mehr in der reinen Beobachterrolle. Er erhält den Eindruck, sich in einer virtuellen Welt zu bewegen. Eine häufige Anwendung in einem Cave ist die Simulation, wie zum Beispiel eine Flugsimulation oder Motorradsimulation, wobei der Benutzer in einem Cave ein größeres Blickfeld als bei gewöhnlichen Simulationen bekommt und so seine gesamte Umgebung wahrnehmen kann. In unserem Projekt entschieden wir uns für eine Floßsimulation und erbauten einen sechsseitigen Cave.

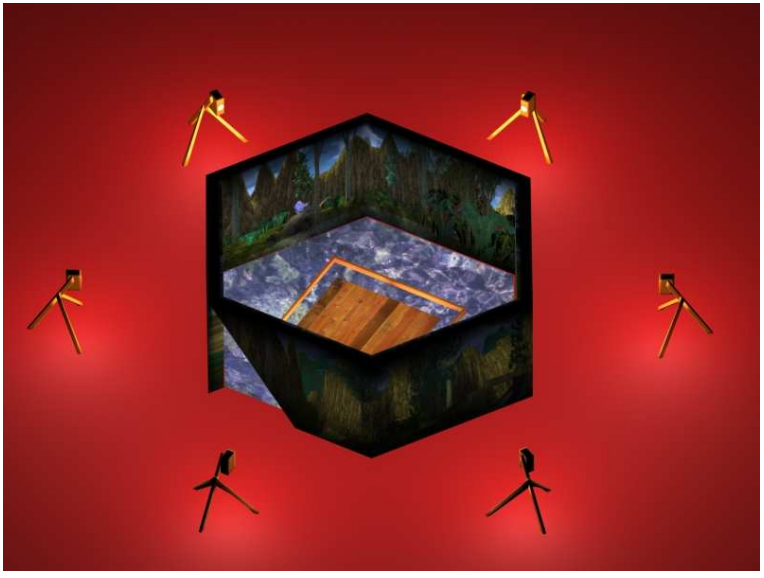

Abbildung 1.1: Unser sechsseitiger Cave

1.2 Was ist Mixed Reality

Der Begriff Mixed Reality wurde von Milgram und Kishino 1994 eingeführt und stellt die Verbindung und Vermischung von virtueller Realität und Realität, also der externen Welt, dar [MC99]. Als

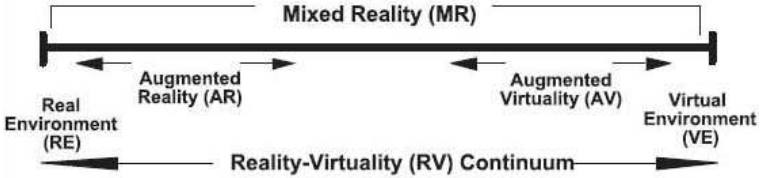

Abbildung 1.2: Definition von Mixed Reality bezogen auf das Reality-Virtuality Continuum von Milgram

virtuelle Realität bezeichnet man die Darstellung und gleichzeitige Wahrnehmung der möglichen Wirklichkeit und ihre physikalischen Eigenschaften in einer in Echtzeit computergenerierten virtuellen Umgebung. Virtuelle Realitäten können sowohl reale und natürliche Umgebungen imitieren, beispielsweise ein Museum oder das Innenleben eines Menschen, als auch unrealistische und abstrakte Umgebungen wie z.B. eine Märchenwelt sein. Wenn man nun die virtuellen Realitäten um reale Umgebungen erweitert, z.B. durch Videoaufnahmen oder Fotografien, so entsteht Augmented Virtuality.

Erweitert man jedoch die realen Seiten einer Umgebung um virtuelle, entsteht Augmented Reality. Mixed Reality nennt man den ganzen Bereich zwischen virtueller Realität und Realität, wobei man die reine Virtualität und die reine Realität ausklammert. Um diese Begriffe zu unterscheiden, hat Milgram das *Reality-Virtuality Continuum* definiert (siehe Abbildung 1.2).

In unserem Cave erschaffen wir Mixed Reality, genau genommen sogar Augmented Virtuality, da der Benutzer sich auf die virtuelle Welt konzentriert, aber erst mit Hilfe von realen Geräten durch diese virtuelle Welt navigiert. Das bedeutet, dass die virtuelle Umgebung durch die reale Umgebung erweitert wird und der Benutzer auf natürliche Weise mit der virtuellen Umgebung interagiert. Es kommt also zu einer „vom Computer generierten Illusion von Wirklichkeit, in der Weise, dass die Nutzerin den Eindruck bekommt, sie agiere in und mit der realen Welt statt mit einem Computersimulations-Modell“ [Bru03, S. 5]. Da der Benutzer glaubt, sich in einer virtuellen Welt zu bewegen und sich bei vielen Konstruktionen in der realen Welt aber nicht bewegt, kann es zu *Motion Sickness* kommen. Motion Sickness bezeichnet ein Übelkeitsgefühl, das durch eine Täuschung bzw. Irritation der Sinnesorgane zu Stande kommt. Ein guter Vergleich ist Seekrankheit, bei welcher der Gleichgewichtssinn im Ohr bei leichtem Seegang eine Bewegung spürt, das Auge jedoch keine sieht.

Bei Mixed Reality geschieht der gleiche Konflikt zwischen Auge und Ohr oft nur umgekehrt. Der Benutzer steht fest in seiner Umgebung und die virtuelle Welt bewegt sich.

— KAPITEL 2 —

Zielsetzung des Projekts MiCaDo

Die Zielvorgabe, die Entwicklung eines Mixed Reality Caves, bietet ein breites Spektrum möglicher Konstruktionsszenarien. Da aber die Absteckung der genauen Zielsetzung und einer präzisen Definition Teil der zu leistenden Aufgabe war, konnte sich unsere Vision vom zu bauenden Cave erst nach einem Dreivierteljahr generisch herauskristallisieren.

Die Implementierung einer Dschungelwelt mit eigenständigen, kommunikationsfähigen Geschöpfen sowie einer geeigneten Navigations- und Interaktionsmöglichkeit für den Benutzer stellten wir uns als Aufgabe. Der eigentliche Cave sollte die Erkunder der Dschungelwelt mit sechs Leinwänden umschließen. Nachdem dieses grobe Szenario entworfen war, hatten die einzelnen Untergruppen die verschiedenen Aspekte zu konkretisieren, wie die Gestaltung der Navigation, der künstlichen Intelligenz und die Erschaffung der Kreaturen sowie das Sound- und Leveldesign. Jede dieser Gruppen konnte dabei relativ autark arbeiten und ihre eigenen Wünsche umsetzen. Somit war das Ziel unseres Projektes mehr ein im Machen Erreichtes als ein im Vorhinein Gestecktes. Unsere Dschungelwelt beherbergt Schnecken, Schildkröten, Spinnen, Säbelzahntiger und eine Eigenkreation namens Florifant. Diese Kreaturen bewegen sich völlig eigenständig durch die erschaffene Welt. Sie verfolgen dabei abhängig von ihrer Gemütslage verschiedene Absichten wie Essen, Trinken, Schlafen oder Interesse am Cave-Besucher. Dieser steht auf einem rechteckigen, realen Floß, neben dem sich an jeder Seite ein Sensorgraben befindet. Mit dem Stab, den er in der Hand hält, kann er durch Abstoß- und Ruderbewegungen innerhalb der Gräben durch die Welt navigieren. Die Fahrt auf Land ist ihm genauso gestattet wie auf dem Wasser. In der anderen Hand hält der Besucher eine Fernbedienung.

Mit ihr kann er mit den Kreaturen des Dschungels interagieren. Umherlaufende Tiere können angelockt oder weggescheucht werden. Schlafende Tiere lassen sich mit ihr wecken. Hat der Besucher das Gefühl, die Gewalt über das Floß zu verlieren, befindet sich auf der Fernsteuerung noch eine Bremse. Um das Raumgefühl des Caves zu erhöhen, gibt es einen dreidimensionalen Sound, so dass der Benutzer hört, wenn sich eine Kreatur hinterrücks nähert.
Im weiteren Verlauf dieses Berichts folgt nun eine detaillierte Beschreibung unseres Systems, die die Entscheidungsprozesse innerhalb der Untergruppen ebenso sichtbar machen wird, wie die Ideen und Konzepte, die wir entwickelt und abgelehnt haben (siehe Kapitel 12).

— TEIL II —

Unser System

— KAPITEL 3 —

Die Madness Engine

Als Grundlage für unser gesamtes System benötigten wir zunächst eine *3D-Engine*[1], mit der wir das Bild unserer virtuellen Welt auf die Außenseiten des Caves projizieren können. Entgegen anderen Möglichkeiten entschieden wir uns für die im artec entwickelte Madness Engine. Die Madness Engine ist eine 3D-Engine, die von Martin Faust[2] mit dem Anspruch geschrieben wurde, mit allen für Quake III[3] vorhandenen Modellen kompatibel zu sein. Sie lädt Weltmodelle, die wie bei Quake III im *BSP-Format* gespeichert werden. Um animierte Modelle von Lebewesen oder ähnlichen bewegten Kreaturen in die Welt einzubinden, besitzt die Engine die Möglichkeit, im *md3-Format* abgespeicherte Modelle zu laden. Implementiert ist die Engine in der objektorientierten Programmiersprache C++ [Str00] und unter Zuhilfenahme der Open-Source-Multimediabibliothek SDL. Die Madness Engine bietet zusätzlich zu den Standardfunktionen einen für uns besonders interessanten Modus, mit dessen Hilfe verteiltes Rendern für verschiedene Sichtwinkel auf die Welt möglich ist. Dies bedeutet, dass ein Steuerrechner berechnet, wie die 3D-Welt in der Situation eines bestimmten Frames aussieht und diese Information dann an Hilfsrechner im Netz verteilt, die dann jeweils ein eigenes Bild der Welt berechnen. Der Steuer- oder Hauptrechner kann die Bilder für alle Seiten des Caves nicht allein berechnen.

[1] Eine 3D-Engine berechnet aus einer dreidimensionalen Szene ein zweidimensionales Bild.

[2] Kontakt: faust@artec.uni-bremen.de

[3] Quake III ist ein Computerspiel von IdSoftware, in dem der Spieler sich durch virtuelle Welten bewegen kann (siehe auch http://www.idsoftware.com/games/quake/quake3-arena/).

Die Implementierung erfolgte in Programmmodulen, die z.B. die Funktionalität für das Einlesen des BSP-Baums, das Einlesen der 3D-Modelle und die Szenenrepräsentation zur Verfügung stellen. Das letztgenannte Modul ist für die Darstellung der Welt aus der Sicht des Sichtpunkts (*Viewpoint*) zuständig. Es stellt Informationen bereit, die z.B. die aktuelle Position des Punktes, von dem aus die Welt betrachtet wird, betreffen. Weitere Informationen zu wichtigen Aspekten der Madness Engine befinden sich in den folgenden Abschnitten.

3.1 Das BSP-Format

Das BSP-Format ist ein Dateiformat, um eine zuvor modellierte, statische 3D-Welt in ein Programm zu laden und dann anhand eines bestimmten Sichtpunkts Sichtbarkeitsberechnungen durchzuführen. Es handelt sich dabei um eine Baumstruktur, die nach dem so genannten *Binary Space Partitioning*-Verfahren [Wat02, S. 227–230] erstellt wurde. Die Modellierung erfolgt mit Hilfe eines Level-Editors[4], der nach Abschluß dieser Arbeit aus den Polygonen dieser Szene einen Datenbaum erzeugt und dabei beispielsweise auch Lichtberechnungen vornimmt. Für die Lichtberechnung können dabei verschiedene Algorithmen benutzt werden, die schließlich *Light-Maps* erzeugen. Bei den Light-Maps handelt es sich um Texturen, die die Helligkeit der Polygone angeben, da es keine weitere Beleuchtungsberechnung zur Laufzeit in der Engine gibt.

Zum Erzeugen des Baums wird nun folgendermaßen vorgegangen: Zunächst wird die Szene anhand bestimmter Teilungsebenen zerlegt, der jeweils entstandene linke und rechte Szenenteil weiter geteilt und in den jeweiligen Teilbaum der BSP-Baum-Struktur eingehängt. Am Schluß enthält die so entstande Struktur in den Knoten die Teilungsebenen[5] und in den Blättern die Polygone der Szene. Zum Zeichnen der Szene kann nun relativ einfach die Sichtbarkeit der Polygone und deren Zeichnungsreihenfolge bestimmt werden. Dazu muss jeweils mit Hilfe der Position des Sichtpunkts in Bezug auf die Teilungsebenen der BSP-Baum durchlaufen werden und die Polygone entsprechend ihrer Sichtbarkeit und Entfernung zum Sichtpunkt gezeichnet werden.

Der große Vorteil dieses Verfahrens liegt darin, dass der aufwendigere Teil der Berechnung, nämlich die Berechnung des Baums, nur einmal erfolgen muss und das Durchlaufen des Baums zur Laufzeit relativ ressourcenschonend ist.

[4] Ein Level-Editor ist ein Programm, mit dem eine virtuelle Welt erstellt werden kann.
[5] Im obersten Knoten die erste und von dort ausgehend die jeweiligen Teilungsebenen der entstandenen Seiten.

Um den Aufbau eines BSP-Baums zu verdeutlichen, ist in Abbildung 3.1 eine einfache Szene mit verschiedenen Objekten und der zugehörige BSP-Baum dargestellt. Die nummerierten Objekte aus dem linken Teil der Abbildung sind in dem angegebenen BSP-Baum entsprechend der Normalen[6] der Teilungsebenen A–C eingetragen.

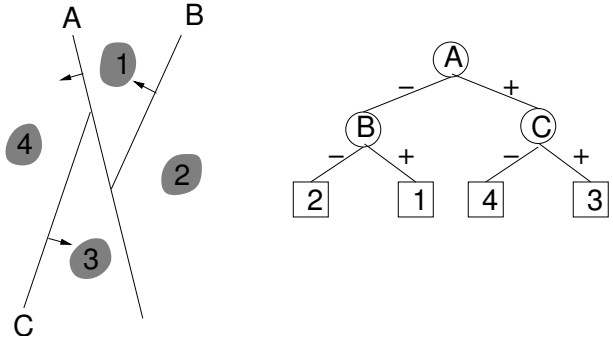

Abbildung 3.1: Eine Beispielszene und der zugehörige BSP-Baum

Zu beachten ist, dass die Teilung nicht wirklich an den Objektgrenzen halt macht, sondern natürlich auch die Tatsache berücksichtigt wird, dass die Objekte aus mehreren Polygonen bestehen. Weitere Informationen zur Modellierung der Welt befinden sich im Kapitel 10.1.

3.2 Das md3-Format

Beim md3-Format handelt es sich um ein Dateiformat, mit dem animierte Modelle für eine 3D-Welt abgespeichert werden können. Dieses Format beschreibt im einzelnen, wo sich ein bestimmtes Vertex[7] eines Modells zu jedem Zeitpunkt der Animation befindet [md304]. Eine Animation besteht immer aus dem oberen und unterem Teil des Modells, dem Kopf sowie einer Konfigurationsdatei. Durch die Teilung in einen oberen, unteren und einen Kopfteil sind Kombinationen von Animationen möglich, ohne diese im Voraus zu berechnen. In der Konfigurationsdatei wird dann festgelegt,

[6] Eine Normale ist ein Vektor $\vec{n}$, der senkrecht auf einer durch zwei Vektoren $\vec{a}$ und $\vec{b}$ aufgespannten Ebene steht.
[7] Bei einem Vertex handelt es sich grundsätzlich um einen Eckpunkt eines Polygons.

welche Frames[8] aus den md3-Dateien zu welchem Typ von Animation gehören, da in einer md3-Datei alle Animationen hintereinander gespeichert sind. Beispielsweise können die Frames 30–65 die Beinbewegung des Modells für die Animation „Laufen“ enthalten. Erstellt werden die Animationen zunächst mit Hilfe eines Modellierungswerzeuges, wie z. B. GMax[9], aus welchem sie dann mit einem Exporter ins md3-Format überführt werden. Die weiteren Aspekte der Modellierung von Animationen werden ebenfalls im Kapitel 10.1 erläutert.

3.3 Von den Daten zum Bild

Die Aufgabe einer 3D-Engine – also in unserem Fall der Madness Engine – ist es, die Daten aus den md3- und den BSP-Dateien zu interpretieren und anhand eines gegebenen Sichtpunktes und -winkels in Echtzeit eine Szene zu rendern. Dazu wird zunächst beim Start des Programms der angegebene BSP-Baum in den Speicher geladen, um darauf Operationen ausführen zu können. Hierfür sind in der Implementierung der Engine geeignete Funktionen vorhanden. Soll nun ein konkretes Bild erzeugt werden, so wird der BSP-Baum anhand der Koordinaten des Sichtpunktes durchlaufen, wobei an jedem Knoten des Baumes bestimmt wird, auf welcher Seite der Knotenebene sich der Sichtpunkt befindet. Dann wird der Unterbaum der hinteren Seite ausgegeben und danach der Unterbaum der vorderen Seite. Hierzu werden die Unterbäume jeweils absteigend durchlaufen. Abschließend wird noch das so genannte *Frustum Culling* [Fru04] durchgeführt. Bei diesem Verfahren wird ein Sichtbarkeitsbereich mit Hilfe von 6 Ebenen bestimmt. Diese Ebenen begrenzen den Raum nach vorn, nach hinten, seitlich sowie nach oben und nach unten. Es wird nun jeweils bestimmt, ob sich ein Polygon innerhalb dieses Bereiches befindet, und nur dann werden seine Daten im letzten Schritt an den Grafikkartenspeicher weitergegeben. Dieses Verfahren wird aus Gründen der Performanz angewendet, da die Grafikkarte dann nur noch die Polygone bearbeiten muss, die auch wirklich sichtbar sind.

Für die md3-Modelle gilt das Gleiche. Sie werden durch geeignete md3-Import-Funktionen nacheinander eingelesen und die verschiedenen vorhandenen Animationen unter ihren Kennnamen zur Verfügung gestellt. Während des Programmablaufs können die Animationen dann mit einer Methode `setAnimation()` abgespielt werden.

[8] Ein Frame ist ein einzelnes Bild aus einer Folge von mehreren Bildern, die fortlaufend gerendert werden.
[9] siehe http://www4.discreet.com/gmax/

Wichtig für den Ablauf des Berechungsprozesses sind nun noch Funktionen, mit denen z.B. die Höhe – also im Fall der Madness Engine die y-Koordinate – eines bestimmten Punktes in der Szene berechnet werden kann. Dies übernimmt in der Madness die Methode `getHeightOfTerrain()`, die unter Zuhilfenahme des BSP-Baums ausgehend von einer gegebenen Position nach unten überprüft, an welchem Punkt eine Kollision mit der Landschaft bzw. dem Boden stattfindet.

Die konkrete Umsetzung der Daten in Bilder erfolgt, wie schon oben erwähnt, mit der Open-Source-Multimediabibliothek SDL (*Simple DirectMedia Layer*) [SDL04]. Diese Funktionsbibliothek stellt Funktionen zur Verfügung, die beispielsweise OpenGL-Aufrufe[10] kapseln oder das Laden und Abspielen von Sounddateien unterstützen. SDL gewährleistet außerdem eine gewisse Portabilität der 3D-Engine, da es außer für die Linux-Plattform beispielsweise auch für Windows verfügbar ist.

3.4 Netzwerkarchitektur

Die ursprüngliche Version der Madness Engine enthielt schon die Möglichtkeit, das Rendern der Welt auf mehrere Rechner zu verteilen. Diesem verteilten Rendern liegt das Prinzip zu Grunde, dass jeder Rechner aufgrund der ihm übermittelten Informationen über die aktuelle Position des Sichtpunkts und den aktuellen Sichtwinkel sein eigenes Bild errechnet. Hierdurch ist es möglich durch eine beim Start festgelegte, feste Verschiebung des Sichtfelds mit Hilfe mehrere Rechner eine 360°-Ansicht zu rendern und diese mit mehreren Beamern auf eine bestimmte Anzahl von Leinwänden zu verteilen.

Grundsätzlich benötigt dazu jeder Rechner zunächst einmal die gleichen Daten, d. h. das gleiche Weltmodell, die animierten Modelle und auch alle benötigten Sounds. Diese Einrichtung muss vor dem Start der Engine erfolgen, da zur Lauftzeit nur noch Positionsangaben über das Netzwerk verschickt werden. Ein Rechner übernimmt nun die Aufgabe des Hauptrechners (*Master*) und übernimmmt die Steuerung der übrigen teilnehmenden Rechner. Genauer gesagt bedeutet dies, dass auf diesem Hauptrechner sämliche Interaktionen mit dem Benutzer, evtl. KI-Berechnungen und ähnliches abläuft und er dann nur die Position des Sichtpunkts und die Position der Modelle an seine Hilfsrechner (*Slaves*) übermittelt. Dies geschieht per UDP-Multicast verschickter Informationspakete, die von den Slaves empfangen werden. Diese berechnen dann jeder für sich aufbauend auf den Informationen, die sie erhalten haben, ihr eigenes Bild. Haben sie ein weiteres Bild gerendert, wird dem Master-Rechner ein Bestätigungspaket gesendet. Der grundsätzliche Aufbau ist in Abbildung 3.2 verdeutlicht.

[10] Diese 3D-Engine basiert auf der Grafikbibliothek OpenGL. [OGL04] [SWN99]

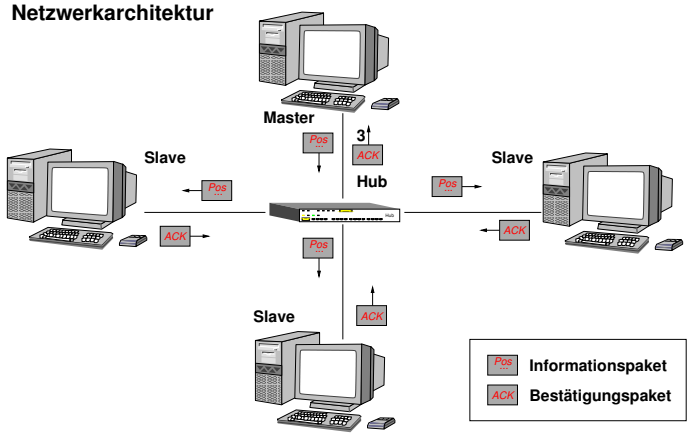

Abbildung 3.2: Die grundsätzliche Netzwerkarchitektur beim verteilten Rendern

— KAPITEL 4 —

Cave-Konstruktion

Allgemein besteht ein Cave aus einzelnen Projektionswänden, auf die jeweils ein Beamer die virtuelle Welt projiziert. Diese einzelnen Segmente werden zu einer Höhle zusammengestellt, so dass der Benutzer von ihnen eingeschlossen wird. Die optimale Cave-Form ist eine Kugel, da in ihr keine Ecken und Bildverzerrungen auftreten. Da eine runde Leinwand nur sehr schwer zu konstruieren ist, musste bei der Konstruktion ein Kompromiss gefunden werden.
Ein erster Prototyp sollte uns helfen eine Vorstellung von einem Cave zu bekommen und diesen Kompromiss zu finden.

4.1 Der Prototyp

Die Anforderung an unseren ersten Prototypen war eine kostengünstige Bauweise, welche schnell realisiert und aufgebaut werden kann. Als Baumaterialien wurden pro Leinwand zwei Holzlatten und ein weißes Bettlaken verwendet. Das Laken wurde an der schmaleren Seite mit Hilfe von Klammern an den Holzlatten befestigt, so dass eine Latte oben und eine andere unten war. Solch eine Leinwand wurde an der oberen Latte mit Paketband an der Decke befestigt. Durch das Gewicht der unteren Latte wurde die Leinwand gespannt. Somit war der Prototyp sehr modular gehalten.

Erste Experimente mit verschiedenen Winkeln zwischen je zwei Leinwänden brachten uns zu dem Schluss, dass wir ein sechsseitiges Cave bauen wollten, denn ein 60^{o}-Winkel vermittelt ein besseres

Raumgefühl als ein 90°-Winkel, wie er bei einem vierseitigen Cave verwendet wird. Bei letzterem nimmt der Betrachter auf ihn unnatürlich wirkende Verzerrungen in den Ecken des Caves wahr, die bei Verwendung eins stumpferen Winkels deutlich weniger auffallen. Das Bildformat wollten wir im Seitenverhältnis 4:3 lassen, wie es bei Bildschirmen und Videoprojektoren üblich ist, da wir so handelsübliche Beamer benutzen konnten, ohne sie vorher zu modifizieren. Auch ein großer Mensch sollte im Cave nicht über die Leinwand schauen können. Aus diesen Gründen kamen wir auf ein Leinwandformat von ca. drei Metern Höhe und zwei Metern Breite. Dies hatte zur Folge, dass wir das Bild um 90° drehen mussten, also mit einem Seitenverhältnis von 3:4 arbeiteten. Unter Berücksichtigung all dieser Bedingungen kam ein Cave-Durchmesser von sechs Metern zu Stande.

Um noch ein besseres Raumgefühl zu bewirken, hatte sich das Projekt überlegt, die Decke und den Boden des Caves zusätzlich zu bestrahlen, was aus Sicht der Madness Engine auch kein Problem dargestellt hätte.

4.2 Anforderungen an das finale Cave

Aus dem Prototyp und den daraus resultierenden Überlegungen im Projektplenum entstanden folgende Anforderungen für das finale Cave:

- Sechseckig
- Robust und modular
- In relativ kurzer Zeit auf- und abbaubar
- Ein Lautsprecherpaar pro Leinwand direkt am Cave befestigt
- Ein Beamer pro Leinwand direkt am Cave befestigt
- Benutzung von Spiegeln zwecks geringerem Platzbedarfs
- Kostengünstig

Die benötigten Projektoren sollten nach Möglichkeit direkt am Cave befestigt sein, damit die Benutzer oder Zuschauer diese nicht verrücken oder beschädigen können.

4.3 Das finale Cave

Die erste Idee der Cave-Bau-Gruppe war eine durchgehende Leinwand, die durch sechs einzelne Pfosten gehalten wird. Über ein Stahlseil sollte die Leinwand gespannt und in Form gehalten werden. Die Pfosten, an denen die Seile zu befestigen waren, sollten sich außerhalb des Caves befinden, so dass sie keine Schatten auf die Leinwand werfen. In der Mitte über dem Cave stützen sich die Pfosten gegenseitig, um der Spannung der Leinwand standzuhalten. Leider erwies sich diese Lösung als nicht realisierbar und zu kostenintensiv.

Das finale Gerüst

Im Plenum wurde nach einer billigeren Lösung gesucht. Es kam der Vorschlag, eine zeltähnliche Konstruktion zu nehmen, bei der die Leinwand an den Stangen befestigt wird. Laut eines Projektmitglieds waren sechseckige Gartenpartyzelte zu einem niedrigen Preis zu erwerben. Basierend auf dieser Aussage machten wir uns auf die Suche nach solch einem Partyzelt. In einem Baumarkt fanden wir ein Ausstellungsstück, welches einen Produktionsfehler in den Seitenwänden hatte, welche wir für unser Vorhaben aber nicht benötigten. Somit hatten wir die Basis für unseren Cave gefunden. Das Zelt wurde um sechs Zeltstangen erweitert, so dass wir eine Seitenlänge von zwei Metern Breite und ca. drei Metern Höhe erhielten. Die Stabilität des Caves wird durch das Zeltdach erhöht, welches wir allerdings nur in sehr hohen Räumen aufbauen können (siehe Abbildung 4.1).

Die Leinwände

Der richtige Leinwandstoff ist einer der wichtigsten Bestandteile eines Caves. Auf ihm muss das projizierte Bild gut darstellbar sein und es darf nicht die Maserung des Stoffes zu erkennen sein. Gleichzeitig muss die Helligkeit gleichmäßig über die komplette Fläche verteilt werden, so dass die Beamerlampe nicht durch den Leinwandstoff hindurch als heller Kreis zu erkennen ist oder gar den Besucher blendet. Um all dies zu erreichen und nur einmal einen Stoff kaufen zu müssen, machte sich eine Gruppe mit einem Beamer auf den Weg in verschiedene Stoffläden in Bremen und versuchte den besten Stoff zu finden.

Die Verarbeitung wurde mit einer handelsüblichen Nähmaschine durchgeführt. Damit die Leinwand glatt und straff blieben, wurden alle sechs Seiten miteinander verbunden. Oben bekam die Leinwand eine Lasche, so dass das Quergestänge des Cave-Gerüstes hindurch geschoben werden kann. Seit-

Abbildung 4.1: Cave-Gerüst mit Dach

lich ermöglichen vier mal zwei Stoffstücke das Festbinden an den vertikalen Stangen. Eine der Verbindungen zwischen zwei Seiten wurde nicht zusammengenäht, sondern lässt sich mittels kleiner Haken und Ösen schließen. Diese Seite wird als Zugang zum Cave genutzt.

Die Lautsprecherhalterungen

Unser Cave sollte auch akustisch zu einem Erlebnis werden. Deshalb hatten wir uns entschlossen, jede Leinwand mit einem Stereokanal auszustatten. Dieser wird jedoch nicht, wie allgemein üblich, für die Darstellung von Links und Rechts verwendet, sondern für Oben und Unten. Dabei wird jedes

Boxenpaar von jeweils einem Cave-Rechner (siehe Abschnitt 3.4) angesteuert. Um die oberen Boxen am Cave zu befestigen, mussten wir uns eine Lösung einfallen lassen, welche das Budget nicht zu sehr belastete. Hieraus entstanden unsere Boxenplattformen, welche aus jeweils einem Stück Holz in den Maßen 20×20×2 cm (B×T×H) bestehen. Jede Boxenplattform wird mit Hilfe einer Rohrschelle an der oberen Querstange des Cave-Gerüstes befestigt.

Die Beamerständer

Die Bauweise des Gerüsts und die fehlenden finanziellen Möglichkeiten zwangen uns dazu, die Beamer außerhalb des Caves zu positionieren und ohne Spiegel zu arbeiten. Da wir das normale Bildformat um 90° gedreht hatten, mussten wir auch die Projektoren um diesen Winkel drehen. Um dies zu ermöglichen bauten wir unsere eigenen Beamerständer. Diese sollten folgende Kriterien erfüllen:

- Günstig in der Anschaffung und umsetzbar durch uns
- Extrem sicherer Stand, so dass unter keinen Umständen ein Beamerständer zu Fall kommen kann
- Sichere Verankerung des Beamers auf dem Ständer
- Leichte vertikale Ausrichtbarkeit eines Beamers
- Nahezu jedes Beamermodell verwendbar (bzgl. Umfang, Gewicht)

Zur Fertigung der Beamerständer benutzten wir Holzbalken, wobei zwei gleich lange Balken ein umgedrehtes V bildeten auf dessen Spitze ein sehr langer Balken lag. An diesem langen Balken wurde ein rechteckiges Brett befestigt, gegen welches der Beamer gespannt wird. Als Auflagefläche für die Seite des Beamers wurde ein schmales rechteckiges Brett im rechten Winkel gegen das größere gesetzt. Dieses schmale Brett wurde nun mit mehreren Löchern versehen, um einer Überhitzung der Projektoren durch mangelnde Luftzufuhr vorzubeugen. Um ein seitliches Wegfallen des Beamers zu verhindern, wird eine lange Schnur mehrfach um den Beamer und durch zwei zusätzlich angebrachte Ösen geführt. Durch all diese Maßnahmen sind die Beamerständer standfest und die Projektoren darauf gut verankert (siehe Abbildung 4.2).

Abbildung 4.2: Beamerständer wird von einem Projektmitglied justiert

Das Floß

Aus einem ausgedienten Bett und dessen Lattenrost konstruierten wir ein Floß. Dabei sind die Abmessungen des Floßes auf die Sensorgräben (siehe Abschnitt 5.2) abgestimmt. Zuerst sollte das Floß eine Seitenlänge von 150 cm haben, aber mit dem vorhandenen Holz war nur eine Seitenlänge von 120 cm möglich. Die Sensorgräben wurden mit Holz verkleidet, welches die Elektronik versteckt und schützt. Sowohl die Gräben wie auch das Floß selbst sind mit Beize und Klarlack gestrichen und verschönern das Ambiente im Cave.
Die Sensorgräben werden über zwei Kabel mit dem IO-Warrior (siehe Abschnitt 5.3.2) verbunden,

der sich außerhalb des Caves am *Master*-Rechner befindet (siehe Abschnitt 3.4). Diese Kabel werden unter der sich im Cave befindenden Bodenfolie versteckt und unter dem Floß mit den Anschlüssen der Gräben verbunden. Dadurch wird ein optimaler Schutz für die Elektronik gewährleistet.

Abbildung 4.3: Floß

4.4 Cave-Testaufbauten

Innerhalb der ersten zwei Monate des Projekts wurde ein erster Cave-Prototyp im artec-Labor 2 fertiggestellt. Um Möglichkeiten für die Navigation zu testen, wurde ein Eingabegerät namens *MadBird*

gebaut. In diesem Aufbau wird jeweils ein Faden an ein Handgelenk einer Testperson geknotet und die Auf- und Abschwingbewegung der Arme über ein Potentiometer gemessen. Die Messergebnisse werden dann in der virtuellen Welt in Bewegungen umgesetzt. Zusätzlich wurde die Navigation per Tanzmatte getestet. Eine Tanzmatte bietet vier Eingabefelder, die jeweils durch Betreten aktiviert werden können. Die Soundausgabe beschränkte sich auf die bereits in der Madness Engine integrierte rudimentäre Soundberechnung. Als virtuelle Welt diente eine normale Quake III-Map.

In einem Vortrag über die verschiedenen Möglichkeiten einer Cave-Konstruktion wurden noch einmal verschiedene Varianten eines Caves vorgestellt. Hier fiel dann endgültig die Entscheidung, einen sechsseitigen Cave zu konstruieren. Zunächst war geplant, das Gerüst des Caves selbst aus Holz zu bauen. Es wurde daran getüftelt, wie man eine Einstiegsmöglichkeit in den Cave integrieren könnte. Diese Variante des Caves kam jedoch nie über das Planungsstadium hinaus. Der nächste Ansatz war bereits deutlich professioneller und basiert auf einer durchgängigen Leinwand, die durch Eckpfeiler aus Stahlseilen in Form gebracht werden. Allerdings ist dieser Ansatz deutlich zu kostspielig. Letztendlich wurde ein sechsseitiger Zeltpavillon günstig in einem Baumarkt gesichtet und gekauft. Somit hatten wir ein Gerüst für den Cave. Bezüglich der Eingabemöglichkeiten hatte sich eine Idee mit so genannten Sensorgräben durchgesetzt. Dazu paddelt der Benutzer mit einem Ruderstab, dessen Spitze er durch einen Sensorgraben führen kann, durch den virtuellen Dschungel. Die Bewegung der Ruderstabspitze innerhalb des Sensorgrabens wird dann als Bewegung in der virtuellen Welt umgesetzt.

In den Sommersemesterferien 2003 wurden in einer Projektwoche die Leinwände, Beamerständer, Sensorgräben und natürlich das Netzwerk im GW2 Raum B1400 aufgebaut. Dazu musste für die Leinwände ein geeigneter Stoff ausgesucht, gekauft, zugeschnitten, genäht und gereinigt, für die Beamerständer das Holz gekauft, zugeschnitten und verschraubt, für die Sensorgräben das Holz gesägt und verschraubt, und die elektrischen Kontakte gekauft und gelötet werden. Das meiste davon wurde in dieser Projektwoche gemacht, die Gräben wurden zum Teil schon vorher fertiggestellt. Nachdem die o.g. vorbereitenden Arbeiten abgeschlossen waren, konnten wir den Cave zum ersten Mal in einer projektinternen präsentationsähnlichen Form testen.

Bei einer ersten Präsentation im Oktober 2003 im SFG ging es in erster Linie darum, einen vorzeigbaren Cave möglichen Sponsoren zu zeigen. Gleichzeitig war dies die erste Möglichkeit für die Öffentlichkeit, sich einen Eindruck von unserem Projekt zu machen. Der Cave bestand damals aus den sechs Leinwänden, von denen nur drei funktionstüchtig waren (aus technischen und aus Platzgründen). Die Gräben waren im Cave vorhanden, jedoch noch nicht einsatzbereit. Ebenso fehlte der Sound und am Ende des Tages eine Zeltstange, welche jedoch nach einigen Tagen wieder auftauchte.

Bei der Generalprobe in den Wintersemesterferien 2004 musste alles funktionieren. Grafik-, Modellierungs-, Sound- und Sensorikgruppe mussten mit ihren Arbeiten fertig sein. Der Cave wurde erneut im GW2 Raum B1400 aufgebaut. Nachdem man sich geeinigt hatte, die Sensorgräben mit Reed-Modulen und einen Stab als Navigations- und Interaktionsmöglichkeit zu bauen, wurden nicht sechs sondern nur vier Gräben gebaut. Es sollte ein Floß gebaut werden, mit dem man sich durch den Dschungel bewegt. Die Sensorgräben bestanden aus Reedkontakten und einer SPS. Die Messdaten der Gräben wurden per IO-Warrior an den Master-Rechner weitergeleitet. Die Soundgruppe ordnete den Kreaturen Sounds zu, so dass das Laufen, Essen, und Schlafen der Tiere von den Besuchern zu hören war. Die Grafik- und Modellierungsgruppe musste danach nur noch Feinheiten klären, und der Projekttag konnte kommen. Durch diese iterative Annäherung an den endgültigen Cave, sowie die Flexibilität, noch späte Änderungen anzunehmen und umzusetzen, hatten wir es geschafft, einen Cave zu konstruieren, der sowohl auf reges Interesse stieß, als auch sich als ausfallsicher entpuppte.

— KAPITEL 5 —

Sensorik

Um sich in einer virtuellen, also computergenerierten Welt fortbewegen zu können, muss man sich dem Computer auf bestimmte Art und Weise mitteilen können. Dieses ist nicht mit der menschlich intuitiven Art vergleichbar.

Man benötigt spezielle Eingabegeräte, um sich dem Computer mitzuteilen. Diese Eingaben werden dann in ein computerlesbares Format umgewandelt. Dieses Format muss dabei eindeutig und klar strukturiert sein, um Fehler auszuschließen. Als Standardeingabegeräte für Computersysteme haben sich die Tastatur zur Texteingabe und die Maus als Zeigeinstrument bewährt. Man kann mit diesen Eingabesystemen fast alle gewünschten Handlungen in den Computer eingeben.

In der virtuellen Welt von MiCaDo wollten wir eine andere Art von Eingabemedium verwenden als Tastatur und Maus. Wir brauchen ein Eingabemedium, welches die Möglichkeit bietet, in einem virtuellen Raum zu navigieren, ohne den Weg über die bekannten Eingabegeräte zu nehmen. In den folgenden Kapiteln werden wir beschreiben, wie wir diese *Mensch-Maschine-Schnittstelle* realisiert haben.

Im Plenum wurden die verschiedensten Vorschläge für eine Schnittstelle gesammelt. Eine erste Implementierung einer Art Flugplattform (MadBird) war bereits begonnen worden, bevor die endgültige Entscheidung auf eine Dschungelwelt fiel, durch die man sich mit Hilfe eines Floßes fortbewegen sollte. Für ein Floß wird ein Ruderstab benötigt über den die Interaktion ausgeführt, und das Floß bewegt wird. Als Sensorikgruppe galt es also einen solchen Stab zu implementieren.

5.1 Positionsbestimmung per Ultraschall

Das erste Konzept, dem wir nachgegangen sind, war die Bestimmung der absoluten Lage des Ruderstabes im Raum. Mit diesen Positionsangaben hätte der Stab auch sehr leicht zu anderen Zwecken, wie z.B. zum Markieren von Tieren und Interagieren mit Kreaturen verwendet werden können. Für den ersten Prototypen verwendeten wir das Tracking-System IS600 der Firma Intersense , welches mit Hilfe von Ultraschall, 3D-Koordinaten der Sender liefert. Die Funktionsweise dieses Gerätes ist einfach zu beschreiben: Die Sendeeinheiten (in unserem Fall vier Stück) senden eindeutige Ultraschallkennungen an die Basiseinheit. Diese ermittelt über drei Mikrofone die absolute Lage eines jeden Senders im Raum. Diese Sender waren an einem Ende des Ruders befestigt. Das Tracking-System selbst war über dem Cave angebracht.
Schon bald traten verschiedene Probleme auf. Der Cave-Benutzer verdeckte beim Rudern oft die Sender, so dass kein Signal die Basiseinheit erreichen konnte. Dieses Problem ließ sich weder durch Experimente mit der Positionierung der Sender, noch durch eine Signalverstärkung durch zusätzliche IR-Scheinwerfer lösen. Zusätzlich machte uns die Erneuerung der Batterien der Sendeeinheiten Probleme, da sie nur durch den Hersteller gewechselt werden konnten. Um dieses Problem in Zukunft zu vermeiden, entschlossen wir uns, die internen Batterien durch externe Batteriepacks zu ersetzen. Diese vergrößerten die Sendeeinheiten, so dass unser Ruderstab unhandlich wurde.

Im Laufe unserer Experimente wurde die Schmelzsicherung im Tracking-System immer häufiger ausgelöst. Die Tests wurden durch den Sicherungswechsel umständlicher und wir befürchteten, dass dies die ersten Symptome eines bevorstehenden Totalausfalls sein könnten. Da ein Ersatzgerät unser Budget gesprengt hätte, entschlossen wir uns, unsere Floßsensorik ohne teure Spezialhardware zu realisieren.

5.2 Sensorgräben

Eine Lösung ohne Tracking System erfordert Kompromisse in der Bewegungsfreiheit des Ruderstabes. Somit mussten wir den Benutzer zwingen, das Ruder in einer festen Position zu halten. Die Navigation sollte jedoch weiterhin möglichst intuitiv sein. Als Lösung entwickelten wir die von uns so getauften Sensorgräben.
Unser Cave wird mit einer variablen Anzahl von Sensorgräben ausgestattet (in unserer Realisierung haben wir vier Gräben). Wird der Stab durch einen Graben gezogen, so misst der Computer Rich-

tung und Geschwindigkeit.
Zur Messung der Stabbewegungen testeten wir verschiedenartige Sensoren die im folgenden beschrieben werden.

