Universität Frankfurt am Main
Fachbereich Biologie und Informatik
Institut für Informatik
Professur für Graphische Datenverarbeitung

Praktikum

Computergraphik mit VRML und JAVA 3D

WS 2001/02

Praktikumsbericht, Anwenderdokumentation und Technische Dokumentation von „PACMAN 3D“

I - INHALT

II - EINLEITUNG

Dieses Dokument beschreibt die Anfertigung einer softwaretechnischen Arbeit im Rahmen des Praktikums „Computergraphik mit VRML und JAVA 3D“ an der Johann Wolfgang Goethe-Universität Frankfurt.

Die anzufertigende Arbeit war die Entwicklung einer dreidimensionalen Variante des Spieleklassikers „Pacman“ mittels Java 3D. Pacman erblickte Anfang der Achtziger Jahre in Japan das damals noch zweidimensionale Licht der Computerwelt, und wurde seitdem unzählige Male auf den verschiedensten Computersystemen wiedergeboren. Seitdem hat Pacman einen regelrechten Kultstatus erlangt; so existieren zahlreiche Pacman Gedenkseiten und Pacman Chronologien im Internet.

Zur Realisierung teilten sich die 15 Praktikumsteilnehmer in fünf Gruppen ein, deren jede jeweils ein spezielles Aufgabengebiet bearbeiten sollte. Diese Gruppen trugen gemäß ihrer Tätigkeitsschwerpunkte die Namen Pacman-, Monster-, Netzwerk-, Kamera- und Labyrinthgruppe. Das vorliegende Dokument wurde von der Labyrinthgruppe verfasst, und beschränkt sich daher auch auf deren Blickwinkel.

Diese Dokumentation, der Quellcode, Bildschirmfotos und weiteres Informationsmaterial über die Software, das Praktikum, und/oder die Autoren sind erhältlich unter http://www.stormzone.de/uni/pacman3d/.

Diese Ausarbeitung wurde mit Hilfe von MS Word angefertigt, die ergänzende Präsentation mit Hilfe von MS PowerPoint. Kostenlose Betrachter für diese Dateiformate sind unter http://www.microsoft.com/office/000/viewers.htm erhältlich.

Die Labyrinthgruppe

Frankfurt am Main, im März 2002

III - PRAKTIKUMSBERICHT

1 Die Aufgabenstellung

...oder: wie alles begann!

Alles begann mit dem Vorlesungsverzeichnis des Wintersemesters 2001/02, in dem uns Prof. Dr.-Ing. Detlef Krömker die „Vermittlung von praktischen Erfahrungen im Umgang mit Graphiksystemen“ versprach.

Die Veranstaltung mit dem klangvollen Namen „Computergraphik mit VRML und JAVA 3D“ begann für uns Studenten im Herbst 2001[1] mit einer Kick-Off-Veranstaltung im Fraunhofer-Anwendungszentrum Computergraphik in Chemie und Pharmazie (Fraunhofer AGC). Auf dieser Veranstaltung wurden die 15 Praktikumsteilnehmer in fünf Gruppen eingeteilt, welche jeweils ein spezifisches Aufgabengebiet bearbeiten sollten.

Die eigentliche Aufgabe des Praktikums bestand auf vier Teilaufgaben, die bis November 2001, Dezember 2001, Januar 2001 und Anfang Februar 2002[2] erfolgreich bearbeitet und abgenommen werden mussten.

Die erste Aufgabe diente der Zusicherung von fundierten Java- und Java 3D-Kenntnissen der Teilnehmer untereinander, und musste daher von jedem Praktikumsteilnehmer individuell bearbeitet werden. Es galt, eine einfache Szene mit drei Objekten mittels Java bzw. Java 3D zu entwickeln. Online verfügbar sind z.B. die Lösungen von Martin Klossek oder die Lösungen von Fabian Wleklinski.

Die zweite Aufgabe markierte den eigentliche Entwicklungsbeginn des Pacman, und verlangte von jeder der fünf Praktikumsgruppen die Vorlage eines technischen Konzeptes für die Erreichung ihres jeweiligen Gruppenzieles. Beginnend mit Aufgabe 2 wurde nicht mehr individuell, sondern gruppenweise im Team gearbeitet.

Aufgabe drei bestand in der Programmierung des jeweiligen Gruppenzieles, beschränkte sich aber auf die softwaretechnische Umsetzung. Modellierungstechnische Aspekte blieben bislang außen vor und kamen erst in Aufgabe vier hinzu, welche die Modellierung der benötigten Java 3D-Modelle verlangte.

Nach Abnahme der vierten Teilaufgabe im Februar 2002 sollte jede der Praktikumsgruppen über eine mehr oder weniger eigenständig konzipierte und entwickelte Teilanwendung verfügen. In den folgenden Wochen galt es nun, die fünf Teilanwendungen zu einer einzigen, funktionierenden Anwendung zusammenzufügen: die Geburtsstunde von Pacman 3D! Nachdem diese Anwendung Anfang März als Ganzes noch einmal abgenommen wurde, wurde sie Mitte März vor einem Teil der Angestellten des Fraunhofer AGC präsentiert. In geselliger Runde erläuterten die einzelnen Gruppen mittels eines Folienvortrages ihre Aufgabenstellungen und Arbeitsergebnisse, und gingen dabei auch auf Probleme und Lösungen des Entwicklungsprozesses ein. Der Vortrag ging bei Speis und Trank in einen gemütlichen Pacman 3D-

[1] 22.10.2001
[2] Die genauen Daten: 12.11.2001, 10.12.2001, 14.1.2002 und 4.2.2002

Spielwettbewerb über, in dem sich Frank Bergmann (Pacmangruppe) gegen seine Kontrahenten durchsetzen konnte.

Während des Praktikums standen uns neben Prof. Dr.-Ing. Detlef Krömker auch die Mitarbeiter des Institutes mit Rat und Tat zur Seite; unser Dank geht insbesondere an[3]:

- Christian Seiler,
- Daniel F. Abawi,
- Paul Grimm und
- Tobias Breiner.

Die 4 Semesterwochenstunden aus der Ankündigung des Vorlesungsverzeichnisses stellten sich im Laufe des Praktikums eher als eine zeitliche Untergrenze heraus. Mit der Erreichung der Praktikumsziele gegen März 2002 konnten wir Praktikumsteilnehmer auf etwa zweieinhalbtausend Arbeitsstunden zurückblicken, welche Pacman 3D zugute gekommen sind.

2 Die Aufgabenteilung

...von gruppierten Pacman, Monstern und einer Kamera!

Bereits auf der Kick-Off-Veranstaltung im Oktober 2001 wurden die 15 Praktikumsteilnehmer in fünf Gruppen eingeteilt, welche die folgenden Aufgabengebiete bearbeiten sollten:

- **Pacmangruppe:**

  Modellierung mehrerer Pacman-Modelle, Import der Modelle in die Anwendung, Entwicklung des Programmcodes für die Steuerung des Pacman durch einen Benutzer im Ein- und Mehrspielermodus.

- **Monstergruppe:**

  Modellierung mehrerer Monster-Modelle, Import der Modelle in die Anwendung, Entwicklung des Programmcodes für die autonome und intelligente Steuerung der Monster durch das System im Ein- und Mehrspielermodus.

- **Netzwerkgruppe:**

  Entwicklung einer auf die Anwendung zugeschnittenen Netzwerkschicht für die Realisierung des Mehrspielermodus, sowie Entwicklung einer interaktiven Introszene zwecks optischer Einleitung in das Spiel und Festlegung von Optionseinstellungen.

- **Kameragruppe:**

  Entwicklung des Programmcodes für die Steuerung der Kamera, Bereitstellung mehrerer Ansichten durch alternative Kameras, Konzeption und Entwicklung einiger Spezialeffekte.

[3] in alphabetischer Reihenfolge

- **Labyrinthgruppe:**

  Entwicklung des Frameworks für Pacman und Monster, innerhalb dessen sich das Spielgeschehen abwickelt, sowie Bereitstellung von mehreren Leveldateien.

Zu allen obigen Praktikumsgruppen ist anzumerken, dass die Menge der Aufgabengebiete im Zuge der Entwicklung stets größer wurde. So entwickelte z.B. die Labyrinthgruppe auch einen netzwerkweiten Nachrichtendienst, XML-Leveldateien, Funktionalität für deren Serialisierung und Deserialisierung, einen grafischen Leveleditor und eine Debug-Klasse.

IV - DIE NETZWERKGRUPPE

1 Vorwort der Netzwerkgruppe

Die Aufgabenstellung der Netzwerkgruppe gliedert sich in zwei Teilbereiche, die im folgenden getrennt beschrieben werden, um Zusammenhänge aufzeigen:

1.1 Intro

Kernaufgabe des Intros ist die Menüsteuerung. Sie ermöglicht dem Benutzer, diverse Einstellungen vor Spielbeginn vorzunehmen. Weitere Aufgabenteile umfassen die Animationssequenz und die Anzeige der Credits, eine Darstellung der am Projekt beteiligten Gruppen.

Die im Intro verwendeten Modelle sollen mit Hilfe eines 3D-Modellierungstool erstellte werden. Wir habe dafür Rhinoceros 2.0 Evaluation (nachfolgend kurz Rhino) verwendet.

1.2 Netz

Die Netzwerkkomponente stellt einen Dienst zur Verfügung, der die Übertragung von Nachrichten zwischen verschiedenen Clients zur Aufgabe hat. Dieser Dienst ermöglicht die Kommunikation zwischen Monstern und Pacman. Die Implementierung folgt der Client/Server-Architektur.

2 Teilnehmer und Aufwand

Die Gruppe setzt sich aus folgenden Personen zusammen:

- Marina Tzanova (tzanova@cs.uni-frankfurt.de)
- Christian Balzer (balzer@cs.uni-frankfurt.de)
- Nicole Kaiser (nnkaiser@cs.uni-frankfurt.de)

Für das Praktikum haben wir insgesamt etwa 400 Mann-/Frau-Stunden aufgewendet. Dabei sind etwa 4.700 Zeilen Code für das Intro und das Netzwerk entstanden.

3 Entwicklung der Komponenten

3.1 Intro Beschreibung

Zu Beginn des Spiels läuft eine kurze Eingangsanimation ab, in welcher der Betrachter auf ein Raumschiff im All zufliegt:

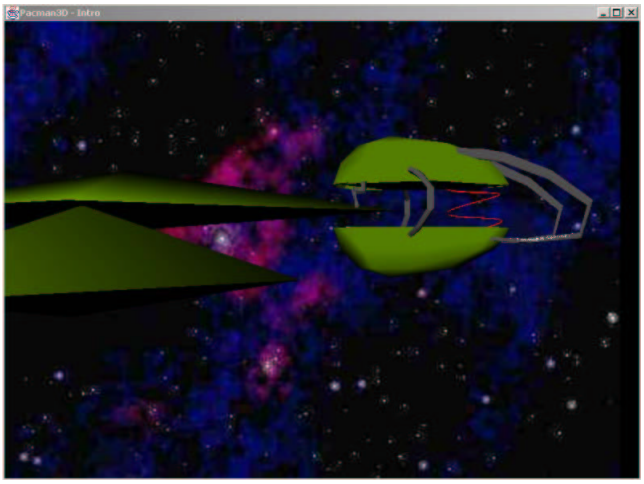

Abbildung 1 - Introanimation

Die Ansicht des Betrachter wechselt dann direkt zur Kommandobrücke des Schiffes:

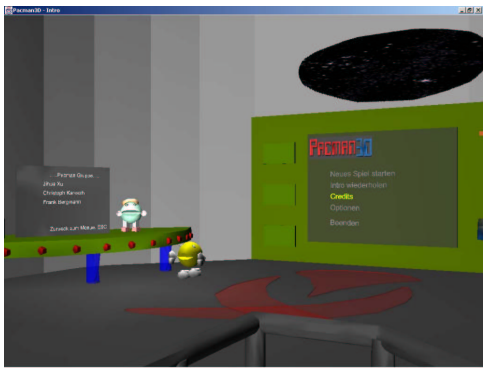

Abbildung 2 - Die Intro-Kommandobrücke

Nach einer weiteren Animation, in der man eine Übersicht auf den Introraum erhält, richtet sich der Blick direkt auf das Menü:

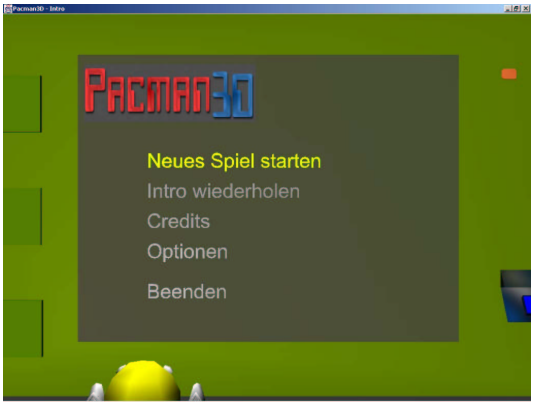

Abbildung 3 - Das Hauptmenü

Dort hat der Spieler die Möglichkeit, diverse Parameter für das Spiel einzustellen. Diese werden bei Spielstart an das Labyrinth/Game übergeben. Weitere Menüeinträge ermöglichen die Sicht auf die Credits-Animation, sowie die Wiederholung der Eingangssequenz.

3.1.1 Spielmodi

Beim Programmstart stehen der Einspielermodus und der Mehrspielermodus zur Auswahl, wobei der letztere in den Server- und den Client-Modus unterteilt ist:

- **Einspielermodus:**

  Im Einspielermodus wählt der Spieler das gewünschte Level und das Modell des Pacman aus und kann dann das Spiel beginnen.

- **Mehrspielermodus (Server):**

  Im Servermodus kann der Spieler ein Mehrspielermodus-Spiel starten, in das sich weitere Spieler einklinken können. Aus einer Liste gefundener IP-Adressen wählt der Spieler diejenige IP aus, an der sich andere Spieler anmelden können. Anschließend wird das gewünschte Spiellevel ausgewählt, sowie Spielmodus (Miteinander/Gegeneinander) und Pacman-Modell.

- **Mehrspielermodus (Client):**

  Über Eingabe der IP-Adresse kann der Spieler eine Verbindung zu einem Server herstellen. Während und nach der Eingabe der IP wird diese auf Korrektheit überprüft. Zur Eingabe sind nur Ziffern und Punkte zugelassen. Im Mehrspielermodus steht dem Spieler als Client keine Levelauswahl zur Verfügung: das Spiel startet automatisch mit dem am Server ausgewählten Level, sobald ein Pacman-Modell ausgewählt worden ist.

Eine Übersicht der Kernbereiche des Menüs liefert die folgende Grafik:

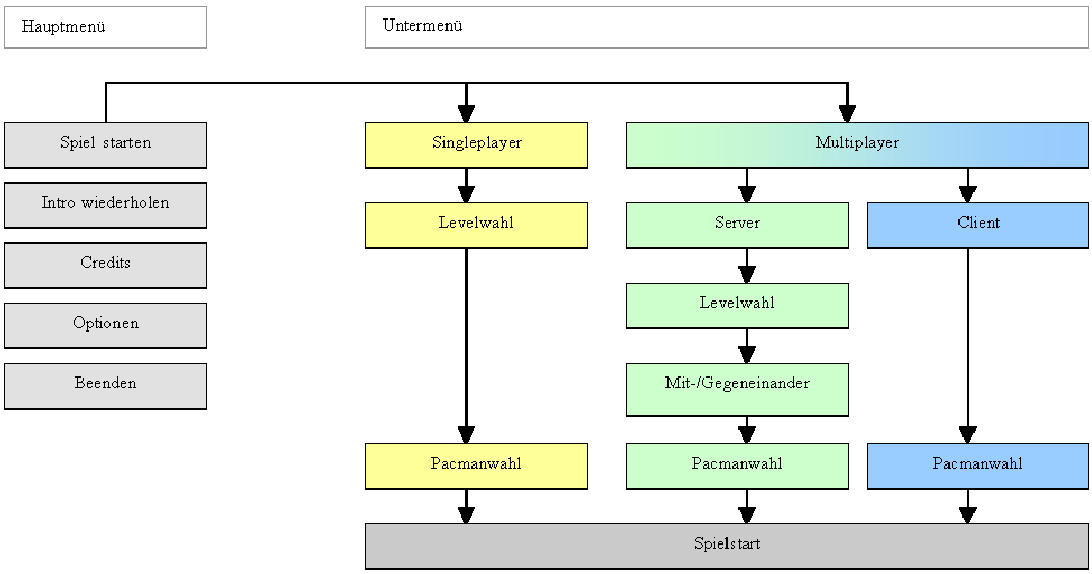

Abbildung 4 - Die Struktur des Hauptmenüs

3.2 Intro Entwicklung

Zu Beginn standen wir vor der Frage, aus welchen Teilen das Intro zusammengesetzt sein sollte. Wir entschieden uns für eine kleine Anfangsanimation und zwei Räume, einen für das Menü und einen für die Credits.

Das Konzept der zwei Räume verwarfen wir allerdings bald wieder. Ein großer Raum, der auch die Credits beinhaltet, ist im Hinblick auf den Speicherbedarf benutzerfreundlicher, als zwei Räume, die noch mit weiteren Accessoires angefüllt werden müssen, damit sie nicht so leer wirken. Im Allgemeinen ist zu sagen, dass uns der Speicherplatzbedarf noch manches Kopfzerbrechen bereiten sollte.

Beim Modellieren mit Rhino waren unsere Modelle noch sehr detailreich, besonders das Raumschiff und die Elemente, die innerhalb des Introraums positioniert sind. Als wir dann zum ersten Mal die gesamte Szene mit all ihren kleinen Feinheiten luden, stellten wir fest, dass es doch einige Zeit dauerte bis alle Elemente geladen waren: auf einem Rechner mit 400MHz-Prozessor und 384MB RAM länger als drei Minuten, auf einem Athlon 1700+ mit 512 MB RAM dagegen lediglich 12 Sekunden.

Um eine einigermaßen akzeptable Ladezeit zu erreichen, mussten wir die Anzahl der Polygone der einzelnen Elemente, sowie die Anzahl der Elemente erheblich verringern. Dies ging soweit, dass es zum aktuellen Zeitpunkt im Raum praktisch kein rundes Modell gibt, selbst der Boden ist vieleckig.

3.3 Netz Beschreibung

Die Netzkomponente sorgt für den Austausch von Datenpaketen im Netzwerkmodus. Diese gelangen als sog. `NetworkMessage`-Objekte zur Netzwerkschnittstelle, werden in ein `byte`-Array zerlegt („Java-Byte-Kodierung“) und über die Java-Netzwerkschicht übertragen. Auf der Empfängerseite wird das `byte`-Array deserialisiert und wieder in ein `NetworkMessage`-Objekt umgewandelt. Dieser, aus dem Java-Umfeld bekannte, Serialisierungsmechanismus erlaubt das einfache Verschicken von Java-Objekten über die Netzwerkschicht.

Das `NetworkMessage`-Interface wurde bewusst generisch entworfen: es implementiert lediglich das Interface `java.io.Serializable`. Dies ermöglicht der Poststelle (Labyrinth-Gruppe) quasi beliebige Objekte über die Netzwerkschicht zu übertragen, sofern sie dieses Interface implementieren:

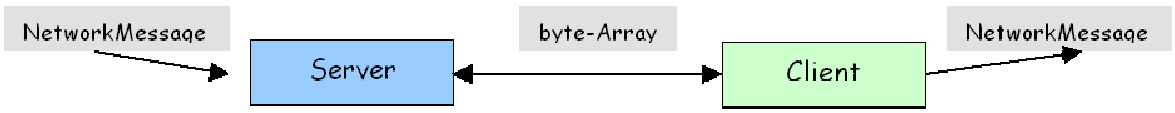

Abbildung 5 - Das NetworkMessage-Interface

Als Transportprotokoll kommt TCP/IP zum Einsatz, welches über die `java.net`-Klassen auf der Anwendungsebene angesprochen wird. Wir haben uns mit der Labyrinth-Gruppe auf TCP/IP geeinigt, da das Protokoll eine verlustfreie Datenübertragung gewährleistet und die Pakete in der abgeschickten Reihenfolge ankommen. Alle Netzwerkpakete gehen standardmäßig über Port 10001, sofern dieser in den Konfigurationsdateien (`TCPClient.config` bzw. `TCPServer.config` unter `/config`) nicht

geändert wird. Wir verwenden in unserer Implementierung einen Kanal, der als Hin- und Rückkanal fungiert.

Im Client/Server-Modus gibt es zwei Teilkomponenten:

1. **Client:**

   Über ein Handshaking-Protokoll autorisiert sich ein Client an einem Server. Von diesem kann er bis zum Abbruch der Verbindung Nachrichten empfangen, sowie Nachrichten an ihn senden. Das Handshaking umfasst zwei Phasen: Ein Client sendet eine Handshaking-Anforderung an den Server. Dieser überprüft die enthaltenen Informationen (einfacher ID-String) und sendet eine Bestätigungsmeldung (im Falle einer akzeptierten Verbindung) oder eine Abweisungsmeldung (z.B. wenn die maximale Anzahl von verbundenen Clients erreicht ist). Nach einer positiven Rückmeldung entnimmt der Client aus dieser seine Server-ID (um sich nach einer Verbindungsunterbrechung wieder am Server anmelden zu können) und schaltet in den Datenmodus, in dem er auf Nachrichten wartet.

2. **Server:**

   Am Server melden sich einzelne Clients an. Schicken die Clients Nachrichten an den Server, werden diese zur registrierten Poststelle des Labyrinths weitergeleitet. Weiterhin kann die Poststelle Nachrichten an alle Clients schicken.

3.4 Netz Entwicklung

Zu Beginn der Entwicklung hatten wir auch UDP als Transportprotokoll im Blick. Dies warf aber schon im Anfangsstadium Probleme auf (siehe 4.4.1 UDP).

Das erste Netzwerkimplementierung ermöglichte das Versenden von Zeichenfolgen, die über eine Konsole eingegeben wurden. Es konnten Strings an den Server geschickt werden, dieser leitete sie dann an Clients weiter, die beim ihm registriert waren.

Realisiert wurde die Netzwerkschicht aus drei Kernklassen: Die Klasse `TCPClient` ist die Implementierung des Clients und enthält zudem noch Methoden zum Umwandeln der Nachrichten in einen Bytestrom, sowie einen Registrierungsmechanismus für die Weiterleitung der Nachrichten (Observer-Pattern). Von dieser Klasse wurde `TCPServer` abgeleitet, um die allgemeine Funktionalität in der Serverimplementierung weiter benutzen zu können. Diese Klasse dient dem Versand von `NetworkMessage`-Objekten. Die Datenpakete von den Clients wurden von einer dritten, der `TCPServerThread`-Klasse, entgegengenommen und an den `TCPServer` weitergeleitet. Zudem existiert noch ein Container für Clientverbindungen, der durch die Klasse `ClientList` implementiert wurde.

3.5 Zusammenspiel

Wählt der Spieler im Intro den Mehrspielermodus, so erzeugt das Intro eine Netzwerk-Instanz im Client- oder Server-Modus. Diese wird vor Spielbeginn an die Poststelle des Labyrinths weitergegeben.

Einen groben Überblick der Komponenten Intro und Netz zeigt das folgende Schaubild:

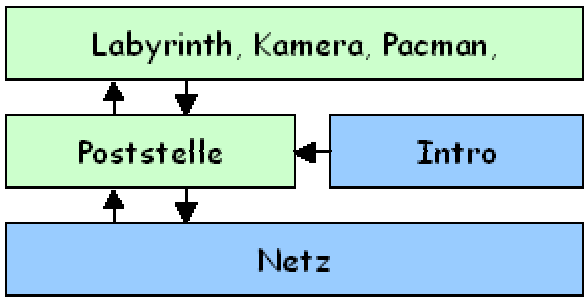

Abbildung 6 - Zusammenspiel Labyrinth, Intro, Netz und Poststelle

4 Probleme

Abschließend möchten wir noch eine Aufzählung einiger Probleme geben, die während der Entwicklung des Spiels in unsere Gruppe aufgetreten sind. Einige Probleme konnten wir lösen, andere mussten wir umgehen.

4.1 Java3D-Loader

Es hat uns etwas Mühe gekostet, einen geeigneten Modell-Loader zu finden, der unseren Anforderungen entsprochen hat. Das Problem lag in der Exportierung der mit Rhino erzeugten Modelle. Bei gewissen Loadern treten in Java Probleme mit den Lichtquellen auf. Dies äußert sich in der Nicht-Sichtbarkeit der Modelle. Wir haben uns letztlich für den Loader von Starfire Research (Loader3DS) entschieden. Die in Rhino erzeugten Modelle werden in 3ds-Dateien exportiert und können durch den Starfire-Loader, ohne Probleme in eine Java-Szene importiert werden. Dabei ist jedoch zu beachten, dass die verwendeten Dateinamen der Texturen im 8+3-Format vorliegen.

4.2 Interpolatoren der Introszene

Auch die im Introraum ablaufende Animation des Pacman warf einige Probleme auf. Die Bewegung des Pacman sollte jederzeit unterbrochen werden können, um dem Spieler ein wiederholtes Betrachten der Animation zu ersparen ;-) Dagegen musste sie auch zu jedem Zeitpunkt wieder startbar sein.

Das Problem haben wir weitestgehend gelöst, indem wir verschiedene Zustände der Navigation eingeführt haben. Wenn das Menü aktiv ist, ist der Benutzer im Menü-Modus und kann die Menüpunkte anwählen. Wenn sich der Spieler die Credits ansehen möchte, wählt er den entsprechenden Menüpunkt aus. Die Navigation wechselt in den „Menu-to-Credits-Modus“. Zu diesem Zeitpunkt ist keine Steuerung innerhalb des Menüs mehr möglich. In dem aktuellen Modus werden durch das Drücken der Escape-Taste alle laufenden Interpolatoren deaktiviert, die Kamera nimmt ihre Position direkt vor den Credits ein und die Navigation wechselt in den „Moving-Credits-to-Menu-Modus“. Durch Drücken der F1-Taste wechselt die Navigation wieder in den Menü-Modus und der Interpolator, der den Weg von den Credits zum Menü beschreibt, wird

aktiviert. Ein Erneutes Drücken von ESC stoppt alle Interpolatoren und die Kamera positioniert sich direkt vor dem Menü.

Zu Anfang haben wir die Reaktion auf das Drücken einer Taste so implementiert, dass immer nur der Interpolator deaktiviert wird, der für die aktuelle Strecke zuständig ist. Dies hat aber Probleme in der Form aufgeworfen, dass beim mehrmaligen oder abwechselnden Drücken von F1 und/oder ESC zwei Interpolatoren gleichzeitig aktiv waren. Das Pacman-Modell sprang dann unkontrollierbar zwischen den verschiedenen Wegstrecken hin und her.

