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 der Labyrinthgruppe

1 Inhalt

2 Einleitung

...worum geht es in diesem Dokument?!

Dieses Dokument dokumentiert 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

3 Praktikumsbericht

3.1 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 (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 Start der eigentliche Pacman-Entwicklung, 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 des Pacman-Spieles! Nachdem diese Anwendung Anfang März als Ganzes noch einmal abgenommen wurde, wurde sie Mitte März vor einem Teil der Angestellten des 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-Spielwettbewerb über, in dem sich Frank Bergmann (Pacmangruppe) gegen seine Kontrahenten durchsetzen konnte.

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

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]:

- Daniel F. Abawi,
- Dipl. Informatiker Christian Seiler,
- Dipl. Informatiker 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 dem Pacman-Spiel zugute gekommen sind.

3.2 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.

- 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.

[3] in alphabetischer Reihenfolge

3.3 Die Labyrinthgruppe

...von Labyrinthen, Leveldateien und Nachrichten!

Jede der fünf Praktikumsgruppen (siehe 3.2 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 5.7 Nachrichten).
- Die Labyrinthgruppe übernahm die Entwicklung der Startdatei der Anwendung (siehe 5.3 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
- Gordon Weckbach
- Lijun Zhou
- Martin Klossek
- Paul Rojaz

3.4 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.

3.4.1 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 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.

3.4.2 November 2001: Bistros und Lokalitäten

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

Für das Austüfteln technischer Details war das 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 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:

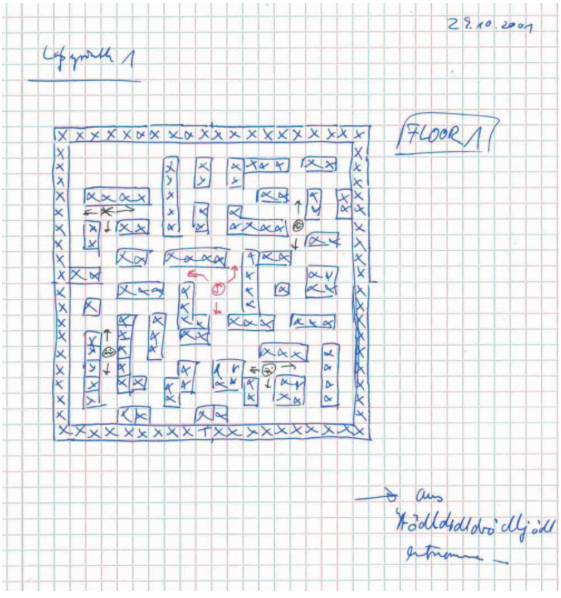

Abbildung 1 - Erster Grobentwurf des Labyrinthes

3.4.3 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 Informationen 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:

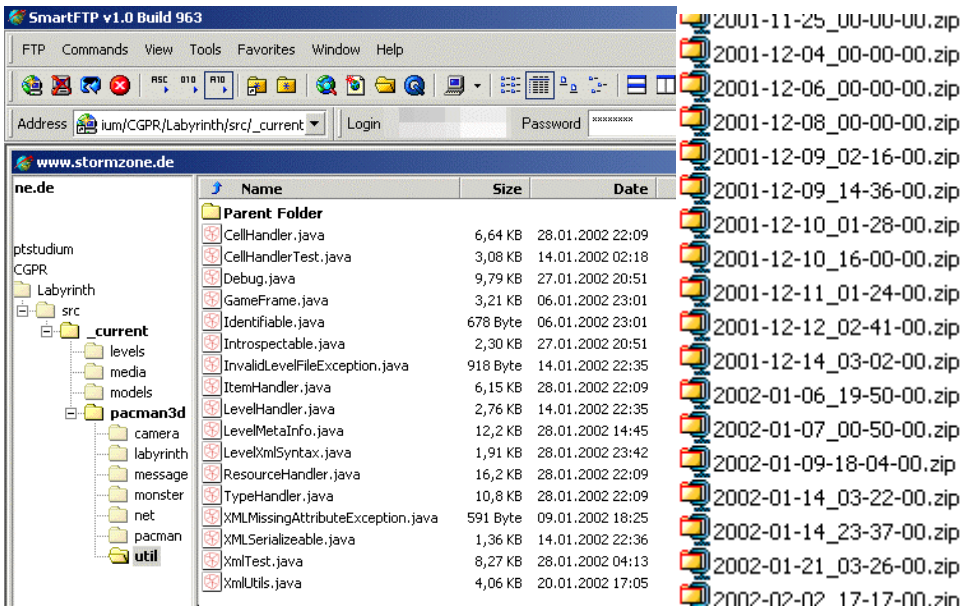

Abbildung 2 - FTP und das Dateichaos

3.4.4 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:

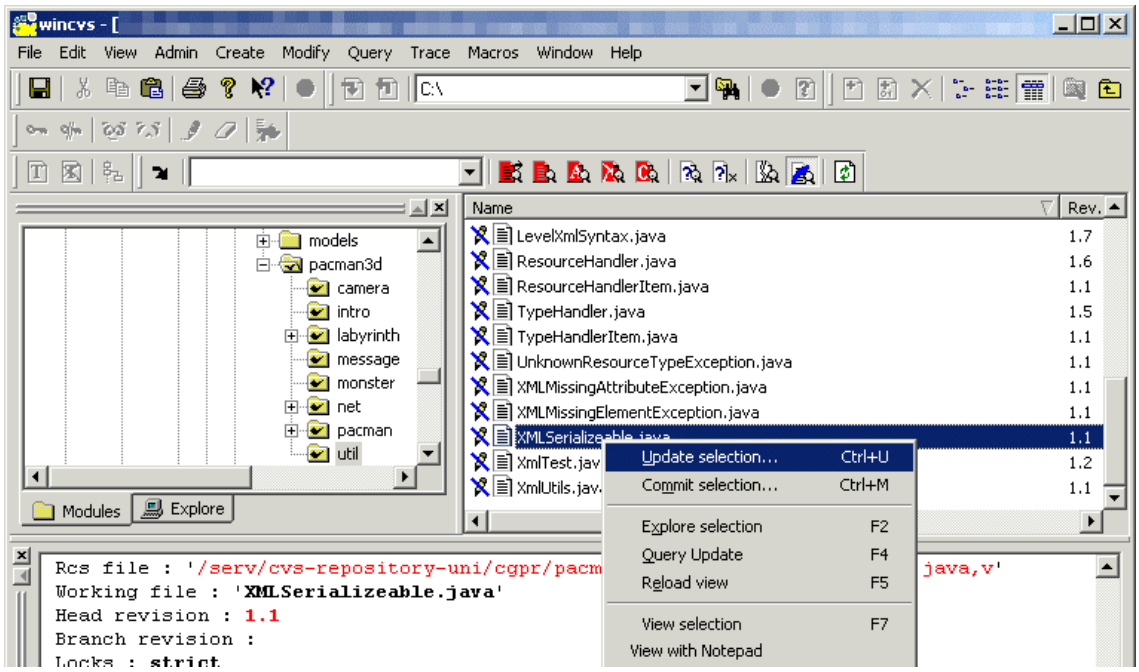

Abbildung 3 - Quellcodeverwaltung mit CVS!

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.

3.5 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.

3.5.1 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.

3.5.2 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).

4 Anwenderdokumentation

4.1 Programmstart

4.1.1 Kommandozeilenparameter

Pacman 3D unterstützt beim Programmstart die folgenden Kommandozeilenparameter:

Parametername	Optionen	Aufgabe
`--help`		zeigt eine Liste der möglichen Kommandozeilen-parameter, ihrer Optionen und Funktionsweise
`--withoutintro`		startet Pacman3D 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 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

Parametername	Optionen	Aufgabe
		(= 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

4.1.2 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 ein Singleplayerspiel ohne Intro mit dem angegebenen Level im `selbstablaufenden Demomodus:`

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

4.2 Der Leveleditor

...der Ursprung aller Leveldateien!