5.2.1 Entfernungsmessung

Jeder Sensorgraben wird mit einem Infrarot-Entfernungsmesser ausgestattet. Das Ruder besitzt in dieser Lösung keine aktiven Sensoren, es ist also leicht, handlich und in beide Richtungen (mit beiden Enden) benutzbar. Der erste Prototyp funktionierte einwandfrei. In definierten Zeitintervallen wurde die Entfernung des Stabes gemessen woraus sich die Geschwindigkeit und Richtung (zum Sensor hin oder vom Sensor weg) errechnen ließ. Das Problem an dieser Lösung ist die zu geringe Reichweite unserer IR-Entfernungsmesser.
Unser Entfernungsmesser kann Entfernungen zwischen 6 cm und 30 cm erkennen. Ein Sensorgraben ist unserer Vorstellung nach mindestens einen Meter lang, erfordert also entweder einen Sensor, der in der Lage ist diese Entfernung zu messen, oder zwei Entfernungsmesser pro Graben, die sich die Grabenlänge teilen. Letzteres würde die Implementierung sehr schwierig machen, da sich gegenübersitzende Sensoren gegenseitig stören können.

5.2.2 Messstationen entlang der Strecke

Unsere nächste Idee bestand darin, die Sensorgräben in Quer- und nicht in Längsrichtung zu erfassen. Hierbei sollen viele kleine Meßstation (z.B. Lichtschranken, Hallsensoren, kapazitative Sensoren, Metallsensoren), die Position das Ruderstabes im Graben korrekt und schnell erfassen.
Die errechneten Kosten für den Kauf dieser aktiven Sensoren in ausreichender Stückzahl waren jedoch zu hoch, um diesen Ansatz realisieren zu können. Um den Ruderstab korrekt zu erfassen ist eine Meßstationsdichte von 3-5 cm erforderlich bei einer Grabenlänge von 100 cm.

5.2.3 Reed-Kontakte

Eine günstigere Variante ist die Benutzung von Reed-Kontakten an Stelle der Lichtschranken. Reed-Kontakte sind kleine Kontaktschalter, die durch magnetische Beeinflussung den Kontakt auslösen.

Am Ruderstab werden starke Dauermagnete angebracht. Wenn sich der Ruderstab neben einem Reed-Kontakt befindet, schließt dieser den Kontakt im Reed-Schalter und die relative Position des

Ruderstabes wird erkannt. Damit die Magnete des Ruderstabes nicht gleich mehrere Kontakte gleichzeitig auslösen, entschieden wir uns für einen Sicherheitsabstand von 4,5 cm. Die Reed-Kontakte wurden dann folgendermaßen geschaltet:

Um die Anzahl der benötigten Erfassungskanäle zu veringern, werden die Reed-Kontakte in sechser Gruppen (auf der Abbildung 5.1 mit 4 Gruppen) angeordnet, so dass jeder 6. Kontakt zur selben Gruppe gehört. Hiermit verringern sich die Anzahl der benötigen Erfassungskanäle auf sechs pro Sensorgraben und die Stabbewegungen werden in periodische Signalsequenzen umgesetzt: Gruppe 1, Gruppe 2, Gruppe 3, Gruppe 4, etc..
Die Bewegungsrichtung und -geschwindigkeit lässt sich aus einer Folge von nacheinander geschalteten Kontakten ermitteln. Ist die Gruppennummer steigend (3, 4, 5, 6, 1, 2, ...), so wird der Stab von „Vorne nach Hinten“ durch den Graben gezogen. Ist die Gruppennummer fallend (3, 2, 1, 6, 5, ...), so ist die Bewegungsrichtung des Stabes von „Hinten nach Vorne“. Je nach Ausrichtung des Grabens im Cave wird die Fortbewegung auf die „Kameraposition“ umgerechnet.
Die Geschwindigkeit ergibt sich aus dem Zeitintervall zwischen den einzelnen Kontaktimpulsen der einzelnen Gruppen. Je kürzer der zeitliche Abstand zwischen zwei Impulsen, desto schneller ist der Stab durch den Graben gezogen worden wodurch die Kamerabewegung beschleunigt wird.

5.3 Schnittstellen zum Computer

Zum Einlesen der Signale der Reed-Gruppen benötigt man eine Schnittstelle zum Computer. Standard-PCs verfügen ab Werk nicht über genügend viele Eingänge, um unsere vier Sensorgräben direkt anschließen zu können.

Neben speicherprogrammierbaren Steuerungen (SPS) gibt es eine Vielzahl unterschiedlicher Schnittstellenlösungen, die sich durch Preis, Anzahl der Ein- und Ausgänge sowie in ihrer Eigenintelligenz unterscheiden. Während einfache Schnittstellen nur die logischen Zustände ihrer Eingänge weiterreichen können, erledigen die intelligenteren Geräte bereits eine Vorverarbeitung der Rohdaten und entlasten dadurch den Hauptcomputer.

Wir haben zuerst Versuche mit einer kleineren SPS, der Conrad C-Control, gemacht, die uns vom artec zur Verfügung gestellt wurde.

5.3.1 Conrad C-Control

Die für uns wichtigen Leistungsmerkmalen der Conrad C-Control Station umfassen:

- programierbare Steuereinheit mit Mikrocontroller
- 6 Digitalports
- 4 Analogports zur Spannungsmessung (0...2,55V)
- serielles Interface zum Anschluss eines PCs

Die Digitalports kann man wahlweise als Eingang oder Ausgang betreiben. Die logischen Pegel betragen 0...0,7V für AUS (Low-Pegel) und 4,3...5V für EIN (High-Pegel). Die Analogports dienen zur Messung von analogen Spannungswerten. Diese Spannungswerte werden durch den Analog-Digital-Wandler des Mikrocontrollers mit einer Auflösung von 8 Bit umgewandelt und liefern somit Werte im Bereich zwischen 0 bis 255. Die obere Referenzspannung beträgt 2,55V und entspricht dem Wert 255.

Für unsere Sensorgräben benötigen wir $4 \cdot 6 = 24$ (4 Gräben mit je 6 Reed-Gruppen) digitale Eingänge. Das bedeutet in der Praxis, dass wir nur einen Graben zur Zeit an der C-Control betreiben können. Um vier Sensorgräben an einer C-Control betreiben zu können, versuchten wir die Sensorgräben an die Analogeingänge anzuschließen: Mit einer Spannungsteilerschaltung erzeugten wir aus einer 5V-Spannung 6 verschiedene Spannungen im Meßbereich der C-Control und legten sie auf die Reedgruppen.

Am Analogport liegt je nach geschalteter Gruppe eine für die Gruppe charakteristische Spannung im Bereich von 0 bis 2,55V an. In einer der ersten Messungen kann schon gut gesehen werden, wie sich die einzelnen Schaltgruppen auf den gesamten Bereich aufteilen. Dennoch sind deutliche Messfehler zu erkennen: Wenn das Ruder einen Reed-Kontakt passiert hat, darf keine Spannung mehr an der SPS anliegen, dementsprechend müsste der Wert 0 im Diagramm angezeigt werden. Allerdings, fällt die Spannung nur relativ langsam, und nicht so steil ab, wie wir erwartet haben. Nach Messungen mit einem Multimeter stellte sich heraus, dass die Analogeingänge der C-Control nie spannungslos sind, und eine Phantomspannung sowie ein offenbar in die C-Control eingebauter Kondensator unsere Messergebnisse verfälscht haben.

Durch einen zusätzlichen Widerstand parallel zum Analogeingang konnten wir die Phantomspannung etwas dämpfen und die Messergebnisse ein wenig verbessern. In dem folgenden Diagramm

(Abbildung 5.2) kann man erkennen, wie die Spannung nach dem Passieren des Schalters schon schneller abfällt.

Der nächste Schritt bestand darin, aus den Messergebnissen die Nummer der auslösenden Reedgruppe zu ermitteln. Durch weitere Messungen benannten wir Bereiche (Abbildung 5.3).

Es stellt einen zum Ende hin schneller werdenden Rudervorgang dar. Der Sensorgraben funktionierte unter Laborbedingungen.

Nach einem weiteren Test mit zwei Sensorgräben stellten wir fest, dass der A/D-Wandler der C-Control zu langsam ist, um die Spannungen mehrerer Schaltvorgänge klar voneinander zu trennen.

5.3.2 Der IO-Warrior – die digitale Lösung

Nach den Problemen mit der analogen Abfrage der Reedgruppen in den Sensorgräben über die Conrad C-Control beschlossen wir, das USB-Interface IO-Warrior der Firma Code Mercenaries [1] anzusehen, über das wir den c't-Artikel [Mey03] gefunden hatten.

Der IO-Warrior ist ein Single-Chip-Prozessor, der über den USB-Port (Universal Serial Bus) an den PC angeschlossen wird und mit minimaler externer Beschaltung die unabhängige Steuerung von bis zu 32 I/O-Pins gestattet. Jedes Pin kann sowohl als digitaler Eingang (Input) als auch als digitaler Ausgang (Output) fungieren.

Der Zustand aller Ports kann ca. 100 mal pro Sekunde gesetzt oder gelesen werden, die genaue Rate hängt vom verwendeten Betriebsystem ab.

Neben den nackten ICs bietet Code Mercenaries auch ein Starter Kit an, das neben dem IC auch eine vorbereitete Platine, die zusätzlichen Bauteile (Quarz, USB-Buchse etc.) und eine Testumbebung (LEDs und einen Schalter) enthält. Zusätzlich liegt eine CD mit Treibern und Programmierbeispielen für alle wichtigen Betriebsysteme und Programmiersprachen bei (siehe unten zum Punkt Software). Das komplette Softwarepaket kann aber auch über die Herstellerhomepage bezogen werden, so dass der Kauf eines Starter Kits nicht unbedingt notwendig ist.

Zur Zeit ist der IO-Warrior in zwei verschiedenen Varianten lieferbar:

[1] (http://www.codemercs.com/)

IOW40
Der IOW40 bietet 32 I/O-Pins und enthält Spezialfunktionen zur Ansteuerung von I2C-Peripherie-Bussen und HD44780-kompatiblen LCD-Displays.

IOW20
Der IOW20 ist die kleine Version, sie unterstützt lediglich 16 I/O-Ports, unterstützt dafür über die Spezialfunktionen das IOW40 hinaus auch die Dekodierung von RC5-Infrarotfernbedienungssignalen und enthält ein SPI-Interface.

Die Programmierung des IO-Warriors erfolgt unter Linux über ein Kernelmodul, welches auf Unix-typische Weise über Gerätedateien im `/dev`-Verzeichnis angesprochen wird. Das Kernelmodul scheint leider noch nicht ganz ausgereift zu sein, gelegentlich hängt sich der PC komplett auf (Kernel Panic). Detaillierte Informationen zur Programmierung und Beispielprogramme befinden sich auf der IO-Warrior-CD.

5.3.3 Der IO-Warrior im Cave

Hardware

Wir haben ein IOW40-Starter Kit gekauft, da ein Komplettbausatz inklusive professionelle Platine eine höhere Betriebssicherheit gewährleistet als eine ausschließlich mit Drahtbrücken verbundene Universalplatine.

Da der IO-Warrior ausreichend Eingänge für alle vier Sensorgräben und eine zusätzliche Funkfernbedienung bietet, ist die Verdrahtung sehr simpel. Alle Reedgruppen und auch der 8-Kanal-Fernbedienungsempfänger (1 Schaltkontakt pro Taste) können direkt angeschlossen werden.

Software

Um die Meßwerte der Sensorgräben zur Floßsteuerung einsetzen zu können, müssen diese zunächst aufbereitet und interpretiert werden. Diese Aufgabe übernimmt die von uns entwickelte Software, über die das Klassendiagramm (Abbildung 5.4) und die Funktionsbeschreibung (Tabelle 5.1) einen Überblick geben.

Klassenname	Funktion
Raft	Hauptklasse und Schnittstelle zu den anderen Cave-Systemen. Enthält u.a. die Floßphysik.
RaftHardwareLayer (RHL)	Bindeglied zwischen Raft und Sensorgraben.
RaftEmulationLayer (REL)	Emulation des Floßes für Testzwecke (nicht implementiert)
Sensorgraben	Auswertung der Signale der Sensorgräben, z.B. Geschwindigkeitsberechnung.
IOWarrior	Schnittstelle zum IO-Warrior Kernelmodul

Tabelle 5.1: Funktionsbeschreibung

Die Integration der Floßsensorik in das Hauptprogramm erfolgt ausschließlich über die Klasse Raft. Alle anderen Klassen sind Raft untergeordnet und werden nur intern benutzt. Diese modulare Aufteilung hat den Vorteil, dass die Floßsteuerung leicht an verschiedene Hardwaresysteme angepasst werden kann, falls z.B. noch ein weiteres Hardwareinterface zum Einsatz kommen soll. Zur Zeit existiert lediglich das Modul RaftHardwareLayer, das mit Hilfe der Klassen Sensorgraben und IO-Warrior die Floßsteuerung über die oben beschriebene Hardware realisiert.

Ursprünglich war auch eine Klasse RaftEmulationLayer geplant, die für Testzwecke eine Floßsteuerung über normale Eingabegeräte wie Tastatur, Maus oder Joystick ermöglichen sollte. Im Laufe des Projektes stellte sich jedoch heraus, dass eine Emulation nicht benötigt wurde und auf eine Implementation dieser Klasse daher verzichtet werden konnte. Dennoch erlaubt die modulare Architektur der Floßsteuerung eine problemlose spätere Nachrüstung dieses oder ähnlicher Features.

Als Testhilfe bei Problemen mit der Steuerung steht das Programm sgtest zur Verfügung, welches die Rohdaten aller Sensorgräben auf der Konsole ausgibt. Wenn sich das Floß beispielsweise scheinbar selbstständig bewegt, ist dies oft auf einen kurzgeschlossenen Reedkontakt zurückzuführen, der mit Hilfe dieses Testprogramms lokalisiert werden kann.

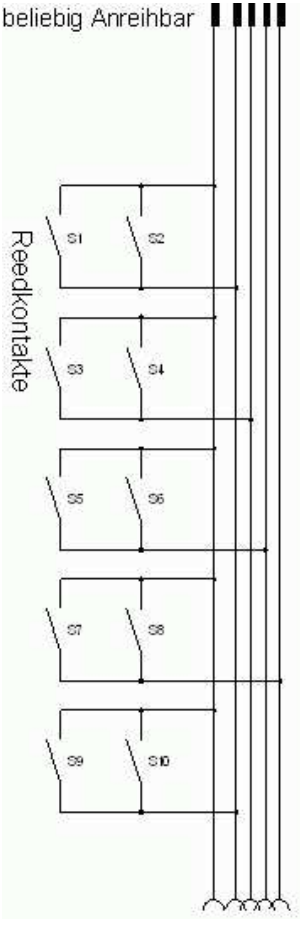

Abbildung 5.1: Schaltplan der Reedkontakte

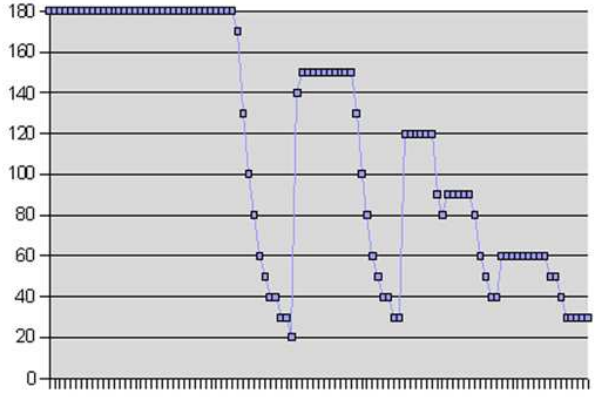

Abbildung 5.2: A/D-Werte im zeitlichen Verlauf

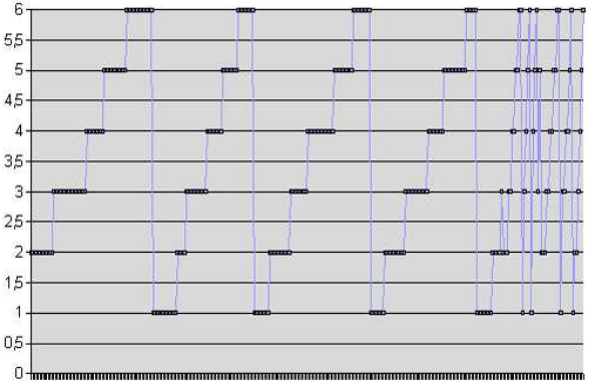

Abbildung 5.3: Bereiche der einzelnen Reedgruppen

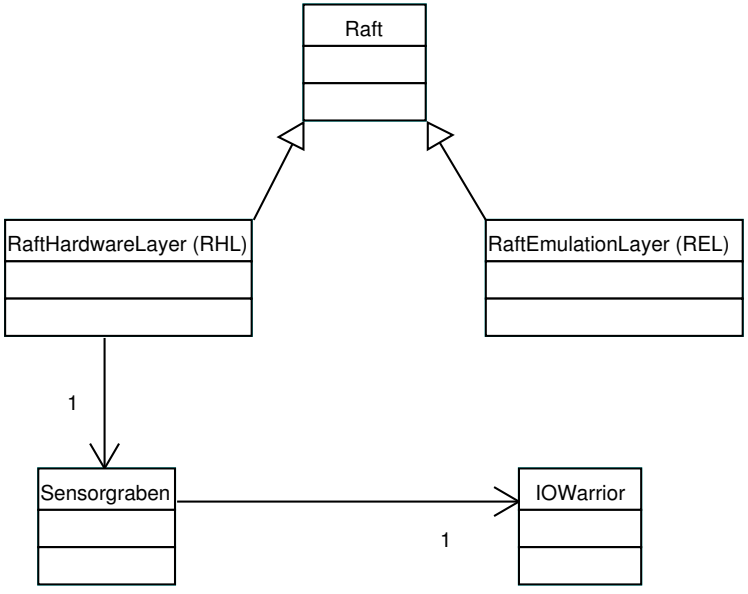

Abbildung 5.4: Klassendiagramm

— KAPITEL 6 —

Interaktionen mit den Kreaturen

Der Begriff Interaktion bezieht sich in diesem Artikel lediglich auf die Schnittstelle zwischen dem Benutzer und den Tieren der virtuellen Welt. Eine weitere Schnittstelle innerhalb des Caves zwischen realer und virtueller Welt wird durch die Floß-Steuerung (siehe Abschnitt 5.2) bereitgestellt.
Die hier thematisierte Interaktion ermöglicht es dem Benutzer, mit den Bewohnern des Dschungels in gewisser Weise zu kommunizieren und sie zu bestimmten Tätigkeiten zu animieren. Es wird versucht, die virtuelle Welt durch die Möglichkeit der Interaktion für den Benutzer realer erscheinen zu lassen. Je undeutlicher die Grenze zwischen der virtuellen Welt und dem Benutzer verläuft und je weniger Unterschiede dieser zwischen der Virtualität und seiner Realität empfindet, desto mehr wird er sich in die fremde Welt hineinversetzt finden. Die Beeinflussung dieser Umwelt durch Interaktion ist dabei natürlich ein wichtiger Aspekt, da sie der Benutzer in seiner realen Welt täglich erfährt. Eine auf diesen Grundsätzen basierende ideale Mixed-Reality-Umgebung wäre somit eine, in der die Vermischung unsichtbar wird.

6.1 Grundsätzlicher Interaktionsablauf

Die Kreaturen in der virtuellen Welt führen auf der Grundlage einer künstlichen Intelligenz (KI) eigenständig unterschiedliche Aktionen aus (siehe Kapitel 8). Ihnen geht es dabei primär um das Befriedigen ihrer grundlegenden Bedürfnisse, es seien hier nur Essen, Trinken und Schlafen genannt.

Dem Benutzer ist es möglich, aktiv in den „Tagesablauf“ einer Kreatur einzugreifen. Die Reaktion auf solch eine Initiative des Benutzers ist das Ausführen spezieller Aktionen seitens der Kreatur, die bis auf eine Ausnahme[1] ansonsten nicht eigenständig aus dem bedürfnisorientierten Grundverhalten selbiger hervorgeht. Es ist grundsätzlich diejenige Kreatur dem Benutzer ein Gegenüber bei der Interaktion, die sich ihm am nahesten befindet. Sie führt die entsprechende Aktion daraufhin unmittelbar aus, damit der Benutzer seiner initiativen Handlung ein direktes Feedback zuordnen kann. Einige dieser Re-Aktionen sind zeitlich begrenzt, während andere erst durch das Eintreten bestimmter Ereignisse, sei es nun erneut auf Initiative des Benutzers oder aus der Welt heraus, beendet werden. Es gilt allerdings jeweils, dass die Kreatur wieder in ihr eigenständiges Verhaltensmuster zurückfällt, sobald der Benutzer sich nicht mehr mit ihr beschäftigt. In einem bestimmten Fall (im Anschluss an das „Anlocken“, siehe Abschnitt 6.2) ist dafür jedoch zunächst eine anschließende Aktion zur Orientierung und Positionierung in der Welt notwendig.

6.2 Realisierung der Interaktion

Es wurde einleitend von Interaktion durch Initiative des Benutzers gesprochen, ebenso wurde erwähnt, dass die Kreaturen ihre Aktionen eigenständig auf Grundlage einer KI auswählen. Um nun im Folgenden die genaue Funktionsweise einer Initiative und ihre Art der Wirkung auf die KI der Kreaturen beschreiben zu können, ist es wichtig, den Begriff des Motivationslevels einzuführen. Es ist dabei in diesem Kontext ausreichend, ein solches zunächst als simplen Zahlenwert anzusehen, dem jeweils eine Aktion zugeordnet ist. Diese Zahlenwerte unterliegen Veränderungen, eine Kreatur führt immer diejenige Aktion alleinstehend aus, deren Motivation (Zahlenwert) am größten ist. Eine detailliertere Sicht auf die Motivationslevel und wie sie das Ausführen von Aktionen bestimmen, befindet sich im Kapitel 8. Es liegt nun nahe, Kreaturen zum Ausführen bestimmter Aktionen zu bewegen, indem man auf ihre Motivationslevel einwirkt. An dieser Stelle setzen die *Events* an. Prinzipiell wird eine Kreatur vom Eintreten eines bestimmten Ereignisses informiert und reagiert darauf. Konkret ist ein Event ein im `main`-Modul der Cave-Anwendung getätigter Funktionsaufruf auf der jeweiligen (empfangenden) Kreaturklasse. Das Ausführen dieser Funktion wirkt dann auf die kreatureigenen Motivationslevel ein und veranlasst die Kreatur in Folge dessen zum Ausführen einer bestimmten Aktion. Es ist somit grundsätzlich möglich, dass unterschiedliche Kreaturklassen ihre Motivationslevel aufgrund eines Event-Empfanges unterschiedlich abändern. Das `main`-Modul ist bei diesem Verfahren jeweils der Ausgangspunkt des Funktionsaufrufes, da in ihm etwaige Tasta-

[1] Die Ausnahme bildet die Aktion „Folgen“. Eine Kreatur kann aus Eigenantrieb dem Benutzer folgen. Siehe Abschnitt 6.2

tureingaben ausgelesen werden; die Schnittstelle für eine Initiative des Benutzers.[2]
Grundlegender Ablauf ist somit, dass der Benutzer zum Auslösen einer Interaktion mit einer Kreatur eine Taste drücken muss, was im `main`-Modul registriert wird. Das Modul seinerseits bringt durch das Event-System die Kreatur dazu, eine Funktion auszuführen, die auf ihre Motivationslevel einwirkt und sie somit veranlasst, eine bestimmte Aktion als Reaktion auf die Benutzerinitiative auszuführen.

Unterstehender Code konkretisiert diese Wirkungskette anhand der Implementation:

Datei *main.cpp:*

```
/// Handles SDL events
bool handle_events(double delta_t) {
  SDL_Event event;

  while (SDL_PollEvent(&event)) {
    switch(event.type) {
    case SDLK_2:
          {
            wakeupCreature();
            if (micado_spectator::getActive())
              micado_spectator::getActive()->playSound(MICADO_SOUND_WAKEUP);
          } break;
}

// Ermittelt und benachrichtigt die (zum Benutzer naechste) Kreatur,
// die aufgeweckt werden soll
void wakeupCreature() {
  double temp_distance;

  // naechste Kreatur ermitteln
  if(micado_creature::num() > 0) {
    micado_creature* act_creature = 0;
```

[2]Wir werden später sehen, dass das Auslösen von Events durch den Benutzer über eine spezielle Tastatur, eine Druck-Fernbedienung geschieht.

```
    double act_distance = MADNESS_FAR_AWAY;

    for(int i = 0; i < micado_creature::num(); ++i ) {
      micado_creature *c = micado_creature::get(i);
      // nicht sich selber (den Benutzer) finden, da dieser auch Kreatur ist
      if (!c || c->isOfType(&micado_spectator::type))
                continue;
          // Entfernungsberechnung Benutzer - Kreatur
      temp_distance = madness_vec3d::squaredDistance(c->getPosition(),
       madness_viewpoint->getPosition());
      if(temp_distance < act_distance) {
                act_creature = c;
                act_distance = temp_distance;
      }
    }

    // Event an diese Kreatur senden:
    if (act_creature)
      act_creature->event_wakeup(0, 0);
  }
}
```

Datei *micado_creature1.cpp:*

```
void micado_testcreature::event_wakeup(micado_creature *sender,
          micado_creature *wakeupped_by) {

  // Veringerung des Motivationslevels fuer Schlafen um 20%
  levels.val[MICADO_MOTIVATION_SLEEP] *= 0.8;
}
```

Es bleibt noch ein letzter, wichtiger Punkt zu nennen. Um die Interaktion mit den Kreaturen einen Schritt intuitiver und begreifbarer zu machen, soll der Benutzer jeder seiner Aktionen eine Reaktion zuordnen können, sprich ein individuelles Feedback erhalten. In dem Cave bekommt er deshalb gleich über zwei Sinne Feedbacks auf einen Tastendruck, ein visuelles sowie ein akustisches. Das

visuelle Feedback bezieht sich auf die Reaktion der Kreatur, mit der die Interaktion abläuft, d.h. die Aktion, die aufgrund des erhaltenen Events auszuführen ist, muss unmittelbar gestartet werden, um dem Benutzer den direkten Bezug zu seiner Initiative aufzuzeigen. Zu diesem Zwecke erhöhen Events in ihrer Ausimplementierung die Motivation für die entsprechende Interaktions-Aktion höher, als es die anderen Motivationen in diesem Moment sein können, über eine Maximalgrenze hinaus also. Die genaue programmiertechnische Umsetzung wird im Kapitel 8 aufgegriffen und im dortigen Kontext erläutert. Als akustisches Feedback ertönt zu jedem Tastendruck ein entsprechendes, deutlich wahrnehmbares und gegenüber den anderen gut unterscheidbares Geräusch. Es werden nun die Events zur Realisierung der Interaktion einzeln beschrieben. Dabei wird auf die konkrete Ein- bzw. Auswirkung auf die Motivationslevel und die daraus resultierende Aktion sowie die entsprechenden Feedbacks eingegangen.

- `event_scare` : Dieses Event wirkt steigernd auf `MICADO_MOTIVATION_FLEE`. Die Initiative des Benutzers ist also das Verjagen einer Kreatur. Diese startet die Aktion `micado_std_flee`, infolge derer sie versucht, vom Benutzer weg zu flüchten. Sie erhöht ihr Tempo[3] und wählt auf Grundlage der Restriktionen des Pathfindings (siehe Kapitel 7) denjenigen Weg, der am besten vom Benutzer wegführt. Diesem Fluchtweg folgt sie normalerweise bis zum nächsten Wegpunkt[4], das Erreichen desselben ist die Abbruchbedingung der Aktion. Die Kreatur kann dieses Event allerdings mehrfach empfangen, woraufhin sie eine weitere Strecke über mehrere Wegpunkte (jeweils einen Wegpunkt weiter) fliehen wird. Hierbei existiert eine gewisse Beschränkung der möglichen Empfangsanzahl (ähnlich Semaphoren), um eine Kreatur also durch den gesamten Dschungel zu jagen, muss der Benutzer schon hinter ihr her sein und von Zeit zu Zeit das Event erneut aussenden.
  Das visuelle Feedback wurde bereits beschrieben, als akustische Rückkopplung ertönt ein abschreckender Hupton (der so einigen bekannt sein dürfte).

- `event_follow`, `event_dismiss` : Beide Events zusammen implementieren das Anlocken und Ziehenlassen einer Kreatur, sie werden im Normalfall jeweils abwechselnd versandt und liegen daher auf derselben Taste. Drückt der Benutzer diese also zum ersten mal, wird das `event_follow` ausgelöst, welches auf `MICADO_MOTIVATION_FOLLOW` wirkt, die Motivation steigt. Die Aktion `micado_std_follow` wird gestartet und die Kreatur folgt dem Benutzer auf direktem Wege. Nähert sie sich dabei bis auf einen Mindestabstand, setzt sie sich und wartet. Bewegt sich der

[3] Die Geschwindigkeiten beim Gehen und Rennen werden in den `cfg-Files` angegeben

[4] Ein Weg besteht aus mehreren geraden Abschnitten, die durch die so genannten Wegpunkte begrenzt und zusammenhängen. Der genaue Zusammenhang ist im Kapitel 7 Pathfinding nachzulesen.

Benutzer wieder, folgt sie ihm weiterhin. Ein erneuter Tastendruck löst nun letzteres Event aus, welches ebenfalls (nur diesmal gegenteilig) auf `MICADO_MOTIVATION_FOLLOW` wirkt. Im gleichen Zuge wird aber `MICADO_MOTIVATION_FINDBACK` erhöht, somit ist sichergestellt, das die Aktion `micado_std_findback` startet. Der genaue Sachverhalt der Aktion `micado_std_findback` und u.a. der mit dieser Aktion im Zusammenhang stehenden Aktion `micado_std_respawn` wird im dem Abschnitt 8.4 Kreaturbewegung erklärt.
Im Normalfall bewirken nacheinander ausgeführte Tastendrucke so alternierend das Aussenden der Events `event_follow` sowie `event_dismiss`. Durcheinander kommt diese Reihenfolge nur, wenn die Kreatur von sich aus das Folgen abbricht, was sie tut, wenn sie auf ein Hindernis (etwa einen Baum) stößt. Dann bewirkt ein erneuter Druck natürlich wieder das Anlocken. Eine noch weiterführende Verfeinerung dieses Zusammenspieles beider Events lässt sich durch das Kreatur-Attribut `ATTRIB_OPENESS` herbeiführen (siehe Kapitel 8)
Beide Events werden schließlich von eigenen, aber doch zusammengehörenden, akustischen Feedbacks begleitet. Eine aufsteigende Panflöten-Tonfolge verdeutlicht das Anlocken, während eine absteigende (umgekehrte) Folge das Ziehenlassen begleitet.

- `event_wakeup` : Dieses Event wirkt durch das Absenken der Motivation `MICADO_MOTIVATION_SLEEP` zwar immer gleich, trotzdem kann es sich auf zwei verschiedene Arten auswirken. Erreicht es eine schlafende Kreatur, so wird sie (durch das Absenken ihrer Müdigkeit) bald erwachen und eine andere Aktion ausführen. Der Vorgang kann durch mehrfaches „aufwecken“ beschleunigt werden. Durch genau dieses Vorgehen kann aber ebenso eine bereits wache Kreatur dauerhaft wach gehalten werden, denn ihre ständig niedrig gehaltetene Müdigkeit wird nie eine der anderen grundlegenden Motivationen übersteigen.
  Als akustisches Feedback der Wirkung des Events ertönt bei der Ausführung ein lautes Hahnenkrähen.

Das reale Mittel, das dem Benutzer zur Interaktion mit den Kreaturen zur Hand gegeben wird, ist natürlich keine Computer-Tastatur (Wo sollten sich im Dschungel auch Anschlüsse dafür finden). Tatsächlich reicht eine kleine 8-Tasten-Fernbedienung (siehe Abschnitt 5.3.3) aus, die durch ihre geringe Größe auch die Handhabung des Ruderstabes (siehe Abschnitt 5.2) nicht beeinträchtigt. Jeder Tastendruck wird per Funk an den Rechner übertragen, im `main`-Modul registriert und löst dort die oben beschriebenen Events aus. Die Fernbedienung ist keine Selbstentwicklung (die Empfangsanlage dagegen schon), sondern ist ein Standardprodukt, welches uns vom Institut artec zur Verfügung gestellt wurde.

— KAPITEL 7 —

Pathfinding

Unser virtueller Dschungel besteht aus den unterschiedlichsten Pflanzen, riesigen Bäumen und vielen verschiedenen nie zuvor gehörten Geräuschen. Doch erst durch seine Bewohner, die merkwürdigen, skurrilen und farbenprächtigen Kreaturen, wird dieser zum Leben erweckt. Sie bewegen sich von einem Ort zum nächsten, trinken Wasser aus dem Bach, schlafen im Schatten eines Baumes und essen die Früchte der verschiedenen Sträucher.

So schön und einfach sich dieses Szenario auch beschreiben läßt, bedurfte es bei der Umsetzung einiger Vorüberlegungen: Wie werden spezielle Regionen der Welt zum Beispiel als Futterstellen, Schlafplätze, etc. kenntlich gemacht? Woher wissen wir, wie die Kreatur vom Schlafplatz zu einer Trinkstelle kommen kann und ob dies der kürzeste Weg ist? Wie werden Hindernisse erkannt und wie wird diesen ausgewichen? Ist ein Bereich für die Kreatur begehbar oder würde sie dann beispielsweise über Wasser gehen?

Wir benötigten also verschiedenste Informationen über die Geometrie der virtuellen Dschungelwelt, wie zum Beispiel welche Polygone sind für die Kreaturen begehbar und welche genauen Koordinaten hat der Bereich, in dem die Kreatur trinken kann. Außerdem benötigten wir Algorithmen, mit denen wir einen Weg durch die Geometrie der virtuellen Dschungelwelt finden können. Hinzu kamen einige zu beachtene algorithmische Aspekte, wie zum Bespiel der Rechenaufwand eines Wegfindungskonzeptes und die Möglichkeit, eventuell Teile der Algorithmen offline, also vor dem eigentlichen Ablaufen der Simulation, auszuführen. Um einen für uns geeigneten Wegfindungsalgorithmus zu finden, haben wir einige Konzepte, auf die wir später noch genauer eingehen wer-

den, betrachtet. Zunächst jedoch geben wir noch einen Überblick über die Zusammensetzung eines Wegfindealgorithmus. Abschließend wird das Konzept und die Umsetzung des von uns gewählten Wegpunktalgorithmus behandelt.

7.1 Struktur eines Wegfindealgorithmus

Wegfindealgorithmen lassen sich meist grundlegend in eine Hierachie mit drei Ebenen gliedern [Sur02, S. 161–162]. Auf der ersten Ebene muss die virtuelle Welt abstrahiert und partitioniert werden. Rasterung, Wegpunktesystem und Navigationsgitter sind Beispiele hierfür, die wir im nächsten Abschnitt noch genauer erklären werden. Der entstehende Suchraum kann dann von einem Algorithmus durchlaufen werden, der in diesem Suchraum Knoten und Kanten definiert. Ein Knoten entspricht dabei einer Sektion bzw. einem Ort in der Welt, eine Kante gibt eine Verbindung zwischen diesen Knoten an. Eine Folge von Knoten und Kanten beschreibt dann einen Weg.

Die zweite Ebene dient der Abschätzung der Güte eines Weges. Die Länge eines Weges kann auf einfache Weise durch die Anzahl seiner Kanten beschrieben werden, was aber bei ungleich langen Kanten eher wenig aussagekräftig ist. Meistens werden deshalb die einzelnen Kanten mit Kosten belegt. So können beispielsweise einer langen Kante höhere Kosten zugewiesen werden als einer kurzen, ebenso einer Kante, die einen „steilen“ Wegabschnitt beschreibt gegenüber einer „ebenen“ Kante. Die Summe dieser Kosten beschreibt dann die Kosten des betrachteten Pfades. Die Pfadauswahl variiert also, je nachdem um was für eine Welt es sich handelt und was die Kriterien einer sich in ihr befindlichen Kreatur sind.

Auf der dritten Ebene befindet sich der Suchalgorithmus selbst. Seine Aufgabe ist es, auf Anfrage den bestmöglichen Pfad von einer beliebigen Position (Knoten) zu einem beliebig definierten Ziel (Knoten) zu ermitteln. In der Künstlichen Intelligenz wird zu diesem Zweck meist der so genannte A^*-Algorithmus [RN03, S. 97–104] benutzt, welcher unter Benutzung der vorliegenden Informationen einen optimalen Weg berechnet. Er tut dieses unter Berücksichtigung des bereits zurückgelegten Weges und einer Abschätzung des weiteren Weges zum Ziel (Luftlinienfunktion). Damit der Algorithmus arbeiten kann, muss diese Funktion unterschätzend sein, d.h. ihr Ergebnis muss immer geringer sein, als der tatsächlich noch zurückzulegende Weg. Wenn man bei der Wegewahl die Luftlinie des nächsten auszuwählenden Punktes zum Zielpunkt benutzt, so ist diese Bedingung erfüllt. Grundlegend gilt für den Suchalgorithmus, dass er möglichst wenig Rechenleistung der CPU in Anspruch nehmen soll.

An dieser Stelle sollte über die Verwendung eines *Preprocessing-Tools* nachgedacht werden, d.h. ob nicht Teile oder sogar ganze Algorithmen vor der eigentlichen Cave-Anwendung ausgeführt werden können. Runtime-Werkzeuge müssen bei jeder Anfrage nach einem Pfad quasi wieder bei Null beginnen. Sie legen Speicherplatz für nahezu jeden der Knoten an, die zur Pfadberechnung verarbeitet werden müssen. Jede Anfrage – diese können natürlich auch parallel auftreten – löst dabei ein solches Verfahren aus, was den großen Nachteil dieser Methode ausmacht. Eine Möglichkeit eines *Preprocessing-Tools* ist es deswegen, eine Tabelle mit allen möglichen Pfaden anzulegen. Die Komplexität sinkt dadurch bei einer Pfadanfrage, da die Pfade direkt aus der Tabelle entnehmbar sind. Die vorverarbeitende Berechnung kann Informationen über das jeweilige Gelände sowie Verbindungsinformationen und Anwendungsdaten der partitionierten Welt mit einbeziehen. Allerdings muss sich hierbei auf rein statische Informationen der virtuellen Welt beschränkt werden. Die Navigation betreffende Änderungen zur Laufzeit, wie zum Beispiel eine plötzlich im Weg stehende andere Kreatur, können nicht berücksichtigt werden.

7.2 Möglichkeiten der Navigationsdatenerzeugung

Wie im vorigen Artikel bereits angesprochen, muss in der ersten Ebene eines Wegfindealgorithmus die virtuelle Welt abstrahiert und partitioniert werden. Es müssen Navigationsdaten erzeugt werden. Rasterung, Wegpunktesystem und Navigationsgitter sind Beispiele hierfür, die wir nun im folgenden näher erläutern möchten.

7.2.1 Rasterung

Ein erster Ansatz, den wir für Erhebung von Navigationsdaten in Betracht gezogen haben, ist die Einteilung der gesamten Fläche des virtuellen Raumes in ein schachbrettartiges Raster [Rab02, S. 264–265]. Für jedes einzelne Quadrat, das sich dadurch ergibt, wird nun jeweils festgelegt, ob es im einfachsten Fall für die Kreaturen begehbar ist oder nicht. Liegt es beispielsweise im Wasser oder handelt es sich dabei um eine Mauer, wird es dementsprechend markiert. In komplexeren Ansätzen ließe sich zusätzlich definieren, welche Eigenschaften einer Kreatur zum Betreten der jeweiligen Fläche nötig sind. Eine Kreatur, die schwimmen kann, könnte dann natürlich eine Wasserfläche betreten; eine ohne diese Fähigkeit bliebe wiederum davor stehen. Für die Berechung der Wege, welche Kreaturen auf diesem Raster zu ihrem angestrebten Ziel bringen, müssen geeignete Wegewahlalgorithmen eingesetzt werden, wie zum Beispiel der A^*-Algorithmus.

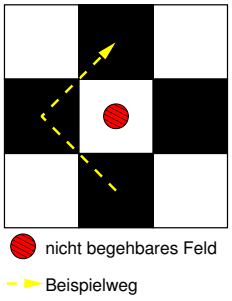

Abbildung 7.1: Beispielweg, erstellt mit Hilfe von Rasterung und Wegewahlalgorithmus

Abbildung 7.1 zeigt ein Beispielraster mit einem als nicht begehbar markiertem Feld und einen Beispielweg, der durch einen Wegewahlalgorithmus ermittelt wurde.

Der größte Nachteil dieses Ansatzes liegt in dem Problem, dass eine gute Rastergröße sehr schwierig festzulegen ist. Stellt man sich beispielsweise vor, dass ein Stein irgendwo in einem Quadrant des Rasters liegt, kann dieser dazu führen, dass der komplette Quadrant des Rasters als unbegehbar markiert wird, falls die Rastergröße zu groß gewählt wird. Kleinere Raster führen aber irgendwann zu Komplexitätsproblemen, da für unseren Zweck die Welt eine gewisse Größe besitzen muss und wir gleichzeitig für die sinnvolle und realistische Bewegung in ihr ein kleines Raster benötigen. Die Informationmenge ist dadurch bei diesem Ansatz zu groß.

Durch die Rasterung der Welt ergibt sich ein weiteres Problem: Es lassen sich keine Höheninformationen berücksichtigen. Darauf stößt man z.B. dann, wenn eine Brücke in die Welt eingebaut werden soll. Da es für die Wasseroberfläche unter der Brücke und den Weg auf der Brücke jeweils nur ein Rasterfeld als Repräsentation geben würde, ließe sich nicht mehr eindeutig festlegen, ob der Quadrant nun begehbar ist oder nicht.

7.2.2 Wegpunktesystem

Wegpunkte sind eine weitere Möglichkeit Navigationsdaten zu erzeugen. Hierzu werden innerhalb der virtuellen Welt bestimmte Koordinaten als Wegpunkte deklariert. Aus diesen Wegpunkten kann dann, in Abhängigkeit von der Beschaffenheit der Welt, ein Graphennetz erstellt werden. Die Wegpunkte entsprechen hier den Knoten des Graphen, die Kanten repräsentieren die möglichen Wege

innerhalb der Welt. Jeder Knoten besitzt zu den Knoten Kanten, die direkt, d.h. auf geraden Wege von Wegpunkt zu Wegpunkt durch Welt, erreicht werden können. Da das Festlegen der Wegpunktkoordinaten in Anlehnung an die virtuelle Welt erfolgen muss, bietet es sich an, dieses direkt im Level-Editor[1] zu tun, mit dem auch die Welt erstellt wird.

Ein großer Nachteil dieser Möglichkeit ist, dass die Wegpunkte nur schwer automatisch oder wenigstens teilweise durch einen Algorithmus gesetzt werden können. Dies hätte gerade in sehr großen, virtuellen Welten einen Vorteil, da dann der Level-Designer[2] nicht alle Wegpunkte per Hand setzen müsste.