4.3 Darstellung des Menüs

Gelegentlich kommt es zu einem Zerfall der Menüstruktur. Dies äußert sich in im Raum willkürlich hängenden Buchstaben, die dann im Menü fehlen. Es kann auch vorkommen, dass im Menü fehlende Buchstaben sich mit dem Pacman bewegen, wenn dessen Animation abläuft. Während dieser Ereignisse funktioniert die Steuerung des Menüs allerdings ohne Probleme. Aktive Menüpunkte werden farbig hervorgehoben und das Auswählen eines Punktes ruft auch die richtige Menüseite auf. Dies wirft weiter keine Probleme beim Starten des Spiels auf, es wird nur leider schwierig Menüpunkte auszuwählen, wenn die Menüstruktur soweit zerfallen ist, dass man kein sinnvolles Wort erkennen kann. Für dieses Phänomen habe wir leider keine Erklärung und damit auch keine Lösung gefunden.

Beim Verwenden von Text3D ist darauf zu achten, dass Java3D gelegentlich den letzten Buchstaben eines Textstrings abschneidet. Dies kann man vermeiden, indem man jeden String mit einem Leerzeichen enden lässt.

Die Positionierung des Textes in der Anfangssequenz warf Probleme auf, weil der Text auf verschiedenen Rechnern jeweils etwas anders positioniert war. Wir mussten verschiedene Werte auf unterschiedlichen Rechnern testen, um auf allen Rechnern eine akzeptable Darstellung zu gewinnen.

4.4 Netzwerk

4.4.1 UDP

Wie schon erwähnt, hatte wir zu Beginn auch UDP als Transportprotokoll im Blickfeld. Beim Verschicken von Strings über eine Konsole zeigte sich schon ein Nachteil: Die Zeichenketten kamen nicht immer an. Das hätte später im Spiel Probleme aufgeworfen, wenn z.B. ein Pacman stirbt und diese Nachricht nicht bei allen Clients ankommt. Es kam auch vor, dass die Strings nicht in der richtigen Reihenfolge ankamen. Hätten wir UDP verwendet, wäre noch die Implementierung einer Routine nötig gewesen, die feststellt, ob die Pakete in der richtigen Reihenfolge ankommen, und gegebenenfalls nicht-angekommene Pakete nochmals verschickt.

V - DIE PACMANGRUPPE

1.1 Vorwort der Pacmangruppe

Dieses Dokument dient als Dokumentation zu den Aufgaben der Pacman Gruppe des Praktikums Computergrafik WS 2001 / 2002. Ziel des Praktikums war es das Spiel Pacman, mittels Java 3D[4] und JAVA neu zu erschaffen. Dabei sollten die zu durchlaufenden Labyrinthe 3-dimensional sein. Zunächst wurden die 15 Praktikumteilnehmer in Kleingruppen eingeteilt. Diese Dokumentation schildert die Arbeit aus Sicht der Pacman Gruppe. Mitglieder der Pacman Gruppe sind:

- Frank Bergmann (bergmann@cs.uni-frankfurt.de),
- Christoph Karwoth (karwoth@cs.uni-frankfurt.de) und
- Jihua Xu (xu@cs.uni-frankfurt.de).

Die Aufgabe unserer Gruppe war es zu diesem Projekt einen Spieler-Charakter zu erschaffen, der den folgenden Ansprüchen genügen sollte:

- Anpassbarkeit und Erweiterbarkeit,
- Tauglichkeit für den Mehrspielermodus,
- Steuerbarkeit durch Tastatur,
- Verschiedenes Aussehen

In diesem Handbuch soll nun Schritt für Schritt dargelegt werden, wie unsere Gruppe dabei vorgegangen ist und welche Lösungen geschaffen wurden. Zum Schluss soll dann noch darauf eingegangen werden, welche Probleme es bei der Lösung der Aufgabe gab.

Abbildung 7 - Der Pacman in Aktion!

[4] Siehe auch http://java.sun.com

1.2 Pacman Entwicklungskonzept

1.2.1 Definition – Was ist der Pacman?

In einer ersten Phase des Projektes, definierten wir, was der Pacman denn für Eigenschaften haben sollte. Dieser Abschnitt soll darüber Auskunft geben, was für uns der Pacman eigentlich ist.

Zunächst machten wir uns Gedanken, über die Eigenschaften, die der Pacman haben sollte. (Die Eigenschaft der Steuerung findet man in der Übersicht zunächst nicht. Sie ist eher als Kommunikations-Ergebnis zu verstehen und wird später ausführlich beschrieben).

- **Aussehen:**

  Es entstanden verschiedenste, Modelle, welche bei der Konstruktion des Pacman ausgewählt werden können. Damit wird dann eine `TransformGroup` erstellt, welche dann später in das Labyrinth integriert wird. Über das Pacman Objekt werden später die Positionen verändert und die Animationen gestartet.

- **Punkte:**

  Es gibt zwei Arten Punkte zu sammeln. Entweder durch das Aufsammeln von Items, oder aber durch das Umbringen eines Monsters. Nach einer Veränderung des Punktestands wird eine Nachricht, an das Labyrinth gesandt, welche den Punktestand dann visualisiert.

- **Leben:**

  Natürlich verfügt der Pacman auch über eine bestimmte Anzahl an Leben. Zu Beginn des Spieles wird er mit 5 Leben definiert. Wird der Pacman von einem Monster oder gegnerischem Pacman umgebracht, verliert er ein Leben. Durch das Aufsammeln entsprechender Items, kann sich die Lebensanzahl auch wieder erhöhen. Ähnlich wie bei der Punkteverwaltung wird auch hier bei einer Änderung eine Nachricht an das Labyrinth gesandt, so dass die Anzeige aktualisiert werden kann. Sind alle Leben aufgebraucht, so wird das Spiel beendet, auch in diesem Fall erhält das Labyrinth eine entsprechende Nachricht. Im Mehrspielermodus existiert keine Einschränkung der Lebensanzahl. In diesem Fall werden die Anzahl der Tode gezählt.

- **Inventar:**

  Läuft der Pacman über ein Item, so versucht er zunächst es auszuführen. Ist es jedoch nicht ausführbar, so steckt er es in seine Taschen und hebt es für eine spätere Verwendung auf. Wie auch bei Punkten und Leben wird jede Veränderung dem Labyrinth mitgeteilt.

- **Spielmodi:**

  Schließlich besitzt der Pacman verschiedene Spielmodi. Zum einen ein Unverwundbarkeitsmodus, in den der Pacman nach Verwendung eines Items gerät. Des Weiteren existiert noch ein „Flugmodus“. Dieser wird durch das Labyrinth aktiviert, je nach dem, ob in einem Level Gravitation vorhanden ist, oder nicht. Ein letzter Spielmodus schließlich beeinflusst das Verhalten der Pacman untereinander. Entweder sie spielen miteinander, und beeinflussen sich nicht gegenseitig. Bei

einem Spiel gegeneinander wird jeder gegnerische Pacman als feindlich angesehen und er kann ebenso wie Monster getötet werden.

Somit haben es alle geplanten Eigenschaften bis in die endgültige Spielversion geschafft. In späteren Abschnitten werden einzelne Eigenschaften noch ausführlicher beschrieben.

Im Folgenden das Entwicklungskonzept des Pacman noch einmal als Schema:

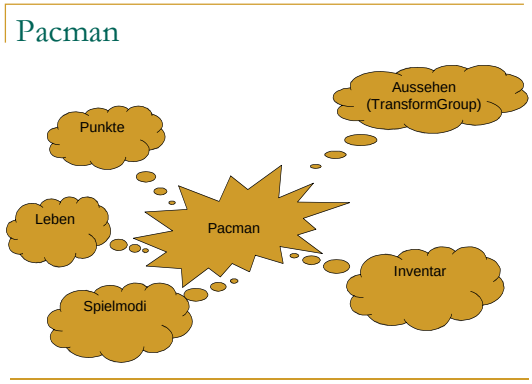

Abbildung 8 - Pacman-Konzept

1.2.2 Steuerung

Dieser Abschnitt beschreibt die Entwicklung der Steuerung des Pacman. Dazu fällt sowohl die eigentliche Steuerung des Spielercharakters, als auch die Kommunikation mit weiteren Netzwerkspielern, dem Labyrinth und den Monstern.

Die Situation, in der sich der Pacman befindet ist die folgende. Der Pacman wird zu Beginn des Spieles an eine Position innerhalb des Labyrinths versetzt. Zusätzlich zur Position erhält der Pacman auch noch eine Blickrichtung, die angibt, in welche Richtung der Pacman zu Beginn des Spieles schaut. (Als weitere administrative Parameter wird auch noch angegeben, welches Modell geladen werden soll, und welcher Spielmodus gewählt wurde, darauf wird jedoch später noch genauer eingegangen).

An dieser Startposition angekommen, befindet sich der Spieler in einer diskreten Welt. Ihm stehen theoretisch 6 mögliche Richtungswechsel zu Verfügung:

- Bewegung nach Vorne,
- Bewegung nach Hinten,
- Bewegung nach Links,
- Bewegung nach Rechts,
- Bewegung nach Oben,
- Bewegung nach Unten,

Der Benutzer bekommt von der Kamera die nötigen visuellen Informationen, nach denen er die Entscheidung für die nächste Bewegungsänderung trifft. Nun gibt es zwei Möglichkeiten:

1. entweder der vom Benutzer gewählte Weg ist ungültig, und muss abgebrochen wird, (Dieser Fall tritt ein, wenn das Level zurückmeldet, dass die gewählte Zelle nicht begehbar ist.)

2. oder die Bewegung ist gültig und muss ausgeführt werden.

Doch wie geht nun das Auswählen einer gültigen Bewegung vonstatten? Und wie soll dieses animiert werden? Diese Fragen sollen im Folgenden behandelt werden.

1.2.3 Idee: das Nachrichten-Konzept

Aus dem Ansatz heraus, dass der Spielercharakter im Mehrspielermodus simultan auf mehreren Rechnern bewegt werden muss bestand die Notwendigkeit, die Eingaben vom Pacman zu trennen. Das von uns erdachte Modell sieht nun vor, dass der Tastatur-Handler[5] mit der ID des Pacman initialisiert wird. So können die entsprechenden Tastendrücke an den Pacman gesandt werden.

Das folgende Schaubild zeigt die Kommunikation von Pacman, Labyrinth und Tastatur:

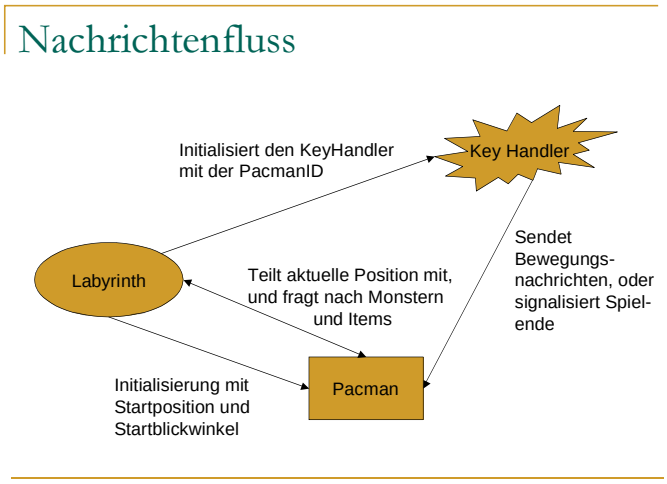

Abbildung 9 - Der Pacman-Nachrichtenfluss

Neben den eigentlichen Bewegungsnachrichten gibt es jedoch noch andere Nachrichten, auf die der Pacman reagieren muss. So schicken die Monster eine Nachricht an den Pacman, um ihn zu töten. Ebenso schickt der Pacman auch Nachrichten an Monster oder gegnerische Pacman, sollte er sie gefasst haben.

Doch wie funktioniert nun die Steuerung über mehrere Rechner? Darüber soll mehr im nächsten Abschnitt geschrieben werden.

1.2.4 Mehrspielermodus

Der Grundgedanke des Mehrspielermodus, ist der, dass es zwei verschiedene Arten von Pacman-Objekten gibt:

- „Masterpacman“
- „Schattenpacman“

Der „Schattenpacman“ ist eine leichte Abwandlung des „Masterpacman“. Er besitzt nur noch die Fähigkeit Bewegungsnachrichten zu empfangen, die Unverwundbarkeit anzuzeigen und schließlich noch die Sterbenachricht der Monster oder gegnerischer

[5] Dieser Name ist geprägt in Anlehnung an die EventHandler im Java System. Etwas anderes ist es auch nicht. Sobald eine Taste gedrückt wurde, wird diese Methode in einem Extra Thread aufgerufen.

Pacman auszuwerten. Somit kann der Schattenpacman, keine Items einsammeln, oder benutzen. Darüber hinaus kann er auch keine Monster oder gegnerischen Pacman bekämpfen. Zu jedem Masterpacman existiert auf jedem Client, der an dem Spiel teilnimmt ein Schattenpacman.

Wird nun eine Taste gedrückt, so schickt der Tastatur-Handler eine Nachricht an die ID des für ihn registrierten Masterpacman. Diese Nachricht wird nicht nur lokal verschickt, sondern auch über das Netzwerk propagiert. Somit erhalten die Schattenpacman der anderen Rechner, welche dieselbe ID haben, wie der ursprüngliche Masterpacman die Nachricht. Auf diese Art bewegt sich der Pacman sowohl auf dem lokalen Rechner, wie auch auf allen angeschlossenen Rechnern.

Doch was geschieht bei mehreren ankommenden Nachrichten? Was passiert, wenn nicht genügend Zeit zum Abarbeiten einer Nachricht vorhanden ist. Darüber soll mehr im nächsten Abschnitt geschrieben werden.

1.2.5 Idee: Das Tick Konzept

In diesem Abschnitt soll der zeitliche Ablauf beschrieben werden. Der Pacman wird (wie auch die Kamera, die Monster oder auch das Labyrinth) von der zentralen Game-Instanz in unregelmäßigen Abständen mit einem Tick versorgt. Dieser beinhaltet die Anzahl der Update Schritte, nach dem letzten Aufruf:

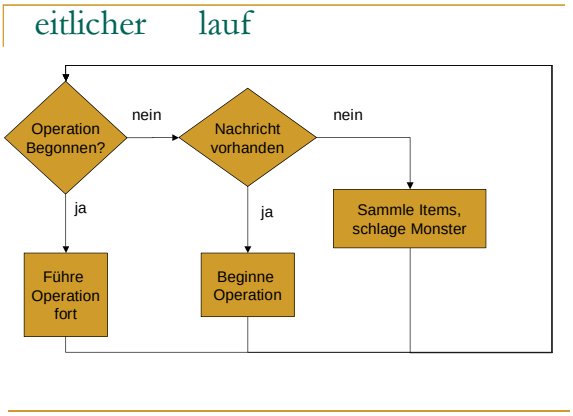

Abbildung 10 - Zeitlicher Ablauf

Somit bestimmt also die zentrale Game Instanz der Labyrinth Gruppe über das Tempo und Verhalten des Pacman. Da sie auch Monster, die Camera und sich selbst mit Ticks versorgt, ist gewährleistet, dass kein Programmteil zu viel Zeit für sich in Anspruch nimmt. Mehr über das Tick Konzept wird man dann in der Dokumentation der Labyrinth Gruppe finden.

Am Beispiel des Pacman wird als nächstes beschrieben wie das Verarbeiten der Nachrichten funktioniert. Nachrichten werden übrigens unabhängig von einem bestimmten Tick empfangen. Der Grund dafür ist einfach der, dass die User-Eingaben asynchron kommen werden. Auch hier gibt es Unterschiede zwischen Master- und Schattenpacman. Der Masterpacman filtert die ankommenden Nachrichten in der Art, dass nur die neuesten Bewegungsnachrichten akzeptiert werden. Ein Schattenpacman auf der anderen Seite muss jede Nachricht, die er bekommt akzeptieren. (Er bekommt die Nachrichten vom Masterpacman, welcher sie schon gefiltert hat).

Wird nun der Pacman mit einer gewissen Anzahl von Ticks aufgerufen, so entscheidet er zunächst, ob eine Operation[6] schon begonnen wurde, oder nicht. Wurde eine Operation schon begonnen, so wird diese fortgeführt. Im anderen Fall wird eine Nachricht aus der Message-Queue entnommen. Aufgrund des Nachrichtentyps wird bestimmt, welche Tickanzahl benötigt wird, um die Operation abzuschließen. Dabei wird auch der Zeitpunkt bestimmt, zu dem das Labyrinth über den Eintritt in eine neue Zelle informiert werden muss. (Damit wird verhindert, dass Monster und Pacman ohne Schlagabtausch „durcheinander“ laufen.) Schließlich wird die Operation mit der zur Verfügung stehenden Tickzahl durchgeführt.

Wird die Operation erfolgreich beendet wird die behandelte Nachricht gelöscht.

Sollte der Fall eintreten, dass einmal keine Nachricht vorhanden sein sollte, so überprüft der Pacman, ob an seiner Position ein Item aufgetaucht ist, oder ob sich ein Monster nähert. Ist er im Kampfmodus, startet er dann einen Angriff.

1.2.6 Tastenbelegung des Pacman

Für die Aktionen des Pacman wurden die folgenden Tasten vorbelegt[7]:

Taste	Aktion
Nach-Oben	Bewegung nach Vorn
Nach-Unten	Bewegung nach Hinten
Nach-Links	Drehung nach Links
Nach Rechts	Drehung nach Rechts
(Strg) Nach-Oben	Bewegung nach Oben
(Strg) Nach-Unten	Bewegung nach Unten
(Strg) Nach-Links	Bewegung nach Links (Sidestep Links)
(Strg) Nach-Rechts	Bewegung nach Rechts (Sidestep Rechts)
A	Im Inventar nach Links Blättern
S	Im Inventar nach Rechts Blättern
D	Ausgewähltest Item benutzen

[6] Eine Operation könnte zum Beispiel eine Drehung, eine Bewegung, oder ähnliches sein.
[7] Ursprünglich war vorgesehen, dass die Steuerung der Tasten frei einstellbar ist. Somit gibt es die entsprechenden Funktionen. Allerdings ist diese Möglichkeit nicht vom Intro aufgegriffen worden.

Taste	Aktion
Escape	Spiel beenden
Shift	Rennen

Weitere Tasten wurden von der Kameragruppe definiert, siehe VII - 2.3 Tastenbelegung der Kamera.

1.3 Modelle

Nachdem nun die Funktionsweise des Pacman dargelegt ist, soll nun noch auf die verwendeten Models eingegangen werden. Es wurden zwei verschiedene Arten von Models verwendet. Zum einen ein Modell, bei dem nur Java3D Primitiven verwandt wurden und zum anderen ein Modell, durch das Laden von 3D Modellen.

Jedes von uns verwendete Modell unterstützt zumindest zwei verschiedene Modi. Dies ist zum einen der Standard Modus, in dem der Pacman sich normal bewegt und zum anderen ein Modus, in dem der Pacman stirbt.

1.3.1 Modell durch Verwendung von Java3D Primitiven

Das erste Modell, das von uns erzeugt worden ist, besteht nur aus Java3D Primitiven. Im Standardmodus sieht man den Pacman wie folgt:

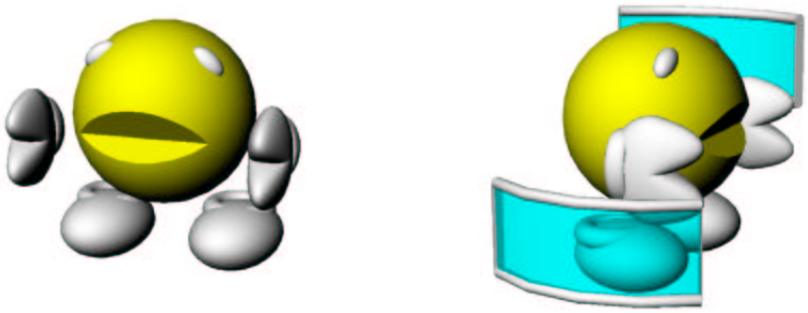

Abbildung 11 - Pacman mit Java 3D Primitiven

Dieser Pacman besteht aus den folgenden Einzelteilen:

1. **Gesicht auf linker Seite(bzw. auf rechter Seite):**

   Das Gesicht wurde durch Verwenden eines `TriangleFanArrays` realisiert. Somit besteht das „runde“ Gesicht aus 26 Dreiecken. Die Fläche wird zweimal verwendet. Einmal zeigt die Normale in die positive Z Richtung, ein anderes Mal zeigt die Normale in die negative Z Richtung. Somit wird erreicht, dass die Fläche von beiden Seiten sichtbar ist. Dieses Gesicht wird nun durch Morphing animiert. So öffnet und schließt sich der Mund des Pacman.

2. **Augen:**

   Auch die Augen entstanden durch verwenden eines `TriangleFanArrays`. Hier sind es 32 Dreiecke, die den Augen zu ihrem runden Aussehen verhelfen. Im Gegensatz zum Gesicht ist hier die Fläche geschlossen.

3. **Körper:**

   Der Körper verbindet die beiden Seitenteile miteinander. (Eigentlich sind es 4, da jedes Seitenteil doppelt vorhanden sein muss, da es sonst nicht immer sichtbar ist. Siehe auch Erklärung unter 1.). Diese Verbindungen bestehen aus 26 Vierecken.

4. **Skateboard:**

   Das „Skateboard“ auf dem der Pacman steht besteht aus einer Box.

5. **Räder:**

   Für die Räder des Skateboards fanden schließlich 4 Zylinder Verwendung welche sich alle im die X Achse(als Rotationsachse) drehen.

Der Standardmodus bleibt solange sichtbar, bis der Pacman getötet wird. Geschieht dies, so erscheint eine folgende Animation: Zunächst erscheint eine Flasche (im Folgenden auch „Spiritbottle“[8] genannt). Dann sieht man den Pacman in der Flasche verschwinden.

Die Spiritbottle besteht aus den folgenden Einzelteilen:

1. Flaschenbasis: Als Basis wird ein `Cylinder` ohne Deckel verwendet, auf den eine Textur geklebt wurde.
2. Der Flaschenhals besteht aus einer transparenten `Cone`, die es erlaubt den Pacman auch noch in der Flasche zu erkennen.
3. Den Abschluss der Flasche bilden schließlich noch 2 weitere Zylinder.

Durch die reine Verwendung der Primitiven ergibt sich ein ziemlicher Geschwindigkeitsvorteil. Diesen Vorteil erkauft man sich jedoch mit einer komplizierteren Erstellung des Models. Der Grund dafür ist der, dass kein Tool verwendet werden kann, mit dem ein Modell erzeugt werden kann.

Auch dieser, leicht „bewaffnete“ Pacman, besteht aus Java 3D-Primitiven:

Abbildung 12 - Der bewaffnete Pacman

[8] Der Name soll darauf hinweisen, dass diese Flasche die Essenz des Pacman aufnimmt.

1.3.2 Modelle durch Verwendung eines Loaders

Alle Modelle, die wir durch einen Loader erstellen lassen, liegen entweder als 3DS oder als VRML Datei auf der Platte vor. Diese Modelle unterstützen 3 verschiedene Modi:

- zum einen der Standardmodus,
- einen Unverwundbarkeitsmodus
- und schließlich einen Sterbemodus.

Im Standardmodus, sieht man den Pacman seine Füße bewegen, so dass der Eindruck des Laufens entsteht. Währendessen erscheinen im Unverwundbarkeitsmodus Schilde um den Pacman, die es keinem Gegner gestatten dem Pacman etwas anzutun. Im Gegenteil, sie verletzen sich nur selbst.

Ein Modell wird etwas anders gehandhabt. Der Name dieses Modells ist „Moriya“. Dieses Modell ist mit einer Katana[9] ausgestattet und kann sich dank der seiner hervorragenden Kampftechnik gut genug Verteidigen, so dass es keine Schilde benötigt:

Abbildung 13 - Der Pacman Moriya

Sollte es dennoch vorkommen, dass der Pacman von einem Monster erwischt wird, so wird er sterben. Dann sieht man den Pacman zusammenklappen.

1.4 Schnittstellen zu anderen Gruppen

Natürlich ist der Pacman nur ein Teil des gesamten Kuchens. Ohne die anderen Gruppen könnte er recht wenig, und würde wohl auch nie das sein, was er heute ist. In diesem Abschnitt soll es also um die Zusammenarbeit mit den anderen Gruppen gehen. Dazu wird auf jede der folgenden Gruppen kurz eingegangen:

- Labyrinth,
- Camera,
- Monster,
- Netzwerk

1.4.1 Labyrinth

Zuerst gilt es die Labyrinth Gruppe zu nennen. Ohne diese Gruppe gäbe es wohl keine Nachrichtenstruktur in der jetzigen Form. Darüber hinaus dient das Labyrinth als zentrale Anlaufstelle für alle Interaktionen mit den anderen Gruppen.

[9] japanisches Langschwert

Das beginnt schon mit der Erzeugung des Pacman. Da die Labyrinth Gruppe auch die eigentliche Spielinstanz erschafft bleibt es nicht aus, dass sie auch den Pacman nach den Introvorgaben erzeugen und dafür sorgen, dass im Mehrspielermodus der Pacman auch auf allen verbundenen Rechnern erscheint. Auch danach steht der Pacman in ständigem Kontakt mit dem Labyrinth. Entweder sendet er die jeweiligen Positionen, oder die Punkteanzeige muss aktualisiert werden. Um sich der Camera zu bedienen muss der Pacman auch den Umweg über das Labyrinth gehen, da er sonst die Instanz der Camera nicht kennen würde.

1.4.2 Camera

Eine Zusammenarbeit mit der Camera Gruppe gab es in den folgenden zwei Punkten. Zum einen war es notwendig, dass die Kameraansichten verändert werden. Dafür wurden einige Nachrichten vereinbart, die bei entsprechendem Tastendrücken gesandt werden. Auch hier soll wieder auf die Dokumentation der Kameragruppe hingewiesen werden.

Die Aufgabe der Camera Gruppe war es auch verschiedene Effekte zu Entwickeln. Ein Effekt, der in der endgültigen Version des Spiels enthalten ist, ist das Beben. Mit anderen Worten, die Camera fängt an zu Schwingen. Dieser Effekt findet seine Anwendung bei einem Ableben des Pacman.

1.4.3 Monster

Natürlich war auch eine Zusammenarbeit mit der Monster Gruppe notwendig. Hierbei ging es hauptsächlich um den gegenseitigen Schlagabtausch. Dabei einigte man sich auf eine Bestimmte Art von Nachrichten, die den Pacman bzw. das Monster zum sofortigen Ableben bringen. Um allerdings an die genaue ID und Position eines Monsters oder Pacman zu kommen, war eine Anfrage an das Labyrinth notwendig, welche als zentrale Anlaufstelle auch diese Informationen verwaltete.

1.4.4 Netzwerk

In Zusammenarbeit mit der Netzwerkgruppe entstand die Pacman Auswahl, wie sie in der aktuellen Version zu finden ist. Leider fanden nicht alle Features auch Verwendung in der endgültigen Version. So gab es Schnittstellen zur Darstellung einer Voransicht des gewählten Pacman. Auch wäre es möglich gewesen die Tastatursteuerung an die persönlichen Wünsche anzupassen.

1.5 Probleme

Natürlich treten in einem solchen Projekt immer wieder auch kleinere und größere zumeist aber zeitraubende Probleme auf. In diesem Abschnitt soll eine Auswahl dieser Probleme besprochen werden.

