D I P L O M A R B E I T

Mobile und Wireless Entertainment

Eine Implementierung am Beispiel von Java 2 ME und MIDP

FH Furtwangen

Fachbereich Digitale Medien

Vorgelegt von:
Boris Schmidt
Studiengang Medieninformatik
Kirchhausener Str.4
74078 Heilbronn

Betreuer:
Prof. Dipl.-Inf. Wilhelm Walter
Prof. Dr. Ullrich Dittler
Dipl.-Inf. Peter Rudolph, Wearix GmbH

Heilbronn, den 28.02.2002

Inhaltsverzeichnis

Abbildungen

Listings

1 Einleitung

Lange Zeit war Mobile-Computing als ein wichtiger Zukunftsmarkt angesehen worden. Doch der „UMTS-Lizenz-Kosten-Schock“ und der ökonomische Abwärtstrend 2001 ernüchterten schnell die Martkbeobachter und -teilnehmer. WAP-Dienste floppten in Europa und die Pleitenserie vieler Internetfirmen tut ihr Übriges, um die Stimmung im mobilen IT-Bereich auf ein Tiefstniveau zu senken. Mancherorts wird der japanische Mobilservice i-mode[1] als Vorbild für ein erfolgreiches Geschäftsmodell mit seinen zahlreichen Unterhaltungsangeboten angesehen. Andere verweisen diesbezüglich wiederum auf die kulturellen Unterschiede zwischen Japan und Europa und sehen hierin zumindest einen Teil des Erfolges begründet[2].

Viele sehen auch das Spielen per Handy allgemein, als einen wichtigen Schlüssel für einen Markterfolg von Mobilfunktechnologien der nächsten Generation, wie GPRS und UMTS an. Doch das Angebot mobiler und drahtloser Spiele ist noch mager. Vergleicht man die Anzahl der Spiele für Handies, sei es nun für WAP oder SMS, mit der Menge der verfügbaren Titel für Nintendos GameBoy, wird die Euphorie über das mobile drahtlose Spielen per Mobiltelefon schnell gedämpft. Doch nicht nur die Menge, auch die Qualität der auf Handies erhältlichen Spiele, fällt weit hinter die von tragbaren Spielkonsolen gebotenen Qualität zurück.

Als eine Technologie, die den Spielen auf Handies zum Durchbruch verhelfen soll, wird die Java 2 Micro Edition gehandelt. Doch wie sehen die Möglichkeiten dieser Technologie im Vergleich zu anderen Technologien und Plattformen aus? Kann sie den Ansprüchen gerecht werden, die an „Mobile und Wireless Entertainment“ gestellt werden?

1.1 Inhaltliche Übersicht

Um diese Frage beantworten zu können, sollen in den folgenden Kapiteln dieser Diplomarbeit zunächst die Entwicklungsgeschichte der mobilen Unterhaltung

1 www.nttdocomo.com

2 vgl. [Puha01]

vorgestellt und gängige Spielgenre auf ihre Realisationsmöglichkeiten auf mobilen Endgeräten untersucht werden. Des Weiteren werden spieltheoretische Grundlagen vorgestellt und ihre Inhalte auf Relevanz für mobile und drahtlose Spiele überprüft. Als Basis dieser Betrachtungen dienen Überlegungen und Untersuchungen zu den Besonderheiten des mobilen und drahtlosen Spielens, sowie den technischen Rahmenbedingungen, die nicht nur maßgebend für die Umsetzungsmöglichkeiten bestehender Computerspiele sind, sondern vor allem für Spielideen und -designs an sich.

Auf der Grundlage dieser Betrachtungen und Untersuchungen wurde im „praktischen“ Teil dieser Diplomarbeit eine Basis-Software für die Spieleentwicklung unter Java 2 ME und MIDP entworfen und implementiert. Neben der Realisierung der für Spiele relevanten Spezifikationen behandelt dieser Teil auch, inwiefern objektorientierte Entwicklungs-Paradigmen auf Handies und mobilen Endgeräten unter den harten Performanceanforderungen anwendbar sind.

Die Diplomarbeit untersucht in Kapitel 7 mit der Erstellung eines Demospiels die in den Kapiteln 5 und 6 spezifizierte und implementierte Spiele-*API* und liefert damit Aufschluss darüber, inwieweit die Umsetzung eines Spieletitels auf mobilen Endgeräten mittels eines Standard-Sets von Software-Funktionen möglich ist.

Die Diplomarbeit schließt mit einer zusammenfassenden Betrachtung der erzielten Ergebnisse, sowie mit einem Ausblick auf zukünftige Entwicklungen und Möglichkeiten der mobilen und drahtlosen Spiele.

1.2 Begriffsdefinition

Mobile und Wireless Entertainment bedeutet im Zusammenhang dieser Diplomarbeit vor allem interaktive Unterhaltung auf mobilen und drahtlosen Endgeräten, sprich Computerspiele. „Mobile“ bzw. „mobil“ bedeutet hierbei, dass die Unterhaltung von einem beliebigen Ort stattfinden, „Wireless“ oder „drahtlos“, dass eine drahtlose Netzwerkverbindung zwischen verschiedenen Geräten bestehen kann.

In diesem Zusammenhang sind aber nicht nur die Software, das Spiel an sich, sondern auch die darunterliegende Hardware wichtig. Um der Vielfalt der verschiedenen drahtlosen Endgeräte wie Handies, *Smartphones*, Personal Digital Assistants (*PDAs*) und *Organizern* gerecht zu werden, wird hier der Begriff „mobile Clients“ stellvertretend für diese Geräte einführt. Dabei charakterisiert „mobile“ natürlich die Mobilität der Engeräte, „Clients“ hingegen implizit die drahtlose Funkanbindung an ein Netzwerk in beliebiger Form, evtl. auch an einen *Server* – wobei diese Serververbindung eher optional anzusehen ist, also keine absolute Voraussetzung für den Betrieb des Clients darstellt. Des Weiteren muss das nicht heißen, dass nur *Client-Server*-Verbindungen möglich sind, sondern durchaus eine *Peer-to-Peer*-Vernetzung direkt zwischen den Clients stattfinden kann.[3]

1.3 Ziele der Diplomarbeit

Bei dieser Diplomarbeit sollen nicht die wirtschaftlichen und markttechnischen Rahmenbedingungen untersucht werden. Diese können nur einen ungefähren Aufschluss darüber bieten, wie sich der Markt der mobilen und drahtlosen Unterhaltung entwickeln könnte, da sich der Markt dieser Form der Computerspiele erst noch im Aufbau befindet. Wie etwa WAP zeigte, ist sicherlich noch mit der einen oder anderen Überraschung zu rechnen – allerdings in positiver wie in negativer Hinsicht. Vielmehr sollen grundlegende Überlegungen und Untersuchungen stattfinden, die „Mobile und Wireless Entertainment“ in seinem Grundwesen und Charakter beschreiben und dabei ein besonderes Augenmerk darauf legen, was aus Sicht des Entwicklers wichtig ist. Das sind neben gestalterischen Grundüberlegungen vor allem technische Untersuchungen, wie sie im Zuge der Entwicklung der Spiele-API und des Spieledemos stattfinden.

Dabei soll eine wiederholte Betrachtung von den in Informatiker-Kreisen bekannten Technologien wie Bluetooth, GPRS und UMTS vermieden werden. Diese und andere Technologien unterscheiden sich in den für die Spieleentwicklung relevanten Dingen nur in Kernpunkten. Daher soll nur vorgestellt werden,

3 Für eine Erklärung der Begriffe Java 2 ME und MIDP, sei auf das Kapitel *4.1*, speziell *4.1.1 Java 2 ME* verwiesen.

was z.B. mit den neuen Übertragungsverfahren im Gegensatz zu GSM-Netzen, wie sie bisher in Europa üblich waren, möglich ist.

Mit dieser Diplomarbeit soll dem Spieleentwickler ein Mindestmaß an Basiswissen geliefert werden, mit dem er selbst im Bereich der mobilen und drahtlosen Unterhaltung tätig werden kann. Darüberhinaus soll die Spiele-API einen Ansatz für Architektur und Funktionsumfang eines Frameworks zur Spieleentwicklung auf mobilen Clients bieten und damit einen ersten Schritt in der Entwicklung von Computerspielen auf diesen Geräten darstellen.

1.4 Notation

Da in dieser Diplomarbeit immer wieder auf Computerspiele referenziert wird, wird an dieser Stelle folgende Notation eingeführt. Spieletitel und dessen Hersteller werden durch einen Schrägstrich voneinander getrennt: „[Spieletitel]“/[Hersteller], Bsp.: „Pac Man“/Namco.

Für Quellenangaben wird folgende Notation verwendet: [Autor + Erscheinungsjahr], Bsp.: [Gamma00]. Die volle Quelleninformation findet sich im Anhang A. Sollte aus einer Quelle mehrfach von unterschiedlichen Seiten zitiert werden, so findet sich die Seite, von der das Zitat stammt, im Fliesstext.

Verweise auf andere Textbereiche werden *kursiv* gedruckt. Dabei verweisen nummerierte Verweise (Bsp.: *2 Die Historie der mobilen interaktiven Unterhaltung*) auf Kaptiel oder Abschnitte der Diplomarbeit. Für allein stehende, kursiv gedruckte Bezeichnungen (Bsp.: *Arcade*) findet sich in *B Glossar* eine nähere Erklärung des Begriffes. Methodennamen der Spiele-API werden ebenfals kursiv gekennzeichnet (Bsp: *`WLogic.gameInit()`*) und wie alle anderen Quelltext-Fragmente mit der Schriftart „`Courier New`“ gesetzt. Für eine detailierte Erklärung sei auf die beiliegende CD im Anhang D verwiesen. Hierauf ist eine Dokumentation des Quelltextes im JavaDoc-Format enhalten und mit jedem HTML-Browser einsehbar. Der Quellcode ist übrigens vollständig kommentiert und ebenfalls auf der CD enthalten.

2 Die Historie der mobilen interaktiven Unterhaltung

Da sich das Mobile und Wireless Entertainment erst noch in den Kinderschuhen befindet, soll an dieser Stelle die Geschichte der mobilen Computerspiele allgemein vorgestellt werden, um einen Eindruck über den Entwicklungsverlauf in diesem Segment der Interaktiven Unterhaltung gewinnen zu können. Dabei wird auf die Entwicklung der Spielinhalte und der Technik eingegangen, um hieraus Tendenzen und den inhaltlichen Charakter der Spiele bestimmen zu können.

2.1 Die erste Generation – LCD-Spiele

Mobile Computerspiele haben wie vielleicht wider erwarten eine lange Geschichte hinter sich. Ihren Anfang nahm die Entwicklung in der breiten Masse mit der „Game&Watch"-Serie von Nintendo, die in Deutschland etwa zu Beginn der 80er Jahre erhältlich war. Diese tragbaren, Handflächen-großen „*LCD*-Spiele" hatten jeweils ein Spiel integriert, zusätzliche waren nicht ladbar. Die Darstellung eines anderen Spieles wäre auf Grund des LC-Displays auch gar nicht möglich gewesen. Es wurden hier nämlich nicht Rastergrafiken erzeugt, sondern das Display schaltete ganze Grafiken ein und aus. So wurde der Eindruck einer recht ruckartigen Animation erzeugt. Diese erste Generation der mobilen Spiele kam sehr stromsparend mit Knopfzellen-Batterien aus und ermöglichte mehrere Tage (strom-)netzunabhängigen Spielspaß. Mit der Zeit adaptierte eine Vielzahl von Herstellern sowohl Hardware als auch Inhalte der Spiele. Bis heute findet man LCD-Spiele im Sortiment des Spielwareneinzelhandels.

2.1.1 Spielinhalte der LCD-Spiele

Die Spielinhalte der LCD-Spiele konzentrierten sich meist auf einfache Geschicklichkeits- und Reaktionstests. So muss man meist zwischen verschiedenen Positionen wechseln, um Gegnern auszuweichen oder um Spielobjekte einzusammeln. Einen ähnlichen Inhalt stellt das Zu-einem-Ziel-steuern dar, wobei auch wieder herannahenden Gegnern ausgewichen werden muss, während man sich ein paar Positionen weiter bewegt. Wie schon die Schilderungen der Spiel-

inhalte andeuten, waren und sind diese Spiele extrem einfach. Die Spielprinzipien variieren nur marginal über die verschiedenen Spielgeräte, und auch die audiovisuelle Ausprägung der einzelnen Spiele unterscheidet sich nur geringfügig auf Grund der, mit heutigen Möglichkeiten verglichen, primitiven Technik.

2.1.2 Technik der ersten Generation

Meist setzen LCD-Spiele auf eine einfache Audio-Ausgabe und erzeugen Piep-Töne unterschiedlicher Tonhöhe und Länge, ähnlich denen, die ein PC ohne Soundkarte macht. Ähnlich verhält es sich bei der Grafik. Wie schon erwähnt bieten die Geräte keine Rastergrafik, sondern schalten einzelne Spielerdarstellungen über den Bildschirm verteilt an und aus, wodurch eine sehr simple Animation erzeugt wird. Allerdings variiert die Darstellung eines Spielobjektes an einer Stelle i.d.R. nicht, eine „Animation“ findet also nur zwischen den Darstellungsorten statt.

2.2 Die zweite Generation mit Rastergrafik

Ende der 80er, Anfang der 90er erschien die zweite Generation auf der Bildfläche, oder besser gesagt auf den LCDs. Eine breite Pallette von Anbietern startete auf der ganzen Welt verschiedene Plattformen für mobile Computerspiele. Unter ihnen das Atari Lynx, Segas Gamegear, das TurboExpress von NEC und natürlich der GameBoy von Nintendo. Als einzige Plattform setzte Nintendo damals auf eine monochrome Darstellung ohne Hintergrundbeleuchtung des Displays. Dies sorgte einerseits für günstigere Geräte[4], andererseits waren die Batterielaufzeiten um ein vielfaches höher als bei den Farbgeräten. Während die Farbgeräte nach wenigen Stunden den Betrieb ohne Netzanschluss oder neue Batterien verwehrten, lief der GameBoy rund 10h mit den gleichen Batterien. Die Spiele waren mit DM 50,- auch recht günstig und so verwundert es nicht, dass Nintendo mit dem GameBoy und seinen Nachfolgern GameBoy Pocket, Color und Advance einen mittlerweile in der Spielgeschichte unerreichten Erfolg verbuchen konnte[5]. Nach dem Scheitern von Atari, Sega und NEC versuchten andere Formate immer wieder einen Angriff auf den mobilen Unterhaltungsthron Nintendos, zuletzt die

4 GameBoy ca. DM 150,-, Lynx etwa DM 250,-

5 mittlerweile 110 Millionen verkaufte Geräte laut [EDGE01a]

traditionsreiche Firma SNK mit ihrem NeoGeo Pocket, die übrigens dieses Jahr Pleite ging.

2.2.1 Spielinhalte auf GameBoy und Co.

Die Spiele auf den Handhelds der zweiten Generation waren meist mehr oder weniger komplexe Action-Spiele, *Jump'n'Runs* wie „Super Mario Land"/Nintendo etwa oder Klassiker wie „California Games"/Epyx. Sie zeichnen sich durch leichte Zugänglichkeit aus, waren und sind ideal für die Unterhaltung zwischendurch geeignet.

Teilweise gab es hier schon Titel, die Spiele mit mehreren Teilnehmern per Verbindungskabel zuließen– „Slime World" für Lynx und „RC Pro Am" für GameBoy boten sogar einen Vier-Spieler-Modus, in dem man vier Geräte vernetzen konnte.

Oft wurden auch alte *Arcade*-Hits für die mobilen Plattformen neu aufgelgt. Beispiele hierfür sind etwa „Qix" oder „Centipede". Bezeichnend ist für alle Spiele auf diesen Handhelds, wie schon erwähnt, die einfache Zugänglichkeit der Titel.

Um hier eine einfache Aussage darüber zu treffen, welche Spielinhalte für mobile Geräte erfolgreich sind ist sehr schwer. Die Zielgruppe für GameBoy-Spiele ist gegenüber Playstation und PC sehr jung, Spiele werden auch für Spieler unter zehn Jahren zugeschnitten. Entsprechend hoch ist wahrscheinlich deshalb der Erfolg bekannter Charaktere. Unter den Titeln finden sich z.B. Spiele mit Charakteren aus „Die Sendung mit der Maus" oder mit einer „Janosch"-Figur. Hinzu kommt, dass für Kinder oft die Eltern das Spielzeug und damit die Computerspiele kaufen. Sie kaufen also nicht unbedingt, das was das Kind gerne spielt, sondern viel eher das, bei dem sie denken, dass es das Kind spielen will oder gerne wollen, dass es das spielt. Beispielhaft für diese Problematik ist das Spiel Pokémon, das in den letzten Jahren den GameBoy-Markt bestimmt hat und noch immer bestimmt. Pokémon hatte und hat als Marke wie als Computerspiel einen sehr großen Erfolg. So finden sich Pokémon-Titel seit den letzten zwei Jahren auf

Spitzenplätze der VUD-Spiele-Charts[6]. Sicherlich ist das Spielprinzip vom strategisch-orientierten Fangen und Kämpfen nicht schlecht, doch bleibt fraglich, ob der Erfolg von Pokémon nicht eher durch die Marke als durch das Spielprinzip zu begründen ist.

Auf den restlichen Plätzen der Charts positionieren sich die „üblichen Verdächtigen“: Teilweise tauchen sogar wieder Tetris-Varianten oder Super-Mario-Klassiker auf. Alle Spiele haben jedoch eines gemeinsam, meist sind sie kleine Actionspiele, wie *Jump'n'Runs*, *2D-Shooter* oder im Vergleich zu „Epen“ wie „Ultima“, kleine Rollenspiele wie „Zelda“.

Auch aus meiner eigenen Erfahrung als Spieleentwickler kann ich sagen, dass Publisher in der Regel nur dann ein GameBoy-Spiel vermarkten, wenn es eine starke Lizenz, also einen umsatzversprechenden Charakter aus der Spielzeug- oder Comicwelt beinhaltet.

Erst der GameBoy Advance hat die Haltung der Publisher gelockert. Das Problem, dass der Erfolg eines GamBoy Spieles bisher weitgehend durch die Marke bestimmt war, bleibt jedoch bestehen. Aus analytischer Sicht heißt dies gleichzeitig, dass sich kein direkter Schluss aus irgendwelchen Verkaufcharts ziehen lässt, welche Inhalte nun als Killerapplikation der mobilen Spiele zu gelten haben.

6 Verband der Unterhaltungssoftware Deutschland e.V., www.vud.de

2.2.2 Technik der mobilen Spielkonsolen der zweiten Generation

Anders als die LCD-Spiele führten die ersten echten mobilen Spielekonsolen Rastergrafik auch im mobilen Bereich der interaktiven Unterhaltung ein. Es war nun möglich, ganze Landschaften als Hintergrund zu bewegen und einzelne Spielobjekte auf dem Bildschirm zu verschieben und zu animieren – auf entsprechenden Endgeräten sogar farbig. Außerdem hatte man nun die Möglichkeit, neue Spiele zum Basissystem hinzu zu kaufen, um immer wieder neue Spiele spielen zu können. Sound wurde meist über FM-Synthese oder ähnliche Verfahren generiert. Die Leistungsfähigkeit der Geräte entspricht damit im Wesentlichen denen der 8-Bit Homecomputer und Spielekonsolen, wie Commodore C64, Armstrad CPC oder Sega Master System.

Damit boten sie Realisationsmöglichkeiten von Spielen, die bereits über die der ersten großen Spielekonsole, dem Atari VCS[7], hinaus gehen.

Dies ist unter anderem dadurch möglich, da sie neben kleinen 8-Bit CPUs, wesentliche Grafikfunktionen wie *Scrolling* und *Sprites* als Hardwarefunktionen zur Verfügung stellen, welche natürlich viel schneller arbeiten als eine Softwareimplementierung.

2.3 Der GameBoy Advance – eine dritte Generation?

Mit dem GameBoy Advance setzte Nintendo 2001 den Erfolgskurs des GameBoys fort. Mittlerweile neben dem GameBoy Color die einzige aktiv vermarktete mobile Konsole in Europa, stellt der GameBoy Advance zumindest technisch die dritte Generation mobiler Unterhaltung dar. Leistungsfähiger und bunter lassen sich hier interaktive Unterhaltungsinhalte gestalten.

2.3.1 Technische Evolution des GameBoy

Technisch und von Spieleinhalten her ist der GameBoy Advance in etwa auf dem Niveau der 16-Bit Homecomputer Generation, wie Amiga oder Atari ST, oder 16-

7 auch als Atari 2600 bezeichnet

Bit-Konsolen, wie dem Super Nintendo oder Sega Genesis, anzusiedeln. Im Wesentlichen stellt er eine konsequente Weiterentwicklung des GameBoy Colors dar – mehr Farben, beserer Sound und eine leistungsfähigere CPU[8].

2.3.2 Spielinhalte Advance

Mittlerweile bewirbt Nintendo auch explizit die Multispieler-Fähigkeiten des GameBoy Advance, etwa bei „Mario Kart Advance“. Teilweise sollen auch Spiele auf GameBoy Advance und auf Nintendos neuer Heimkonsole GameCube gleichzeitig über ein Verbindungskabel spielbar sein. Die Steuerung des Game-Cube Titels auf dem Fernseher kann dann über den GameBoy Advance erfolgen, wobei zusätzliche Szenarien über das Display des GameBoy Advance dargestellt werden können. Die erhältlichen Spiele entsprechen in ihrer Summe Jump'n'Runs und Actionspielen, es tauchen jedoch auch erste *First-Person-Shooter*, wie z.B. Doom, auf. Spielerisch bieten sie, verglichen mit der alten GameBoy-Generation, in der Regel nur wenige Innovationen. Lediglich die „Darreichungsform“ ist teilweise etwas anders, statt eines „Ottifanten“ hüpft nun z.B. ein Neandertaler durchs Bild. Die Spiele sprechen nun auch teilweise ältere Spieler an, oder zumindest versucht dies Nintendo mit seiner Werbung „24:7-Gaming“, also spielen „rund um die Uhr“ und überall.