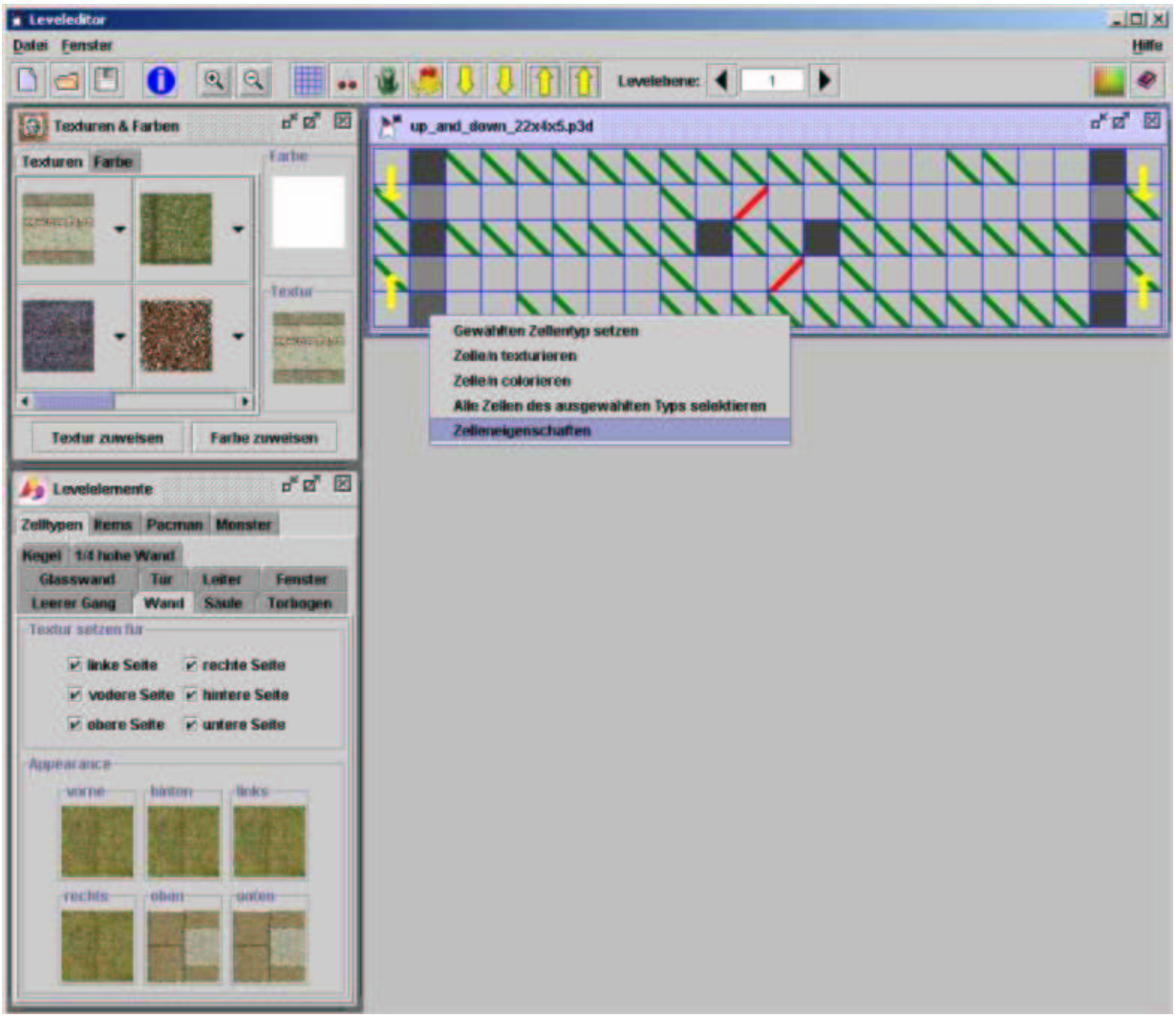

Abbildung 4 - Der Leveleditor in Aktion!

4.2.1 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 5.8 Leveldateien), und dass offensichtlich ein Labyrinth visuell schneller und intuitiver erstellt werden kann, wenn dies mittels eines WYSIWYG[4]-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

[4] what you see is what you get

wäre. Somit ist der Leveleditor in seiner jetzigen Form - als 2D-Editierungswerkzeug - entstanden.

Im Grunde genommen stellt der Leveleditor 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 5.5 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 5.6 Items), Monster und Pacmanstartpositionen hinzugefügt werden können.

4.2.2 Die Klassen des Leveleditors

Insgesamt haben alle Klassen des Leveleditors, nämlich:

- `pacman3d.labyrinth.leveleditor.AppearanceJPanel`,
- `pacman3d.labyrinth.leveleditor.CellsJPanel`,
- `pacman3d.labyrinth.leveleditor.CellVector`,
- `pacman3d.labyrinth.leveleditor.ColorChooser2DComponents`,
- `pacman3d.labyrinth.leveleditor.ExampleFileFilter`,
- `pacman3d.labyrinth.leveleditor.ItemJPanel`,
- `pacman3d.labyrinth.leveleditor.JToolBarLeveleditor`,
- `pacman3d.labyrinth.leveleditor.LevelEditingJPanel`,
- `pacman3d.labyrinth.leveleditor.Leveleditor`,
- `pacman3d.labyrinth.leveleditor.LeveledtiorHelpJPanel`,
- `pacman3d.labyrinth.leveleditor.LevelMetaInfoJPanel`,
- `pacman3d.labyrinth.leveleditor.LevelSizeDialog`,
- `pacman3d.labyrinth.leveleditor.MonsterJPanel`,
- `pacman3d.labyrinth.leveleditor.MyFileFilter`,
- `pacman3d.labyrinth.leveleditor.PacmanJPanel`,
- `pacman3d.labyrinth.leveleditor.PreviewJLabel` und
- `pacman3d.labyrinth.leveleditor.TextureSelectionJPanel`

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

- `pacman3d.labyrinth.leveleditor.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.

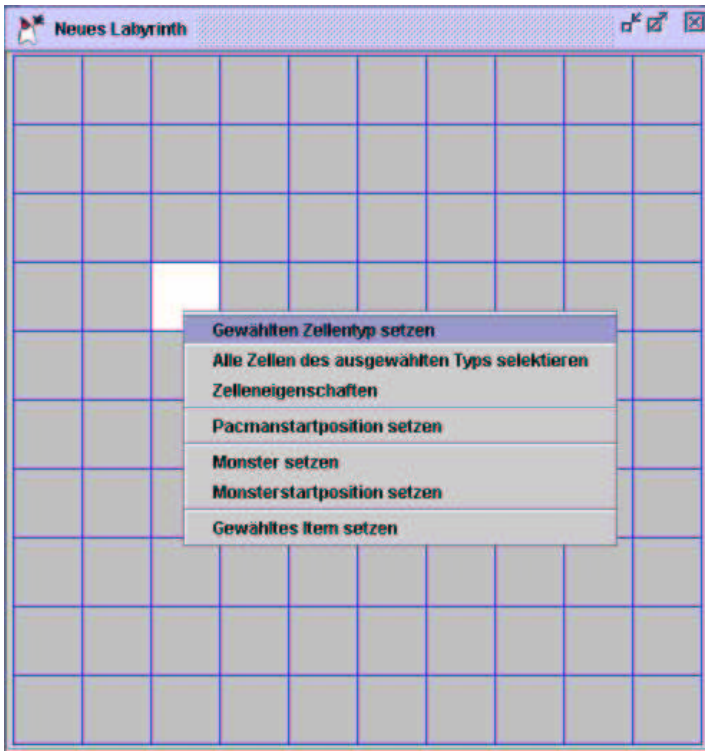

Abbildung 5 - Festlegen einer Zelle

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:

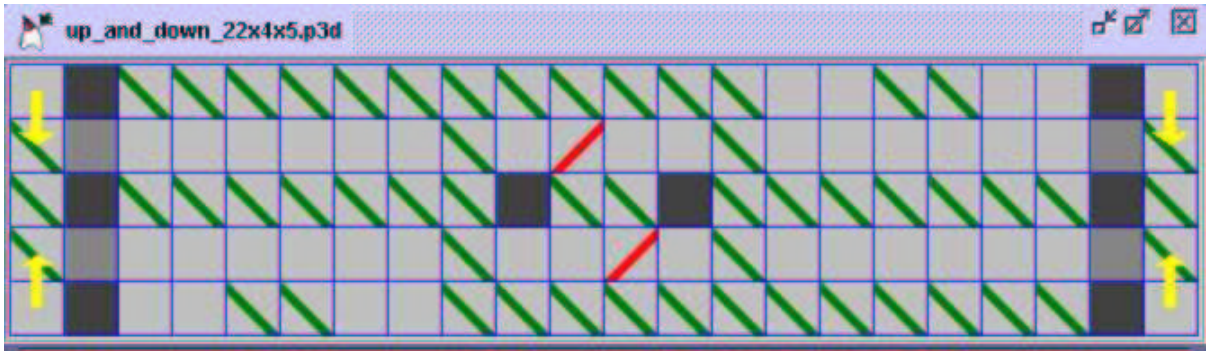

Abbildung 6 - Editierte Labyrinthebene

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.levelleditor.Levelleditor`

In einem anderen Sinn stellt die Klasse `Levelleditor` 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.

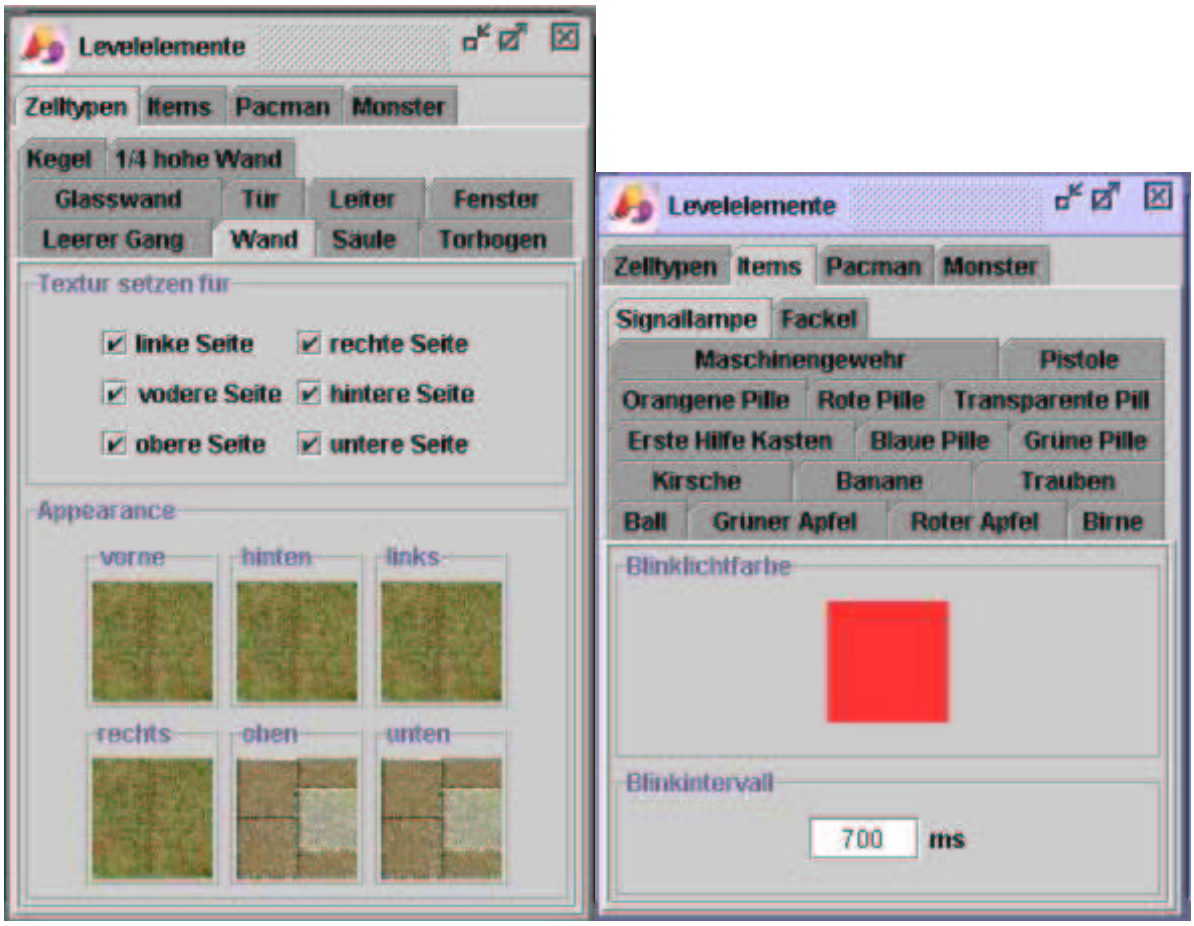

Abbildung 7 - Zell- und Itemeigenschaften

- `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.