1.5.1 GELÖST: Interpolieren eines Vektors

Das erste Problem zeigte sich schon in einer recht frühen Praktikumphase. Die Situation war die folgende. Es gab eine erste Version von Camera, Labyrinth und Pacman. Die optimale Vorraussetzung, um die ersten Laufversuche durchzuführen. Das Ergebnis zeigte eine Linksdrehung um 90° wie beabsichtigt. Auch die Kamera drehte sich um 90°, so dass keine Grafikeffekte zu beobachten waren. Dann kam die Rechtsdre-

hung. Auch hier drehte sich der Pacman um 90° nach rechts und schaute in die richtige Richtung. Doch die Drehung der Camera zeigte, dass sich der Pacman in Wirklichkeit um 270° nach Links drehte, anstatt 90° nach Rechts. Ein kleiner Fehler der sich jedoch nicht einfach zu lokalisieren war.

Es zeigte sich schließlich, dass die von Java3D implementierte Funktion zum interpolieren eines Vektors zum nächsten nur nach der Funktion

$$(1-)v_1 + \quad v_2$$

realisiert. Nachdem die Ursache gefunden war, half schließlich endlich langes Nachdenken, um eine Lösung zu finden. (Zwischenzeitlich kam man mal auf die Idee eine Kreisgleichung mittels Sinus und Kosinus zu beschreiben und dann …).

Die letztendliche Lösung gestaltete sich recht einfach. (Man muss sie eben nur kennen):

$(1-)v_1 - \quad v_2$Aber das war sicherlich nicht das letzte aufgetretene Problem.

1.5.2 GELÖST: Das Problem der Beleuchtung

Die Verwendung der 3D Models war zwar schön und gut, doch nach einem ersten Laden der Models in das Labyrinth, sah man nur, dass man nichts sah. Das mit viel Liebe erstellte Modell war im Großen und Ganzem nur noch schwarz. Und das trotz ambienter Lichtquelle. Das Problem schien sich einfach Lösen zu lassen. Man fügte einfach eine direktionale Lichtquelle ein, und hoffte auf das Beste. Doch hier zeigte sich dann, dass unterschiedliche Grafikkarten die Lichtquellen unterschiedlich stark in Betracht zogen. Und so waren die Modelle natürlich zu hell. Ein Umsteigen auf Punktlichtquellen, und ein Wechseln des Dateiformats von 3DS auf VRML brachte schließlich das gewünschte (und endgültige) Ergebnis.

1.5.3 OFFEN: Weichheit der Bewegungen des Pacman

Seit dem wir von der Pacmangruppe zum ersten Mal gesehen hatten, wie flüssig sich die Monster bewegen können hatten wir daran gearbeitet die Bewegung des Pacman ebenso flüssig zu gestalten. Dazu wurde ein erstes Bewegungskonzept, welches darauf beruhte, den genauen Pfad zur nächsten Zelle schon vorzuberechnen, verworfen. Das aktuelle Modell berechnet zu jeder Anzahl der bekommenen Ticks die Position, an welcher der Pacman sein müsste und setzt ihn dahin. Doch nach welchem Modell man sich auch bewegte immer schien es nach der Hälfte des Weges zu ruckeln. Erst der funktionierende Netzwerkcode und damit das testen der Schattenpacman hat aufgezeigt, warum das der Fall war. Der Grund ist der folgende, anders als die Monster, die einen Pfad mit einer Länge von mehreren Zellen ablaufen, läuft der Pacman immer nur den Weg von einer Zelle zur nächsten. Auf dem halben Weg jedoch teilt er zunächst dem Labyrinth den Eintritt in die neue Zelle mit, und danach beginnt er die Items der neuen Zelle einzusammeln(oder zu benutzen). Natürlich muss beim Eintritt in eine neue Zelle auch überprüft werden, ob sich nicht ein Monster oder ein gegnerischer Pacman in ihr befindet, die es zu vertreiben gilt. Erst danach kann die Bewegung fortgeführt werden.

1.6 Schlussworte der Pacmangruppen-Mitglieder

In diesem Abschnitt sollen nun noch mal alle Mitglieder der Pacmangruppe zu Wort kommen. Auf diese Art hat jeder noch mal die Chance die Eigenen Gedanken zum Praktikum loszuwerden. So stay tuned...

Frank Bergmann:

„Der ursprüngliche Grund an diesem Praktikum teilzunehmen, war es, die Funktionsweise eines Modellierungssystems näher zu erlernen. In der Beschreibung im Semester davor hieß es genauer, dass 3DS Max[10] verwandt werden sollte. Umso größer war dann meine Überraschung, als ich mitbekam, dass der uns zur Verfügung stehende Modellierer Rhinoceros[11] sein sollte. Der Grund dafür war eine einfachere Lernbarkeit. Nur war das nicht ganz das, was ich wollte. Also bemühte ich mich in den Treffen unserer Gruppe schon früh um eine klare Aufteilung der Aufgaben. Christoph Karwoth kümmerte sich fortan um die Modellierung und Erzeugte all die Modell, die über den Loader eingeladen werden konnten, inklusive der netten Animationen im Kampfmodus, oder beim Dahinscheiden des Pacman. Darüber hinaus schrieb er die Basisklasse für die Verwaltung der Pacman-Informationen. Jihua Xu bekam die Arbeit des Erstellens weiteren Models aus den Java3D Primitiven. Später hatte er dann die Basisklasse zum Abspielen der Sounds geschrieben. Nach dieser Aufteilung entstand das Nachrichten-Konzept. In enger Zusammenarbeit mit der Labyrinthgruppe gab es dann auch recht schnell eine erste Version eines Pacman. Die nächste Zeit ging es dann um die Überlegung, was man denn nun mit den ganzen eintreffenden Meldungen macht. Eine erste Version arbeitete einfach alle Nachrichten nach dem FIFO Konzept ab. Dies war jedoch nicht allzu gern von den Leitern des Praktikums gesehen, da es so möglich wäre ganze Wegstrecken, durch schnelles Tippen schon zu puffern. So entstand dann eine Mischform. Bewegungsnachrichten[12] werden gefiltert, so dass nur noch die aktuellste Nachricht beachtet wird, andere Nachrichten werden jedoch nach wie vor nach dem FIFO Modell bearbeitet. Auf diese Art ist nun das Puffern nicht länger möglich. Nachdem dieses Problem aus der Welt geschafft war trat das Problem mit den Drehungen auf, welches oben schon beschrieben wurde. In dieser Phase entstanden etliche neue Verfahren, den Pacman zu drehen. Da ich keine Möglichkeit hatte aus einer gedrehten TransformGroup den Blickwinkel des Pacman abzuleiten, brauchte ich neben der eigentlichen Drehung der TransformGroup noch eine Interpolation für den Blickwinkel, den die Camera benötigt. Schließlich benötigte dann noch die Öffnung des Pacman gegenüber dem Intro noch einige Arbeit. Leider sind die dafür erstellten Features einer Voransicht des gewählten Pacman, oder weitere möglichen Parameter nicht in der endgültigen Version erschienen. Soweit der Rückblick über die Arbeit am Praktikum.

Nun aber erst einmal ein Wort des Dankes an alle Praktikumteilnehmer. Es ist schon erstaunlich, was wir da geschaffen haben. Ein Dank natürlich auch an unsere Praktikumleitung, die es uns ermöglichte weitgehend ungebremst etwas auf die Beine zu stellen. Besonderen Dank vor allem auch nochmals der Labyrinth-Gruppe, die in einigen Nachtstunden dafür gesorgt haben, dass der Pacman so stabil im Spiel verankert ist.

[10] Siehe http://www.discreet.com
[11] Siehe http://www.rhino3d.com/
[12] also Nachrichten, die den Pacman veranlassen sich zu bewegen oder zu drehen

Zusammenfassend lässt sich sagen, dass die Arbeit am Praktikum sehr viel Spaß gemacht hat. Es war interessant mal etwas anderes zu programmieren. Wenn ich etwas an diesem Praktikum gelernt habe, dann den engen Zusammenhang von 3DSpiele-Entwicklung und Linearer Algebra. Im Nachhinein hätte man die Lösung der Aufgabe natürlich noch weiter Planen können, die Phase des Programmierens hat einfach zu früh begonnen. So, jetzt habe ich aber genug geschrieben, vielleicht sollte ich noch ein paar Sachen am Pacman feilen … wäre doch gelacht, wenn wir keine weichere Animation des Pacman hinbekommen würden."

Christoph Karwoth:

„Es war früh und der Nebel löste sich langsam auf, als sich unsere Gruppe kurz nach Mittag zum Brainstorming versammelte. Das ging nur mühselig voran, bis wir uns einen Kaffee geholt haben. Der war auch unser ständiger Begleiter. Schon am ersten Tag entstanden viele Picassos und andere Zeichnung, die uns noch sehr lange beschäftigten. Viele Bäume mussten dran glauben, bist wir endlich Strom vergeudeten konnten und auch das kunstvolle Gekritzel dechiffrieren mussten. Doch davor mussten wir uns in der GDV-Kongresshalle mit den anderen Treffen. Diese Debatte war vom großen Vorteil. Alle Parteien waren vertreten. Alle haben ihre Schnittstellen enthüllt und alle hatten eine Vision von voll 3-D animierten Pacman, der Hechtrollen á la Tomb Raider macht, beamen und schissen wie die in Star Treck kann und ein Intro á la Herr der Ringe besitzt. Die Netzwerkstruktur sollte die Telekom Aktie noch weiter in den Keller treiben. Die Kameraführung sollte Spielberg zum Staunen bringen und Labyrinthe, die Schumacher aus Geschwindigkeitsgründen und Kurvenreichhaltigkeit fürchten würde. Und das war nur der Anfang, aber jeder fängt ja klein an.

Erstaunlicherweise haben wir nach wenigen Stunden dem Pacman das Laufen beigebracht. Wir hatten ihn zwar ins Verlies (durch Leveldesign) gepackt, aber er machte sich nichts draus. Mit lachendem Gesicht erkundete er jede Ecke. Teilweise wie Copperfield, konnte er durch die Mauern spazieren. Dies haben wir ihm aber sehr schnell abgewöhnt, da er ja irgendwann feststecken könnte.

Durch die Mailliste und CVS war unsere Kommunikation auch in jedes Dorf möglich, so als ob alle in einem Raum saßen. Man hatte beinah Platzangst.

Lifting vom Pacman hat uns unzählige Kaffeetassen gekostet. Er wurde immer besser. Auch das Designen von der Behausung des Pacman ging auch durch den feinen Leveleditor großartig. Es schien alles sich zum Besten zu wenden, doch das Ungeziefer (Monster) breitete sich aus. Das Kammerjägergeschäft boomte. Fast jeder hat einen Gefangen. Leider wie man es aus unsere Branche kennt, kann man nicht alle ausmerzen. Die Jagt geht also weiter.

Es war ein tolles Praktikum."

Jihua Xu:

„Vor einem Jahr habe ich das Plug-In für VRML herunter geladen, um die Struktur und die Syntax von VRML kennen zu lernen. Ich kam zu dem Schluss, dass zwar sehr schöne Objekte erzeugen kann, es aber zu aufwendig für komplexe Objekte ist. Da Java3D ähnlich zu VRML ist, hatte ich Zweifel an der Kompetenz der Programmiersprache in diesem Bereich. Nachdem ich einige sehr beeindruckende Szenen der GDV-Übungsaufgaben gesehen hatte, hatte ich große Interesse daran, in diesem Semester an dem Praktikum teilzunehmen um eine Antwort auf die Frage, wie man effizient komplexe Objekte erzeugen kann, zu finden. Im Laufe des Praktikums verwendeten wir

dann das Modellierungsprogramm Rhinoceros. Die Möglichkeit die Modelle zu exportieren und mittels eines Loaders in Java zu integrieren, überzeugte mich.

Vor dem Praktikum habe ich noch nie an einem größeren Projekt mitgearbeitet. Ich hatte zwar mit jemandem zusammen in einer Gruppe programmiert, durch strikte Aufgabenteilung war jedoch nicht zu viel Kontakt nötig. In diesem Praktikum: Java3D und VRML lief das anders. Sowohl eine Mailingliste als auch CVS wurden intensiv genutzt, und somit war ein besseres Zusammenarbeiten möglich. Ich erfuhr somit, wie sich eine abstrakte Problemstellung hin zu einer Lösung entwickelt.

Meine Aufgabe in der Gruppe war es einen Pacman zu modellieren. Am Anfang nutzte ich dazu die Eigenschaft Billboard, welche es ermöglichte einen zweidimensionalen Pacman zu erzeugen. Jedoch war dieser Ansatz noch nicht sehr schön. Als ich sah, dass Christoph mittels eines Loaders ein schönes Modell ins Spiel gebracht hat, dachte ich, dass ich lieber mit den Java3D Primitiven arbeite, statt wieder ein Modell mit Hilfe von Rhinoceros zu erschaffen. Somit sammelten wir Erfahrungen bei der Modellierung von beiden Seiten (Modellierung mittels Modeller und Erstellung eines Models mittels Primitiven).

Zum Schluss danke ich Allen für ihre Mithilfe am Praktikum.“

VI - DIE MONSTERGRUPPE

1 Aufgabenstellung

Die Aufgabe der Gruppe „Monster“ war die Programmierung und Modellierung der Monster (oder „Geister“) des Pacman 3D-Spiels. Das umfasste das Entwickeln von Monstern mit verschiedenen Strategien der Jagd auf den Pacman und einer Fluchtstrategie. Die Strategien sollten visualisiert werden.

Außerdem sollten die Eigenschaften der Monster während der Laufzeit einstellbar sein und es sollte verschiedene Monster-Modelle geben. Die Ereignisse „Pacman gefangen“ und „Monster gefangen“ mussten bearbeitet werden. Die Monster sollten – wie alle Komponenten des Spiels - netzwerkfähig sein und zum Spielspaß und einer flüssigen Performance beitragen.

2 Das Monster lebt - Ergebnisse

Im Spiel gibt es verschiedene Arten Monster, die sich im Labyrinth bewegen. Sie laufen entweder zufällig oder suchen gezielt nach dem Pacman, begeben sich dann dorthin und kommunizieren optional noch untereinander, um sich die Pacman-Position mitzuteilen. Alle diese Strategien haben wir auf eine Breitensuche in einem erstellten Suchbaum zurückgeführt, so dass sie – quasi als Variationen desselben Themas – auf durch bestimmte Suchtiefen oder Abbruchbedingungen gekennzeichnete Breitensuchen rückführbar sind. Falls der Pacman eine Vitaminpille gefressen hat, die ihm ermöglicht zu jagen, flüchten die Monster. Die jeweiligen Eigenschaften eines Monsters sind (auf dem Server) über das Setup-Fenster einseh- und auch einstellbar. Hier gibt es ein Feld, in dem empfangene oder versendete Nachrichten angezeigt werden, ein Feld, in dem der Status angezeigt wird und drei Schieberegler. Mit diesen sind Kommunikationsradius, Suchradius und „Intelligenz“ (d.h. die vom Monster verfolgte Strategie) einstellbar. Insgesamt gibt es drei verschiedene Monster-Modelle, die per XML leicht ausgetauscht werden können. Momentan kommen alle drei Monster im Spiel vor: der Geist des „klassischen“ 2D-Spiels, ein Modell namens Kodos und ein assimilierter „Pac-Borg“. Im Netzwerkspiel werden die Monster hauptsächlich auf dem als Server ausgewiesenen Rechner erzeugt; an die Client-Rechner werden nur Datenpakete verschickt, welche die aktuellen Positionen der Monster beinhalten.

3 Struktur der Programmierung

Wir entschieden uns dafür, unsere Programmierung mit Hilfe eines Agentenmodells zu strukturieren. Dieses besteht aus vier Modulen (Selbstbild, Kontrollmodul, Kommunikationsmodul und Response-Modul), welche den Agenten darstellen. Als Schnittstelle zum Labyrinth dient die Kommunikationsebene, über welche die gesamte Kommunikation mit den anderen Komponenten des Spiels läuft. (vgl. Abbildung 1)

Im Folgenden sollen die einzelnen Module und die sie repräsentierenden Klassen genauer erläutert werden:

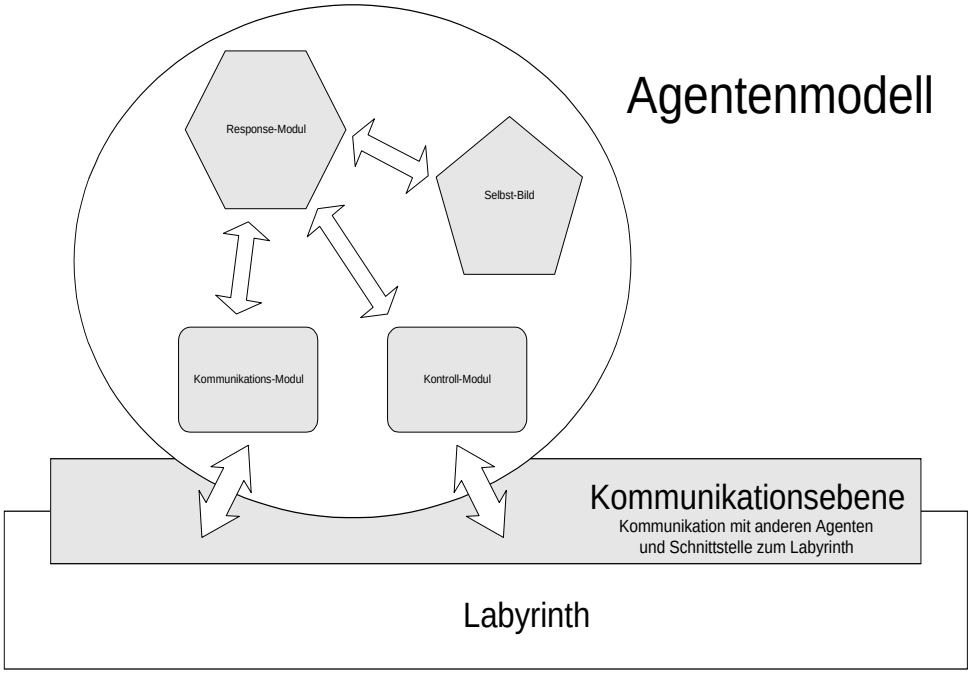

Abbildung 14: Agentenmodell und Schnittstelle zum Labyrinth

4 Die Module des Agentenmodells

Ziel war die Implementierung unabhängiger Module, welche durch Kooperation der Metapher eines Agenten-Systems gerecht werden sollten. Leider konnte aus Zeit- und Performanzgründen kein adaptives System entwickelt werden; auch andere Eigenschaften eines Agenten, beispielsweise Proaktivität, sind nicht implementiert. Durch die offene Gestaltung der Schnittstellen könnten diese in Zukunft leicht nachgerüstet werden. Das Flussdiagramm soll den Ablauf eines Game-Loop verdeutlichen.

4.1 Kontrollmodul

Das Kontrollmodul besteht aus den Klassen `MonsterControl` und `MonsterSearchTree`. Die Klasse `MonsterControl` implementiert die ausführenden Methoden des Monsters: alle Bewegungen, die Ansicht, die Suchmethoden (die Breitensuche) sind hier gekapselt. Die Aufrufe der Methoden erfolgen durch das Response-Modul, alle notwendigen Informationen zum Ausführen der Aufgaben werden über die Kommunikationsebene abgefragt.

Erwähnenswert ist hier die Struktur einer Breitensuche: erzeugt wird ein Baum variabler Tiefe und mit variabler Anzahl Kinder pro Knoten. Dieser Baum kann beliebig durchlaufen werden und stellt somit die grundlegende Funktionalität für eine Suche im Labyrinth zur Verfügung.

Die Abbildungen 2 und 3 zeigen exemplarisch verschiedene Monster-Aktionsradii aus zwei unterschiedlichen Blickwinkeln. In Abbildung 2 hat der Pacman Glück: Er befindet sich nicht im Aktionsradius (aufgehellter Bereich), das Monster wird sich sein

nächstes Ziel zufällig suchen. Im Abbildung 3 hat der Pacman ein schweres Los: Das Monster hat ihn entdeckt, die Pacman-Position befindet sich im `MonsterSearchTree`. Das Monster wird sich auf direktem Weg dorthin begeben...

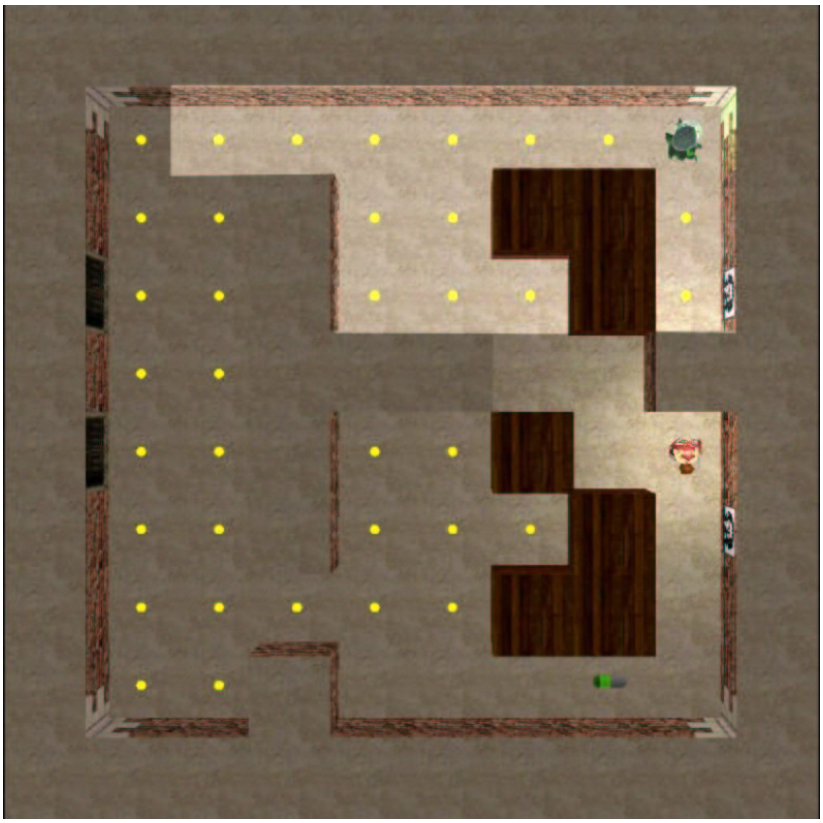

Abbildung 15: MonsterAktionsradius in 2D (Suchtiefe: 6)

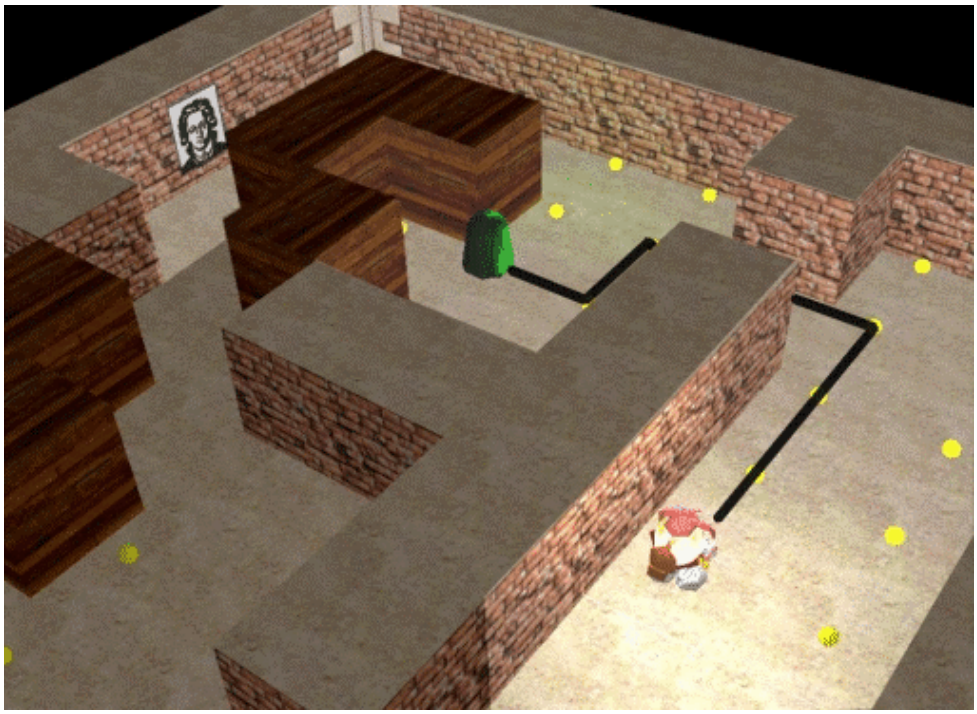

Abbildung 16: MonsterAktionsradius in 3D (Suchtiefe: 8)

4.2 Selbstbild

Das Selbstbild enthält die Eigenschaften und Möglichkeiten des Agenten, Wissen über sich und seine Umwelt. Das Selbstbild findet sich in den Klassen `MonsterAwareness` und `MonsterSetup`.

Durch das `MonsterSetup` ist es möglich, die Eigenschaften jedes Monsters während des Spiels zu verändern, was Teil der Aufgabe war. Für jedes Monster wird ein Setup-Fenster erzeugt, in dem mit Schiebereglern Kommunikationsradius (Kommunikation zwischen Monstern), Intelligenz (Strategie) und Suchradius eingestellt werden können. Dabei wechseln die Rahmen der Piktogramme, welche die jeweilige Funktionalität des Reglers symbolisieren, die Farbe (je greller, desto gefährlicher!). Rote Schrift weist darauf hin, dass das Monster sich auf der Jagd befindet, blau bedeutet Flucht. (vgl. Abb. 4 und 5)

Die Klasse `MonsterAwareness` dient als Komponente zur Speicherung und Bereitstellung von Wissen. Sie dient auch als Schnittstelle zum Setup-Fenster, in welchem vom Spieler Einstellungen vorgenommen werden können und steuert die Visualisierung des aktuellen Zustands des Monsters.