Desweiteren bringt diese Methode auch das Problem mit sich, dass das Wegpunktesystem nur Informationen über die geraden Wege zwischen den Wegpunkten besitzt, jedoch keine über den unmittelbaren Bereich neben den Wegen. Also beschränkt sich der Bewegungsspielraum auf die geraden Wegen.

7.2.3 Navigationsgitter

Eine weitere sehr interessante Möglichkeit, Daten zur Navigation (Suchraum) in eine virtuelle Welt zu bekommen, ist das Navigationsgitter, engl. *Navigation Mesh* [Toz02, S. 171–185]. Es handelt sich hierbei um eine Zusammenstellung von konvexen Polygonen, welche die gesamte begehbare Oberfläche in der Welt beschreiben. Auf Grundlage dieses Navigationsgitters könnte in unserem Fall eine Kreatur ihren Weg von A nach B planen und dabei den gesamten begehbaren Raum zur Wegsuche mit einbeziehen. Durch diese große Bewegungsfreiheit der Kreatur ist eine sehr realistische Bewegung von einem Punkt zum nächsten gegeben.

In Abbildung 7.2 und 7.3 sind zwei identische Räume mit jeweils einer Säule in der Mitte dargestellt, wobei im linken die Navigationsmöglichkeiten bzw. der Navigationsraum über Wegpunkte und im rechten über Navigationspolygone dargestellt sind.
Es ist hierbei sehr gut zu sehen, dass die Navigationsmöglichkeiten in dem Wegpunktesystem deutlich eingeschränkter sind. Die Kreaturen sind hier nur in der Lage über die Wegpunkte zu navigieren. Außerhalb des Wegpunktesystems (z.B. neben einer Verbindung zweier Wegpunkte) besitzen sie keine Informationen über ihre Umgebung. Das Navigationsgitter hingegen überspannt den gesamten Raum, so dass eine Kreatur jeder Zeit genaue Informationen über die Umgebung besitzt. Innerhalb der Grenzen eines Polygons ist sogar eine völlig freie Bewegung der Kreatur gewährleistet.

[1] Ein Level-Editor ist ein Editor, mit dem eine virtuelle Welt erstellt werden kann.
[2] Als Level-Designer wird die Person bezeichnet, die die virtuellen Welten am Level-Editor erstellt.

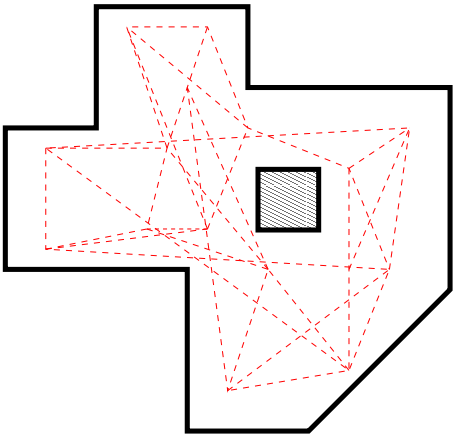

Abbildung 7.2: Navigationsraum mit Wegpunkten

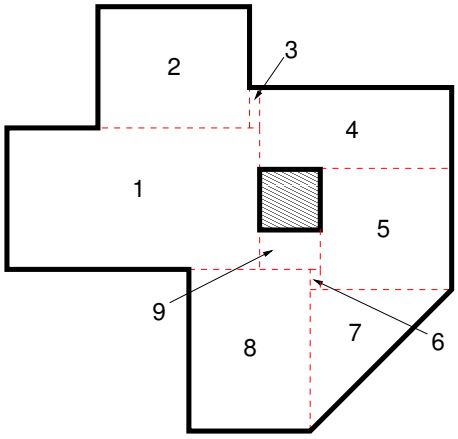

Abbildung 7.3: Navigationsraum mit Navigationsgitter

Ein Navigationsgitter, ein Wegpunktesystem oder eine Rasterung beschreiben nur die Navigationsmöglichkeiten in der statischen Umgebung. Jegliche bewegliche Objekte, wie z.B. andere Kreaturen, rollende Steine usw., können nicht erfasst werden und müssen gesondert behandelt werden. Die Berechnung eines Ausweichweges um die beweglichen Objekte herum wird zur Laufzeit vorgenommen. Ein Navigationsgitter hat allerdings einen großen Vorteil gegenüber dem Wegpunktesystem, denn hier ist der gesamte mögliche Raum zur Navigation bekannt. Kreaturen können so flexibler dem beweglichen Objekt ausweichen.
Ein weiterer Vorteil des Navigationsgitters ist die Unabhängigkeit vom Level-Designer, denn die Polygone können durch Algorithmen gefunden werden und müssen nicht vom Level-Designer eingearbeitet werden.

7.3 Routenberechnung mit dem `pathtool`

Wir haben uns letztendlich für das Wegpunktesystem entschieden. Ein Grund hierfür ist die Möglichkeit die Wege nicht zwangsläufig zur Laufzeit der Cave-Anwendung berechnen zu müssen, sondern

alle möglichen Wege vor zu berechnen. Die Engine wird dann durch derartige Berechnungen nicht mehr ausgebremst. Es ist also auch völlig unproblematisch, dass wirklich zunächst jeder Weg berechnet wird, auch solche, die möglicherweise nie benutzt werden. Der Arbeitsaufwand schien uns außerdem bei der Implementierung des Wegpunktesystems im Gegensatz zu einer Realisierung des Navigationsgitters eher abschätzbar.

Auf Grundlage des Wegpunktesystems können wir durch entsprechende Algorithmen eine so genannte *Wegpunkt-Routingtabelle* aufbauen, die zu allen Wegpunkten jeweils Informationen über Nachbarpunkte und einen Verweis auf den Wegpunkt, der zu einem gewünschten Zielwegpunkt der günstigste ist, besitzt. Von diesem Wegpunkt wird eine Kreatur dann wieder zum jeweils nächsten weitergeschickt bis diese beim Zielwegpunkt angekommen ist. Die Idee zu einem derartigen „Verkehrsleitsystem“ fußt auf dem Grundprinzip der in der Welt der Rechnernetze bekannten Routing-Verfahren. Hierbei entspricht ein Wegpunkt einem Router und eine Kreatur einem zu routenden Paket.

Es stellt sich allerdings noch die Frage, auf welche Art und Weise die Wegpunkte in der virtuellen Welt angelegt und verteilt werden können. Darüber hinaus müssen zwischen den Punkten (die Knoten des Wegpunktgraphen) auch die Kanten gezogen werden, auf denen die Kreaturen laufen sollen.

7.3.1 Erstellung des Wegpunktgraphen

Ein möglicher Ansatz zur Verteilung der Wegpunkte wäre, für diese Aufgabe eigens einen dedizierten Editor zu implementieren, mit Hilfe dessen die Wegpunkte beispielsweise mit der Maus direkt in der Welt platziert und die Kanten zwischen ihnen per Hand aufgespannt werden können. Die Verteilung der Wegpunkte durch den menschlichen Benutzer erscheint sinnvoll, da dieser so große Freiheit und Kontrollmöglichkeit bezüglich der Gestaltung der Kreaturnavigation in der Quake III Map (siehe Abschnitt 3) hat. Dem gegenüber stehen automatisierte Verfahren, welche algorithmisch versuchen, eine sinnvolle Anzahl von Wegpunkten in der Welt zu verteilen. Das Hauptproblem der letzteren Herangehensweise ist, zu definieren, was für die gegebene Map nun eine sinnvolle und vorteilhafte Platzierung von Wegpunkten darstellen würde – hier ein automatisches Verfahren zu finden erweist sich als schwierig. Sind die Wegpunkte verteilt, so ist das Ziehen der Kanten zwischen ihnen eine Aufgabe, welche ein automatisches Tool ohne jegliche Probleme bewältigen kann. Die Geometrie der Welt ist in einem BSP-Baum (*Binary Space Partitioning Tree* [Wat02, S. 227–230]) gespeichert, welcher einfach traversiert werden kann, um beispielsweise zu entscheiden, ob ein Punkt, eine Linie oder eine Kugel mit der Geometrie kollidiert, d.h. sich mit dieser schneidet bzw. diese durchdringt.

So ist es leicht feststellbar, ob eine Kreatur auf ihrer Wanderung von einem Wegpunkt zu einem anderen z.B. gegen ein statisches Hindernis wie einen Baum oder einen Felsen laufen würde. Täte sie dies, so ist zwischen beiden Knoten keine Kante zu ziehen. Liegt entlang der potentiellen Kante keine Kollision mit der Welt vor, so ist diese auch zu ziehen und die Kreaturen können sie zur Laufzeit der Cave-Anwendung zu Zwecken der Bewegung verwenden. Die Gewichtung der Kante ist abhängig von der Distanz der beiden Punkte zu wählen, im einfachsten Fall also anhand der Länge der direkten Luftlinie.

Eine Anforderung bei der Entwicklung des Navigationssystems für die Kreaturen war es, relativ schnell zu einem brauchbaren Ergebnis zu kommen. Es stand uns nicht ernsthaft die Zeit und Arbeitskraft zur Verfügung, die erforderlich gewesen wäre, um einen Editor mit grafischer Bedienungsoberfläche (GUI) für das Anlegen und Verbinden der Wegpunkte zu entwickeln. Nach obiger Überlegung ist es aber durchaus wünschenswert, zumindest für die Platzierung der Wegpunkte in die Map einen solchen Editor zur Verfügung zu haben. Eine Eigenschaft des Spiels Quake III Arena, dessen BSP-Dateiformat die Madness-Engine bekanntlich zur Speicherung der Weltgeometrie verwendet (siehe Abschnitt 3), können wir uns hierbei allerdings zu Nutze machen: Dieses Dateiformat sieht es vor, neben der eigentlichen Geometrie auch zusätzliche Elemente der Welt textuell in einem so genannten *Entity Chunk* zu beschreiben[3]. In ihm sind beispielsweise Licht- und Soundquellen, Startpositionen für die Spieler, Bonus-Pakete, Waffen, Munition und vieles mehr inklusive Position und etwaiger weiterer Eigenschaften definiert. Diese Objekte können mit den herkömmlichen Level-Editoren, welche frei für Quake III verfügbar sind[4] (siehe Abschnitt 10.1), in der virtuellen Welt positioniert und entsprechend eingestellt werden. Da viele dieser möglichen Gegenstände von unserer Cave-Anwendung beim Laden der Map vollständig ignoriert werden, bietet es sich an, sie als vorerst als Wegpunkte zu missbrauchen. Die breite Palette der verschiedenen Objekte macht es zudem möglich, die Wegpunkte gleich in unterschiedlichen Gruppen zu definieren.

Das Anlegen der Wegpunkte in der Landschaft wird durch diese Lösung abgedeckt, nicht jedoch das Aufspannen der Kanten des Wegpunktgraphen – die Level-Editoren erlauben eine derartige Operation nicht. Die Definition von Relationen zwischen den einzelnen Elementen des Entity Chunks, auf welche die Kanten abgebildet werden könnten, ist mit Quake-Bordmitteln so nicht zu bewerkstelligen. Wir haben uns daher für die naheliegende Kombination aus manuellen und automatischen Verfahren entschieden, um den Wegpunktgraphen zu erzeugen: Das Setzen der Knoten erfolgt, wie oben erläutert, innerhalb des grafischen Quake-Editors. Aus diesem heraus wird eine BSP-Datei erzeugt, in deren Entity Chunk die Wegpunkte in Form von Spiel-Bonuspaketen abgelegt sind. Anschließend

[3] Diese Beschreibung erfolgt freilich in einer festen Syntax und ist nicht natürlichsprachig.
[4] Wie beispielsweise der von uns verwendete GtkRadiant

wird mit einem von uns zu diesem Zweck entwickelten automatischen Werkzeug, dem `pathtool`, die BSP-Datei einmalig verarbeitet, um die Knoten des Wegpunktgraphen auszulesen und dort mit Kanten zu verbinden, wo zwischen ihnen ein begehbarer Weg existiert. Hierzu wertet das `pathtool` eine von seinem Benutzer eigens für die Map erstellte Konfigurationsdatei aus, das so genannte *Translation File*. In letzterem ist u.a. definiert, wie die gesetzten Bonuspakete als Wegpunkte zu interpretieren sind, wie das Ziehen der Kanten von Statten gehen soll und entlang welcher Oberflächenmaterialien (die so genannten *Shader*) keine Kanten verlaufen dürfen. In anschließenden Schritten berechnet das Tool die kürzesten Wege, d.h. für alle Knotenpaare n_1, n_2 des gerichteten und mit Kantenkosten versehenen Graphen die jeweils kostengünstigste Folge von Kanten, die von n_1 nach n_2 führt. Mit dem Ergebnis dieser Berechnung wird dann die eingangs beschriebene *Wegpunkt-Routingtabelle* aufgebaut, welche um Informationen über alternative Routen in einem weiteren Berechnungsschritt ergänzt werden kann. Alle Wegpunkte werden dann zusammen mit der Routingtabelle in eine Textdatei geschrieben, im Folgenden von uns als *Path File* oder *Pfaddatei* bezeichnet. Diese Datei wird von der Cave-Anwendung zur Laufzeit geladen und verwendet, so dass die im Entity Chunk der BSP-Datei gespeicherten Bonusobjekte nicht länger als Wegpunktinformation benötigt werden und folglich von der Cave-Anwendung weiterhin ignoriert werden können. Die Berechnungen für die Wegewahl, welche das `pathtool` so im Voraus durchführt, fallen somit nur ein einziges Mal an, was im Cave kostbare Rechenzeit spart. Das Verarbeitungsprinzip ist in Abbildung 7.4 dargestellt.

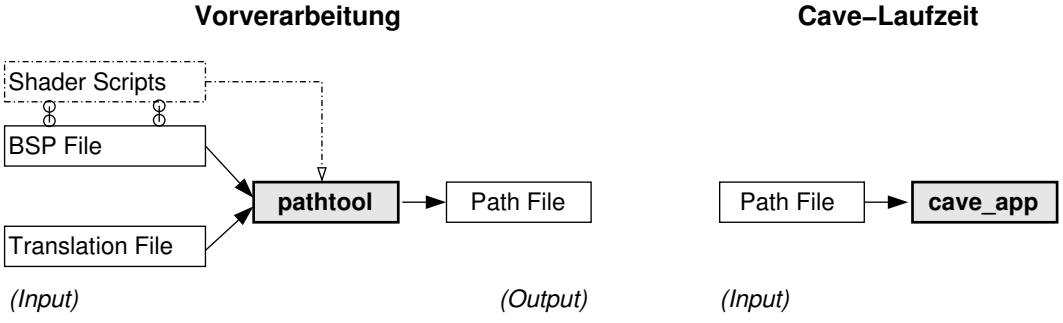

Abbildung 7.4: Wegvorverarbeitung mit dem `pathtool`

7.3.2 Eigenschaften der Wegpunkte

Jeder Wegpunkt verfügt über eine eindeutige Identifikationsnummer $id \geq 0$, eine Gruppennummer sowie eine Position im dreidimensionalen Raum. Die Punkte werden in der Regel leicht über dem Boden der Welt liegend verteilt, auf letzterem kommen die Kreaturen in jedem Fall zu stehen, ganz gleich wie weit sich der Wegpunkt oberhalb befindet[5]. Die y-Koordinate der Wegpunkte wird also „weich“ interpretiert. Wir werden im Folgenden stets sagen, eine Kreatur steht *auf* einem Wegpunkt, auch wenn sie sich streng genommen vielleicht eher unter diesem befindet.
Die Anforderung, dass die Kreaturen stets auf dem Boden der Map laufen sollen, hat Auswirkungen auf die Art und Weise, wie das Pfadwerkzeug auf Begehbarkeit einer potentiellen Kante zu testen hat. Der Test, ob die Strecke zwischen zwei Wegpunkten, d.h. die direkte Luftlinie (*Line of Sight)* zwischen ihnen, mit der Geometrie kollidiert, gestaltet sich relativ einfach und schnell. Er führt uns jedoch nicht weiter, da sich die Kreaturen in den wenigsten Fällen strikt entlang dieser bewegen werden. Wir testen stattdessen ein geometrisches Objekt (bei uns Punkt oder Kugel) auf Kollisionen mit der Landschaft, indem wir es in Schritten fester Größe entlang der zu prüfenden Kante verschieben und nach jedem Schritt an die Höhe des Bodens anpassen. Dieses Verfahren ist einfach und führt zu korrekten Ergebnissen, es hat jedoch den Nachteil, dass in *jedem* Schritt der BSP-Baum traversiert werden muss und somit die benötigte Rechenzeit um ein Vielfaches höher ist als bei einem einfachen Test gegen die Line of Sight.

In unserer Implementierung in C++ sind alle Wegpunkte Objekte einer allgemeinen Wegpunktklasse namens `micado_waypoint`. Jedes Wegpunktobjekt n speichert seinen eigenen Teil der geladenen Routingtabelle[6]. Dieser Teil ist nichts anderes als für alle anderen Wegpunkte m die jeweils erste Kante der Kantenfolge, welche von n nach m führt. Jeder Wegpunkt kann mit Hilfe einer Memberfunktion[7] danach gefragt werden, zu welchem Wegpunkt man sich als nächstes begeben muss, wenn man von ihm aus zu einem bestimmten anderen möchte, ohne mit der Geometrie der Welt zu kollidieren. Wegpunkte können zudem zur Laufzeit von den Kreaturen als belegt markiert werden, um zu verhindern, dass sich zu einem Zeitpunkt zu viele Kreaturen um sie herum aufhalten. Es ist anzumerken, dass in der verteilten Cave-Umgebung mit einem Master-Rechner und vielen Slaves, die Wegpunkte nur vom Master geladen werden, da sich lediglich auf ihm die Berechnung der Kreatur-KI abspielt (siehe geeignetes Kapitel des Projektberichts).

[5] Die Gesetze der Schwerkraft sollen schließlich auch in unserem virtuellen Dschungel gelten.
[6] Diese wird so auf die Wegpunkte verteilt im Speicher gehalten.
[7] `getRoute()`

Eine weitere wichtige Eigenschaft sind die so genannten *Capabilities*. Diese kann man sich als eine vom Benutzer definierbare Menge C von Flags vorstellen, welche global durch Zuhilfenahme einer Symboltabelle verwaltet wird. Sie stellen einen wichtigen Bestandteil der Kreatur-KI dar, da sie erlauben, Kreaturen und Wegpunkten bestimmte Eigenschaften zuzuweisen und auf deren Vorhandensein zu testen. Die Namen der Eigenschaften können vom Benuzter frei gewählt werden, ohne dass sie im Programmcode als Konstanten bekannt sein müssen. Es handelt sich bei ihnen um einfache Zeichenketten, welche zur schnellen internen Verarbeitung in eindeutige natürliche Zahlen übersetzt werden.
Ein Wegpunkt hält zwei Mengen $P \subset C$ und $R \subset C$ von zugewiesenen Capabilities, P steht hierbei für *provides*, R für *requires*. Beide Flag-Mengen geben pro Wegpunkt an, welche Eigenschaften er für Kreaturen in der Welt zur Verfügung stellt (wie beispielsweise Wasser, Futter oder einfach nur einen schönen Ausblick – der Kreativität seien hier keine Grenzen gesetzt) und welche er von ihnen erwartet, damit diese dort eine bestimmte Aufgabe oder Tätigkeit verrichten können. Im Translation File wird festgelegt, welche Capabilities eine Gruppe von Wegpunkten respektive Klasse von Bonuspaketen automatisch vom `pathtool` zugewiesen bekommt. Darüber hinaus kann auch festgelegt werden, dass Wegpunkte, welche oberhalb eines mit einem bestimmten Shader versehenen Polygons liegen, automatisch weitere Capabilities erhalten. Auf einer Grasfläche liegenden Punkten kann so z.B. eine Capability `"grass"` $\in P$ zugewiesen werden. Im Translation File wird außerdem noch ein weiterer wichtiger Parameter angegeben – der Parameter `penalty`. Er ist wichitg, da viele intelligente Lebewesen die Tendenz haben, einfache Wege mit wenigen Richtungsängerungen[8] geringfügig kürzeren Wegen mit vielen Richtungsänderungen vorzuziehen. Dies mag einerseits in einer gewissen Faulheit begründet liegen, andererseits in der Tatsache, dass sie Distanzen und Weglängen deutlich schlechter abschätzen können, als ein Computer dies kann. Im Zweifelsfall wird von ihnen der Weg um ein Labyrinth aus Hindernissen herum gewählt, statt eines Weges durch dieses, selbst wenn der Weg aussen ein gutes Stück länger ist. Um dieses Verhalten nachzuahmen, kann das `pathtool` bei der Berechnung der Kosten eines Weges durch den Graphen ab der zweiten eingeschlagenen Kante zusätzliche Strafpunkte auf das Gesamtergebnis aufschlagen. Dieser Parameter gibt diese Punkte pro zusätzlicher Kante an. Seien $dist(e_i)$ die Kosten einer Kante e_i, dann werden die Gesamtkosten einer Kantenfolge $p = e_1 \ldots e_n$ nach der folgenden Formel berechnet:

$$pathcost(p) = dist(e_1) + \sum_{i=2}^{n} (dist(e_i) + penalty)$$

[8] Wir nehmen an, dass an den meisten Knoten entlang eines Weges durch den Wegpunktgraphen eine Richtungsänderung nötig ist.

```
waypoint {
    position = 896.000000 32.000000 1744.000000
    id = 129
    group = 1
    requires = wok cook_chinese
    provides = grass food
}
waypoint {
    position = 200.000000 32.000000 2216.000000
    id = 128
    group = 1
    requires = wok cook_chinese
    provides = grass food
}
...
route {
    129 128 -> 128 118 119  @  000840.95184 001037.69971 001113.32922
    129 127 -> 127 119 117  @  001445.54492 001512.49768 001572.43872
    129 126 -> 116 119 117  @  003227.13818 003301.97314 003323.08813
    129 125 -> 116 119 117  @  004303.43945 004378.27441 004399.38965
    129 124 -> 116 119 117  @  007431.45801 007506.29297 007527.40771
    ...
}
```

Abbildung 7.5: Beispiel Path File *jungle.path*

7.3.3 Die resultierende Pfaddatei

Ein Beispiel für eine vom `pathtool` erzeugte Wegpunktdatei ist in stark gekürzter Fassung Abbildung 7.5 zu entnehmen. Eine solche Datei kann in der Praxis ohne weiteres eine Länge von über 8.000 Zeilen aufweisen, wobei ein Großteil von der Routingtabelle beansprucht wird. Die folgenden Blocktypen und Parameter sind möglich:

waypoint: Definiert einen Wegpunkt. Parameter:

- `position:` Gibt nacheinander die x, y und z-Koordinate des Wegpunktes in der Map an.
- `id:` Die eindeutige Identifikationsnummer des Wegpunkts in Form einer natürlichen Zahl.
- `group:` Die Gruppenzugehörigkeit des Wegpunkts in Form einer natürlichen Zahl.
- `provides:` Die von diesem Wegpunkt zur Verfügung gestellten Capabilities
- `requires:` Die von diesem Wegpunkt als Anforderung gestellten Capabilities

route: Gibt die Routinginformation an. Die einzelnen Einträge der Routingtabelle werden hierbei durch Zeilenumbrüche voneinander getrennt. Eine Zeile liest sich wie folgt:

origin_id target_id `->` *first_id second_id third_id* `@` *first_cost second_cost third_cost*

- *origin_id* und *target_id* sind die Identifikationsnummern des Start- und Zielwegpunkts dieser Route.
- *first_id* ist die Nummer des Wegpunkts, welcher als nächstes auf dem Weg vom Start- zum Zielwegpunkt besucht werden muss. Dieser ist die beste Wahl aus den zur Verfügung stehenden Möglichkeiten. Hat das Pfadwerkzeug ebenfalls alternative Routen für die nächste einzuschlagende Kante berechnet, so geben *second_id* und *third_id* die zweit- und drittbeste Wahl für den Folgeknoten an. Der Wert -1 (dieser ist keine gültige Wegpunktnummer) zeigt hierbei jeweils an, dass kein solcher Punkt existiert. Haben alle drei Felder den Wert -1, so wird die Zeile idR. ganz weggelassen.
- *first_cost*, *second_cost* und *third_cost* geben für die drei möglichen Wegfortsetzungen die Kosten des gesamten noch ausstehenden Weges vom Start- zum Zielwegpunkt an. In dieser Summe von Kantengewichtungen sind die Penalties der Folgekanten bereits enthalten. Der Wert -1 zeigt jeweils an, dass die Kosten unbekannt sind (z.B. falls keine entsprechende Kante existiert).

7.3.4 Ein Beispielszenario

Nachdem wir in den vorherigen Abschnitten einen kleinen Einblick in die Wegvorverarbeitung mit dem `pathtool` gegeben haben, möchten wir die von diesem produzierten Ergebnisse an einem einfachen Beispiel veranschaulichen. Abbildung 7.6 zeigt schematisch eine einfache Landkarte bestehend

aus vier hohen Mauern auf einem ebenen Boden, die im 90°-Winkel zueinander angeordnet sind. Die linke Wand besitzt in der Mitte eine Tür, durch welche man durch sie hindurchgehen kann. In der Map sind Wegpunkte auf zwei disjunkte Wegpunktmengen verteilt, die einen sind als schwarz gefüllte Quadrate dargestellt (Menge A), die anderen als kleine Kreise (Menge B). Die in den Wandumriss eingezeichneten Punkte liegen oben auf den Mauern. Wegpunkt Nummer 26 liegt bspw. oben auf dem Türbalken, nicht unten in der Türöffnung. Angenommen, die Welt liegt in der BSP-

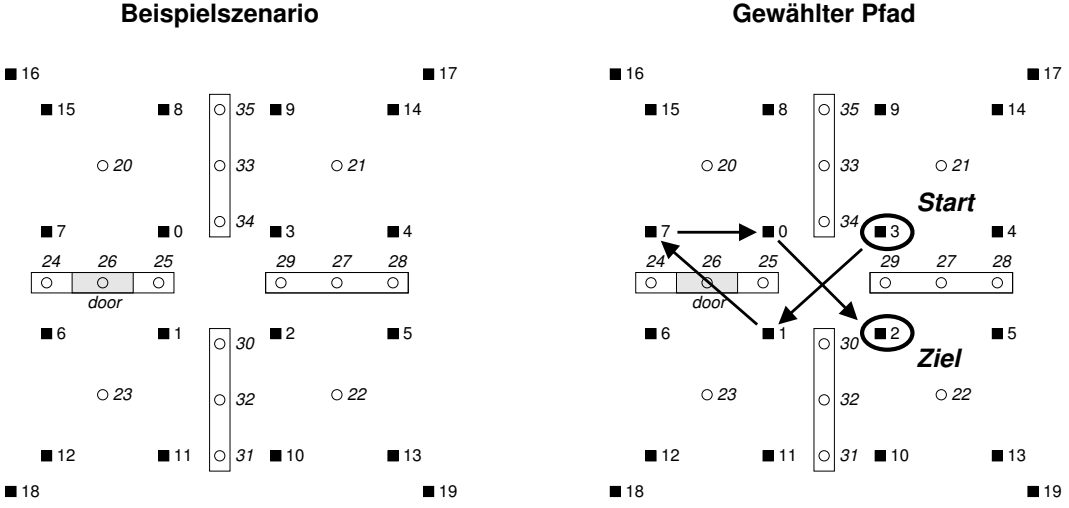

Abbildung 7.6: Einfaches Beispielszenario zur Wegewahl

Datei *demo.bsp* vor, *demo.trans* sei das für sie erstellte Translation File, und wir möchten das Ergebnis der Verarbeitung in die Datei *demo.path* geschrieben haben, dann startet die folgende Kommandozeile das Pfadwerkzeug zum Ziehen der Kanten und zur anschließenden Berechung der kürzesten Wege sowie alternativen Routen:

```
$> pathtool demo.bsp demo.trans demo.path
```

Ein repräsentatives Ergebnis der Verarbeitung[9]: Von 380 möglichen Kanten der zwanzig-elementigen Wegpunktgruppe *A* findet das Tool laut Bildschirmausgaben (von deren Abdruck wir an dieser Stelle aufgrund ihrer exzessiven Länge absehen) genau 136, von den 240 möglichen Kanten der sechzehn-elementigen Menge *B* findet es 30. Ein in die Jahre gekommener Testrechner mit einer AMD Athlon 600-CPU benötigte in unserem Beispiel-Run etwa 3 Sekunden für die erste Menge und knapp eine Sekunde für die zweite. Die Berechnung der kürzesten Wege war nach weiteren 370 ms beendet, das Finden der alternativen Routen dauerte nicht länger als 2 ms, so dass die gesamte Verarbeitung inklusive Laden der Dateien keine 8 Sekunden in Anspruch nahm. Es ist anzumerken, dass in der Praxis die benötigte Rechenzeit deutlich über den Werten dieses sehr einfachen Beispielszenarios liegen, da die BSP-Dateien in der Regel deutlich komplexer ausfallen und in ihnen zudem mehr Wegpunkte verteilt werden. Für die zweite Version unserer Cave-Dschungellandschaft mit circa 130 Wegpunkten benötigte der Testrechner z.B. über eine Stunde, für sehr komplexe Welten sind mehrere Tage denkbar.

Mit Hilfe eines zum `pathtool` gehörigen Toolprogramms namens `rtquery` ist es möglich die erzeugte Pfaddatei nach dem Pfad von einem Wegpunkt zu einem anderen abzufragen. Wir interessieren uns z.B. für den Weg von Wegpunkt 3 nach 2:

```
$> rtquery demo.path 3 2
```

Der beste Weg führt über die Wegpunkte 3, 1, 7 (durch die Tür hindurch), 0 und schließlich 2 (in dieser Reihenfolge). Vergleiche hierzu Abbildung 7.6. Das `rtquery`-Programm liefert in seinen Ausgaben zudem, auf Basis der vom `pathtool` berechneten alternativen Routen, noch zwei Pfade schlechterer Wahl:

```
| rtquery | query route from path file v1.1 |

==> reading path file "demo.path"
==> routes from [4] 3 to [4] 2:
  best:   3 -> 1 -> 7 -> 0 -> 2  (len:4 tab:1435.000000 acc:1236.000000)
  second: 3 -> 12 -> 6 -> 0 -> 2  (len:4 tab:1707.000000 acc:1507.000000)
  third:  3 -> 4 -> 6 -> 0 -> 2  (len:4 tab:1763.000000 acc:1563.000000)
==> legend:
```

[9]Das Ergebnis kann selbstverständlich in der Praxis, abhängig von den Einstellungen im Translation File, von den hier gemachten Angaben abweichen.

```
    len: the number of followed transitions
    tab: path cost from source route table (including penalties)
    acc: accumulated path cost (without penalties)
```

Aufgrund der Tatsache, dass im Zentrum der vier Mauern kein Wegpunkt liegt, welcher das Problem vereinfachen würde, verlaufen alle drei Pfade über vier Kanten und durch die Tür in der linken Wand. Für einen menschlichen Benutzer mag dies nicht immer die naheliegendste Lösung darstellen, so führt z.B. ebenfalls die Wegpunktfolge 3, 17, 19, 2 vom Start zum Ziel und besucht hierbei lediglich drei Kanten. Diese Lösung weist zudem weniger Richtungsänderungen auf. Durch Erhöhen des Penalty-Wertes im Translation File (beispielsweise von 60 auf 600) kann auch sie in der Welt als „beste Wahl“ zur Verwendung kommen:

```
| rtquery | query route from path file v1.1 |

==> reading path file "demo.path"
==> routes from [4] 3 to [4] 2:
  best:   3 -> 17 -> 19 -> 2  (len:3 tab:2846.000000 acc:1646.000000)
  second: 3 -> 1 -> 7 -> 0 -> 2  (len:4 tab:3036.000000 acc:1236.000000)
  third:  3 -> 18 -> 19 -> 2  (len:3 tab:3208.000000 acc:2008.000000)
==> legend:
  len: the number of followed transitions
  tab: path cost from source route table (including penalties)
  acc: accumulated path cost (without penalties)
```

— KAPITEL 8 —

Kreatur-KI

Der Grundgedanke bei der Erschaffung unserer Dschungel-Anwendung für den Cave war, eine in sich schlüssige Welt zu schaffen, die sich vor allem durch die Verhaltensweisen der hierin lebenden Kreaturen auszeichnen sollte. Hauptansatzpunkt war es deshalb, die Kreaturen weitgehend selbstständig agieren zu lassen, ihnen also in gewissem Maße eine Intelligenz zu verpassen. Sie folgen dabei persönlichen Bedürfnissen, reagieren untereinander und auf den Besucher. Dieser nimmt beim Eintauchen in die Welt zunächst automatisch eine passive Beobachterrolle ein, kann allerdings durch von ihm ausgehende Interaktion (siehe Kapitel 6) mit den Kreaturen aktiv werden.
Das konzeptuelle Grundgerüst des eigenständigen Handelns beruht auf einer Auswahl von so genannten Motivationsleveln, deren Zusammenspiel die Bedürfnisse der Kreatur bestimmt und durch das Auslösen entsprechender Aktionen ihr Verhalten vorgibt. Die Motivationslevel einer Kreatur ändern sich in der Regel zeitabhängig und relativ konstant, es kann aber auch zu Motivationssprüngen durch das Eintreten spezieller Events[1] kommen. Alle Events, sofern sie nicht durch Interaktion vom Besucher ausgelöst werden, hängen unmittelbar mit dem Wahrnehmungsradius[2] jeder Kreatur zusammen.

[1] dt.: Ereignis. Ein Event ist z.B. das Aufeinandertreffen zweier Kreaturen, woraufhin die Motivation zu Grüßen temporär stark ansteigt. Andere Events können auch vom Benutzer innerhalb des Caves ausgelöst werden.

[2] Der regelbare Wahrnehmungsradius jeder Kreatur bestimmt, ab welcher Entfernung sie auf andere Kreaturen oder den Besucher reagiert.

8.1 Motivationslevel und Events

Wie bereits erwähnt, beruht die künstliche Intelligenz der Kreaturen auf Motivationsleveln. Wird eine Kreatur etwa durch eine Klasse `micado_testcreature` beschrieben, erbt sie alle Motivationslevel, die in der Vaterklasse `micado_creature` definiert sind. Konkret sind dieses Zahlenwerte, die die Dringlichkeit der folgenden Bedürfnisse angeben:

- Schlafen
- Trinken
- Essen
- Folgen
- Fliehen
- Grüßen
- Nichts tun

Jedes Motivationslevel ist dabei ein `struct` in Form eines Tripels von `doubles` mit den Bezeichnungen (`val`, `mul`, `mte`). `val` steht dabei für Value und gibt konsequenter Weise den aktuellen Wert der Motivation an; `mul` steht für Multiplikator und repräsentiert den kreaturcharakterspezifischen zeitlichen Einfluss auf die Motivation; `mte` letztlich steht für Mutator, ein Wert, der ebenfalls auf die Motivation einwirkt, sich aber zur Laufzeit ändert und somit ein linerar konstantes Verhalten der Kreatur verhindert. Der Mutatorwert ändert sich immer dann, wenn eine Aktion durch eine andere abgelöst wird.
Ebenfalls von der Vaterklasse übernommen und in `micado_testcreature` erweitert ist die Funktion `do_ai()`, die dafür sorgt, dass die Kreatur ihre Motivationslevel verändert. Es wird dabei über alle zeitlich veränderlichen Motivationen iteriert und der Wert ensprechend dem Multiplikator und des Mutators verändert. Die Werte erhöhen sich nur bis zu einer Maximalgrenze. Erreicht ein Wert diese Grenze, wird dieser und alle ebenfalls zeitabhängigen Werte mit einem verhältniswahrenden Offset belegt. Die Maximalgrenze sorgt dafür, dass bestimmte Aktionen, deren Ausführung elementar ist und deren Motivation bei Bedarf mit einem Wert oberhalb dieser Grenze belegt wird, von den übrigen nicht gestört werden.

Folgend auf diese Erhöhung wird die nun stärkste Motivation herausgesucht. Stimmt sie mit der aktuellen Aktion überein, wird diese beibehalten und fortgeführt, andernfalls wird die aktuelle Aktion beendet und eine neue entsprechend der stärksten Motivation gestartet. Dieses Verfahren wird im Abschnitt 8.2 noch einmal genauer aufgegriffen werden. Hier soll zunächst genauer die Struktur des Systems Motivationslevel und Kreaturen und sein Zusammenhang mit den Events erläutert werden. Es wurde von Kreaturen gesprochen, die durch eine eigene Klasse beschrieben sind und grundlegende Eigenschaften von einer Klasse `micado_creature` erben. Diese Klasse hält weiterhin eine Liste aller von ihr abgeleiteten und erschaffenen Kreaturen, den Vector `CREATURES`. Der Kreaturenmangager `CM()` (Creature Manager), eine weitere grundlegende Funktion, sorgt bei Ausführung der Anwendung dafür, dass alle Kreaturen in einer bestimmten Zeit ihr `do_ai()` ausführen. Zusätzlich hat er die Aufgabe, die Position von jeweils zwei Kreaturen zueinander zu überprüfen und dabei zu ermittlen, ob sich eine Kreatur in dem schon erwähnten Wahrnehmungsradius der aktuell bearbeiteten Kreatur befindet, wofür er den Zugriff auf den Vector aller Kreaturen benötigt. Ergibt gerade beschriebene Abfrage ein `true`, wird ein Pointer auf diese Kreatur in ihre Wahrnehmungsliste `percpt_list` eingehängt, womit schon die grundlegende Funktion des einfachsten Events `event_enter` beschrieben wurde. Die Funktion des Events `event_leave` ergibt sich analog.

- `event_enter`: Dieses Event wird vom Kreaturmanager an eine Kreatur geschickt, wenn eine weitere Kreatur oder der Benutzer (welcher per Definition auch eine Kreatur ist) in deren Wahrnehmungsbereich trifft. Durch dieses Event wird zunächst die Motivation zu Grüßen stark erhöht, was in den meisten Fällen zu einem direktem Ausführen der entsprechenden Aktion führt. Weiterhin besteht für die Kreatur nur solange die Möglichkeit, aus eigenem Antrieb die Motivation zum Folgen des Benutzers zu erhöhen, solange er in der Wahrnehmungsliste aufzufinden ist.

- `event_leave`: Hier wird lediglich ein Eintrag aus der Wahrnehmungsliste entfernt, auf die Motivationslevel der Kreatur wird direkt kein Einfluss genommen. Indirekt verhindert das Herausnehmen aus der Wahrnehmungsliste natürlich das Steigen der Motivation zum Grüßen dieser Kreatur und eventuell des Folgens des Benutzers.

Die weiteren vier Events `event_follow`, `event_dismiss`, `event_wakeup` und `event_scare` kann allesamt der Benutzer initiieren. Sie erhöhen dabei die betreffenden Motivationen über die Maximalgrenze hinaus, um den Benutzer durch deren sofortige Auswirkung ein entsprechendes Feedback zu geben. Es wird an dieser Stelle nicht weiter auf sie eingegangen, da ihre Funktion schon in Kapitel 6 erläutert wurde.

8.2 Aktionen

Eine Aktion repräsentiert die Ausführung genau einer Tätigkeit. In den Kreaturklassen selbst sind keinerlei Steuerungs-Algorithmen implementiert, die die Motivationswerte wieder senken. Statt dessen wird auf Grund der aktuellen Motivationswerte entschieden, welche Aktion zu starten ist. Die Kreatur gibt dann im folgenden ihre „Rechenzeit“ an die von ihr gestartete Aktion weiter, was sie solange tut, bis diese beendet ist bzw. wird und die Kreatur zur Auswahl einer neuen Aktion wieder aktiv werden muss. Eine Kreatur führt also im Normalzustand immer genau eine Aktion aus. Was sie dabei nicht mitbekommt, ist, dass die Aktionen ihrerseits Unteraktionen starten können. Eine typische Unteraktion ist dabei das Gehen, welche beispielsweise immer dann notwendig wird, wenn die Kreatur aufgrund von Hunger oder Durst einen geeigneten Ort in der Welt aufsuchen möchte. Ist das Ziel erreicht, wird die Unteraktion Gehen beendet, woraufhin das Bedürfnis befriedigt werden kann und schlussendlich auch diese Aktion terminiert. Wie eingangs beschrieben, haben die „Interaktions-Aktionen“ Folgen und Fliehen Vorrang, um den Benutzer ein sofortiges Feedback zu geben, d.h. die Motivationswerte für diese Aktionen steigen bei Eintritt des entsprechenden Events höher als es die „Bedürfnis-Aktionen“ können und unterbrechen diese damit augenblicklich.
Um ein „Flimmern“ zwischen zwei Zuständen zu vermeiden (dies würde auftreten, wenn zwei Aktionen sich ständig innerhalb von wenigen Augenblicken gegenseitig ablösen würden.), erhöhen wir am Anfang einer Aktion die Motivation für diese leicht. Beim Beenden einer Aktion wird die Motivation um den gleichen Wert wieder verringert, um das Kreaturverhalten durch dieses Hilfsmittel nicht nachhaltig zu beeinflussen.
Aktionen werden von einer Vaterklasse `micado_action` abgeleitet. Gemäß ihrer Aufgabe beinhalten sie dann den Programmcode, der in Ausführung die entsprechende Tätigkeit ermöglicht. Ihre genauere Beschreibung erfolgt durch untenstehende Punkte.

Die ersten vier im Folgenden beschriebenen Aktionen lösen beim Aktivieren die momentan aktive Aktion erst dann ab, wenn die Kreatur sich auf einem Wegpunkt befindet. Zweck ist dabei, dass die Kreatur sich problemlos einen Pfad zu einem Wegpunkt suchen kann, an dem das entsprechende Bedürfnis zu befriedigen ist. Das ist nur bei einem gegebenen Quell-Wegpunkt möglich.

- Durst: Bekommt eine Kreatur Durst, so startet sie die Aktion `micado_std_thirst`. Die Aktion überprüft den Standortwegpunkt der Kreatur auf Tauglichkeit bezüglich der Bedürfnisbefriedigung. Ist es ein geeigneter Wegpunkt, so wird als Unteraktion sogleich die Aktion `micado_std_drink` gestartet, die die konkrete Aufgabe übernimmt. Wie bereits erwähnt, beendet sich nach dem Trinken zunächst die Unteraktion, daraufhin die Aktion und die Kreatur

trifft eine nächste Tätigkeitsentscheidung. Ist der Standortwegpunkt allerdings keine Trinkstelle, so muss natürlich zunächst eine erreicht werden. Dabei sucht sich die Kreatur aus der Menge der Wegpunkte einen heraus, der Wasser bietet. Der genaue Sachverhalt ist auch hier aus dem Kapitel 7 Pathfinding zu entnehmen. Dann wird die Unteraktion `micado_std_walk` mit dem zuvor ermittelten Zielwegpunkt eingesetzt. Beim Erreichen des Zieles beendet sich `micado_std_walk` und obiger Fall tritt ein.