Natürlich ist es abgesehen von der obig beschriebenen Konvergenz zwischen Heim- und Mobilkonsole und den nicht wirklich neuen Multiplayerfähigkeiten fraglich, ob mit dem GameBoy Advance wirklich eine neue Generation eingeläutet wurde oder einfach nur die Spezifikationen etwas höher geschraubt wurden.

Auch die genaue Analyse des GameBoy Advance-Marktes fällt schwer, bzw. ist nicht seriös durchzuführen. So ist er in Europa gerade ein mal ein halbes Jahr auf dem Markt und auch Jahresbücher der Verbände wie ELSPA[9] und VUD dürften auf Grund dieser Gegebenheit nur eine verzerrte Darstellung der Marktlage liefern–sobald sie erhältlich sind. Außerdem sind Verkaufstarts von Konsolen immer

8 Der GameBoy Advance bietet im Gegensatz zum Game Boy Color ca.32.000 gegenüber ca. 32 Farben, zusätzlich zum 4Bit-Sound und FM-Synthese zwei 8-Bit Soundkanäle und eine 32Bit ARM-CPU mit ca.17Mhz statt einer 8Bit CPU mit ca. 8Mhz, vgl. [Frohwein02].

9 European Leisure Software Publishers Association, www.elspa.com

mit etwas Vorsicht zu geniessen. So lässt sich sicherlich erst in einigen Monaten detailliert über die Spielinhalte des GameBoy Advance genaueres sagen, bzw. welche Inhalte von den Publishern oder den Kunden bevorzugt werden.

2.4 Wireless Gaming – die vierte Generation

Zweifellos stellen die bei uns langsam erhältlichen Spiele für Handys und PDAs und vornehmlich durch WAP-Spiele und SMS vertretenen Inhalte mit ihrer Wireless-Komponente, eine neue, eine vierte Generation von mobilen Spielen dar. Erstmals ist es möglich, auch über große Distanzen hinweg, unterwegs mit anderen Spielepartnern zu interagieren. Auch wenn die visuellen und auditiven Möglichkeiten zunächst hinter den Fähigkeiten aktueller Konsolen und PCs zurückliegen, entstehen durch die neue Form des *Multiplays* und der Mobilität neue Möglichkeiten, die von den Spieleentwicklern erst entdeckt und genutzt werden müssen.

Wie auch der i-mode Service in Japan zeigt, sind hier vor allem Spiele möglich, die den Kindertagen der Computerspiele entsprechen. Teilweise sind das kleine Fussball-Spiele und einfache Schiessspiele, in ihrer Gesamtheit aber wieder einfache kleine Geschicklichkeitsspiele und Reaktionstests.

Dass diese in ihrer audiovisuellen Präsentation besser dastehen als europäische und nordamerkanische Inhalte, liegt vor allem daran, dass die technischen Spezifikationen für Handies in Japan auch für andere Services wie J-Phone wesentlich höher liegen. Das heißt, dass viele in Japan erhältliche Mobiltelefone, z.B. ein farbiges und im Vergleich zu den meisten europäischen Handies ein größeres Display bieten.

Eine alternative aber ähnliche Darstellung zur Geschichte der mobilen interaktiven Unterhaltung findet sich übrigens in [EDGE01a]. Da diese Diplomarbeit im Speziellen „Mobile und Wireless Entertainment“ behandelt, sei für eine tiefergehende Analyse der inhaltlichen und technischen Gegebenheiten des mobilen und drahtlosen Spielens der vierten Generation auf die nächsten Kapitel verwiesen.

Die technischen Möglichkeiten der Spiele haben sich über die Jahre hinweg kontinuierlich gesteigert. Die Spielinhalte aber sind über die Jahre hinweg relativ gleich geblieben, sieht man einmal von der starken Interaktionssteigerung von LCD-Spielen zu mobilen Spielekonsolen der ersten Generation ab. Die Spiele werden charakterisiert durch einfach zugängliche Spielinhalte, ein Einstieg in das Spiel ist somit schnell gegeben. Diese These, dass einfache Spiele für mobile und drahtlose Unterhaltung geeignet sind, soll in den nächsten Kapiteln näher untersucht werden.

3 Analyse der gestalterischen Möglichkeiten

In diesem Kapitel sollen die gestalterischen Möglichkeiten der mobilen Clients untersucht werden, um bestimmen zu können, welche Formen der interaktiven Unterhaltung interessant, grundsätzlich technologisch umsetzbar und wie Inhalte gegebenenfalls umzusetzen sind. Dabei schließt diese technische Betrachtung keine detailierte Analyse der speziellen Möglichkeiten bestimmter mobiler Plattformen ein[10]. Vielmehr findet hier eine Zusammenfassung der grundsätzlichen technischen Voraussetzungen statt, um davon die gestalterischen Möglichkeiten ableiten zu können.

Die gestalterischen Grundlagen schließen in diesem Zusammenhang auch die Inhalte eines Spieles ein, da auch sie gestaltet werden müssen. Hierzu werden neben Theorien zur Computerspielerstellung und -analyse auch detailiert einzelne Spielkategorien auf ihre Realisationssmöglichkeiten auf mobile Clients untersucht.

Um nun überhaupt vom Standpunkt des Entwicklers bestimmen zu können, wie in etwa Spiele für mobile Clients aussehen sollen, wird auf Grund des jungen Marktes eine Arbeitsgrundlage für das Nutzungsverhalten mobiler und drahtloser Unterhaltung geschaffen, die ehrlicherweise als hypothetisch zu bezeichnen ist. Hypothetisch deshalb, weil der Markt noch so jung ist, dass sich noch keine seriöse Aussage durch Umfragen oder ähnliche Studien fundieren lässt, da mobile Clients erst allmählich auf den Markt kommen.

Auch liessen sich in den konsultierten Quellen, keine Informationen zum bisherigen Nutzungsverhalten speziell von mobilen Spielekonsolen finden, so dass dieses Vorgehen die praktikabelste Lösung sein dürfte. Hinzu kommt, dass die Nutzerstruktur des GameBoy sich nicht unbedingt auf die Nutzer der mobilen Clients übertragen lässt. Der GameBoy ist traditionell eher ein Gerät für Kinder,

10 diese findet sich in Kapitel 4

was die Nutzerstruktur in [VUD01] belegt. Hinweise hierzu stellen auch die Charts der GameBoy-Verkäufe dar, die überwiegend durch kinderrelevante Titel, wie etwa Pokémon, dominiert werden. Im Gegensatz dazu belegen die Untersuchungen des Instituts für Demoskopie in Allensbach [IDA01], dass die „Kernzielgruppe“ der Handybesitzer, die als Nutzer für mobile und drahtlose Computerspiele in Frage kommt, breit über alle Altersschichten verteilt ist[11].

Das Hauptproblem eines Vergleichs ist darüber hinaus, dass es sich bei der Wireless Komponente, der drahtlosen Netzwerfunktionalität, um eine völlig neue Möglichkeit in der Gestaltung von Spielen handelt. Das heißt, dass zu einem neuen wesentlichen Faktor, keine aussagekräftigen Untersuchungsergebnisse existieren können, da sich sowohl die technischen, wie auch die gestalterischen Grundlagen noch in der Entwicklung befinden. Gleichzeitig hat dieser das Potenzial, das mobile Spielen durch seine exponentiell gesteigerten Kommunikationsfähigkeiten nahezu vollständig zu verändern.

Ob mobile Clients an den Erfolg des GameBoy Advance anknüpfen können ist nicht klar. Martin Chudley, „Bizarre Creations“-Gründer, sieht in [EDGE01c] etwa in der Größe der Geräte ein großes Problem, welches erst gelöst werden muss: „Do you want a phone the size of a GBA (GameBoy Advance, Anm. d. Autors), or a GBA the size of a phone?“. Es bestehen hier offensichtlich schon beim Design der Endgeräte große Differenzen in der angesprochenen Zielgruppe. Es ist also festzustellen, dass mobile Clients eine Zielgruppe ansprechen, die wahrscheinlich von der des GameBoy Advance abweicht. Konkrete Aussagen lassen sich auch hier (noch) nicht machen, der GBA befindet sich noch in einer frühen Entwicklungsphase des Marktes, er ist erst seit Mitte 2001 auf dem Markt (je nach Kontinent).

3.1 Zeitliche und räumliche Voraussetzungen

Nutzer von mobilen drahtlosen Computerspielen werden sich wahrscheinlich in sehr unterschiedlichen Situationen wiederfinden, in denen sie von den Unterhal-

11 Handybesitzer, die als Nutzer für mobile und drahtlose Computerspiele in Frage kommen, sind hier Menschen, die ein besonderes Interesse an Spielen auf Handies haben – laut [IDA01] stieg das Interesse seit 1999 sogar von einem Index von 100 Zählerpunkten auf 310.

tungsfunktionen ihres digitalen Begleiters gebrauch machen wollen. Beispielhaft wäre hier der Weg zur Arbeit oder zur Schule zu nennen, im Bus oder in der Straßenbahn. Das Warten an der Haltestelle und die Nutzung zu Hause, um nicht erst das Computersystem „hochfahren“ zu müssen und um kurz einmal zu spielen, sind weitere Einsatzszenarien – die Möglichkeiten sind dank der Einfachheit und geringen Größe mobiler Endgeräte nahezu unerschöpflich.

Ebenso verhält es sich bei der Art der Spiele. Sollen sie allein oder zu mehrt spielbar sein, findet man tatsächlich immer einen Spielepartner oder will man über verteilte Zeiträume hinweg, ähnlich einem Rollenspiel in eine andere Welt abtauchen oder in einer abgeschlossenen Sitzung die Zeit vertreiben? Die vorstellbaren Einsatzszenarien sind sehr vielfältig.

Eine eindeutige Kategorisierung scheint nicht möglich. Vielmehr ist es notwendig die vielfältigen Bedürfnisse des Konsumenten anzusprechen. Ob dies nun innerhalb eines Spieles oder in einem ganzen *Line-Up* geschieht, bleibt dem Produzenten überlassen.

Es erscheint jedoch wichtig zu betonen, dass im Vergleich zu bisherigen Unterhaltungslösungen mit mobiler und drahtloser Unterhaltung das Nutzungsspektrum vor allem um ortsunabhängige und kommunikative Elemente erweitert wird. Andererseits bleiben bisherige Spielbedürfnisse der Nutzer erhalten und werden mit neuen Technologien nicht einfach verschwinden, so dass auch Produkte für nur einen Spieler angeboten werden sollten.

Die Nutzung ist also weder zeitlich, räumlich, noch inhaltlich genau fixierbar. Einer der wenigen Anhaltspunkte mag vielleicht sein, dass der „mobile drahtlose Spieler“ nicht mit der neuesten Spieletechnologie, wie aufwändige 3D-Grafiken und aufwändiger Klanguntermalung geködert werden kann[12], sondern sehr stark über die gesuchte Kurzweil und den Spielspaß angesprochen werden muss. Ziel muss es also sein, einfach zugängliche Unterhaltung zu bieten, die evtl. um zeitlich langfristig angelegte Komponenten, wie *Communities*, *Highscores*, oder große Spielewelten für mehrere Spieler erweitert ist und somit die neuen überall

12 wie *3.2 Technische Voraussetzungen* zeigt

verfügbaren Kommunikationsfähigkeiten ausnutzt. Ebenso muss ein schneller Ein- wie Ausstieg aus dem Spiel gewährleistet sein, um auf die hohen Flexibilitätsanforderungen während der mobilen drahtlosen Nutzung eingehen zu können.

Wir brauchen also für mobile drahtlose interaktive Unterhaltung möglichst leicht zugängliche Produkte, die einfach zu handhaben sind, optimalerweise mehrere Spieler auf unterschiedliche Weise einbeziehen[13] und dabei auch noch ein Spiel in kurzer Zeit ermöglichen, bzw. Spielsitzungen auf kurze Zeit begrenzbar machen[14].

3.2 Technische Voraussetzungen

Um neben den konzeptionellen Rahmenbedingungen auch die technischen Voraussetzungen zu berücksichtigen, findet sich auf den folgenden Seiten eine Analyse der Voraussetzungen, welche eventuelle Zielplattformen mit sich bringen.

Darüber hinaus werden grundlegende Eigenschaften der Plattformen, sowie Vor- und Nachteile erläutert. Eine detailierte Analyse der spezifischen Fähigkeiten findet hingegen erst in *4 Analyse der technischen Möglichkeiten* statt, da der dort gebotene Fokus und Detailgrad vor allem für die Softwareentwicklung notwendig ist.

3.2.1 Allgemeine Systemgegebenheiten

Die Geräte, die man als Entwickler im Bereich mobiler und drahtloser Anwendungen vorfindet, sind sehr unterschiedlich. Angefangen bei portablen Computersystemen bis zu kleinen Mobiltelefonen, findet man beinahe alles, was irgendwie tragbar ist. Glücklicherweise lassen sich für die mobile drahtlose Unterhaltung, die Geräte auf die Kategorie der mobilen Clients einschränken. Doch auch die verbleibenden Geräte sind unterschiedlich genug, um sie einer genaueren Analyse zu unterziehen.

13 gemeinsame Highscores bis simultanes Spielen in einer Welt

14 siehe auch [Seppänen01], worin ähnliche Anforderungen speziell an WAP-Spiele gestellt werden.

3.2.1.1 Visuelle Ausgabemöglichkeiten

Eines der größten Probleme des Spieleentwicklers sind die stark unterschiedlichen Displaygrößen der einzelnen Geräte. Angefangen bei Handies, wie dem „Siemens SL 45i“ mit einer Auflösung von 101x80 Pixeln[15], über den „Nokia Communicator 9210“ mit 640x240 Pixeln oder den Organizern wie dem „Palm Vx“ mit 160x160 oder dem „Compaq iPaq“ mit 240x320 Pixeln Auflösung. Problematisch sind hier nicht nur die verschiedenen Auflösungen, sondern die auch sehr unterschiedliche Seitenverhältnisse der Displays.

Spieleentwicklungen müssen diesem Umstand gerecht werden, da Standards wie das MIDP für Java 2 ME zwar eine Auflösung von etwa 96x96 bei quadratischen Pixeln empfehlen, diese jedoch, wie die Geräte von Siemens und Nokia zeigen, nicht sehr streng realisiert werden.

Ähnlich divergieren die Darstellungsmöglichkeiten in technologischer Hinsicht. Während beispielsweise auf einem PocketPC 2002 dank des leistungsfähigen StrongARM-Prozessors aufwendige 3D-Grafikalgorithmen zum Einsatz kommen können, dürfte auf einem Palm mit Prozessoren der 68k Serie kaum ein 3D-Spiel realisierbar sein.

Gemeinsam ist den Geräten jedoch die im Vergleich zu PCs und Spielekonsolen niedrigere Auflösung, so dass sich die direkte Übernahme des Grafikmaterials auf das Format der mobilen Clients als unpraktikabel herausstellen muss. Dies zwar auch aus technischen Gründen (Farbtiefe), doch vor allem aus der Tatsache heraus, dass es wohl keinen Sinn macht, auf einem 100x100 Pixel großen Display eine Spielfigur mit 50x50 Pixeln Größe gegen ebensolche Gegner antreten zu lassen[16].

3.2.1.2 Steuerungsmöglichkeiten

Das nächste Problem ist die Steuerung des Spieles. Die Tastenausstattung und damit die Steuermöglichkeiten sind auf den unterschiedlichen Geräten nicht ein-

15 laut Debug-Ausgabe

16 Eine genauere Beschreibung von Lösungsmöglichkeiten zum Umgang mit der Displayproblematik findet sich in *3.5 Gestalterische Überlegungen zu Grafik und Ton von mobilen Computerspielen*.

heitlich. Zum Beispiel findet man auf dem „Communicator“ von Nokia fast eine komplette Tatstatur wieder und auf dem „Siemens SL45i“ immerhin ein Steuerkreuz und ein, wie für Handies üblich, numerisches Eingabepad. Auf dem „Accompli 008“ von Motorola hat man dann zwar einen *Touchscreen*, dafür aber nur vier Tasten, was für die meisten Spiele eindeutig zu wenig ist, wenn man mehr als nur die Richtung steuern möchte.

Weiterer wichtige Gesichtspunkte, die durch die unterschiedlichen Steuerungsmöglichkeiten und -realisierungen entstehen, sind die Auswirkungen auf den Spielverlauf, bzw. auf die Spielbarkeit. So ist davon auszugehen, dass ein Benutzer ein Spiel mit einem dafür optimierten Steuerkreuz besser spielen kann, als über ein einfaches numerisches *Pad*. Im Falle eines Multispieler-Titels multiplizieren sich diese Eigenheiten, bedenkt man welche großen Vorteile hier allein durch das verwendete Gerät gegeben würden. Die Auswirkungen sind sehr schwer abschätzbar. Dem entsprechend ist es schwer, mit dieser Gegebenheit einheitlich umzugehen, bzw. eine Lösung zu finden, die diese automatisch berücksichtigt.

3.2.1.3 Auditive Ausgabemöglichkeiten

Soundunterstützung gibt es in der mobilen drahtlosen Welt nur teilweise. Unterstützende Plattformen sind „PocketPC“, „Symbian OS“, „PalmOS“ und unter „Java 2 ME“ das „Personal Profile“, jedoch nicht das von Sun als Quasi-Spiele-Standard propagierte „MIDP“. Die Soundmöglichkeiten unterscheiden sich von Plattform zu Plattform, d.h. in der Regel wird oder muss man auf die spezifischen Möglichkeiten der Systemumgebung eingehen. Einige Anbieter von Mobiltelefonen erweitern das MIDP um proprietäre Sounderweiterungen, so dass deren Vorhandensein im Spieldesign unter Umständen berücksichtigt werden sollte.

Während die visuellen Eigenschaften für Spiele in der Regel weitaus schwerwiegender sind als die auditiven – der Mensch ist ein „Augentier“, [Hentrich97]– lassen sich so für die Soundmöglichkeiten auf den unterschiedlichen Java-Systemen, eine Lösung durch eine abstrakte Programmierschnittstelle finden, die die jeweiligen Möglichkeiten optimal ausnutzt.

Die Problematik der Visualisierung und der Steuerung muss jedoch schon bei der Konzeption des Spieles berücksichtigt werden, ebenso wie das evtl. vollständige Fehlen von Soundwiedergabemöglichkeiten. Möglicherweise erlaubt auch der Einsatzort nicht das Verwenden von Sound. Für den Spieldesigner bedeutet dies, dass er das Spiel so gestalten muss, dass durch die unterschiedlichen Anzeigen, Soundwiedergabe- Steuerungslösungen möglichst geringe Vor- bzw. Nachteile beim Spielen entstehen.

3.2.2 Technologien zur Datenübertragung zwischen mobilen Endgeräten

EDGE, GPRS, UMTS, MMS – die Namen für neue Datenübertragungsstandards sind vielfältig und verwirrend. Doch haben sie eines gemeinsam: Sie sollen die Netzwerkfähigkeiten der mobilen Clients verbessern. Dazu bieten neue Übertragungsstandards wie UMTS, EDGE und GPRS eine so genannte Packet-Switched-Datenübertragung an. Für den Endkunden bedeutet dies, dass er nur für die Daten bezahlen muss, die er tatsächlich überträgt, dass er immer online ist[17] und dazu die Daten auch noch schneller übertragen kann – je nach Technologie zwischen 50kBit/s[18] und maximal 2Mbit/s bei UMTS[19] (vgl. [Hübner01]).

Für Spieleentwickler stellt dies erstmals ausreichend schnelle Vernetzungstechnologien für schnelle interaktive Titel in Aussicht. Bisher waren GSM-Netze nur in der Lage, Daten bei maximal 9,6kBit/s (vgl. [Hübner01]) zu übertragen, was aus meiner Erfahrung einfach zu langsam für Computerspiele ist, rechnet man die Ausfälle durch die drahtlose Übertragung hinzu.

Doch auch Bluetooth als Standard für Lokale Private Netzwerke (LPANs) könnte eine wertvolle Bereicherung für Spiele darstellen, da bei der Nutzung keine Gebühren anfallen. Allerdings ist es auch nur im Umkreis von ca. 10m einsetzbar, was aber ausreichen dürfte, um Spielesitzungen im engeren Kreis abzuhalten. Die Übertragungsrate ist ebenfalls mit ca. 800kBit/s für Spiele vollkommen ausreichend (vgl. [NokiaXX]).

17 auch als always-on bezeichnet

18 bei GPRS und EDGE, welche auch als 2.5G-Technologien bezeichnet werden

19 allerdings nur im Idealfall bei dieser 3G-Technologie

MMS (Multimedia Messaging Service) als multimediale Evolution von SMS, wird vielleicht für Spiele interessant, die ohnehin auf zeitversetzte Interaktion setzen, wie sie unter *3.6 Marktübersicht der aktuellen mobilen drahtlosen Computerspiele* behandelt werden. MMS bietet im Gegensatz zu SMS, Möglichkeiten zur Bild und Tonübertragung[20] (vgl. [Nokia01b]).

3.3 Spieltheoretische Grundlagen und Modelle

Um die theoretischen Grundlagen und Modelle für mobile und drahtlose Spiele berücksichtigen zu können, soll hier zunächst eine Überischt zu Theorien über (Computer-)Spiele gegeben werden, um dann die einzelnen Theorien im Hinblick auf ihre Relevanz für Mobile und Wireless Entertainment untersuchen zu können.