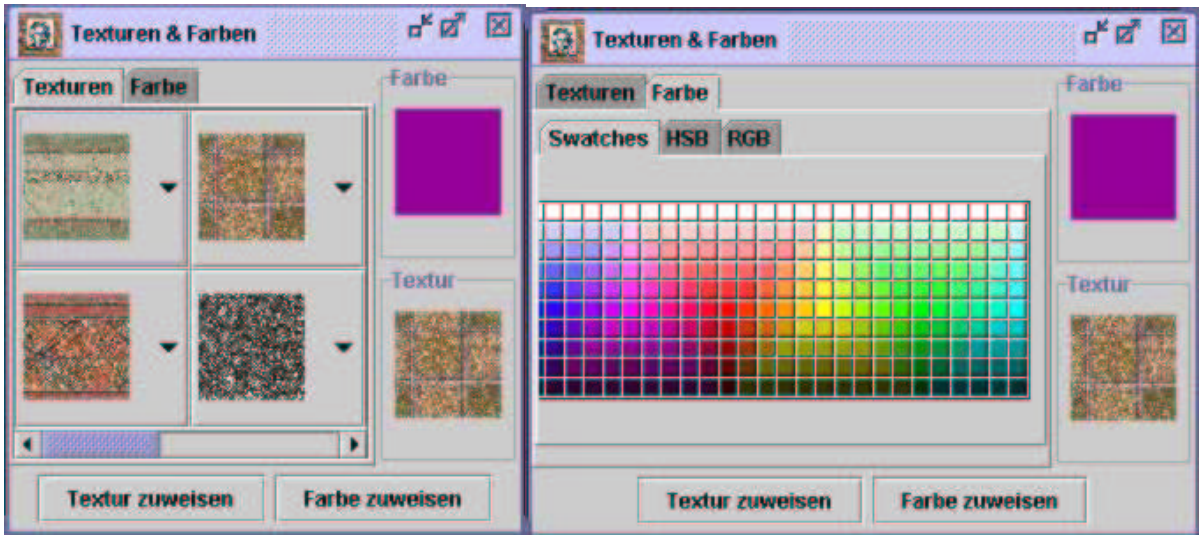

Abbildung 8 - Texturen & Farben

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.leveleditor.JToolBarLeveleditor`:

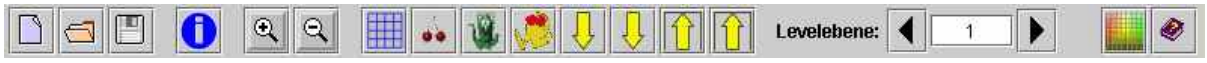

Abbildung 9 - Werkzeugleiste des Leveleditors

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

Abbildung 10 - Hilfefenster des Leveleditors

4.2.3 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.

4.2.4 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.

4.2.5 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.

4.2.6 Mit dem Leveleditor erzeugte Labyrinthe

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

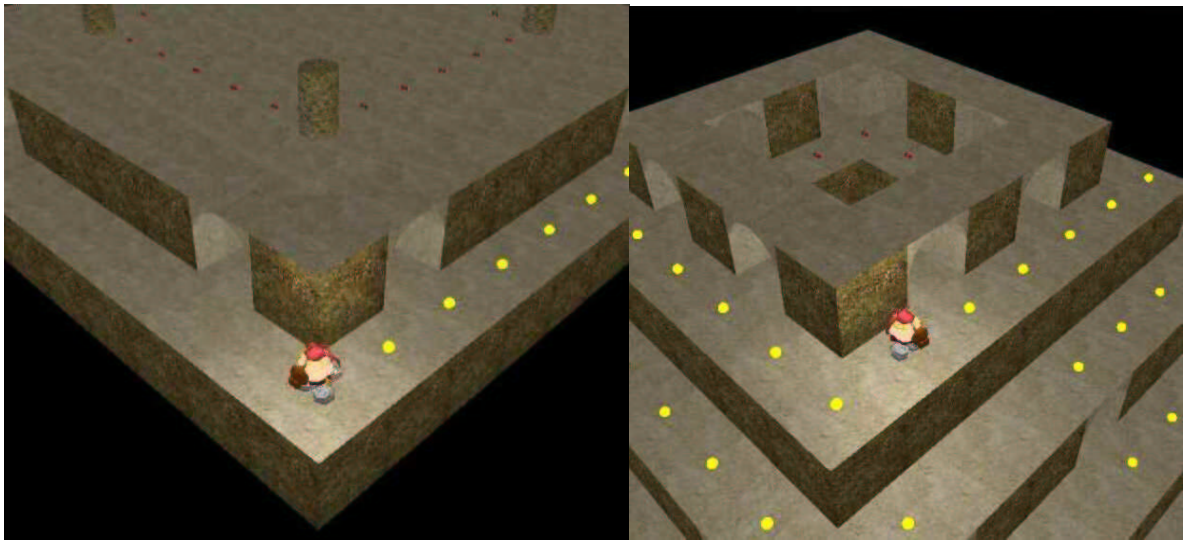

Abbildung 11 - Level "Climb up the pyramid"