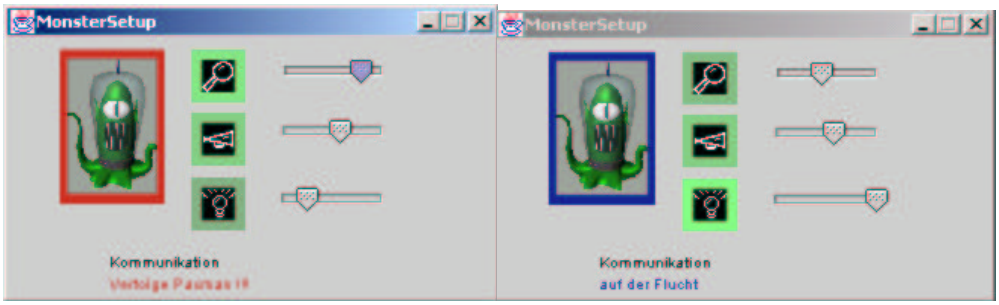

Abbildung 17: Monster Setup-Fenster (Monster auf Jagd bzw. Monster auf Flucht)

Flussdiagramm für den Durchlauf eines Game-Loops,
Monster-Module

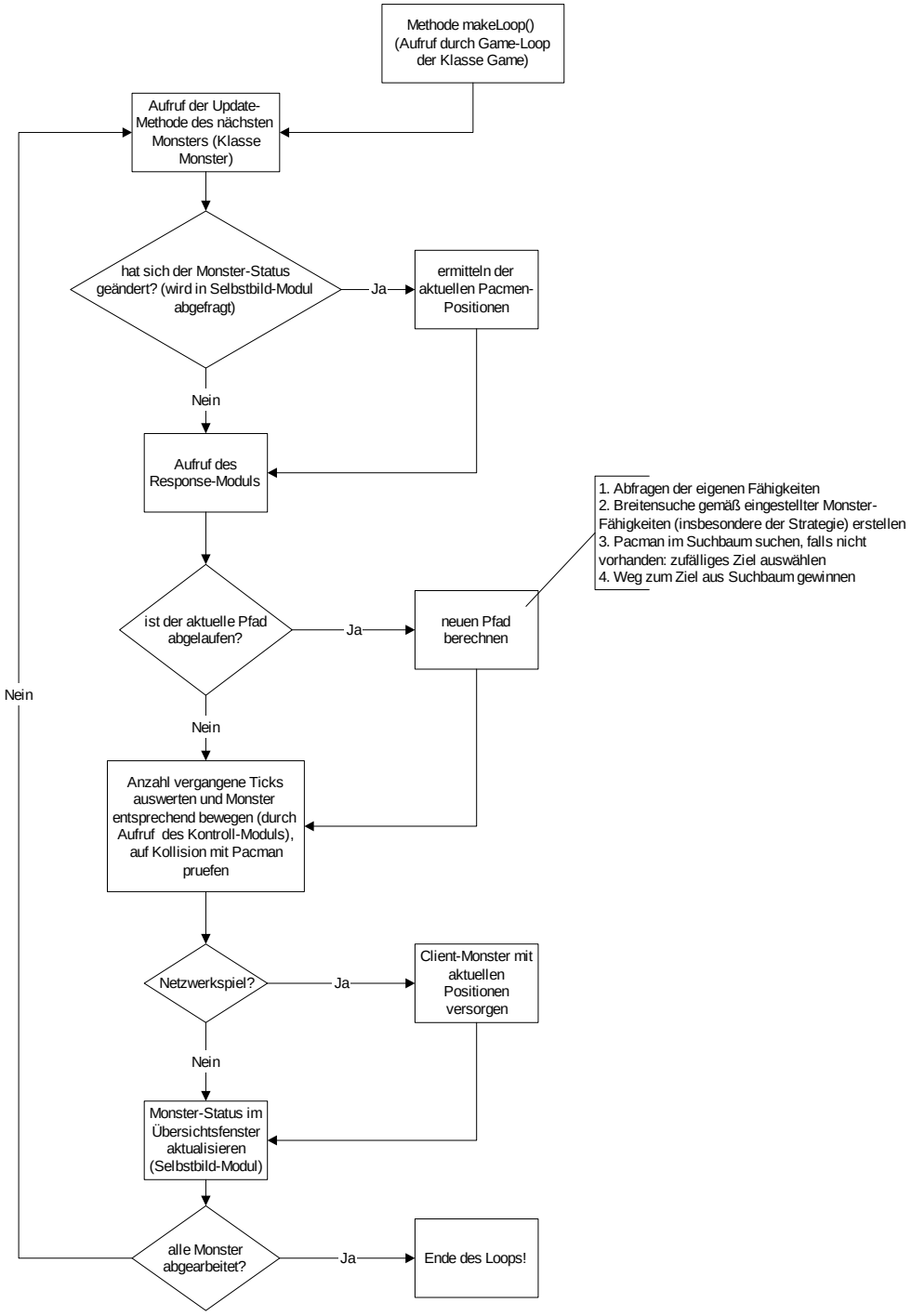

Abbildung 18: Monster Flow Chart

4.3 Kommunikationsmodul

Mit diesem Modul können die Agenten untereinander kommunizieren, was eine grundlegende Eigenschaft eines Agentensystems ist. Implementiert ist diese Form der Kommunikation (nicht zu verwechseln mit Netzwerkkommunikation oder Kommuni-

kation zwischen Monster und Pacman) durch die Klasse `MonsterVoice`. Sie stellt Methoden zum Senden und Empfangen von Nachrichten zur Verfügung.

4.4 Response-Modul

Das Response-Modul ist die zentrale Stelle für die „Intelligenz“ des Monsters. Wünschenswert wäre hier eine adaptive Struktur gewesen, welche jedoch (wie erwähnt) aus Zeit und Performanzgründen nicht zum Einsatz kam. Ersetzt wurde die Intelligenz durch einstellbare Strategien, welche durch Einstellung per Schieberegler oder durch Laden eines Level-Files ausgewählt werden können.

Jede der implementierten Strategien ist auf einen Spezialfall der Breitensuche zurückzuführen, was die Erweiterbarkeit der Fähigkeiten des Monsters vereinfacht.

Realisiert wurde dieses Konzept durch die Klasse `MonsterResponse`. Die `MonsterResponse` stößt die Berechnung von neuen Pfaden, einer Strategie, eine Suche nach einem Pacman oder die Kommunikation zwischen Monstern an.

4.5 Kommunikationsebene

Die Kommunikationsebene wird repräsentiert durch die Klasse `MonsterChannel`, welche die zentrale Schnittstelle der Monster zum Labyrinth darstellt. Hier können beispielsweise `Cell`-Objekte (VIII - 4.5 Zellen) vom Labyrinth abgefragt (`getCellFromLaby()`) oder die aktuelle Pacman-Position bestimmt werden (`getPacmanPosFromLaby()`) - Informationen, die anschließend in der internen Breitensuche verarbeitet werden.

Über die Kommunikationsebene präsentiert sich das Monster auch der Außenwelt: So kann die MonsterID abgefragt werden; durch Aufruf der Methode `getJ3DNode()` wird der Monster-Szenegraph zurückgeliefert. - Auch die Methode `makeLoop()` befindet sich in dieser Klasse: Ein Aufruf mit der entsprechenden Anzahl von zur Verfügung stehenden Ticks lässt die Monster auf den Pacman los...

Durch den Einsatz dieses zusätzlichen Moduls konnten viele Methoden aus dem Agentenmodell ausgelagert und gleichzeitig die Flexibilität für Erweiterungen des gesamten Projekts gesichert werden. Möglich ist der Einsatz beliebig vieler Monster in beliebig großen Labyrinthen. Gleichzeitig dient die Kommunikationsebene als Kommunikationskanal für Intra-Monsterkommunikation.

5 Die Modellierung

Es gibt insgesamt drei verschiedene Monster-Modelle. Eines ist der „Kodos“ (187 kB als wrl-Datei; vgl. Abb. 7), ein relativ aufwändiges Modell (Ähnlichkeiten zu einem Alien einer beliebten Fernsehserie sind nicht zufällig). Um auch Spielvarianten zu unterstützen, die auf weniger schnelle Rechner zugeschnitten sind, gibt es das etwas kleinere Modell „Geist“ (67 kB als wrl-Datei, vgl. Abb. 8), das dem traditionellen Geist der 2D-Pacman-Spiele nachempfunden ist. Als drittes Monster spukt der „Pac-Borg“(96 kB als wrl-Datei, vgl. Abb. 9) durch das Labyrinth, ein in einem früheren Spiel assimilierter Pacman mit den für Borgs üblichen Schläuchen.

Je nach Kapazität können verschiedene Modelle geladen werden, momentan kommen alle drei Modelle im Labyrinth vor.

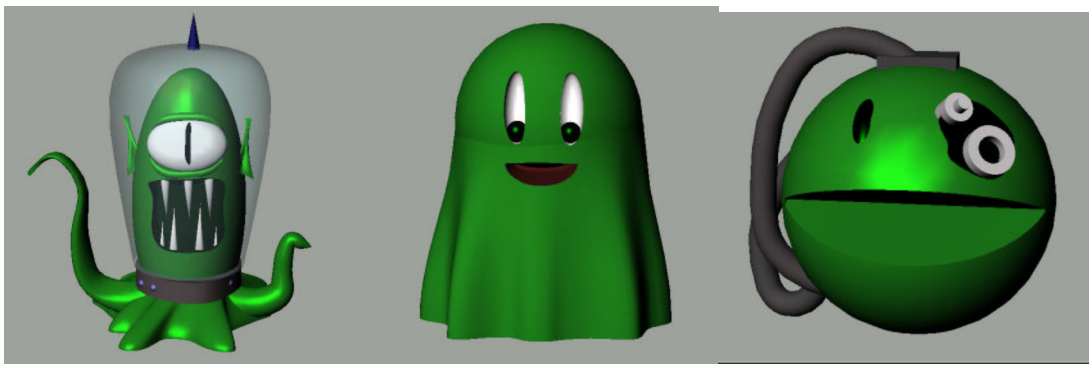

Abbildung 19: Kodos, der freundliche Geist und PacBorg

Alle Modelle sind gleichfarbig, nämlich giftgrün; dies soll der besseren Übersichtlichkeit dienen, so dass die Monster leichter von den verschiedenfarbigen Pacman unterscheidbar sind. Alle Modelle wurden mit Rhinoceros 2.0 modelliert, als VRML-Modelle (`*.wrl`) exportiert und dann mit dem ModelLoader aus dem NCSA-Portfolio in die Java-3D-Szene geladen. Anschließend wurden Lichtquellen hinzugefügt, so dass die Monster sichtbar wurden. Ohne eine eigene Lichtquelle, die mit herumgetragen wird, waren sowohl Monster als auch Pacman nur als dunkle Schatten in der Szene zu sehen.

6 Spezielle Probleme und Lösungen

6.1 Strategie

Die Strategieberechnung des Monsters ist gekapselt in der Klasse `MonsterResponse`. Hier werden aufgrund der aktuellen Eigenschaften des Monsters und abhängig vom Status des Pacman neue Zielzellen berechnet. Anschließend sucht das Monster den kürzesten Weg dorthin und legt diesen Pfad in der `MonsterControl` ab: Die Jagd hat begonnen.

6.2 Visualisierung der Monster-Eigenschaften und Strategie-Status

Es war Teil unserer Aufgabe, die verschiedenen Monster-Zustände und -Eigenschaften zu visualisieren, um während der Test-Phase die Einflüsse der unterschiedlichen Parameter auf das Monster-Verhalten genauer untersuchen zu können. Zu diesem Zweck wurde für jedes Monster ein eigenes Konfigurationsfenster bereitgestellt, über das man die entsprechenden Einstellungen individuell für dieses Monster vornehmen kann (VI - 4.2 Selbstbild). Insbesondere lässt sich über einen Schieberegler die zu wählende Monster-Strategie einstellen; auf diese Weise ist es auf einfache Art möglich, ein für das aktuelle Spiel-Level adäquates Monster-Ensemble zusammenzustellen.

Im eigentlichen Spiel-Modus werden diese `MonsterSetup`-Fenster nicht angezeigt und das Monster bezieht seine Eigenschaften aus dem bereitgestellten Level-File.

6.3 Anfänger und Fortgeschrittene

Im Rahmen dieses Praktikums kam es uns vor allem darauf an, eine sinnvolle Grundstruktur für das Klassen-Modell zu finden und saubere Schnittstellen zu den anderen Programmiergruppen zu definieren. Dennoch sind schon in der vorliegenden „Ausbaustufe“ verschiedene Jagdstrategien und eine Fluchttaktik implementiert, die als Rahmendaten ihrer Berechnung jedoch ausschließlich die aktuelle Pacman-Position (sofern dieser erreichbar ist) und die kürzesten Pfade aus einem Breitensuchbaum berücksichtigen.

Fortgeschrittene Strategien könnten dynamischer ablaufen und weitere Informationen bei der Berechnung berücksichtigen. Wenn auch bislang wegen der Kürze der Zeit nicht umgesetzt, wollen wir doch an dieser Stelle unsere Ideen präsentieren:

- Intelligente Gewichtung der Knoten und Kanten im Breitensuchbaum, so dass zum Beispiel Pfade über Knoten mit großer Child-Anzahl (In-/Out-Degree) bevorzugt abgelaufen werden. Genau diese Knoten erlauben dem Pacman eine Flucht in viele unterschiedliche Richtungen; das heißt, das Monster würde auf diese Weise versuchen, die Freiheitsgrade des Pacman einzuschränken und ihn somit „in die Enge“ treiben.
- Wenn Knotengewichte eingeführt werden, könnte man versuchsweise ein Monster-Gedächtnis installieren. Das soll heißen: Zellen/Knoten, die ein zufällig laufendes Monster besucht hat, werden mit einem Strafzoll belegt, so dass genau diese Knoten bei einer nächsten Pfadberechnung benachteiligt wären. Das Ziel hierbei ist natürlich, dass das Monster möglichst rasch (und nicht erst nach einer für eine gute stochastische Verteilung notwendigen Zeit) gezielt alle Zellen des Labyrinths besucht und auch Pacman entdeckt werden, die sich in einer entlegenen Ecke verborgen halten und in aller Ruhe ihre Pillen fressen...
- Eine erweiterte kooperierende Strategie könnte sich bei ihrer Berechnung auch auf die berechneten Pfade der Partner-Monster beziehen und bei der Verfolgung des Pacman einen Pfad mit einer minimalen Zellüberdeckung wählen. Bei einer durchschnittlichen Labyrinth-Struktur wäre hier also zu erwarten, dass der Pacman von verschiedenen Seiten eingekesselt wird und den Weg allen Irdischen geht.
- Ein Problem, das im letzten Punkt erkennbar wird, betrifft das Wissen um die zeitliche Dimension der Verfolgung. Es macht beispielsweise nicht unbedingt Sinn, wenn sich zwei Partner-Monster, die noch viele Zellen vom Pacman entfernt sind, ihre Pfade abgleichen und dann auf unterschiedlichen Wegen zur selben Zielzelle laufen. Wenn die beiden Monster dort ankommen, wird der Pacman längst in andere Gefilde entfleucht sein und sich seines Lebens freuen. Ein erster Ansatz wäre hier, dass die Monster bei ihrer Verfolgung genau diese zeitliche Komponente mit berücksichtigen und eine mögliche „Fluchtsphäre“ des Pacman in Betracht ziehen.
- Der Begriff „Fluchtsphäre“ deutet es bereits an: Der Pacman wird seinerseits bemüht sein, bei seiner Flucht vor einem Monster einen möglichst großen Abstand zu anderen Geistern und Kobolden einzuhalten. Genau dieses Verhalten könnte das verfolgende Monster ausnutzen: Das Wissen um die Position der anderen Monster erlaubt es ihm, bestimmte Fluchtregionen des Pacman von vornherein auszuschließen.

6.4 Ein Monster ist ein Monster ist ein Monster...

Die hier besprochenen Ausbaustufen lassen sich Dank des allgemeingehaltenen Agentenmodells leicht integrieren und würden lediglich eine Erweiterung der `MonsterAwareness` um weitere Attribute und die entsprechende Auswertung derselben in der `MonsterResponse` bedeuten. Die Schnittstellen zu den anderen Monster-Klassen können hierbei unverändert übernommen werden und würden in gleicher Funktion weiter bestehen wie bei den von uns umgesetzten Monster-Verhaltensweisen:

- Die einfachste Monster-Strategie realisiert ein zufälliges Laufen durch das Labyrinth. Eine neue Zielzelle wird also nach dem Zufallsprinzip aus der Menge der vorhandenen Zellen bestimmt und ein kürzester Pfad dorthin berechnet. Diese Strategie verbindet minimalen Aufwand mit maximalem Nutzen, denn das Monster wird im Laufe des Spiels alle Zellen des Labyrinths mit gleicher Wahrscheinlichkeit besucht haben und somit den gesamten Spielraum gleichmäßig abdecken. Außerdem ist der Überraschungseffekt bei dieser Strategie gerade in einem 3D-Labyrinth nicht zu unterschätzen: Jederzeit kann ein Monster von einem oberen Level in den unteren wechseln und plötzlich genau vor dem Pacman auftauchen, ohne dass der Spieler vorhersagen kann, in welche Richtung es sich im nächsten Schritt bewegen wird: Für Spannung ist gesorgt.
- Die von uns implementierte kooperative Strategie setzt genau an diesem Punkt an: Ein zufällig laufendes Monster (in diesem Zusammenhang „Kundschafter“ genannt) sendet, sobald es fündig geworden ist, die Pacman-Position an die Partner-Monster, die sich sogleich auf den Weg dorthin machen. Bei der Levelgestaltung hat sich herausgestellt, dass es an dieser Stelle sinnvoll sein kann, die Geschwindigkeit der unterschiedlichen Monster-Typen zu variieren: der Kundschafter könnte sich zum Beispiel ein wenig schneller bewegen als das „Jäger-Monster“; auf diese Weise würde man die geringere „Intelligenz“ des Kundschafters also mit anderen Fähigkeiten recht zielgenau aufwerten können.
- Das „Jäger-Monster“ baut einen Suchbaum mit einer dem Aktionsradius entsprechenden Tiefe auf und versucht in diesem Baum den Pacman zu finden. Wenn das Monster fündig wird, begibt es sich auf direktem Weg zur gefundenen Pacman-Position. Wenn der Pacman ab diesem Zeitpunkt keine konsequente Fluchtstrategie umsetzt (sondern zum Beispiel Umwege zum Sammeln weiterer Pillen nimmt, kurze Zeit auf einer Zelle verharrt oder Ähnliches), wird sich der Abstand zwischen Jäger und Pacman immer weiter verringern. Die einzige Rettung des Pacman würde dann in einer geschickt eingesetzten Jagd-Pille bestehen, die ihn unverwundbar macht. Auch hier (siehe Punkt 2) scheint es also für ein ausgewogenes Game-Play sinnvoll zu sein, die Geschwindigkeit des Jäger-Monsters leicht abzusenken (unter die des Pacman), um in diesem Punkt Waffengleichheit herzustellen.

Sollte es dem Pacman gelungen sein, seinen Schutzschild mit der entsprechenden Pille zu aktivieren (die sagenumwobene „green pill“), wird dies von Monstern in der Nähe registriert und eine entsprechende Fluchtstrategie eingeschlagen. Das Monster versucht durch seine Bewegung den Abstand zum Pacman konstant zu halten. Die dahinter stehende Idee beruht auf dem Prinzip sich abstoßender gleichnamiger Pole: Pacman- und Monster-Position werden sich daher nur annähern, wenn das Monster in die Enge getrieben wird. An dieser Stelle können nachfolgende studierende Generationen

kreativ tätig werden, denn auch hier ließe sich Einiges finden, was das Monster gerade vor einer solchen misslichen Lage bewahren kann.

7 Fazit

In diesem Praktikum haben wir alle drei zum ersten Mal an einem so großen Programmierprojekt teilgenommen. Zum einen bedeutete das eine relativ intensive Zusammenarbeit innerhalb unserer Gruppe, und dann ziemlich viel Absprache mit anderen Gruppen über Schnittstellen und Zusammenhänge. Vorweg sei vielleicht gesagt, dass wir jetzt, wo es vorbei ist, ein leichtes Bedauern verspüren und schon wieder Lust auf ein nächstes größeres Projekt haben…

Trotzdem gab es auch Punkte, die uns zweifeln ließen: Ein gemeinsames Vorgehen in einer Gruppe von immerhin rund 15 Leuten zu koordinieren war sehr aufwändig. Glücklicherweise gab es seit Anfang des Jahres 2002 WinCVS, um das größte Chaos zu vermeiden und immer alle auf dem aktuellsten Stand zu halten. Aber auch das barg ungeahnte Tücken: Kaum ein Code lief sofort, wenn man ihn herunter geladen hatte und die meiste Zeit haben wir sicher mit debuggen und dem Warten auf Ausbesserungen der anderen verbracht, um endlich weiter machen zu können. (was nicht heißt, dass es den anderen Gruppen anders ging!)

Obwohl sich innerhalb unserer Kleingruppe Spezialisten für einzelne Teile des Codes herausbildeten, haben wir versucht, das Meiste gemeinsam zu machen. Wir finden, dass das sehr gut gelungen ist (in fast allen Klassen finden sich Spuren von uns allen) und würden diese Vorgehensweise empfehlen. Um weiteres Vorgehen oder Probleme zu besprechen, war es sehr gut, dass wir immer alle wussten (oder zumindest ungefähr…), um was konkret es sich da handelte.

Dass das Erfolgserlebnis viel größer ist, wenn das zusammengesetzte Ergebnis etwas so großes wie ein ganzes Spiel ist (was wir zu dritt nie hätten bewältigen können), brauchen wir ja wohl nicht zu sagen…

VII - DIE KAMERAGRUPPE

1 Aufgaben

Die programmiertechnischen Aufgaben der Kameragruppe waren die Ausarbeitung und Implementierung verschiedener Kamera-Ansichten, mit denen sich Pacman 3D spielen lässt.

Weiterhin sollte sich überlegt werden, ob man zusätzliche Effekte, die sich auf die Ansichten auswirken, finden und umsetzen kann.

Im Laufe der Zeit gesellte sich noch die Aufgabe hinzu, einige Items für das Spiel zu modellieren.

1.1 Ansichten

Die nachfolgenden Ansichten wurden gefunden und stellten sich als sinnvoll und vor allem als spielbar heraus. Manche dieser Ansichten bieten die Möglichkeit einen unterschiedlichen Stil einzustellen, bei dem sich die Kamera speziell verhält. Bei Umschalten in eine Ansicht wird zuerst immer der Normalstil eingestellt.

1.1.1 EGO-View

Die EGO-View ist die Sicht, die der Pacman als solcher hat. Der Spieler steckt somit im Pacman, bzw. ist der Pacman. Der Vorteil dieser Sicht ist, dass man Auf- und Abgänge gut als Löcher im Boden und der Decke erkennen kann:

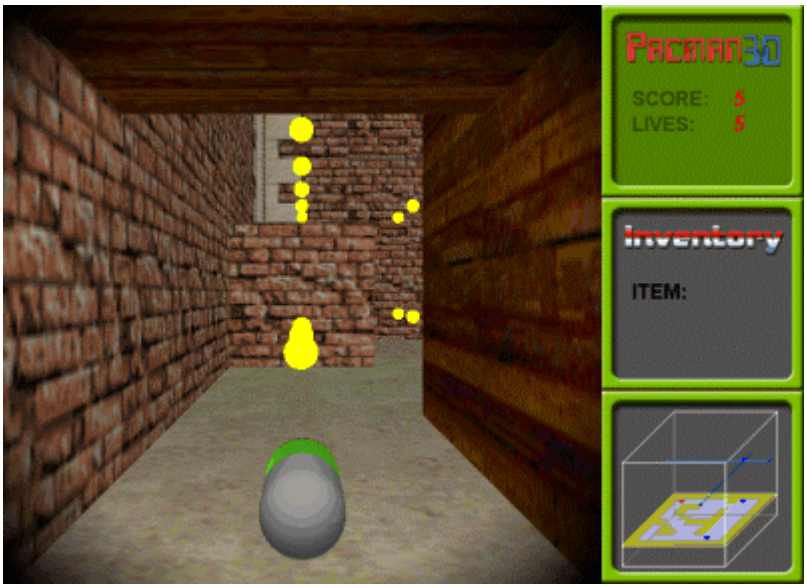

Abbildung 20 - Die EGO-View

Der Nachteil ist, dass man viel weniger von dem sieht, was sonst noch im Level passiert:

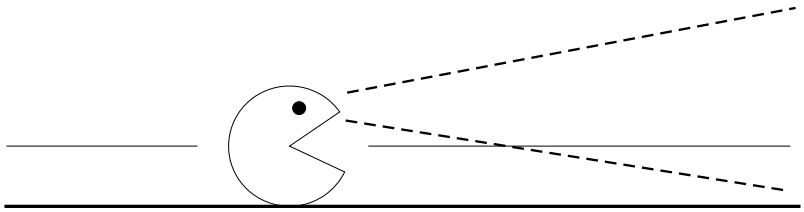

Abbildung 21 - Sichtwinkel der EGO-View

1.1.2 TRACKER-View

Bei dieser Ansicht sitzt die Kamera in einem bestimmten Abstand vom Pacman entfernt mit Blick auf das Zentrum des Pacman. Rotiert der Pacman, so rotiert auch die Kamera um den Mittelpunkt des Pacman. Die Position relativ zum Pacman wird also immer konstant gehalten:

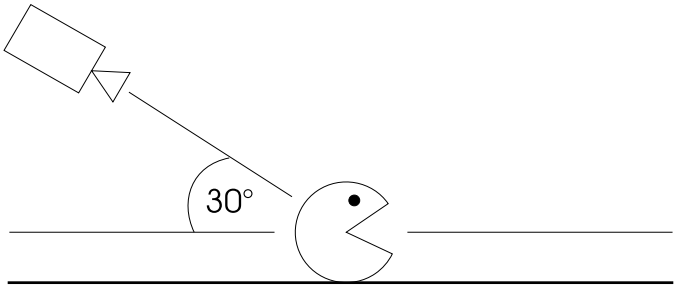

Abbildung 22 - Die TRACKER-View

- **Normal-Stil:**

  Die Kamera sitzt etwas erhöht hinter dem Pacman fest und verfolgt ihn dadurch immer von hinten:

Abbildung 23 - Normal-Stil der TRACKER-View

- **Freerotation-Stil:**

  Durch bestimmte Tasten kann man die Kamera auf ihrem Orbit um den Pacman um das Zentrum des Pacman rotieren lassen. Diese Position wird gespeichert und bei Eigenrotation des Pacman entsprechend mitverändert.

  Zusätzlich kann man den Abstand der Kamera zum Pacman verringern oder vergrößern:

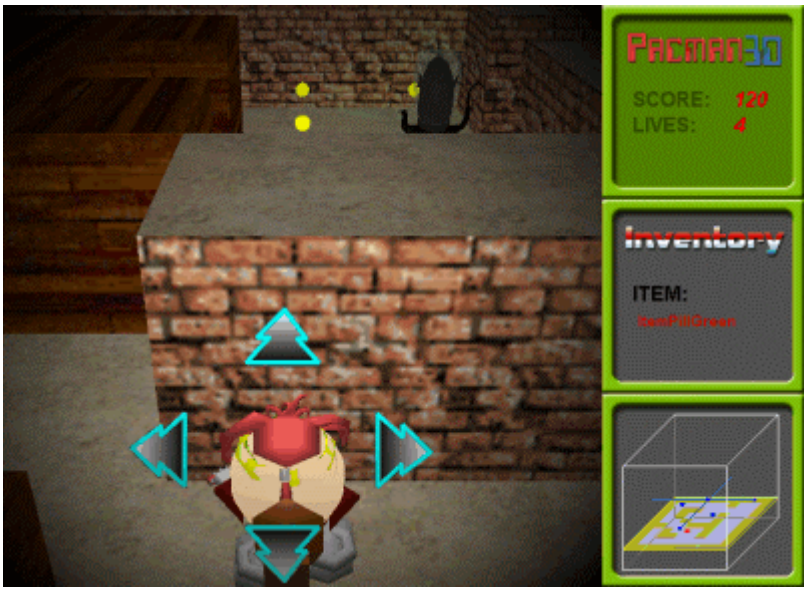

Abbildung 24 - Freerotation-Stil der TRACKER-View

1.1.3 BIRD-View

Die Kamera befindet sich in einem bestimmten Abstand genau über dem Pacman und ist auf ihn gerichtet.