- Hunger: Funktioniert analog zu Durst, nur ist hier die bedürfnisbefriedigende Unteraktion sinnigerweise `micado_std_eat`.

- Schlafen : Da jeder Wegpunkt ein möglicher Schlafplatz ist, ist die oben getätigte Unterscheidung zweier Fälle hinfällig. Es wird lediglich sicher gestellt, dass die Kreatur die Aktion `micado_std_sleep` auf einem Wegpunkt ausführt. Aus diesem Grund braucht auch keine Unteraktion zur Bedürfnisbefriedigung eingesetzt zu werden, dies übernimmt die Aktion selbst.

- Untätigkeit: Die Aktion `micado_std_idle` funktioniert analog zu Schlafen.

- Gehen: Es wurde schon deutlich, dass `micado_std_walk` eine reine Unteraktion ist. Sie wird immer dann eingesetzt, wenn eine Kreatur zur Bedürfnisbefriedigung einen entfernten Wegpunkt erreichen muss und terminiert, sobald dieser erreicht ist. Wie unten beim Grüßen beschrieben wird, kann wiederum eine Ebene tiefer dann als Unterunteraktion Grüßen eingesetzt werden.

- Folgen: Eine Kreatur kann auf zwei unterschiedliche Arten dazu gebracht werden, die Aktion `micado_std_follow` auszuführen. Einerseits als Reaktion einer Initiative des Benutzers, wie es bereits in Kapitel 6 beschrieben wurde. Hierbei endet die Aktion erst bei entsprechender Initiative des Benutzers. Andererseits motivationsgesteuert durch die Kreatur selbst. Damit dies nicht zu häufig geschieht, der Benutzer dann auch etwas davon mitbekommt und er weiterhin nicht von ganzen Horden von Kreaturen belagert wird, wurde die Ausführung dieser Art des Folgens mit folgenden Restriktionen belegt. Die Motivation für Folgen steigt kreaturseitig nur, wenn der Benutzer sich in deren Wahrnehmungsradius befindet. So wird Häufigkeit reduziert und Benutzeraufmerksamkeit erlangt. Weiterhin kann immer nur eine Kreatur gleichzeitig dem Besucher motivationsgesteuert folgen, damit es nicht zu einem chaotischen Getümmel kommt. Die Motivation der Kreatur sinkt mit der Dauer des eigenangetriebenen Folgens bis zu dessen Beendung.
  Für beide Fälle ein sehr wichtiger Punkt ist, dass nach dem Terminieren dieser Aktion unbedingt (durch Belegen einer Motivation über der Maximalgrenze) die Aktion

micado_std_findback (siehe Abschnitt 8.4) startet, um die Kreatur ins Wegpunktsystem zurückzuführen.

- Fliehen: micado_std_flee wurde ebenfalls schon im Kapitel 6 beschrieben. Da das Fliehen auf dem Wegpunktsystem stattfindet, ist keine anschließende micado_std_findback-Ausführung von Nöten.

- Grüßen: Die Motivation, jemand anderes zu grüßen, steigt nicht zeitlich, sondern hängt davon ab, ob jemand in den Wahrnehmungsradius der Kreatur gelangt. Ist dies der Fall, steigt die Motivation, den Gruß auszuführen, sofort in einer solchen Weise, dass jeder andere Motivationswert überflügelt und der Gruß somit auf jeden Fall ausgeführt wird.
  Bemerkenswert ist in diesem Fall, dass micado_std_greet sowohl als Hauptaktion, als auch als Unteraktion von micado_std_walk gestartet werden kann. Wenn zum Beispiel eine Kreatur an einer nahrungsaufnehmenden anderen vorüber kommt, grüßen sich beide. Letztere startet den Gruß als Hauptaktion, wonach über do_ai() geregelt wird, was als nächstes gemacht wird, höchstwahrscheinlich wird die Nahrungsaufnahme danach fortgesetzt.
  Erstere soll natürlich durch den Gruß nicht vollkommen aus dem Konzept gebracht werden. Würde nämlich die eigentlich anstehende Aufgabe (z.B. Trinken) beendet und über do_ai() neu gestartet werden (da der Durst vermutlich immer noch den größten Motivationswert hat), würde auch ein neuer Ziel-Wegepunkt ausgewählt werden. Bei entsprechender Bevölkerung in der Dschungelwelt könnte dies dazu führen, dass solche Aktionen niemals zum Ziel führen. Deshalb wird der Gruß als Unteraktion vom Laufen ausgeführt. Nach dem Gruß wird dann die vorherige Aktion fortgesetzt.

Weitere Aktionen, die Kreaturen ausführen können, sind micado_std_avoid, micado_std_circulate, micado_std_findback, micado_std_respawn sowie micado_std_wait, deren Beschreibung sich in Abschnitt 8.4 befindet.

8.3 Kreaturspezifische Konfigurationsdatei

Wie deutlich wurde, bevölkert den Dschungel eine Vielzahl von sehr unterschiedlichen Kreaturen. Damit sich dieses nicht nur in ihrem Aussehen widerspiegelt, sondern darüber hinaus körperliche Eigenschaften ihrem Aussehen entsprechend gestaltet werden können und eine gewisse individuelle Charakterformung stattfinden kann, wurde für jede Kreatur eine eigene Konfigurationsdatei

angelegt. Diese wird beim Anwendungsstart bei der Kreaturerzeugung ausgelesen und ihr jeweils zugewiesen.

In Bezug auf die körperlichen Eigenschaften werden die konstanten Attribute `ATTRIB_SIZE`, `ATTRIB_STRENGTH`, `ATTRIB_AGILITY`, `ATTRIB_SPEED_WALK`, `ATTRIB_SPEED_RUN` und `ATTRIB_RADIUS` mit Werten belegt. `ATTRIB_SIZE` und `ATTRIB_RADIUS` (Größe und Umfang) sind Werte, die elementar für die Ausweichroutinen der Kreatur sind (siehe Abschnitt 8.4). `ATTRIB_STRENGTH` und `ATTRIB_AGILITY` (Stärke und Agilität) werden in der derzeitigen Anwendung nicht verwendet. Es war angedacht, die Stärke als einen Faktor zu nehmen, der ein Aufeinandertreffen zweier Kreaturen bestimmt; die Agilität sollte Einfluss auf die Manövrierfähigkeit der Kreaturen beim Wenden und Ausweichen nehmen. `ATTRIB_SPEED_WALK` und `ATTRIB_SPEED_RUN` geben normale Gehgeschwindigkeit sowie die Laufgeschwindigkeit beim Fliehen an.

Ein weiteres konstantes Attribut ist das `ATTRIB_OPENESS` (Offenheit), welches grundsätzlich das Verhalten der Kreatur bei der Interaktion (siehe Kapitel 6) mit dem Benutzer angibt, also eine charakterliche Eigenschaft. Der Sinn besteht hierbei darin, dass sich die Kreaturen in zwei Klassen einordnen lassen (gültige Werte des `ATTRIB_OPENESS` sind lediglich 0.0 und 1.0). Die eine Seite ist grundsätzlich zutraulich, ihre Vertreter nähern sich beim Anlocken des Benutzers und Fliehen, wenn er sie verscheucht. Sie reagieren somit erwartungsgemäß. Die andere Seite hält es genau andersherum, ihre eher scheue Art lässt sie bei Annäherungsversuchen des Benutzers (Anlocken) fliehen, auf das verscheuchen reagieren sie verärgert und laufen auf den Benutzer zu. Es sollte so für die Interaktion darin ein weiterer Anreiz geschaffen werden, als dass die Kreaturen erst durch Lernen des zunächst unwissenden Benutzers diesen Klassen zugeordnet werden können. Praxistests haben allerdings ergeben, dass sich der reale Benutzer durch dieses ihm indeterministisch scheinende Verhalten der Kreaturen verwirrt zeigte, woraufhin bei folgenden Präsentationen alle Kreaturen mit dem Wert der erwartungsgemäßen Offenheit belegt wurden.

Alle weiteren charakterlichen Eigenschaften der Kreaturen werden direkt in die schon vorgestellte Struktur (`val`, `mul`, `mte`) eines Motivationslevels eingepflegt: Den konkreten Werten `val` kann man Initialwerte zuweisen und somit beeinflussen, mit welcher Aktion eine Kreatur in die Welt startet und welche sie wahrscheinlich als nächstes ausführen wird. Mit zunehmender Laufzeit (Bedürfnisbefriedigung) verliert diese Startbelegung allerdings ihren Einfluss. Der eigentliche Charakter einer Kreatur wird durch die Multiplikatoren `mul` generiert. Diese Werte regeln dann die Relationen zwischen einzelnen Motivationswerten. Wenn zum Beispiel die Multiplikatorvariable `MOTIVATION_HUNGER` den Wert 0,6 und die entsprechende Variable zu `MOTIVATION_THIRST` den Wert 1,2 hat, steigt die Motivation für Durst doppelt so schnell wie jene für Hunger. Folglich wird diese Kreatur öfter eine Wasserstelle aufsuchen als einen Ort, an dem sie Nahrung aufnehmen kann. So können grundle-

gend hungrige, durstige, schläfrige, faule, zutrauliche Kreaturen erschaffen werden. Kombinationen verschiedener Eigenschaften sind natürlich auch möglich.

8.4 Kreaturbewegung

Mit dem `pathtool` haben wir uns die Möglichkeit verschafft, dass sich die Kreaturen in der Welt bewegen können (siehe Kapitel 7). Diese Vorarbeit nutzen wir nun und lassen die Kreaturen von Wegpunkt zu Wegpunkt gesteuert von der Aktion `micado_std_walk` laufen. In der ersten Version der Implementierung dieser Aktion konnten die Kreaturen über die von der Path-Datei vorgegebenen Wegpunkte navigieren. Kreaturen, deren Wege sich kreuzten, sind einfach durcheinander durch gelaufen, was zugegeben sehr seltsam aussah. Um diesem Problem aus dem Weg zu gehen, gab es eigentlich nur eine Möglichkeit: Die Kreaturen müssen sich gegenseitig ausweichen können. Kreaturen können sich allerdings nur ausweichen, wenn sie den vorgegebenen Weg zwischen zwei Wegpunkten verlassen. Wie schon im Abschnitt 7.2.2 erwähnt, besitzt ein Wegpunktsystem aber keine Information über den Zustand der Welt neben dem Weg. Ein weiteres Problem entsteht beim Auslösen der Aktion `micado_std_follow`. Durch diese Aktion verlässt eine Kreatur ihren Pfad, um den Benutzer auf direktem Weg anzulaufen und ihm dann zu folgen. Wird die Aktion beendet, so muss die Kreatur irgendwie wieder auf einen Weg des Wegpunktesystem kommen. Ein neue Aktion, wie z.B. `micado_std_eat` kann nur gestartet werden, wenn die Kreatur sich auf einem Weg oder genau auf einem Wegpunkt befindet.

8.4.1 Anderen Kreaturen ausweichen

Wie handeln wir Menschen eigentlich, wenn wir einer anderen Person ausweichen? Diese Frage stand zunächst bei der Entwicklung eines Konzeptes im Mittelpunkt. Wir stellten fest, dass dies ein sehr komplexer dynamischer Prozess ist, in dem meistens beide betroffenen Personen aufeinander reagieren. Bewegen sich zwei Personen aufeinander zu und würden zusammenstoßen, so reagieren beide und schlagen eine leicht veränderte Richtung ein. Schritt für Schritt scheint nun der zukünftige Weg des anderen abgeschätzt zu werden, wobei es schon bei der ersten Reaktion beider zu einer falschen Entscheidung kommen kann, so das sie sich wieder aufeinander zu bewegen. Es gibt aller-

dings auch den Fall der einseitgien Reaktion, wenn z.B. eine Person eine andere überholen möchte. In diesem Fall kann die zu überholende Person meist nicht reagieren, da sie die andere nicht wahrnimmmt. Ein weiterer Faktor ist die Umgebung der Personen, denn eine Person kann nur dann ausweichen, wenn es die Umgebung zulässt und nicht z.B. eine Mauer oder sogar eine andere Person im Weg steht.

Mit der Benutzung eines Wegpunktesystems hatten wir aber eine Einschränkung in Kauf genommen, die uns gerade hier sehr viele Probleme bereitete, denn der Prozess des Ausweichens setzt ein genaues Wissen über die Umgebung voraus, und dieses ist hier auf den Weg zwischen zwei Wegpunkten beschränkt. Wir mussten also bei der Entwicklung eines Konzeptes einige Anpassungen an die Realität vornehmen: Der wohl größte dabei war die Reduzierung der Dynamik des Vorgangs, d.h. es sollte immer, auch bei einem Überholvorgang, nur eine Kreatur aktiv ausweichen und somit den Weg des Wegpunktesystems verlassen. Die andere Kreatur hingegen sollte während des Ausweichvorgangs lediglich stehen bleiben und beispielsweise die ausweichende Kreatur grüßen. Wir waren so in der Lage den neuen Weg im Vornherein zu berechnen.

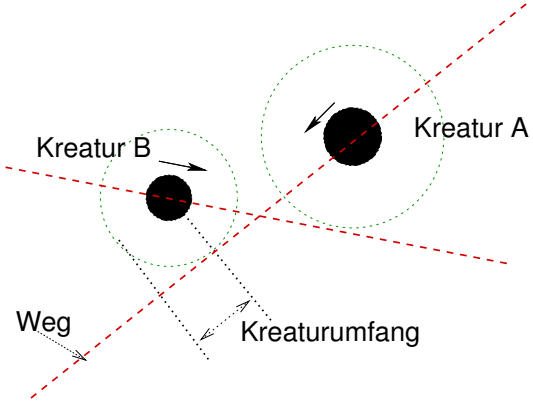

Abbildung 8.1: Zwei Kreaturen, deren Wege sich kreuzen

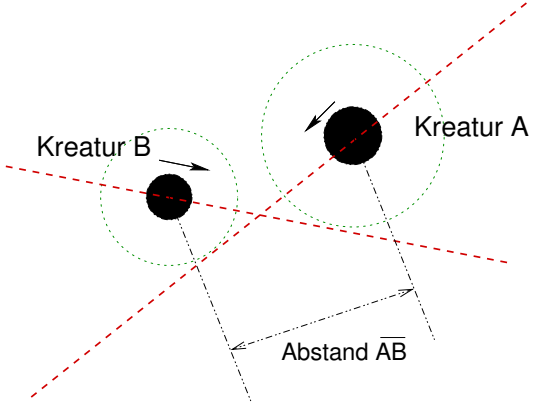

Abbildung 8.2: Eine Kollision zweier Kreaturen wird über deren Abstand zueinander bestimmt

Kollisionserkennung

Auf dieser Grundlage überlegten wir uns zunächst, wie zwei Kreaturen eine herannahende Kollision überhaupt erkennen können. Ein Beispiel für eine nahende Kollision ist in Abbildung 8.1 zu sehen. Die Kreaturen A und B sind durch einen dicken schwarzen Punkt gekennzeichnet und bewegen sich jeweils auf einem Weg (rot gestrichelte Linie). Wie man erkennen kann, kreuzen sich ihre Wege. Die herannahende Kollision wird nun durch eine einfache Berechnung des Abstandes der beiden erkannt (siehe Abbildung 8.2). Der Abstand muss größer als die Summe aus dem Umfang der Kreatur A und B (in der Abbildung grün gestrichelt gekennzeichnet) und einem Toleranzfaktor, damit eine Kollision „frühzeitig“ erkannt wird, sein. Ist der Abstand kleiner als die Summe, so wird noch überprüft, ob die Kreaturen sich relativ aufeinander zu bewegen. Ein Beispiel hierfür ist in der Abbildung 8.3 dargestellt. Im ersten Berechnungsschritt würde hier eine Kollision erkannt werden, doch die Kreatur B entfernt sich von dem Weg der Kreatur A, so dass sie letztendlich nicht kollidieren werden. Berechnet wird dies durch eine Schnittpunktberechnung der beiden Linien, die sich jeweils von der Position einer Kreatur zu ihren Zielwegpunkt aufspannt (in Abbildung 8.3 spannt sich z.B. eine Linie von der Position der Kreatur A zum Wegpunkt b). Später beim Testen unser Implementierung stellten wir noch fest, dass es trotzdem zu einer Kollision kommen konnte, auch wenn sie sich nicht relativ aufeinander zu bewegen. Grund hierfür ist ein möglicher großer Geschwindigkeitsunterschied in der Fortbewegung der beiden Kreaturen. In der Abbildung 8.3 würde es z.B. zu einer Kollision kommen, wenn die Kreatur B sich nicht schnell genug fortbewegt. Es darf also letztendlich ein Minimalabstand[3] nicht unterschritten werden, sonst muss ein Ausweichvorgang eingeleitet werden. Diese Überprüfungen werden mit jedem Frame in der Aktion `micado_std_walk` durchgeführt.

Berechnung des Ausweichweges

Ist erstmal eine Kollision erkannt, stellt sich nun die Frage: Wie soll der neue Weg (Ausweichweg) einer Kreatur aussehen, um einer anderen, die dann stehen bleibt und wartet, auszuweichen? In unserem ersten Konzept beschlossen wir, dass die ausweichende Kreatur sich auf einer Kreisbahn oder auf einer sinusförmigen Kurve um die stehende Kreatur bewegen soll. In den Abbildungen 8.4 und 8.5 wird die Berechnung eines kreisförmigen Ausweichweges als Beispiel dargestellt. Zunächst wird der Punkt auf dem Weg der Kreatur A gesucht, der am nahesten zur Kreatur B liegt. Dieser Punkt bildet nun den Mittelpunkt des Halbkreises auf dem die Kreatur später ausweichen soll.

[3] Der Minimalabstand ist die Summe aus den beiden Umfängen der Kreaturen und einem sehr kleinen Toleranzfaktor

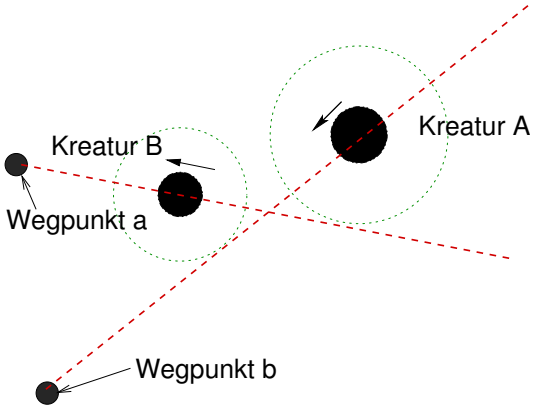

Abbildung 8.3: Die beiden Kreaturen kollidieren nicht, falls die Kreatur *B* sich schnell genug fortbewegt

In unserer ersten Implementierung haben wir das Konzept eines sinusfömigen Ausweichwegs umgesetzt, denn wir versprachen uns davon ein relativ realistisches Aussehen des Ausweichweges. Doch schon bei den ersten Tests stellten wir fest, dass wir einen solchen Weg nicht variabel anpassen können, denn einen gravierenden Aspekt hatten wir nicht bedacht: Der ursprüngliche Weg der ausweichenden Kreatur muss gerade verlaufen bis diese wieder vom Ausweichweg auf ihn zurückkommt, sonst muss der Ausweichweg entsprechend angepasst werden. Also mussten wir eine einfachere Lösung zur Berechnung eines Ausweichweges finden, den wir leicht an den Verlauf des ursprünglichen Wegs anpassen könnten. Die Lösung bestand darin, einen zuvor berechneten Ausweichpunkt von der ausweichenden Kreatur direkt auf einem geraden Weg anlaufen zu lassen. Von dort aus kann die Kreatur wieder direkten Weges einen zu berechnenden Wegzielpunkt anlaufen. Ein Beispeil ist in den Abbildungen 8.6 und 8.7 zu sehen. Wie schon zu Beginn erwähnt, besitzen wir keine Information über die Umgebung neben den Wegen des Wegpunktesystems, also müssen wir zur Laufzeit der Cave-Anwendung diese Informationen beschaffen. Sehr genaue Informationen können wir erhalten, indem wir den Ausweichweg genau wie beim `pathtool` mit Hilfe einer Kugel, die die Ausmaße der ausweichenden Kreatur besitzt, abtasten. Allerdings ist dies eine sehr rechenaufwendige Berechnung, die zur Laufzeit vorzugweise nicht eingesetzt werden sollte. Stattdessen gehen wir davon aus, dass es ausreicht lediglich die Sichtbarkeit des Ausweichpunktes vom Start-

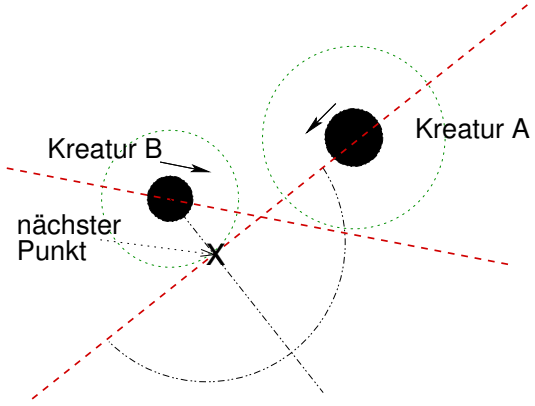

Abbildung 8.4: Berechnung des Ausweichweges der Kreatur *A* über den nahesten Punkt auf seinem ursprünglichen Weg zur Kreatur B

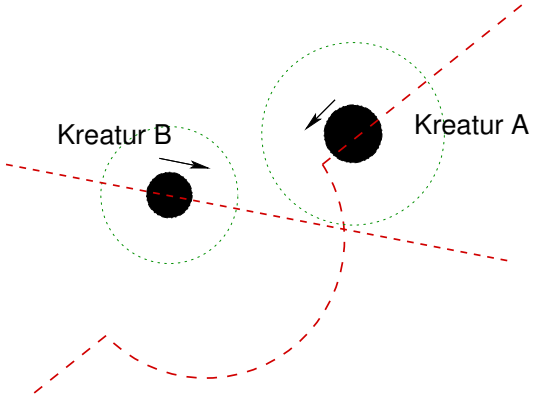

Abbildung 8.5: Der neue Weg der Kreatur *A*

und Zielpunkt des Ausweichweges zu prüfen. Ein Baum oder Ähnliches kann so mit dieser Sichbarkeitslinie (*Line of Sight*) entdeckt werden. Die Sichbarkeitslinie muss natürlich in entsprechender Höhe zum Boden der virtuellen Welt gezogen werden. Diese sehr einfache Technik beansprucht nur wenig Rechenleistung der CPUs und führte in unserer Realisierung zu sehr guten Ergebnissen.

Kommen wir nun zu einigen Sonderfällen, die bei der Berechnung eines Ausweichweges enstehen können: Zwei Kreaturen, die sich auf dem gleichen Weg (also zwischen den gleichen Wegpunkten) befinden, müssen sich ausweichen (siehe Abbildung 8.8). Dieses stellt zwar kein großes Problem dar, ist jedoch bei der Berechnung des Ausweichweges zu beachten, und es besteht die Wahl, auf welcher Seite die Kreatur ausweicht. Ist ein Ausweichweg nicht benutzbar, da er von z.B. einem Baum versperrt wird, so kann noch der andere Weg benutzt werden. Es ist weiterhin möglich, dass eine Kreatur nicht ausweichen kann, da ihr Zielwegpunkt ihr zu nahe ist (siehe Abbildung 8.9). Die Kreatur *A* würde sich also über ihr Ziel hinaus bewegen. In diesem Fall muss die Kreatur *B* zunächst versuchen auszuweichen. Kann diese auch nicht ausweichen, so muss die Kreatur *A* sich zurückbewegen und versuchen, einen neuen Weg zum Zielwegpunkt zu finden.

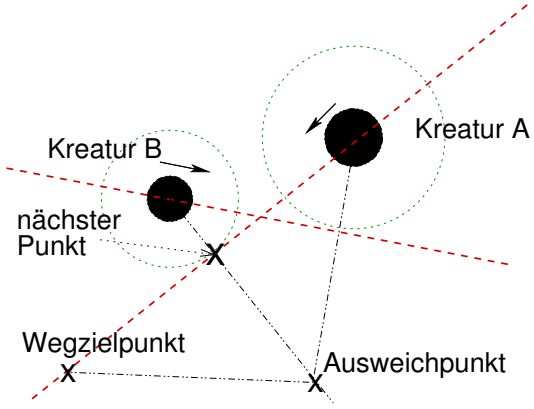

Abbildung 8.6: Realisierung der Berechnung des Ausweichweges der Kreatur A

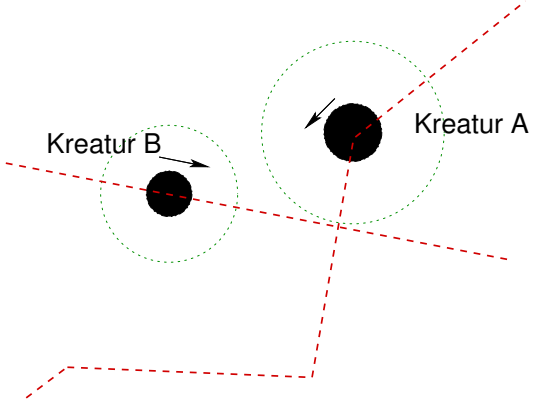

Abbildung 8.7: Der neue Weg der Kreatur A in unserer Realisierung

Ist der nächste Wegpunkt hingegen nicht der Zielwegpunkt, sondern nur ein Zwischenwegpunkt des Gesamtweges, dann wird der ab diesen Wegpunkt veränderte Weg in die Berechnung des Ausweichpunktes und des Wegzielpunktes mit einbezogen (siehe Abbildung 8.10). Normalerweise ist der Wegzielpunkt genau doppelt so weit entfernt, wie die Kreatur A von B, also $2 \cdot \overline{AB}$. In diesem Fall jedoch muss die noch fehlende Strecke auf den veränderten Weg umgelegt werden. Der Wegzielpunkt ist dann genau noch der fehlenden Strecke $((2 \cdot \overline{AB}) - \overline{AW})$ vom Zwischenwegpunkt entfernt. Bevor dieser Ausweichweg allerdings benutzt werden kann, wird noch überprüft, ob sich der Ausweichpunkt auch in einem entsprechenden Bereich befindet (siehe Abbildung 8.11), damit die Kreatur nicht im Zickzack läuft. Andernfalls wird ein Wegzielpunkt berechnet, der genau die Strecke $\overline{AW}$ vom Zwischenwegpunkt entfernt ist und der dann direkt von der Kreatur angelaufen wird. Dieser Fall tritt nur dann ein, wenn der Winkel zwischen den Geraden AW und WZ zu spitz ist. Dieser Sonderfall entsteht natürlich nicht nur wenn sich zwei Kreaturen auf dem selben Weg bewegen, sondern auch wenn die Wege sich kreuzen (siehe Abbildung 8.12 und 8.13).

Die Berechnung des Ausweichweges unter Beachtung aller möglichen Fälle wird in der Aktion `micado_std_avoid` durchgeführt. Welche der kollidierenden Kreaturen nun wartet und welche ausweicht, wird dabei auch festgelegt, wobei zunächst die Kreatur, die als erstes die Kollision erkannt hat, versucht auszuweichen. Kann eine Kreatur ausweichen, so wird an die `micado_std_avoid`-Aktion

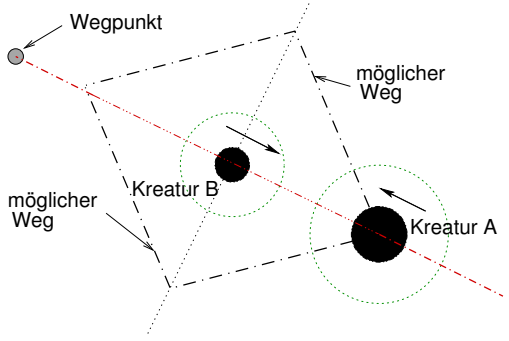

Abbildung 8.8: Zwei Kreaturen bewegen sich auf dem gleichen Weg direkt aufeinander zu

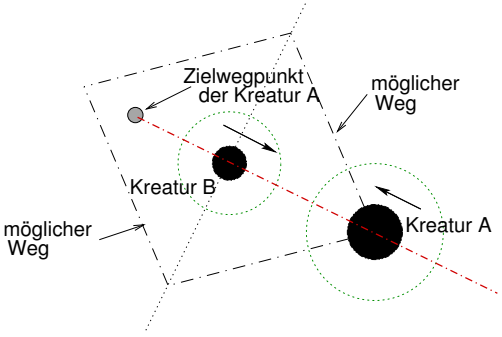

Abbildung 8.9: Die Kreatur *A* kann nicht ausweichen, da sie ihrem Zielwegpunkt zu nahe ist

die Aktion `micado_std_circulate` gehängt. Diese übernimmt nun die Steuerung der Bewegung der Kreatur und lässt diese sich über den Ausweichweg bewegen. Die andere Kreatur bekommt die Aktion `micado_std_wait` eingehängt, so dass sie so lange wartet, bis sich die erste Kreatur weit genug entfernt hat. Nach Abschluss des Ausweich- oder Wartevorgangs wird die entsprechende Aktion beendet und die Aktion `micado_std_walk` übernimmt wieder die Steuerung der Kreatur.

8.4.2 Zum Wegpunktesystem zurückfinden

Wie schon zu Beginn erwähnt, können die Kreaturen das Wegpunktesystem auch verlassen, wenn sie dem Benutzer, d.h. dem Besucher des Caves, folgen. Diese Aktion `micado_std_follow` wurde schon in Abschnitt 8.2 besprochen, allerdings wurde noch nicht erklärt, was nach der Beendigung der Aktion passiert: Beendet werden kann die Aktion beispielsweise direkt vom Benutzer und durch eine Kollision der Kreatur mit einem im Weg stehenden Gegenstand der Umgebung. Nun wird die Aktion `micado_std_follow` durch `micado_std_findback` ersetzt, welche im Folgenden einen Wegpunkt sucht, den die Kreatur anlaufen kann, um wieder in das Wegpunktesystem zu gelangen. Damit die Aktion möglichst schnell einen anlaufbaren Wegpunkt findet, wird zunächst nach den der Kreatur am nahesten liegenden Wegpunkten gesucht. Die Wahrscheinlichkeit, die am nahesten ligenden Wegpunkte erreichen zu können, ohne auf größere Hindernisse der Umgebung zu stoßen, ist hier

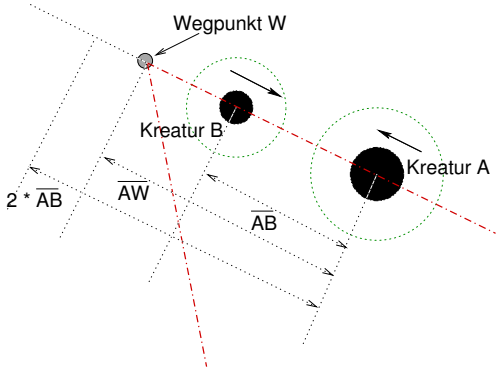

Abbildung 8.10: Berechnung des Ausweichweges mit einem zur Kreatur *A* zu nahen Wegpunkt

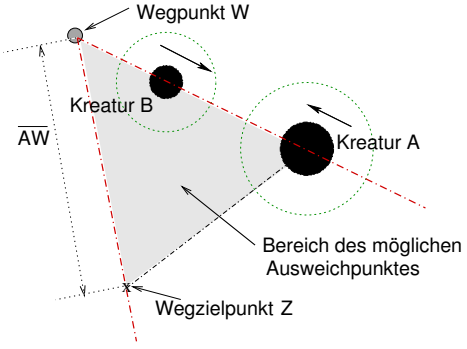

Abbildung 8.11: Ein Ausweichpunkt muss in einem bestimmten Bereich liegen, sonst wird der Wegzielpunkt direkt angelaufen

am höchsten. Die Strecke zwischen der Kreatur und dem Wegpunkt wird auch hier, wie schon beim Ausweichen, nur mit einer *Line of Sight* Anfrage auf Begehbarkeit getestet, damit wir nicht unnötig viel Rechenleistung der CPU in Anspruch nehmen. Wird hingegen kein anlaufbarer Wegpunkt gefunden, dann kann die Kreatur nur direkt auf einen Wegpunkt versetzt werden. Hierfür wird die Aktion `micado_std_respawn` gestartet, die dann zunächst eine Sterben-Animation der Kreatur abspielt und dann die Kreatur auf einem Wegpunkt versetzt. Dort startet die Kreatur wieder ihre Aktionen ganz normal nach ihren Motivationsleveln (siehe Abschnitt 8.1).

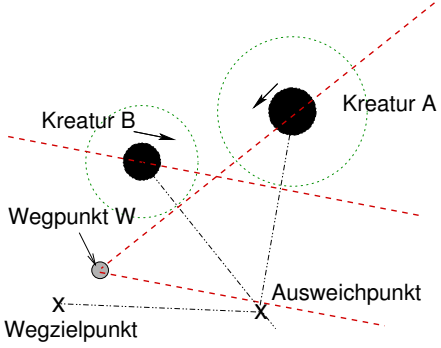

Abbildung 8.12: Ein weiteres Beispiel der Berechnung eines Ausweichweges

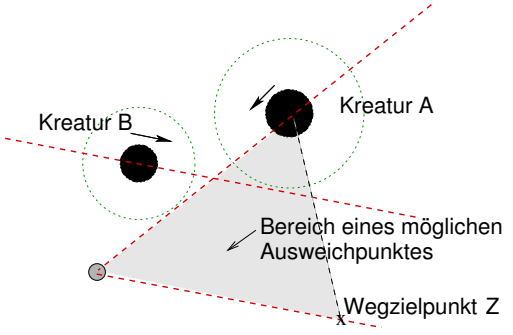

Abbildung 8.13: Bereich eines moeglichen Ausweichpunktes fuer das vorherige Beispiel

— KAPITEL 9 —

Sound

Mag die Cave-Sensorik, Modellierung sowie KI- und Grafikprogrammierung den größten und wichtigsten Teil der Projektarbeit ausmachen – das richtige Dschungel-Feeling will sich ohne eine passend gestaltete Soundkulisse nicht so recht einstellen. Daher waren zum einen die konkreten Handlungen der modellierten Tiere zu vertonen und darüber hinaus für den typischen Umgebungssound zu sorgen, welchen man gemeinhin von einer belebten Wildnis erwartet: Grillen im Gebüsch, Vögel in den Baumwipfeln, Frösche im Laub, etc. Obwohl die Verursacher der Geräusche der zweiten Kategorie für den Benutzer nicht sichtbar sind[1], ergibt sich in beiden Fällen die Anforderung der Ortbarkeit aus den Cave-Eigenschaften: Da die virtuelle Welt den Benutzer seitlich auf mehreren Leinwänden umgibt, soll dieser den Eindruck bekommen, dass er sich ebenfalls inmitten der Klangkulisse befindet, und die einzelnen Sounds aus verschiedenen, gut feststellbaren Richtungen kommen. Im Falle einer sichtbaren Klangquelle (wie beispielsweise einem knurrenden Tiger) soll die wahrgenommene Richtung des Geräusches mit dem Ort der visuellen Projektion seines Verursachers so gut wie möglich übereinstimmen. Während das Floß durch den Urwald treibt, bewegen sich somit die einzelnen Sounds am Benutzer vorbei und um ihn herum, werden dabei entfernungsbedingt leiser und lauter, wandern von einer Leinwand auf eine andere. Wir werden im Folgenden erläutern, welche Möglichkeiten uns die Madness Engine in der Soundgestaltung bereits zur Verfügung stellt, und welche Funktionen wir zusätzlich implementieren mussten, um diesen Effekt zu realisieren. Abschließend beschäftigen wir uns in Abschnitt 9.4 mit dem gestalterischen Aspekt der Vertonung.

[1] Für sie haben wir keine 3D-Modelle erstellt, lediglich ihr Klang soll auf ihre Existenz hinweisen.

9.1 Unser Ausgangspunkt

Während die Version der Madness Engine, mit der wir unsere Arbeit begannen, von Anfang an das verteilte Rendern auf mehreren Rechnern unterstützte und sich somit hinsichtlich der Grafik bereits optimal für den Einsatz in einem Cave eignete, waren ihre Soundfähigkeiten noch nicht umfangreich genug, als dass sie für den gewünschten dreidimensionalen Klangeindruck hätten sorgen können. Durch Verwendung der SDL-Library[2] und der dazugehörigen SDL_mixer-Erweiterungsbibliothek ist die Madness in der Lage, einzelne Sounddateien zu laden und auf eine Stereosumme pro Rechner zusammengemischt abzuspielen. Sie stellt dafür zwei Klassen zur Verfügung: `madness_sound_sample` zur Repräsentation der geladenen digitalen Klangsignale und `madness_sound` zum Abspielen der Klänge mit einstellbarer Lautstärke, Reichweite und Position im dreidimensionalen Raum. Die Art und Weise, wie die letzten drei Parameter in die Berechnung des Ausgabesignals der Soundkarte einflossen war anfänglich sehr einfach und eher für den Einsatz an einem Rechner mit einem Stereokopfhörer zur Erzeugung eines eindimensionalen Rechts/Links-Effekts geeignet. Es ist offensichtlich, dass mit lediglich zwei Lautsprechern an einer einzelnen Stereosoundkarte der beabsichtigte dreidimensionale Effekt für unser Cave-Szenario nicht erreicht werden kann. Hier ist entweder eine Mehrkanalsoundkarte erforderlich oder mehrere synchronisierte Stereokarten. SDL unterstützte während unserer Projektlaufzeit jedoch noch keine Mehrkanalsoundkarten, auch sahen weder SDL noch Madness eine Synchronisation mehrerer Zweikanalkarten vor – die einzelnen Rechner des Master/Slave-Verbunds waren bezüglich ihrer Audioausgabe vollkommen unabhängig voneinander, da die Soundunterstützung nicht an die Netzwerkimplementierung der Engine gekoppelt war. Mit der Erzeugung des räumlichen Klangs beschäftigen wir uns in Abschnitt 9.2.

Ein weiteres Problem hat seinen Ursprung in der Weise, wie mehrere gleichzeitig abzuspielende Klänge softwareseitig zusammengemischt werden: Es wird das arithmetische Mittel der Signalwerte gebildet. Hierdurch sinkt jedoch der Anteil, den ein individueller Sound zum Ausgabesignal beiträgt: Die einzelnen Klänge werden um ein vielfaches leiser und bezüglich ihrer Qualität resp. Auflösung in Bits schlechter. Es ist somit nicht ratsam, für alle möglichen Soundquellen der virtuellen Dschungelwelt dauerhaft einen Kanal des Software-Mixers zu reservieren, zumal sich zu jedem Zeitpunkt die wenigsten dieser Quellen in ausreichender Nähe zur Kamera befinden, so dass sie tatsächlich hörbar sind. Es war daher eine Verwaltung zu implementieren, die eine kleine Menge von Softwarekanälen dynamisch an jene Soundquellen vergibt, welche sich am nahesten zur Kamera

[2] *Simple DirectMedia Layer*, siehe [SDL04].

befinden, und alle anderen Quellen stummschaltet[3]. Diese Soundquellenverwaltung fungiert zudem als einheitliches Interface zur Kreatur-KI, mit dessen Hilfe die simulierten Dschungelbewohner auf einfache Art und Weise Klänge abspielen und stoppen können, ohne deren tatsächliche Hörbarkeit in Betracht ziehen zu müssen. Eine Besprechung unserer Implementierung findet sich in Abschnitt 9.3.

9.2 Spatial Hearing: Die Soundmodelle

Die Mechanismen, durch die der Mensch mit seinen nur zwei Ohren Geräusche in seiner dreidimensionalen Umgebung lokalisieren kann, sind erst in Teilen verstanden und stellen ein aktuelles Forschungsgebiet dar[4]. Dass es zwei Haupthinweise gibt, die das Gehirn zur Abschätzung der Position einer Soundquelle verwendet, ist jedoch sicher: Dies ist einerseits die *Interaural Level Difference (ILD)* und zum anderen die *Interaural Time Difference (ITD)*. Die ILD bezeichnet den frequenzabhängigen Lautstärkeunterschied zwischen beiden Ohren aufgrund der Eigenschaft des menschliches Kopfes, Schallwellen abzudämpfen, die ITD meint die frequenzbedingte Signalverzögerung zwischen den Ohren aufgrund deren unterschiedlicher Distanz zur Soundquelle. Die ITD dient vor allem als Lokalisationshinweis für tiefe Frequenzen bis 1,5 kHz, die ILD hingegen hauptsächlich als Hinweis für hohe Frequenzen ab 1,5 kHz (vgl. dazu [Pul01]).

Während bei einer Mono-Ausgabe mit einem Lautsprecher eine Punktsoundquelle erzeugt werden kann, deren vom Hörer wahrgenommene Richtung mit der tatsächlichen Position des Lautsprechers zusammenfällt, kann mit einer Stereokonfiguration die vom menschlichen Gehirn festgestellte Soundposition entlang der Strecke (bzw. des Kreisbogenstücks) zwischen den beiden Boxen variiert werden. Da an dieser Stelle in der Regel keine reale Soundquelle existiert, sprechen wir hier von einer *virtuellen* Quelle, wohingegen die Lautsprecher *reale* Soundquellen darstellen. Abbildung 9.1 zeigt auf der linken Seite ein solches Stereoszenario. Durch Hinzunahme weiterer Boxen kann der Klangeindruck auf eine Ebene (beispielsweise bei einem 5.1 Surround System) bis hin auf drei Dimensionen ausgedehnt werden. Der Autor von [Pul01] stellt z.B. ein Verfahren namens *Vector Base Amplitude Panning (VBAP)* vor, mit dem eine beliebige Anzahl von Lautsprechern (etwa in gleichem Abstand zueinander) um die Hörumgebung herum angeordnet werden kann, um einen dreidimensionalen Effekt zu erreichen.
Amplitude Panning (AP) bezeichnet hierbei die Ausnutzung der ILD durch die gleichzeitige Ausgabe

[3] Alternativ hätte eine neue Mischfunktion mit eingebauter Dynamikkompression des Ausgabesignals implementiert werden können, was wir jedoch aufgrund des hohen Entwicklungsaufwands nicht näher in Betracht zogen.

[4] Siehe dazu z.B. [Pul01] und [PK01].