- „Up and down“ (22x4x5 Zellen):

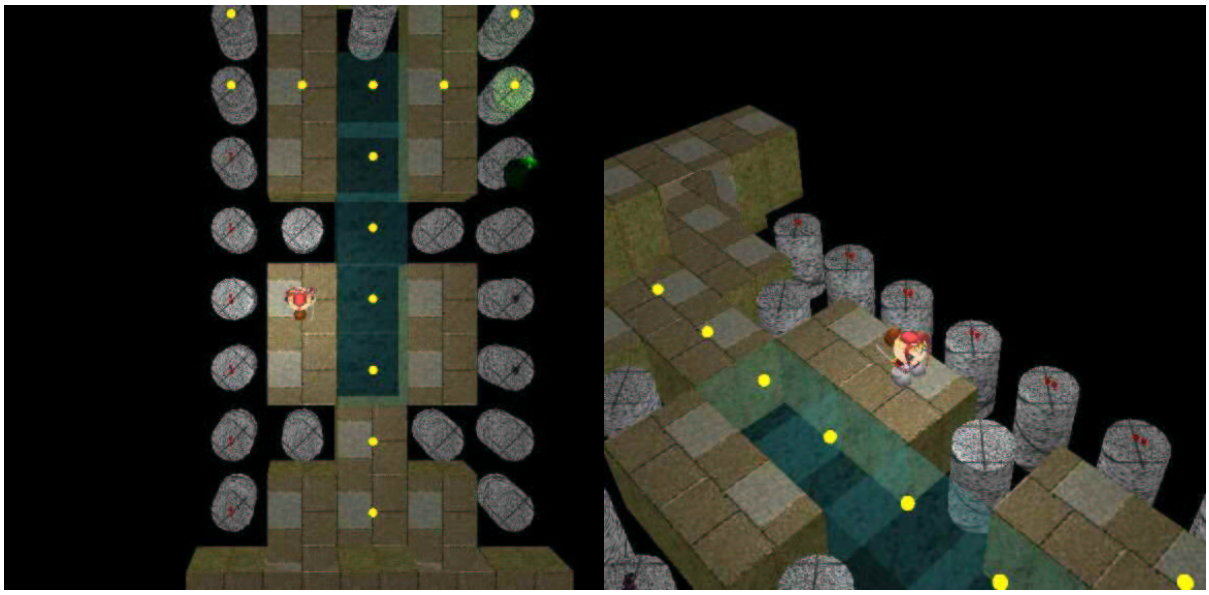

Abbildung 12 - Level "Up and down"

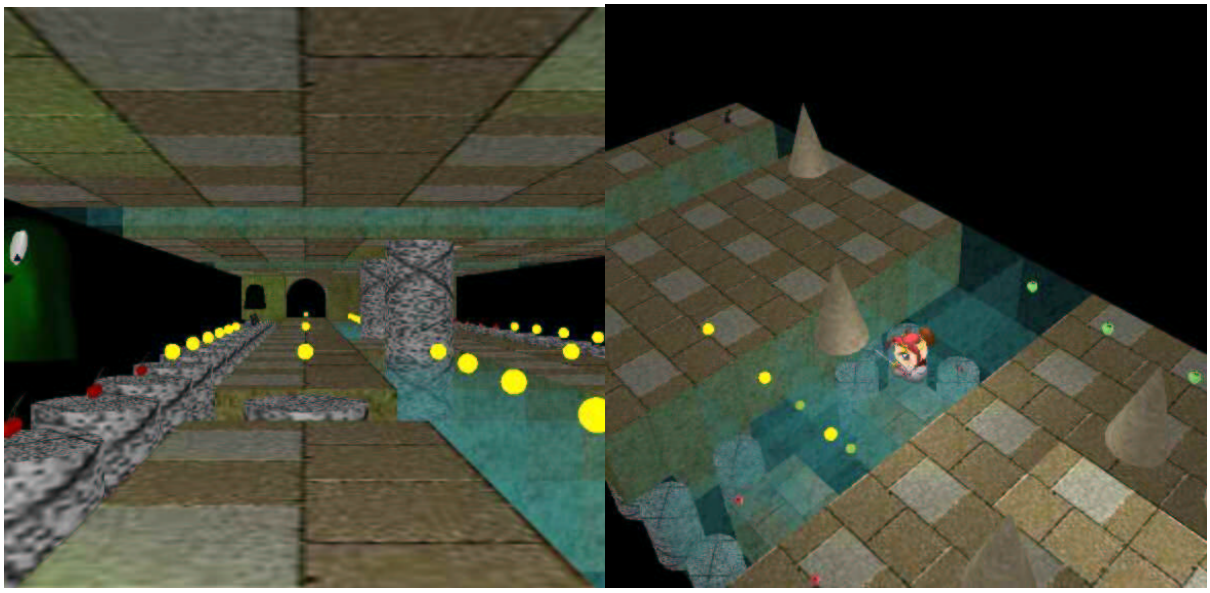

Abbildung 13 - Level "Up and down"