Diese Ansicht mutet etwas wie eine 2D-Ansicht an und erinnert dadurch stark an das Original-Pacman-Spiel:

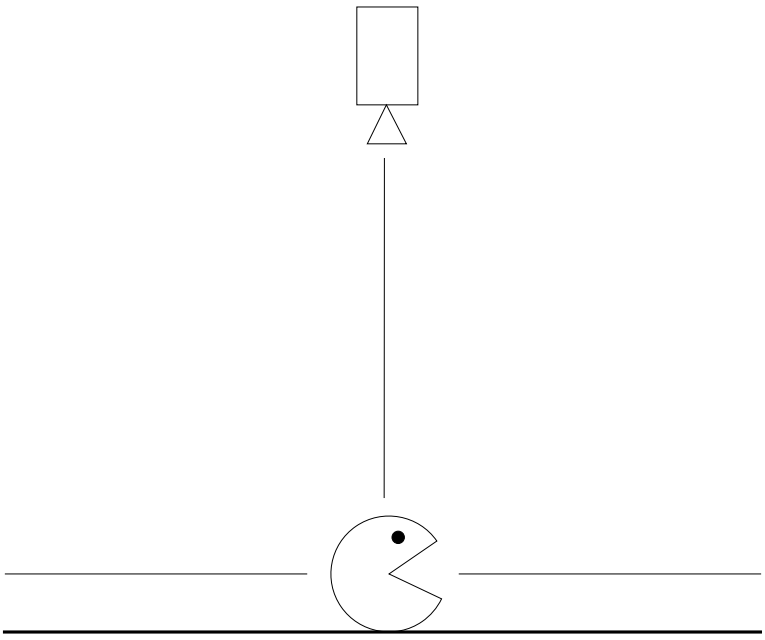

Abbildung 25 - Die BIRD-View

- **Normal-Stil:**

  Die obere Kante des Bildschirms ist immer senkrecht zur Blickrichtung des Pacman. Dreht sich der Pacman, so dreht sich auch die Kamera entsprechend um die Y-Achse.

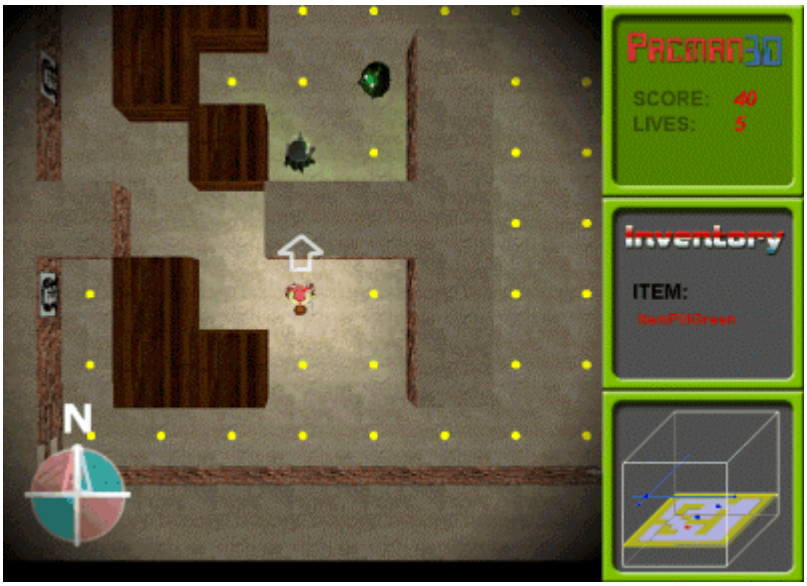

Abbildung 26 - Der Normal-Stil der BIRD-View

- **Unbound-Stil:**

  Die Ausrichtung der Kamera bleibt fest und dreht sich nicht, obwohl der Pacman frei rotieren kann:

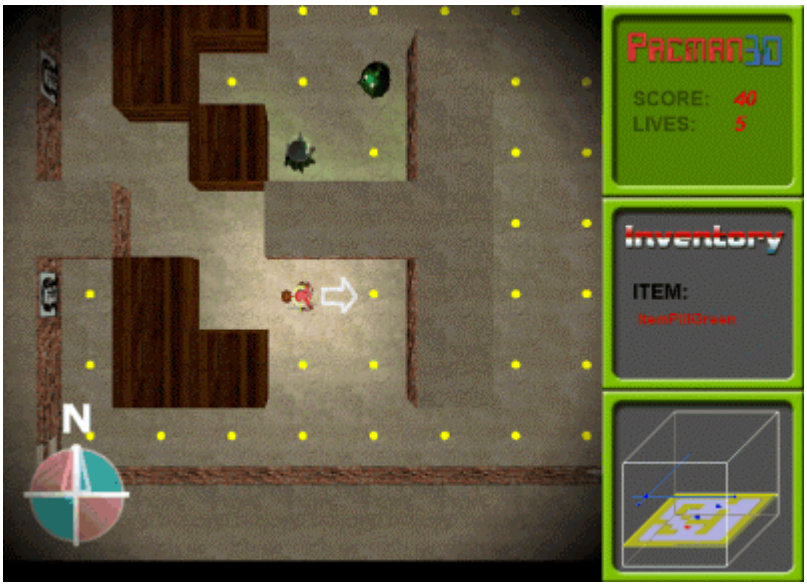

Abbildung 27 - Der Unbound-Stil der BIRD-View

- **Cage-Stil:**

  Auch hier bleibt die Kamera ohne Rotation und dreht nicht mit dem Pacman. Allerdings folgt die Kamera nicht ständig, sondern lässt den Pacman in einem festen imaginären Käfig frei bewegen und verschiebt den Käfig und damit ihre Position erst, wenn der Pacman an eine der vier Begrenzungen stößt:

Abbildung 28 - Der Cage-Stil der BIRD-View

1.1.4 ISO-View

Die Kamera hat einen festen (isometrischen) Blickwinkel auf das Labyrinth. Das Labyrinth liegt also quasi schräg vor dem Betrachter, so daß alle Objekte auf den Kacheln trotz der Mauern zu sehen sind. Die Kamera folgt dem Pacman durch Änderung der Position, gibt aber niemals den Blickwinkel auf.

- **Normal-Stil:**

  Die Kamera folgt dem Pacman ständig:

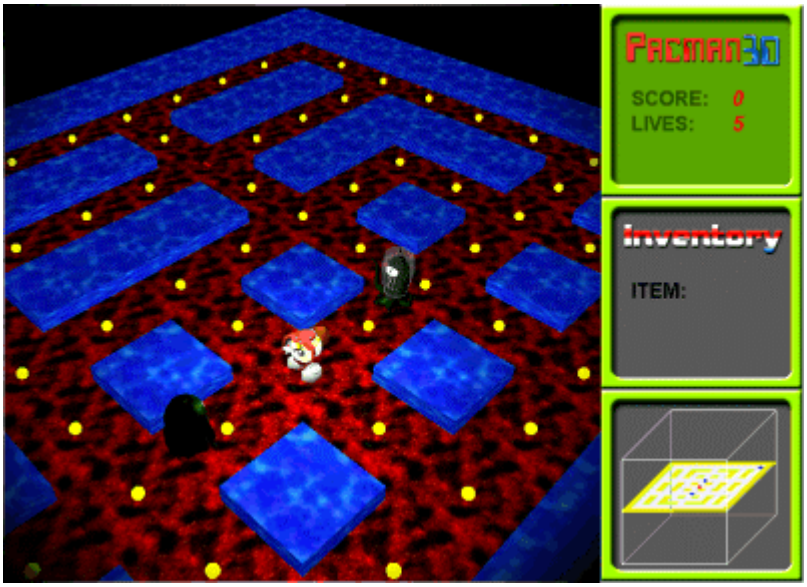

Abbildung 29 - Der Normal-Stil der ISO-View

- **Cage-Stil:**

  Wie bei der BIRD-View befindet sich ein imaginärer Würfel um den Pacman, in dem sich der Pacman frei bewegen kann ohne dass die Kamera ihre Position ändert:

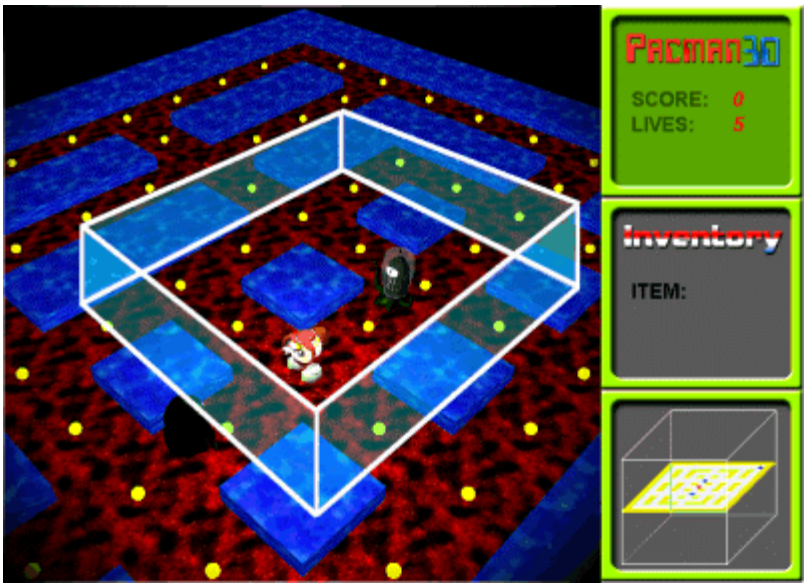

Abbildung 30 - Der Cage-Stil der ISO-View

1.2 Kamera-Effekte

Zusätzlich zu den Ansichten wurden zwei Effekte eingebaut. Zum Einen kann in der EGO-Ansicht das Blickfeld („Field-Of-View“) verändert werden, wodurch der Eindruck entsteht, dass man durch ein Fernglas schaut. Zum Anderen kann man die Kamera, wie durch eine Erschütterung, hin- und herwackeln lassen.

1.3 Modellierung

Zu modellieren waren die sogenannten „Items“ (siehe VIII - 4.6 Items), die der Spieler im Level aufsammeln kann, bzw. muss. Dazu gehören mindestens die normalen „Fresspunkte“, Objekte, die Bonuspunkte bringen und ein Objekt, welches den Pacman die Monster jagen lässt:

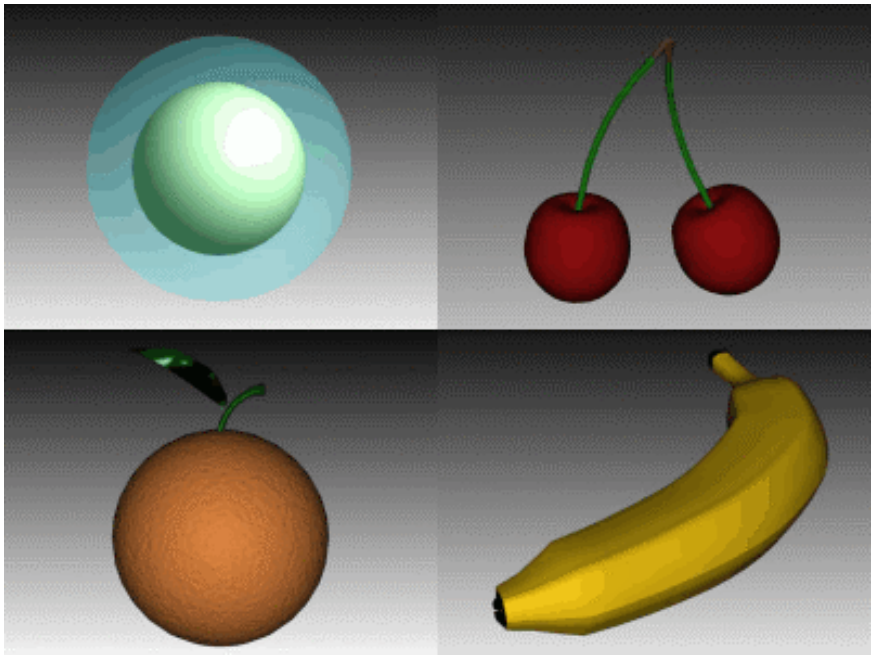

Abbildung 31 - Items

2 Programmierung

2.1 Implementierung

2.1.1 Die Technik dahinter

Eine Szene in Java3D befindet sich in einem lokalen Koordinatensystem, in dem jedes dreidimensionale Objekt seine Position definiert. Betrachtet wird diese Szene von einem bestimmten Blickpunkt aus. Die Transformationen, die zum korrekten Rendern der Szene relativ zu diesem Blickpunkt notwendig sind, übernimmt Java3D automatisch, d. h. man muss sich lediglich um die Positionierung des Blickpunktes kümmern, was über das `View`-Objekt der `View`-Klasse geschieht. Das `View`-Objekt beinhaltet alle Parameter um eine dreidimensionale Szene zu rendern. Ein `View` enthält eine Liste von Canvas3D-Objekten in welche die Ansicht gerendert wird. Es existiert außerhalb des Szenegraphs ist aber mit dem `ViewPlatform`-Knoten im Szenegraphen verknüpft. Es enthält auch eine Referenz zu einem `PhysicalBody`- und einem `PhysicalEnvironment`-Objekt.

Das `View`-Objekt ist das Java3D-Hauptobjekt, welches das Java3D `Viewing-Model` steuert und wird direkt nach der Konstruktion des `VirtualUniverse` aufgesetzt. Alle Komponenten welche die Transformation der Ansicht zum Rendern auf die 3DCanvases spezifizieren sind entweder im `View`-Objekt enthalten oder in den Objek-

ten, die vom `View`-Objekt referenziert werden. Deswegen wird es auch häufig als „rendering infrastructure“ bezeichnet.

Java3D erlaubt es Anwendungen, mehrere, gleichzeitig aktive, `View`-Objekte zu definieren, die jeweils ihre eigene `3D Canvases` besitzen.

Die Kamera unseres Projektes entspricht dem besagten Blickpunkt. Veränderungen am Blickpunkt bewirkt man, indem man über die `Transformgroup` der `Branchgroup`, die dem `View` zugewiesen ist, zugreift. Dort kann man z. B. über die Funktion `setTransform()` eine neue Position, sowie eine andere Blickrichtung festlegen.

Sind die verschiedenen Ansichtsmodi der Kamera, was Position und Blickrichtung angeht, einmal festgelegt kann man diese somit ohne größere Probleme implementieren.

Weitere Einstellungen, die das Rendern betreffen, können im `View`-Objekt verändert werden. Dazu gehören parallele / perspektivische Projektion, Skalierung relativ zum Bildschirm, Interpretation der Augenposition, Clipping, etc. . An den Grundeinstellungen wurde für das Projekt außer dem „Field-of-View“ (Sichtkegel) nichts verändert. Um einen Fernglas-Effekt zu bekommen, wird hier der Sichtkegel verkleinert, was einem Blick durch ein Objektiv entspricht.

2.1.2 Der Aufbau des Szenegraphen

Ein Java3D-Szenegraph besteht grundsätzlich aus einem `ContentBranch` und einem `ViewBranch`. Der `ContentBranch` enthält die Objekte, die sich in der Szene befinden und der `ViewBranch` steuert deren Darstellung. Um einen `ViewBranch` zu konstruieren erstellt man einmal einen `Canvas3D` und ein `SimpleUniverse`. Die Erstellung `SimpleUniverse` liefert uns

- ein `VirtualUniverse`-Objekt,
- ein hochauflösendes `Locale`,
- eine `Branchgroup` die an das `Locale` angehängt ist,
- eine `Transformgroup` in der `Branchgroup`,
- eine `ViewPlatform` in der `Transformgroup` und
- ein `View`-Objekt auf der `ViewPlatform`, die auf den Canvas3D rendert.

Die nachfolgende Grafik stellt die Hierarchie der genannten Objekte dar:

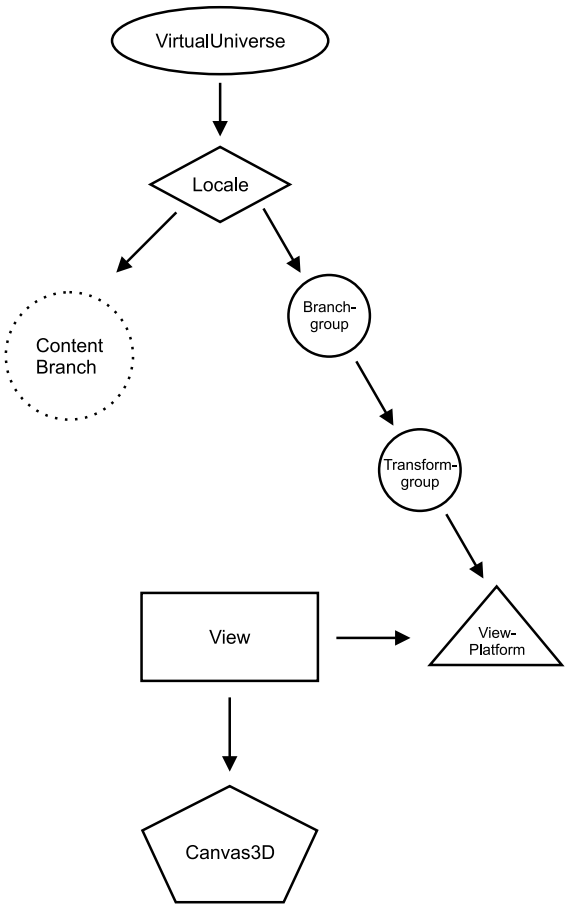

Abbildung 32 - Der Aufbau des Szenegraphen

2.1.3 Zusammenhang Spiel und Kamera

Das Spielfeld von Pacman 3D besteht aus einem Quader, hat also 3 Dimensionen. Eine einzelne Ebene ist aus kubischen Zellen zusammengesetzt, wobei durch solide Zellen Wände erstellt werden (siehe VIII - 4.4 Das Labyrinth, VIII - 4.5 Zellen).

Beim Aufbau dieses dreidimensionalen Labyrinths wird der Quader in einzelne Ebenen aufgeteilt, die komplett ein- und ausblendbar sind. Bei den Kamera-Ansichten „Tracker“, „Bird“ und „Iso“ werden jeweils die Ebene in der sich der Pacman befindet und alle darunterliegenden eingeblendet. Die darüberliegenden Ebenen werden ausgeblendet, da sie den Pacman zum Teil verdecken würden.

Bei der „Ego“-Ansicht werden sämtliche Ebenen eingeblendet, da es hierbei keine Gefahr der Verdeckung gibt. Der Grund dafür, dass neben den Flächen direkt unter oder über der Fläche, auf der sich der Pacman befindet, noch weitere eingeblendet werden, ist, dass bei manchen Auf- und Abgängen der Blick auf ein tieferliegendes Niveau freigegeben wird und dann ein unschönes schwarzes Loch sichtbar wäre.

Von der Labyrinth-Klasse erhält die Kamera-Klasse das Objekt des aktuellen Pacman, also des Pacman, der gerade gesteuert wird, übergeben. Über die Funktion `getRealPacmanPos` erhält man die absolute Position dieses Pacman als `Point3d`. Die Funktion `getRealLineOfSight` liefert die Blickrichtung als `Vector3d`. Damit hat die Kamera-Klasse alle notwendigen Werte um die Kamera entsprechend der gewählten Ansicht zu positionieren.

2.1.4 Der Quake-Effekt

Das Beben der Kamera, wenn der Pacman ein Leben verliert, wird durch Schwingen entlang der X-, Y- und Z-Achse erreicht. Die Funktion hierfür ist:

$$\sin(x * p) * a / x$$

wobei p die Frequenz und a den Ausschlag steuert. Diese beiden Werte sind für jede einzelne Achse einzeln einstellbar. Durch unterschiedliche Werte wird ein natürlich wirkendes, unregelmäßiges Beben erreicht. Die Laufvariable x wird pro tick hochgezählt.

2.2 Benötigte externe Schnittstellen

Die Kamera-Klasse greift auf die folgenden Schnittstellen zu:

- lesen der Blickrichtung des Pacman als 3D-Vektor (Pacman),
- lesen der exakten Position des Pacman als 3D-Koordinate (Pacman),
- ein- und ausschalten des Pacman (Pacman)
- ein- und ausschalten von Ebenen (Labyrinth)
- lesen der Anzahl „Ticks“, die seit dem letzten Kamera-Update vergangen sind (Labyrinth bzw. Gameloop)

2.3 Tastenbelegung der Kamera

Taste	Aktion
F1	EGO-View
F2	TRACKER-View
F3	BIRD-View
F4	ISO-View
SPACE	Wechsels des Kamerastiles
NUMPAD8	Kamera nach oben rollen
NUMPAD2	Kamera nach unten rollen
NUMPAD4	Kamera nach links rollen
NUMPAD6	Kamera nach rechts rollen

Taste	Aktion
NUMPAD1	Kamera herauszoomen
NUMPAD3	Kamera hineinzoomen
NUMPAD0	Kameraposition zurücksetzen

Weitere Tasten wurden von der Pacmangruppe definiert, siehe V - 1.2.6 Tastenbelegung des Pacman.

3 Modellierung

Die Modelle wurden in 3D Studio Max R3 erstellt und als VRML- bzw. 3ds-Modell exportiert. Es gibt zwei Versionen der einzelnen Objekte, hochauflösende mit hoher Polygonzahl und niedrigauflösende. Die niedrigauflösenden Objekte haben ca. 15 mal weniger Polygone und benötigen deswegen entsprechend weniger Rechenzeit. Insgesamt wirken diese Objekte eckiger, allerdings werden sie so klein im Spiel dargestellt, dass man dieses verschmerzen kann.

Bei der Modellierung wurde darauf geachtet, dass keine Texturen benutzt wurden und alle Oberflächen durch reine Materialeigenschaften definiert wurden. Transparenz wurde nicht verwendet, da Java3D bei der Darstellung von transparenten Flächen deutliche Geschwindigkeitseinbußen verbucht.

Das Objekt „Banane“ hat in der hochauflösenden Version 4.376 Polygone und in der niedrigauflösenden nur 290 Polygone:

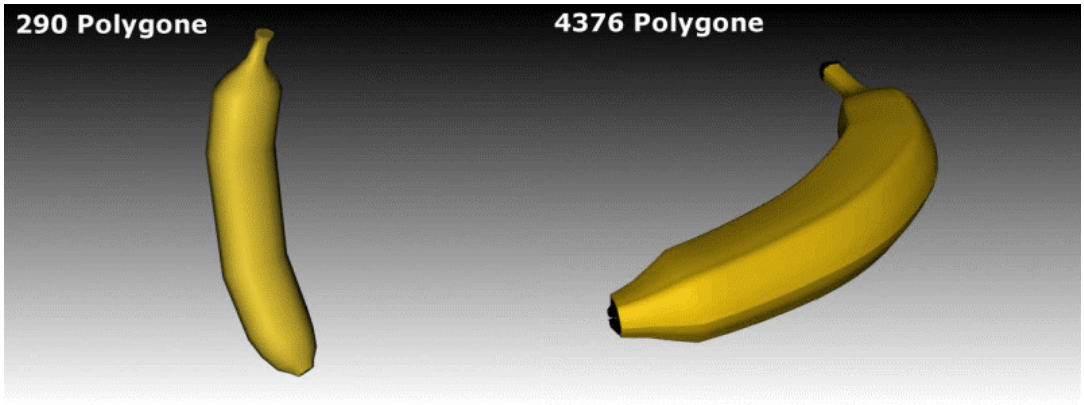

Abbildung 33 – Bananen-Item

4 Probleme, Java3D, Sonstiges

Probleme mit den anderen Gruppen gab es keine, da die Kamera relativ abgekapselt zu implementieren war. Die erste Ansicht war eine absolut freie Kamera, die bei Erzeugung eines `Canvas3D` automatisch erstellt wird. Somit konnten die anderen Gruppen ihre Entwicklungen begutachten, ohne dass die Kameragruppe ihre verschiedenen Modi fertig umgesetzt haben musste. Anhand der wenigen Schnittstellen, die für die Kamera benötigt werden, sieht man, dass man nicht viel Programmieraufwand von den anderen Gruppen für sich verlangen musste.

Das Modell, das Java3D zugrunde liegt, ermöglichte es uns, die Aufgaben mit extrem wenig Programmieraufwand umzusetzen und man konnte die Zeit nutzen um Einstellungen zu optimieren. Bei Java3D gibt es ein festes Koordinatensystem und die Kamera bewegt sich frei darin. Somit musste man nur die Position der Kamera abhängig von der des Pacman setzen und anschließend entsprechend dem Modus den Blickwinkel ausrichten. Die Veränderung des FOV („Field-Of-View“) ist sogar durch eine Funktion der `View`-Klasse vorgesehen und konnte so direkt genutzt werden. Um das Beben zu simulieren, musste eine gedämpfte Schwingungsfunktion gefunden und auf die 3 Achsen gelegt werden. Die jeweilige Verschiebung wurde dann auf die Position der Kamera aufaddiert.

VIII - DIE LABYRINTHGRUPPE

1 Vorwort der Labyrinthgruppe

...von Labyrinthen, Leveldateien und Nachrichten!

Jede der fünf Praktikumsgruppen (siehe III - 2 Die Aufgabenteilung) bekam eine eigene Teilaufgabe zugewiesen. Die Aufgabe der Labyrinthgruppe war es einerseits, ihrem Namen nach die Funktionalität und die Modellierung für die Repräsentation des Labyrinthes zu entwickeln, in welchem sich Pacman und Monster später einmal bewegen können sollten. Andererseits war es die Aufgabe der Labyrinthgruppe, als Bindeglied zwischen allen anderen Gruppen zu fungieren:

- Die von der Pacman- bzw. Monstergruppe geschaffenen Pacman und Monster sollten sich später einmal nicht nur durch das Labyrinth bewegen können, sondern das Labyrinth auch als Ansprechpartner zur Kommunikation mit ihrer Außenwelt benutzen. Dazu fällt z.B. die Kommunikation von Pacman zu Monstern und umgekehrt, aber auch die Kommunikation von Pacman bzw. Monstern mit allen anderen Systemkomponenten.
- Die von der Kameragruppe entwickelten Kameras integrieren sich in die vom Labyrinth bereitgestellte, dreidimensionale Szene, um verschiedene Kameraansichten bereitzustellen. Durch die Integration von Pacman und Monstern in das Labyrinth werden diese automatisch auch zu Bestandteilen der Szene.
- Die von der Netzwerkgruppe entwickelte Netzwerkschnittstelle sollte bei Fertigstellung transparent in das Labyrinth integriert werden können, ohne dabei für die anderen Gruppen Nachbesserungsaufwand zu verursachen. Dadurch lag die koordinierende Verantwortlichkeit für den Mehrspielermodus im Netzwerkbetrieb bei der Labyrinthgruppe. Der Anforderung wurde entsprochen, indem die Labyrinthgruppe bereits zu Beginn der Entwicklung ein netzwerkweites Nachrichtensystem entwickelt hat, in welches die Netzwerkschicht bei Fertigstellung transparent integriert werden konnte (siehe 4.7 Nachrichten).
- Die Labyrinthgruppe übernahm die Entwicklung der Startdatei der Anwendung (siehe 4.3 Die Startdatei), und damit ein Sammelsurium an verschiedensten Aufgaben wie z.B. die Auswertung von Kommandozeilenparametern, das Einlesen der verfügbaren Leveldateien und Ressourcen, die Instanzierung der benötigten Klassen und die Steuerung des Hauptprogrammflusses.