3.3.1 Mathematische Grundlagen

Auf der Suche nach geeigneten Quellen auf wissenschaftlich fundierter Basis wird man zunächst nur schwer fündig. Eines der wenigen Werke, die sich mit der Spieleentwicklung auf diese Art beschäftigen, ist [Bewersdorff01], das sich mit mathematischen Modellen zur Beschreibung von Spielen allgemein auseinandersetzt. Darüberhinaus ist diese Quelle hilfreich, um Mobile und Wireless Entertainment in den Kontext von „normalen“ Spielen setzen zu können.

Laut [Bewersdorff01], liessen sich Spiele grundsätzlich in kombinatorische, strategische und Glücksspiele unterteilen, bzw. würden sich aus Teilelementen dieser drei Grundtypen zusammensetzen (S.Vff).

Spiele, die rein auf zufälligen Ereignissen beruhen, werden wenig verwunderlich als Glücksspiele klassifiziert. Hier werden als Beispiele vor allem Würfelspiele genannt (S.V).

Kombinatorische Spiele sind laut [Bewersdorff01] Spiele, bei denen der Spieler bei jedem Spielzug die Möglichkeit habe, „durch die Spielregeln fixierte Hand-

20 Für eine weiterführende Betrachtung der Netzwerktechnik sei auf *5.3.3.5 Netzwerkkomponente* verwiesen.

lungsmöglichkeiten“ auszuwählen. „Bereits nach wenigen Zügen können sich die erlaubten Möglichkeiten zu einer kaum noch überschaubaren Vielfalt kombinieren, so dass die Konsequenzen eines einzelnen Zuges nur noch schwer zu erkennen sind.“, so Bewersdorff. Als Beispiele werden hier die klassischen Denkbrettspiele wie Schach oder Dame angeführt (S.VI).

Strategische Spiele seien hingegen solche, bei welchen es überwiegend darum gehe, einen Erfolg auf Basis eines Zuges des Gegners zu erzielen, wie z.B. bei „Papier-Steine-Schere“. Den teilnehmenden Parteien sei nicht bekannt, welchen Zug der Spielpartner als nächstes vornehme, dieser Zug sei jedoch essentiell für den eigenen Erfolg (S.VIff).

Natürlich seien in den genannten Beispielen für die einzelnen Spielkategorien auch Teilelemente der jeweils anderen enthalten. Dies zeigen auch andere Beispiele wie „Mensch ärgere dich nicht“, das neben Glücksselementen durch das Würfeln auch kombinatorische Elemente und meiner Meinung nach abweichend von [Bewersdorff01], S.VI, strategische Elemente enthält. Beispielsweise bei der Ungewissheit, welche Spielefigur ein Gegner beim nächsten Zug bewegt und damit unter Umständen eine der eigenen Figuren in Gefahr bringt.

Als mathematische Grundlagen der einzelnen Spielkategorien werden folgende Modelle aufgeführt:

„Glücksspiele können mit Hilfe der Wahrscheinlichkeitsrechnung analysiert werden.“(S.VIII), so Bewersdorff, was angesichts des Versuches dieser mathematischen Disziplin, Zufälliges berechenbar zu machen, auch nicht verwundert.

„Strategische Komponenten eines Spieles“ lassen sich auf Grund ihrer Natur durch die so genannte Spieltheorie beschreiben (S.VIII). Sie beschäftige sich mit Analyse und Beschreibung der Möglichkeiten und Konsequenzen, die Entscheidungen interagierender Parteien haben, bei gleichzeitigem Unwissen über die Entscheidung der anderen Partei.

Bisher wurde die Spieltheorie als Disziplin der Mathematik überhaupt nicht in den für Spieleentwicklern wichtigen Quellen, wie [Bates01] oder [Rollings00] behandelt. Das liegt wahrscheinlich daran, dass Spiele in der Spieltheorie eigentlich eine andere Funktion haben: „Spiele fungieren dort als Modell, auf deren Basis interaktive, ökonomische Prozesse in Abhängigkeit von getroffenen Entscheidungen untersucht werden.“ ([Bewersdorff01], S.VIII).

„Für die kombinatorischen Elemente in Spielen gibt es keine einheitliche Theorie.“ ([Bewersdorff01], S.VIII). Vielmehr werde durch spezifische Modelle eine mathematische Beschreibung erzielt. Prominentes Beispiel hierfür sei Schach, wobei der Sieg des Rechners Deep Blue über den bis dahin ungeschlagenen Schachweltmeister Kasparov, diese Kategorie ins breite Licht der Öffentlichkeit gerückt habe (S.172).

Vergleicht man nun diese Möglichkeiten der Mathematik, so fallen einem in Verbindung mit Computerspielen einige gravierende Nachteile auf. Dies ist zum einen die fehlende Beschreibung des hohen Grades der Interaktivität, den Computerspiele spätestens seit den frühen 80er Jahren haben. Diese wird in den beschriebenen Modellen nicht berücksichtigt.

Allerhöchstens lassen sich diese Spiele als solche mit schnell hintereinanderfolgenden Spielzügen betrachten, wie es auf Entwicklungsebene eines Computerspieles auch geschieht – und auch in [Rollings00] und [Loki01] beschrieben wird, allerdings nicht im Zusammenhang mit mathematischen Theorien. Die Auswirkungen dieser Interaktivität auf das eigentliche Spiel wird jedoch nicht erfasst. Es bleibt bei einer Beschreibung von einzelnen Spielzyklen, bzw. Zügen. Lediglich einzelne Beziehungen innerhalb eines Spieles, werden in [Rollings00] mit Hilfe von mathematischen Gleichungen beschrieben.

Es fehlt an einer Beschreibung von wesentlichen Elementen wie des Spielspaßes, der durch mathematische Hilfsmittel allenfalls eingeschränkt analysiert werden kann. Diese Hilfsmittel reichen auch nicht aus, um ein Spiel in seiner Gesamtheit mit psychologischen Effekten, wie Spannungsaufbau, Glücksempfinden usw., zu

beschreiben, die auch durch ganz andere Elemente wie Gestaltung hervorgerufen werden.

Ein (Computer-)Spiel lässt sich mittels der Mathematik also nur in Teilpunkten beschreiben, bzw. analysieren. Weitere wichtige Teilpunkte, wie die Gestaltung, werden im Folgenden betrachtet.

3.3.2 Theorien aus dem pädagogischen Umfeld

[Fritz97] beinhaltet einige Inhalte, die ich als Basis für meine weiteren Hypothesen hier nochmals aufführen möchte:

Computerspieler wählen die Spielwelt in der sie spielen möchten, „Wie mediale Welten sind auch virtuelle Welten Wunschwelten nach Wahl“ (S.27). „Diese virtuellen Welten erlauben es, die Ich-Grenzen immer weiter auszudehnen, indem diese Welten die Möglichkeiten bieten, vielfältige Rollen und Funktionen wahrzunehmen, die einem ansonosten verschlossen sind, und darin Erfolg zu haben.“ (S.26f).

Hier trennen sich die Computerspiele der virtuellen Welten, wie „Tomb Raider“, „Quake“ oder „Super Mario 64“ – Spiele die eine eigene Welt generieren, von denen die z.B. nur ein einfaches Abbild eines Kartenspieles, wie Solitaire darstellen.

Doch sollten hier nicht voreilig Umsetzungen von Brettspielen wie Schach oder Monopoly hinzugezählt werden. Auch sie erzeugen schon in ihrer nicht elektronischen Variante eine eigene Welt, die des Krieges und die des ökonomischen Handelns[21], wobei der Computer meiner Meinung nach dazu dient, diese Spiel-Atmosphären zu verdichten.

Diese Form der Verdichtung erwähnt auch [Fritz97], allerdings wird die Verdichtung erst auf Ebene der weiterführenden Hilfsmittel erwähnt, „wie Datenhelm, Datenhandschuh und Datenanzug“ (S.27). Es ist jedoch fraglich, in wie weit der

21 eine eigene Welt, die Fritz als Spielwelt bezeichnet (vgl. S.19ff)

Gebrauch von jeglichen Hilfsmitteln, seien sie elektronischer oder simpler materieller Natur, nicht zur Verdichtung des Spielerlebnisses beitragen (auch schon die Figuren eines Brettspieles) – die natürlich ihren Gipfel in den Hilfsmitteln der virtuellen Realität findet.

Hierbei ist vielleicht auch anzumerken, dass [Fritz97] in „Lebenswelt und Wirklichkeit“, mehr die Beziehung zwischen den Erlebniswelten des Menschen und der Realtität untersucht, hier hingegen die konkreten Computerspiele-Welt betrachtet wird. Jürgen Fritz nimmt hierbei eine weitere Abgrenzung in Spielewelt, mediale Welt und virtuelle Welt vor, die sich eigentlich in Computerspielen je nach Produkt unterschiedlich mischen.

Was die Dichte der erzeugten Spielwelt angeht, so wird klar, dass mit einem Handy oder PDA mit kleinem Bildschirm und minderwertigen Sound, nicht die selbe Atmosphäre wie mit einer Hochleistungskonsole wie PS2, Xbox, Dreamcast oder GameCube erzeugt werden kann, was zunächst einmal einen klaren Nachteil darstellt.

Laut [Renner97] fallen Computerspiele in die Kategorie der Regelspiele. Regelspiele seien solche, „bei denen die Regel zum Spielinhalt wird oder zumindest stark in den Vordergrund rückt“ (S.68). Tatsächlich ist dies praktisch bei allen Computerspielen der Fall, da die ihnen inne wohnende Logik durch Algorrithmen diskret beschrieben wird und eigentlich keine Ungenauigkeiten und Regelverstösse zu lässt.

Bemerkenswert scheint in diesem Zusammenhang, dass Renner Computerspiele im Vergleich zu menschlichen Spielepartnern (natürlich) als emotionslos ansieht, dass aber diese fehlenden Emotionen durch die des Spielers kompensiert werden würden: „Was dem Computerprogramm an Emotionalität fehlt, wird durch die Reaktionen aus der Innenwelt der SpielerInnen ergänzt.“ (S.70). Und weiter: „Im Videospiel spiegelt sich nicht nur der technologische Stand der Industrienationen wider, sondern auch die Innenwelt der 'Videonauten': ihre Ängste, Sehnsüchte, Wünsche, Gefühle, Wertvorstellungen, Normen und Lebensortientierungen.“ (S.71).

Bringt man diese Überlegungen mit den betrachteten Theorien von [Fritz97] in einen Kontext, lässt sich daraus folgern, dass ein Großteil eines Computerspiels im Kopf entsteht und deshalb seine mediale Repräsentation gar nicht so wichtig ist, wie man vielleicht glaubt. Dies würde übrigens auch erklären, warum sich simple 2D-Spiele sehr gut in den Verkaufscharts positionieren können. Beispiele hierfür wären „Pulleralarm“ oder „Wer wird Millionär?“ im Dezember 2000, die im Vergleich zu 3D-Spielen eher schwächlich wirken. Sicherlich, diese Spiele haben oder hatten ein gewaltiges Marketing im Rücken, aber dieses diente letzendlich dazu, einen Teil der Spielewelt schon vorher im Kopf des Spielers zu bauen.

Im Hinblick auf die Fähigkeiten von mobilen drathlosen Clients bedeutet dies, dass deren eingeschränkte Fähigkeiten, gar nicht so schwer wiegen, bzw. dass sich trotz dieser Einschränkungen Computerspiele produzieren lassen, die den Benutzer fesseln können.

Als weitere wichtige Eingenschaften eines erfolgreichen Spieles werden in [Fritz97], „Erfolgreiche Videospiele“ von Werner Rudolph, Eigenschaften wie Spieltiefe (Möglichkeiten der Interaktion), einprägsame Charaktere (bspw. „Mario“ und „Luigi“ von Nintendo) und eine interessante Rahmenhandlung (Bsp. „Legend of Zelda“) genannt (vgl. S.171ff). Am Beispiel von Nintendo-Produkten wird genau auf diese Eigenschaften eingegangen. Dadurch belegt sozusagen der Verweis auf den Nintendo GameBoy (vgl. S.172), dass genau diese wichtigen Eigenschaften auch auf einem mobilen Endgerät mit einfacher Grafik möglich sind.

3.3.3 Theorien aus der Welt der Spieleentwickler

Die bisherigen Theorien aus dem Umfeld der Mathematik und Pädagogik konnten bislang entweder nur analytische Hilfsmittel oder ganzheitliche Betrachtungen, wenn auch sehr aufschlussreiche, liefern.

In [Rollings00] finden sich Ansätze, die die Erschaffung eines Computerspieles mehr mit der Produktion eines Spielfilmes in Verbindung bringen. Prinzipiell basieren laut dieser Quelle alle Computerspiele mehr oder weniger auf klassischen Elementen des Dramas:

„The player is in control of the game, after all. But I would say that an interactive game is really no different from a work of more 'classical' art. For example, if you read the epic poem, *The Iliad*, you construct your own unique narrative, which certainly differs from what Homer had in mind. Even puzzle games like *Tetris* in one sense tell a story, and are thus 'dramatic' to an extent.“ (S.8).

Und weiter:
„All good computer games must entertain, and most gain a great deal of their entertainment value from drama. The fact that the player has greater control of the drama than in any other genre of art is a difference in degree, not form. The undelying rules of drama therefore still apply; because they are aesthetic rules dictated by the human mind. So it's instructive to look at the dramatic elements of a game...“ (S.8).

Die Handlung eines Computerspieles werde dabei weitgehend vom Spieler selbst bestimmt (S.9). Sogar Strategiespiele würden so nach den Beschreibungen in [Rollings00] eine Handlung entwickeln (S.10). Und auch Spiele, die scheinbar keinen Hauptcharakter oder Protagonisten haben würden, würden selbst in extremen Fällen, wie bei Wirtschaftssimulationen, eben doch einen besitzen, nämlich den Spieler selbst (S.10). Computerspiele gäben dabei meist den Handlungshintergrund, das Setting, vor (S.10), während der Spieler in manchen Spielen sogar das Thema selbst bestimmen könne (S.11). Wie etwa bei „Deus Ex“/Eidos, in welchem er selbst wählt, ob er auf Seiten von Terroristen oder von Regierungseinheiten kämpfen will oder etwa bei zahlreichen „Star Wars“-Spielen, in denen man auch die Seite der „Dunklen Macht“ ergreifen kann und damit selbst entscheidet, ob das Böse über das Gute siegt.

Auch diese Theorie korreliert mit den bisherigen Betrachtungen. Computerspiele geben damit dem Rezipienten die Möglichkeit, Dinge auszuprobieren, die über

den eigentlichen Erfahrungshorizont herausgehen, wie es in [Fritz97] beschrieben wird.

Neben diesen Betrachtungen liefert [Rollings00] konkrete Anleitungen und Entwurfswerkzeuge zur Entwicklung von Computerspielen. Das sind neben Anleitungen zum Aufbau und Inhalt der Dokumentation, vor allem auch ein theoretisches Fundament zur Entwicklung eines *Gameplays*, also wie das Spiel prinzipiell funktioniert. Gameplay setze sich nach [Rollings00], aus interessanten Auswahlmöglichkeiten zusammen. Wobei die Möglichkeit des Verwendens von unterschiedlichen Strategien und Interaktionsmöglichkeiten, das Spiel für den Nutzer reizvoll mache (S.38ff).

Dabei wird auch eine deutliche Distanz zwischen Interaktivität und Gameplay geschaffen. Laut [Rollings00] gibt es nämlich durchaus Unterhaltungssoftware, die zwar kein echtes Spielprinzip besitzen würde, aber durch ihre Interaktionsmöglichkeiten das Produkt nutzvoll mache. Die These geht sogar noch weiter: „Interactivity is what computers do best. Thus, interactivity, much more than gameplay, is the heart and soul of entertainment software." (S.53)[22].

Neben diesen Betrachtungen finden sich detailierte Anleitungen zur Balancierung der Spielmechanismen, -inhalte und -ziele. Balancierung[23] bedeutet in diesem Zusammenhang, dass Möglichkeiten des Spielers und Schwierigkeit des Spiels unter- und gegeneinander ausgewogen werden. Diese Beziehungsgefüge werden teilweise auch mit Hilfe mathematischer Gleichungen dargestellt, wie im mathematisch-theoretischen Teil dieses Kapitels bereits angedeutet wurde.

Das Balancing wird dabei in „Player/Player Balance", „Player/Gameplay Balance" und „Gameplay/Gameplay Balance" aufgeteilt, die wiederum eine „Component and Attribute Balance" enthält.

22 Zumindest im Hinblick auf die Entwicklungsgeschichte der Unterhaltungssoftware ist das streitbar, beinhalten doch Pionierwerke, wie „Asteroids" und „Pong", stets auch ein Gameplay als essentiellen Teil.

23 siehe auch *Game Balancing*

„Player/Player Balance“ behandelt die Balance zwischen einzelnen wählbaren Spielcharakteren, wie etwa bei einem Prügelspiel. Dabei sei laut [Rollings00] wichtig, dass die unterschiedlichen Charaktere unterschiedliche Vor- und Nachteile haben sollen, die sich in ihrer Summe jedoch wieder ausgleichen (S.74).

„Player/Gameplay Balance“ meint vor allem, dass das Spiel nicht unnötig hinderlich für den Spieler sein soll: „This aspect of game balance can be reduced to three simple rules: Reward the player. Let the machine do the work. Make a game that you can play *with,* not *against*“ (S.78).

„Gameplay/Gameplay Balance“ bedeutet eine Balancierung des Gameplays in sich: „Broadly, we must consider three things: We want there to be a variety of interesting choices rather than a single choice that always dominates. This isn't easy to establish because the optimum choices depend on the choices other players make. It's not easy to see how frequently different choices will be worth making, yet we need to know that to balance the game.“ (S.82). Die „Component and Attribute Balance“ geht dabei auf das Ausbalancieren der atomaren Spielelemente ein, also z.B. welchen Grad der Verwundung ein bestimmter Schlag in einem Prügelspiel verursacht.

Im Wesentlichen lässt sich zusammenfassend sagen, dass sich die hier im Design-Teil von [Rollings00] beschriebenen Theorien, auch für die Erstellung von Spielen für mobile Endgeräte eignen, selbst wenn hierauf kein Fokus liegt. Für diese Diplomarbeit bedeutet das, dass auch für die Konzeption von „Mobile Games“ theoretische Grundlagen existieren und anwenden lassen.

3.3.4 Anwendbarkeit der Spieletheorien für mobile Endgeräte

Wie die obigen Betrachtungen gezeigt haben, lassen sich die grundlegenden Theorien, soweit solche vorhanden sind, durchaus auf ihre mobilen Varianten übertragen.

Spiele für mobile Clients gewinnen aber duch ihre theoretisch ständige Verbindung mit einem Netzwerk (zumindst ab GPRS) eine weitere Dimension dazu, die die bisherige theoretische Basis der Computerspielkonzeption sprengt.

Somit wird die Kategorie der Netzwerk-Spiele, wie in [FritzXX] beschrieben, sicherlich ein bedeutender Teil der Inhalte. Den Spielern sei es laut [FritzXX] möglich, ein unmittelbares Feedback über das Erlebte auszutauschen. Hierbei scheint es notwendig, im Hinblick auf mobile Netzwerk-Spiele zwei Klassifizierungen vorzunehmen: zum einen in Spiele, die sich lokal mit mehreren Spielern spielen lassen und deren Teilnehmer sich in der Regel kennen oder zumindest unmittelbaren Kontakt haben, zum anderen in Netzwerkspiele, die über eine weite Entfernung gespielt werden und dadurch alternative Kommuniktionsmöglichkeiten der Spieler untereinander bieten sollten. Sicherlich sind hier auch Mischformen sinnvoll. Der Spieleentwickler sollte jedoch diese Überlegungen über das wie, wo, wann und mit wem ein Spiel gespielt wird, bei der Konzeption für eine mobile Plattform besonders berücksichtigen.

Interessant in diesem Zusammenhang ist die Entwicklung im Bereich der Cybercommunities, wie sie auch in [Fritz97], S.137 Erwähnung finden. Also Welten, in denen sich Menschen „nur so zum Spaß“ virtuell treffen. Inwiefern hier die mobilen Clients dazu beitragen können, in der Netzwelt eine ständige Präsenz zu schaffen, ist sicherlich spannend. Hier sei auch auf den Bereich des Pervasive Computings verwiesen, das Computer als ständigen persönlichen Begleiter sieht. Sicherlich wäre es interssant und reizvoll, diese Communities mit mehr Spiel zu füllen und trotzdem ein starke Bindung zwischen den Teilnehmern zu erhalten.
In der Projektarbeit an der FH-Furtwangen „visionen b2b“ wurde von einem Team mit meiner Teilnahme, die Welt des Pervasive Computings untersucht. Interessant ist hierbei, wie sich die Entwicklung von der SMS-Kommunikation zur hoffentlich bald verfügbaren Videokommunikation vollzieht und hierbei sicherlich eine Chance bietet, weitergehende Funktionen einzubringen. So wäre es theoretisch möglich 3D-Darstellung oder einfache 2D-Communities über den

PDA oder das Handy darzustellen und an diesen ortsunabhängig teilzunehmen und/oder mitzuspielen.[24]

Stützend für den Erfolg mobiler drahtloser Spiele wird aus [Fritz97] ersichtlich, dass der typische jugendliche Computerspieler kein „Stubenhocker“ ist (vgl. S.51), sondern durchaus „normale“ Freizeitaktivitäten wahrnimmt. Und gerade bei diesen, ist man auch mal unterwegs. Somit würde sich der Spieleeinsatz weiter auf bisher mit dem Computer nicht erreichbare Lokalitäten ausdehnen. In [EDGE01d], beschreibt ein Leser zum Beispiel das gemeinsame Spielen am GameBoy Advance im englischen Pub. Was mit dem GameBoy schon teilweise möglich ist, würde mit mobilen Clients wie Smartphones und der damit einhergehenden Verbindung/Konnektivität zu anderen Spielern, mit denen man in der Freizeit „herumhängt“, weit gesteigert werden.