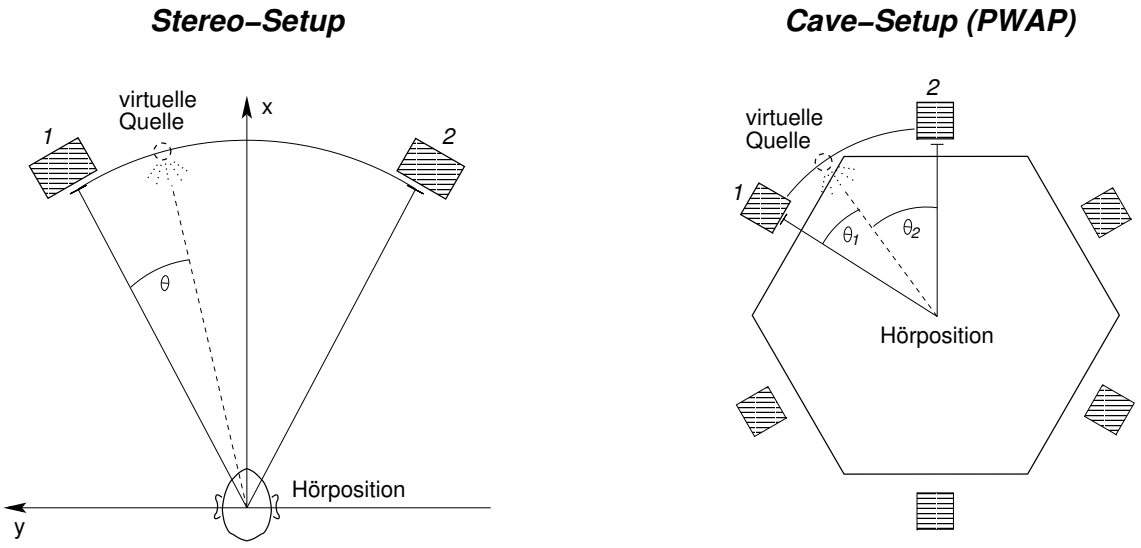

Abbildung 9.1: Amplitude Panning: Stereo- und Cave-Konfiguration von oben betrachtet

eines Sounds auf einem bis zu mehreren (jedoch nicht zwingend allen) Lautsprechern, auf jedem mit individueller Lautstärke. Die einzelnen Lautstärken sind abhängig von der gewünschten wahrgenommenen Richtung der virtuellen Quelle. Bei *Pair Wise Amplitude Panning (PWAP)* wird ein Soundsignal von maximal zwei direkt benachbarten Boxen des Setups gleichzeitig ausgegeben, die virtuelle Quelle befindet sich also stets auf genau einem Lautsprecher oder zwischen zweien. Bei VBAP werden hingegen Lautsprechertripletts betrachtet. Abbildung 9.1 zeigt rechts ein PWAP-Setup für unseren Cave mit jeweils einem Lautsprecher in der Mitte jeder Cave-Seite.

Sei unser auszugebenes Soundsignal als Funktion über die Zeit $x(t)$ gegeben und sei $S = \{1, \ldots, n\}$ die Menge der benachbarten Lautsprecher, zwischen denen die virtuelle Quelle liegen soll, mit bspw. $n = 2$ für PWAP. Dann sei

$$x_i(t) = g_i \cdot x(t)$$

das von Lautsprecher $i \in S$ ausgegebene Signal. Hierbei sind $0 \leq g_i \leq 1$ die *Gain-Faktoren*: Es ist g_i die Stärke, mit der Lautsprecher i das Signal $x(t)$ ausgibt. Alle Boxen, welche sich nicht in direkter Nachbarschaft zur Soundquelle befinden, also nicht in der Menge S enthalten sind, bekommen den Gain-Faktor 0. Die Werte sind abhängig von der Richtung der virtuellen Quelle so zu wählen, dass die von allen Boxen erzeugte Gesamtlautstärke (bei gleichbleibender Distanz der virtuellen Quelle zum Hörer) richtungsunabhängig stets konstant bleibt (*Equal Power Panning*). Hierzu ist für nicht-schalltote Räume die folgende Forderung zu erfüllen:

$$\sqrt{\sum_{i=1}^{n} (g_i^2)} = c, \quad \text{z.B. } c = 1 \tag{9.1}$$

Es ist folglich eine Funktion zu suchen, mit der aus dem Winkel der virtuellen Soundposition zu den Boxen aus S die zu verwendenden Gain-Faktoren so berechnet werden können, so dass (9.1) stets gilt. Wegen $sin^2(\varphi) + cos^2(\varphi) = 1$ bieten sich für $\varphi \in [0, \frac{\pi}{2}]$ und $n = 2$

$$g_1 = cos(\varphi), \quad g_2 = sin(\varphi)$$

an, wobei φ der Winkel zwischen Lautsprecher 1 und der virtuellen Quelle ist. Beträgt der Winkel zwischen den Boxen 1 und 2 genau 90°, so kann, wegen $cos(\varphi) = sin(\varphi + \frac{\pi}{2})$, der Faktor g_i allgemeiner mit

$$g_i = cos(\theta_i)$$

angegeben werden, wobei θ_i der Winkel zwischen Lautsprecher i und der virtuellen Quelle ist. Für Aufstellwinkel ungleich 90° ist θ_i einfach passend auf den Wertebereich $[0, \frac{\pi}{2}]$ zu skalieren, für unseren Cave mit 60° also $g_i = cos(\frac{3}{2} \cdot \theta_i)$. Abbildung 9.2 illustriert auf der linken Seite, wie für zwei benachbarte Lautsprecher, zwischen denen die virtuelle Quelle liegen soll, die Gain-Faktoren bestimmt werden: Es sind die Cosinusfunktionen und Normalenvektoren der Leinwände eingezeichnet sowie die Winkel θ_1 und θ_2 für die Cave-Seiten 1 und 2, zwischen denen in unserem Beispiel die Soundquelle liegt.

Bis hierhin beschränken sich unsere Überlegungen auf einen Audiokanal pro Leinwandseite und die zweidimensionale Ebene (die xz-Ebene des Kamerakoordinatensystems), auf der in diesem Fall alle sechs Boxen liegen. Aufgrund der Equal Power Eigenschaft (9.1) der Gainfunktionen können wir

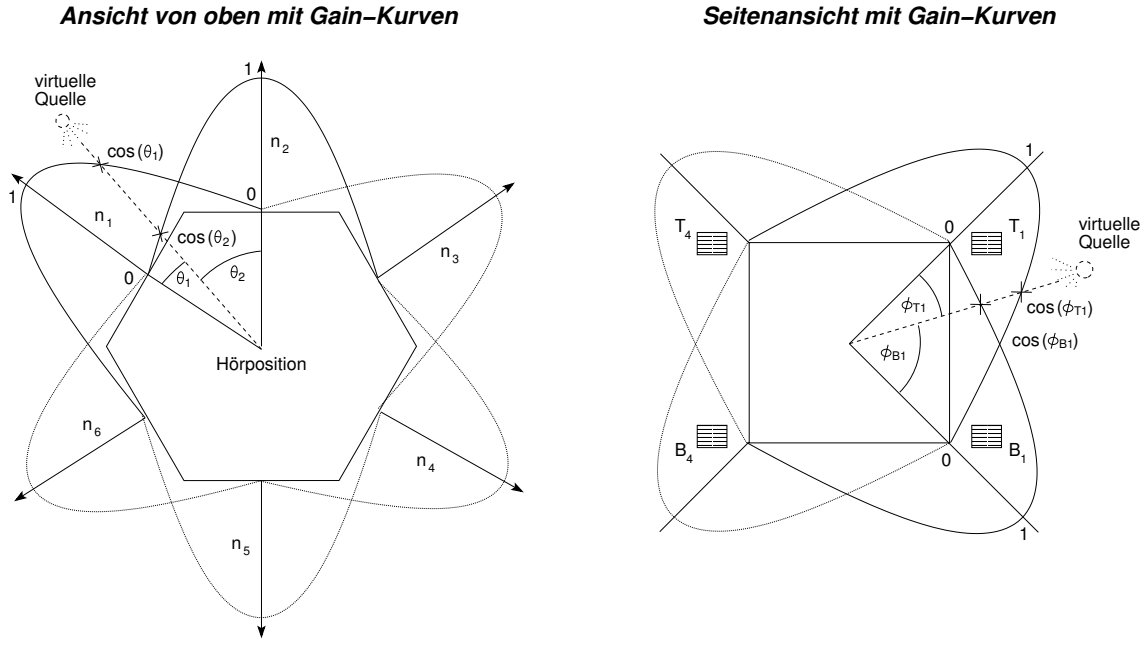

Abbildung 9.2: Amplitude Panning im Cave: Veranschaulichung der Gain-Funktionen

den Ansatz jedoch leicht um die noch fehlende Höheninformation ergänzen, indem wir zwei Audiokanäle pro Cave-Seite i verwenden und dabei einen Lautsprecher T_i an der oberen Leinwandkante, den anderen B_i an der unteren Kante jeweils mittig montieren. Abbildung 9.2 zeigt rechts diese Erweiterung auf zwei Boxen pro Leinwand. In einem ersten Schritt wird dabei anhand des Winkels θ_i die Gesamtlautstärke g_i einer Leinwand bestimmt, danach aus den Winkeln ϕ_{Ti} und ϕ_{Bi} die Anteile g_{Ti} und g_{Bi} berechnet, mit denen T_i und B_i zu dieser beitragen.

Da die verteilte Grafikberechnung der Madness nicht an eine bestimmte Leinwandanzahl gebunden ist, bleibt bei der Realisierung des erweiterten Ansatzes das Audiosystem ebenfalls maximal modular, wenn wir pro Rechner (und somit pro Seitenfläche des Caves) eine Stereosoundkarte verwenden, und jeder Computer des Verbunds selbst für die Berechnung seiner Gain-Faktoren zuständig ist. Die

Positionen der Klänge sowie Informationen darüber, wann ein solcher abzuspielen und wieder zu stoppen ist, sind dabei vom Master an alle Slaves zu verteilen.

Problematisch ist hierbei, dass eine perfekte Synchronisation der Soundkarten über das Netzwerk praktisch unmöglich ist. Sendet beispielsweise der Master allen Slaves via UDP Multicast den Befehl, einen bestimmten Klang abzuspielen, so werden ihn die Empfänger in der Regel alle mit unterschiedlichen Verzögerungen ausführen. Letztere sind beispielsweise davon abhängig, wann ein Rechner das jeweilige Netzwerkpaket erhält, dessen Betriebssystem das Paket an die Cave-Anwendung weitergibt[5], diese es bearbeiten kann, und wann die von ihr gemachten neuesten Audioausgaben spätestens hörbar werden[6]. Werden diese Verzögerungen zu groß, dann nimmt der Benutzer ein, von zwei benachbarten Rechnern wiedergegebenes, Geräusch nicht mehr als eine einzige virtuelle Quelle wahr, sondern als zwei getrennte. Auch wird dann bei Richtungsveränderungen die Equal Power Eigenschaft hörbar verletzt, wenn zwei Rechner zu einem Zeitpunkt stark unterschiedliche Werte für den Zeitparameter t verwenden.

Nach einer Betrachtung, mit welchen Verzögerungen zwischen den Leinwänden in der Praxis zu rechnen ist, haben wir uns jedoch für eine Realisierung des oben vorgestellten Systems entschieden, mit der Option, bei zu schlechten Ergebnissen auf die Verwendung einer Mehrkanalsoundkarte zurückzugreifen[7]. Bei Verwendung eines Ringpuffers von 512 Bytes für die Stereo-Soundausgabe in CD-Qualität[8], deckt dieser genau 128 Stereo-Abtastwerte ab und somit etwa 2,9 ms. Bei den Netzwerkpaketen gehen wir einmal von einer Verzögerung von grob 0,3 ms aus, was uns nach einzelnen Tests durchaus realistisch erscheint. In dieser Verzögerung ist eine Scheduling Latenz von unter 0,1 ms in etwa 92% der Fälle[9] bereits enthalten. Insgesamt haben wir demnach mit einer durchschnittlichen Latenz zwischen 3 und 4 ms zu rechnen. Wenn man bedenkt, dass der Schall bei Raumtemperatur etwa 2,92 ms benötigt, um eine Strecke von einem Meter zurückzulegen, scheinen dies relativ gute Rahmenbedingungen zu sein: In einem Kinosaal der Größe 15 m $\times$ 15 m sind abhängig von der Hörposition Verzögerungen von bis zu 60 ms zwischen dem Eintreffen der Schallwellen der einzelnen Boxen beim Hörer möglich[10]. Unser Cave hat deutlich kleinere Abmessungen, der Schall benötigt daher nur wenige Millisekunden, um von einer Leinwandseite zur gegenüberliegenden zu gelangen. Im Vergleich zum Szenario des Kinosaals erschien uns die im Cave zu erwartende (eben-

[5] Stichwort: *Kernel Latency*.
[6] Dies ist abhängig von der Größe des verwendeten Audio-Ringpuffers.
[7] Unter Wechsel von SDL_mixer auf eine andere Soundbibliothek wie OpenAL, siehe auch [OAL04].
[8] 16 Bit pro Abtastwert und Kanal bei 44,1 kHz Abtastrate ergibt eine Datenrate von 176.400 Bytes/sec.
[9] Wir stützen uns hier einmal auf [Wil02], eine Untersuchung des Linux Kernels 2.4.17 zu diesem Thema.
[10] Im Gegensatz zu unserem Szenario ist diese Verzögerung über die Zeit konstant.

falls hörpositionsabhängige) Gesamtverzögerung ausreichend klein zu sein, um eine Umsetzung des verteilten Soundsystems zu wagen.

Unsere Implementierung des in diesem Abschnitt vorgestellten Amplitude Panning Ansatzes ist in `madness_sound.cpp` zu finden und sorgt dafür, dass in jedem Frame die Gain-Faktoren pro Sound und Lautsprecher aktualisiert werden. Drei Soundmodelle sind möglich: Das vorgestellte System mit herkömmlicher Projektoraufstellung, mit um 90° gedrehter Beamermontage (in unserem Cave verwendet) sowie das alte Modell für Stereokopfhörer. Das gewünschte Szenario ist auf jedem Rechner des Verbunds über die Umgebungsvariable `MADNESS_SOUND_MODEL` einzustellen. Darüber hinaus gibt es weitere Variablen zur Festlegung der Boxenaufstellwinkel und zur Links/Rechts-Umkehr der Stereoausgabe zwecks Vereinfachung der Lautsprechermontage.
Zur Netzwerksynchronisation haben wir `micado_network_sample` und `micado_network_sound` als Unterklassen von `madness_sound_sample` resp. `madness_sound` implementiert.

9.3 Das Interface zur Kreatur-KI: Die Soundmanager

Passend zu den verschiedenen Aktionen, die eine Kreatur im Laufe ihres virtuellen Lebenswandels ausführen kann, gibt es eine Menge von Soundeffekten, welche diese akustisch untermalen. Für jede Kreatur ist in ihrer Konfigurationsdatei anzugeben, welche Sounddateien für sie verwendet werden sollen und mit welcher Lautstärke, Reichweite und Priorität diese abzuspielen sind, sowie in welchem Loop-Modus. Nach Laden der Konfiguration befinden sich die nötigen Samples auf allen Rechnern des Verbunds im Speicher und die Kreatur unterhält auf dem Master eine Tabelle mit den Abspieloptionen.

Um die einzelnen Klänge abzuspielen, verwenden die Kreaturen aus eingangs aufgeführtem Grund nicht direkt die Klassen `micado_network_sample` und `micado_network_sound`, sondern benutzen die Soundmanager, welche ein einheitliches, einfaches Interface zur Soundwiedergabe zur Verfügung stellen. Jeder dieser Soundmanager ist für die Verwaltung einer ihm unterstellten Menge von Kanälen des SDL-Softwaremixers zuständig und führt eine Liste aller prinzipiell von ihm abzuspielenden Soundquellen. Nur die mit der kürzesten Distanz zur Kamera[11] werden dabei berücksichtigt und bekommen von ihm einen Kanal zugeteilt, der Rest wird stummgeschaltet. Die Abspiellisten werden vor dem Rendern eines jeden Frames automatisch aktualisiert. Wie die Kreatur-KI arbeiten die So-

[11] Präziser: Mit dem kleinsten Produkt von Abspielpriorität und Distanz zur Kamera.

undmanager ausschließlich auf dem Master, die Netzwerksynchronisation geschieht durch die von uns implementierten Netzwerk-Soundklassen.

Es gibt zwei Typen von Soundmanagern: Objekte der Klasse `micado_oneshot_manager` sind für einmalig abzuspielende Sounds zuständig und `micado_loop_manager`-Objekte für die Wiedergabe in Schleifenwiederholung. Welcher der beiden Manager für einen Klang zu verwenden ist, ist abhängig von dessen in der Kreaturkonfiguration angegebenen Loop-Modus. Beide bieten drei Hauptfunktionen an: Zur Aufnahme eines Sound in die Playlist zwecks Wiedergabe dient `play()`, zum Entfernen dienen `stop()` und `abort()`. Letztere zwei unterscheiden sich in ihrer Funktion lediglich bei `micado_oneshot_manager`: Dieser spielt einen Effekt einmalig ab und entfernt ihn selbständig wieder aus seiner Liste, sobald die Wiedergabe abgeschlossen ist. Die `stop()`-Funktion hat bei ihm daher keinen Effekt, `abort()` hingegen führt auch bei ihm zum sofortigen Verstummen des Sounds. Der `play()`-Funktion kann entweder eine statische Position für die virtuelle Quelle übergeben werden oder ein Pointer auf ein Kreatur-Objekt, so dass das Soundsystem automatisch in jedem Frame die aktuelle Soundposition von der gegebenen Kreatur beziehen kann. Die Manager ignorieren eine `play()`-Anfrage, wenn der betreffende Sound an der angegebenen Position bzw. für die jeweilige Kreatur bereits in der internen Liste vorhanden ist, `stop()` und `abort()` hingegen, wenn der gegebene Sound nicht (mehr) in ihr steht.

Unsere Cave-Anwendung verteilt die von ihr verwendeten 18 Softwarekanäle auf drei Soundmanager: Einen `micado_oneshot_manager` und einen `micado_loop_manager` zur Verwendung für die Kreatursounds, sowie einen weiteren Loop-Manager zur Verwaltung der statischen Umgebungssounds. Letztere sind in der BSP-Datei, welche die Geometrie unserer Dschungellandschaft enthält, definiert und werden bereits beim Laden der virtuellen Welt in die Abspielliste eingetragen.

9.4 Sounddesign

Die Soundeffekte für die Bewohner unseres Dschungels wurden im Anschluss an die Gestaltung der animierten 3D-Modelle erstellt. Hierzu wurden u.a. Exkursionen in den Bremer Bürgerpark unternommen, um mit einem digitalen Camcorder samt Richtmikrofon auf möglichst kreative Art und Weise akustisches Rohmaterial zu sammeln. Zudem stellten sich einzelne Projektmitglieder als Geräuschquelle zur Verfügung – beispielsweise durch lautstarkes Essen einer Pizza oder Leeren einer Flasche Wasser. Mit Hilfe diverser Audiobearbeitungssoftware wurde so weit am Ausgangsmaterial gearbeitet, bis es sich am Ende tatsächlich nach einer kriechenden, schleimigen Schnecke anhörte,

oder nach einem fressenden Tiger. Des Weiteren wurden verschiedenste Soundsammlungen[12] nach passenden Klängen durchforstet, die dann auch für die Gestaltung der Umgebungsgeräusche verwendet wurden.

Für die Umgebungssoundkulisse, welche klanglich sicherlich am meisten zum angestrebten Dschungeleindruck beiträgt, wurden zahlreiche Einzelsounds, wie z.B. singende Vögel, summende Insekten und quakende Frösche, zu mehreren längeren, in Endlosschleife abspielbaren, Samples zusammengemischt und nach Erstellung der virtuellen Dschungelwelt im Leveleditor in ausreichendem Abstand zueinander in der Map platziert.

Während das von uns mit einfachen Mittel erreichte Gesamtergebnis den virtuellen Urwald recht gut vertont, entspricht es qualitativ selbstredend nicht dem Ergebnis eines anspruchsvollen Sounddesigns unter Verwendung professioneller Tonstudiotechnik sowie dazugehörigem Know-How. Dies war auch nicht unser Anspruch, zumal die von uns im Cave montierten Aktivboxen aufgrund unseres sehr schmalen Projektbudgets dem untersten Preissegment angehören und sich somit qualitativ für die Wiedergabe von hochwertigem Audiomaterial ohnehin nicht wirklich eignen. Der Cave-Sound präsentiert sich daher zwar von seiner quäkenden Seite, weist jedoch durchaus, und das trotz der in Abschnitt 9.2 besprochenen Latenzproblematik zwischen den Leinwandseiten, den beabsichtigten dreidimensionalen Effekt auf: Die Geräusche scheinen stets aus der Richtung zu kommen, in der auf der Leinwand die sie verursachende Kreatur zu sehen ist. Weiterentwicklungen des Soundsystems unseres Caves bieten sich daher einerseits im Verbau etwas höherwertiger Audiokomponenten in Verbindung mit einem anspruchsvollen Sounddesign an, zudem in einem Umstieg von der Implementierung mit mehreren netzwerksynchronisierten Stereosoundkarten zu einer Verwendung einer einzigen Mehrkanalkarte durch den Master.

[12] Wie diverse Spielfilme, Audio-CDs, Samplearchive für Akai-Studiosampler, das Internet, etc. pp.

— KAPITEL 10 —

Modellierung

10.1 Levelmodellierung

Ein Kernstück unserer Dschungelanwendung ist natürlich der Dschungel selbst. Hierfür muss ein 3D-Modell des virtuellen Dschungels erstellt werden. Neben der eigentlichen Modellierung der 3D-Welt, müssen Grafiken und Töne zur Verwendung in der Welt erstellt werden. Auch wenn einiges Wissen über die zur Erstellung verwendeten Programme angeeignet werden muss, ist dieser Prozess eher als künstlerisches Schaffen anzusehen. Die Herausforderung dieses Schaffensprozesses liegt darin, die statische, eckige Welt lebendig wirken zu lassen.

Das Generieren eines Levels (einer virtuellen Welt) für Quake III bzw. für die Madness Engine, bedarf mehrerer Schritte. Zunächst müssen Texturen erstellt werden, mit denen später die Polygone des Levels belegt werden können. Diese Texturen müssen teilweise mit besonderen Fertigkeiten versehen werden. Dies geschieht durch Texturskripte, so genannte *Shader*. Für unsere Designanforderung – die Erstellung einer realistischen Dschungellandschaft – ist es sinvoll, sich hilfreicher Tools zu bedienen, mit denen man leicht Berg-, Tal- und Flussverläufe erzeugen kann. Dies kann man mit *Heightmaps* realisieren.
Erst jetzt beginnt die Arbeit mit dem *Level-Editor*. Ist das Level soweit fertig, muss man mit einer entsprechenden Kompilierung dafür sorgen, dass das Level in ein für die Madness Engine lesbares Format konvertiert wird.

10.1.1 Arbeitsablauf

Der Arbeitsablauf ist in Abbildung 10.1 schematisch dargestellt.

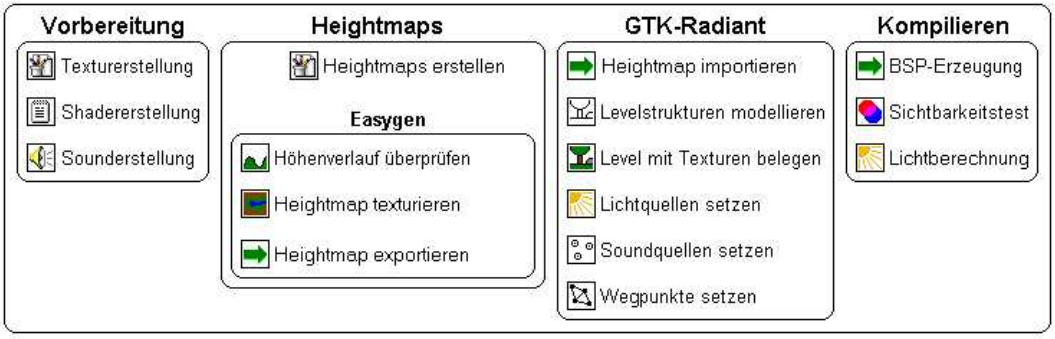

Abbildung 10.1: Schematische Darstellung des Arbeitsablaufes

Textur-Erstellung

Eine Textur, die im Level verwendet wird, kann ein jedes beliebiges digitales Bild sein. Das Bild kann mittels entsprechender Bildbearbeitungstools gezeichnet werden, oder wenn man realistische Ergebnisse erzielen möchte, können reale Objekte und Strukturen mit einer Digitalkamera fotografiert werden. Hierbei muss man natürlich darauf achten, sinnvolle Bildausschnitte zu wählen. Möchte man zum Beispiel einen Baum modellieren und mit entsprechenden Texturen belegen, ist es wenig sinnvoll den gesamten Baum zu fotografieren. In diesem Fall würde man eine Nahaufnahme der Rinde machen, die dann auf alle *Polygone* des Stamms und der Äste gelegt wird und ein zweites Bild, das je nach gewünschtem Detailgrad der Modellierung einzelne Blätter oder kleine Äste mit Blättern enthält.
Einzige durch die 3D-Engine vorgebene Einschränkungen sind bestimmte Formatierungen des Bildes, die eingehalten werden müssen. Die Werte sowohl für Höhe, als auch für Breite des Bildes müssen Zweierpotenzen entsprechen. Höhe und Breite müssen nicht gleich sein. Die kleinste sinnvolle Textur ist wohl 8×8, jedoch sind 64×64 bis 512×512 die gebräuchlicheren Größen.
Als Dateiformate können *JPEG* (.jpg) und *TARGA* (.tga) verwendet werden. Um jedoch eine bessere

Farbqualität zu erreichen, sollte das TARGA-Format in 24-Bit-Qualität verwendet werden. Will man bei einzelnen Texturen Transparenz-Efekte nutzen, d.h. diese partiell durchsichtig machen, müssen 32-Bit-TARGAs verwendet werden. Beim 32-Bit-TARGA kann der vierte, nicht für Farbinformationen genutze Kanal, zur Festlegung der transparenten Bereiche genutzt werden (*Alpha-Kanal*).
Ein Sachverhalt muss bei Texturen beachtet werden, die großflächig genutzt werden: Wird eine Textur auf ein Polygon gelegt, das größer ist als die Textur selber, wird die Textur wiederholt dargestellt. Dies ist zum Bespiel bei grossen Rasenflächen der Fall. Hier muss darauf geachtet werden, das die gegenüberliegenden Ränder der Textur in Farbe und Muster nicht zu stark entgegenlaufen. Sind die Texturränder zu verschieden, entsteht auf dem Polygon ungewollt ein Karomuster in Abständen der Texturgröße. Dies kann einen gewollten natürlichen Eindruck der Fläche zerstören.

Shader-Erstellung

Viele Texturen können nicht einfach ohne Weiteres auf die Polygone gelegt werden. Soll eine Textur oder das dazugehörige Polygon besondere Anforderungen erfüllen, muss ein Texturskript erstellt werden, das besondere Eigenschaften der Textur definiert. Diese Skripte werden als Shader bezeichnet. Alle Shader eines Levels werden in einer Datei zusammengefasst. Diese Shader-Datei lässt sich mit einem beliebigen Texteditor bearbeiten.
Im Grunde muss für jede Textur ein Shader vorliegen, denn einige Informationen, wie z.B. ob die Textur für den Benutzer passierbar ist oder nicht, müssen für jede Textur vorliegen. Sollte für eine Textur in der Shader-Datei kein Eintrag vorliegen, nimmt die 3D-Engine für diese Textur einen Satz von Standardeinstellungen an.

Die Möglichkeiten, die die Shader bieten, umfassen ein weites Spektrum. Einige Einstellungen betreffen mehr das Polygon selbst, als die aufliegende Textur. Die bereits genannte Information, ob ein Polygon passierbar ist oder nicht, gehört zu den wichtigsten. Definitionen von Texturmaterialien, z.B. dass es sich um Wasser handelt, können festgelegt werden. Hinzu kommen einige Quake III-spezifische Funktionen, die für uns nicht von Belang sind, wie z.B.: Sprungflächen.
Für uns sehr wichtig sind die Shader-Befehle zur grafischen Darstellung. Man kann die Transparenzeigenschaften der Textur festlegen, man kann Animationen aus bis zu acht Einzelbildern definieren, Bewegungen und Verzerrungen der Textur lassen sich definieren und man kann zahlreiche Eigenschaften das Lichtes betreffend definieren, wie z.B., ob eine Textur Licht emmitiert, das Licht nicht reflektiert oder einen konstanten Helligkeitswert unabhängig von der Umgebung hat.
Aus einer Kombination all dieser Shader-Befehle lassen sich dann Texturen, wie zum Bespiel, sich

leicht bewegendes, stellenweise durchsichtiges Wasser realisieren. Solche Texturen wirken sehr viel lebendiger als statische Bilder.

Erstellung einer Grundlevelstruktur mittels Heightmaps

Zur Erstellung realistisch wirkender Naturlevels, ist es wichtig, einen natürlich wirkenden Höhen- und Tiefenverlauf der Landschaft zu erschaffen. Gerade Flüsse, regelmäßige Berge und planare Ebenen wirken sehr künstlich. Es ist schwer, eine realistisch wirkende Landschaft per Hand zu Modellieren.
Um zu realistischen Ergebnissen zu kommen, bietet sich die Verwendung von *Heightmaps* an. Heightmaps sind Grauwertbilder beliebiger Größe. Die Grauwerte von 0 bis 255 repräsentieren, ähnlich einer geologischen Landkarte, Höhenwerte der Landschaft. 0 steht für einen, später zu definierenden tiefsten und 255 für einen höchsten Punkt. Mit einem beliebigen Bildbearbeitungsprogramm kann eine solche Höhenkarte leicht erstellt werden. Flussläufe werden sehr dunkel gezeichnet, ebene Landschaft etwas heller, kleine Erhebungen wieder etwas heller und hohe Berge sehr hell. Indem man anschließend mit einem, oder mehreren verschiedenen chaotischen Mustern die gesammte Karte leicht überlagert, erhält man einen sehr natürlich wirkenden Höhenverlauf.
Ist die Heightmap fertiggestellt, kann man sie in das Freewaretool Easygen importieren. Dieses Tool stellt die Heightmap als 3D-Modell dar, ermöglicht Korrekturen am Höhenverlauf und kann die Heightmap in das Format des Standard-Quake III-Editors GTK-Radiant überführen. Darüber hinaus bietet es die Möglichkeit, das 3D-Modell der Landschaft sehr einfach mit Texturen zu belegen. Außerdem erstellt es automatisch die Shader, die notwendig sind, um Überblendungen zwischen zwei Lanschaftstexturen, zum Bespiel einem weichen Übergang von Gras zu Fels, zu ermöglichen. Sind alle Korrekturen am Höhenverlauf und die Belegung mit Texturen abgeschlossen, kann das gesammte 3D-Modell in den Standard-Quake III-Editor GTK-Radiant exportiert werden.

Levelmodellierung mit dem GTK-Radiant

Der Standard-Quake III-Editor GTK-Radiant bietet alle nötigen Funktionen, um ein komplettes Quake III-Level zu erstellen. Die komplette Funktionalität hier im Einzelnen abzuhandeln ist wenig sinnvoll und würde wohl auch den Rahmen sprengen. Eine Beschreibung des grundlegenden Ablaufs einer Levelmodellierung ist hier sinnvoller.
Die GUI des Editors ist in fünf grundlegende Bereiche aufgeteilt. Ein Abschnitt zeigt eine grundrissartige Struktur des Levels, ein anderer stellt das Level dreidimensional dar. In diesen beiden

Bereichen kann das Level direkt manipuliert werden. Der dritte Abschnitt zeigt die zur Zeit geladenen Texturen, die im Level verwendet werden können. Eingefasst werden diese drei zentralen Abschnitte von einem umfassenden Menü oben und einem kleinen Bereich für textuelle Fehler- und Rückmeldungen unten.

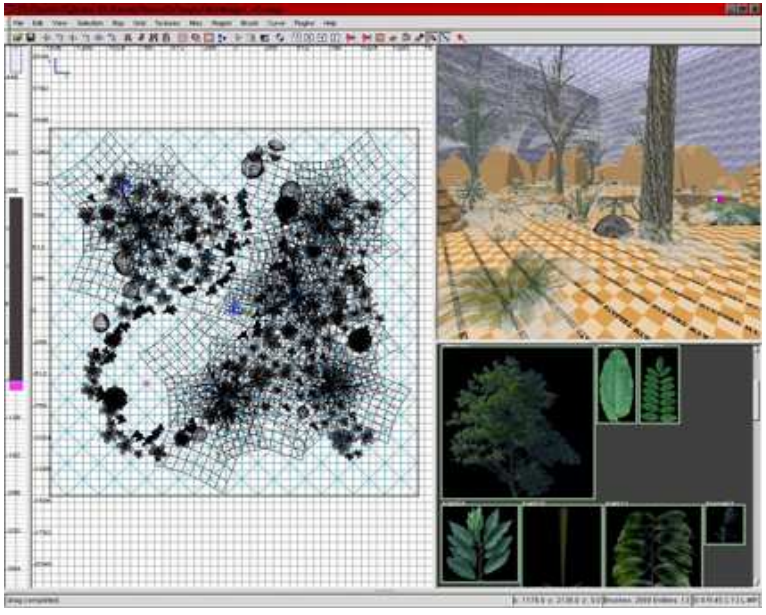

Abbildung 10.2: Arbeitsumgebung in GTK-Radiant

Vereinfacht gesagt, ist das Modellieren eines Quake III-Levels ähnlich dem Spielen mit Bauklötzen. Die Strukturen des Levels werden erstellt, indem man so genannte *Brushes* erzeugt. Diese Brushes sind 3D-Objekte vorher definierter Form, die in das Level eingefügt werden. Diese Brushes sind immer nur von der Außenseite definiert. Durch das Aneinander- und Aufeinandersetzen dieser Brushes erzeugt man die Struktur des Levels.
Will man z.B. einen viereckigen Raum erzeugen, so defieniert man sechs quaderförmige Brushes, die man in ihrer Form und Position so definiert, dass sie Decke, Boden und die vier Wände bilden. Durch das Aneinanderreihen, Positionieren, Drehen und Verformen der Brushes kann so ein kom-

plettes Level geformt werden. Wichtig ist immer, dass das Level nach außen hin abgeschlossen ist. Alle Hohlräume des Levels müssen geschlossen und miteinander verbunden sein. Auch ein Außenlevel unter freiem Himmel ist abgeschlossen. Die Texturen, die den Raum nach oben hin abschliessen, sind hier lediglich mit einem speziellen Shader versehen, der dem Betrachter ein Offensein grafisch vorspielt.
In Ausnahmefällen können auch einzelne Flächen anstelle der Brushes verwendet werden. Diese Flächen nennt man *Meshes*. Sie dürfen nicht zur Abgrenzung des Levels verwendet werden und können also nur in durch Brushes abgegrenzten Hohlräumen eingesetzt werden. Diese Meshes können sinnvoll für Strukturen eingesetzt werden, die quasi keine Tiefe haben, wie z.B. Blätter, Stoffe, Glas, Spinnennetzte oder ähnliches.
Alle Flächen der Brushes und Meshes können mit Texturen versehen werden. Es ist sinnvoll, Objekte kurz nach der Erstellung mit Texturen zu versehen. Es dient der Übersicht und es passiert seltener, dass man einzelne Polygone zu texturieren vergisst.
Ist das Level soweit fertig modelliert und texturiert, können noch bestimmte Positionen innerhalb des Levels mit besonderen Eigenschaften versehen werden. Diese positionsbezogenen, besonderen Eigenschaften nennt man *Entities*. Entities umfassen Spieler-Startpositionen, Lichtquellen, Soundquellen und *Items*, wie z.B. Waffen und Munition. Da wir in unserer Anwendung weder Waffen noch Munition brauchen, werden diese Entities in unserer Anwendung leicht zweckentfremdet verwendet. Da das von uns implementierte Wegsuchesystem der Kreaturen auf Wegpunkten basiert, werden mit Hilfe der Waffen- und Munitions-Entities diese Wegpunkte im Level definiert. Sie werden nur zur Erstellung des Wegenetzes verwandt und nicht dargestellt.

Kompilieren des Levels

Ist das Level nun komplett fertiggestellt, muss es noch in das von Quake III bzw. der Madness Engine verwendete BSP-Format (siehe Abschnitt 3.1) umgewandelt werden. Hierzu muss es kompiliert werden. Der GTK-Radiant bietet zwar einen integrierten Compiler, dieser ist jedoch veraltet und bietet keine genügende Funktionalität und Geschwindigkeit. Wir verwendeten den Compiler q3map2 von Shaderlab. Da es sich um ein Komandozeilenprogramm handelt, ist es sinnvoll zusätzlich ein Tool zu verwenden, das eine GUI für diesen Compiler bereitstellt, um die Benutzung des Programms zu vereinfachen. Die Kompilierung besteht aus drei Schritten: Dem Erstellen des BSP-Baumes, der Datenstruktur in dem das 3D-Modell der Welt abgelegt wird, dem Sichtbarkeitstest, bei dem ermittelt wird welche Bereiche von welchem anderen Punkt im Level einsehbar sind und der Lichtberechnung. Die ersten beiden Teile lassen sich bis auf einige Optimierungs- und Debugoptionen nicht

sehr stark beeinflussen. Sie müssen immer komplett durchlaufen werden. Bei der Lichtberechung gibt es zahlreiche Optionen, die den Gesamteindruck des Levels entscheident beeinflussen können. Als wichtigste Entscheidung muss die Wahl zwischen Gouraud-Shading und Radiosity, zwei grundlegenden Lichtberechungsalgorithmen, getroffen werden. Radiosity dauert wesentlich länger, bietet aber auch entschieden bessere Ergebnisse. Viele weitere Optimierungsoptionen können ebenfalls den optischen Eindruck des Levels verbessern, aber auch den Rechenaufwand entscheidend in die Höhe treiben. Eine genaue Abwägung muss hier von den zur Verfügung stehenden Zeit und Resourcen abhängig gemacht werden. Auch auf schnellen Rechnern kann die Lichtberechnung je nach Einstellung zwischen einigen Minuten und einigen Tagen schwanken.

10.1.2 Probleme beim Modellieren des Dschungel-Levels

Durch das Erstellen immer neuer Levelversionen sind wir irgendwann zu dem Ergebnis gekommen, dass die Quake III/Madness Engine für die Darstellung großer Außenlevels eher schlecht geeignet ist. Eines der grundlegenden Prinzipien der 3D-Engine ist die Nichtdarstellung von Sektoren des Levels, die von der aktuellen Position aus nicht einsehbar sind. Dies verlangt von dem Level jedoch einen Aufbau, der verwinkelt ist und viele kleine Räume und Gänge beinhaltet. Eine solche Levelkonfiguration bietet die Möglichkeit Portale zwischen diesen Sektoren zu definieren.
In den Quake III-typischen klaustrophobischen Innenlevels ist dies optimal zu realisieren. Unsere Anwendung verlangt jedoch nach großen Außenlevels. Auch nach ausgiebigen Versuchen ist es nicht gelungen, in einem großen offenen Areal sinnvoll Sektorportale einzubauen. Das Level besteht so immer zwingend aus einem großen Sektor. Dies ist bei der verwendeten 3D-Engine jedoch absolut ineffizient.
Um die Geschwindigkeit der 3D-Darstellung auf einem akzeptablen Niveau zu halten, muss das Level klein gehalten werden und die Anzahl der Polygone muss soweit reduziert werden, dass die Bildwiederholrate im annehmbaren Bereich bleibt. Dies ist besonders dann deprimierend, wenn sich ein bereits fertiges, optisch sehr ansprechendes Level als zu komplex erweist und ein Prozess des Einsparens von Polygonen beginnen muss und man wieder da landet, wo man zur Mitte des Leveldesigns war.

10.2 Entwicklung eines Charakters

Der Dschungel soll zum Leben erwachen. Dafür müssen neben der typischen Pflanzenwelt, den Flüssen und Höhlen, auch die exotischen Bewohner erstellt werden. Diese unterschiedlichen Wesen zu erschaffen, sie zu entwerfen, dreidimensional zu modellieren, zu animieren und mit Texturen zu belegen, war die Aufgabe der Modellierungsgruppe.

10.2.1 Modellierung

Am Anfang des Prozesses, haben wir uns in Gruppen zusammen gesetzt und mögliche Tiere und Phantasielebewesen erarbeitet. Jede Gruppe hatte die Aufgabe ein oder mehrere Lebewesen zu zeichnen, mit Charaktereigenschaften zu versehen und Verhaltensweisen festzulegen. Aus diesen Ergebnissen suchten wir uns die für unsere Gruppe sinnigsten Charaktere heraus und entwickelten aus zweidimensionalen Zeichnungen mit Unterstützung von dafür geeigneter Software dreidimensionale Modelle am Rechner.

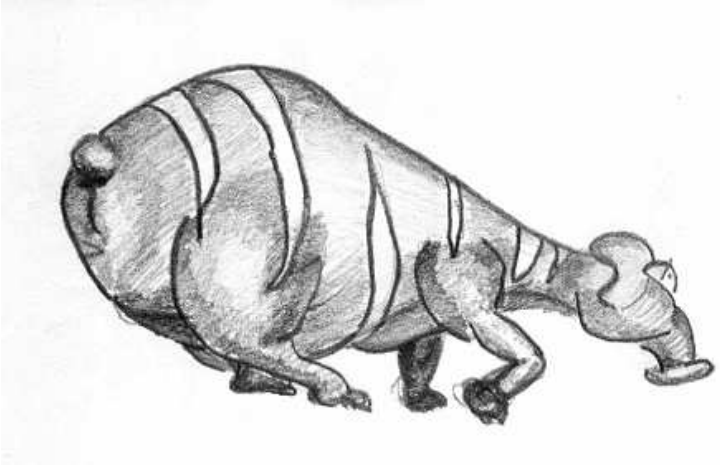

Abbildung 10.3: Zeichnung des Florifanten

Im Laufe der Semester entstanden unterschiedlichste Wesen, die einfache, aber für eine Dschungelwelt sehr typische Eigenschaften besitzen. Es handelt sich dabei um einen „Florifanten“, ein sehr gemütliches Tier, welches eine Mischung aus einem Nasenbär und einem Elefanten ist und einen sehr lieben, schläfrigen aber auch hungrigen Eindruck macht. Außerdem leben in unserer Dschungelwelt eine hyperaktive Rennschnecke, eine langsame, träge Schildkröte, ein gefährlicher Säbelzahntiger und zwei giftige Spinnen. Die Tiere entsprechen jedoch nicht alle der Realität, d.h. die Farben und Eigenarten sind frei gewählt.
Zur dreidimensionalen Darstellung und Animation entschlossen wir uns mit dem frei erhältlichen Programm GMax von discreet, ähnlich der kommerziellen Variante 3D Studio Max, zu arbeiten. Die Grundlage der Erstellung von Objekten mit diesem Programm war ein Einführungskurs, der im zweiten Projektsemester angeboten wurde. Anschließendes Weiterlernen der Programmbedienung war individuell verschieden, entweder über die im Internet verfügbaren Tutorials oder über „Trial and Error“, bzw. das Austesten der Funktionalitäten des Programms.
Dabei gab es zwei verschiedene Herangehensweisen an das Modell: Entweder wurden die Tiere mit Hilfe von *Splines* erstellt oder durch *Low-Polygon-Modellierung*.
Die Modellierung mit Splines baut auf zweidimensionalen Objekten auf (*Shapes*) und formt sie durch Anwendung von Extrudierungsverfahren oder anderen Modifikatoren zu einem dreidimensionalen Objekt. Splines sind allgemeiner gesprochen Linien, die an Knoten aufgehangen sind. An den Knotenpunkten könnten sie durch so genannte *Beziers* verformt werden. Ziel der Modellierung ist es ein Gitternetz der äußeren Körperhülle zu erstellen. Ausgehend von dem Profil der Kreatur, das aus einer durchgezogenen Linie besteht, die, wie in Abbildung 10.4 zu sehen, durch die Bezierknoten in Form gebracht ist, entsteht durch weitere Splines, die an den Knotenstellen mit den Profil verschmolzen werden, ein dreidimensionales Modell.