Insgesamt fünf Personen wurden der Labyrinthgruppe zugewiesen. Die Mitglieder sind im Einzelnen, und in alphabetischer Reihenfolge:

- **Fabian Wleklinski (Fabian@Wleklinski.de)**

  Poststelle, Ressourcenhandling, Laden/Speichern, Administration, Dokumentation, Debug-Klasse

- **Gordon Weckbach (pacman@wackyweed.de)**

  Leveleditor, Leveleditor-Dokumentation, Zellen und Leveldateien.

- **Lijun Zhou (zhou@informatik.uni-frankfurt.de)**

  Modellierung und Programmierung von Items

- **Martin Klossek (martin@klossek3000.de)**

  Game-Sockel, 2D-Elemente, XML-Format, Laden/Speichern, Administration, Website

- **Paul Izquierdo Rojas (pir19@gmx.de)**

  Grundstrukturen, Programmierung von Zellen

2 Der Entwicklungsprozess

...wie Pacman 3D entstanden ist!

Dieser Abschnitt dokumentiert den Entwicklungsprozess von Pacman 3D aus der Sicht eines Mitglieds der Labyrinthgruppe. Andere Praktikumsteilnehmer können naturgemäß andere Erfahrungen und Eindrücke gesammelt haben.

2.1 Oktober 2001: Angesicht in Angesicht

Es war die Kick-Off-Veranstaltung am 22. Oktober 2001, als sich die Teilnehmer des Praktikums das erste Mal Angesicht in Angesicht gegenübersaßen. Erst an diesem Tag lichtete sich plötzlich und endgültig, mit welchen anderen Teilnehmern jeder in den nächsten sechs Monaten zusammenarbeiten würde.

In der Zeit von Oktober 2001 bis März 2002 sorgten knapp ein Dutzend Treffen vor Ort im Fraunhofer AGC dafür, dass sich die Praktikumsteilnehmer nicht sprichwörtlich „aus den Augen verloren“. Bei diesen Treffen wurden die Arbeitsfortschritte begutachtet, und das grundsätzliche, weitere Vorgehen festgelegt.

2.2 November 2001: Bistros und Lokalitäten

Für die gruppeninterne Feinkonzeption erwiesen sich die Treffen im Fraunhofer AGC sehr bald als nicht optimal geeignet:

Für das Austüfteln technischer Details war das Fraunhofer AGC mit 15 Teilnehmern meist zu voll. Auch gab es immer wieder einmal Gesprächsthemen, die gruppeninterner Natur waren, und dies nach Möglichkeit auch für alle Zeit bleiben sollten. Und last, but not least waren die Treffen im Fraunhofer AGC eine gute Möglichkeit, mit den Teilnehmern der anderen Gruppen ins Gespräch zu kommen - warum also sollte man diese Gelegenheiten für gruppeninterne Diskussionen vergeuden?

Aus den oben genannten drei Gründen verlagerte sich die gruppeninterne Feinkonzeption der Labyrinthgruppe Anfang November 2001 in nahegelegene Bistros und Lokalitäten. Während einer Handvoll Treffen von jeweils drei bis fünf Labyrinthgruppen-Mitgliedern, die oftmals zu zwei- bis dreistündigen Sitzungen ausuferten, wurden dort wichtige Vorgehensweisen für die spätere Implementierung debattiert und festgeklopft, lange bevor mit der eigentlichen Implementierung begonnen worden war.

Einige Mitglieder der Labyrinthgruppe wurden angesichts der fortschreitenden Zeit, und der nicht zunehmenden Anzahl von Codezeilen immer nervöser, und drängten energisch darauf, alsbald mit der Implementierung zu beginnen.

In dieser Zeit entstand die folgende Skizze – der erste Grobentwurf der diskreten Labyrinthstruktur:

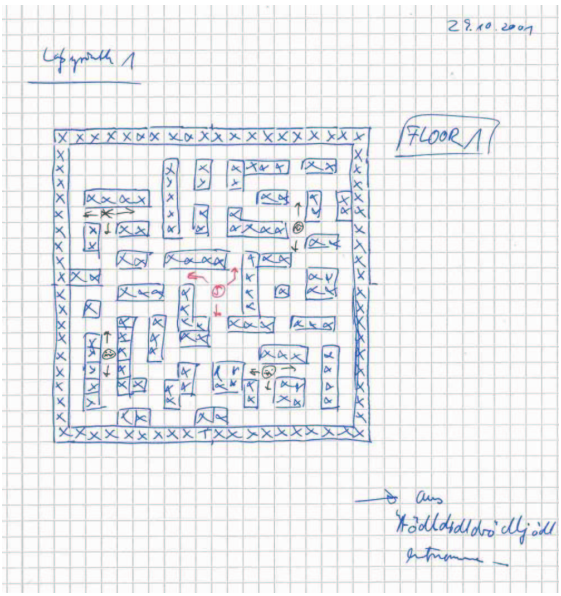

Abbildung 34 - Erster Grobentwurf des Labyrinthes

2.3 Dezember 2001: Mailingliste, HTTP und FTP

Es war bereits die Mitte des Novembermonats überschritten, als erste Implementierungsversuche der Labyrinthgruppe begannen.

Seit der Kick-Off-Veranstaltung waren nun rund vier Wochen der Feinkonzeption ins Land gezogen, und erst nun begann für die Labyrinthgruppe die Implementierung der bislang nur gedanklich modellierten Software.

Anfänglich verzögerten der Abspracheaufwand mit den anderen Gruppen und auch die immer wieder erforderliche, konzeptionelle Verfeinerung und Überarbeitung den Implementierungsfortschritt so stark, dass die Implementierung erst im Dezembermonat bedeutsam an Fahrt gewann.

Mit Beginn der Implementierung durch immerhin 15 Personen (!) stieg der Kommunikationsaufwand rapide an: Innerhalb der eigenen Gruppe erwies sich die Kommunikation per E-Mail als probates Mittel, da der Adressatenkreis klein, und die Empfänger bekannt waren. Die Informationsdichte betrug allerdings bis zu 30 E-Mails pro Tag – und das auch sonn- und feiertäglich!

Für die Kommunikation mit anderen Gruppen kam das Medium E-Mail ohne Weiteres nicht in Frage. Zum Einen waren nicht jedem Teilnehmer die Namen aller anderen Teilnehmer bekannt, geschweige denn deren E-Mail Adressen. Eine Kommunikation ausschließlich per E-Mail hätte also langfristig bei den meisten Teilnehmern zu fehlenden Informationen geführt. Zum Anderen waren auch nicht jedem Teilnehmer die Zuständigkeiten der anderen Teilnehmer bekannt. Langfristig wären also Fragen und Entscheidungen an die falschen Personen delegiert worden, was zu Abstimmungsproblemen geführt hätte. Von den zusätzlichen Negativfaktoren, die fehlende Informa-

tionen und falsche Entscheidungen in Form von Motivationseinbußen oder gar Frustration mit sich bringen, ganz zu schweigen.

Für die Kommunikation zwischen den Gruppen stellte sich statt dessen die Nützlichkeit einer externen Mailingliste heraus, welche zwei Mitglieder der Labyrinthgruppe für dieses Projekt eingerichtet hatten. Dadurch besaß nun jeder Praktikumsteilnehmer die Möglichkeit, die anderen Teilnehmer schnell, kostengünstig, zuverlässig und nachvollziehbar zu kontaktieren.

Mit Beginn der Implementierung entstand in der Labyrinthgruppe Bedarf für den Austausch von Dateien. Nachdem die ersten Versuche, diese Dateien per Mail zu versenden, kläglich gescheitert waren, erschien der Austausch über einen externen FTP- und HTTP-Server probat, den wiederum zwei Mitglieder der Labyrinthgruppe eingerichtet hatten.

Mitte Dezember 2001 entstand dann langsam der Bedarf für den Austausch von Dateien zwischen den verschiedenen Gruppen. Die ersten Wochen funktionierte der Austausch über den zentralen FTP- und HTTP-Server, allerdings erforderte der gleichberechtigte Zugriff vieler Personen auf eine einzige, zentrale Dateiablage strenge Konventionen bzw. Verhaltensanweisungen (z.B. Dateinamens- und Verzeichniskonventionen), die mitunter auch zu heftigen Diskussionen und Missstimmung führten.

Trotz der extrem strengen Konventionen konnte diese zentrale Art der Dateiablage nicht langfristig funktionieren, und endete gegen Ende des Dezembers 2001 im Chaos: Die Dateimenge auf dem Server nahm überhand, niemandem war mehr klar, wo die aktuellste Version einer Datei zu finden war. Änderungen an Dateien und damit an Schnittstellen wurden von Teilnehmern oftmals überhaupt nicht wahrgenommen, und wenn doch, nicht gefunden, oder gar von anderen Teilnehmern wieder mit alten Versionen überschrieben. Als Nebeneffekt der vielen vollständigen, und zum Teil auch überflüssigen Up- und Downloads stieg der Netzverkehr dieses Servers zwischenzeitlich bis auf 2 Gigabyte/Monat an.

Hier Bildschirmfotos des nur scheinbar aufgeräumten FTP-Bereiches, und des tatsächlichen Dateichaos' dahinter gegen Ende Dezember:

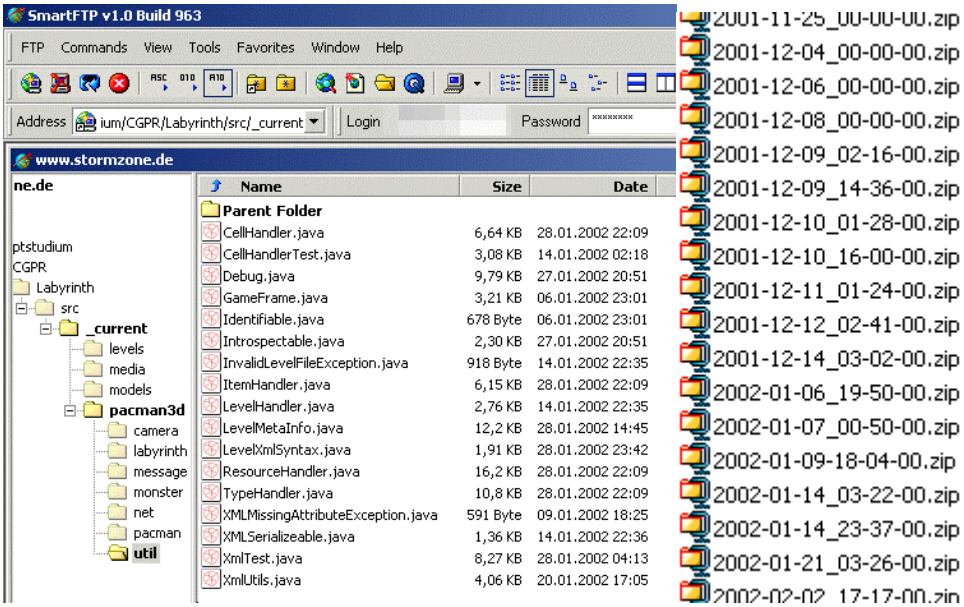

Abbildung 35 - FTP und das Dateichaos

2.4 Januar 2002: CVS

Sylvester 2001 richteten zwei Mitglieder der Labyrinthgruppe zur Lösung der obengenannten Probleme das freie Versionskontrollsystem CVS (http://www.cvshome.org/) auf einem weiteren, ebenfalls externen Server ein.

In den folgenden Wochen vergab die Labyrinthgruppe CVS-Konten für alle Praktikumsteilnehmer, und kümmerte sich um die Hilfestellung bei Installation und Benutzung des Systems durch die Teilnehmer.

Das folgende Bildschirmfoto zeigt einen CVS-Client in Aktion:

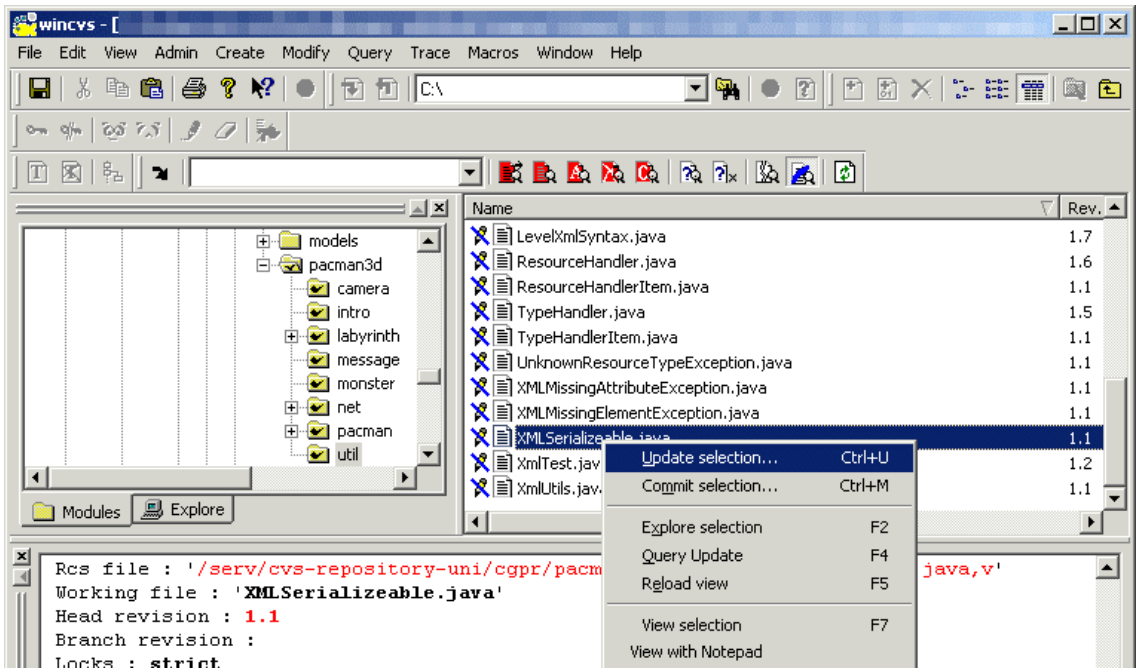

Abbildung 36 - Quellcodeverwaltung mit CVS!

Der lange Weg, und der unermessliche Aufwand durch die Administrationstätigkeiten lohnten sich schließlich:

Seit dem Einsatz von CVS war es ein Leichtes, die von anderen Teilnehmern modifizierten Dateien zu aktualisieren, bzw. seine eigenen Veränderungen zu propagieren. Es wurde sogar möglich, den Programmcode der anderen Teilnehmern zu bearbeiten, was sich als extrem hilfreich für die Veränderung eigener Schnittstellen herausstellte.

3 Anwenderdokumentation

3.1 Programmstart

3.1.1 Kommandozeilenparameter

Pacman 3D unterstützt beim Programmstart die folgenden Kommandozeilenparameter:

Parametername	Optionen	Aufgabe
`--help`		zeigt eine Liste der möglichen Kommandozeilenparameter, ihrer Optionen und Funktionsweise
`--withoutintro`		startet Pacman 3D ohne Intro und lädt direkt das Default-Level oder das mit dem **`--levelfile`** übergebene Level
`--levelfile`	**`filename`**	lädt das angegebene Level mit dem Dateinamen **`filename`**, wenn der Parameter **`--withoutintro`** ebenfalls angegeben wurde. Zu beachten ist, dass der Pfad relativ zum aktuellen Verzeichnis gesetzt sein muss (also beispielsweise `level/level_10x10.p3d`)
`--withmouselistener`		aktiviert Mouselistener für die gesamte Spielszene, mit denen Rotation, Translation und Skalierung vorgenommen werden kann
`--wireframesonly`		statt der ausgefüllten Zellen des Labyrinths werden nur Linienquader gezeichnet, was insbesondere für große Level auf leistungsschwächeren Rechnern interessant ist
`--debuglevel`	**`notice | warning | critical`**	setzt den Debugmodus des Spiels. **`critical`** beinhaltet schwere Fehler, **`warning`** in bestimmten Situationen ungünstige aber nicht kritische Probleme und **`notice`** einfache Statusmeldungen.
`--servermode`	**`host`**	startet ein Netzwerkspiel im Servermodus. Der Zusatz **`host`** ist eine IP-Adresse oder ein Hostname, mit dem der Server auf

Parametername	Optionen	Aufgabe
		Clientanfragen horchen soll (nötig, wenn mehrere IPs im lokalen Rechner vorhanden sind)
--clientmode	**host**	startet ein Netzwerkspiel im Clientmodus. Der Zusatz **host** ist die IP-Adresse bzw. der Hostname des Zielservers
--monsteramount	**N**	setzt die Anzahl der Monster auf den Zusatz **n**. Es werden allerdings nur maximal so viele Monster geladen, wie im Level vorgesehen. Im Netzwerkspiel hat die Einstellung nur Auswirkungen auf dem Rechner, auf dem die Monster erzeugt werden (d.h. auf den Serverrechner)
--demomode		führt den Demomodus des per Kommandozeile gesetzten oder im Intro gewählten Levels aus
--recordfile	**filename**	speichert das Level nach Beendigung des Spiels im angegeben **filename** ab und legt darin die aufgezeichneten Tastaturkommandos zur Pacmansteuerung als Demomodus-Daten ab

3.1.2 Kommandozeilenparameterbeispiele

Mit dem folgenden Aufruf wird das Level aus der Datei `levels/level_10x10.p3d` geladen, dabei auf das Intro verzichtet und zum Rendern der Drahtmodell-Modus gewählt:

```
game --withoutintro --levelfile levels/level_neon.p3d –wireframesonly
```

Bei diesem Programmaufruf werden Mouselistener zur Rotation, Translation und Skalierung in die gesamte Spielszene eingefügt und das Debuglevel wird auf **critical-**Meldungen setzen (wodurch **notice-** und **warning-**Meldungen unterdrückt werden).

```
game --withmouselistener --debuglevel critical
```

Startet ein Netzwerkspiel als Server mit dem angegebenen Level und geht dabei direkt ohne Intro ins Spiel.

```
game --servermode 141.2.114.129 –-withoutintro –-levelfile levels/test.p3d
```

Startet ohne das Intro mit dem angegebenen Level im selbstablaufenden Demomodus:

```
game –-withoutintro --levelfile levels/level_kingsize_demo.p3d --demomode
```

3.2 Der Leveleditor

...der Ursprung aller Leveldateien!

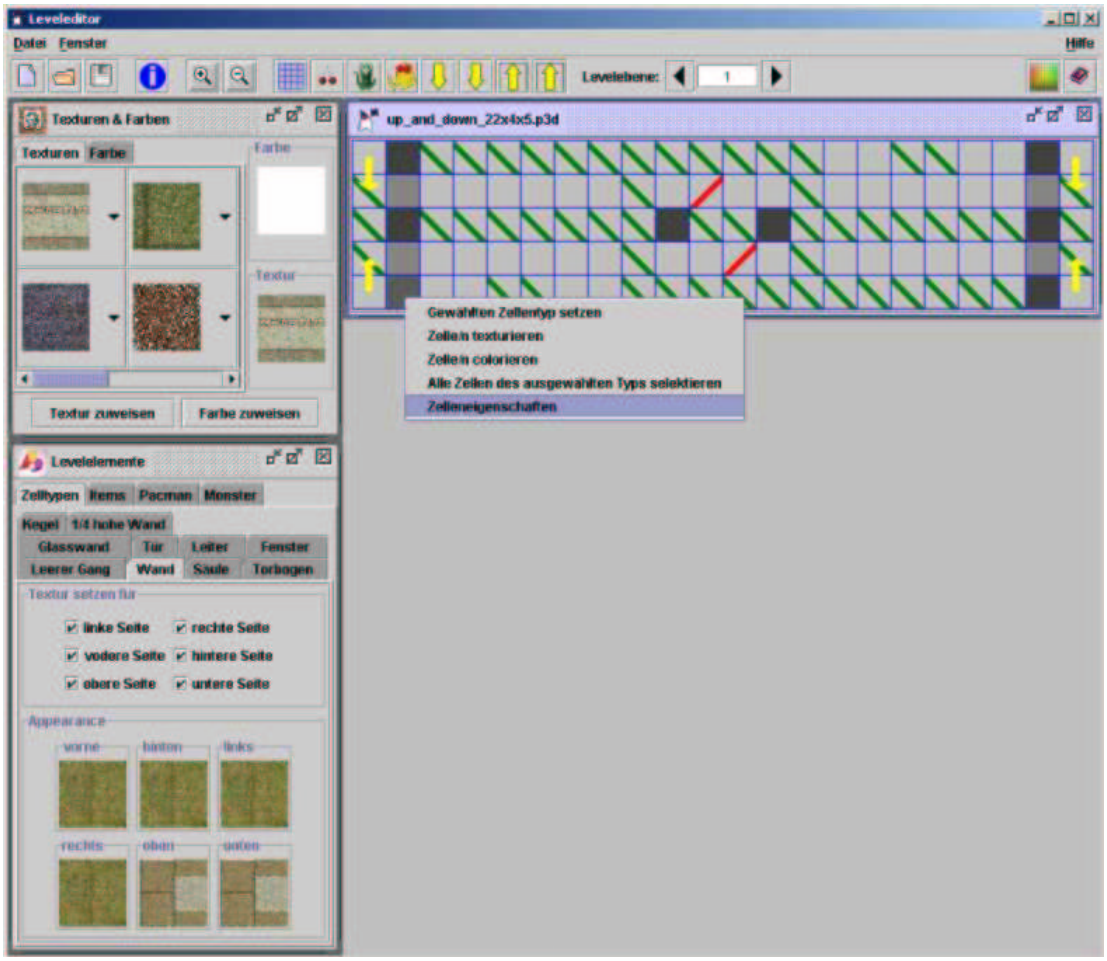

Abbildung 37 - Der Leveleditor in Aktion!

3.2.1 Einleitung

Der Leveleditor war nicht Bestandteil der Aufgabenstellung, doch kam die Labyrinthgruppe schnell zu der Einsicht, dass ein optionaler Leveleditor sinnvoll sei. Grund dafür war, dass Leveldatei mittels einer XML-Sprache definiert werden sollten (siehe 4.8 Leveldateien), und dass offensichtlich ein Labyrinth visuell schneller und intuitiver erstellt werden kann, wenn dies mittels eines WYSIWYG[13]-Werkzeuges erfolgen kann. Hierzu gab es zwei mögliche Ansätze, und zwar den Leveleditor als ein 3D-Editierungstools bzw. als ein 2D-Editierungswerkzeug zu konstruieren. Die erste Möglichkeit schied allerdings aus, da sie erstens für eine einzige Person einen zu hohen Programmieraufwand bedeutet hätte und zweitens Erfahrungen und Versuche mit Java 3D zeigten, dass Java 3D sehr ressourcenhungrig ist und somit schon für die Labyrintherstellung ein hochperformanter Rechner notwendig gewesen wäre. Somit ist der Leveleditor in seiner jetzigen Form - als 2D-Editierungswerkzeug - entstanden.

[13] what you see is what you get

Im Grunde genommen stellt der Levleditor ein Werkzeug dar, um die verschiedenen Felder einer Instanz der Java-Klasse `pacman3d.labyrinth.Labyrinth` zu lesen, zu verändern, zu schreiben und darzustellen. Bei der Editierung eines Labyrinthes erfolgt also eine Änderung des Objektzustandes einer Instanz der Klasse `pacman3d.labyrinth.Labyrinth`. Die visuelle Bearbeitung eines Labyrinthes erfolgt hierbei - bedingt durch den 2D-Ansatz - ebenenweise. Hierzu kann der Benutzer die verschiedenen Labyrinthebenen auswählen, welche daraufhin auf dem Bildschirm in Quadrate unterteilt dargestellt werden. Jedes Quadrat stellt hierbei eine Zelle dar (siehe 4.5 Zellen), die vom Benutzer durch andere Zellen ersetzt werden kann, wie z.B. eine Säule, Glasswand, Torbogen und Fenster, der Farbe und Textur zugewiesen werden können, und der verschiedene Items (siehe 4.6 Items), Monster und Pacmanstartpositionen hinzugefügt werden können.

3.2.2 Die Klassen des Levleditors

Insgesamt haben alle Klassen des Levleditors, nämlich:

- `pacman3d.labyrinth.levleditor.AppearanceJPanel`,
- `pacman3d.labyrinth.levleditor.CellsJPanel`,
- `pacman3d.labyrinth.levleditor.CellVector`,
- `pacman3d.labyrinth.levleditor.ColorChooser2DComponents`,
- `pacman3d.labyrinth.levleditor.ExampleFileFilter`,
- `pacman3d.labyrinth.levleditor.ItemJPanel`,
- `pacman3d.labyrinth.levleditor.JToolBarLevleditor`,
- `pacman3d.labyrinth.levleditor.LevelEditingJPanel`,
- `pacman3d.labyrinth.levleditor.Levleditor`,
- `pacman3d.labyrinth.levleditor.LeveledtiorHelpJPanel`,
- `pacman3d.labyrinth.levleditor.LevelMetaInfoJPanel`,
- `pacman3d.labyrinth.levleditor.LevelSizeDialog`,
- `pacman3d.labyrinth.levleditor.MonsterJPanel`,
- `pacman3d.labyrinth.levleditor.MyFileFilter`,
- `pacman3d.labyrinth.levleditor.PacmanJPanel`,
- `pacman3d.labyrinth.levleditor.PreviewJLabel` und
- `pacman3d.labyrinth.levleditor.TextureSelectionJPanel`

gemeinsam mehr als 9.000 Codezeilen! Die wichtigsten Klassen des Leveleditors und deren Funktionen sind:

- `pacman3d.labyrinth.levleditor.LevelEditingJPanel`

Die Hauptklasse und gleichzeitig das Herzstück des Leveleditors stellt die Klasse `LevelEditingJPanel` dar, die mit ihren 2.255 Codezeilen den Hauptteil der Gesamtcodezeilen trägt. Diese Klasse ist insofern das Herzstück des Leveleditors, da durch sie die 2D-Editierung eines Labyrinths erfolgt. Diese Klasse stellt ein zu editierendes Labyrinth ebenenweise dar.