Abschließend lässt sich also konstantieren, dass bestehende Theorien, vor allem aus dem Bereich der Spieleentwicklung und Pädagogik, die Möglichkeit mobiler und drahtloser Spiele mit ihrem Grundlagenfundus stützen, andererseits aber diese Modelle auf die Wireless-Komponente, wie etwa oben vorgeschlagen, übertragen werden müssen.

3.4 Spiel- und Nutzerkategorien und ihre Relevanz im Hinblick auf die mobile interaktive Unterhaltung

Im Folgenden werden die heute am Markt vertretenen Spielkategorien auf Ihre Umsetzungsmöglichkeiten auf mobile Endgeräte untersucht. Hierzu erfolgt eine auf aktuellen Quellen basierende Aufzählung der einzelnen Kategorien.

3.4.1 Spielkategorien

In [Bates01], S.8ff findet sich eine relativ umfassende Auflisung und Beschreibung der unterschiedlichen Spielgenre, deren Kategorisierung hier übernommen wird.

24 Wobei hier das Einsatzgebiet sicherlich nicht nur auf Spiele begrenzt ist, sondern auch auf den Geschäftsbereich, mit virtueller Präsentation des Egos oder des Produkts, ausgeweitet werden könnte.

3.4.1.1 Adventures

Abenteuer-Spiele („Adventure-Games“, S.8) sind Spiele, in denen der Spieler auf dem Weg zum Ziel verschiedene Rätsel lösen müsse, um weiter zu kommen. Diese Spiele hätten sich aus den so genannten Textadventures, die über textuelle Eingaben gesteuert wurden, entwickelt und konzentrierten sich i.d.R. auf das Lösen der einzelnen Rätsel als spielerisches Haupelement.

Die Umsetzung dieses Spielgenres gestaltet sich zumindest nach traditioneller Manier, wie sie etwa von „King's Quest“ realisiert wird, schwierig, da diese Art von der Adventures in der Regel auch auf eine Maussteuerung setzen, die so bei mobilen Clients nicht vorhanden ist. Daher empfiehlt sich eine Vogelperspektive auf das Spielfeld, bei der der Spieler seine Figur direkt steuert, um dann Aktionen an einzelnen Objekten vor zu nehmen. Beispiel hierfür könnten die „Legend of Zelda“-Titel auf dem GameBoy sein, die die Spielewelt auf diese Weise darstellen.

3.4.1.2 Action-Spiele

Action-Spiele („Action-Games“, S.9) werden in der heutigen Zeit überwiegend von First-Person-Shootern (FPS), wie „Quake“ oder „Unreal“ dominiert. Früher, d.h. in den 80er und 90er Jahren, war dieses Genre von einer Vielzahl unterschiedlicher Titel und Subgenre geprägt. Allein die Vielfalt der Bezeichnungen gibt einen groben Eindruck: Jump'n'Run (Bsp.: „The Great Giana Sisters“/Time Warp), Jump'n'Shoot (Bsp.: „Turrican“/Factor 5), Hack'n'Slay (Bsp.: „Barbarian II“/Palace Software), um nur ein paar zu nennen.

Sicherlich sind FPS nur schwer auf mobilen Clients zu realisieren, auch wenn es schon Versionen von „Doom“ für PocketPC, Symbian OS und GameBoy Advance gibt. Zentrale Frage ist, ob die Atmosphäre, die mit einem großen Bildschirm auf PC und Konsole mit einer 3D-Darstellung erzielt wird, sich so auf einen kleinen Screen übertragen lässt. Vernünftiger scheint es, hier bewährte Umsetzungsprinzipien aus den 80ern und 90ern zu verwenden, wie sie bei den bereits erwähnten Titeln zu finden sind. Statt einer 3D-Perspektive würde dann

eine 2D-Ansicht des Spielgeschehens geboten werden, mit der auch die Prozessoren der mobilen Clients eher klar kommen würden.

3.4.1.3 Rollenspiele

Rollenspiele („Role-Playing Games“, S.10) seien den Adventure-Spielen ähnlich, würden aber viel mehr Wert auf die Entwicklung der gesteuerten Charaktere legen. Diese würden im Verlauf des Spieles neue Fähigkeiten und Eigenschaften erlangen und seien mit einer Vielzahl von Gegenständen ausrüstbar. Computerrollenspiele stammen in ihrer Grundform von Brettspielen ab. In der Regel steuert der Spieler auch eine ganze Gruppe von Abenteurern durch Kerker, Verliesse und unwegsames Terrain. Klassiker dieses Genres sind „Bard's Tale“/Electronic Arts oder „Ultima“[25] . Als Prominentester Vertreter der heutigen Rollenspiele ist wohl „Final Fantasy“/SquareSoft zu nennen, das sogar mit einem eigenen Film in die Kinos kam.

Auf Grund der eingeschränkten Möglichkeiten mobiler Clients, sind hier vor allem Klassiker wie „Bard's Tale“ als Lösungsmöglichkeit zu sehen, die einen rundenbasierten Ansatz des Rollenspiels implementieren. Die Weise, auf die z.B. die schon erwähnte „Legend of Zelda“-Serie von Nintendo auf dem GameBoy implementiert ist, kann ebenso als Lösungsansatz dienen – also mit einer Echtzeitsteuerung, aber mit einer einfachen 2D-Darstellung aus der Vogelperspektive. Problematisch erscheint dieses Genre aber aus dem Kontext heraus, einfache und leicht zugängliche Unterhaltung zu bieten, was sicherlich nur schwer möglich ist. Kurze Einstiegs- und Austiegszeiten sind ebenso schwer zu realisieren.

3.4.1.4 Strategie-Spiele

Eine weitere Gruppe bilden die Strategie-Spiele („Strategy-Games“, S.11). Hier gelte es, durch taktisches Geschick, die eigene Partei in eine Vorteilhafte Position zu manövrieren um gewinnen zu können. Beispiel sei z.B. die „Command & Conquer“-Serie von Electronic Arts. Diese sei ein Vertreter der Echtzeit-Strategie-Spiele, d.h. Aktionen und Handlungen fänden unmittelbar als Reaktion auf die Steuerung des Benutzers statt. Früher wären Strategie-Spiele rundenbasiert

25 damals von Origin, heute wird die Serie von Electronic Arts weitergeführt

gewesen, d.h. der Spieler hätte die Befehle für alle zu steuernden Elemente gegeben und erst dann wären diese in einem Zug ausgeführt worden, anschließend wären andere Mitspieler oder Gegner zum Zug gekommen.

Aus dem Strategie-Genre ist am ehesten für die rundenbasierte Variante eine Umsetzung auf mobile Clients möglich. Ähnlich wie bei Rollenspielen ist hier eine kurze Ein- und Austiegszeit nur schwer möglich, wodurch dieses Genre nur sehr bedingt für Umsetzungen auf mobile Clients geeignet ist.

3.4.1.5 Simulationen

Simulationen („Simulations“, S.12) versuchten, wie der Name schon sagt, Bedingungen aus der Realität nachzuempfinden, um ein möglichst echtes Abbild zu erzeugen. Beispiele hierfür seien klassischerweise Flugsimulatoren (Bsp. „Flightsimulator“/Microsoft) oder Simulationen für Militärfahrzeuge.

Abgesehen von Action-Simulationen, wie z.B. „AeroDancing“/CRI, sind Simulationen daher prinzipbedingt sehr komplex und somit auch nur sehr bedingt für mobile Clients geeignet.

3.4.1.6 Sport-Spiele

Sport-Spiele („Sports Games“, S.12) widmen sich, wenig verwunderlich, dem Sport. Die Spielefirma Epyx war in den späten 80ern mit zahlreichen Titeln in diesem Genre vertreten, mittlerweile gibt es von jedem größeren Hersteller Spiele zu den verschiedensten Sportarten. Sehr beliebt sind beispielsweise neben Fussball („Fifa“-Serie von Electronic Arts) und Golf auch Exoten wie Fischen („Bass Fishing“/Sega). Interessanterweise führt [Bates01] auch Management-Simulationen, die Sport lediglich als Thema haben, als Sportspiele auf.

Prinzipiell sind Sportspiele für mobile Clients als geeignet anzusehen. Sie ermöglichen einen schnellen Ein- und Ausstieg, lassen sich in der Regel leicht erlernen und bieten durch den prinzipbedingten Wettbewerb eine gute Augangsbasis für Multispieler-Spiele.

3.4.1.7 Prügelspiele

So genannte Prügelspiele („Fighting Games“, S.13) seien meist Duelle zwischen zwei Charakteren, die sich mit verschiedenen Schlägen und Schlagkombinationen traktierten, bis einer der Gegner am Boden liege.

Sie sind zwar in der Regel leicht zugänglich, aber in der Regel auf ein Zwei-Spieler-simultan-Spiel eingeschränkt. Interessant könnten solche Einer-gegen-Einen-Spiele dann werden, wenn durch Community-Funktionen eine Integration mehrerer Spieler, z.B. durch eine Rangliste stattfinden würde. Dadurch würde der Wettbewerb und das Spiel auf eine Vielzahl von Spielern ausgeweitet werden.

3.4.1.8 Casual Games

„Casual Games“ (S.13) sind solche, die für das Spiel „zwischen durch“ geeignet sind und nur eine sehr kurze Einarbeitungszeit benötigen, weil sie sehr einfach zu steuern oder Umsetzungen bekannter Gesellschaftspiele sind. „Monopoly“ oder „Solitaire“, aber auch „Pulleralarm“ oder „Die Jagd auf das Moorhuhn“ werden üblicherweise abweichend von [Bates01] als Casual Spiele gehandelt.

Damit sind sie auch für mobiles und drahtloses Spielen geeignet. Lediglich interessante Netzwerkfunktionen sind ihnen nicht unbedingt innewohnend, hier müsste durch eine Community-Funktion ein entsprechender Anreiz für mehrere Spieler geschaffen werden.

3.4.1.9 God Games

In so genannten „God Games“ (S.13) steuere der Spieler, wie ein übernatürliches Wesen, das Geschick seiner Untertanen. Er gebe teilweise ganzen kleinen Völkern Befehle (z.B. bei „Creatures“) und kontrolliere diese oder schaue einfach nur bei deren Entwicklung zu („The Sims“/Electronic Arts). Abweichend von Bates werden oft auch Strategie-Titel wie „Populous“/EA als God Games bezeichnet, tatsächlich schlüpft der Spieler auch als Stratege in die Rolle eines Gottes, um fortan das Schicksal seines Volkes zu steuern.

God Games, in denen der Spieler das Geschick einer oder mehrerer Spielfiguren eher indirekt steuert, sind sicherlich eine interessante Spiel-Möglichkeit auf mobilen Endgeräten. Eine ständige Interaktion mit dem Spiel wäre nicht notwendig. Es könnte im Hintergrund weiterlaufen oder ruhen, und die Integration mehrer Spieler in einer Welt wäre möglich, ähnlich wie das in einfacher Form beim SMS-Spiel „D2-Zoink“ von wearix stattgefunden hat. Hinderlich erscheint jedoch der Punkt, dass Interaktivität im Vergleich zu anderen Genres tendenziell zurückfällt. Dies wirkt umso schwerer, als dass [Rollings01], Interaktivität als den zentralen Bestandteil eines Computerspiels sieht, wie in *3.3* beschrieben.

3.4.1.10 Lernspiele

„Educational Games“ (S.14), also Lernspiele, widmeten sich, wie der Name schon sagt, dem Näherbringen von lehrreichen und (hoffentlich) pädagogisch wertvollen Inhalten. Sie seien meist für Kinder ausgelegt und werden oft auch als „Edutainment“-Titel bezeichnet. Inwieweit Lernspiele auf mobilen Clients Sinn machen, ist fraglich. Schließlich setzt Lernen allgemein eine gewisse Konzentration voraus, die wohl nur selten beim mobilen Einsatz gegeben ist.

3.4.1.11 Puzzlespiele

Puzzlespiele („Puzzle-Games“, S.14) hätten als zentrales Spielelement das Lösen von Rätseln oder Denkaufgaben – oft auch mit kleinen Geschicklichkeitsübungen verbunden, wie z.B. bei „Tetris“/Nintendo, „Klax“/Atari oder „Puzznic“/Ocean.

Puzzlespiele eignen sich auf Grund ihrer Einfachheit sehr gut für den mobilen Einsatz. Sie sind geradezu Ideal dafür geeignet, da ihre Spielprinzipien und ihre Steuerung meist leicht zu erlernen sind und damit den Anforderungen aus *3.1-3.5* entsprechen.

3.4.1.12 Online Spiele

„Online-Games“ seien laut [Bates01], S.15 Spiele, die die vorangegangenen Spieletypen integrieren würden, aber überwiegend über das Internet oder über ein

Netzwerk gespielt werden würden. Prominente Beispiele seien laut Bates „Everquest“ und „Ultima Online“.

Vom Prinzip her stellen Online-Spiele sicher einen für mobile Clients interessantes Genre dar. Gerade die Netzwerk-Fähigkeit wird durch sie erst optimal genutzt und bietet so einen Anreiz für die Nutzung auf mobilen Clients. Doch beschreibt damit ein Online-Spiel eher das Verwenden einer Netzwerktechnik als den Spielinhalt, von dem die Eignung für den mobilen Einsatz aber letztlich überwiegend abhängt.

3.4.2 Nutzergruppen

Folgende Hypothese liegt den kommenden Überlegungen zu Grunde. Handynutzer in ihrer Breiten Masse sind keine so genannten „Hardcore-Gamer“. Unter „Hardcore-Gamer“ versteht man mit mehr oder weniger großen Abweichung in der Spieleindustrie den Nutzertypus, der die neuesten und spektakulärsten Spiele mit den oppulentesten Grafiken massiv nutzt (meist 3D-Ego-Shooter, FPS) und dabei spezielle Spielehardware einsetzt – also z.B. bei PCs spezielle 3D-Grafikkarten[26].

Vielmehr scheint es wahscheinlicher, dass es sich bei der zukünftigen breiten Basis der Handy- bzw. Smartphone- oder PDA-Spielern um so genannte Casual- und Social-Gamer handelt.

Als Casual-Gamer werden in der Regel solche Spieler bezeichnet, welche einfache Inhalte den komplexen vorziehen und einen einfachen Einstieg in das Spiel wünschen. Social-Gamer sind hingegen solche Spieler, die das Spielvergnügen in der Gruppe suchen.

Hier wird der Zusammenhang zur Handy-Nutzung als Spieleplattform allmählich deutlich. Mobile Clients sind im Gegensatz zu einem GameBoy oder einer anderen Handheld-Spielekonsole kein dediziertes Spielegerät, sondern ein Pro-

26 Diese Annahme wird in Entwicklerkreisen geteilt, wie [Puha01] zeigt.

dukt mit Allround-Fähigkeiten. Diese und die Konnektivitätsfunktion, lassen die beiden letztgenannten Nutzerkategorien als sehr wahrscheinlich erscheinen.

Die Vermutung liegt also Nahe, dass sich jemand der speziell Computerspiele bevorzugt sich zunächst eine Handheldkonsole zulegt. Inwieweit zukünftige mobile Clients und dafür programmierte Spiele daran etwas ändern können bleibt offen. Für einige Spielegenres wie Strategiespiele ist der PC die erste Wahl. Vielleicht wird es bei den Handyspielen ähnliche Entwicklungen geben, wie Community-Spiele, die einerseits stark die schon erwähnten Social-Gamer ansprechen, andererseits die den mobilen Clients innewohnenden Kommunikationsfähigkeiten voraussetzen würden. Doch auch für den GameBoy Advance ist ein erster Bluetooth-Adapter angekündigt und auch für den Vorgänger, den GameBoy Color, gab es einen Handy-Adapter zumindest in Japan, sodass die zukünfitge Entwicklung in dieser Hinsicht sicherlich spannend bleibt.

3.4.3 Welche Spiele für welche Nutzer

Die obigen Betrachtungen haben einerseits mögliche Spielgenre vorgestellt, andererseits die wahrscheinlichen Nutzer aufgezeigt. Die Bedingungen für Mobile und Wireless Entertainment aus *3.1* und *3.2* dienen als Filter für die Auswahl der geeigneten Genre. Es lässt sich folgern, dass komplexe Spielgenres wie Strategiespiele und Simulationen nur bedingt für mobile Geräte geeignet sind, insbesondere wenn die notwendigen Sitzungen nicht in ausreichend kurzer Zeit vollzogen werden können, bzw. die Spiele in zeitlicher Hinsicht nicht flexibel genug handhabbar sind. Dies trifft leider bei den meisten Strategie-Titeln zu, wie etwa bei „Command & Conquer“ oder „Heroes of Might and Magic“.

Ein simples Abspeichern und Laden, also ein Persistenz-Management, der Spieldaten scheint ungenügend, da sich der Spieler bei erneutem Eintauchen in das Spiel wieder in die komplexen Gegebenheiten des Spiels einfinden muss. Das könnte den Nutzer leicht überfordern und damit das Spiel unspielbar machen.

Es ist also notwendig, komplexe Genre wie etwa Strategiespiele, Rollenspiele oder Simulationen so zu vereinfachen, dass sie auf den mobilen Clients spielbar

werden. Ansätze hierfür können etwa Action-Simulationen oder rundenbasierte Strategiespiele sein, die die Spielsitzungen in kleinere Einheiten verpacken[27].

Einfache, leicht zugängliche Kategorien sind für mobile Clients besser geeignet. Hier sind nicht nur Actionspiele gemeint. Puzzlespiele sind auch für Leute geeignet, die nicht nur ihre Geschicklichkeit, sondern auch ihr Denkvermögen unter Beweis stellen möchten. Ein Nutzertypus also, der prinzipiell ein denkorientiertes Genre, wie das der Strategiespiele, Actionspielen vorziehen würde. Damit wären auch Spieler angesprochen, die ihre Kurzweil nicht in Geschicklichkeitstest suchen würden.

Rollenspiele würden riesige Welten mit mehreren Spielern ermöglichen (Beispiel hierfür wären auf PC-Seite „Ultima Online“ oder „Everquest“), Sportspiele wären ebenso sehr gut für Mehrspieler-Titel geeignet. Interessant sind sicherlich auch Wirtschaftsimulationen, die ebenfalls eine einfache Multiplayer-Integration ermöglichen könnten, wie z.B. eine Börsen-Broker-Simulation, in der derjenige gewinnt, der am Besten „performt“.

Wichtig ist bei Mehrspieler-Titeln jedoch, dass sich die Anzahl der Spieler auf ein vernünftiges Maß begrenzt. So sollten Spiele durch zu viele Teilnehmer nicht unübersichtlich oder zu komplex werden. Neben simplen Begrenzungen der Spieleranzahl pro Welt könnten auch Mechanismen und Spielinhalte zum Einsatz kommen, die die Spielwelt in Parallelwelten aufteilen würden. Dabei sollte dann eine Kommunikation der Spieler untereinander über die Grenzen der einzelnen Parallelwelten hinweg möglich sein, damit eine Verknüpfung der Geschehnisse in den verschiedenen Welten möglich wird. Idealerweise ist die Spieleranzahl also nicht nur eine technische Begrenzung, sondern hat auch noch einen inhaltlichen Hintergrund in der Spielgeschichte.

So werden neben einer Renaissance der klassichen Video- und Computerspiele durch die Konnektivität und Mobilität auch völlig neue Ausprägungen und Formen der bisherigen Spielkategorien möglich. Bekannte Spielprinzipien und

27 s.a. *3.7 Umsetzungsmöglichkeiten aktueller PC und Konsolentitel*

Modelle werden dadurch um eine neue Komponente bereichert und interessanter gemacht.

3.5 Gestalterische Überlegungen zu Grafik und Ton von mobilen Computerspielen

Die gestalterischen Möglichkeiten bei der Erstellung von Computerspielen auf mobilen drahtlosen Clients sind gegenüber denen auf PC und Heimkonsole stark eingeschränkt. Augenscheinlich fallen hier sofort die kleinen *Screens* und die unergonomischen Steuerungsmöglichkeiten auf.

3.5.1 Bildgestaltung

Vor allem der kleine Bildschirm und dessen meist geringe Auflösung machen ein Gestalten der graphischen Darstellung eines Spieles sehr schwer. Orientierung können hier wieder einmal die Klassiker der Computerspiele aus den 80ern wie „Pac Man"/Namco oder „Dig Dug"/Namco bieten. Jemand der diese Spiele gespielt hat, wird sicherlich nicht die einprägsamen Charaktere und typischen Grafiken vergessen – und das trotz der eingeschränkten Darstellungsmöglichkeiten auf den damaligen Endgeräten.

Der Schlüssel der grafischen Umsetzung liegt hier nicht in einer möglichst photorealistischen, sondern in einer stilisierten und typisierten Darstellung, ähnlich wie sie bei Comics und Cartoons stattfindet. Das Werkzeug des Grafikers sind im Gegensatz zur Comicgrafik jedoch nicht vornehmlich Linien und Flächen, sondern die Zusammensetzung der Grafik aus einer blockartigen Struktur, die sich durch die geringe Auflösung ergibt. Bei der Erstellung muss der Grafiker die Bilder also von Hand rastern. Die Herausforderung liegt im Finden eines geeigneten Stiles, um den gewünschten gestalterischen Ausdruck zu erreichen. Diese Methode unterscheidet sich signifikant von der Arbeitsweise, wie sie heute sonst bei Computerspielen üblich ist[28].

28 In der Regel besteht ein aktuelles Spiel nur aus 3D-Grafiken. Selbst bei 2D-Spielen werden die Grafiken in der Regel mit 3D-Tools erstellt und per Raytracing oder ähnlichen Verfahren vorberechnet.

Probleme ergeben sich bei der Gestaltung von niedrig auflösenden Grafiken vor allem dann, wenn eigentlich eine nicht-stilisierte Darstellung erwünscht ist. Die stilisierte Darstellung erzeugt eine gewisse Distanz zum Betrachter, da Emotionen nur noch elementar und stereotyp ausgedrückt werden können. Hier ergibt sich leider einfach ein Nachteil gegenüber hoch auflösenden Grafiken, der sich nur durch gekonnte Gestaltung umgehen lässt.