Da unser Modell eine geschlossene Oberfläche besitzen muss, gilt es ein Paradigma für den Oberflächenmodifikator zu berücksichtigen. So können Oberflächen erst als solche erkannt und dargestellt werden, wenn sie durch weniger als fünf Knoten begrenzt werden.

Bei der Low-Polygon-Modellierung beginnt man mit einem Grundkörper, meist einem Quader, den man sich in beliebig viele benötigte Abschnitte unterteilt und aus dem man z.B. Polygone extrudieren (herausziehen) oder verformen kann (siehe Abbildung 10.7 und 10.8), die Polygone teilen und weiter verarbeiten oder den Körper mit anderen Grundkörpern verbinden kann. Dabei kann man von Anfang an auf eine niedrige Polygonenzahl achten und das Modell sieht noch ziemlich eckig aus. Später erhöht man die Polygonenzahl soweit, dass das Modell einigermaßen rund erscheint. Man kann jedoch durch eine gute Texturierung die Ecken des Modells etwas verbergen.

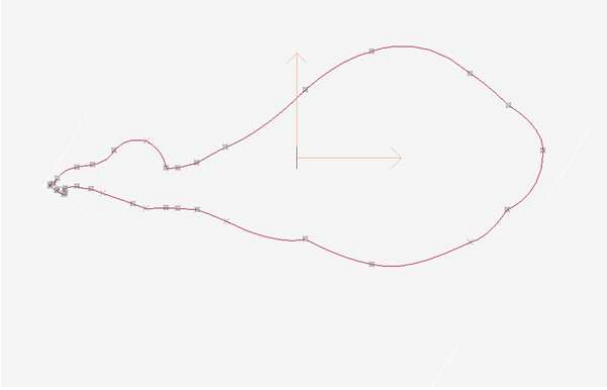

Abbildung 10.4: Florifant im Profil mit Splines

10.2.2 Animation

Damit das starre Modell Bewegungsabläufe vollziehen kann, versehen wir es mit einem virtuellen Skelett. Jeden Knochen müssen wir zusätzlich mit so genannten *Envelopes* versehen, die den Einzugsbereich von Knochenbewegungen innerhalb eines Körpers festlegen. Alles, was sich innerhalb dieses Einzugsbereichs befindet, wird mit dem Knochen mit bewegt.

Animiert wird die Kreatur durch zeitabhängige Transformation des Knochengerüsts, die jeweils durch *Keyframes* begrenzt werden. Nach Erstellung der oben erwähnten Gerüste, können Keyframes gesetzt werden. Es wird immer ein Key gesetzt, wenn die Bewegung an einem Wendepunkt angekommen ist, d.h. an diesem Punkt wird sich die Bewegungsrichtung ändern. Die Wesen werden nach einer von uns festgelegten Bewegungsreihenfolge animiert. Dabei handelt es sich um verschiedene Animationen, die jeweils einen Animationsanfang, eine Animationswiederholung (*Loop*) und ein Animationsende oder -übergang zu einer neuen Animation haben. Die Keys, die den Übergang zur einer neuen Animation herleiten und die Keys, die eine völlig neue Animation einführen, müssen in der anschließend erstellten `animation.cfg`-Datei notiert werden. Die festgelegten Animationen der Lebewesen sind *stehen*, *gehen*, *rennen*, *sitzen*, *essen*, *trinken*, *schlafen*, zwei Animationen die jedem Wesen individuell zugewiesen werden können und *sterben*.

Hier ein kleiner Auszug der `animation.cfg`-Datei:

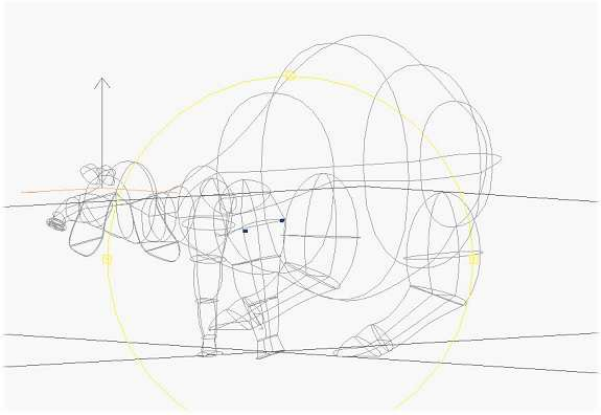

Abbildung 10.5: Florifant mit angezeigten Splines

```
0       29      29      25      //MICADO_ANIMATION_STAND,
30      1       0       0       //MICADO_ANIMATION_WALK_START,
30      149     149     150     //MICADO_ANIMATION_WALK_LOOP,
180     1       0       0       //MICADO_ANIMATION_WALK_TO_RUN,
```

Der Aufbau der Datei wird im späteren Abschnitt über die Exportbedingungen (10.2.4) und den Export noch ausführlicher behandelt.

10.2.3 Texturierung

Anschließend wurden die Modelle texturiert, d.h. die einzelnen Wesen mit einer passenden Farbe und einem für das jeweilige Tier passenden Material (Haut, Fell) versehen. Dazu haben die einzelnen Modellierer im Internet, Filmen oder auch in der freien Natur nach *Skins* (Oberflächen) gesucht. Diese wurden um die Wesen gelegt. Da in der Madness Engine kein globales Licht gesetzt ist, wir dieses aber durch Schattierungen auf den Lebewesen simulieren wollten, haben wir mehrere Lichtquellen in GMax gesetzt und Screenshots von verschiedenen Perspektiven des Modells gemacht. Zusätzlich haben wir während des Sceenshots die Anzahl der Polygone erhöht, damit das Modell durch die detailiertere Textur runder erscheint.Diese Methode fördert nebenbei auch den dreidimensionalen

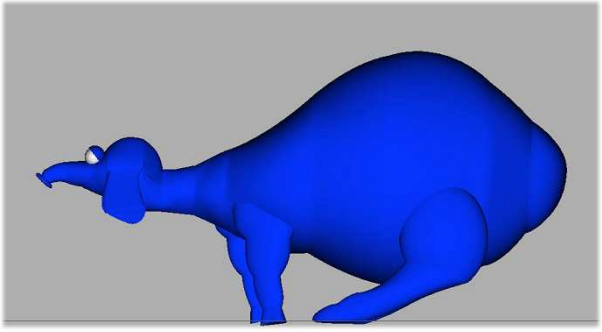

Abbildung 10.6: Fertig modellierter Florifant

Eindruck der Kreaturen. Die Screenshots haben wir in Photoshop mit einem einfarbigen Hintergrund versehen, um damit später in GMax besser arbeiten zu können.

Um die von uns zusammengestellte Bilddatei mit ihren Oberflächeneigenschaften und den Schattierungen passend auf das Modell zu positionieren, nutzen wir das *Mapping*. Dazu wenden wir auf unserem Wesen den *UVW-Map-Modifier* an, der die von uns gewählte Bilddatei planar auf das Modell projiziert. Es gibt verschiedene Projektionsformen (z.B. kugelförmig oder zylinderförmig) die man wählen kann, damit die Textur schon möglichst genau auf dem Modell liegt. Anschließend weist man dem Modell den *Unwrap-Modifier* zu, durch den man ein Polygon-Netz auf der Textur angezeigt bekommt. Dieses Netz ist die zweidimensionale Projektion des Gitternetzes der äußeren Hüllen, so dass es uns möglich ist, den einzelnen Oberflächen ihren genauen Texturausschnitt zuzuweisen.

10.2.4 Export

Die Madness Engine ist Quake III kompatibel, somit ist es uns möglich ein Modell aus 3D Studio Max bzw. GMax zu exportieren. Dabei wurde uns deutlich, dass einige Richtlinien bereits beim Modellieren zu beachten sind.

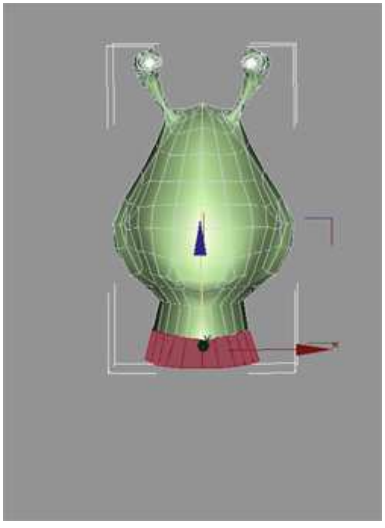

Abbildung 10.7: Schneckenkopf mit Low-Polygon-Modellierung

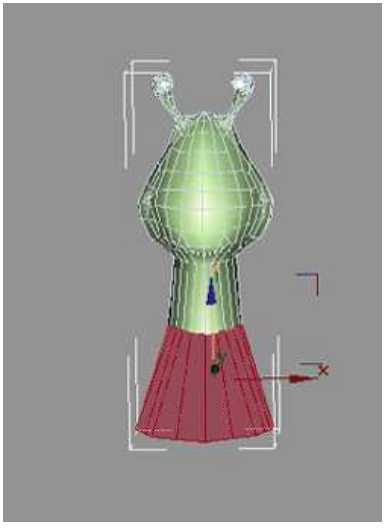

Abbildung 10.8: Extrudieren des Schneckenhalses

Vorraussetzungen für den Export

Ein Quake III-Modell ist in drei Segmente unterteilt. Der untere Teil eines Modells nennt sich `l_leg`, und enthält die Beine des Avatars. Dieser Teil wird über die Tastatur im Spiel bewegt. Der zweite Teil nennt sich `u_torso` und enthält den Torso des Avatars, welcher über die Mausausrichtung und Tastatur gesteuert wird. Im dritten Teil names `h_head` ist der Kopf enthalten. Der Kopf wird ausschließlich über die Maus gesteuert und sieht immer in Richtung Fadenkreuz. Beim Kreieren des Modells ist also auf die Dreiteiligkeit und die Namenskonventionen zu achten. `l_` steht für lower, `u_` für upper und `h_` für head. Eine weitere in Quake III verwendete Namenskonvention ist `w_weapon`. Da wir aber im Dschungel keine Waffen verwenden, haben wir diese vernachlässigt.

Ein andere wichtige Richtlinie beim Modellieren ist die Anzahl der Polygone. Da das Spiel Quake III aus heutiger Sicht stark veraltet ist, ist die Anzahl der Polygone sehr gering. Es dürfen maximal 1.024 Polygone pro Segment verwendet werden. Um dies zu verwirklichen, wurde bei der Entwick-

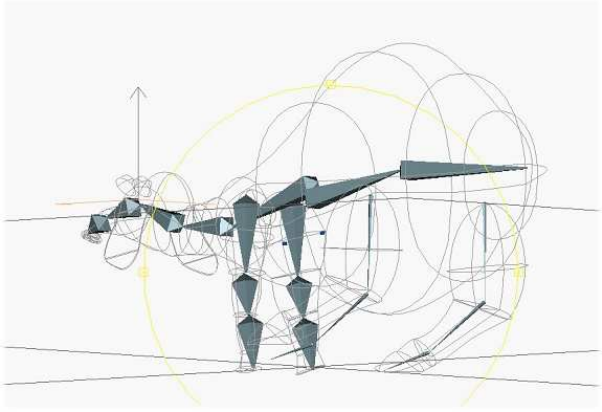

Abbildung 10.9: Florifant mit Knochen

lung der Originalmodelle das Low-Polygon-Modelling verwendet. Das so sehr eckige Modell wurde durch gute Texturen aufgebessert.

Eine weitere Bedingung stellen die so genannten *Tags* dar. Ein Tag ist ein einfaches Dreipunktpolygon mit einem 90°-Winkel. Es gibt vier Tags, wobei jede Geometrie des Modells in Relation zu einem Tag steht. Auch bei den Tags gibt es Namenskonventionen. `tag_floor` liegt auf dem Boden und steht in Relation zu allen anderen Tags und Geometrien, da es die Höhe Null definiert. `tag_torso` stellt die Verbindungsstelle zwischen `l_leg` und `u_torso` dar, sowie `tag_head` die Verbindungsstelle zwischen `h_head` und `u_torso` darstellt. Die Tags müssen mit großer Vorsicht platziert werden, damit das Modell nicht zerbricht. Das vierte Tag nennt sich `tag_weapon`, und bestimmt den Verbindungspunkt zwischen `u_torso` und `w_weapon`. Das Modell steht in Relation zur Größe der Tags. Ist das Tag besonders klein, erscheint das Modell sehr groß in der Madness, und ist das Tag besonders groß, wird das Modell sehr klein. Auch die Ausrichtung der Tags ist wichtig, da die Ecke mit dem spitzesten Winkel immer nach vorne zeigt. Der Avatar sollte also in die gleiche Richtung blicken.

Da uns das Modellieren mit der Dreiteiligkeit große Schwierigkeiten bereitet hat, haben wir das komplette Modell in `l_leg` modelliert. Dies ist möglich, da man unter `l_` mehrere Geometrien modellieren darf, wie zum Beispiel `l_linkesAuge`, `l_rechtesAuge`, etc. Ebenfalls missachtet haben wir die emp-

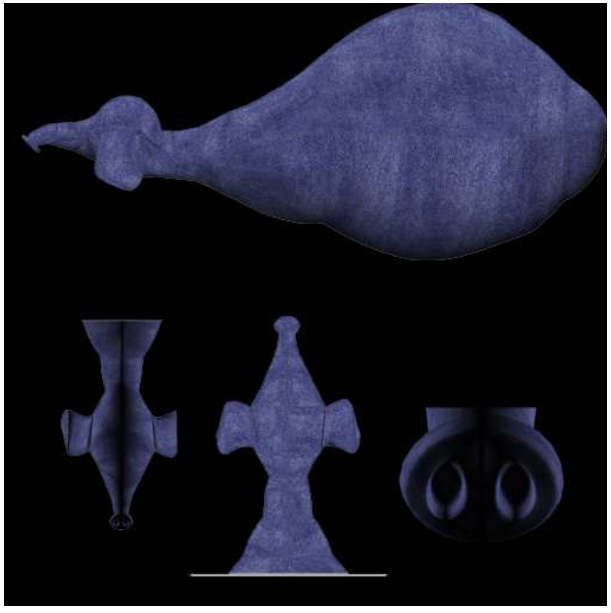

Abbildung 10.10: Textur mit Schattierungen

fohlene Polygonanzahl – in unserem Falle sind es 2.000–5.000. Nun lassen sich unsere Modelle ab einer GeForce-Grafikkarte problemlos bewältigen.

Beim Animieren des Modells ist auf die durch Quake III vorgegebene Animationsreihenfolge zu achten. Zu jedem Quake III-Modell gehört eine `animation.cfg`. Eine solche Datei sieht wie im folgenden auszugsweise dargestellt aus (der komplette Inhalt der Datei findet sich im Anhang auf Seite 149):

```
153     8       8       20              // LEGS_WALKCR
161     12      12      20              // LEGS_WALK
173     11      11      21              // LEGS_RUN
184     10      10      20              // LEGS_BACK
194     10      10      15              // LEGS_SWIM
```

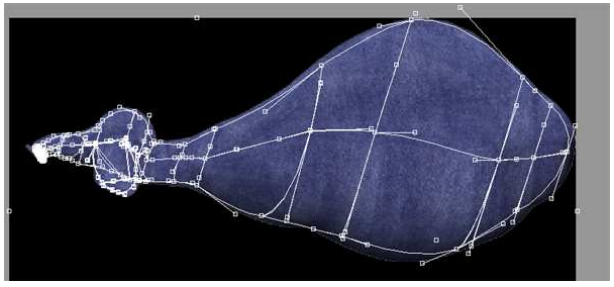

Abbildung 10.11: Positionieren des zweidimensionalen Gitternetzes

Zuerst wird das Geschlecht angegeben, danach werden die einzelnen Animationen definiert. Das geschieht, indem man zuerst das erste Frame, dann die Anzahl der Frames die abgespielt werden sollen, dann die Anzahl der zu wiederholenden Frames und die Abspielgeschwindigkeit in Form von Frames pro Sekunde angibt. 25 Frames entsprechen beispielsweise einer Sekunde Fernsehen nach dem *PAL-Standard*.
Die Reihenfolge der Animation ist durch diese Datei fest vorgegeben. Beim Modellieren ist also darauf zu achten, dass die Animationen in der gleichen Reihenfolge umgesetzt werden.

Wenn das Modell nun modelliert, animiert und texturiert ist, kann es exportiert werden.

Export

Zum Export eines Modells wird bei 3D Studio Max ein Plugin benötigt, um in das md3-Format exportieren zu können. Für GMax gibt es Tempest als Programm zum Export. Das md3-Format steht für Model 3, bei Quake II hieß es md2.

Unter dem Menü Datei, Menüpunkt Export, wird nun das md3-Format gewählt, und die Datei `lower.md3` genannt, wobei dieser Dateiname Pflicht ist. Im nun erscheinenden Dialog (siehe Abbildung 10.13) wird zuerst die Animationslänge angegeben, meistens inklusive der Null. Als nächstes wird die Geometrienamensbeschränkung, nämlich `l_` für alle Geometrien die mit `l_` beginnen, eingegeben, sowie, welche Tags exportiert werden sollen, also `tag_torso`, und zu welchem Tag diese Geometrien relativ sind, also `tag_floor`.

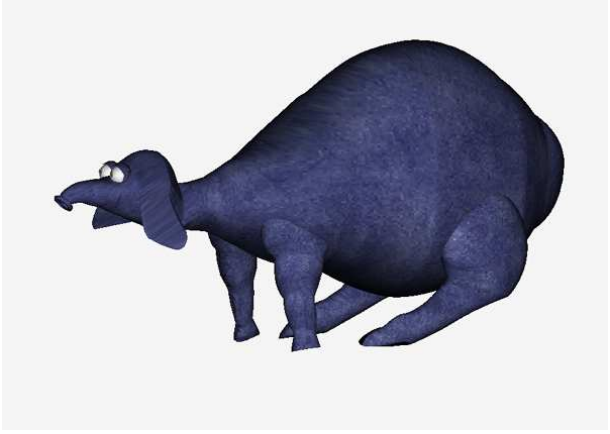

Abbildung 10.12: Fertiger Florifant

Nach dem erfolgreichen Exportieren wird ein weiteres Programm benötigt, nämlich N'Phernos md3 Compile. Mit diesem Programm wird die Datei `lower.md3` geöffnet. Auf der linken Seite sollte nun das Modell in blauen bis roten Tönen zu sehen sein (siehe Abbildung 10.14). Unter dem Modell befindet sich ein Schieber, welcher bei Betätigung die einzelnen Frames durchgeht. Um das Modell Quake III fähig zu machen, müssen nun die Normalen gedreht (siehe Abbildung 10.15), und die Position des Modells leicht verändert werden.

Verwendet man den md3-Compiler nicht, so gibt es Fehlermeldungen beim Starten der Madness über nicht gefundene Shader. Leider war es uns bisher nicht möglich, die md3-Datei von Hand zu editieren, um den Fehler zu finden, bzw. den Fehler im Enginecode aufzuspüren.

Die so umgewandelte md3-Datei muss nun in einen Ordner geschoben werden. Danach befindet sicht die Datei an folgender Stelle:
... /madness/idSoftware/models/players/<Name des Modells>/lower.md3

Damit das Modell nun auch mit der dazu gehörigen Textur dargestellt wird, muss eine weitere Datei erstellt werden. Bei Quake III gibt es für jeden der drei Teile eines Modells eine Datei, die die Pfade zu jeweils einer Textur pro Geometrie enthält. Eine solche Datei sieht folgendermaßen aus:

```
l_legs,models/players/sarge/band.tga
```

Abbildung 10.13: Export Dialog

`tag_torso`,

Das nun fertig erstellte Modell kann mit einem der Madness-Programme getestet werden. Dazu kann `3rdperson` oder `kitest01` unter `engine-apps`, bzw. `player` unter `samples` gestartet werden. Es empfiehlt sich das Modell nach einem ersten Versuch mit `3rdperson` zu testen, da dort die Möglichkeit, um das eigene Modell herumzuschweben, geboten wird.

Da uns die durch Quake III für die Beine vordefinierten Animationen nicht genügten, haben wir weitere Animationen hinzugefügt, und dabei ein neues Animationskonzept eingeführt. Der neue Code der `animation.cfg` sieht wie im folgenden auszugsweise dargestellt aus (der komplette Code findet sich im Anhang auf Seite 150):

```
0       29      29       25      // MICADO_ANIMATION_STAND,
30      1        0         0     // MICADO_ANIMATION_WALK_START,
30      149    149      150      // MICADO_ANIMATION_WALK_LOOP,
180     1        0         0     // MICADO_ANIMATION_WALK_TO_RUN,
180     59      59       150     // MICADO_ANIMATION_RUN_LOOP,
240     1        0         0     // MICADO_ANIMATION_RUN_TO_WALK,
240     1        0         0     // MICADO_ANIMATION_WALK_STOP,
```

Wir haben insofern ein neues Animationskonzept eingeführt, dass nun jede Handlung eine Animation zum Loop hin, den Loop selbst, und aus dem Loop heraus besitzt, ausgenommen das Sterben.

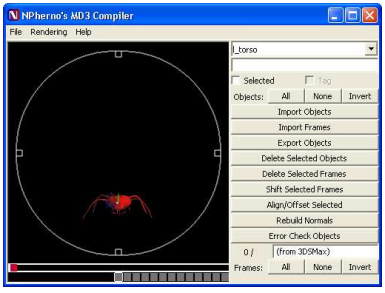

Abbildung 10.14: N'Phernos md3 Compiler

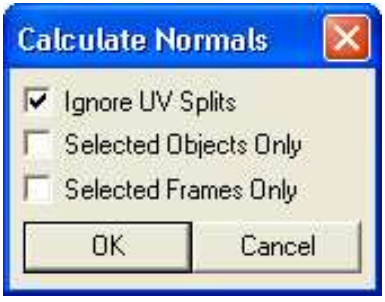

Abbildung 10.15: Normalen drehen

Dadurch werden die Übergänge zwischen einzelnen Handlungen besser dargestellt, und eine länger andauernde Handlung muss nicht immer wieder das initiale Stehen enthalten.

— KAPITEL 11 —

Konfigurations- und Netzwerktools

Unser Cave ist nach der Übersetzung des Quellcodes und dem Aufbau und Anschluss aller seiner physischen Komponenten noch nicht sofort einsatzbereit. Die Rechner der sechs Seiten sind zuvor so einzustellen, dass sie konsistent zusammenarbeiten[1], auch sind die Daten[2] auf ihnen stets auf dem aktuellen Stand zu halten. Bei einer Inbetriebnahme muss die Cave-Anwendung auf allen Seiten gleichzeitig hochgefahren werden, zudem muss festgelegt werden, mit welcher der modellierten Landschaften die virtuelle Exkursion stattfinden soll und an welchen Stellen welche Tiere mit welchen Texturvariationen, Sounds und Verhaltensparametern zu inkarnieren sind. In diesem Kapitel beschäftigen wir uns daher mit den Hilfsmitteln, die wir zur Lösung dieser Probleme implementiert haben. In Abschnitt 11.3 gehen wir kurz auf unsere Modifikation des Netzwerkcodes der Madness Engine ein, welche die Wahrscheinlichkeit eines Verlustes von UDP-Paketen aufgrund einer zeitweisen Überlastung eines Rechners des Master/Slave-Verbunds reduziert.

11.1 Die Weltkonfiguration

Da es wenig zweckmäßig und eher kontraproduktiv ist, Informationen über die zu ladende Welt samt Kreaturen als hartcodierten Startup-Code fest im Quelltext der Cave-Anwendung anzugeben,

[1] z.B. bezüglich des individuellen Bildausschnitts, der Sound- und Lautsprecherkonfiguration, etc.
[2] Wie Texturen, Sounds und 3D-Modelle.

war eine einheitliche Schnittstelle zur Definition einer so genannten Weltkonfiguration zu schaffen – schließlich wollten die einzelnen Projektmitglieder zwecks Tests ihrer Arbeitsergebnisse aus KI-Programmierung, Modellierung, etc. zwischen unterschiedlichsten Konfigurationen wechseln und diese schnell modifizieren können. Alle zu einer solchen Konfiguration gehörenden Informationen werden für unseren Cave textuell in einer Weltkonfigurationsdatei zusammengefasst, deren Dateisystempfad dann der Cave-Anwendung des Master-Rechners als Kommandozeilenparameter übergeben wird[3]. Die Weltkonfigurationen liegen in einer einfachen Syntax vor und können so leicht bearbeitet, ausgewählt und zwischen den Projektteilnehmern ausgetauscht werden, ohne dass eine Modifikation oder ein Neukompilieren der Anwendung erforderlich wäre. Pro Datei ist die für die Konfiguration zu verwendende BSP-Datei mit der Weltgeometrie anzugeben, die dazu passende Pfaddatei für die Kreaturnavigation sowie für jede zu erzeugende Kreatur der Name der für sie zuständigen C++-Kreaturklasse, der Pfad zur MD3-Datei mit dem animierten 3D-Modell, der Name der gewünschten Texturvariation, der Startpunkt in der Welt sowie der Pfad zur Kreaturkonfigurationsdatei mit den für sie zu benutzenden KI-Einstellungen und Sounds. Optional kann für jede Kreatur ein interner Name vergeben werden, mit dem sie in den Debugausgaben der Engine identifiziert wird. Für die zu ladende Welt können zudem die gewünschten Texturen für die *Sky Box* angegeben werden. Letztere ist für die Darstellung des Himmels zuständig und entspricht einem Einheitswürfel, der die gerenderte Welt umgibt und dessen sechs Seitentexturen so erstellt bzw. generiert wurden, dass die ursprüngliche Würfelform des Himmelskörpers für den Betrachter nicht offensichtlich wird[4].

Die technischen Einstellungen, die den konkreten Cave-Aufbau betreffen, wie beispielsweise die Aufstellwinkel der Projektionsflächen und Lautsprecher, die Bildschirmauflösung in Pixel, eine etwaige Kamerarotation um 90° oder das Auflösungslimit für Texturen, sind nicht in der Weltkonfiguration vorzunehmen, sondern in verschiedenen Umgebungsvariablen individuell für jeden Rechner[5], siehe dazu auch Abschnitt 11.2 und Anhang Seite 158. Da die Weltkonfiguration den Zweck hat, alle für ein Dschungelszenario relevanten Einstellungen zusammenzufassen, soll sie völlig unabhängig vom vorliegenden Cave-Aufbau sein, folglich für sämtliche Aufbausituationen funktionieren.

[3] Die Datei muss lediglich auf dem Master liegen, die Slaves benötigen sie nicht.

[4] Vergleichbar mit den sechs Texturen beim *Cube Environment Mapping*, einem Verfahren mit dem reflektierende Oberflächen simuliert werden können.

[5] Ist der Cave einmal aufgebaut, werden sie sich in der Regel nicht mehr ändern, so dass hier nicht häufig zwischen unterschiedlichen Werten umgeschaltet werden muss. Ein schneller Wechsel der virtuellen Welt sollte hingegen möglich sein.

Die von uns implementierte Klasse `micado_worldconf` beinhaltet den Parser für die Weltkonfiguration. Nach dem Laden einer solchen führt sie alle Operationen und Objektinstantiierungen in der Reihenfolge durch, wie sie für das Hochfahren des spezifizierten Szenarios nötig sind. Beim Erstellen der Weltkonfiguration muss sich deren Autor somit an keine bestimmte Reihenfolge der Einstellungen halten, wohl jedoch an die vorgegebene Syntax. Wird diese verletzt oder treten beim Laden der Objekte Fehler auf, gibt das System unter Angabe der Nummer der betroffenen Zeile und Spalte der Datei eine detaillierte Fehlermeldung aus und bricht ggf. die Ausführung ab, so dass der Benutzer eine Korrektur leicht vornehmen kann. Für eine Beispielkonfiguration siehe Anhang Seite 154.

11.2 Das `remad`-Fernsteuerungsprogramm

Um den Cave hochzufahren, ist zuerst auf allen Slaves das Programm für den Slave-Betriebsmodus zu starten, welches in einem Standby-Zustand auf das Aktivwerden des Masters wartet, dessen Anwendung zuletzt gestartet wird. Beim Start sucht der Master nach aktiven Slaves und beginnt nach dem Laden der gewünschten Weltkonfiguration mit dem Ausführen der Kreatur-KI sowie dem verteilten Rendern der Grafik. Da es unpraktisch ist, die Programme auf allen Rechnern jedes Mal per Hand zu starten, haben wir das einfache Fernsteuerungstool `remad` erstellt. Dieses wird auf jedem Rechner einmalig im Servermodus gestartet (bspw. automatisch beim Booten als Daemonprozess) und bekommt dabei den Pfad zu zwei ausführbaren Programmen als Kommandozeilenparameter übergeben. Über das Netzwerk können ihm dann verschiedene Befehle geschickt werden, um z.B. eine der beiden Anwendungen als Kindprozess mit mitgegebenen Parametern zu starten, sie durch den Versand der Signale `SIGINT`, `SIGTERM` oder `SIGKILL` wieder zu beenden, oder eine bestimmte für den Kindprozess relevante Umgebungsvariable auszulesen, bzw. auf einen bestimmten Wert zu setzen. Die Beschränkung auf zwei ferngesteuert ausführbare Anwendungen und deren Festlegung bei Programmstart erfolgt aus Sicherheitsgründen.

Im Clientmodus können mit `remad` die oben genannten Befehle an einen bestimmten Rechner gesendet werden. Da es sich um ein reines Kommandozeilentool handelt, eignet es sich bestens für einen Einsatz in Shell-Skripten, so dass sehr einfach Skripten zum Hoch- und Herunterfahren des gesamten Caves erstellt werden können. Das entfernte Starten von Programmen kann zwar auch mit Hilfe von SSH durchgeführt werden, gerade in Bezug auf das ferngesteuerte Setzen von Umgebungsvariablen ist `remad` jedoch etwas eleganter in der Handhabung, zudem verhindert es von Haus aus, dass auf einem Rechner zu einem Zeitpunkt mehr als ein Prozess mit der Cave-Anwendung gestartet wird.

Um sämtliche Programme und Dateien auf allen Rechnern des Verbunds konsistent zu halten resp. zu aktualisieren, bietet sich das von uns verwendete Open Source Tool `rsync` an. Es ist dabei ratsam, Änderungen nur auf dem Master vorzunehmen und anschließend auf alle Slaves zu spiegeln.

11.3 Netzwerk Receiver-Thread

Der von Martin Faust implementierte Netzwerkcode der Madness Engine behandelt die Möglichkeit eines Verlusts von UDP-Paketen nach dem Straußenprinzip: Er ignoriert diesen Fall schlicht und ergreifend. Da die Netzwerktopologie des Caves mit sechs Rechnern an einem Hub oder Switch (vgl. dazu Abbildung 3.2 auf Seite 16) äußerst einfach ausfällt, sind bei intakter Hardware Paketverluste in der Praxis auch eher eine Seltenheit. Paketkollisionen kommen aufgrund des geregelten Protokolls zwischen Master und Slaves ebenfalls sehr selten vor und werden in der Regel bereits auf der Ebene des Ethernetprotokolls erkannt und behandelt.

Während der Verlust eines Netzwerkpakets mit beispielsweise der neuen Position und Ausrichtung der Kamera relativ unkritisch ist (da im nächsten Frame ein aktualisiertes gesendet wird und die Welt somit nur für einen kurzen Zeitraum Inkonsistenzen zwischen den Leinwänden aufweist), fällt der Verlust anderer Pakete deutlich ungünstiger aus: Kommt ein Paket, welches die Slaves auffordert, ein Objekt einer bestimmten Klasse mit bestimmten Parametern zu erzeugen (beispielsweise eine Kreatur während des Bootens der Weltkonfiguration oder einen abzuspielenden Sound im laufenden Betrieb), bei einem Rechner nicht an, so weiß dieser über das betreffende Objekt nicht bescheid, wenn er später ein Paket zur Aktualisierung desselben bekommt. Die Kreatur wird dann auf der jeweiligen Leinwandseite nicht angezeigt, bzw. der Sound nicht abgespielt. Deutlich kritischer ist der Verlust bestimmter Pakete, welche zur Synchronisation der gerenderten Frames dienen: Geht nur eines von ihnen verloren, so friert der gesamte Cave ein. Dies kam während unserer ersten Testaufbauten durchaus mehr als einmal vor und ist natürlich überaus suboptimal.
Eine saubere Lösung wäre hier die Reimplementierung des Netzwerkprotokolls der Madness, beispielsweise unter Verwendung von TCP/IP. Bei TCP müssten wir allerdings auf die Multicastfunktionalität verzichten und mit größeren Latenzen zwischen den Leinwänden rechnen. Gerade in Bezug auf die in Abschnitt 1.2 erwähnte Motion Sickness erscheint eine schlechtere zeitliche Synchronität der projizierten Frames nur wenig erstrebenswert, gleiches gilt für das Soundsystem aus den in Sektion 9.2 besprochenen Gründen. Einmal mehr entschied sich unser Projekt für eine erste schnelle „Lösung“ in Form eines Workarounds und sah zunächst von einer kompletten Reimplementierung

ab[6]: Die beobachteten Paketverluste waren stets auf einen Überlauf des vom Betriebssystem bereitgestellten Socket-Puffers zurückzuführen[7], weil die Cave-Anwendung die ankommenden Pakete nicht so schnell bearbeiten konnte, wie sie am UDP-Socket eintrafen. Dies lag darin begründet, dass anwendungsseitig das Rendern der Grafik sowie das Abholen und Bearbeiten der Netzwerkpakete abwechselnd von einem einzigen Thread vorgenommen wurde, der neue Pakte nur dann abholte, wenn er gerade nichts anderes zu tun hatte.

Das Workaround bestand daher aus der Einführung eines dedizierten Receiver-Threads, welcher nichts anderes tut, als Pakete vom UDP-Socket abzuholen und in einer verketteten Liste zwischenzuspeichern, aus welcher der Hauptthread der Cave-Anwendung sie nacheinander abholt – also in der Schaffung eines sekundären Paketpufferpools innerhalb der Engine. Unsere Implementierung versucht hierbei den durch das Memory Management entstehenden Overhead möglichst klein zu halten, indem sie die Allokation und Freigabe von Heapspeicher auf ein Minimum reduziert. Das Workaround behebt zwar nicht die Auswirkungen eines Paketverlustes, da es nur die Wahrscheinlichkeit eines solchen reduziert, dennoch kann sich das Ergebnis sehen lassen: Während des gesamten Projekttages blieb der Cave nicht ein einziges Mal aufgrund eines Paketverlustes stehen. Eine Weiterentwicklung der Madness bietet sich jedoch nach wie vor in einer Realisierung der sauberen Lösung an – der paketverlusttoleranten Reimplementierung des Netzwerkcodes.

[6] u.a. aufgrund berechtigter Bedenken, kurz vor einer wichtigen Deadline, bzw. dem Projekttag, möglicherweise mehr Fehler einzubauen, als am Ende zu beheben.

[7] Ist dieser Puffer voll, gehen alle weiteren eintreffenden Pakete verloren.

— TEIL III —

Projektverlauf

— KAPITEL 12 —

Historischer Abriss über den Verlauf des Projekts

Projektstart

Am 18.10.2002 fanden sich ca. 60 Studenten zusammen, um gemeinsam das Projekt MiCa unter der Leitung von Wilhelm Bruns zu bilden. Da diese Anzahl zu groß für ein studentisches Projekt war, den Wünschen der Anwesenden aber nachgekommen werden wollte, beschloss man drei Teilprojekte zu bilden. Unter der Leitung von Dieter Müller, Daniel Cermak-Sassenrath und Bernd Robben entstanden MiCarpet, MiCasa und MiCaDo. Letzteres wurde der Name unseres Projektes, den wir bereits am gleichen Nachmittag beschlossen hatten. Er repräsentiert unsere Motivation etwas zu tun, ohne unsere ursprüngliche Zugehörigkeit zu verleugnen. Sesshaft wurden wir alsbald im GW2 B1810, der über die gesamte Dauer von vier Semestern Zentrum unserer Arbeit sein sollte.

Schließlich begann unser erstes Plenum. Diese Form der wöchentlichen Kommunikation sollten wir über die gesamte Projektzeit fortsetzen, wobei unter uns zu diesem Zeitpunkt die Wenigsten genau wussten, wie ein solches Plenum abzulaufen habe. Großes Glück unserer Gruppe war, dass sich viele schon zuvor untereinander kennen gelernt hatten, sei es durch Übungsgruppen im Grundstudium oder durch Kneipenabende, die wir in dem vorherigen Sommer des Öfteren abgehalten hatten. Völlige Bekanntschaft untereinander herrschte aber gewiss noch nicht. Nach einer Vorstellungsrunde wandte man sich den ersten organisatorischen Problemstellungen zu, die ein jedes Projekt auf sich

nehmen muss: Wie stellen wir die Einbruchssicherheit im Projektraum sicher? Wie viele PCs kriegen wir wo? Wie sieht es mit einer Kaffeemaschine und einem Wasserkocher aus? Brauchen wir eine Projektkasse? Wie läuft die Kommunikation untereinander? Wir entschieden uns für einen Schlüsselsafe und einen Mailverteiler plus Forum. Lange konnten wir uns mit diesen Fragen gewiss nicht aufhalten, da uns aufgetragen worden war, binnen eines Monats einen ersten Prototypen herzustellen.

Im zweiten Plenum ging es dann auch schon schneller voran. Das Moderations- und Protokollierungsproblem löste man durch die Reihenfolge des Alphabets. Scheinkriterien wurden festgesetzt. Eine Gruppe von vier Leuten wurde beauftragt, ein Führungsteam zu bilden, das die wöchentliche Planung des Plenums und andere organisatorische Aufgaben zu übernehmen hatte. Jeder sollte diesem Team über die Projektdauer hier und da mal angehören und so einigten wir uns zunächst auf einen kontinuierlichen, maximalen Verbleib von zwei Monaten pro Mitglied, wobei beim Wechseln nie das gesamte Team den Platz räumen sollte. Die XAkte, welche die Archvierung der Protokolle übernahm, sollte unser Freund und Feind zugleich werden.

Realisierung des Prototyps

All diese Punkte lieferten uns aber nicht den Prototypen, für den uns eine vierwöchige Frist auferlegt worden war. Es wurde ein ordentliches Brainstromming auf den Nachmittag verlegt, das erste Produktionsskizzen mit Abmessungen lieferte. Produktionsflächen wurden provisorisch durch einige Bettlaken aus einem hauseigenen Bestand geliefert. Dennoch waren Treffpunkte innerhalb der Woche in den nächsten Wochen keine Seltenheit. So wurde gebügelt und gehämmert und zwischendurch noch ein neues Mitglied aufgenommen. Da wir uns darauf verständigt hatten, die Madness Engine von Martin Faust zu benutzen, die uns keinerlei Beschränkung in der Projektionsflächenanzahl bot, wollten wir unseren Prototyp diese Grenzen ebenfalls nicht unterwerfen. Schon hier zeigte sich ein Geist der Vereinfachung, der ein Wegbegleiter über die Projektzeit bleiben sollte. So bauten wir Module, die aus einem Bettlaken bestanden, an dem an der oberen und unteren Kante jeweils eine Holzlatte befestigt waren, und hingen sie dann an die Decke des artec-Labors, das allen drei Projekten zugänglich war. Schnell war uns klar, dass wir beim Endprodukt einen sechsseitigen Cave bauen wollten, da wir enorme Vorteile in dem vergrößerten Panorama, gegenüber einem vierseitigem Cave, sahen. Drei Laken, mit denen wir verschieden Blickwinkel getestet hatten, legten dies nahe. Einem Projektmitglied, dem einige Wochen zuvor eine Trittmatte übergeben wurde, überraschte zum Präsentationstag schon mit einer lauffähigen Navigation im Cave. Obendrein stellte er eine Entwicklung namens MadBird vor. Dabei handelte es sich um eine Art Vogelflug-Navigationssystem,

bei der die vertikale Position der Arme über Schnürverbindungen zu einem Fischertechnik-Modell, das an eine SPS (speicherprogrammierbare Steuerung) gekoppelt war, ausgelesen werden konnte.

Quo Vadis?

Unser Prototyp war damit fertig. Nun galt es eine Richtschnur für die weitere Projektarbeit zu entwickeln. Da die bisherigen Arbeiten sich in dem Bau des Prototypen sowie in organisatorischen Aufgaben ergeben hatten, erachteten wir es als sinnig, uns gegenseitig tieferen Einblick in verschiedene Bereiche, zu geben, um nach Weihnachten eine grundlegende Vorstellung von den Möglichkeiten, die im Cave realisierbar wären, zu haben und anschließend ein Projektziel festzusetzen. Neben den in Kleingruppen ausgearbeiteten Referaten zur Bildpsychologie, dem Quake III Editor, Sound, Virtual Reality, Cave Planung, Wearables und Sensorik verfolgten wir praxisorientierte Arbeiten. So versuchten wir uns in den Untergruppen Sensorik und Engine in der Anbindung des MadBird an die Engine und der Entwicklung eines neuen MadBird-Modells. Weitere Ausblicke über die Realisierungsmöglichkeiten im Cave bekamen wir durch die 10:00-Uhr-Veranstaltungen von Prof. Bruns und einem externen Pneumatik-Workshop bei Herrn Gathmann. Das erste Semester zeichnete sich durch ein sehr sozialverträgliches, gemeinsames Arbeiten ohne wirkliches Ziel aus. Durch kleinere Fehlplanungen und der allgemeinen Unerfahrenheit der Gruppe, kam es zu kleineren Terminverschiebungen, allerdings wurde unsere Moral nie untergraben. Zwei von uns hatten sogar schon in Abstimmung mit dem Projekt eine einwöchige Auslandreise nach Dänemark organisiert, die zur endgültigen Zielfindung und Stärkung der Truppe genutzt werden sollte.