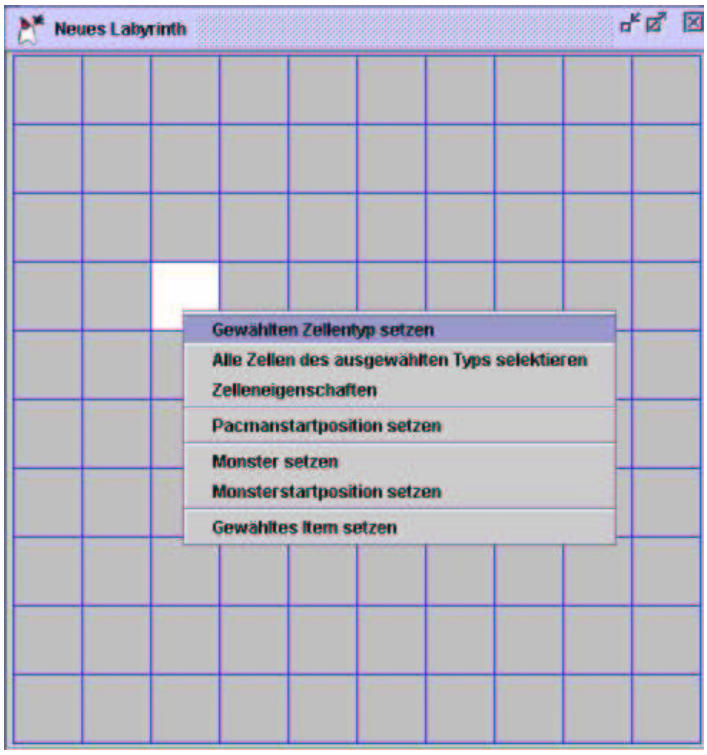

Abbildung 38 - Festlegen einer Zelle

Durch diese Klasse kann der Benutzer mittels der Maus bestimmte Positionen/Zellen in einer Labyrinthebene auswählen und dorthin bestimmte Zellentypen, Items, Pacman oder Monster setzen. Hierzu stehe diverse kontextsensitive Popupmenüs zur Verfügung, wie man z.B. auf dem obigen Bildschirmfoto sehen kann. Eine editierte Labyrinthebene mit verschiedenen Zellentypen, Items, Pacmanstartpositionen und Monstern sieht wie folgt aus:

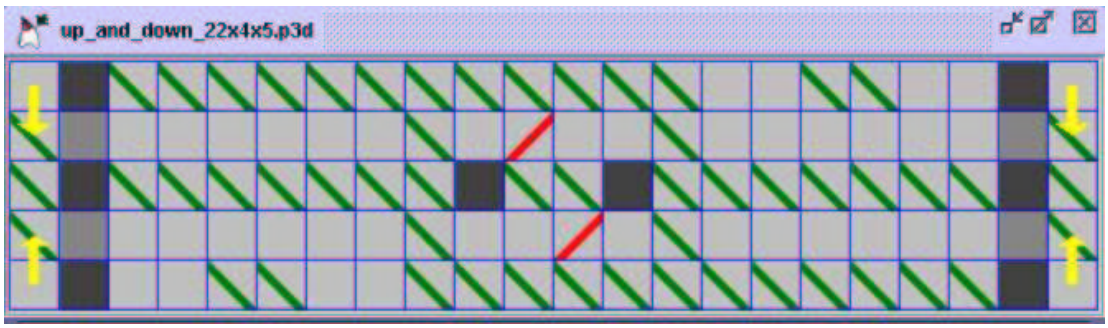

Abbildung 39 - Editierte Labyrinthebene

Zur Verdeutlichung der verschiedenen Labyrinthelemente werden verschiedene Symbole und Farben verwendet. Z.B. bedeuten obige grüne Schrägstriche, dass sich an den entsprechenden Positionen bzw. in den entsprechenden Zellen ein Item befindet, rote Striche, dass sich dort zu Spielbeginn initial ein Monster befindet und die gelben Pfeile stellen Startpositionen für Pacman dar, wobei die Pfeilrichtung angibt, in welche Richtung bei Spielstart ein Pacman blicken soll. Die in verschiedenen Grautönen ausgefüllten Quadrate stellen die verschiedenen Zellen der Labyrinthebene dar. Alle Symbolfarben können allerdings vom Benutzer beliebig eingestellt werden, um eine bessere Unterscheidung der gesetzten Zellentypen zu ermöglichen.

- `pacman3d.labyrinth.levelеditor.Levelеditor`

In einem anderen Sinn stellt die Klasse `Leveleditor` ebenfalls die Hauptklasse des Leveleditors dar, wie der Namen bereits vermuten lässt. Diese Klasse trägt für das Zusammenspiel aller Klassen des Leveleditors Sorge. Sie kümmert sich um die korrekte Initialisierung, Interaktion und vor allem um die Kommunikation zwischen den einzelnen Klassen des Leveleditors. Des weiteren stellt diese Klasse das Hauptfenster des Leveleditors dar.

- `pacman3d.labyrinth.leveleditor.CellsJPanel` und `pacman3d.labyrinth.leveleditor.ItemJPanel`

Die Klassen `CellsJPanel` und `ItemJPanel` dienen zur Darstellung der dem Benutzer bei der Labyrintherstellung zur Verfügung stehenden Zellenklassen und Items. Über diese beiden Klassen erfolgt die Auswahl einer bestimmten Zellenklasse bzw. Itemklasse, als auch bei Selektion einer Zelle bzw. Items auf dem 2D-Editierungsbereich, eine Anzeige, welche Zellenklasse bzw. Itemklasse selektiert ist und welche Eigenschaften diese trägt, wie z.B. gesetzte Texturen oder Blinklichtfarbe.

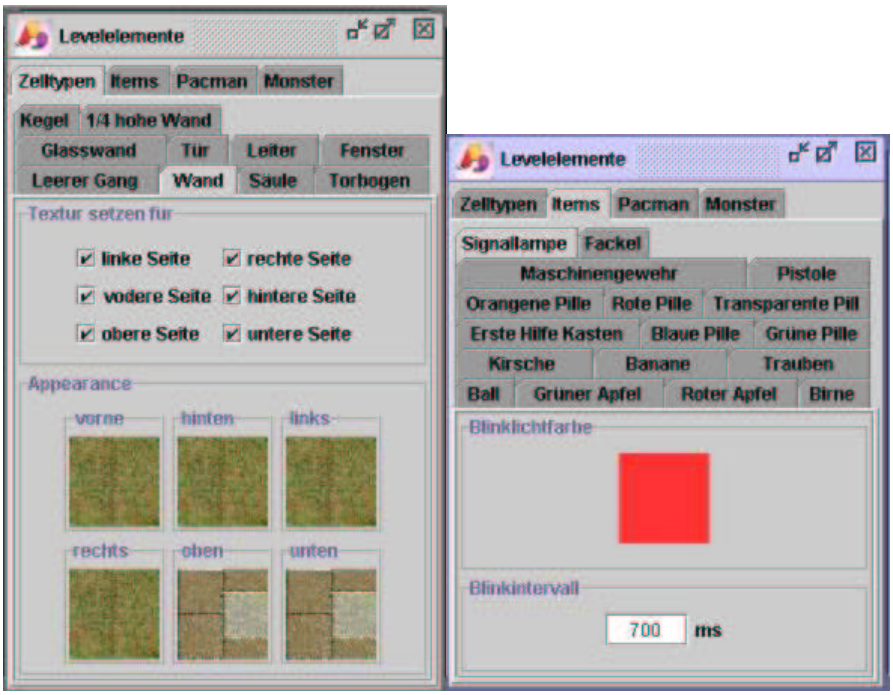

Abbildung 40 - Zell- und Itemeigenschaften

- `pacman3d.labyrinth.leveleditor.TextureSelectionJPanel`

Die Klasse `TextureSelectionJPanel` stellt diejenigen Komponenten zur Verfügung, damit eine Auswahl einer Textur oder Farbe durch den Benutzer erfolgen kann, um diese ausgewählten Zellentypen zuzuweisen.

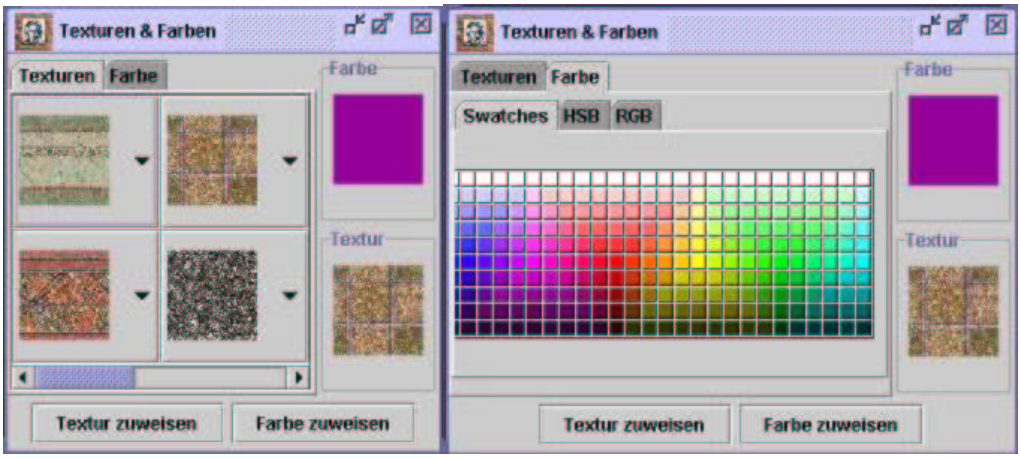

Abbildung 41 - Texturen & Farben

Die übrigen oben aufgeführten Klassen sind zwar nicht ganz unerlässlich für den Leveleditor, doch kommt ihnen eher eine nebenläufige Bedeutung zu, wie z.B. die Darstellung der Werkzeugleiste durch die Klasse `pacman3d.labyrinth.levelеditor.JToolBarLevelеditor`:

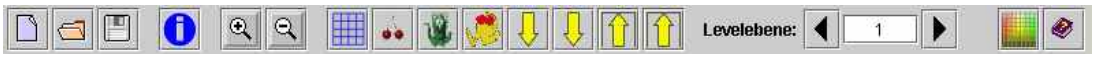

Abbildung 42 - Werkzeugleiste des Leveleditors

sowie die Anzeige eines Hilfefensters durch die Klasse `pacman3d.labyrinth.-leveleditor.LeveleditorHelpJPanel`, in welchem die Bedienung des Leveleditors erklärt wird:

Abbildung 43 - Hilfefenster des Leveleditors

3.2.3 Ausblick und mögliche Verbesserungen

1. **2D-Ansicht:**

   Das größte Problem der derzeitigen Version des Leveleditors ist die 2D-Ansicht: der Benutzer besitzt bei der 2D-Levelbearbeitung keine Möglichkeit, sich eine von ihm editierte Labyrinthebene vorzustellen, d.h. zu sehen, welche Zellentypen gesetzt sind, und vor allem, welche Texturen verwendet worden sind.

   Ersteres Problem ist zwar durch die Verwendung verschiedener Farben für die verschiedenen Zellentypen handhabbar – sofern sich der Benutzer in die Bedienung des Leveleditors eingearbeitet hat -, aber das zweite Problem ist nicht lösbar, so lange der Leveleditors ein reines 2D-Editierungswerkzeug ist, da sonst der Benutzer mit einer Flut verschiedener Symbole und Farben konfrontiert werden müsste.

   Somit wäre es wünschenswert, den Leveleditor als ein 3D-Editerungswerkzeug zu gestalten, so dass ein dreidimensionales Labyrinth interaktiv vom Benutzer in einer 3D-Ansicht konstruiert werden könnte. Doch ist dies scheinbar zur Zeit nicht mit Java 3D und der aktuellen Hardware realisierbar zu sein, falls man eine kurze Reaktionszeit vom Leveleditor erwünscht, um ein schnelles interaktives Arbeiten mit dem Leveleditor zu ermöglichen.

2. **Monsterklassen und Jagdstrategien:**

   Zur Erstellungszeit des Leveleditors bestand leider noch nicht die Möglichkeit, in einer Leveldatei abzuspeichern, welche Monsterklassen vom Benutzer gesetzt wurden, und welche Eigenschaften diese haben. Somit kann mit dem Leveleditor zur Zeit nur die „Standardmonsterklasse“ mit der Strategie „Jagen“ gesetzt werden. Sollte der Leveleditor weiterentwickelt werden, so sollte dieses Features realisiert werden, da sonst weiterhin eine Editierung einer Labyrinthdatei mittels eines XML-Editors erfolgen müsste, um andere Monsterklassen und/oder Jagdstrategien verwenden zu können.

3. **Introspektion:**

   Die verschiedenen Zellen-, Item-, Monster- und Pacmanklassen bieten die Möglichkeit der Introspektion, welche zu Beginn der Konzeption dazu verwendet werden sollte, ohne zusätzlichen Programmieraufwand neue Zell- und Itemklassen im Leveleditor verwenden zu können. Hierzu wären die einstellbaren Parameter der neuen Zell- bzw. Itemklassen abzufragen und entsprechende GUI-Komponenten vom Leveleditor zwecks Einstellung zur Verfügung zu stellen.

   In der aktuellen Version des Leveleditors wurde keine Introspektion verwendet, weil die Auswahl der passenden GUI-Komponenten zur Einstellungen der zur Verfügung stehenden Parameter innerhalb des Leveleditors hätte erfolgen müssen, als auch die Überprüfung, ob vom Benutzer gemachte Eingaben sinnvoll sind (Validierung), um z.B. den Leveleditor robuster zu gestalten.

   Das Konzept der Introspektion sollte bei eventueller Weiterentwicklung des Leveleditors einfließen, allerdings sollte auch in Erwägung gezogen werden, das Konzept der JavaBeans zu übernehmen, so dass jede verwendbare Klasse ihre eigenen GUI-Komponenten zur Einstellung mitbringen sollte, da es nur so möglich ist, auf alle Anforderungen bezüglich Datentypen und Eingabekomfort einzugehen.

3.2.4 Erfahrungen mit dem Leveleditor

Was die Editierung eines Labyrinthes anbetrifft, so hat sich der Leveleditor als ein gut handhabbares Werkzeug zur schnellen Erstellung eines Labyrinthes herausgestellt, so dass nach einiger Einarbeitungszeit recht abwechslungsreiche und interessante Labyrinthe erstellt werden konnten, und zwar fast beliebiger Größen - insofern das erzeugte Labyrinth nicht mittels Java 3D dargestellt werden sollte.

Es stellte sich nämlich heraus, dass Labyrinthe der Abmessungen 15x15x15 Zellen aufwärts, schnell einen Speicherplatzbedarf von weit über 100 MB haben, sogar schnell Größen von 200 MB annehmen können und somit leider nicht mehr spielbar sind.

Solche große Labyrinthe sind ohne nennenswerten Unterschied weder auf einem Athlon 800 MHz, 384 MB RAM und GeForce I, noch auf einem Pentium IV 1600 MHz, 512 MB RAM und GeForce III spielbar, was wohl darauf zurückzuführen ist, dass Java 3D zur Darstellung des Labyrinths einen immensen Speicherplatzbedarf hat. Genau diese Tatsache ist auch der Grund, warum der Leveleditor kein 3D-Editierungswerkzeug wurde, da der hohe Speicherplatzbedarf langsame Reaktionszeiten des Leveleditors zur Folge gehabt hätte und somit eine interaktive Bearbeitung durch den Benutzer fast unmöglich gewesen wäre.

3.2.5 Entstehung des Leveleditors

Das Bedienungskonzept des Leveleditors wurde iterativ während der Entstehung desselben entwickelt; in dem Sinne, dass es der Entwickler am Intuitivsten fand. Dies mag der Grund sein, warum die Bedienung auf viele Benutzer recht eigenartig wirkt. Dies war jedenfalls des öfteren von anderen Teilnehmern des Praktikums zu vernehmen, die den Leveleditor ausprobiert hatten.

Das vorherige Studium eines bereits existierenden und erprobten Leveleditors wäre in diesem Zusammenhang vielleicht hilfreich gewesen.

3.2.6 Mit dem Leveleditor erzeugte Labyrinthe

- „Climb up the pyramid“ (15x8x15 Zellen):

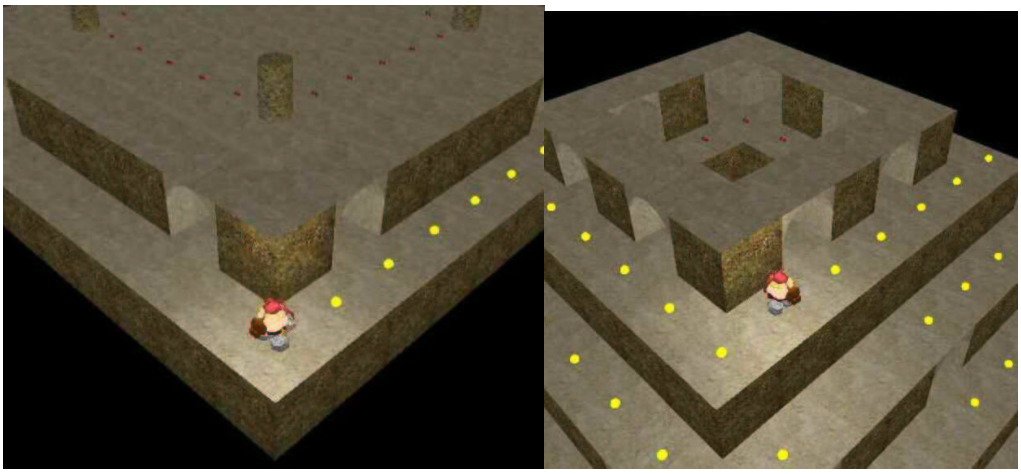

Abbildung 44 - Level "Climb up the pyramid"

- „Up and down“ (22x4x5 Zellen):

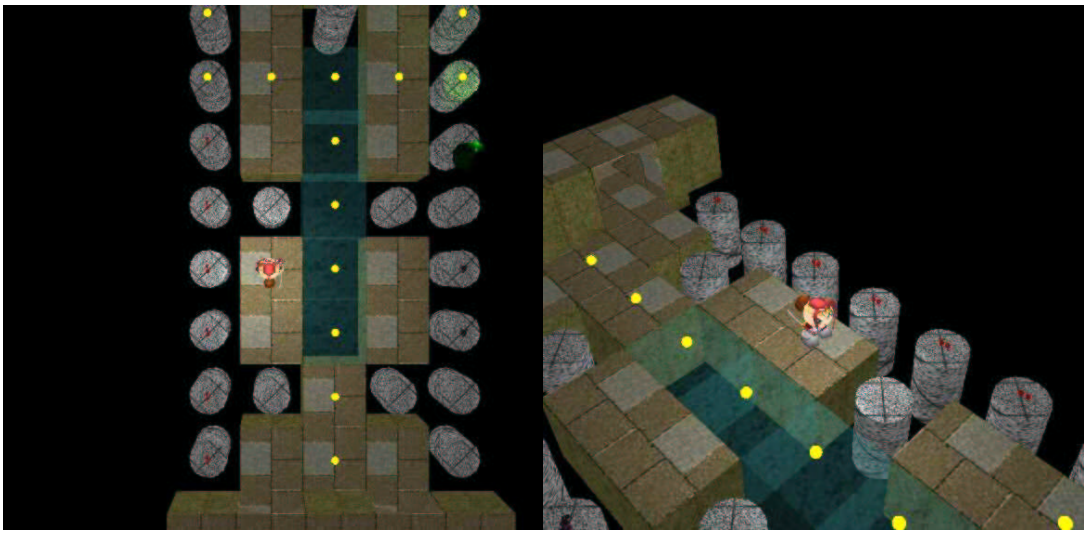

Abbildung 45 - Level "Up and down"

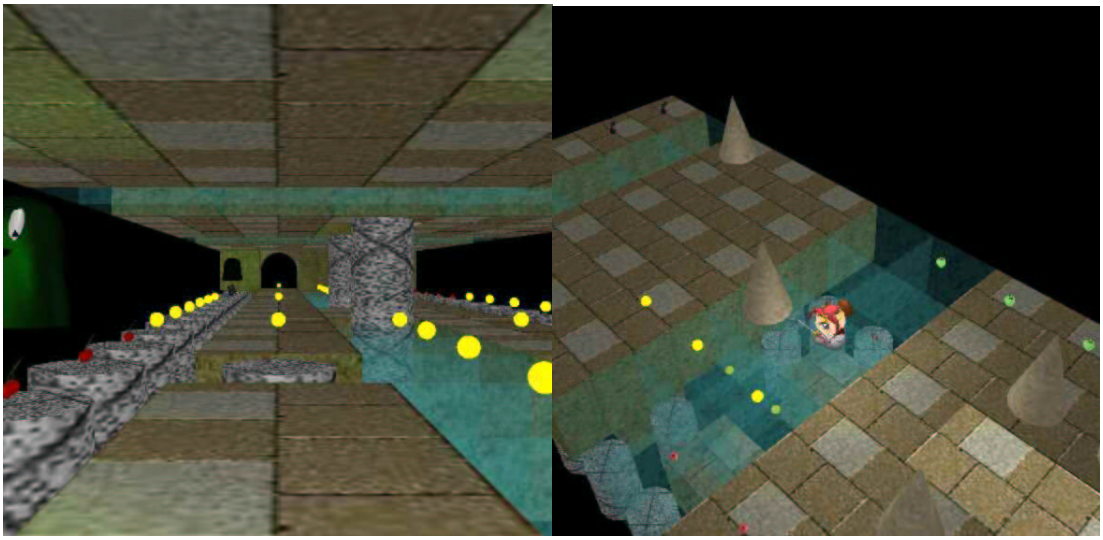

Abbildung 46 - Level "Up and down"

- „Nearly traditional“ (15x3x15 Zellen):

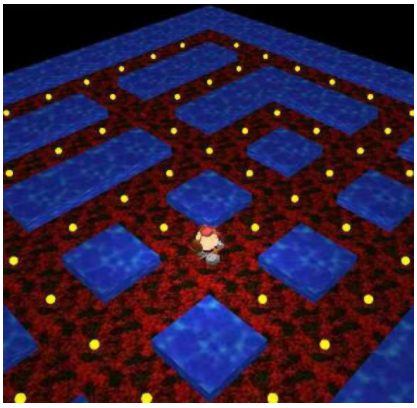

Abbildung 47 - Level "Nearly traditional"

4 Technische Dokumentation

4.1 Die Quellcodestruktur

...Ordnung im Chaos!

Alle Klassen des zu Pacman 3D gehörenden Quellcodes befinden sich im Java-Package `pacman3d`, und in Packages darunter. Der Teil der Klassen, welcher durch die Labyrinthgruppe entwickelt worden ist, befindet sich in den Packages `pacman3d`, `pacman3d.labyrinth`, `pacman3d.labyrinth.cells`, `pacman3d.labyrinth.items`, `pacman3d.labyrinth.leveleditor`, `pacman3d.labyrinth.message` und `pacman3d.labyrinth.util`.

Die einzelnen Packages enthalten Quellcodedateien der folgenden Funktionen:

- `pacman3d`

  Das Package `pacman3d` ist die Wurzel der Package-Struktur von Pacman 3D, und ist demzufolge auch als erstes Package des Projektes entstanden. Es beinhaltet einerseits alle anderen Packages von Pacman 3D, und andererseits die Startdatei der Anwendung (siehe 4.3 Die Startdatei).

- `pacman3d.labyrinth`

  Das Package `pacman3d.labyrinth` folgte chronologisch gesehen auf das Package `pacman3d`, und beinhaltet die Kernfunktionalität des Labyrinthes. Dazu gehören die Datenstrukturen des Labyrinthes sowie die Schnittstellen für die anderen Praktikumsgruppen.

- `pacman3d.labyrinth.leveleditor`

  Losgelöst von allen anderen Packages begann die Entwicklung eines grafischen Editors für die Leveldateien in dem Package `pacman3d.labyrinth.leveleditor`.

- `pacman3d.util`

  Von mehreren Packages gemeinsam genutzte „Hilfsklassen“ (z.B. diverse Exceptionklassen) wurden in das Package `pacman3d.util` verschoben, um die Übersichtlichkeit der anderen Packages nicht durch Überfrachtung zu gefährden.

- `pacman3d.labyrinth.message`

  Das Package `pacman3d.labyrinth.message` enthält die Funktionalität für das Nachrichtensystem (siehe 4.7 Nachrichten) und folgte in der Entwicklungsreihenfolge dem Package `pacman3d.labyrinth`. Es wurde entwickelt sobald der Bedarf für ein Nachrichtensystem festgestellt worden war, und Selbiges konzipiert worden war.

- `pacman3d.labyrinth.cells`, `pacman3d.labyrinth.items`

  Die Packages `pacman3d.labyrinth.cells` und `pacman3d.labyrinth.items` entstanden chronologisch zu letzt, und beinhalten diverse Klassen für Zellen und Items (siehe 4.5 Zellen, 4.6 Items). Bis dato wurden Zellen und Items im Package `pacman3d.labyrinth` vorgehalten, welchem aber durch die stetig zunehmende

Anzahl an Zellen und Items die Unübersichtlichkeit drohte. Bereits entstandene Zellen und Items wurden in die neuen Packages verschoben.

Gemäß der in Java üblichen Konvention bezüglich von Dateinamen und Verzeichnissen werden alle Klassen in Dateien ihres jeweiligen Namens mit der Dateiendung „`.java`“ bzw. „`.class`“ für die Quelle bzw. das Kompilat der Klasse gehalten. Beispielsweise befindet sich der Quellcode der Klasse `pacman3d.Game` in der Datei `Game.java` im Verzeichnis `pacman3d`.

Für nähere Informationen existiert eine online verfügbare Javadoc-Dokumentation.

4.2 Systemstruktur

...und wie arbeiten die Systemkomponenten mit dem Labyrinth zusammen?!

Das folgende Strukturdiagramm zeigt die verschiedenen Systemkomponenten und ihre Beziehungen untereinander:

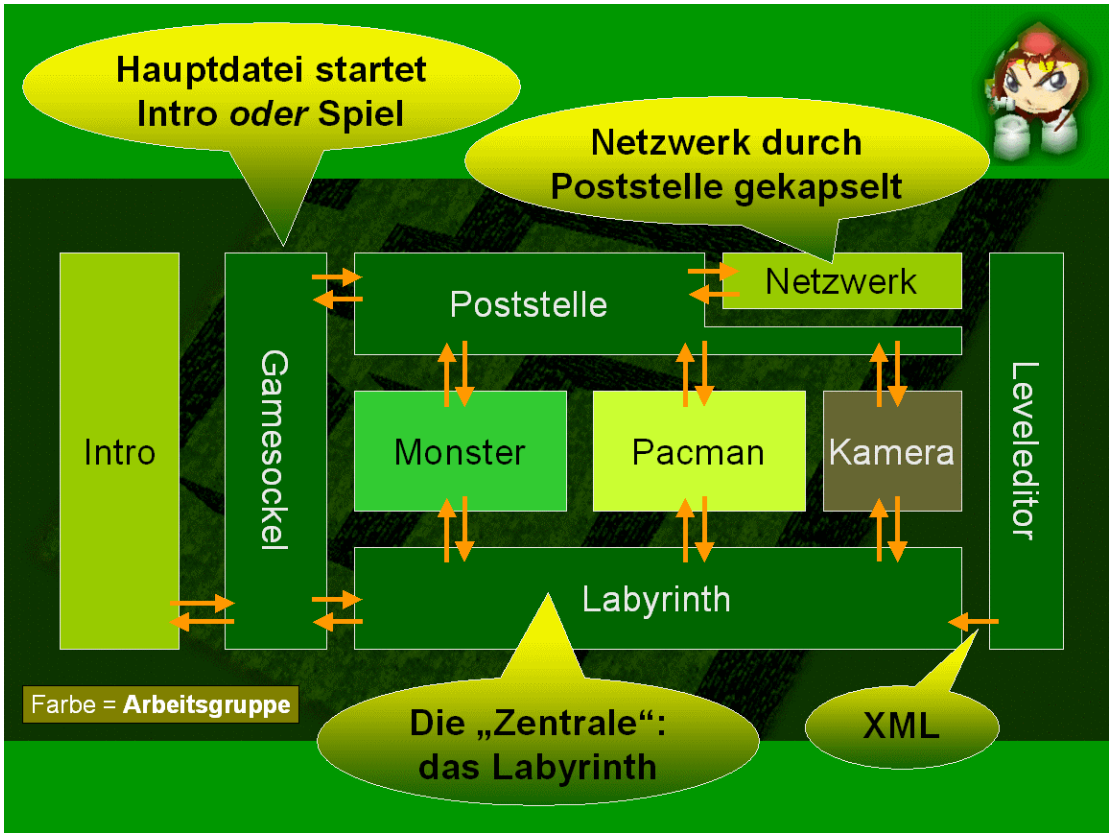

Abbildung 48 – Systemstruktur

Es ist zu sehen, wie die das Netzwerk vollständig kapselnde Poststelle als zentraler Ansprechpartner für die Monster, die Pacman, die Kamera und die Startdatei (hier „Gamesockel“ genannt) fungiert (`pacman3d.message.MessageService`).