- „Nearly traditional“ (15x3x15 Zellen):

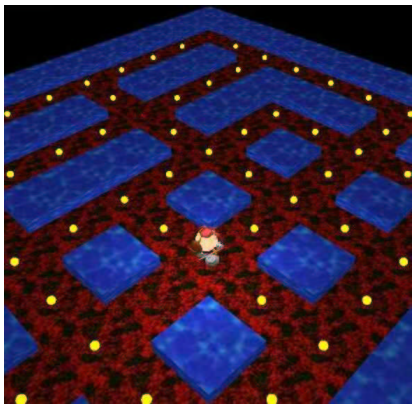

Abbildung 14 - Level "Nearly traditional"

5 Technische Dokumentation

5.1 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 auch 5.3 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 5.7 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 5.5 Zellen, 5.6 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.

5.2 Systemstruktur

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

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

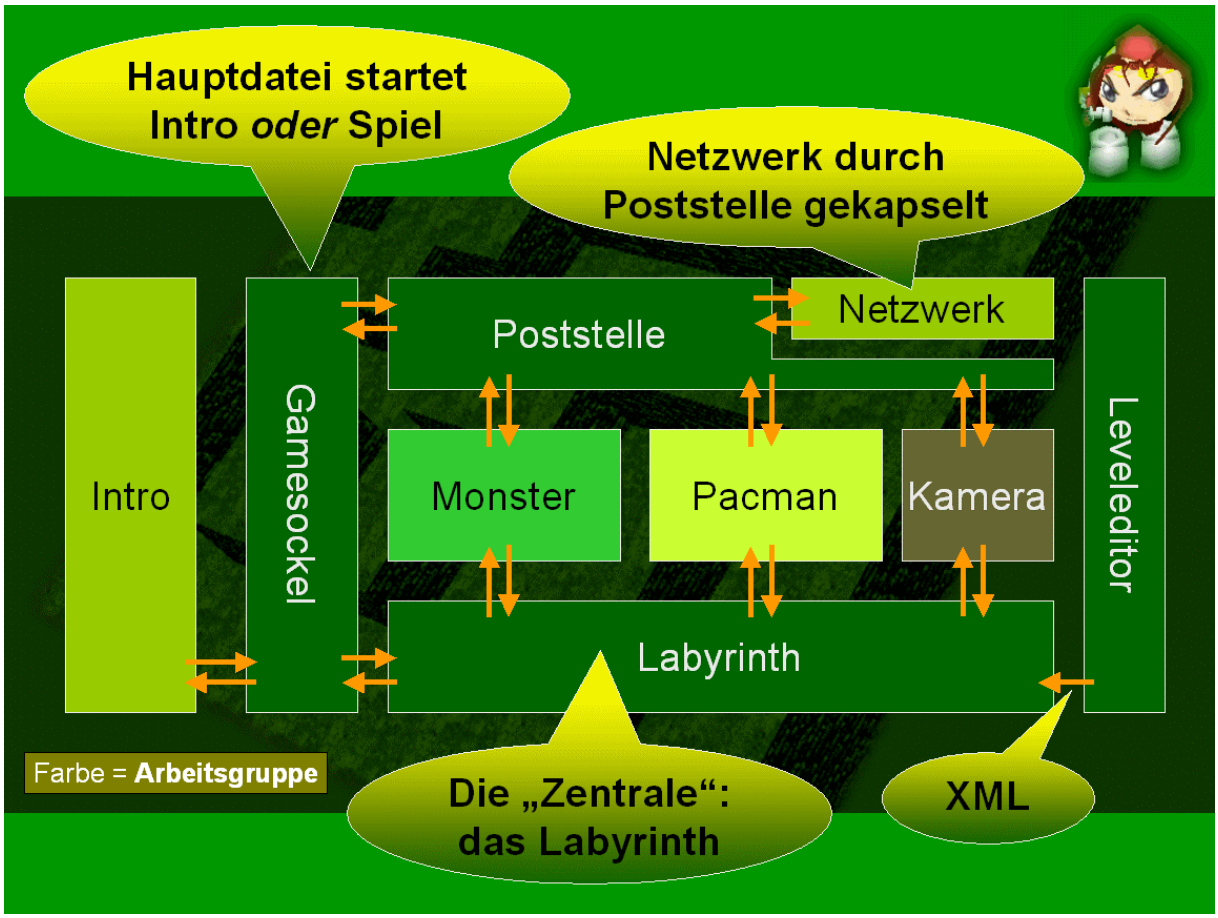

Abbildung 15 – Systemstruktur

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 4.2 Der Leveleditor, 5.4 Das Labyrinth, 5.8 Leveldateien).