Im Bezug auf die *Multiplayer*-Fähigkeit lässt sich folgende Annahme machen: Bedingt dadurch, dass es keine Handies mit Steuerungsmöglichkeiten für zwei Spieler an einem Gerät gibt, machen Techniken wie *Splitscreens* keinen Sinn[29]. Dadurch entsteht für die Spieleimplementierung eine Vereinfachung. Grafiker müssen sich nicht um eine eventuelle Aufteilung der Anzeige Gedanken machen.

Bezüglich der Größe der Spielfläche gibt es eine gewichtige Problematik. Die unterschiedlichen Auflösungen der Endgeräte machen es notwendig, dass die Größe der dargestellten Spielfläche möglichst flexibel ist. Dies erscheint umso notwendiger, da eine Skalierung der Grafiken zumindest auf Clientseite aus Performanzgründen nicht möglich ist, andererseits bei diesen geringen Auflösungen die Grafik zerstören würde. Man stelle sich vor, ein Pac-Man wird herunterskaliert und verliert sein Auge – keine Unwahrscheinlichkeit. Es ist also notwendig ein so genanntes *Scrolling*, d.h. ein Bewegen des sichtbaren Bildausschnitts, zu berücksichtigen und zur Verfügung zu stellen, um das Spielfeld nicht auf die Größe des kleinsten Displays reduzieren zu müssen. Damit wird gewährleistet, dass das Spiel auch auf kleinen Geräten lauffähig ist. Grafiker und Spieldesigner müssen dabei berücksichtigen, dass Grafiken in unterschiedlichen Größenverhältnissen zur Bildschirmgröße stehen und mitunter nicht sichtbar sind.

3.5.2 Tongestaltung

Ähnlich wie bei den Grafiken verhält es sich bei den Soundmöglichkeiten der mobilen Endgeräte. Der GameBoy bietet immerhin so etwas wie einen kleinen FM-Synthesizer und einfache digitale Klangwiedergabe mit 4 Bit Auflösung. Die Java-APIs von Nokia und Siemens machen higegen nur einfache einstimmige

29 Splitscreens teilen den Bildschirm für mehrere Spieler in unabhängige Ansichten, um so ein *Multiplay* zu ermöglichen

Tonwiedergabe möglich. Die Kunst der Komponisten/Tontechnikers ist es, aus diesem kleinen Set an Ausdrucksmöglichkeiten, tatsächlich eine Klangatmosphäre zu bilden und sie mit der grafischen Darstellung in Einklang zu bringen. Dass dies möglich ist, belegen wiederum einmal die Hits der Computerspielgeschichte, die ebenso mit kargen Möglichkeiten auskommen mussten, wie z.B. auf dem Atari VCS.

Problematisch ist die Fähigkeiten eines kleinen mobilen Systems zu sehen, den Spieler allein durch die audiovisuelle Ausgabe zu begeistern. Während PCs und Heimkonsolen aufwändige 3D-Welten mit großen Bildschirmen, sozusagen um den Betrachter erzeugen und dazu mit atmosphärischen Sounds unterlegen können, müssen die mobilen Endgeräte bisher prinzipiell mit kleinen Screens, 2D-Darstellungen und piepsigen Tönen aus kleinen Lautsprechern auskommen[30].

Es ist also unwahrscheinlich, dass ein Titel auf einem mobilen Endgerät durch seine audiovisuelle Darstellung eine ähnliche dichte Athomsphäre erzeugen kann, wie auf einem PC oder einer Heimkonsole. Daher ist es umso wichtiger eine packende Story mit dem Spiel zu liefern.

Mit den vorgestellten Lösungen zur Gestaltung wären wir zusammen mit den relevanten Spielkategorien aus *3.4* prinzipiell bei einer Erfüllung der Anforderungen angelangt, wie sie in [Fritz97], [Rollings00] und *3.1* behandelt werden.

3.6 Marktübersicht der aktuellen mobilen drahtlosen Computerspiele

Um beurteilen zu können, welche Spiele in der Praxis auf mobilen Clients möglich sind, sollen hier stichprobenartig einige Spiele vorgestellt werden, um die bisherigen theoretischen Überlegungen und Behauptungen verifizieren und belegen zu können.

30 PocketPCs bieten prinzipiell die Möglichkeit einer Sounwiedergabe in CD-Qualität über Kopfhörer, aber dies ist doch eher eine Ausnahme im Vergleich zu den restlichen Vertretern dieser Produktkategorie, und das Problem des kleinen Screens bleibt weiterhin bestehen.

Dazu werden neben der Betrachtung der aktuellen Titel in Europa, stellvertretend dafür Deutschland, auch Computerspiele aus Japan vorgestellt.

3.6.1 Aktuelle Titel in Europa

Am einfachsten lassen sich die bisher beschriebenen Theorien zur notwendigen Einfachheit und schnellem Einstieg ins Spiel dadurch belegen, dass die auf Handies vor-installierten Spiele, wie z.B. „Snake“ von Nokia und „Balloon Shooter“ von Siemens, diesen Anforderungen entsprechen. Bei „Snake“ etwa geht es darum, eine Schlange auf einem Spielfeld zu steuern und kleine Dinge aufzufressen, wodurch die Schlange wächst und schwieriger zu steuern wird. Dabei darf der Spieler nicht mit der Wand oder dem eigenen Körper kollidieren. Bei „Balloon Shooter“ geht es darum, nach oben fliegende Ballons abzuschiessen. Je mehr Ballons er erwischt, desto mehr Punkte erhält er. Verloren hat der Spieler dann, wenn er eine bestimmte Anzahl von Ballons verpasst hat.

Diese einfachen Spiele belegen zwar die These zur Einfachheit und leichten Zugänglichkeit, jedoch nur bedingt andere bisher erarbeitete Grundlagen. So wird keine Welt kreiiert die den Spieler richtig fesselt, deren Spielfiguren Charakter haben (*3.3.2 Theorien aus dem pädagogischen Umfeld*) und deren Spielverlauf dramaturgische Elemente enthält (*3.3.3 Theorien aus der Welt der Spieleentwickler*). Die Anforderungen, die eine gewisse Komplexität bei der Konzeption und Umsetzung voraussetzen, werden also nicht erfüllt.

Abgesehen von „D2 Load-A-Game“-Titeln, die jedoch nur auf bestimmten Handies lauffähig sind, dominieren momentan SMS- und WAP-Spiele den Markt für ladbare Spiele. Sofern man überhaupt von dominieren sprechen kann, vielmehr ist es so, dass zwar viele Titel auf den verschiedenen Portalen der Mobilfunnkanbieter verfügbar sind, jedoch ist der Markt an sich zumindest so klein, dass sich Statistiken des VUD noch nicht darum kümmern. Ebenso zeigen Diskussionen in einschlägigen Foren[31], dass der Markt an sich auch von Seiten der Wertschöpfungskette und des Business-Models noch sehr undefiniert ist.

31 www.radio-gamer.com

Die Qualität der Spiele ist sehr unterschiedlich. Das SMS-Spiel „fishsnapper“ von 12snap.com bietet z.B. weitreichende Community-Funktionen wie Chat und Tausch von Spielgegenständen. Jedoch gestaltet sich das eigentliche Spiel sehr einfach. Der Spieler hat die Aufgabe Fische zu fangen. Dazu wählt er die passende Ausrüstung und einen geeigneten Ort per SMS. Der Rest besteht aus Warten, bis ein Fisch gefangen wurde. Einzige Spielmotivation besteht im Sammeln der Fische und der Community-Funktion.
Andere SMS-Spiele, wie die von Handy-Games.com, erscheinen hier schon spielerisch interessanter. Beispiele hierfür sind „battleship“, eine Art „Schiffe-Versenken“, und „tanks“, ein Panzerspiel, bei dem man unter Berücksichtigung von Wind und Position des Gegners, diesen zu treffen versucht. Bei diesen Spielen kann man trotz der einfachen Darstellung von einer gewissen Spieltiefe sprechen. Es sind genug Optionen vorhanden, um notwendige Entscheidungen in der Spielwelt treffen zu können, dies wiegt umso stärker, als dass das Medium SMS nicht gerade durch grafische Brilianz besticht.

Problematik aller Spiele per SMS ist jedoch die Darstellung per Buchstaben und Sonderzeichen, was im Vergleich zum heutigen Stand der Technik, doch sehr einfach ist und an die ersten Computerspiele auf Textterminals von Großrechnern erinnert.

Prinzipiell unterscheiden sich WAP-Spiele von ihren SMS-Pendants in der grafischen Darstellung. Durch die eingeschränkten Möglichkeiten der WAP-Technologie sind hier auch nur rundenbasierte Spiele möglich. Bei WAP-Spielen kommen jedoch bisher als großer Hinderungsgrund für einen breiten Markterfolg die hohen Zugangskosten und die langsame Datenübertragung hinzu, was hoffentlich durch Technologien wie GPRS und UMTS behoben wird.

Trotzdem sind ein paar interessante Titel verfügbar, wie etwa „Chop Suey Kung Fu“/nGame, das die Idee des Stein-Schere-Papier-Spiels auf ein Prügelspiel mit comicartiger Grafik überträgt. Dieses Spiel zeigt, was aus einer relativ simplen Technologie an spielerischem Wert herauszuholen ist und wurde auf Grund seines Erfolgs auf Gamasutra[32] in [Kelland01], stellvertretend für die neue Generation

32 www.gamasutra.com

der Wireless-Spiele, vorgestellt. Die Charaktere sind einprägsam und mit einem hohen Wiedererkennungswert verbunden, vor allem im Vergleich mit den sonstigen Handyspielen. Das Spiel wegen des Stein-Papier-Schere-Prinzips einfach zu erlernen, jedoch fehlt es etwas an der Spieltiefe, d.h. an den zur Verfügung stehenden Möglichkeiten.

3.6.2 Aktuelle Titel in Japan

Die in Japan erhältlichen Spiele sind eine Entwicklungsstufe weiter. Ob Segas[33] „Space Harrier“ oder ein kleine Sportspiele von Hudson[34], diese Spiele bestechen im Vergleich mit ihren europäischen Pendants durch bessere Grafik und größere Interaktionsmöglichkeiten. Dadurch lassen sich die Thesen zur Inhaltsgestaltung für mobile drahtlose Spiele besser stützen, die eine einprägsame und prägnante Gestaltung fordern, die SMS oder WAP-Spiele in der Regel nicht erreichen.

Die vorgestellten Beispiele zeigen, dass grundsätzlich Spiele mit den verfügbaren technischen Mitteln möglich sind. In ihrer Gesamtheit entsprechen sie den bisherigen Thesen zu einfachem Gameplay, eigenen einprägsamen Spielwelten usw., wie sie in *3.1 -3.5* beschrieben sind. Spannend bleibt, ob sich SMS-Spiele mit MMS weiter entwickeln können oder ob mit der Weiterentwicklungen von WAP und vor allem Java, SMS-Spiele mittelfristig verdrängt werden.

3.7 Umsetzungsmöglichkeiten aktueller PC und Konsolentitel

Portierungen bekannter Spieletitel sind bei Entwicklern wie Endkunden beliebt, da sie bekannte Titel auch auf anderen Plattformen verfügbar machen. Umsetzungen von PC und Heimkonsolen auf mobile Clients scheinen nur schwer möglich. Neben dieser Problematik ist die Untersuchung aber noch aus einem anderen Grund notwendig. Die folgende Analyse soll zeigen, wie mit den technischen Einschränkungen in konkreten Fällen umgegangen werden soll. Dadurch werden

33 www.sega.co.jp

34 www.hudson.co.jp

spieltechnische Grundlagen geliefert, die nicht nur für die Umsetzung von Spielen von anderen Plattformen, sondern auch für die Neuerstellung von Spielen hilfreich sind.

Der Schlüssel liegt hier nicht in der originalgetreuen Übertragung und Portierung des Titels auf den mobilen Client, sondern in einer technischen und spielerischen Anpassung. Dabei lässt sich durchaus auf bisherige Erfahrungen und Entwicklungen der Computerspielgeschichte zurückgreifen. Es werden bewusst auch schwierige Beispiele heran gezogen, um an diesen eine Vielzahl von Lösungsmöglichkeiten beschreiben zu können. Darüber hinaus sind die gewählten Beispiele populäre Vertreter ihres Genres.

3.7.1 Rennspiele

Schauen wir uns zunächst einmal das Spielgenre der Rennspiele an. Klassischerweise zwischen Sportspiel und Simulation angesiedelt, enthält es überwiegend Geschicklichkeits- und Reaktionstests.

Ein Vertreter dieser Kategorie ist „Midnight Street Racing“ von Bizzare Creations. Aus einer 3D-Perspektive verfolgt und steuert der Benutzer das Spiel aus einer Verfolgerperspektive. Spieltechnisch geht es darum, in Rennen gegen andere Fahrer und gegen die Zeit zu bestehen.

Spieltechnisch wäre hier eine Umsetzung nach dem Vorbild von „Pole Position“/Atari oder „Super Sprint“/Atari denkbar. „Pole Position“ setzt den Fahrer in eine Pseudo-3D-Perspektive hinter das Fahrzeug, der Rennspielspaß bleibt trotz fehlendem echten 3D erhalten. Super Sprint wählt dagegen eine Vogelperspektive, aus der der Spieler das Geschehen verfolgt. Extras wie Turbo oder ein Reperaturschlüssel rücken das Spiel dabei von der Simulation zum Actionspiel hin.

Für Multiplayerspiele wäre dieser Ansatz sehr gut geeignet, da Rennspiele von Natur aus für mehrere Spieler ausgelegt sind. Eine Vereinfachung der Perspektive auf 2D käme auch sicher der Reduktion der Komplexität des Spieles entgegen und

würde das Spiel damit eindeutig in den Bereich der Anforderungen bringen, wie sie von den bisherigen Thesen (*3.1-3.5*) an ein drahtloses Spiel gestellt werden.

3.7.2 First Person Shooter

Eines der wichtigsten Genre auf dem PC ist gegenwärtig der so genannte First-Person-Shooter (FPS), wie die zahlreichen Titel und Fortsetzungen, wie z.B. die „Quake"-, „Doom"-, „Wolfenstein" und „Unreal"-Serien, belegen. Der Spieler kämpft sich hier meist durch Kerker, Höhlen und Verliese und tötet dabei, was sich ihm in den Weg stellt. Als 1:1-Umsetzung für ein Handy nahezu unmöglich[35], ist eine Übertragung des primitiven Spielprinzips „Schiesse und Laufe" durchaus als einfache 2D-Variante denkbar. Schiessspiele und Klassiker wie „Galaga"/Namco, „R-Type"/Irem oder „Metal Slug"/NeoGeo belegen die Realisationsmöglichkeit.

Ob dieses Genre jedoch massentauglich und damit für den Handy-Markt geeignet ist, bleibt anzuzweifeln. Zumindest beim GameBoy tummeln sich in den Charts meist „Pokémon" und Co. Auch wenn Handys und Organizer auf eine ältere Zielgruppe abzielen, so ist zumindest der First-Person-Shooter eher ein Genre der Hardcore-Gamer. Hinzu kommt, dass durch die fehlende 3D-Perspektive ein Großteil der genrespezifischen Atmosphäre verloren geht. Somit ist eine direkte Umsetzung nur bedingt denkbar, vielmehr müssten auch Anpassungen spielerischer Natur gemacht werden. Etwa in der Form, dass Gewalt nicht mehr so sehr im Vordergrund steht und eine freundlichere, weniger abschreckende Atmosphäre geschaffen wird.

3.7.3 Strategiespiele

Ein weiteres wichtiges Genre auf dem PC sind Echtzeitstrategie-Spiele wie „Command & Conquer" anzusehen. Als Ziel haben diese Spiele üblicherweise, einen Feind mit Mitteln der strategischen Kriegsführung zu besiegen. Das heißt, verschiedene Einheiten müssen gebaut und gegen den Feind in einen Kampf entsendet werden. Alle Aktionen finden unmittelbar nach der Benutzereingabe statt, deshalb auch der Name Echtzeitstrategie. Eine Umsetzung ist aus technischen

35 auch wenn DOOM inzwischen für GameBoy Advance erhältlich ist

Gründen nur sehr eingeschränkt möglich. Bei heutigen Echtzeitstrategie-Spielen tummeln sich hunderte von Einheiten auf dem Screen, und noch mehr Objekte in der gesamte Spielewelt. Mit diesen Berechnungen wären die relativ kleinen CPUs der Handhelds überlastet. Hinzu kommt, dass durch die kleinen Bildschirme die großen Spielewelten sehr unübersichtlich und ein Scrolling die Hektik beim Steuern der Einheiten vergrößern würde.

Doch auch hier kommt einem die Entwicklungsgeschichte des Spielgenres entgegen. Rundenbasierte Strategiespiele waren lange Zeit en vogue und sind auch heute noch üblich und vermarktbar, was z.B. „Civilization“ von Sid Meier belegt. So liessen sich beispielsweise auch Strategie-Klassiker wie „M.U.L.E.“ oder „Kaiser“ auf mobilen Clients realisieren. Da sie rundenbasiert sind, würde die Problematik des Ein- und Ausstiegs auch nicht mehr so schwer wiegen. Letztere Beispiele verpacken zusätzlich interaktive Elemente in kleine Einheiten, wodurch sie auch besser für Spieler geeignet sind, die mit Strategiespielen sonst nicht so viel anfangen können.

Wie die genannten Beispiele zeigen, lässt sich in der Regel auf eine vorhergehende Entwicklungsstufe des Spielgenres zurückgreifen, um ein Spiel in einer Version für mobile Clients zu designen. Dabei unterscheidet sich die Konvertierungsmöglichkeit natürlich von Genre zu Genre. Die zweifellos vorhandenen technischen Hürden sind dabei aber nicht unüberwindbar.

4 Analyse der technischen Möglichkeiten

Da eine grundlegende Betrachtung der technischen Möglichkeiten bereits stattgefunden hat (*3.2 Technische Voraussetzungen*), findet sich im Folgenden eine Analyse der verschiedenen Systemplattformen, die als mobile und drahtlose Spieleclients in Frage kommen. Dabei werden die Systeme und vor allem das Betriebssystem, bzw. die Laufzeitumgebung, dahingehend untersucht, inwieweit sie die Entwicklung und auch das Verwenden von interaktiver Unterhaltungssoftware erleichtern.

4.1 Systemplattformen

Ein Kriterium hierfür ist eine Buttonabstraktion der wichtigsten Steuerungselemente eines Spiels, d.h. die Laufzeitumgebung entscheidet, wohin idealerweise die Steuerknöpfe für das Spiel auf die Tastatur oder ähnliches des Geräts gelegt werden. So kann der Nach-Oben-Knopf auf einem einfachen Handy z.B. auf der „2“ liegen, auf einem anderen Gerät aber auf dem Steuerkreuz, ohne dass der Entwickler sich darum kümmern muss.

Hinzu kommen natürlich die üblichen Kriterien der Software-Entwicklung, d.h. Einfachheit und Handhabbarkeit des Systems, Debugging-Möglichkeiten und Optimierungsmöglichkeiten, d.h. inwieweit kann das Spiel noch schneller und erlebnisreicher oder überhaupt ausreichend schnell lauffähig gemacht werden.

Insbesondere der letzte Teil ist nicht zu unterschätzen, sind die mobilen Clients doch im Vergleich zu Desktop-Computern noch recht schwache Geräte mit relativ kleinen Prozessoren[36]. Natürlich lassen sich MHz unterschiedlicher Architekturen nicht direkt vergleichen, die Größenordnungen, die wir hier vorfinden, geben jedoch Aufschluß über den Leistungsunterschied, mit dem momentan noch zu kämpfen ist.

36 Palm Vx 20Mhz Dragonball, Desktop-PC ca. 1GHz x86

4.1.1 Java 2 ME

Sun bietet für mobile und kleinere Geräte eine ganze Familie an Spezifikationen an (vgl. [Sun01]), die man unter dem Überbegriff Java 2 Micro Edition (Java 2 ME) wiederfindet. Hierin finden sich zunächst Konfigurationen, die die grundlegenden Eigenschaften der Geräte spezifizieren, sowie darauf aufbauend so genannte Profile, welche genauere Details, wie ungefähre oder angenommene Screengröße und genauere Speichervoraussetzungen festlegen oder bestimmte Klassen-Bibliotheken verfügbar machen.

Eine genauere Beschreibung ist z.B. in [Riggs01], [Giguère00] und [Wolf00] enthalten, für diese Diplomarbeit sind aber vor allem die folgenden Eigenschaften interessant. Sun führt unter dem Personal Profile das bisherige Personal Java und damit indirekt Embedded Java weiter. Es setzt auf der Connected Device Configuration (CDC) auf, eine Konfiguration, die neben PDAs auch auf „größere“ Konsumergeräte wie Settopboxen abziehlt. Hier ist bereits ein etablierter Standard vorhanden, da er schon längere Zeit verfügbar ist und genutzt wird. Der zweite wichtige Standard ist das Mobile Information Device Profile (MIDP). Es spezifiziert die Anforderungen und Leistungsmerkmale einer Java-Laufzeitumgebung auf mobilen Endgeräten.

Das MIDP baut auf der Connected Limited Device Configuration (CLDC) auf. Eine Konfiguration, die, wie der Name schon sagt, für kleinere Geräte mit eingeschränkten Systemeigenschaften (wenig Speicher, kleiner Prozessor ohne *FPU*, evtl. nur 16Bit-Architekur) gedacht ist – also auch und vor allem für mobile Endgeräte mit Konnektivität.