Eine ähnlich zentrale Rolle wie die Poststelle nimmt im Gesamtfluss das Labyrinth ein (`pacman3d.labyrinth.Labyrinth`), welches ebenfalls von den genannten Komponenten verwendet wird.

Die einzige Schnittstelle zum Leveleditor hin besteht im Labyrinth, der sich die Datenhaltungsklassen sowie Serialisierung und Deserialisierung (XML) mit dem Labyrinth teilt (siehe 3.2 Der Leveleditor, 4.4 Das Labyrinth, 4.8 Leveldateien).

4.3 Die Startdatei

...die Mutter aller Klassen!

Pacman 3D wird mittels der Java-Klasse `pacman3d.Game` gestartet (siehe 3.1 Programmstart). Das Verhalten der Anwendung kann durch zahlreiche Kommandozeilenparameter beeinflusst werden, siehe 3.1.1 Kommandozeilenparameter.

4.4 Das Labyrinth

...die Welt des Pacman!

Die Welt des Pacman besteht aus einer homogenen, diskreten und begrenzten Struktur, wie dies seit der Geburt des Pacman anno 1980 der Brauch ist. Neu für den Pacman ist hingegen die Abkehr vom planaren Spielfeld hin zur räumlichen Welt. Gegenüber dem flachen Spielfeld von einst besitzt der Pacman nun einen Freiheitsgrad mehr, um seinen Verfolgern zu entkommen.

Der Pacman und die Monster bewegen sich innerhalb eines Labyrinthes, welches technisch durch eine dreidimensionale Struktur von Spielfeldern repräsentiert wird. Jeder Pacman und jedes Monster halten sich zu jedem Zeitpunkt auf exakt einem Spielfeld auf, niemals kann sich ein Pacman oder ein Monster „zwischen“ zwei Spielfeldern befinden. Es ist jedoch durchaus möglich, dass sich mehrere Pacman und/oder Monster auf demselben Spielfeld aufhalten; ist jedoch mindestens ein Monster auf einem Spielfeld, sterben alle Pacman, welche sich auf dem selben Spielfeld befinden.

Das Labyrinth wird primär durch die Klassen im Package `pacman3d.labyrinth` implementiert, hervorzuheben ist hier insbesondere die Klasse `pacman3d.labyrinth.Labyrinth`.

4.5 Zellen

...quadratisch, praktisch, gut!

Wie in 4.4 Das Labyrinth erklärt, besteht die Welt des Pacman aus einer dreidimensionalen und diskreten Struktur von „Spielfeldern“. In der Pacman 3D-Terminologie wird ein Spielfeld als „Zelle“ bezeichnet, bzw. im Quellcode die englische Bezeichnung „Cell“ verwendet.

Jede Zelle (d.h. jedes Spielfeld) besitzt die gleiche, einheitliche Größe (1,0*1,0*1,0) und wird durch eine Instanz der Klasse `pacman3d.labyrinth.cells.Cell`, oder eines ihrer Nachfahren erzeugt. Ein Spielfeld der Größe 10*15*3 besteht also aus 450 Instanzen von `pacman3d.labyrinth.cells.Cell` oder eines Nachfahren davon, und besitzt die Java 3D-Abmessungen 10,0*15,0*3,0.

Zellen sollten sich alleine der Übersichtlichkeit halber im Package `pacman3d.labyrinth.cells` befinden, wenngleich dieses keine technische Notwendigkeit ist.

Gegenwärtig sind 22 Klassen von Zellen verfügbar, dazu gehören beispielsweise:

- `Cell`

  Basisklasse aller anderen Zellen, die jedoch als „abstrakt“ deklariert ist, und daher nur zum Ableiten weiterer Zellen verwendet werden kann.

- `CellFloor`

  Eine „leere“ Zelle, die keinerlei Inhalt hat. Wird verwendet, um die Flure in den Welten des Pacman zu erzeugen.

- `CellWall`

  Eine Zelle die einer Wand entspricht.

- `CellLadderEast`

  Eine Zelle die einer nach Osten gerichteten Leiter entspricht.

- `CellHalfWall`

  Eine Zelle die einer Wand mit nur halber Höhe entspricht. Der Pacman kann diese Wand dennoch nicht begehen.

- `CellGlassWall`

  Eine Zelle die einer Wand aus Glas entspricht, die zwar durchsichtig, aber nicht begehbar ist.

- `...`

4.5.1 Aussehen und Verhalten

Jede Zelle verfügt über zwei unterschiedliche Aspekte, unter denen sie betrachtet werden kann, und denen sie genügen muss:

1. **Aussehen:**

   Jede „Zelle“ besitzt ihr eigenes, charakteristisches „Aussehen“, mit dem sie dem Spieler gegenübertritt. Das „Aussehen“ einer Zelle ist derjenige Teil der dreidimensionalen Szene, welcher durch die Zelle selber erzeugt wird, d.h. im technischen Sinne einer durch die Zelle frei definierbare Menge von Java 3D-Objekten.

   Je nach Kamera, Spielfeld und Position des Pacman sieht der Benutzer Zellen von innen, oder auch von außen.

   Bei der Definition ihres Aussehens sind einer Zelle keinerlei Grenzen gesetzt: sie kann z.B. Modelle aus VRML- oder 3DS-Dateien laden, genauso gut kann sie Modelldaten auch fest einkompiliert haben. Zellen können parametrisiert werden, wenn der Entwickler der Zellklasse dieses für hilfreich hält (siehe 4.8 Leveldateien). Entwickler von Zellklassen sollten jedoch beachten, die zelltypische Größe von 1,0x1,0x1,0 nicht zu überschreiten.

   Zur Zurückgabe des Aussehens muss jede Zelle die Methode `getJ3Dgroup()` unterstützen, welche eine Instanz von dem Java 3D-Standardobjekt `javax.media.j3d.Node` zurückgibt. (Im Nachfolgenden als „Knoten“ bezeich-

net.) Dieser Knoten ist der Wurzelknoten einer hierarchischen Struktur von Java 3D-Knoten, die in ihrer Gesamtheit das Aussehen der Zelle repräsentieren. Eine aufrufende Klasse, im Allgemeinen das Labyrinth, kann den zurückgegebenen Knoten nach Belieben in eine existierende Szene einfügen.

2. **Verhalten:**

   Das „Verhalten“ einer Zelle definiert ihre Aktionen und Reaktionen im Spielverlauf. Insbesondere ist das Verhalten einer Zelle unabhängig von ihrem Aussehen, und es wird vollkommen losgelöst von Java 3D definiert.

   Beispielsweise ist es möglich, Zellen zu entwickeln, die zwar das Aussehen einer Wand besitzen, sich aber dennoch beschreiten lassen, also das „Verhalten“ eines Flures besitzen. Oder umgekehrt: Zellen mit dem Aussehen von Fluren und dem „Verhalten“ von Wänden, die sich also entgegen ihres optischen Anscheines nicht beschreiten lassen.

4.5.2 Entwicklung eigener Zellen

Es können beliebig viele, neue Zellklassen durch Ableitung bestehender Zellklassen erzeugt werden, und auf diese Weise beliebige Spielfelder für den Pacman und die Monster erzeugt werden. Die Anwendung der Zellklassen geschieht durch die Definition entsprechender Leveldateien, siehe 4.8 Leveldateien.

4.6 Items

...kleiner als klein: Items!

Genau wie die im vorangegangenen Abschnitt eingeführten Zellen sind auch „Items“ Gegenstände der dreidimensionalen Szene, die der Benutzer sehen kann, und mit denen er interagieren kann. Im Gegensatz zu Zellen können sich Monster und Pacman aber niemals *innerhalb* von Items befinden, sie können Items stets nur von außen betrachten. Weiterhin sind Items stets vollständig in genau einer Zelle enthalten, was insbesondere impliziert, dass jedes Item kleinere Abmessungen als eine Zelle, d.h. Abmessungen kleiner als 1,0x1,0x1,0 besitzen muss.

Beispiele für Items sind z.B. die aus dem originalen Pacman-Spiel bekannten „Vitaminpillen“, oder auch bekannte Spielelemente wie Erste-Hilfe-Kästen, Früchte, Waffen und Werkzeuge, die ein Pacman benutzen oder gar mitnehmen kann.

Eine Szene voller Items sieht z.B. wie folgt aus:

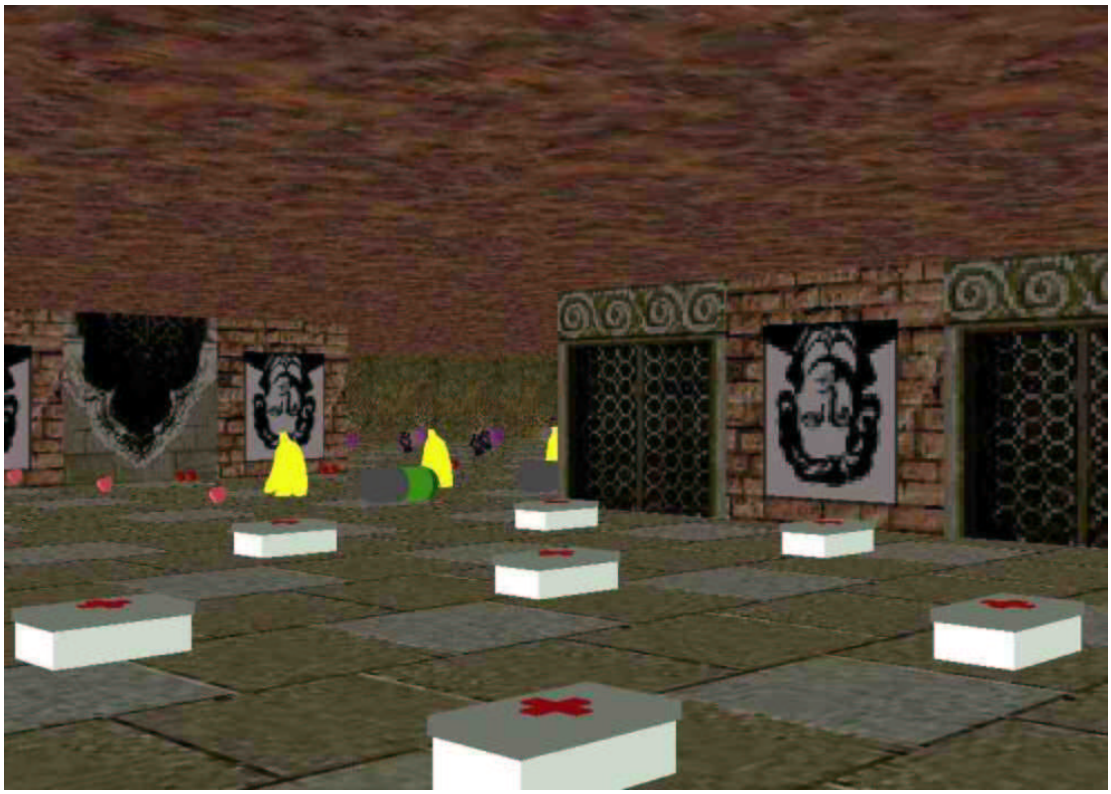

Abbildung 49 - Eine Szene voller Items

Gegenwärtig sind 19 Klassen von Items verfügbar, dazu gehören beispielsweise:

- `Item`

  Basisklasse aller anderen Items, die jedoch als „abstrakt“ deklariert ist, und daher nur zum Ableiten weiterer Items verwendet werden kann.

- `ItemBanana`

  Ein Item welches einen punkterhöhende Banane darstellt.

- `ItemCherry`

  Ein Item welches einen punkterhöhende Kirsche darstellt.

- `ItemFirstAidBox`

  Ein Item welches einen punkteerhöhenden Erste-Hilfe-Kasten darstellt.

- `ItemTorch`

  Ein Item welches eine Fackel an der Zellenwand darstellt.

- `ItemSignalLamp`

  Ein Item welches eine Rundumleuchte an der Zellendecke darstellt.

- `...`

4.6.1 Aussehen und Verhalten

Auch Items unterstützen genau wie die Zellen die Trennung von Aussehen und Verhalten, siehe 4.5.1 Aussehen und Verhalten. Ein Item spezifiziert beispielsweise über die Methode `isTakable()`, ob der Pacman dieses Item „mitnehmen“ kann, oder nicht.

Ein Pacman seinerseits kann Items „ausführen“. Sofern es sich um ein transportables Item (wie z.B. eine Waffe) handelt, kann er dies jederzeit tun, vorausgesetzt, der Pacman hat das Item zuvor irgendwo aufgesammelt. Falls das Item aber nicht transportabel ist (wie z.B. ein Lichtschalter oder ein Beamapparat) muss der Pacman das Item an Ort und Stelle ausführen. Die „Ausführung“ eines Items geschieht über die Methode `execute(Pacman oPacman)` des Items, wobei der ausführende Pacman seine eigene Instanz als Parameter übergibt. Das Item wird daraufhin in seiner spezifischen Weise auf den Pacman einwirken, ohne dass der Pacman zuvor Kenntnis davon besitzt, welche Konsequenzen die Ausführung des Items auf ihn haben wird.

4.6.2 Entwicklung eigener Items

Es können beliebig viele, neue Itemklassen durch Ableitung bestehender Itemklassen erzeugt werden, und auf diese Weise beliebige Erweiterungen bestehender und neuer Zellen vorgenommen werden. Die Zuweisung von Itemklassen zu konkreten Zelleninstanzen geschieht durch die Definition entsprechender Leveldateien, siehe 4.8 Leveldateien.

4.7 Nachrichten

...„Monster an Pacman“, „Alice an Bob“!

Genau wie jedes andere, komplexe Softwareprodukt besteht Pacman 3D aus zahlreichen Einzelkomponenten. Die Kommunikation zwischen verschiedenen Komponenten einer Software findet im einfachsten Fall mittels Methodenaufruf statt, im Falle von Pacman 3D wurde eine alternative Möglichkeit integriert: das Versenden von Nachrichten.

Der Grund für die Integration dieses alternativen Kommunikationsmodells liegt im Entwicklungsprozess begründet, und ist in Abschnitt 2 Der Entwicklungsproze näher erläutert. An dieser Stelle soll lediglich von Belang sein, dass die Kommunikation mittels Nachrichten einerseits eine Abstraktion von der Netzwerkschicht bringt, und andererseits eine losere Kommunikation unter den verschiedenen Entwicklern voraussetzt, als es bei Methodenaufrufen der Fall ist.

Dreh- und Angelpunkt für das Nachrichtensystem ist das Package `pacman3d.message`, genaugenommen die Klasse `pacman3d.message.MessageService`. Diese Klasse stellt alle Funktionalität zur Verfügung, die in Zusammenhang mit dem Versenden und Empfangen von Nachrichten benötigt wird. Eine Instanz dieser Klasse erstellt der Aufrufer nicht etwa selber, sondern fordert sie gemäß dem Factory-Konzept über einen Aufruf von `MessageService.getInstance()` an.

Die zu versendenden bzw. zu empfangenden Nachrichten ihrerseits sind Instanzen der Klasse `pacman3d.message.Message`, oder eines Nachfahren davon.

4.7.1 Absender und Empfänger

Damit Nachrichten an ihr Ziel gelangen können, verfügen sie über ein Empfängerfeld, und für den Fall, dass der Empfänger antworten möchte, auch über ein Absenderfeld. Jede Nachricht besitzt exakt einen Absender, und mindestens einen, aber beliebig viele Empfänger.

„Absender“ und „Empfänger“ einer Nachricht sind nicht etwa textuelle oder numerische Bezeichner, sondern Instanzen der Klasse `pacman3d.message.ID`. Die exakte Repräsentationsform der ID ist nicht veröffentlicht; hier soll lediglich von Belang sein, dass jede erzeugte ID in Raum und Zeit eindeutig ist. Anders ausgedrückt: zwei erzeugte IDs sind immer ungleich.

4.7.2 Subjekte und Objekte

Wie können nun z.B. Pacman-Instanzen auf verschiedenen Rechnern miteinander kommunizieren? Um diese Frage zu beantworten, muss etwas ausgeholt werden:

Bei Pacman 3D im Mehrspielermodus handelt es sich um eine verteilte Anwendung mit einer Echtzeit-Synchronisation der beteiligten Rechensysteme untereinander. In einem Szenario mit z.B. 4 beteiligten Rechensystemen existieren auf jedem Rechensystem jeweils vier Pacman, d.h. vier Instanzen von `pacman3d.pacman.Pacman`. Insgesamt existieren in diesem Szenario über alle Rechensysteme also 16 Pacman-Instanzen (!), von denen aber nur vier Stück direkt durch einen Anwender gesteuert werden. Die restlichen zwölf Pacman-Instanzen fungieren als „Slave-Pacman“, und vollziehen lediglich die Aktionen der vier „Master-Pacman“ nach. Mit den Monstern ist es entsprechend.

Zusammenfassend ausgedrückt, gibt es in diesem System solche Instanzen, die eine „Master-Slave-Beziehung“ zueinander haben, und solche Instanzen, die nicht „zusammengehören“. Für diese Unterscheidung tritt der Begriff des „Subjektes“ auf den Plan:

In der gebräuchlichen Terminologie verwendet man den Begriff des Subjektes z.B. für Benutzer, die im Gegensatz zu den Objekten eines Rechensystems eigenständig handeln, und nicht den Regeln des Computersystems unterworfen sind. In der Pacman 3D-Terminologie wird der Begriff des Subjektes erweitert: Subjekte sind alle Teilnehmer des Pacman 3D-Spieles, die eigenständig handeln können.

Daraus folgt, dass ein und derselbe Pacman auf zwei Rechnern, der ja zwei verschiedenen Java-Instanzen, aber nur einer einzigen „handelnden“ Java-Instanz bzw. einem einzigen Benutzer entspricht, nur ein einziges Subjekt ist. In dem obigen Szenario lägen also trotz der 16 Pacman-Instanzen nur vier Pacman-Subjekte vor.

Die Beantwortung der obigen Frage ist, dass IDs nicht eindeutig für eine Instanz, sondern eindeutig für ein Subjekt vergeben werden. Wenn also eine Pacman-Instanz alle anderen Instanzen desselben Subjektes (d.h. auf anderen Rechnern) aktualisieren möchte, muss sie dazu lediglich eine Nachricht an sich selber schicken.

4.7.3 Versenden von Nachrichten

Zum Versenden einer Nachricht erzeugt der Versender eine Instanz von `pacman3d.message.Message`, oder eines Nachfahren davon, und weist ihr einen Absender, beliebig viele Empfänger, und eine Nutzlast zu.

Der Absender einer Nachricht muss dabei die ID der sendenden Instanz sein. Bei den Empfänger kann es sich entweder um IDs von Instanzen sein, oder für den Fall, dass die konkreten IDs nicht bekannt sind, können auch „virtuelle Empfänger“ angegeben werden. „Virtuelle Empfänger“ sind logische IDs wie z.B. „alle Monster“, „alle Pacman“ oder auch „alle Nachrichtenempfänger“.

Bei der Nutzlast einer Nachricht kann es sich um eine beliebige Instanz einer beliebigen Klasse handeln.

4.7.4 Empfangen von Nachrichten

Jede Klasse, die Nachrichten empfangen möchte, implementiert die Interfaces `pacman3d.util.Identifiable` und `pacman3d.message.MessageListener`, und ruft auf der Instanz des Nachrichtensystems (`pacman3d.message.MessageService`) die Methode `addMessageListener()` auf.

Das Interface `pacman3d.util.Identifiable` schreibt Klassen eine Methode zur Identifizierung vor, welche die ID für die jeweilige Instanz zurückgibt.

Das Interface `pacman3d.message.MessageListener` schreibt Klassen eine Methode zum Nachrichtenempfang vor, welche eine eingetroffene Nachricht entgegennimmt.

4.8 Leveldateien

...wo kommen all die Levels her?!

Niemand möchte immer wieder und wieder das selbe Spiel spielen: statt dessen muss Abwechslung her! Und so ist es eine langgepflegte Tradition im Bereich der Computerspiele, mehrere Level anzubieten. Auch Pacman 3D will hier mithalten, und bietet gegenwärtig rund 20 verschiedene Levels an!

Um auch Personen ohne Java-Kenntnissen die Erstellung von neuen Spielfeldern zu ermöglichen, und auch, um den internen Pacman 3D-Entwicklungsprozess von der Levelerstellung abzukoppeln, sind die Level für Pacman 3D *nicht* im Programmcode einprogrammiert! Statt dessen werden die Pacman 3D-Level in separaten Leveldateien abgespeichert.

Leveldateien für Pacman 3D sind XML-Dateien mit der Dateiendung „`.p3d`“, z.B. „`keep_alive_as_long_you_can_10x2x10.p3d`“. Damit Pacman 3D Leveldateien auffinden kann, müssen sich die Leveldateien im Verzeichnis „`levels`“ befinden, welches sich in gleicher Tiefe wie „`pacman3d`“ befinden muss.

Die grammatikalische Beschreibung des XML-Formates würde den Rahmen dieser Dokumentation sprengen, und ist dank des Leveleditors auch nicht notwendig, um eigene Leveldateien zu erstellen (siehe 3.2 Der Leveleditor). Wer dennoch per Hand Leveldateien erstellen möchte, dem dürfte eventuell ein Blick in die bereits bestehenden Leveldateien ausreichende Informationen verschaffen.

Tipp: Um „mal eben schnell in eine Leveldatei reinzuschauen“, ohne Pacman 3D starten oder gar erst installieren zu müssen, eignet sich eine XSLT-Konvertierung vorzüglich! Zur Konvertierung einer Leveldatei einfach auf „Levelvisualisierung“ klicken, und den Inhalt der Leveldatei per Copy & Paste in das Formularfeld einfügen: fertig!!

Das Ergebnis einer solchen Konvertierung sieht z.B. wie folgt aus:

Abbildung 50 - Ergebnis einer Levelfile XSLT-Konvertierung

5 Designentscheidungen

...warum hat sich die Labyrinthgruppe so, und nicht anders entschieden?

Bei der Entwicklung eines so umfangreichen Softwareprojektes wie Pacman 3D werden zahlreiche Designentscheidungen getroffen. Viele davon sind offensichtlich trivialer Natur, d.h. beruhen auf allgemeinen Programmierparadigmen, oder weit verbreiteten Erfahrungen. Andere sind spezifisch für dieses Projekt, bzw. diesen Entwicklungsprozess, und verdienen daher Beachtung, da sie sich in der endgültigen Anwendung niederschlagen.

5.1 Designentscheidung: Nachrichten

Genau wie jedes andere, komplexe Softwareprodukt besteht Pacman 3D aus zahlreichen Einzelkomponenten. Die Kommunikation zwischen den verschiedenen Komponenten einer Software findet im einfachsten Fall mittels Methodenaufruf statt, im Falle von Pacman 3D wurde eine alternative Möglichkeit integriert: das Versenden von Nachrichten.

Die verschiedenen Softwarekomponenten von Pacman 3D sind auf vielfältige Art und Weise miteinander verwoben. Wegen der losen Kommunikation der entwickelnden Gruppen untereinander, wurde es immer schwieriger, Veränderungen an den zentralen Schnittstellen der Komponenten vorzunehmen. Ändert sich z.B. eine Schnittstelle des Pacman, müssten sowohl die Monster- als auch die Labyrinthkomponenten daran angepasst werden, wenn sie sich direkt dieser Schnittstelle bedienen würden.

Die Kommunikation der Softwarekomponenten mittels Nachrichten hingegen erlaubt eines sehr lose Kopplung der Komponenten. Veränderungen im Interface einer Komponente schlagen sich dadurch nicht sofort in funktionsuntüchtiger Software nieder, oder gar in Kompilierfehlern.

Obwohl damit die Verwendung von Nachrichten schon hinreichend begründet wäre, gab es noch ein zweites Argument: die transparente Integration der Netzwerkschicht:

Es war die Aufgabe der Labyrinthgruppe, die Netzwerkschicht bei Fertigstellung durch die Netzwerkgruppe transparent in das System einzubauen, so dass für die anderen Gruppen allenfalls minimaler Nachbesserungsaufwand entstehen würde. Dieser Anforderung wurde das Nachrichtenkonzept gerecht, indem es die Zustellung der Nachrichten (insbesondere die Unterscheidung lokal/nicht-lokal) vor den Verwendern und Empfängern der Nachrichten abstrahiert.

5.2 Designentscheidung: Leveldateien

Die ursprüngliche Vorgabe an die Labyrinthgruppe war es, Leveldateien in Javaklassen zu speichern, d.h. pro Level eine Javaklasse bzw. eine Ableitung vorzusehen.

Die Labyrinthgruppe hat sich aus mehreren Gründen gegen diese Vorgehensweise, und für die Verwendung von XML-Leveldateien entschieden:

1. Die Verwendung einer deklarativen Sprache (XML), anstelle einer imperativen Sprache (Java) vereinfacht die Erstellung der Leveldateien sowohl für programmiertechnische Laien als auch für Editorenwerkzeuge.

   (Für Editoren dürfte es schwer werden, komplexen und fehlerfreien Java-Code zu erzeugen, und für programmiertechnische Laien unmöglich.)

2. Durch die Auslagerung der Levelinformationen in separate Dateien wird der Prozess der Anwendungsentwicklung von dem der Levelerstellung abgekoppelt.

   Dadurch enthalten auch nicht direkt am Labyrinth arbeitende Personen die Möglichkeit, Leveldateien zu erstellen.

3. Die Verwendung einer XML-Sprache bringt alle XML-Vorzüge mit sich, wie z.B. die Verwendbarkeit allgemein verfügbarer XML-Werkzeuge (z.B. InternetExplorer, XML Spy, ...), Validierung gegen DTD oder XML Schema,

Konvertierung mittels XSLT (siehe z.B.
http://www.stormzone.de/uni/pacman3d/services/levelvisualize.html).

IX - WEBSITE

Unter www.stormzone.de/uni/pacman3d/ sind neben dem vorliegenden Dokument, weiteren Dokumenten und Präsentationen zu Pacman 3D in den verschiedensten Formaten und Screenshots von Pacman 3D auch eine JavaDoc-Dokumentation, und eine Liste der Autoren verfügbar:

Abbildung 51 - Website von Pacman 3D

X - ABBILDUNGSVERZEICHNIS

XI - INDEX

Z	Zelle 18, 21, 26, 37, 38, 48, 52, 53, 57, 60, 61, 62, 64, 65, 66, 67, 69, 70, 71, 72, 73