Zielfindung in Dänemark

Nachdem wir uns am Ende des ersten Semesters zum ersten Mal gegenseitig benotet hatten, traten wir im März 2003 die Fahrt nach Langeland an. Abgesehen von einem eintägigen Ausflug nach Aarhus gab es dort kein wirkliches Programm. So nutzten wir den ersten Abend zunächst einmal für eine ausgiebige, für den einen oder anderen zu ausgiebige Feier, um am darauffolgenden Morgen frisch in die Zielfindung zu starten. Diese begann mit einer Runde. Dies meint, jeder muss sich zu einem gewissen Gegenstand äußern oder eine Entscheidung begründen und hat den Zweck, den sprachfaulen Menschen zur Meinungsäußerung zu zwingen, damit ein vollständiges Bild der momentanen Gemütslage des Projektes entsteht. Dieses langwierige Prozedere sollte sich noch etliche Male in den fortlaufenden Wochen wiederholen. So erklärte jeder einzelne seine Wünsche und Vorstellungen zu der weiteren Projektarbeit. Daraufhin teilten wir uns in Gruppen auf und klügelten

eifrig mehr oder weniger ambitionierte Projektziele aus, ohne eine konkrete Realisierungsmöglichkeit mit einzubeziehen. Diese stellten wir am nächsten Tag vor. Die Liste bot eine weite Bandbreite von simulationslastigen Ideen wie Tauchen, Rollstuhl fahren oder ein Lufttaxi, Ansätzen therapeutischer und edukativer Art, über Kunstinstallationsideen, verschiedenen Werkzeugen zur biometrischen Datenerfassung, hin zu Spielereien wie ein Balance- oder Rollenspielen. Hier alle Vorschläge aufzuzählen wäre müßig. Der nächste Tag kann rückblickend nicht unterschätzt werden. Wir brachen nach Aarhus auf, um uns im Aarhus University Centre for Pervasive Computing vordergründig durch verschiedene virtuelle Anwendungen, wie das dortige 3D-Kino, Bluescreen Studio oder den Lego-Robotern zu unterhalten. Speziell die Lego-Roboter machten durch die heitere Präsentation von Herrn Ole Caprani im Laboratory for Robot Technology und durch ihr seltsames Eigenleben großen Eindruck auf das Projekt. In welcher Weise dies die Zielfindung beeinflusste, wird noch geklärt werden, galt es doch in den nächsten Tagen die gesammelten Ideen zu konkretisieren und sich in Streitigkeiten zu ergeben.

Diese ergaben sich aus zwei Richtungen, die sich als mögliches Ziel herauskristallisierten und von verschiedenen Menschen präferiert wurden. Zum einen gab es den Kunst- und Emotionsansatz, der mit Hilfe von biometrischen Daten und suggestiver Musikuntermalung eine Art Installation zum Ziel hatte, zum anderen vertraten eine ähnlich große Gruppe die Auffassung, sich einer zu dem Zeitpunkt konkretisierteren Idee, dem Balance-Spiel, dass auf einen Kurzfilm zurückgeht, zu stellen. Dabei galt es durch Gewichtsverlagerung auf einer Plattform eine Kiste gegen Gegenspieler zu ergattern. Einer von uns hingegen hatte auf der Rückfahrt von Aarhus Gedanken über eine Dschungelwelt, in der Tiere umherstreunern und untereinander, sowie mit dem Cave-Benutzer in Interaktion treten, verloren. Diese Idee führt sicher auf die Legoroboter und ihrem faszinierendem, interpretierbarem Verhalten zurück. Während sich die zwei großen Gruppen weiter um ihre jeweilige Idee rangen, kam dieser nie ganz von seiner Idee ab, um Befürworter einer der anderen Ideen zu werden. So kam es am Ende der Woche zu einer Abstimmung, die Anhänger des Kunst- und Emotionsansatzes gewannen und eine knappe Hälfte des Projekts relativ deprimiert die Heimreise antreten ließ. Um kein falsches Bild entstehen zu lassen, muss hier gesagt werden, dass es viele, lichte Momente innerhalb der Woche gab und das allabendliche Feiern auch großen Spaß bereitet hatte, doch plagte viele von uns eine gewisse Perspektivlosigkeit und das Gefühl kein wirkliches Ziel gefunden zu haben. Mit gutem Gewissen kann man in der näheren Folgezeit von dem motivationstechnischen Tiefpunkt des Projektes reden.

Konkretisierung einer Idee

Wir hatten uns verabredet, in drei Teilgruppen dem Thema eine definitive Richtung zu geben und in möglichst kurzer Zeit über diese abzustimmen. Dazu trafen sich diese Gruppen innerhalb der Ferien und arbeiteten sie für das gesamte Projekt aus. Darunter war zum einen eine Idee, die zunächst sich einmal X-Men nannte und auf eine Sequenz aus dem gleichnamigen Film von Brian Singer basierte. Dabei ging es um das Austauschen und die Darstellung von Gedanken verschiedener Individuen in einem bizarren Raum. Die zweite Idee war ein lichtschrankenbasierter Laser-Cave. Die dritte Idee rang sich dann schließlich mit der X-Men Idee um das Projektziel. Das eine Projektmitglied hatte es geschafft seine Dschungelwelt in ein Kunst- und Emotionsgewand zu hüllen und fand dabei gerade bei den deprimierten Anhängern der Balanceidee regen Zuspruch. Schließlich kam es zur Abstimmung und die Dschungelidee gewann. Die Implementierung konnte beginnen und in den Bereichen Sensorik, Modellierung, Grafik, Künstliche Intelligenz, Sound fanden alle Projektteilnehmer einen Bereich in dem sie anfangen konnten zu arbeiten. Nun reaktivierten wir das Orgateam, das wir innerhalb des ersten Semesters irgendwann einmal begraben hatten und strafften unsere Arbeitsweise. Dazu teilten wir uns in die besagten Bereiche auf und steckten uns im Drei-Wochen-Rhythmus Ziele, die es innerhalb der Untergruppen zu erreichen galt.

Realisierung der Idee

Die Sensorikgruppe musste sich eine Navigation durch den Dschungel überlegen und Wege finden, Biometrische Daten zu erfassen, die Grafik-und-KI-Gruppe hatte sich in die Programmierung der Engine und die Grundlagen der Künstlichen Intelligenz zu knien, die Modellierungsgruppe hatte sich in das Programm GMax[1] einzuarbeiten und Vorschläge aus dem Projekt sowie eigene in die Tat umzusetzen. Die Sound-Gruppe hatte einen 3D-Sound zu implementieren. So ging es über das gesamte zweite Semester mal schneller mal langsamer voran. Erste Vorstellungen von unserem späteren Produkt, bekamen wir als ein erstes selbstgebautes Level präsentiert wurde und eine zuvor gezeichnete Kreatur namens Florifant es in die Welt schaffte. Die Sensorik überlegte sich, nach einigen Startschwierigkeiten, also Entwicklungen die schlussendlich immer wieder scheiterten wie zum Beispiel das Intersense-Positionserkennungsystem, der Lügendetektor oder das Augmented-Reality-Tool-Kit, eine Art Floß, auf dem man durch die Welt gleitet. Die Grafik-und-KI-Leute heckten ein Wegewahlsystem aus und arbeiteten an der Künstlichen Intelligenz der Kreaturen. Die Soundgruppe nahm fälschlicherweise an, dass ihr 3D-Sound bereits vollends funktionierte und teilte sich schon im lau-

[1] siehe http://www.discreet.com/

fenden Semester zur Verstärkung der übrigen Gruppen auf. Dennoch verabschiedeten wir uns am Ende des zweiten Semesters nicht in die Ferien, sondern bauten unseren endgültigen Cave. Dieses Unterfangen hatte eine Cave-Bau-Gruppe, die wir innerhalb des Semesters ins Leben gerufen hatten, zwar von langer Hand geplant, doch mussten wir wieder einmal zum Mittel der Vereinfachung greifen. Die vorgestellte Konstruktion überspannte sowohl unsere finanziellen Mittel als auch den uns zur Verfügung stehenden Raum. Nach langer Diskussion kam die Idee auf, ein Partyzelt als Grundgerüst für den Cave zu nutzen. Zwar bestand noch arger Zweifel innerhalb des Projekts, ob es dieses überhaupt in einer sechsseitigen Ausführung gebe, doch war das Glück auf unsere Seite und wir fanden Besagtes noch am selben Nachmittag zum kleinen Preis.

So bestand die Aufgabe in den drei Wochen, die wir für den Bau des Caves veranschlagt hatten, darin, Beamer-Ständer zu bauen, Stoffe zu testen und zusammenzunähen, das Gerüst auf eine Gesamthöhe von 2,70 m zu bringen und schließlich den fertigen Cave mit allen Seiten in Betrieb zu nehmen. Völlig zufrieden waren wir allerdings nicht, da der Test etwas mager ausfiel und von der eigentlichen Anwendung bisher noch zu wenig zu sehen war. Um das Arbeitstempo aufrecht zu erhalten, musste sich jede Gruppe Ziele stecken, die es bis zum Ende der Semesterferien zu erreichen galt. Darüberhinaus hatten wir mit den Nachbarprojekten beschlossen, dass am 24.10.03 jedes Projekt die bisherigen Ergebnisse präsentieren sollte. Die Idee war, zum einen zu schauen wie die Zwischenergebnisse der anderen so aussehen und zum anderen etwaige Sponsoren dazu zu bewegen, unsere Arbeiten finanziell zu unterstützen. Zu diesem Zweck bastelte unsere PR-Untergruppe einen ersten Flyer, der als Einladung dienen sollte. Zurückblickend kann man sagen, dass dieser Präsentationstag, doch einiges an Motivation für das Projekt bot und großen Arbeitseifer freisetzte. Die Sensorik-Gruppe konnte schon zwei ihrer Navigationsgräben präsentieren, deren Ankopplung an die Engine jedoch noch nicht ganz geschehen war. Das erste Modell, der Florifant, lief erstmals durch das selbstgebaute Dschungellevel. Dem war ein wahrer einmonatiger Kampf mit dem Export der Kreaturen aus dem Modellierungsprogramm in die Madness-Umgebung vorausgegangen. Auch die Grafik-und-KI-Gruppe konnte zufrieden sein, da nun ein eigenes Modell frei durch die Welt über ihr Wegesystem lief und so zeigte, dass man korrekt gearbeitet hatte. Bisher war jedoch ein Konzept zur Interaktion zwischen Benutzer und Kreaturen in der Welt noch nicht zu erkennen, auch besaßen die Kreaturen bis auf einige Dummy-Sounds keinen eigenen Sound. Konsens bestand darüber, dass es noch einiges im nächsten Semester zu tun gebe. Die PR-Gruppe hatte sich mittlerweile etwas gewandelt und arbeitete an einer Broschüre, die unser Projekt detailliert vorstellen und für Sponsoringzwecke genutzt werden sollte. Unser Projektleiter Bernd hatte in Dänemark vorgeschlagen, in Kooperation mit dem Kino 46 eine Aktionswoche am Ende des Projektes zu gestalten. Dieser Vorschlag stieß damals auf Begeisterung und wurde stets weiterverfolgt, woraufhin auch Alfred

Tews vom Kino 46 nach der Sichtung unseres Cave sein Interesse bekannte. Dass diese überuniversitäre Kooperation auch für Sponsoren interessant sein könnte, zeigte sich bald in einer großzügigen Rechner-Spende von der Firma Beck GmbH & Co KG. Ein Projektmitglied machte die Firma darauf aufmerksam und das Ergebnis waren insgesamt 12 Rechner die wir und MiCasa für einen begrenzten Zeitraum nutzen konnten.

Projekt vs. Streik

Im dritten Semester gab es ein Ereignis, dass die Projektarbeit etwas dämpfte: der Streik um die Einführung der Übernahme der Verwaltungsgebühr durch die Studenten. Die Vollversammlung der Studenten der Universität Bremen hatte am 03.12.2003 den Vollstreik aller Studenten beschlossen.

Da wir uns aber abgekoppelt vom Lehrbetrieb als eigenständiger, selbstverwaltender Bund von Studenten sahen, vermochten wir es nicht, die Projektarbeit ruhen zu lassen. Schließlich ging es darum, bis zum Projekttag am 09.07.2004 optimal zu arbeiten, da wir keine Möglichkeiten hatten, diese Deadline zu ändern. Dennoch lässt sich nicht leugnen, dass der Streik unser Arbeitstempo doch arg abbremste. Zur Jahreswende von 2003 nach 2004 sah es dementsprechend düster aus. Keiner konnte erahnen, wie lange sich der Streik hinziehen würde. Bisher liefen erst einige kleinere Kreaturen getrennt voneinander in einer Testumgebung. Dabei beschränkten sie sich nur aufs Laufen und Stehen. Sound gab es noch gar nicht. Es gab Probleme bei der Kollisionserkennung, den Motivationen der Kreaturen und bei der Interaktion untereinander, ganz zu schweigen von der Interaktion mit dem Benutzer. Der Druck der Broschüre zog sich immer weiter hin, eine geplante Homepage bestand nicht. Gerudert war bisher auch noch keiner im Cave. Auch war für den Projektbericht noch kein Wort geschrieben. Wir hatten zwar in den 16:00-Uhr-Veranstaltungen des laufenden Semesters, eine Art freitäglicher Abschlussveranstaltung, einige Referate für uns und die Nachbarprojekte gehalten, doch lagen noch keine Ausarbeitungen vor, die für eine Verwendung im Bericht geeignet wären. So verwundert es auch nicht, dass die Notendiskussion am Ende des dritten Semesters die bisher schwierigste war. Durch die Arbeitsteilung in verschiedene Gruppen, verschwand in vielen Gruppen der Blick für den Einzelnen. Die Grafik-Gruppe hatte sehr gute Arbeitsmöglichkeiten in der Universität und konnte dadurch ihre Fortschritte transparenter darstellen. Die Modellierer konnten freitags nur sehr bedingt arbeiten und mussten daher den Großteil der Woche für ihre Arbeit in Anspruch nehmen. Die Sensorik-Gruppe konnte ihre Arbeit kaum aufteilen und arbeitete zeitweise zu Hause. Daraus resultieren natürlich gewisse Vorbehalte untereinander. So beschlossen wir, dass sich die Untergruppen anhand an einer von Bernd gegebenen Gesamtnote orientieren und sich gegensei-

tig vorbenoten, um schließlich im Gesamtplenum konfliktreich und demoralisierend über die Noten zu diskutieren.

Endspurt

Dennoch fing man sich und arbeitete in den Semesterferien wieder angestrengter. Am Ende dieser Ferien hatte man erneut eine Testwoche angesetzt, zu der möglichst alle Kreaturen die Welt betreten haben sollten. Auch erste Sounds sollten zu hören sein. Ernüchternd musste man jedoch feststellen, dass viele Kreaturen es zwar in die Welt geschafft hatten, sie aber bisher nur liefen, schlafen und aßen und das ohne Animationsübergänge. Das Interaktionskonzept, Tiere anlocken und verscheuchen zu können, bestand noch nicht. Sounds waren von den Kreaturen noch keine zu hören. Gerudert werden konnte aber bereits und das nun in vier Gräben, die als Quadrat um den Benutzer angeordnet waren. Nun lag also das letzte Semester vor uns und es gab einiges zu tun. Erfreulicherweise verzahnte sich die Arbeit zwischen den Gruppen zunehmend. Der Grafik-Gruppe gelang es, die Animationsübergänge in den Ablauf der Kreaturen zu implementieren. Die Modellierer konnte selbstständig Wesen in eine Testwelt einfügen und ihre Arbeiten überprüfen. Der 3D-Sound wurde neu entwickelt und funktionierte nun korrekt. Interaktionen zwischen den Tieren und dem Benutzer sowie untereinander wurden realisiert. Die Broschüre wurde endlich gedruckt. Sponsoren wurden gesucht und gefunden. Die Zusammenarbeit mit dem Kino 46 konkretisierte sich und man entwickelte ein Programm für Ende Oktober namens mira.

Letzte Vorbereitungen zum Projekttag

Der Projekttag wurde weitestgehend von der Sensorikgruppe organisiert, da ihre Arbeit bis auf Floßverschönerungen relativ abgeschlossen war. Die Grafik-Gruppe öffnete sich allgemein mehr für Fragestellungen und besserte Fehler und Versäumnisse aus. Gegenseitige Hilfe war allen Ortes zu spüren und die Projektarbeit kam wöchentlich voran. Umgebungsgeräusche wurden von einer neu geschaffenen Soundgruppe mittels Digital Camcorder und Stereomikrofon aufgenommen, nachbearbeitet und sowohl für die Modelle verwendet, als auch in die Levels eingepflegt, von denen es nun mehrere Versionen gab. Irgendwann kamen schließlich sogar die von Becks versprochenen Rechner und versuchten uns mit ihren Kompatibilitätsproblemen jedwede Stimmung zu rauben. Durch beherztes Beratschlagen und selbstbewusstes Ausprobieren konnten diese Schwierigkeiten aber aus dem Weg geräumt werden. Sogar eine Homepage wurde in Angriff genommen. Einzig Performanzprobleme bereiteten uns größere Sorgen, so dass wir am Projekttag auf das spannende-

re und aufwändigere Level weitestgehend verzichten mussten. Dennoch kann beim 09.07.2004 von einem erfolgreichen Projektabschluss gesprochen werden. Zwar war unsere Präsentation durch eine erhöhte Latenzzeit bei der Videokonferenz unfreiwillig komisch, doch wurde unser Cave sehr gut besucht, wobei die Resonanz durchweg positiv ausfiel. Ausfallsfrei konnten alle die Lust dazu hatten unsere Welt zu betreten, Kreaturen anlocken, verscheuchen, aufwecken oder einfach nur umherrudern. Rückblickend erkennt man im Projektverlauf so etwas wie eine Sportfilmdramaturgie wieder: Dem motivierten Start, folgt nach der Auslotung der Möglichkeiten, ein sehr steiniger Weg, der an einem Tiefpunkt endet, worauf sich die Mannschaft wieder zusammenreißt und schlussendlich doch etwas liefert, hinter dem man selbstbewusst stehen kann.

— KAPITEL 13 —

Projektmanagement

Wie bringt man die unterschiedlichsten Studenten dazu, gemeinsam an einem Endprodukt zu arbeiten, Hindernisse zu überwinden, Probleme der verschiedensten Art zu bewältigen und das alles mit viel Motivation und guter Laune? Rückblickend können wir berichten, dass es oft schwierig und manchmal unmöglich war diese hoch gesteckten Ziele zu erreichen. Jedoch ziehen wir insgesamt ein positives Fazit aus unserer Form der Projektorganisation.

Das Arbeiten von 17 Studenten an einem gemeinsamen Projekt über vier Semester muss koordiniert werden, es gibt entweder einen Betreuer der dieses von Aussen in die Gruppe trägt, oder die Gruppe organisiert sich selbst.

Unser Betreuer ermutigte uns die Organisation des Arbeitens selbst zu übernehmen und so wurde schnell die Idee des so genannten Orga-Teams geboren, welches sich aus vier Projektteilnehmern zusammensetzte und dem jeder mindestens einmal angehören sollte. Das Wechseln von Orga-Team-Mitgliedern stand einmal im Monat an, wobei immer nur zwei Leute hinaus und zwei neue hinein kamen. Eine alphabetische oder andere Reihenfolge wurde hierbei nicht beachtet, vielmehr wurden immer nur Leute ins Orga-Team aufgenommen die auch gerade Zeit hatten, diese zusätzliche Arbeit zu übernehmen.

Das nie vollständige Wechseln, bzw. Austauschen des Orga-Teams stellte sicher, dass Ideen, Ziele und Beobachtungen eines Teams immer in das nächste mit übergingen.

Das Orga-Team traf sich einmal wöchentlich und stellte die Tagesordnung für das nächste Plenum auf. Das Protokoll des letzten Plenums wurde besprochen und etwaige Fehler wurden dem Protokollanten gemeldet, und er berichtigte sie. Die Leitung des Plenums übernahm auch das Orga-Team. Protokoll wurde immer projektübergreifend, in alphabetischer Reihenfolge geführt und es gab die Auflage, dass die Protokolle immer in der Woche nach dem Plenum bis zwei Tage vor dem nächsten Treffen fertig sein mussten.

Im Projektverlauf war das Orga-Team im ersten Semester nicht sehr stark gefordert, dies änderte sich nach der Zielfindung und der Aufteilung in die so genannten Arbeitsgruppen, also Grafik, Sensorik, Sound und Modellierung. Mit dem Ziel, das Arbeiten der Einzelgruppen für alle Projektteilnehmer nachvollziehbar zu machen, führte das Orga-Team die so genannten 3-Wochen-Ziele ein. Jede Arbeitsgruppe steckte sich ein Ziel, welches nach drei Wochen erreicht sein sollte. In jedem Plenum wurde dann vom gesetzten Ziel, dem aktuellen Stand oder Abschluss des Ziels berichtet. Dies hatte auch den Vorteil, dass die Gruppen untereinander immer wussten, wann sie mit Ergebnissen einer anderen Gruppe rechnen und selber damit arbeiten konnten. Diese Neuerung setzte sich im weiteren Projektverlauf fort und führte zu einem harmonischeren Arbeiten in den Gruppen und zwischen den Gruppen, da das gesamte Arbeitsvorhaben nun einer gewissen Transparenz unterlag, und somit jeder wusste woran gerade von wem gearbeitet wurde.

Dass es leicht werde würde, mit und in einer Gruppe dieser Größe zu arbeiten, hat sicher kein Teilnehmer vorher angenommen. Konflikte offen lösen, mit Demotivation fertigwerden, über persönliche Grenzen gehen, im Umgang mit anderen völlig neue Seiten an sich kennen lernen, Meinungen verteidigen, Meinungen ändern, Kompromisse eingehen lernen – die Liste ließe sich noch viel weiter verlängern. Sie soll zeigen, dass nicht nur unser eigentliches Endprodukt – das Cave – ein Produkt des Projekts ist. Die Organisation dieser Gruppe, an der jeder Teilnehmer beteiligt war, ist ein mindestens genauso großes und bedeutendes Endprodukt.

— KAPITEL 14 —

Sponsoring

Das Sponsoring unseres Projekts durch externe Firmen entstand aus der Problematik, dass das Institut artec drei Teilprojekte mit dem Geld für ein Projekt finanzieren musste. Die Ausgaben, die das artec für die drei MiCa-Teilprojekte hatte, sollten durch das Sponsoring wieder ausgeglichen werden. Somit wurde beschlossen, dass jeweils die Teilprojekte versuchen sollten, bei Firmen vorstellig zu werden und dadurch Sponsoren zu gewinnen.

Wir, MiCaDo und MiCasa, hatten beschlossen, unser gemeinsames Projekt MiCa auch gemeinsam bei Firmen zu vertreten. Da wir drei Teilprojekte waren, sollte sich auch MiCarpet dem gemeinsamen Sponsoring anschließen, um besseren Erfolg zu haben und das Gesamtprojekt besser nach außen hin vertreten zu können. MiCarpet hatten sich aber schon eine eigene Werbestrategie nur für ihr Teilprojekt überlegt und schlossen sich somit auch nicht unserem Vorhaben an.

Damit wir uns nach außen verkaufen konnten, brauchten wir Material, mit dem wir uns bei Firmen vorstellen konnten. Es wurde ein erstes Faltblatt entworfen, das an die Anforderungen des Corporate Designs der Universität Bremen angepasst wurde. Dieses Faltblatt verwendeten wir sowohl als Auslage-Flyer für das artec bei Terminen außerhalb der Uni, als auch um potentiellen Firmen zu unserem Sponsoringgesuch etwas Bildliches dazu zu geben. Ferner wurde dieser Flyer als Einladung zu einer Zwischenpräsentation der drei MiCa-Caves verwendet.

Die Idee, auf möglichst großem Raum unsere Ideen und Arbeiten darzustellen, um Sponsoren für uns begeistern zu können, kam auf. Dafür sollte eine Image-Broschüre erstellt werden, wobei es das Ziel war, die Kosten möglichst gering zu halten. Gemeinsam mit der MiCasa-Gruppe erarbeiteten wir

ein Grundlayout für die Broschüre und gewannen mit dem Vater eines MiCasa-Projektteilnehmers unseren ersten Sponsor, der uns die Finanzierung des Drucks sicherte. Nachdem das endgültige Layout feststand und in den Druck gegeben wurde, fingen wir an Firmen herauszusuchen, die wir anschließend mit der Broschüre besuchen und als Sponsoren gewinnen wollten. Um auch überregionale Firmen ansprechen zu können, setzten wir einen Text für ein Anschreiben auf, das den betreffenden Firmen zusammen mit der Broschüre zugeschickt werden sollte.

Nachdem dann die Broschüren fertig waren, begannen wir, die vorher festgelegten Firmen zu besuchen. Hierzu kann dem Anhang auf Seite 147 entnommen werden, wer uns aufgrund dieser Bemühungen sponserte.

Eine weitere Aktion unseres PR-Teams war die Bewerbung des Projekttags an der Uni. Hier stellten alle Projekte ihre Ergebnisse aus vier Semestern Arbeit vor. Hierzu wurde ein Plakat entworfen und an vielen Stellen in der Universität verteilt. Abbildung 14.1 zeigt dieses. Das Anschreiben und die Sponsoren-Liste finden sich im Anhang.

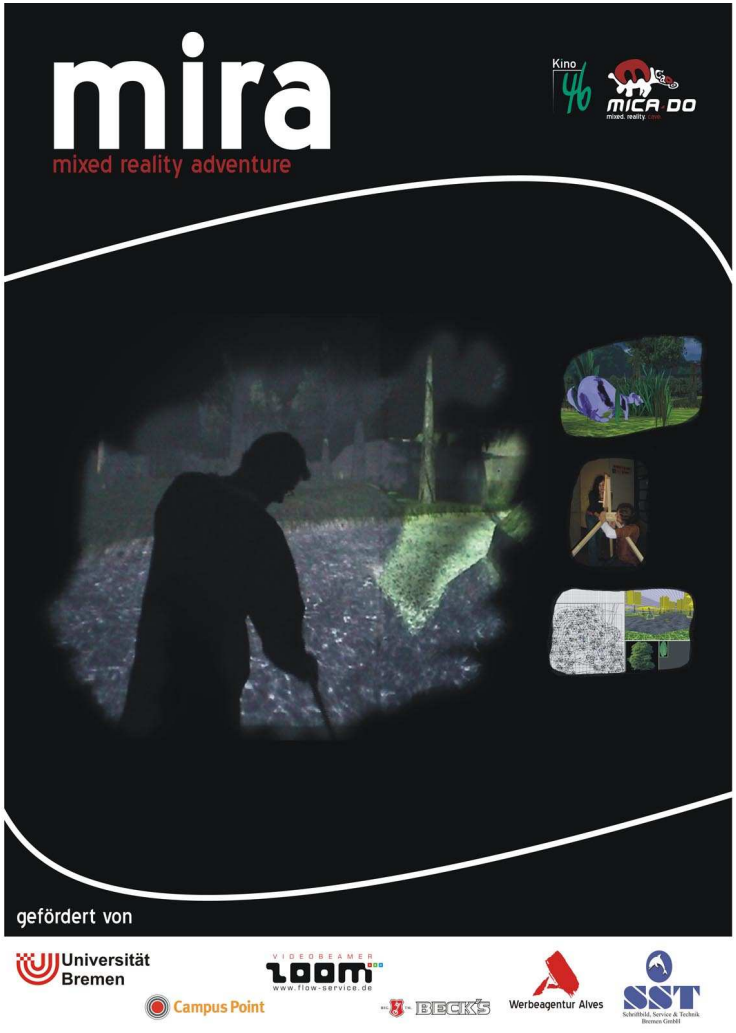

Abbildung 14.1: Das Projekttagsposter unseres Projekts

— KAPITEL 15 —

Der Projekttag

Der Projekttag[1], lange Jahre schon Tradition am Fachbereich Informatik der Bremer Universität, stand auch in diesem Jahr wieder auf der Tagesordnung, diesmal am 9. Juli 2004. Er steht für den Tag, an dem alle auslaufenden Projekte des Hauptstudiums ihre Arbeit einem Publikum, bestehend aus Kommilitonen, Professoren/innen und anderen Gästen, vorstellen. Die erste Ebene des MZH (ein Gebäude der Universität) diente an diesem Tag als Veranstaltungsort in vielerlei Hinsicht. Die Räume 1400 und 1380 stellten den Raum für die dreißigminütigen Präsentationen der jeweiligen Projekte. In der ersten Ebene selbst war jedes Projekt mit einem Informationsstand vertreten, an denen sich die Besucher einen genaueren Einblick in die Arbeit der jeweiligen Projekte verschaffen konnten. Dieses Jahr war es so, dass aus Platzgründen nicht alle Projekte ihre Arbeit direkt in der ersten Ebene ausstellen konnten. Die drei MiCa-Projekte[2] waren mit ihren Cave-Aufbauten außerhalb der ersten Ebene. MiCarpet hatte ihren Cave im Theatersaal der Universität aufgebaut, MiCasa war im Studierhaus zu finden und MiCaDo im MZH 5210. Neben den Informationsständen gab es wie immer ein reichhaltiges Sortiment an selbstgebackenen Kuchen und Getränken[3].

[1] Ablaufplan unter: http://www.informatik.uni-bremen.de/fb3/sonstiges/projekttag.html
[2] MiCarpet, MiCasa und MiCaDo
[3] inklusive der Brauerei Beck GmbH & Co KG Getränkespende

15.1 Vorbereitung

Die Arbeit von vier Semestern in 30 Minuten erzählen, vor diesem Problem standen nun auch wir. Um die Ideenfindung ein wenig zu erleichtern, wurde eine Projekttagsgruppe gebildet, die sich um die Idee und den groben Ablauf Gedanken machen sollte. Als Bedingung hatten wir uns vorgenommen, keine schon vorhandenen Ideen umzusetzen, also kein Theaterstück, keine Talkshow und was sonst noch so auf den letzten Projekttagen umgesetzt wurde bzw. uns bekannt war. Die Vorstellung sollte einerseits informativ, unterhaltsam und nicht zu langatmig wirken. Noch dazu kam, dass wir unsere eigentliche Arbeit nicht in der ersten Ebene des MZH ausstellen konnten und so das Publikum auf irgendeinem Wege nach oben in den Raum 5210 locken mussten. Es kam die Idee auf, unsere Präsentation mit unserem eigentlichen Standort zu verknüpfen. Eine Liveschaltung war das Stichwort. Das Szenario sollte folgendes sein: Ein Forscherteam bestehend aus Herrn Dr. MiCaDo und Frau Dr. Jane waren unterwegs bei einer Forschungsreise auf einen außergewöhnlichen Dschungel gestoßen. Nach einer Weile gelang es ihnen mit der Heimatwelt Kontakt aufzunehmen, um von der bemerkenswerten Vegetation und den dort lebenden Tieren zu berichten. Auf einer Pressekonferenz, zu der Experten aus verschiedenen Gebieten[4] eingeladen wurden, sollte dieses Ereignis der Öffentlichkeit vorgestellt werden. Es sollte eine erneute Verbindung zu dem Forscherteam geschaffen werden, um dann eine direkte Kommunikation zwischen den Forschen und den Experten zu ermöglichen. Im Anschluss an den Dialog zwischen den eingeladenen Experten und dem Forscherteam, sollten ein paar Fragen aus dem Publikum kommen und die Antworten dazu, zu unserem eigentlichen Standort im MZH 5210 verweisen.
Von diesem Konzept waren nach einer kurzen Überzeugungsphase alle Projektmitglieder überzeugt. So wurden Drehbücher für die Schauspieler geschrieben und eine technische Umsetzung entwickelt, die es möglich macht eine Liveschaltung vom Pressesaal zum Raum 5210 im MZH herzustellen.

15.2 Ablauf

Natürlich begann der Projekttag für uns nicht erst um 9:45 Uhr, wir hatten uns schon einen Tag vorher daran gemacht den Cave aufzubauen und einige von uns trafen sich am „Tag der Tage“ schon gegen 7 Uhr, um ein paar abschließende Tests zur eigentlichen Liveübertragung zu machen, sowie

[4] Prof. Dr. Dijkstra (Wegefindung), Graf Neuronal (künstliche Intelligenz) sowie Dr. Festo (Sensorik/Aktorik)

Abbildung 15.1: Aufbau des Caves für den Projekttag

ein Sicherungsvideo aufzunehmen, dass wir im „Fall der Fälle“ einspielen konnten. Die letzten Tests liefen zufriedenstellend und das Video war im Kasten, die Präsentation konnte also kommen.

Der letzte Slot vor der Mittagspause war unser. Nachdem die Bühne in einen Pressesaal verwandelt wurde, sollte es losgehen. Die unzähligen Vorproben hielten aber nicht das was sie versprachen, der Start verzögerte sich aufgrund technischer Probleme um ein paar Minuten. Es ließ sich einfach keine Videoverbindung zum Aussenteam herstellen und die Audioverbindung verweigerte ebenso fast gänzlich ihren Dienst. Quasi in letzer Sekunde, als man sich schon fast dazu entschieden hatte das Sicherungsvideo einzuspielen, kam doch noch eine Verbindung zu Stande und die Präsentation ging los. Diese Verbindung war zu Anfang allerdings ziemlich schlecht, genaugenommen bestand eine zeitliche Verzögerung von mindestens 10 Sekunden. Diese besondere Situation führte zu einigen unvorhergesehenen Effekten.
Der Moderator stellte eine Frage an das Forscherteam, wartete kurz auf eine Antwort und ging dann, als keine Antwort kam, zum nächsten Punkt über. Bei der nächsten Frage an das Forscherteam kam

dann eine Antwort, nur gehörte sie zu der letzten Frage. Dieses und weitere verblüffende Ereignisse sorgten für eine gemischte Stimmung im Raum, einige von den Besuchern dachten es sei von uns so geplant gewesen, andere fanden es einfach lustig und waren überrascht wie unterhaltsam doch eine Projektpräsentation sein kann.
Ein Neustart der Verbindung nach ungefähr der Hälfte der Zeit brachte Besserung. Die Verbindung war plötzlich nahezu perfekt und es existierte nur noch eine zeitliche Verzögerung von ca. 3 Sekunden. Ab diesem Zeitpunkt war wieder eine normale Kommunikation möglich und die Vorführung wurde erfolgreich zu Ende geführt. Das Publikum quittierte unsere Leistung mit einem tosenden Applaus wie man ihn nur selten erlebt hat.
Im Nachhinein betrachtet also ein voller Erfolg und vielleicht hat ja gerade das Unvorhergesehene unsere Abschlussvorführung noch ein bisschen spannender gemacht.

— KAPITEL 16 —

mira im Kino 46

Die Idee zu einer Präsentation unseres Projektergebnisses außerhalb der Uni kam im Laufe des ersten Projektsemesters von Bernd Robben, unserem Projektleiter. Er fragte uns, ob wir Interesse an einer solchen Vorführung unseres Caves hätten. Es gab eine große Resonanz im Projekt und so wurde es im zweiten Semester als eine feste Sache betrachtet. Bernd Robben nahm im Verlauf der Projektzeit mit dem Kino 46 Kontakt auf und bekundete unser Interesse an einer solchen Veranstaltungsreihe. Das Kino 46 hatte schon vorher einmal eine Vortrags- und Filmreihe zu einem bestimmten Thema in Kooperation mit der Uni gemacht, sodass die Chance sehr groß wäre, dass das Projekt MiCaDo seinen Cave austellen könnte. Das Oberthema der Veranstaltung sollte Höhlen- und Grenzerfahrungen sein, im speziellen sollte es eine Film- und Vortragsreihe um die Cave-Installation sein. Nachdem das Kino 46 uns eine Zusage erteilt hatte, machte sich zunächst Skepsis breit, weil die Veranstaltungsreihe nur nach der offiziellen Projektzeit stattfinden konnte. Da aber die Mehrheit der Projektmitglieder ein solches Projekt als Chance sah, das Ergebnis eines studentischen Projekts der Informatik einmal nicht nur den anderen Studierenden am Projekttag vorzustellen, sondern auch anderen Interessierten, wurde beschlossen, diese Chance zu nutzen. Damit war klar, dass das Projekt MiCaDo nicht wie alle anderen Projekte nach dem Projekttag geschlossen würde, sondern von da an mit neuem Ziel, der Vortrags- und Filmreihe mira weiterlaufen und den Cave der breiten Öffentlichkeit vorstellen würde.

16.1 Planung

Damit die Planung für dieses Event strukturiert ablaufen konnte, wurde eine Gruppe gebildet, die sich eben um die Planung kümmern sollte. Die Kino 46-Gruppe bestand aus 4 Projektmitgliedern und wurde von Bernd Robben und Ralf E. Streibl unterstützt. Zunächst bestand die Aufgabe der Gruppe darin, einen passenden Termin zu finden. Dieser sollte möglichst nah am Semsterende, also am eigentlichen Projektende liegen, um dieses nicht unnötig nach hinten zu schieben. Da das allgemeine Programm des Kino 46 schon relativ früh für die nächsten Monate feststeht, wurde der nächstmögliche freie Termin im Oktober 2004, der durch das Kino 46 anboten wurde, ausgewählt.

Da das Projekt MiCaDo nicht alleiniger Veranstalter dieser Film- und Vortragsreihe war, sondern auch andere Institutionen, die nicht unbedingt etwas mit dem eigentlichen Cave zu tun haben, musste ein neuer Name für das Event gefunden werden. Dieser Name sollte gut zu merken, kurz und aussagekräftig sein. Zudem sollten die Filme und Vorträge und der Cave mit dem Namen in Verbindung stehen. Das Projekt entschied sich nach einer Abstimmung für mira, was für MIxed Reality Adventures steht und im Spanischen „Schau hin!“ bedeutet. Damit wurde sowohl der Mixed Reality Cave beschrieben, als auch ein guter Name für Kinofilme und Vorträge gefunden.

Für die Auswahl der Filme und Referenten der Vorträge war sowohl das gesamte Projekt MiCaDo, Alfred Tews vom Kino 46 und Ralf E. Streibl zuständig. Zunächst wurde vom Projekt MiCaDo eine Vorschlagsliste mit Filmen erstellt, die dann bei den Treffen im Kino 46 mit den andern Beteiligten besprochen wurde und aus der die entgültigen Filme für das Programm herausgesucht wurden. Alle Filme hatten im weiteren Sinne das Thema Höhlen und Grenzerfahrungen. Desweiteren wurde beschlossen, dass es eine Kurzfilmreihe mit elektronischen Filmen von der Siggraph und eine Kurzfilmreihe mit Filmen von der ARS-Elektronica geben soll. Die Siggraph-Filme wurden an zwei Vorauswahlabenden im Kino 46 bestimmt, damit diese eine Zeit von ca. zwei Stunden Film ergaben. Die Referenten der Vorträge wurden ebenso von MiCaDo, Alfred Tews, Bernd Robben und Ralf E. Streibl vorgeschlagen.

Unserem Projekt lag aber nicht nur die Film- und Vortragsreihe am Herzen, sondern vor allem, wie man es schafft, möglichst viele Menschen in unseren Cave zu locken. Dabei musste man bedenken das der Cave, aufgrund von Platzproblemen, nicht direkt im Kino 46 aufgebaut werden konnte und wir für unsere Cave-Installation deswegen ein eigenes Programm finden mussten. Der Vorschlag unseren Cave als Mixed-Reality-Installation zu bezeichnen und das Ganze wie eine Art Ausstellung zu präsentieren, ließ uns auf die üblichen Vorgehensweisen bei normalen Austellungen stoßen. Das hieß, wir entschlossen uns eine Vernissage am Anfang und eine Finissage am Ende der Ausstellungs-

zeit zu machen. Das verlieh unserer Installation auch einen kleinen Exclusivitätstouch und wir hatten die Möglichkeit Leute direkt zur Vernissage einzuladen und somit mehr Publikum zu bekommen.

Das Veranstaltungprogramm der Woche des Kino 46 zu gestalten, war eine sehr arbeitsintensive Aufgabe. Eine noch weitaus schwierigere Aufgabe war es, einen geeigneten Raum zu finden, in dem wir unseren Cave für 10 Tage aufbauen konnten. Die Raumsuche für den Cave gestaltete sich deshalb als sehr schwierig, da es in Bremen sehr wenig verkehrsgünstig gelegene Räume in der benötigten Größe gibt, die man für 10 Tage günstig mieten oder kostenlos benutzen kann. Es wurden verschiedene Räume vorgeschlagen, in denen aber entweder der Cave vom Platz her nicht hinein passte oder er nicht die Zeit über aufgebaut bleiben konnte, weil der Raum auch noch für andere Zwecke benutzt wurde. Ein weiteres Problem stellte die Versicherung der gesamten technischen Anlage dar, weil diese nur für abschließbare Räume abgeschlossen werden kann. Also fielen auch alle Räume weg, die nicht verschlossen werden konnten. Bei einer Diskussion über den Raum wurde schließlich das Lichthaus beim Pier2 vorgeschlagen, das einen genügend großen, abdunkelbaren Vortragsraum besitzt, den man für solche Zwecke bekommen kann. Zusätzlich ist dieser Raum durch eine hausinterne Alarmanlage geschützt und abschließbar, sodass man die Geräte auch ohne Aufsicht stehen lassen kann. Da das Lichthaus die mit Abstand besten Bedingungen für unseren Cave bot, wurde dieser Raum für die Dauer vom mira reserviert.

Mit dem Ziel möglichst viele Menschen für unseren Cave im Lichthaus und die dazugehörige Film- und Vortragsreihe zu gewinnen, wurden ein Flyer entworfen, der in Bremen verteilt und an potentiell Interessierte verschickt wurde, Zeitungsartikel für Bremer Stadtillustrierte geschrieben und Kontakt zu Bremer Radio- und Fernsehsendern aufgenommen. Um das Ganze nicht nur im Vorfeld zu bewerben, wurde eine Pressekonferenz am Tag vor der Eröffnung anberaumt, bei der Reportern von interessierten Zeitungen und Rundfunkanstalten Rede und Antwort gestanden werden sollte.