Das Personal Profile bietet im Gegensatz zum MIDP mit der CDC eine vollständige Java Virtual Machine (JVM/VM), hier manchmal auch CVM genannt. Die KVM (Kilo Virtual Machine) der CLDC und MIDP hat hingegen folgende Einschränkungen:

- keine Unterstützung für Gleitkommaartihmetik
- keine benutzerdefinierten Class-Loader
- kein Native Interface (Java Native Interface/JNI)
- eigene User-Interface-Klassen, kein AWT oder gar Swing/JFC

kein Finalisieren von Klasseninstanzen
keine Reflection-Funktionalität
keine Thread-Gruppen oder Daemon-Threads
keine weak-Referenzen
eingeschränkte Error- und Exceptionbehandlung, d.h. für nicht spezifierte Errorevents ist das Gerät selbst verantwortlich
nur Http1.1-Verbindungen
das Fehlen von einem Filesystem
kein Sound-Support
allgemein eingeschränkter Klassenumfang

Neben diesen Einschränkungen bietet die MIDP folgende Erweiterungen:

Unterstützung persistenter Speicherung von Daten
Buttonabstraktion/Kommandoabstraktion

Vor allem die Buttonabstraktion ist für Spieletitel, wie eingangs schon erwähnt, wichtig. Auf persistente Speicherung von Daten wird natürlich bei einem fehlenden Filesystem gerne zurückgegriffen.

Die verbleibenden Einschränkungen wiegen bei näherer Betrachtung nicht so sehr schwer. Schließlich sind die meisten fehlenden Mechanismen, wie z.B. Reflection, rechenintensiv und auf kleinen Geräten nur schwer vernünftig einsetzbar. Andere stellen wiederum eher einen Luxus dar, wie z.B. das AWT. Sie machen auf diesen kleinen Geräten mit schwachen Prozessoren und kleinen Displays auch wenig Sinn und können nicht ihren eigentlichen Zweck erfüllen.

Fehlender Sound-Support ist jedoch abgesehen von proprietären Erweiterungen (etwa bei i-mode, Siemens oder Nokia), ein schmerzhaftes Manko, da es das Spieleerlebnis stark einschränkt. Hier wäre es überlegenswert, eine eigene abstrakte Schnittstelle für die verschiedenen Implementierung der Hersteller zu schaffen. Für die nächste MIDP-Version, „Mobile Information Device Next Generation“ [Sun02], ist eine Sound-Unterstützung geplant.
Ein weiteres Manko das in der mobilen Java-Welt oft anzutreffen ist, ist das Fehlen einer Remote-Debugging Funktion auf dem Gerät. Software lässt sich also

nur auf soweit verfügbaren Emulatoren debuggen und tracen, was die Entwicklung und Anpassungen der Software unnötig erschwert.

Das Fehlen einer nativen Schnittstelle, ist differenziert zu betrachten. Zwar gehen hier Optimierungsmöglichkeiten verloren, jedoch sind native Implementierungen auf Grund der Gerätevielfalt kaum sinnvoll.

Mittlerweile finden sich einige mobile Clients mit MIDP-Unterstützug auf dem europäischen Markt. Hierzu gehören der „Nokia Communicator 9210“, das „Siemens SL45i“ und das „Accompli 008“ von Motorola. Für Palm-Handhelds ist sogar eine MIDP-Referenzimplementierung von Sun verfügbar. Vor allem Nokia forciert die Unterstützung von Java mit seiner auf Symbian OS basierenden S60 Plattform, auf der das im zweiten Quartal 2002 erscheinende „Nokia 7650“ basiert.

Auf dem Communicator sind, genauso wie unter PalmOS, auch Personal Profile-konforme Programme lauffähig. Personal Java-Implementierungen finden sich auch auf PocketPCs. Aus momentaner Sicht scheint eine Gewichtung der Relevanz der beiden Profile für Computerspiele schwer. Noch sind zu wenig Produkte verfügbar, um eine eindeutige Aussage treffen zu können. Handies sind nur wenige mit MIDP erhältlich, aber für den Entertainment-Markt interessanter, weil günstiger und zielgruppenrelevanter, da einfach zu handhaben und eindeutiger als ein Consumerprodukt einzustufen als ein Organizer, der mehr Geschäftskunden anspricht und auch so beworben wird (zumindest von Nokia). In Japan gibt es, innerhalb der i-mode Services, einen MIDP sehr ähnlichen Dialekt, der schon von namhaften Herstellern wie Sega oder Hudson unterstützt wird. So sind in Japan auch schon einige interessante Spiele aus dem Action- und Puzzlegenre verfügbar, wie schon in *3.6 Marktübersicht der aktuellen mobilen drahtlosen Computerspiele* gezeigt wurde.

Tendenziell stehen für den Unterhaltungssektor die Zeichen eher für MIDP als für das Personal Profile, wobei diese Aussage mit Vorsicht zu genießen ist, zumal Microsoft mit Windows CE, Pocket PC und Stinger wohl als eigentlicher Konkurrent von MIDP und Java allgemein zu sehen ist.

4.1.2 Windows CE

Der „Compaq iPaq“ verhalf dem lange vor sich hindümpelnden PocketPC-Betriebssystem[37] zum Durchbruch. Mittlerweile ist „PocketPC 2002“ verfügbar, zusammen mit einer Vielzahl von Geräten zahlreicher Herstellter. Microsoft setzt mittlerweile bei der Hardwareplattform voll auf die ARM-Architektur, die wir auf Grund ihrer Bedeutung später noch etwas detailierter betrachten werden.

PocketPC bietet fast alle Vorzüge, die MIDP nicht bieten kann: direkter und damit schneller Framebuffer-Zugriff, Remote-Debugging und nativer Zugriff auf die Hardware. Alles Punkte, die für die Spieleentwicklung durchaus relevant sind. Ebenso hat Microsoft an eine Buttonabstraktion gedacht. Die verfügbare Game API (GAPI) ist mit einer kleineren Version von DirectX vergleichbar (vgl. [Harbour01]), das in der PC-Welt den Spiele-API-Standard darstellt, neben OpenGL als weitere 3D-API.

Die bereits verfügbaren Spiele sind beeindruckend. Es finden sich neben 3D-Autorennen, auch Flugsimulatoren und Ballerspiele. Allerdings ist fraglich, inwieweit Microsoft den Siegeszug, bzw. das Monopol, der Windows-Desktop-PCs auf den Handies und PDAs fortsetzen kann. Zumal Palm trotz schwächlicher Hardware und weniger buntem Betriebssystem, wacker Marktantteile in gleicher Größenordnung halten (vgl. [Rink02]) kann.

4.1.3 PalmOS

Das PalmOS[38] ist eine schon seit Längerem etablierte Plattform für Mobilecomputing und mittlerweile in die Jahre gekommen. Mit den Spezifikationen der PocketPCs mit RISC-StrongARMs, können die vergleichsweise schwachen CISC-Dragonballs der 68k Serie von Motorola nicht mithalten. Spezielle Spielefunktionen findet man nicht in den APIs. Trotzdem sind einige interessante Titel verfügbar, die größtenteils den Action- oder Puzzlespiele-Sektor abdecken.

37 www.microsoft.com/mobile

38 www.palmos.com

4.1.4 Symbian OS

Symbian[39] ist ein Betriebssystem-Spin-Off des Handheld Pioniers Psion, der mittlerweile seine Hardwareproduktion stark eingeschränkt hat. Zahlreiche große Mobiltelefonhersteller, wie z.B. Motorola und Nokia, halten Anteile oder sind Lizenznehmer.

Auch das Symbian OS bietet leider keine spezielle Spieleunterstützung in den APIs. Doch letzendlich wird der Markt entscheiden welche Plattform sich als Spieleplattform etablieren kann. Hier hat das Symbian OS mit Nokia als größtem Mobiltelefonhersteller keine schlechten Karten.

4.1.5 Linux

Linux ist mittlerweile auch auf PDAs verfügbar. Neben installierbaren Versionen für verschiedene PocketPCs von HP und Compaq, gibt es auch solche mit vorinstalliertem Linux, wie z.B. die Geräte von Agenda Computing. Weitere wurden zwar von Sharp und Sanyo angekündigt, jedoch nur als Entwicklerversionen ausgeliefert.

Problematisch stellen sich unter Linux vor allem die unterschiedlichen APIs für das GUI dar. Hier gibt es eigentlich nur proprietäre Lösungen, da XWindow zu mächtig für die kleinen Geräte ist und wohl auch etwas über das Ziel hinaus schießt. Das führt leider dazu, dass die wichigste Spielelibrary unter Linux, SDL, noch nicht für diese GUI-Lösungen verfügbar ist. Theoretisch wäre sie dank open-source einfach portierbar, diesen Schritt hat aber noch niemand vollzogen – wohl weil Linux in der mobilen Welt vorerst ein Nischenprodukt ist und nicht in den üblichen Verkaufscharts mit Palm, Pocket PC und Symbian, bzw. Nokia, zu finden ist.

4.1.6 Amiga DE

Das Amiga Digital Environment[40] (Amiga DE) ist eine auf einer Entwicklung der englischen Firma intent basierende Technologie. Implementiert wird in einer abs-

39 www.symbian.com

40 www.amiga.com

trakte Assembler Sprache. Zur Ladezeit wird der assemblierte Code vom Laufzeitsystem in native Maschinensprache übersetzt.

Zwar ist Amiga DE vor allem auch für kleine Endgeräte wie mobile Clients gedacht, momentan ist eine Laufzeitumgebung jedoch nur für Linux und Windows verfügbar. Eine handvoll Spiele ist bei Amiga erhältlich, jedoch fehlt ebenso die Installationsbasis, um Amiga DE tatsächlich als Zielplattform ernst zu nehmen.

4.1.7 Brew

Brew von Qualcomm[41] wurde anfangs als Konkurrenzprodukt zu Java 2 ME positioniert. Mittlerweile sieht der Hersteller Qualcomm aber auch eine Integration mit Java, sowie anderen Betriebssystemen vor.

4.1.8 ARM-Hardware-Plattform

Dass als einzige Hardware detailliert auf die ARM-Architekur[42] eingegangen wird, liegt an der großen und auch weiter steigenden Bedeutung dieses Prozessors im mobilen Bereich. Microsoft hat sich mittlerweile bei PocketPC für ARM entschieden und von MIPS und Hitachi-Prozessoren abgewandt. Psion setzte schon sehr früh auf ARM-Prozessoren und auch Palm wird auf diese Plattform nach eigenem Bekunden wechseln.

Gründe hierfür liegen sicher im geringen Stromverbrauch, der zur großen Verbreitung des Prozessors beitragen konnte. Hinzu kommt das ARM und andere 3rd-Parties ein breites Leistungsspektrum an zusätzlichen Komponenten anbieten, darunter auch Software-Lösungen, wie z.B. MP3- und MPEG-4-Dekoder, sowie ganze *3D-Engines*.
Vor allem für die Java-Entwicklung von Bedeutung ist die von ARM vorgestellte Jazelle-Technologie, die die ARM-Architektur um Java-Bytecode-Fähigkeiten erweitert. Java-Bytecode wird damit nativ ausführbar und im Vergleich zu Softwarelösungen sehr schnell.

41 www.qualcomm.com/brew

42 www.arm.com

Markttechnisch wird dies deshalb interessant, da sich Microsoft als Java-Konkurrent zu ARM bekannt hat. Ob sich die PoketPC-Hersteller diese zusätzliche Java-Fähigkeit entgehen lassen können ist zweifelhaft, sollte Jazelle tatsächlich realisiert werden. Dafür spricht jedoch einiges, ist es doch mittlerweile von einigen Firmen lizensiert worden.

4.2 Bestehende Lösungen

Da Mobile Computing zumindest eine Zeit lang mit großen Erwartungen verknüpft wurde, haben sich auch einige Firmen der Entwicklung von Entertainment-Lösungen für mobile Clients gewidmet. Die folgenden Betrachtungen sollen dabei einen Überblick über die verfügbaren Lösungen bieten und den Stand der Entwicklung aufzeigen.

Lösungen für nicht-interakive Unterhaltung wie Audio- und Video-Wiedergabe sind mittlerweile von Microsoft und von PacketVideo[43] verfügbar. PacketVideo stellt eine ganze Infrastruktur für Hosting und Aufbereitung der Inhalte zur Verfügung, sprich neben einem Player auch Server und Authoring-Komponenten. Interessanterweise sind weder Apples Quicktime[44] noch RealNetworks Technologien für drahtlose Geräte verfügbar.

PacketVideo setzt bei seinen Lösungen auf MPEG-4, das für die AV-Verarbeitung laut MPEG-Konsortium für mobile Endgeräte bestens geeignet ist (vgl. [MPEGXX]). PacketVideo ist als Player für einige Embedded-Plattformen verfügbar, darunter auch für Symbian und Windows CE.

In-Fusio[45] bietet ein ganzes Portfolio an Lösungen und Spielen. Als prominentester Kunde in Deutschland ist vodafone zu nennen. Die als „Load-A-Game“-Handies angepriesenen Geräte implementieren die „ExEn“-Spieleengine, die

43 www.packetvideo.com

44 Apple hat in den letzten Tagen (16.2.2002) Quicktime 6 mit MPEG-4-Unterstützung vorgestellt. Auf Grund von Lizenzfragen in Bezug auf MPEG-4 wurde es aber noch nicht an Endkunden weiter gegeben.

45 www.in-fusio.com

somit in der Lage sind, für diese Plattform geschriebene Spiele auszuführen. Zu den Herstellern, die die Software unterstützen gehören Sagem, Philips, Mitsubishi und mittlerweile auch Siemens. Trotzdem ist diese Lösung eher eine von mittelfristiger Bedeutung, In-Fusio selbst arbeitet an einem in Java implementierten Nachfolger, der wie ExEn den als „Gamezilla"-bezeichneten Spieleserver für *Billing,* Spiele-und Spielermanagement unterstützt.

Eine breite Angebotspalette für interactive Content preist gegenwärtig Synovial[46] an. Mit einer Lösung bestehend aus Server und Client, und der Integration verschiedener Client-Plattformen, sollen nahezu plattformunabhängige Multiplayer-Spiele möglich sein. In einem Kooperations-Projekt mit Sega und Compaq soll die neue „iPaq 3800"-Serie auf Basis dieser Technologie mit einem GameGear-Emulator ausgestattet werden.

Ähnliche Lösungen finden sich bei Partnern und Mitgliedern des Mobile Games Operability Forums[47] und Mobile Entertainment Forums[48], etwa bei den Lösungen von terraplay[49], die auch Billing-Funktionalitäten bieten.

Fathammer[50] bietet momentan für PocketPC auf ARM eine Demo ihrer X-Forge-Architektur, die 3D-Spiele auf PDAs ermöglichen soll. Die ersten Beispiele sind beeindruckend und zeigen das Potenzial der PocketPC-Plattform, wenn hierfür speziell optimierter Code verwendet wird. Allerdings ist anzuzweifeln, inwieweit die stark Hardcore-Gamer relevante 3D-Fähigkeit überhaupt für einfache Unterhaltung „zwischendurch" notwendig ist. Was ebenso für „SWERVE" von Superscape[51] gilt, das für Symbian OS und Windows CE entwickelt wird. SWERVE ist eine Plattform, die mit einem sehr kleinen *Memory-Footprint*, 3D-Grafiken ermöglichen soll.

46 www.synovial.com

47 www.mgif.org

48 www.mobileentertainmentforum.org

49 www.terraplay.com

50 www.fathammer.com

51 www.superscape.com

Die vorgestellten Lösungen zeigen, dass der Markt für Entwicklungs-Werkzeuge bereits in Bewegung ist. Als trendweisend sind vor allem solche Lösungen anzusehen, die ein komplettes Framework mit Client und Server bieten, für mehrere Plattformen verfügbar sind, andererseits aber stark optimiert werden, um ein Höchstmaß an Leistung aus den mobilen Endgeräten herauszukitzeln. Diese Faktoren sind als notwendig anzusehen, um einerseits die Wireless- und die Konnektivitätseigenschaften und andererseits die eingeschränkte Leistungsfähigkeit zu berücksichtigen.

4.3 Vorstellung des weiteren Analyse Ansatzes

An dieser Stelle wird die theoretische Betrachtung des Themas beendet, und die Diplomarbeit widmet sich der praktischen Evaluation der Entwicklungsmöglichkeiten unter Java 2 ME und MIDP. Hierzu wird zunächst eine Basissoftware zum Erstellen von Spielen auf mobilen Clients entworfen und in den folgenden Kapiteln realisiert.

MIDP wurde deshalb gewählt, da seine Plattformunabhängigkeit zumindest theoretisch den Einsatz auf einer Vielzahl von Endgeräten möglich macht. Damit soll nicht nur von konzeptioneller Seite eine breite Anzahl von Geräten berücksichtigt werden, sondern auch bei der praktischen Umsetzung.

5 Konzeption und Design

Um die Möglichkeiten innerhalb der mobilen und drahtlosen Unterhaltung detailliert von technischer Seite untersuchen zu können, bedarf es einer praktischen Evaluation. Hierzu beschreibt dieses Kapitel zunächst die grundsätzlichen Anforderungen an die Software, deren Entwurf und Implementierung in den restlichen Teilen der Diplomarbeit erörtert und erläutert wird.

Wie die vorhergehende Systemplattform-Analyse zeigte, haben die Plattformen Symbian, Windows CE, bzw. PocketPC, und Java, speziell das MIDP, gute Aussichten auf eine relevante Position in Entwicklerkreisen. Da sich Spiele weniger an Business-Kunden mit Organizern, sondern an den einfachen Konsumenten mit Handy wenden, zielt die Entwicklung der geplanten Spielesoftware auf eine Implementierung mit Java 2 ME und MIDP ab, das bei Entwicklern auf eine erste Resonanz bauen kann (vgl. [Fox01]).

5.1 Pflichtenheft und Systemanforderungen

Die zu erstellende Middleware, am besten mit einer API oder einem kleinen *Framework* vergleichbar, hat verschiedene, zum Teil recht hohe, Anforderungen zu erfüllen.

Grundsätzlich handelt es sich bei den meisten Spielen um „weiche“ Echtzeitsysteme (vgl. [Douglass99], S.8 und [Rollings00], S.510) . Weich deshalb, da leichte Verletzung der Zeitgrenzen im Regelfall keine gravierenden Konsequenzen haben, wie z.B. bei manchen medizinischen Anwendungen (Herzschrittmacher), und die Aufgaben nur in einem gewissen Zeitrahmen erfüllt werden müssen. Trotzdem sind die Anforderungen an die Zeittreue und die Geschwindigkeit des Systems im Vergleich zu „gewöhnlichen“ Anwendungen wie einer Textverarbeitung sehr hoch.

Die Echtzeitfähigkeit auf den kleinen mobilen Systemen zu gewährleisten ist eine Herausforderung, das selbe mit Java zu erreichen eine noch viel größere. Java ist nicht gerade bekannt für seine hohe Performanz, wie einige Werke zur Spie-

leentwicklung (vgl. [Kauert01]) und Java-Performanz allgemein (vgl. [Schatzmann01]) belegen.

Deshalb sei hier im Vorfeld schon bemerkt, dass, um die Echtzeitfähigkeit überhaupt erreichen zu können, einige Kniffe und Tricks angewendet werden, die nicht unbedingt der reinen Lehre der objektorientierten Softwareentwicklung entsprechen, wie in [Walter99] behandelt, aber leider notwendig sind.

Wie Kapitel 3 dieser Diplomarbeit gezeigt hat, sind die Anforderungen an ein Spiel auf mobilen Clients vor allem durch relativ einfache 2D-Funktionen charakterisiert. Prinzipiell soll die *Spiele-Engine* in der Lage sein, beliebige 2-D oder Pseudo-3D-Spiele zu realisieren, bzw. bei der Erstellung zumindest hilfreich sein, wenn spezielle Fähigkeiten von Nöten sind, die die Middleware nicht bietet.

Folgende, in Spieleentwicklerkreisen bekannte Anforderungen, müssen dabei erfüllt werden:

- bewegliche („scrollbare“) Hintergründe
- unabhängig voneinander steuerbare grafische Objekte (Sprites)
- Hintergründe, die größer sind als der darstellbare Bereich der Anzeige
- von der Anzeige unabhängiges Spielgeschehen
- Transformation der globalen Spieledaten in lokale Daten des Screens
- Gegenersteuerung, „*KI*“
- ausreichende Geschwindigkeit

Hinzu kommen folgende grundsätzliche Anforderungen an die Architektur:

- einfache Integration der Spielelogik
- einfache Erweiterbarkeit um Netzwerkfähigkeit
- Portierbarkeit auf andere Java-Dialekte oder Integration von herstellerspezifischen Erweiterungen durch Modularität

Anhand von einer technologischen Demonstrationen sollen die Fähigkeiten der Spiele-API in *7 Technologie Demonstration* gezeigt und damit die Basis für die Überprüfung der hier gemachten Aussagen über den Leistungsumfang in *8 Auswertung der praktischen Arbeit* gelegt werden.

Es sei weiterhin angemerkt, dass sich, bedingt durch den Mangel an Kenntnissen über die MIDP-Plattform, der Entwicklungsprozess stark iterativ vollzieht. Es geht also auch darum, welche Entwicklungsmethoden und Implementierungstechniken zum Einsatz kommen können, ja sogar das Algorithmen-Design selbst muss erst noch entschieden werden[52], da über die Performanz von MIDP als Spieleplattform noch keine wissenschaftlichen Grundlagen existieren.

Der Entwicklungsprozess orientiert sich damit stark am Rapid-Prototyping, und berücksichtigt bei den Design-Patterns [Douglass99], worin die Gegebenheiten in Realtime-Systemen speziell berücksichtigt werden.

52 eigentlich ein Widerspruch zu [Walter97]

5.2 Use-Cases

Um die in 5.1 beschriebenen Anforderungen verifizieren zu können, sollen einige Use-Cases die Einsatzszenarien der Spiele-API „wGame“ verdeutlichen.