5.3 Die Startdatei

...die Mutter aller Klassen!

Pacman 3D wird mittels der Java-Klasse `pacman3d.Game` gestartet (siehe 4.1 Programmstart). Das Verhalten der Anwendung kann durch zahlreiche Kommandozeilenparameter beeinflusst werden, siehe 4.1.1 Kommandozeilenparameter.

5.4 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`.

5.5 Zellen

...quadratisch, praktisch, gut!

Wie in 5.4 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 Pacman-Welten 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.
- `...`

5.5.1 Aussehen und Verhalten

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

4. 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 5.8 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“ bezeichnet.) 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.

5. 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.

5.5.2 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 5.8 Leveldateien.

5.6 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:

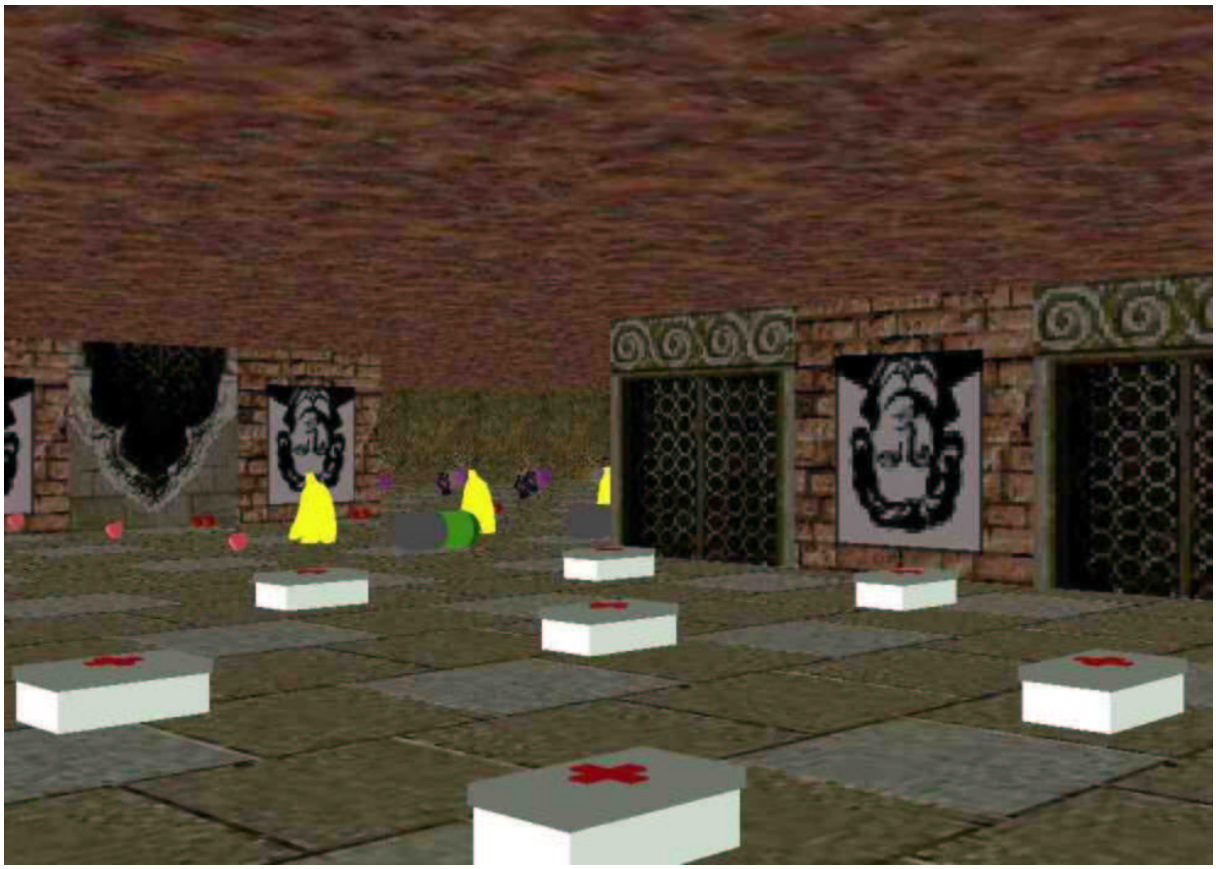

Abbildung 16 - Eine Szene voller Items

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.
- `...`

5.6.1 Aussehen und Verhalten

Auch Items unterstützen genau wie die Zellen die Trennung von Aussehen und Verhalten, siehe 5.5.1 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.

5.6.2 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 5.8 Leveldateien.

5.7 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 3.4 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.

5.7.1 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.

5.7.2 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 In-

stanzen desselben Subjektes (d.h. auf anderen Rechnern) aktualisieren möchte, muss sie dazu lediglich eine Nachricht an sich selber schicken.

5.7.3 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.

5.7.4 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.

5.8 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 4.2 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 17 - Ergebnis einer Levelfile XSLT-Konvertierung

6 Anhang

...was da sonst noch war!

6.1 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 18 - Website von Pacman 3D

7 Abbildungsverzeichnis

8 Index