16.2 Ablauf im Lichthaus

Der Aufbau des Caves und der dazugehörigen Technik lief wie zuvor ausführlich geplant ab. Zwei Tage vor der offiziellen Eröffnung standen sowohl der Cave als auch die Videoprojektoren und Rechner an ihren für die Ausstellungszeit festen Positionen. Am Freitag vor der Vernissage fand die schon oben erwähnte Pressekonferenz statt.

Die Vernissage zu unserer Installation am 23.10.2004 wurde ein gelungener Auftakt unserer Ausstellung im Lichthaus. Bernd Robben übernahm die Moderation des Nachmittags und kündigte die

jeweiligen Akteure an. Sie wurde durch den Rektor der Universität Bremen Wilfried Müller mit einer Rede über Hauptstudienprojekte der Informatik und das Platonsche Höhlengleichnis eröffnet.

Nach der Eröffnung wurde folgende experimentelle Musik vom Atelier Neue Musik gespielt:

- **1.Stück:** Joachim Heintz, *Koax Blau*
  mit filmischer Untermalung von Charlotte Boesen
- **2.Stück:** Ali Gorji, *Verzweigte Wasser*
  Marianne Schulte, Violine
  Florian Oberlechner, Akkordeon
- **3.Stück:** Joachim Heintz, *Koax Rot*
  mit filmischer Untermalung von Charlotte Boesen

Um den Gästen unser Projekt und den Cave etwas näher zu bringen und das Programm abzurunden, hielten zwei Projektmitglieder von MiCaDo einen Vortrag über die Entstehung unseres studentischen Projekts. Nach dem förmlichen Ablauf wurde der Cave offiziell für die Gäste der Vernissage und somit für jeden Interessierten eröffnet. Die Vernissage wurde am Abend so beendet, wie sie am Nachmittag begonnen hatte: Mit einem Glas Prosecco oder Orangensaft und Knabbereien.

In der Woche der Ausstellung zwischen der Vernissage und der Finissage am 31.10.2004 durften wir insgesamt 114 interessierte Gäste in den Cave führen und Dschungelwelten erforschen lassen. Es gab ausschließlich zufriedene und begeisterte Menschen, also nur postive Reaktionen. Zusätzlich lief von Donnerstag bis Sonntag die begleitende Film- und Vortragsreihe in den Räumen des Kino 46.

Die Finissage sollte einen würdigen Abschluß unserer Installation bilden. Diese wurde unter anderem von Ralf E. Streibl, der gute Kontakte zu der Bremer Shakespeare Company pflegt, organisiert. Da die Shakespeare Company einmal im Monat ein Matinee mit Lesung zu verschiedenen Themen veranstaltet, wurde abgeklärt, ob nicht eine der Matineen im Rahmen unserer Installation in Form einer Finissage stattfinden könnte. Wir konnten Peter Lüchinger von der Shakespeare Company Bremen für eine Lesung bei uns gewinnen, der dann auch mit verschiedenen, thematisch passenden Kurzgeschichten das Publikum unterhielt. Nach der Lesung war der Cave ein letztes Mal geöffnet, sodass auch alle Gäste der Finissage einmal in den Dschungel eintauchen konnten.

Filmmaterial zur Vernissage, den Vorträgen und der Finissage werden auf DVD in digitaler Form beigelegt.

Abbildung 16.1: Der aufgebaute Cave im Lichthaus

— KAPITEL 17 —

Nachwort

Vor über zwei Jahren standen wir als neu gegründete Projektgruppe MiCaDo in einem leeren Raum. Das, was in der Zeit bis heute entstanden ist, war nur in der Fantasie unseres Geistes gegenwärtig. Dieses abstrakte Gebilde von Vorstellungen und Wünschen bahnte sich in der kommenden Zeit einen anstrengenden Weg zu einer realen Manifestation. Es galt die vielen unterschiedlichen Vorstellungen, wie der Raum aussehen sollte, in ein Konstrukt zu verwandeln, das die unterschiedlichen Ideen vereinte und aus der Kreativität ihrer Denker schöpfte. Da, wo die Kreativität vieler Individuen zusammentrifft, um an einem Projekt zu arbeiten, entsteht notwendigerweise Diskussionsbedarf. Wir haben viel diskutiert, um herauszufinden, welche fantastischen Ideen in unseren Köpfen stecken, welche realisierbar sind und welche in die Vorstellungen aller passen würden. Dies ist bestimmt der anstrengendste Abschnitt des Projektes gewesen, aber am Ende stellen wir fest, dass es sehr wichtig gewesen ist, Zeit und Arbeit in die Phase der Ideenfindung zu stecken.

Wir haben viel gelernt in diesen zwei Jahren, nicht nur viel neues fachliches Wissen aufgenommen, sondern auch gelernt, wie eine Gruppe zu agieren und zu denken. Das Arbeiten in einer Gruppe bedarf Regeln und Abmachungen, um konstruktiv zu funktionieren. Es unterscheidet sich stark von der vorher bekannten Gruppenarbeit mit nur drei oder vier Teilnehmern. Das Arbeiten ist näher an der Teamarbeit von Unternehmen angelehnt, mit dem Unterschied, dass wir gleichzeitig Arbeitnehmer und Arbeitgeber sind. Dieses zu erkennen und damit zu arbeiten, ist ein wichtiger Prozess, den das Projekt in uns angestoßen hat.

Wir hätten dieses Projekt nicht abschließen können, ohne die Hilfe und Vorarbeit vieler Helfer, die uns begleitet haben. Einigen möchten wir hier namentlich danken.

Zuerst danken wir unserem Tutor und MiCaDo-Projektbetreuer Bernd Robben, der uns die ganze Zeit über mit seinem Rat und seinen Taten sehr hilfreich unterstützt hat. Bernd, wir danken dir sehr für deine Fähigkeit und Geduld, der „Ausguck" in unserem Boot zu sein, ohne das Ruder zu ergreifen, und für deinen Rat, der uns geholfen hat, Irrwege und Hindernisse zu umfahren.

Großer Dank geht auch an unseren Prof. Dr. F.-W. Bruns, der dieses Projekt ins Leben gerufen hat. Willi, danke, dass du eine so große Mannschaft bei dir aufgenommen hast und die gute Idee hattest, diese auf 3 Boote zu verteilen. Wir sind glücklich darüber, dass unser Projekt bei dir möglich war und wir freie Hand hatten, deinen visionären Ideen zu folgen.

Wir danken auch Martin Faust, der uns seine 3D-Engine nicht nur zur Verfügung gestellt hat, sondern sie für diesen Anwendungszweck optimiert hat. Hierdurch war es möglich, schnell die positiven Erlebnisse zu gewinnen, die unsere Ideen beflügelt haben.

Der letzte Dank geht an das Institut artec, dem diese drei Personen angehören und bei dem wir die letzten zwei Jahre zu Gast waren.

Diese Unterstützung machte aus dem leeren Raum am Anfang eine neue fantasievolle Raumwelt, der wir uns gern gewidmet haben. Die Manifestation unserer Ideen ist gelungen, das Ergebnis zeigen wir stolz hervor, um andere mit Ideen zu inspirieren und gemeinsam die Realitäten zu bestaunen.

Das Projekt MiCaDo
Universität Bremen, 2004

— KAPITEL 18 —

Projektteilnehmer

Bernd Robben
robben@tzi.de
Projektbetreuer

Armstrong Afopezi
lengoch@tzi.de
Modellierung, PR

Jasper Arndt
jarndt@tzi.de
KI/Grafik

Henning Burow
eniac@tzi.de
Modellierung,Video,Sound

Michael Dippold
midi@tzi.de
Sensorik

Johannes Goos
gross@tzi.de
KI/Grafik

Katja Graupner
katie@tzi.de
Sensorik, PR

Jan Hövelmann
morpheus@tzi.de
Modellierung

Melanie Johänning
mella@tzi.de
Modellierung, PR

Hans-Christoph Jakobeit
jakobeit@tzi.de
KI/Grafik

Stephan Leinweber
pseudo@tzi.de
Sensorik

Simon Meyborg
meyborg@tzi.de
Modellierung, Sound

Florian Pantke
fpantke@tzi.de
KI/Grafik, Sound

Lars Struß
lst@tzi.de
Sensorik

Oliver Vahjen
ollivah@tzi.de
Sound

Anne Virnich
kleine@tzi.de
Modellierung, PR

Martin Alexander Walschik
walschik@tzi.de
Sensorik

Sven Werner
swerner@tzi.de
KI/Grafik

Falls Sie Fragen haben, wenden Sie sich entweder an eine Person aus dem bestimmten Fachgebiet oder schreiben Sie eine Email an *micado@informatik.uni-bremen.de*.

Anhang

Anhang Sponsoring

E-Mail Anschreiben

Betreff: Anfrage Sponsoring / Öffentlichkeitsarbeit

Sehr geehrte Damen und Herren,

wir stehen kurz vor dem erfolgreichen Abschluss und der Präsentation unseres zweijährigen Projekts an der Universität Bremen im Fachbereich Informatik. Genaue Informationen zum Projekt MiCaDo entnehmen Sie bitte unserer Info-Broschüre, zu finden im Anhang dieser E-Mail.

Einer erfolgreichen Abschlusspräsentation steht ein Mangel an Gerätschaften entgegen, die uns während der Entwicklungszeit aus universitären Mitteln zur Verfügung gestellt wurden, jedoch am Projekttag auch von vielen anderen Projekten beansprucht werden. Der Projekttag bietet allen Projekten aus dem Fachbereich Informatik (insgesamt 11 Projekte) eine Plattform zur Präsentation ihrer Ergebnisse aus zwei Jahren Arbeit.

Hierzu sind Sie am *(Datum, Uhrzeit)* herzlich eingeladen!

Wir würden uns sehr freuen wenn Sie in Form eines Sponsorings dazu betragen möchten, uns einen erfolgreichen Projektabschluss zu ermöglichen.

Es wäre schön, von ihnen zu hören.

Mit freundlichen Grüßen,

Das MiCaDo-Projekt, Universität Bremen

Liste der Sponsoren

- Campus Point Bremen
- Beck's
- Schriftbild Service Technik Bremen (SST)
- Universität Bremen
- Zoom Projektorverleih

- Werbeagentur Alves

Anhang Modellierung

Quake III konforme animation.cfg

Das folgende Listing zeigt die Konfiguration der Animationssequenzen für einen 3D-Charakter, wie sie im Spiel Quake III Arena verwendet wird:

```
// animation config file

sex m

// first frame, num frames, looping frames, frames per second

0       30      0       20      // BOTH_DEATH1
29      1       0       20      // BOTH_DEAD1
30      30      0       20      // BOTH_DEATH2
59      1       0       20      // BOTH_DEAD2
60      30      0       20      // BOTH_DEATH3
89      1       0       20      // BOTH_DEAD3

90      40      0       18      // TORSO_GESTURE

130     6       0       15      // TORSO_ATTACK
136     6       0       15      // TORSO_ATTACK2

142     5       0       20      // TORSO_DROP
147     4       0       20      // TORSO_RAISE

151     1       0       15      // TORSO_STAND
152     1       0       15      // TORSO_STAND2

153     8       8       20      // LEGS_WALKCR
161     12      12      20      // LEGS_WALK
173     11      11      21      // LEGS_RUN
```

```
184     10      10      20      // LEGS_BACK
194     10      10      15      // LEGS_SWIM

204     10      0       18      // LEGS_JUMP
214     6       0       20      // LEGS_LAND

220     8       0       15      // LEGS_JUMPB
228     1       0       15      // LEGS_LANDB

229     10      10      15      // LEGS_IDLE
239     8       8       15      // LEGS_IDLECR

247     7       7       15      // LEGS_TURN
```

MiCaDo-Erweiterung der animation.cfg

Für den unteren Teil des 3D-Modells haben wir weitere Animationen definiert. Die folgenden Zeilen können daher der oben abgedruckten Standardkonfiguration folgen:

```
0       29      29      25      // MICADO_ANIMATION_STAND

30      1       0       0       // MICADO_ANIMATION_WALK_START
30      149     149     150     // MICADO_ANIMATION_WALK_LOOP
180     1       0       0       // MICADO_ANIMATION_WALK_TO_RUN
180     59      59      150     // MICADO_ANIMATION_RUN_LOOP
240     1       0       0       // MICADO_ANIMATION_RUN_TO_WALK
240     1       0       0       // MICADO_ANIMATION_WALK_STOP
255     29      29      25      // MICADO_ANIMATION_SIT_START
285     44      44      25      // MICADO_ANIMATION_SIT_LOOP
330     74      74      25      // MICADO_ANIMATION_SIT_STOP
420     29      29      25      // MICADO_ANIMATION_EAT_START
450     44      44      25      // MICADO_ANIMATION_EAT_LOOP
495     14      14      25      // MICADO_ANIMATION_EAT_STOP
525     29      29      25      // MICADO_ANIMATION_SLEEP_START
```

```
555     29      29      25      // MICADO_ANIMATION_SLEEP_LOOP
585     29      29      25      // MICADO_ANIMATION_SLEEP_STOP
630     26      26      25      // MICADO_ANIMATION_GESTURE_START
657     32      32      25      // MICADO_ANIMATION_GESTURE_LOOP
690     29      29      25      // MICADO_ANIMATION_GESTURE_STOP
// 735     60      0       25      // MICADO_ANIMATION_DEATH_1
// 810     30      0       25      // MICADO_ANIMATION_DEATH_2
// 855     15      0       25      // MICADO_ANIMATION_DEATH_3
```

Einzelne Animationen können auch auskommentiert werden (siehe die letzten drei Zeilen), wodurch die jeweilige Animation deaktiviert wird und für das Modell nicht zur Verfügung steht.

Anhang Pathfinding

Translation File demo.trans

Das folgende Listing zeigt das Translation File zu dem in Abschnitt 7.3.4 besprochenen `pathtool`-Beispielszenario (vgl. Abb. 7.6).

```
// ---------------------------------------------------------------------------
// This is an example translation file for pathtool. Feel free to edit.
// ---------------------------------------------------------------------------

prefs {

    // -= settings for terrain-scanning: =-

    stepsize = 1.0              // scanning resolution (units per step)
    maxfall = 32.0              // the maximum distance a creature may fall
    heightofs = 10.0            // collision detection offset from floor
    radius = 8.0                // collision detection sphere radius


    // -= settings for shortest-path calculation: =-

    maxdepth = 512              // maximum allowed path length (#transitions)
    penalty = 66.6              // will be added to each transition's cost
                                // (to allow 'believable' creature behaviour)

}
group {
    classname = ammo_bullets    // the name of the goodie to be translated
    group = 3                   // the group number to translate to
    requires = cook_chinese wok // capabilities this group requires
    provides = food             // capabilities this group provides
}
```

```
group {
    classname = item_health
    group = 3                           // same group number as above,
    requires = cook_indian curcuma      //   but different capabilities
    provides = tandoori_chicken         //   to be assigned!
}
group {
    classname = ammo_shells             // well, you get the idea...
    group = 4
    requires = iq_above_150
    provides = PhD
}
shader {
    shadername = textures/liquids/lavahell_750  // name of a shader used in map
    nocross =  3 4                      // groups which cannot walk on it
    requires = asbestos_suit            // additional capabilities to be
    provides = hot_butt                 //   assigned to each waypoint laying
                                        //   on a polygon with this shader
                                        //   assigned to it
}
shader {
    shadername = textures/base_floor/concretefloor1
    provides = concrete
}
```

Anhang Konfigurations- und Netzwerktools

Beispiel-Weltkonfiguration dschungel.world

Das folgende Listing zeigt eine Weltkonfiguration, wie sie von uns während der mira-Veranstaltung im Lichthaus verwendet wurde (bereinigte Version):

```
world {
  // --== OUR JUNGLE WORLD ==--
  label = dschungel
  map   = maps/dschungel_v5.bsp
  path  = maps/dschungel5.path
}

skybox {
  // --== NICE SKY BOX WITH MOUNTAINS ==--
  label = valley
  left  = textures/dschungel/valley_lf.tga
  right = textures/dschungel/valley_rt.tga
  front = textures/dschungel/valley_ft.tga
  back  = textures/dschungel/valley_bk.tga
  up    = textures/dschungel/valley_up.tga
  down  = textures/dschungel/valley_dn.tga
}

creature {
  // --== SPECTATOR #1 ==--
  label    = camera
  class    = micado_spectator
  waypoint = 33
  campos   = 0
  config   = spectator.cfg
}
```

```
creature {
  // --== CREATURE #1 ==--
  label       = tiger02
  class       = micado_testcreature
  model       = models/players/creatures/tiger/tiger2
  skin        = default
  config      = models/players/creatures/tiger/tiger2/creature.cfg
  waypoint    = 27
  orientation = 0.0 0.0 0.0
}

creature {
  // --== CREATURE #2 ==--
  label       = florifant01
  class       = micado_testcreature
  model       = models/players/creatures/florifant/florifant
  skin        = default
  config      = models/players/creatures/florifant/florifant/creature.cfg
  waypoint    = 21
  orientation = 0.0 0.0 0.0
}
creature {
  // --== CREATURE #3 ==--
  label       = florifant02
  class       = micado_testcreature
  model       = models/players/creatures/florifant/florifant2
  skin        = default
  config      = models/players/creatures/florifant/florifant2/creature.cfg
  waypoint    = 33
  orientation = 0.0 0.0 0.0
}
creature {
  // --== CREATURE #4 ==--
  label       = florifant03
  class       = micado_testcreature
```

```
  model       = models/players/creatures/florifant/florifant3
  skin        = default
  config      = models/players/creatures/florifant/florifant3/creature.cfg
  waypoint    = 2
  orientation = 0.0 0.0 0.0
}

creature {
  // --== CREATURE #5 ==--
  label       = tiger01
  class       = micado_testcreature
  model       = models/players/creatures/tiger/tiger
  skin        = default
  config      = models/players/creatures/tiger/tiger/creature.cfg
  waypoint    = 12
  orientation = 0.0 0.0 0.0
}
creature {
  // --== CREATURE #6 ==--
  label       = tiger03
  class       = micado_testcreature
  model       = models/players/creatures/tiger/tiger3
  skin        = default
  config      = models/players/creatures/tiger/tiger3/creature.cfg
  waypoint    = 44
  orientation = 0.0 0.0 0.0
}

creature {
  // --== CREATURE #7 ==--
  label       = Schildi
  class       = micado_testcreature
  model       = models/players/creatures/schildi/schildi
  skin        = default
  config      = models/players/creatures/schildi/schildi/creature.cfg
```

```
  waypoint    = 19
  orientation = 0.0 0.0 0.0
}

creature {
  // --== CREATURE #8 ==--
  label       = Spinne
  class       = micado_testcreature
  model       = models/players/creatures/spinne/spider-slow
  skin        = default
  config      = models/players/creatures/spinne/spider-slow/spider-slow.cfg
  waypoint    = 41
  orientation = 0.0 0.0 0.0
}

creature {
  // --== CREATURE #9 ==--
  label       = schnecke_1
  class       = micado_testcreature
  model       = models/players/creatures/schnecke/schnecke
  skin        = default
  config      = models/players/creatures/schnecke/schnecke/creature.cfg
  waypoint    = 99
  orientation = 0.0 0.0 0.0
 }
```

Umgebungsvariablen für die Madness Engine

Das folgende Listing zeigt die für den Betrieb der Madness Engine und unserer Cave-Anwendung erforderlichen Umgebungsvariablen. Die hier abgedruckten Einstellungen entsprechen den von uns im Rahmen des Projekttages und der Veranstaltung im Lichthaus verwendeten (hier in bereinigter und kommentierter Fassung). Sie sind auf einem Linux-System in der Datei `/etc/profile` für alle Benutzer, oder in der Datei `~/.bashrc` für jeden Benutzer einzeln, vorzunehmen.

```
############################################################
# Madness VR Engine                                        #
# Copyright 2001,2002 by Martin Faust                      #
# Creature Extensions by the MiCaDo project 2002-2004      #
############################################################

# Setup the base madness directory
export MADNESS=$HOME/madness

# Setup the search path for files
MADNESS_PATH=.:$MADNESS/shared:$MADNESS/idSoftware
MADNESS_PATH=$MADNESS_PATH:$MADNESS/idSoftware/models/players
export MADNESS_PATH

# Setup log level: fatal, warn, note, info or debug
export MADNESS_LOG_LEVEL=note

# If you want to use a log file
#export MADNESS_LOG_FILE=/tmp/madness.log

# Where to find the shared library
export LD_LIBRARY_PATH=$MADNESS/engine

# Limit textures to a given extend
# If your graphics card has only a little amount of texture memory then
# enable the texture limit to prevent texture paging which is really slow.
# Use only values of power of two !!!
```

```
#export MADNESS_TEXTURE_LIMIT=128

################################################################
# Madness Quake Settings                                       #
################################################################

# The base Quake directory
export MADNESS_QUAKE_BASE=$MADNESS/idSoftware

# Sets the overall brightness (0..255)
export MADNESS_QUAKE_BRIGHTNESS=32

# Sets the gamma correction
export MADNESS_QUAKE_GAMMA=2.2

# Sets the range for the sounds
export MADNESS_QUAKE_SOUND_RANGE=1500.0

# Enable(=1)/Disable(=0) level models (experimental)
export MADNESS_QUAKE_MODELS=1

################################################################
# Madness MiCaDo Extensions Settings                           #
################################################################

# MiCaDo Sound Setup:

# enable or disable level sounds specified in bsp file.
# must be the same for all computers. values are:
#  'network' or '1': enable network-synchronized level sounds
#  'local'         : use local level sounds (deprecated old implementation)
#  'no' or '0'     : disable all level sounds (default)
export MADNESS_QUAKE_SOUNDS=network
```

```
# set sound model according to projection mode.
# must be the same for all computers. values are:
#  'cave_portrait' : cave setup, projection rotated 90 degrees clock-wise,
#                    speakers on top and bottom of each screen
#  'cave_landscape': cave setup, projection not rotated,
#                    speakers on top and bottom of each screen
#  'desktop'       : single desktop pc setup, display not rotated,
#                    headphones or speakers on left and right of display
export MADNESS_SOUND_MODEL=cave_portrait

# angle in degrees between the normal vector of *this* computer's
# screen and the normal vectors of the adjacent screens equipped
# with speakers. syntax is "<left> <right> <bottom> <top>". this
# only has an effect when a cave sound model is selected.
export MADNESS_SOUND_ANGLES="60.0 60.0 180.0 180.0"

# swap left and right channel of stereo output on *this* computer.
# values: '1' or 'yes' for swapped channels, '0' or 'no' otherwise.
export MADNESS_SOUND_INVERSE_STEREO=0

# default volume for each software mixer channel (range 0..127):
# (will be scaled by sound volumes provided in creature config files)
export MADNESS_SOUND_CHANNEL_VOL=50.0
```

Abbildungsverzeichnis

1.1 Cave: Unser sechsseitiger Cave 4
1.2 Reality-Rirtuality Continuum: Definition von Mixed Reality bezogen auf das Reality-Virtuality Continuum von Milgram aus [MC99] 5

3.1 BSP-Bäume: Eine Beispielszene und der zugehörige BSP-Baum nach [Wat02] 13
3.2 Netzwerkarchitektur: Ein Beispiel für einen Netzwerkaufbau 16

4.1 Cave-Gerüst mit Dach 20
4.2 Beamerständer wird von einem Projektmitglied justiert 22
4.3 Floß 23

5.1 Schaltplan der Reedkontakte 35
5.2 A/D-Werte im zeitlichen Verlauf 36
5.3 Bereiche der einzelnen Reedgruppen 36
5.4 Klassendiagramm 37

7.1 Beispielweg, erstellt mit Hilfe von Rasterung und Wegewahlalgorithmus 48

7.2 Navigationsraum mit Wegpunkten (nach [Toz02]) . . . 50
7.3 Navigationsraum mit Navigationsgitter (nach [Toz02]) . . . 50
7.4 Wegvorverarbeitung mit dem `pathtool` . . . 53
7.5 Beispiel Path File *jungle.path* . . . 56
7.6 Einfaches Beispielszenario zur Wegewahl . . . 58

8.1 Zwei Kreaturen, deren Wege sich kreuzen . . . 69
8.2 Eine Kollision zweier Kreaturen wird über deren Abstand zueinander bestimmt . . . 69
8.3 Die beiden Kreaturen kollidieren nicht, falls die Kreatur *B* sich schnell genug fortbewegt . . . 71
8.4 Berechnung des Ausweichweges der Kreatur *A* über den nahesten Punkt auf seinem ursprünglichen Weg zur Kreatur B . . . 72
8.5 Der neue Weg der Kreatur *A* . . . 72
8.6 Realisierung der Berechnung des Ausweichweges der Kreatur *A* . . . 73
8.7 Der neue Weg der Kreatur *A* in unserer Realisierung . . . 73
8.8 Zwei Kreaturen bewegen sich auf dem gleichen Weg direkt aufeinander zu . . . 74
8.9 Die Kreatur *A* kann nicht ausweichen, da sie ihrem Zielwegpunkt zu nahe ist . . . 74
8.10 Berechnung des Ausweichweges mit einem zur Kreatur *A* zu nahen Wegpunkt . . . 75
8.11 Ein Ausweichpunkt muss in einem bestimmten Bereich liegen, sonst wird der Wegzielpunkt direkt angelaufen . . . 75
8.12 Ein weiteres Beispiel der Berechnung eines Ausweichweges . . . 76
8.13 Bereich eines moeglichen Ausweichpunktes fuer das vorherige Beispiel . . . 76

9.1 Amplitude Panning: Stereo- und Cave-Konfiguration von oben betrachtet (linke Seite nach [Pul01]) . . . 80
9.2 Amplitude Panning im Cave: Veranschaulichung der Gain-Funktionen . . . 82

10.1 Schematische Darstellung des Arbeitsablaufes der Levelmodellierung . . . 88

10.2 Arbeitsumgebung in GTK-Radiant . . . 91
10.3 Zeichnung des Florifanten . . . 94
10.4 Florifant im Profil mit Splines . . . 96
10.5 Florifant mit angezeigten Splines . . . 97
10.6 Fertig modellierter Florifant . . . 98
10.7 Schneckenkopf mit Low-Polygon-Modellierung . . . 99
10.8 Extrudieren des Schneckenhalses . . . 99
10.9 Florifant mit Knochen . . . 100
10.10 Textur mit Schattierungen . . . 101
10.11 Positionieren des zweidimensionalen Gitternetzes . . . 102
10.12 Fertiger Florifant . . . 103
10.13 Export Dialog . . . 104
10.14 N'Phernos md3 Compiler . . . 105
10.15 Normalen drehen . . . 105

14.1 Das Projekttagsposter unseres Projekts . . . 129

15.1 Aufbau des Caves für den Projekttag . . . 133

16.1 Der aufgebaute Cave im Lichthaus . . . 139

Literaturverzeichnis

[AMH02] AKENINE-MÖLLER, TOMAS and ERIC HAINES: *Real-Time Rendering*. A.K. Peters Ltd., 2nd edition, 2002.

[Bru03] BRUNS, F. WILHELM: *Lernen in Mixed Reality*. Kompetenzentwicklung, Seiten 71–112, Mai 2003.

[DFAB03] DIX, ALAN, JANET FINLAY, GREGORY D. ABOWD, and RUSSELL BEALE: *Human Computer Interaction*. Pearson Prentice Hall, November 2003.

[DSP04] *music-dsp: mailing list archive*, May 2004.
Available at http://shoko.calarts.edu/musicdsp.

[Fru04] *Frustum Culling*, November 2004.
Verfügbar unter http://http://de.wikipedia.org/wiki/Frustum_Culling.

[MC99] MILGRAM, P. and H. COLQUHOUN: *A taxonomy of real and virtual world display integration*. In OHTA, YUICHI and HIDEYUKI TAMURA (editors): *Mixed Reality - Merging Real and Virtual Worlds*. Springer, April 1999.

[md304] *Description of md3 format*, April 2004.
Available at http://www.icculus.org/homepages/phaethon/q3a/formats/md3format.html.

[Mey03] MEYER, CARSTEN: *Draht nach draußen*. c't Magazin für Computertechnik, (8):204–206, 2003.

[OAL04] *OpenAL: Open Audio Library*, September 2004. Library source code and documentation available at http://www.openal.org.

[OGL04] *OpenGL: The Industry Standard for High Performance Graphics*, October 2004. Documentation available at http://www.opengl.org.

[PK01] PULLKI, VILLE and MATTI KARJALAINEN: *Directional quality of 3-d amplitude panned virtual sources*. In *Proceedings of the 2001 International Conference on Auditory Display*, Espoo, Finland, 2001.

[Pul01] PULLKI, VILLE: *Spatial Sound Generation and Perception by Amplitude Panning Techniques*. PhD thesis, Helsinki University of Technology, August 2001.

[Rab02] RABIN, STEVE: *A*-Geschwindigkeitsoptimierung*. In: DELOURA, MARC (Herausgeber): *Spieleprogrammierung Gems 1*. mitp, 2002.

[RN03] RUSSELL, STUART and PETER NORVIG: *Artificual Intelligence - A Modern Approach*. Prentice Hall, 2. edition, 2003.

[SDL04] *SDL: Simple DirectMedia Layer*, September 2004. Library source code and documentation available at http://www.libsdl.org.

[Str00] STROUSTRUP, BJARNE: *The C++ Programming Language*. Addison Wesley, special edition, 2000.

[Sur02] SURASMITH, SMITH: *Preprocessed solution for open terrain navigation*. In RABIN, STEVE (editor): *AI Game Programmin Wisdom*. Charles River Media, 2002.

[SWN99] SHREINER, DAVE, MASON WOO, and JACKIE NEIDER: *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley, 3rd edition, 1999.

[Toz02] TOZOUR, PAUL: *Building a near-optimal navigation mesh*. In RABIN, STEVE (editor): *AI Game Programming Wisdom*. Charles River Media, 2002.

[Wat02] WATT, ALAN: *3D-Computergraphik*. Pearson Studium, 3. Auflage, 2002.

[Wil02] WILLIAMS, CLARK: *Linux scheduler latency*. Technical report, Red Hat Inc., March 2002.

Glossar

3D-Engine

Ein Programm, welches Bilder einer 3D-Szene berechnet. Die Szenen bestehen meist aus Polygonen, die mit Texturen überzogen sind. Ein Sichtpunkt legt mit seinen Koordinaten entlang der x-, y- und z-Achse einen Punkt in der Szene fest, von dem die Betrachtung derselben erfolgt. 3.0

Aktivbox

Lautsprecher mit integrierter Verstärkereinheit. 9.4

Augmented Reality

erweiterte Realität, d.h. die Erweiterung und Bereicherung der Realität durch virtuelle Objekte oder Umgebungen. 1.2

Augmented Virtuality

erweiterte Virtualität, d.h. die Erweiterung und Bereicherung der virtuellen Realität durch reale Objekte der gegenständlichen Welt 1.2

C++

objektorientierte Programmiersprache, mit der sich effizienter, schneller Code erzeugen läßt, wie er für Echtzeitanwendungen benötigt wird. 3.0

Cave

Computer Animated Virtual Environment: Eine begehbare Konstruktion, die den Benutzer in eine virtuelle, dreidimensionale Illusionswelt versetzt. 1.0

Daemonprozess

Ein sich (auf einem UNIX-System) in Ausführung befindliches Programm, welches für den Benutzer unsichtbar im Hintergrund arbeitet. 11.2

Dynamikkompression

Reduzierung des Lautstärkeumfangs eines Klangsignals auf einen bestimmten Bereich. Alle Lautstärkespitzen oberhalb einer angegebenen Grenze (*Threshold*) werden um einen vorgegebenen Faktor (*Ratio*) reduziert oder beim *Limiting* hart auf den erlaubten Maximalwert begrenzt. Dies entspricht einem zeitweisen Leiser- und Lauterregeln des Signals. Die Zeit, die dieses dauern soll, wird für gewöhnlich über zwei Parameter *Attack* und *Release* eingestellt. 9.1

Frame

Einzelbild einer animierten Grafik oder eines Films. 3.2

Frustum Culling

Verfahren, das bestimmt, ob ein Vertex im sichtbaren Bereich liegt. Dazu wird überprüft, ob es sich innerhalb eines Kegelstumpfes (engl. Frustum) befindet, der diesen sichtbaren Bereich definiert. 3.3

hartcodiert, hard coded

Angabe bestimmter Einstellungen im Quelltext, so dass sie nicht auf einfache Art und Weise (ggf. zur Laufzeit) geändert werden können. Hartcodierte Einstellungen und Werte führen häufig zu ungünstigen, fehlerprovozierenden und schlecht wartbaren Redundanzen im Quellcode. 11.1

Intersense IS600 Tracking-System

Das Tracking-System IS600 der Firma Intersense bestimmt anhand von Ultraschall die 3D-Position von Sendeeinheiten (Ultraschallsender) an einem Gegenstand im Raum und stellt diese dann dem Computer über eine serielle Schnittstelle zur Verfügung. 5.1

IO-Warrior

Eigenname der uns verwendeten Elektronikplatine für den Anschluss elektronischer Schaltungen zum Messen, Steuern und Regeln an den USB-Port eines handelsüblichen Computers. 5.3

Kamerakoordinatensystem

Auch *Ansichtskoordinatensystem*. Koordinatensystem des $\mathbb{R}^3$, in dessen Ursprung sich die Kamera befindet und in positive oder negative z-Richtung blickt. Die Geometrie der modellierten virtuellen Welt befindet sich ursprünglich im so genannten *Weltkoordinatensystem* und wird mittels einer *Ansichtstransformation* in das Kamerakoordinatensystem überführt. Dies wird durch eine geeignete Verschiebung und Rotation der gesamten Welt erreicht. Zwecks Anzeige der Objekte auf dem zweidimensionalen Bildschirm wird die Geometrie i.d.R. anschließend unter Anwendung einer *perspektivischen Transformation* in *Bildschirmkoordinaten* überführt. 9.2

Kernel Latency, Kernellatenz

Zeit, die das Betriebssystem nach dem Eintreten eines Ereignisses (beispielsweise eines Interrupts) benötigt, um zu dem Prozess umzuschalten, der auf dieses Ereignis wartet. Siehe dazu auch [Wil02]. 9.2

Level-Designer

Als Level-Designer wird ein Gestalter bezeichnet, der die virtuellen Welten erstellt. Virtuelle Welten werden in der Spieleindustrie als Level bezeichnet, wodurch der Begriff Level-Designer geprägt wurde. 7.2

Level-Editor

Ein Level-Editor ist ein Bearbeitungsprogramm, mit dem eine virtuelle Welt erstellt werden kann. Der Editor wird Level-Editor genannt, da die Editoren meistens aus der Spieleindustrie stammen und dort die virtuellen Welten als Level eines Spiels bezeichnet werden. 7.2

Light-Map

Textur, welche die Helligkeit der einzelnen Punkte eines Polygons angibt. In der Madness Engine wichtig, da es keine Beleuchtungsberechnung zur Laufzeit gibt. 3.1

Mixed Reality

Verschmelzung und Vermischung von virtueller Realität und tatsächlicher Realität 1.2

Multicast

Betriebsmodus eines Netzwerkprotokolls (bspw. UDP), bei dem Daten gleichzeitig an mehrere Empfänger gesendet werden können, ohne dass sie vom Sender mehrfach verschickt werden müssen. 11.3

Normalenvektor, Normale

Ein Vektor $\vec{n}$, der senkrecht auf einer durch zwei Vektoren $\vec{a}$ und $\vec{b}$ aufgespannten Ebene steht. 3.1

Preprocessing-Tool

Ein Preprocessing-Tool ist ein Programm, mit dem Daten vorverarbeitet werden. Zum Beispiel kann ein Datensatz von Fehlern befreit und dann dessen Dateneinträge kassifiziert werden. Ein Preprocessing-Tool wird meist dann eingesetzt, wenn das Hauptprogramm schon sehr viele Berechnungen zur Laufzeit vornehmen muss und es sehr wichtig ist, diese Berechnung so gering wie möglich zu halten. 7.1

Projektbudget

Faktor, welcher die Qualität eines Projektergebnisses dramatisch beschränken kann, häufig aber zu ungeahnter Kreativität beflügelt. Siehe `http://www.dilbert.com`. 9.4

Reed-Kontakt

Ein kleiner Kontaktschalter, der durch magnetische Beeinflussung den Kontakt auslöst. 5.2

Rendering, rendern

Bedeutung bei uns: Berechnung eines Bildes einer beleuchteten dreidimensionalen Szene aus ihrem geometrischen Modell. 3.0

Ringpuffer

Zyklisch organisierter Speicherpuffer, welcher die Funktion einer Warteschlange hat. 9.2

Sample

Bedeutung in Kapitel 9: Digitaler Soundschnipsel. Genauer, die komplette endliche Folge der diskreten Abtastwerte eines im Rechner gespeicherten Klangs. Manchmal werden auch die einzelnen Abtastwerte selbst als Samples bezeichnet. 9.3

schalltoter Raum

Eine Hörumgebung, von deren Wände Schall nicht reflektiert wird. In einem nicht-schalltoten Raum hingegen reflektieren die von einer Klangquelle ausgesendeten Schallwellen an den Wänden und etwaigen anderen Hindernissen. Beim Hörer treffen daher einerseits unreflektierte und darüber hinaus reflektierte Schallwellen ein. Letztere sind für einen Halleffekt (*Reverb*) verantwortlich, welcher von einem schalltoten Raum nicht (bzw. nur in äußerst minimalem Maße) erzeugt wird. 9.2

Sensorgraben

Eine von uns gebaute 'High-Tech-Regenrinne', die es der Software ermöglicht, über magnetische Impulse die Geschwindigkeit und Richtung eines Ruderstabes zu ermitteln. 5.2

Sichtpunkt

Ein Ort in einer 3D-Szene, der durch seine x-, y- und z-Koordinaten spezifiziert ist und eine definierte Richtung besitzt – z.B. in Form eines Richtungsvektors oder einer Rotation. Von diesem Punkt aus erfolgt die Betrachtung der Szene, d.h. es wird z.B. beim Zeichnen überprüft, ob bestimmte Polygone von diesem Punkts aus überhaupt sichtbar sind. 3.0

Sky Box

Mit Texturen versehene Geometrie, welche in unserer Cave-Anwendung für die Darstellung des virtuellen Himmels zuständig ist. 11.1

Socket

Vom Beriebssystem bereitgestelltes Objekt, das eine Anwendung mit einem Netzwerkprotokoll verbindet. Über einen Socket können Daten an andere Sockets geschickt werden. 11.3

SPS

Abkürzung für *Speicherprogrammierbare Steuerung*, eine programmierbare elektronische Komponente zum Messen, Steuern, Regeln. 5.3

SSH

Abkürzung für *Secure Shell*. Ein Programm samt dazugehörigem Kommunikationsprotokoll, das ein Login auf (und den Datenaustausch mit) entfernten Rechnern erlaubt, wobei die Verbindung verschlüsselt wird. Das macht es für einen Angreifer schwierig, Passwörter oder vertrauliche Daten auf dem Weg zwischen den Rechnern abzuhören. 11.2

Stereosumme

Ergebnis des Zusammenmischens mehrerer Stereo-Eingangssignale (oder im Stereofeld platzierter Monosignale) auf ein einziges Stereo-Ausgangssignal, beispielsweise auf einem Mischpult. 9.1

TCP

Abkürzung für *Transmission Control Protocol*. Verbindungsorientiertes Netzwerkprotokoll, welches das tatsächliche Ankommen beim Empfänger und die Wahrung der Reihenfolge des gesendeten Datenstroms sicherstellt. 11.3

UDP

Abkürzung für *User Datagram Protocol*. Verbindungsloses Netzwerkprotokoll, mit dem einzelne Datenpakete (*Datagramme*) über ein Netzwerk verschickt werden können, wobei das tatsächliche Ankommen beim Empfänger und die Wahrung der Paketreihenfolge nicht garantiert sind. 11.0

Vertex

Eckpunkt eines Polygons. Die Position des Vertex wird durch einen Ortsvektor $\vec{v}$ bestimmt. Es besitzt zusätzlich eine bestimmte Farbe und Textur. 3.2

virtuelle Realität

Darstellung und Wahrnehmung der Wirklichkeit und ihre physikalischen Eigenschaften in einer in Echtzeit computergenerierten virtuellen Umgebung. 1.2

Das artecLab:

bildet eine experimentelle Gruppe von Wissenschaftlern, Ingenieuren und Künstlern.
Wir analysieren und erproben formale und nicht-formale Methoden der Modellierung, Produktion und Simulation.
Wir konstruieren sensorisierte Computer-Umgebungen und erforschen neue Formen der Mensch-Maschine-Interaktion.
Wir experimentieren mit der Vermischung von realen und virtuellen Welten an der Grenze zwischen maschineller Funktion und menschlicher Phantasie.
Wir sind Grenzgänger auf den Gebieten Kunst, Arbeit und Technik: Art, Work and Technology.

Der Mixed Reality Ansatz eröffnet neue Sichtweisen.
Wir modellieren mit realen Gegenständen, die eine reiche sinnliche Erfahrung mit der Widerspenstigkeit realer Phänomene vermitteln.
Wir formen virtuelle Gegenstände, die vielfältige Übersetzungen zwischen konkreten und abstrakten Sichtweisen realisieren.
Wir bauen Schnittstellen und Interfaces, die komplexe Verhältnisse zwischen der realen Welt der physischen Gegenstände und der virtuellen Informationswelt erfahrbar machen.

Die Computer-Wissenschaften und ihre mathematischen Grundlagen haben eine eigene Ästhetik.
Wir verstehen Ästhetik als Balance zwischen sinnlicher Erfahrung und verstandesmäßiger Durchdringung der uns umgebenden Phänomene.
Wir haben das Ziel, eine spielerische Erfahrung der Mensch-Maschine-Beziehung zu ermöglichen - auch jenseits der Grenzen von Rationalität, Nützlichkeit oder Effizienz.
Wir verfolgen gleichzeitig einen partizipatorischen und sozial verpflichteten Ansatz.

artecLab Paper

1. Jörg Richard, F. Wilhelm Bruns, Mensch und Maschine im Spielraum - Technische Praxis und Ästhetische Erfahrung
2. F. Wilhelm Bruns, Hyperbonds - Applications and Challenges
3. Yong-Ho Yoo, Energy Interface for Mixed Reality Design
4. Micado,Projektbericht des studentischen Projekts micado zu Mixed Reality Caves
5. Micarpet,Projektbericht des studentischen Projekts micarpet zu Mixed Reality Caves
6. Micasa,Projektbericht des studentischen Projekts micasa zu Mixed Reality Caves
7. Bernd Robben, Ralf Streibl, Alfred Tews, Mixed Reality Adventures, Bericht vom Symposium im Kino 46
8. Daniel Cermak-Sassenrath, Martin Faust,Computerspiele