5.2.1 Player-Game

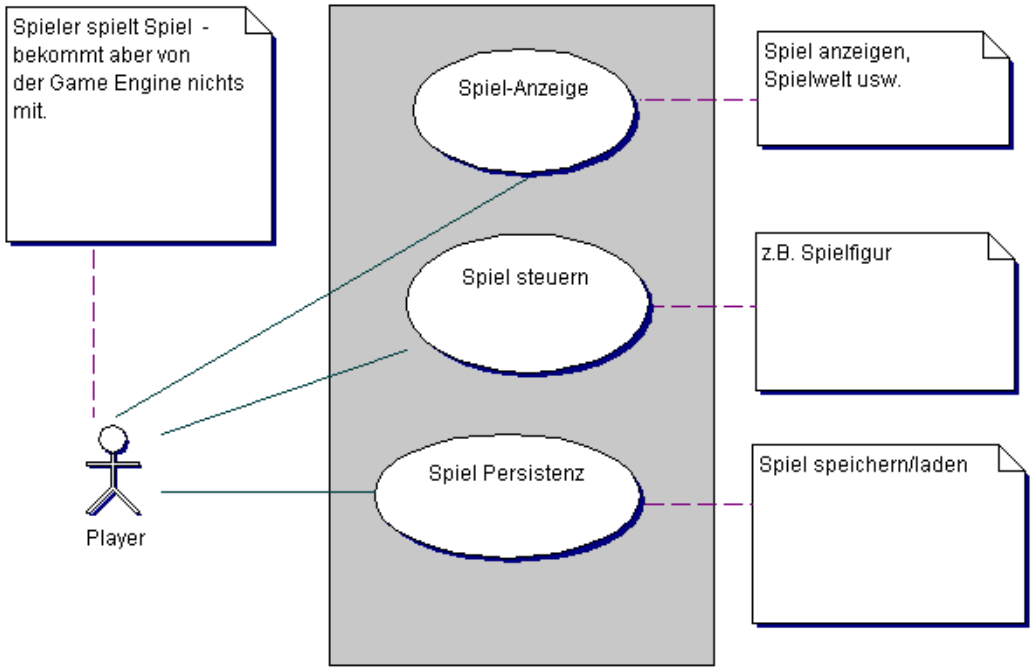

Abbildung 5-1: Player-Game-Use-Case

Obiger Use-Case beschreibt die wesentlichen Interaktionen zwischen Spieler (Player) und Spielsystem. Diese beschränken sich im Wesentlichen auf die Anzeige des Spiels, die Steuerung des Systems, vor allem der Spielfigur, bzw. des Spielgeschehens, und ein Persistenz-Management um Spieldaten zu sichern oder mit anderen Rechnern abzugleichen. Die Anforderungen, die daraus entstehen, lassen sich auf die Anforderungen der Spiele-API abbilden: Einzelne Spielobjekte lassen sich unabhängig steuern, eine Steuerungseingabe (per Steuerkreuz oder numerischem Pad) wird als selbstverständlich angesehen. Der Darstellung des Spiels lässt sich mittels der grafischen Funktionen, wie das Zeichnen des Hintergrundes und der Spielobjekten, realisieren. Das Netzwerkmodul soll das Persistenzmanagement von zu speichernden Daten auf den Server auslagern. Dabei

bemerkt der Spieler von der eigentlichen Spiele-Engine natürlich nichts. Er nimmt nur das Spiel war, wie es durch seine Spiele-Logik und seine audiovisuelle Repräsentation erzeugt wird.

5.2.2 Entwickler-Engine

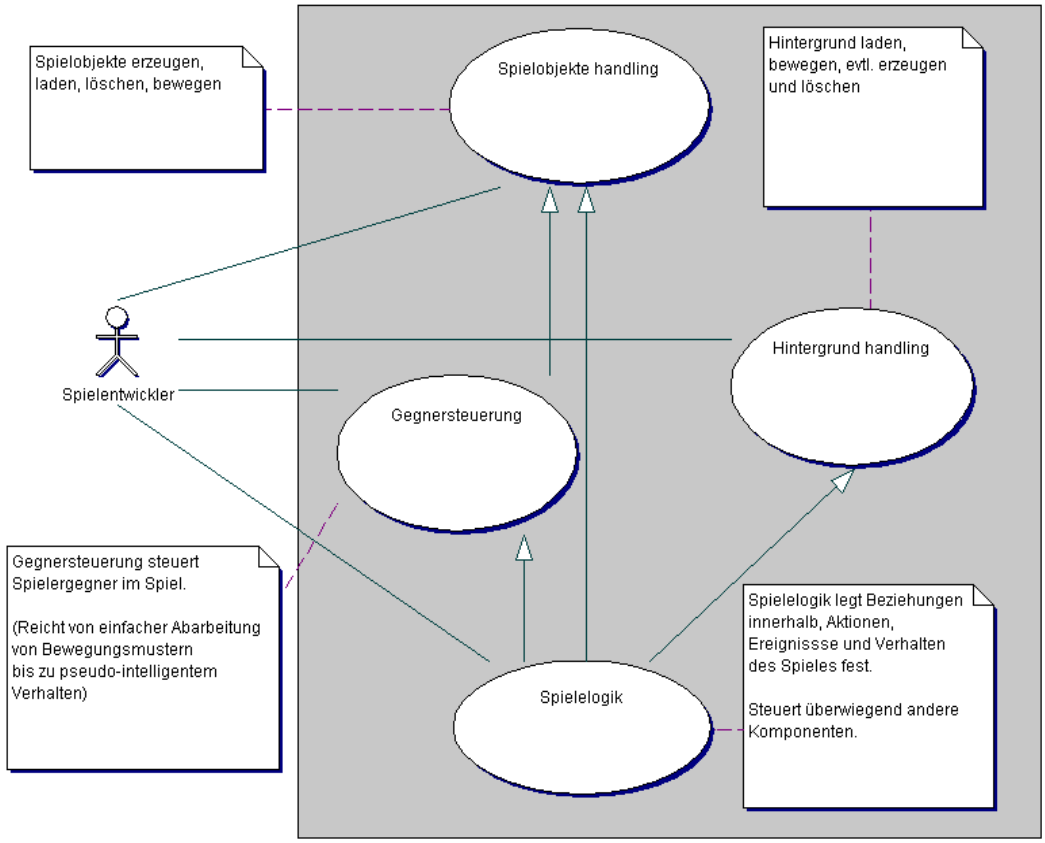

Abbildung 5-2: Entwickler-Engine-Use-Case

Der Use-Case zwischen Spieleentwickler und Spiele-Engine beschreibt die wesentlichen Beziehungen und Funktionen, die die Spieleengine gegenüber dem Programmierer wahrnimmt. Diese Funktionen lassen sich teilweise vom letzten Use-Case zwischen Spieler und Spielsystem ableiten. So ergeben sich aus der Anzeige Spiels die Aufteilung in Hintergrund-Handling und Spielobjekte-Handling. Diese Punkte werden in der Systemanforderung durch die Anforderungen zum Hintergrund und den graphische Objekten abgedeckt. Das Spiel wird in seiner Gesamtheit maßgeblich durch die Spielelogik bestimmt. Wirklich neu ist die Beziehung zwischen Gegnersteuerung und Programmierer. Die Gegnersteuerung findet sich im Pflichtenheft als ebensolche wieder. Sie soll über eine Reihe

von Standardunktionen realisiert werden, die dem Spieler eine Art künstliche Intelligenz suggerieren. Als wesentlichste Schnittstelle zum Programmierer ist die Spielelogik zu sehen. Sie steuert die Funktionalitäten der anderen Komponenten, auf die der Entwickler gegebenenfalls aber auch direkt zugreifen kann.

5.2.3 Client-Server

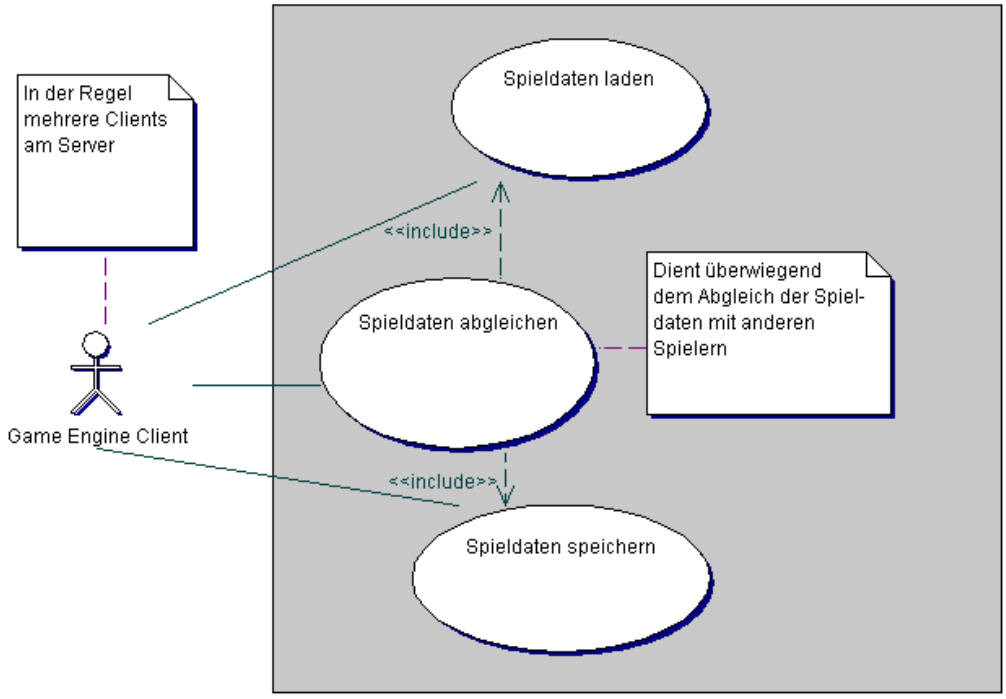

Abbildung 5-3: Client-Server-Use-Case

Die Beziehung zwischen Spieleclient und Spieleserver definiert sich überwiegend durch das Management der Spieledaten. Hier sind die wesentlichen Funktionen „Spieldaten laden“ und „Spieldaten speichern“, welche auch von „Spieldaten abgleichen“ verwendet werden. Die Kommunikation zum Server übernimmt bei wGame ein Netzwerk-Controller, welcher aber zusammen mit dem Spieleserver auf Grund des relativ geringen Zeitraumes der Diplomarbeit leider nicht mehr in diesem Rahmen implementiert werden kann.

5.3 Architektur

Grundgedanke der wGame-API ist es, Voraussetzungen zu schaffen, um die Entwicklung von Spielen auf mobilen Endgeräten mit Java 2 ME und MIDP zu beschleunigen und zu vereinfachen. Dabei muss gewährleistet werden, dass die hiermit erstellten Spiele auf einer möglichst großen Anzahl von Geräten lauffähig sind und eine ausreichende Geschwindigkeit bieten.

Portabilität und Geschwindigkeit gehören deshalb zu den wichtigstenen Kriterien, neben den allgemeinen Kriterien wie Wartbarkeit und Erweiterbarkeit [Walter97]. Da es sich bei Spielen um Systeme handelt, die Echtzeit-Fähigkeiten besitzen, wurde unter anderem auch [Douglass99] konsultiert. Diese Quelle enthält neben [Kauert01] und [Rollings00] jedoch nur eine sehr vage Beschreibung der Architektur einer so genannten *Spiele-Engine.* Deshalb musste im ersten Schritt der Entwicklungsphase zu erst einmal eine Architektur entworfen werden.

Selbst in Werken wie [DeLoura00] und [DeLoura01] finden sich nur grundlegende Beschreibungen zum OOSE. Hinsichtlich einer allgemeingültigen Architektur für Spiele halten sie sich sehr bedeckt, obwohl die sonstigen Beispiele in diesen Büchern technologisch sehr weit fortgeschritten sind und im 3D-Grafik-Bereich alle wichtigen Themengebiete ansprechen.

Ohnehin ist sicherlich fraglich, in wie weit eine Architektur, die für im Vergleich zu Handies und Organizern große Computersysteme (wie in diesen Quellen behandelt) gedacht ist, sich sinnvoll auf mobilen Clients einsetzen lässt.

Es sei an dieser Stelle angemerkt, dass auf einen umfangreichen Einsatz von Patterns verzichtet wird, oder besser gesagt gewisse Patterfunktionalitäten in ein und der selben Klasee Anwendung finden. Wie die Arbeiten [Wolf00] und [Kauert01] zeigen, ist dies notwendig um einerseits hohe Geschwindigkeit und andererseits ein geringes *Memory-Footprint* zu erreichen.

Bei dem grundlegenden Prinzip des Systems handelt es sich um ein MVC-Pattern (vgl. [Gamma00]):

Das Modell wird durch die Spieldaten, wie z.B. Spielerpositionen und Stati bestimmt.
Die Darstellung übernimmt die View-Komponente.
Die Manipulation der Daten erfolgt über Controller, z.B. Kollisionserkennung, KI und Eingabe-Handler.

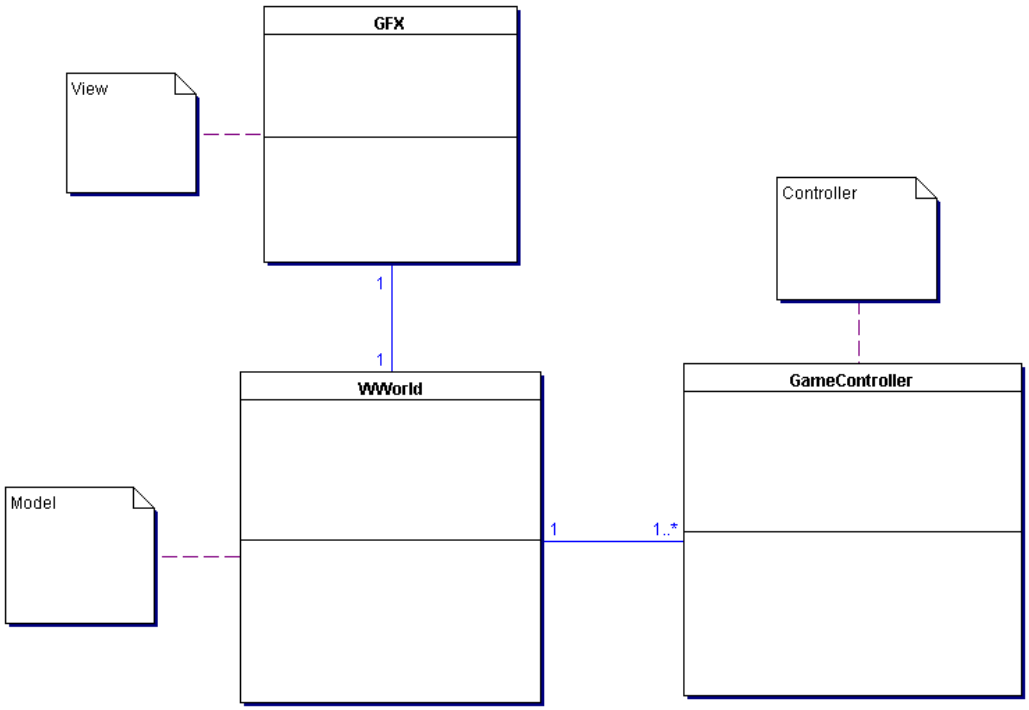

Abbildung 5-4: MVC-Struktur von wGame

Mit der Anlehnung an das MVC ist auch gewährleistet, dass sich die Architektur von wGame leicht in bestehende Frameworks, wie dem MAB (Mobile Application Broker) von wearix, integrieren lässt.

Zu den grundlegenden Komponenten/Controllern zählen, wie teilweise schon erwähnt:

die KI für die Verwaltung von computergesteuerten Spielern
eine Kollisionserkennung um Berührungen und Überschneidungen von beweglichen Objekten (auch Spielern) untereinander und dem Hintergrund festzustellen

einem Input-Handler der die Eingaben des Benutzers puffert und mit dem Spielgeschehen synchronisiert
eine Netzwerkkomponente um den Austausch von Spieldaten mit anderen Rechnern zu ermöglichen
einem Clipping-Modul um die Transformation der globalen Spieledaten in lokale Bildschirmkoordinaten und eine Darstellung zu transformieren
einer Prozesskomponente, die die einzelnen Module aufruft und miteinander synchronisiert
eine Logik-Komponenten als Schnittstelle für Programmierer der eigentlichen Spieleanwendung.

Die Idee hinter der Logikkomponente ist die möglichst einfache Programmierung eines Spieletitels. Dem Anwendungsprogrammierer wird eine Schnittstelle zur Verfügung gestellt, über die er einfach auf die einzelnen Komponenten zugreifen kann, ohne sich intensiv mit der internen Struktur der Game-API beschäftigen zu müssen. Damit ist es möglich, dass sich der Programmierer eingehend mit der Programmierung des Spieles beschäftigen kann und nicht von grundlegenden Fragen der Systems abgelenkt wird. Abgeleitet wurde die Idee für die Logik-komponente durch Skripting-Möglichkeiten, wie sie in „großen“ Spiele-Engines für PCs zur Verfügung stehen[53].

Die weiteren Vorteile gehen mit denen des MVC-Patterns einher. Das heißt durch eine Anpassung der View-Komponente, lässt sich relativ leicht eine Portierung der API auf andere Java-Grafikschnittstellen, wie etwa dem AWT von Personal-Java oder der Java 2 Standard Edition, vornehmen.

Theoretisch ließe sich auch eine Netzwerkkomponente direkt an die View anhängen und das Model könnte Serverseitig mit den Controllern gehostet werden, was jedoch bisher nicht vorgesehen ist. Hierfür wäre auch ein sehr schnelles Netzwerk notwendig, um die Daten für die Darstellung zum Client übertragen zu können.

53 bspw. bei Quake, Unreal, aber auch [Kauert01]

Da jeder Client sein eigenes Model hostet, ist es nicht notwendig mehrere gleichzeitige Views zu unterstützen[54], wovon während der Implementierung als Optimierungsmöglichkeit gebrauch gemacht wird.

5.3.1 Model

Das Model speichert die für das Spiel relevanten Daten. Im Model WWorld, werden die Informationen über die Spielobjekte und die Map des Hintergrundes gehalten.

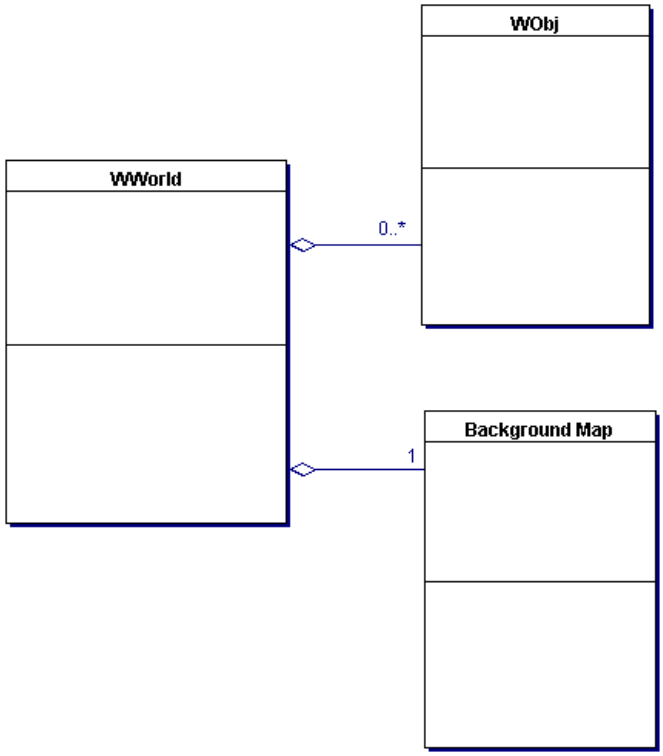

Abbildung 5-5: Klassendiagramm Model

5.3.1.1 Spielewelt WWorld

Die Spielwelt bietet als eine zentrale Einrichtung Zugriff auf die einzelnen Spielobjekte und den Hintergrund. Dadurch sind die Daten nach dem MVC-Prinzip von den Controllern und der Anzeige/View getrennt.

54 *Splitscreens* sind auf Grund der kleinen Bildschirme nicht sinnvoll, wie in *3.5 Gestalterische Überlegungen zu Grafik und Ton von mobilen Computerspielen* schon erwähnt wurde.

5.3.1.2 Spielobjekte WObj

Spielobjekte werden in wGame durch einzelne Objekte repräsentiert, die die notwendigen Eigenschaften speichern. Hierzu gehören nicht nur die Position des Objektes, sondern auch Daten wie Objekttyp, Darstellung, Zustand und weitere Eigenschaften für KI und Kollisionserkennung. Spielobjekte werden mittels des Clippings in Sprites transformiert, falls eine Darstellung erforderlich ist.

Grundsätzlich verläuft der Fluss der Daten, auf die Anzeige des Spieles bezogen, vom Model zur View. Alle Berechnungen werden auf Model-Seite durchgeführt und dann auf eine Darstellung transformiert (was ja auch Grundgedanke des MVCs ist).

5.3.1.3 Spielfeld/Map

Der Hintergrund wird auf Model-Ebene nur durch eine *Map* (2D-Array) repräsentiert. Dabei kann diese Map auch von der Map der Hintergrunddarstellung auf View-Ebene abweichen, falls dies z.B. für die Hintergrundkollisionen notwendig sein sollte. Die Folge ist eine weitgehende Flexibilität und Unabhängigkeit von der View.

5.3.2 View

Die View-Komponente und ihre Klassen übernehmen die Darstellung des Spieles auf dem Bildschirm. Dabei ermöglicht die Trennung der View von den restlichen

Komponenten der API, eine relativ leichte Portierung auf andere Java-Konfigurationen wie Personal Java oder die Java Standard Edition. Die View-Komponente gliedert sich in:

Sprite-Objekte mit Animationen

den Hintergrund mit einer Map und den dazugehörigen Tiles

die eigentliche Grafikschnittstelle GFX.

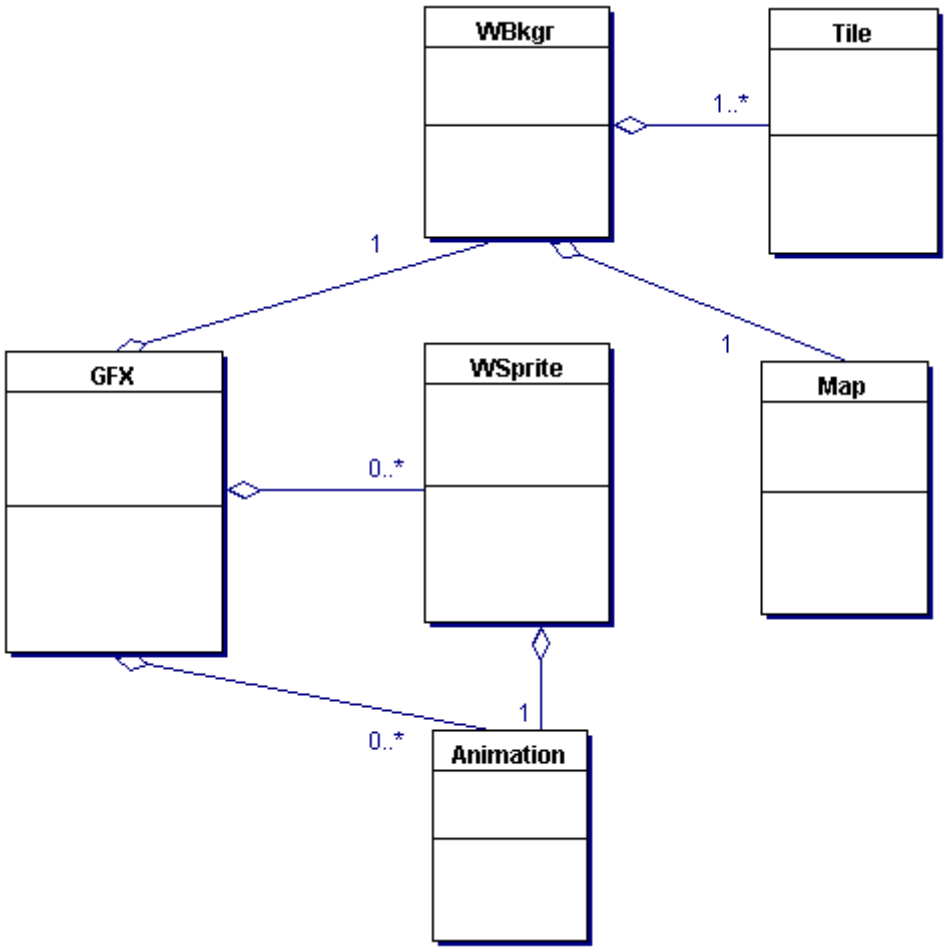

Abbildung 5-6: Klassendiagramm View

5.3.2.1 GFX

Die GFX kapselt die einzelnen View-Komponenten nach aussen und bildet damit eine Facade [Gamma00]. Damit muss sie auch die Funktionalität eines Builders aus [Gamma00] erfüllen, um Sprites und Hintergründe für das Spiel zur Verfügung stellen zu können. Die Buildereigenschaft macht es auch einfach an dieser Stelle ein *Pooling* für die Sprite-Objekte einzusetzen, da es bei Spielen mit großer Spielfläche und vielen Objekten schnell dazu kommen kann, dass eine Vielzahl

von Spriteobjekten reserviert und freigegeben werden muss, was einerseits sehr zeitintensiv ist (vgl. [Shirazi00] oder [Schatzmann01]), andererseits auch den Speicher unnötig fragmentiert und zu Speicherproblemen führen kann, wenn kein geeigneter Speicherbereich mehr zur Verfügung steht.

Bei den gegebenen Speicherbedingungen werden statt komfortablereren Datenstrukturen lediglich Arrays für die Datenhaltung eingesetzt., z.B. für Animations- und Spritelisten. Sie bieten den Vorteil, dass sie gegenüber anderen Klassen von Datenstrukturen wesentlich weniger Speicher belegen und im Bedarfsfall direkt ohne *Accessorfunktionen* ansprechbar sind, was bei den notwendigen Optimierung sehr entgegenkommend ist.

Überhaupt wurde bei der Planung der einzelnen Komponenten viel Wert auf Optimierbarkeit gelegt. Die einzelnen Klassen der API bieten eine ausreichende Kapselung, um Optimierungen getrennt in den einzelnen Komponenten vornehmen zu können und ohne dabei andere Komponenten ändern zu müssen. Für die Evaluation der Software-Engineering-Möglichkeiten im Hinblick auf einsetzbare objekt-orientierte Entwicklungsmethoden heißt das, dass diese die Optimierung und damit die Geschwindigkeit unterstützen.

5.3.2.2 Sprites

Ein wichtiges Feature der Spiele-Engine sind so genannte Sprites. Als Sprites werden die beweglichen Grafik-Objekte bezeichnet. Sie können in wGame beliebig groß sein und werden über den Bildschirmhintergrund gezeichnet. Verwaltet werden die einzelnen Sprites über das GFX-Objekt, der Hauptviewkomponente. Die Spriteobjekte in wGame sind auch für ihre Animation zuständig, d.h. sie können selbständig eine an sie übergebene Bildsequenz abspielen, bzw. anzeigen.

5.3.2.3 Hintergrund

Der Hintergrund „WBkgr“ ist ein weiterer Teil der View. Mittels dieses Objekts wird die Darstellung des Hintergrunds erreicht. Der Hintergrund wird dabei aus kleinen quadratischen Grafiken zusammengesetzt, so genannten *Tiles*. Tiles haben

wie Sprites bei 2D-Spielen eine lange Tradition. Schon beim Commodore C64 wurde der Textmodus für ein Hintergrund-Tiling benutzt, und auch moderne Handheld-Konsolen wie der GameBoy-Advance bieten ein hardwaregestütztes Tiling des Hintergrundes. Auf Java-MIDP muss leider auf eine Software-Implementierung ausgewichen werden. Trotzdem bietet es gegenüber einem einzigen großen Hintergrundbild einige Vorteile. Hintergründe können mittels eines so genannten Map-Files aus nur wenigen Grafikteilen gebaut werden. Das spart eine Menge Speicher bei der Daten-Übertragung, da nur noch ein paar Tiles und ein kleines Map-File geladen werden müssen, nicht eine ganze Hintergrundgrafik. Zudem ermöglicht die Verwendung von Tiling eine flexible Handhabung des Hintergrundes, da dieser nach spezifischen Anforderungen gebaut werden kann, was in *6.2.2 Hintergründe und Maps: WBkgr* noch näher erläutert wird.

5.3.2.4 Clipping

Zu den kritischen Bereichen der Viewkomponente gehört das Clipping. Pro Bildschirmaufbau untersucht das Clipping alle Spielobjekte. Es entscheidet, ob sie dargestellt werden müssen, lädt entsprechend Sprites neu oder löscht diese, je nachdem welcher Bereich der Spielwelt gerade angezeigt wird.

Prinzipiell lässt sich schwer sagen, ob das Clipping nun zur Viewkomponente gehört oder eine Controllerkomponente ist. Es sitzt einerseits als Controller am Model und transformiert die Daten. Andererseits ist es ein integraler Bestandteil der View bei Computerspielen allgemein. Somit ist das Clipping als Ausnahme im MVC-Gefüge anzusehen, ohne deren Eigenschaften und Eigenheit jedoch kaum ein Computerspiel mit schneller Grafik auskommen dürfte. Das Clipping ist auch die Komponente, die als eine der performanz-kritischsten anzusehen ist. Ob sich das Spiel schnell genug zeichnen kann, hängt nicht zuletzt von der Fähigkeit des Clippings ab, die Daten des Models schnell genug in eine View-Darstellung transformieren zu können. Das MVC-Pattern gebietet, dass alle Manipulationen der Daten auf Modelebene durchgeführt werden. Dadurch wird die Bedeutung des Clippings aber auch gleichzeitig gegenüber einer (im Hinblick auf Plattformunabhängigkeit sicherlich nicht sinnvollen) direkten View-Manipulation gesteigert.

In zukünftigen Entwicklungsschritten könnte das Clipping „WClip“ noch andere Aufgaben zugesprochen bekommen. So wäre es möglich und bei großen Spielewelten auch sinnvoll, die Verarbeitung der Spieledaten durch Controller auf sichtbare oder eine anderweitig beschränkte Anzahl von Daten zu reduzieren. Auch diese Markierung der für diese Controller relevanten Daten, würde dann das Clipping übernehmen.

5.3.3 Controller

Über Controller werden innerhalb von wGame datenverarbeitende Funktionaltäten realisiert. Das bedeutet implizit auch, dass die Controller nur auf Model-Ebene operieren. Ausnahme ist das Clipping. Es transformiert die globalen Daten des Models in Viewdaten und hat dabei auch Zugriff auf die Sprites, um eine Möglichst schnelle Verarbeitung zu ermöglichen. Erste Tests von wGame zeigten, wie wichtig das ist. Mit der Programmierung in Java und den für Java relativ schwachen Prozessoren, liegt man mit einer allgemeinen Spielengine sehr hart und nah an der Grenze dessen, was auf diesen Plattformen realisierbar ist.

Aus Performance-Gründen ist bisher auch kein dynamisches Einbinden von Kontrollern über eine zentral verwaltete Liste möglich und dementsprechend auch keine einheitliche Vaterklasse für die Kontroller vorhanden, was prinzipiell aus Software-Engineering-Sicht erstrebenswert wäre. Dies ist auf Grund der modularen Architektur möglich und auch für spätere Portierungen oder Projekte angedacht – speziell für größere Plattformen wie das Personal Profile, die Java 2 Standard Edition oder andere Implementierungssprachen wie C++, die von sich aus mehr Geschwindigkeitsspielraum mit sich bringen.

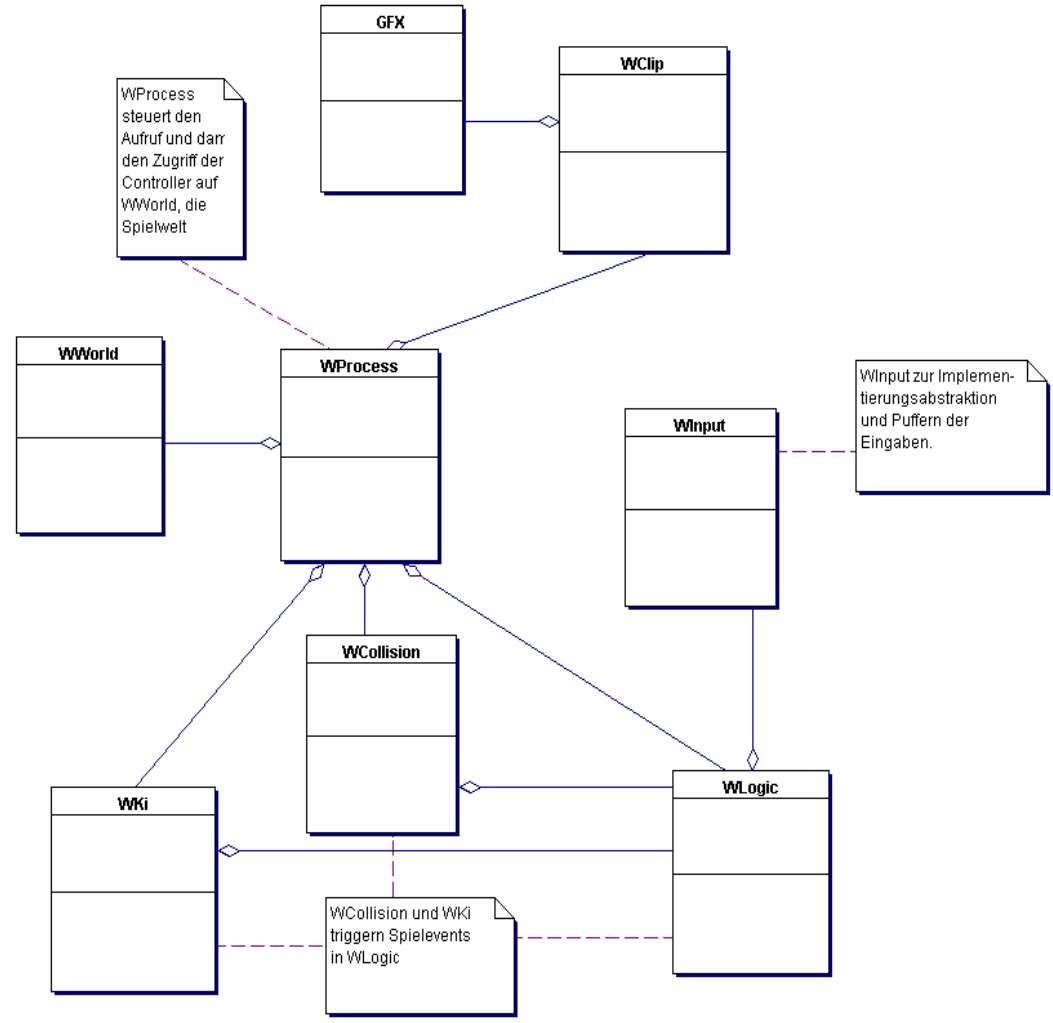

Abbildung 5-7: Funktionsorientiertes Klassendiagramm der Controller

5.3.3.1 Timing- und Prozesssteuerung

Ein zentraler Controller steuert den Aufruf der anderen Controller, dieser heißt innerhalb von wGame „WProcess“. Er frägt darüber hinaus die Systemzeit ab und berechnet die Differenz zum letzten Lauf der so genannten *Game-Loop*.

Die Hauptmethode von WProcess kann als ebensolche Game-Loop angesehen werden, wie sie schon seit den Kindertagen der Spieleentwicklung verwendet wird. Hier werden die zentralen Vorgänge des Spiels angestossen, d.h. die Methoden der einzelnen Komponenten aufgerufen.

5.3.3.2 Input-Handling

Da unter Java MIDP Eingabe-Events über eine Art Listener hereinkommen und nicht über ein Polling abgefragt werden können, ist es notwendig, Eingaben mittels eines Rendezvous-Patterns, wie in [Douglass99] beschrieben, mit dem Spielprozess zu synchronisieren. Dazu puffert das Input Handling „WInput“ die Eingaben. Es sei an dieser Stelle angemerkt, dass das MIDP hier ein Einklinken des Input-Handlers auf Viewseite notwendig macht, da hier auch die Eingabe-Events gefangen werden. Die restliche Architektur stört dies aber nicht, da sie durch den Input-Handler wie über eine Art Bridge (vgl. [Gamma00]) die gepufferten und transformierten Events bekommt.

5.3.3.3 Kollisionserkennung

Die Kollisionserkennung „WCollision“ dient dazu, Berührungen zwischen einzelnen Spielobjekten festzustellen. Kommt es zu einer Kollision, bzw. Berührung, zwischen Objekten, wird ein Event an die Spielelogik gefeuert, bzw. ein *Callback* aufgerufen.

Außerdem stellt der Controller WCollision Methoden zur Hintergrunderkennung aus Spielobjektsicht zur Verfügung, die hierfür wiederum auf Konstanten der Spielelogik, sowie auf Event-Callbacks zugreifen.

5.3.3.4 Gegnersteuerung

Der Controller WKi (KI, Künstliche Intelligenz) übernimmt die Steuerung der gegnerischen Computer-Spieler, auch Non-Player-Characters (NPCs) genannt. Allerdings wird hier keinesfalls tatsächlich eine künstliche Intelligenz, wie z.B. ein Neuronales Netz implementiert. Vielmehr finden sich hier einfache Methoden, damit die Computergegner den Spieler etwa verfolgen oder ein „zufälliges“ Bewegungsmuster vortäuschen können.

5.3.3.5 Netzwerkkomponente

Die Netzwerkkomponente dient zur Vernetzung mehrerer wGame-Systeme über eine beliebige Netzwerkverbindung. Hier wird über einen Kontroller ein Proxy

(vgl. [Gamma00]) realisiert, der beliebige Netzwerkverbindungen nutzen kann, ohne dass sich die restlichen Komponenten darum kümmern müssen, wie eine und welche Verbindung aufgebaut wird, bzw. auch wie die Daten ausgetauscht werden.

5.3.3.5.1 Einschränkungen bei MIDP

Aus zeitlichen Gründen wurde diese Komponente aber nicht implementiert. Hinzu kommt, dass unter MIDP ohne JavaPhone API nur Http1.1-Verbindungen möglich sind. Dabei werden die für Spiele wichtigen Kommunikationsmöglichkeiten wie Socketverbindungen nicht unterstützt. Hinzu kommt, dass das im Vergleich zu TCP/IP und UDP langsame Http, in der unter MIDP implementierten Version erst eine Serverkomponente nötig macht, deren Realisation den Umfang dieser Diplomarbeit sprengen würde.

5.3.3.5.2 Prinzipielle Möglichkeiten

Prinzipiell sollten Peer-to-Peer-Verbindungen (zwischen Spielern direkt) und Client/Server-Verbindungen (zwischen Spielern und Server) möglich sein. Vom technischen Standpunkt liessen sich die Spieler-zu-Spieler-Verbindungen (also Peer-to-Peer) auf alle Fälle auch über Client/Server-Verbindungen realisieren. Es bleibt jedoch anzumerken, dass es im Zusammenhang mit lokal-vernetzten Spielen zu Situationen kommen kann, die eine Peer-to-Peer-Verbindung wünschenswert machen. So ermöglicht Bluetooth eine gebührenfreie Vernetzung im lokalen Bereich, im Gegensatz zu den von Mobilfunkanbietern zur Verfügung gestellten Möglichkeiten (z.B. GPRS, UMTS). Es ist anzunehmen, dass der Konsument auch von diesen Möglichkeiten gebrauch machen will, sofern er die Möglichkeit dazu hat. Somit sollten grundsätzlich beide Ansätze berücksichtigt werden, um alle Einsatzszenarien für eine Vernetzung der Spieler berücksichtigen zu können.

5.3.3.6 Sound

Ähnlich wie bei der Netzwerkkomponente von wGame verhält es sich bei der Soundünterstützung. MIDP sieht keine Erzeugung von Klang in irgendeiner

Weise vor. Lediglich herstellerspezifische Bibliotheken, wie von Nokia ([Nokia01a]) und Siemens ([Siemens01]), ermöglichen das erzeugen von auditivem Output, leider jedoch auf uneinheitliche Weise. Ziel wäre es eine einheitliche abstrakte Schnittstelle zu schaffen, die allgemeine Features von Soundinterfaces unterstützen könnte. Da bei den herstellerspezifischen Libraries noch einiges in Bewegung ist (Nokia etwa stellt nur ein API-Beschreibung [Nokia01a] zur Verfügung, jedoch keinerlei Klassen oder Emulatoren), wird von einer Implementierung abgesehen.

Bewegen sich die Soundmöglichkeiten der Handies einmal aus dem Bereich des einfachen, monophonen Abspielens von Tonfolgen, mit den Variationsmöglichkeiten lediglich in der Tonhöhe und -dauer, wird auch die Soundprogrammierung interessanter. Bis jetzt stehen den mobilen Clients jedoch sehr eingeschränkte Möglichkeiten unter Java zur Verfügung, die sich allenfalls durch native Implementierungen (zumindest unter Personal Java) auf das Niveau eines GameBoys anheben lassen[55].

5.3.3.7 Spielelogik

Die Spielelogik „WLogic“ empfängt Events der einzelnen Controller und stellt die Schnittstelle für die Programmierer der Spielesoftware dar. Die Vorgänge und Reaktionen auf einzelne Ereignisse werden hier ausprogrammiert. Zum Beispiel was passiert, wenn der Spieler mit einem Monster kollidiert, wie viele Punkte er für das Einsammeln eines Extras bekommt usw.

55 Insbesondere der GameBoy Advance bietet neben seinen 3-Synthi-Stimmen zwei DA-Kanäle zur Klangwiedergabe. Durch Softwaremixing lassen sich aber auch mehr Stimmen über die DA-Wandler realisieren, wie etwa [EngineSoftwareXX] zeigt.

6 Implementierung

Bis auf die Diplomarbeit [Wolf00], die übrigens nicht das für diese Diplomarbeit relevante MIDP behandelt, liegen im Bereich der Implementierung kaum Erfahrungswerte vor. Die veröffentlichen Werke zur Java 2 ME beschränken sich mehr auf ein Beschreiben der Features und kleinere Technologie-Demonstrationen. Ebenso verhält es sich bei den bisher für das MIDP erhältlichen Spielen. Sie sind mehr ein schnell zusammengehacktes Programm als ein strukturierter Code, zumindest bei den Beispielen des „J2ME Wireless Toolkit 1.0.3“ von Sun[56].

Hinzu kommen bei diesen Spieledemos ein paar zweifelhafte Techniken, wie das zwischenspeichern des Graphikkontextes in Java (`Graphics`), um ihn dann außerhalb von `Displayable.paint()` zu verwenden. Ob dies auf allen MIDP-Implementierungen funktioniert ist zweifelhaft. Schließlich kann das System normalerweise damit rechnen, dass nach Abarbeiten von `paint()` kein Zugriff auf die Grafikhardware stattfindet. Hier sind meiner Meinung nach Fehlfunktionen vorprogramiert.

wGame sollte im Laufe der Implementierungsphase, die sich stark an einem Rapid-Prototyping-Prozess orientiert, die Fähigkeiten und Möglichkeiten der MIDP-Plattform für Spiele evaluieren und Erfahrungswerte liefern, in wie weit Computerspiele unter dem MIDP implementiert werden können.

Die Berücksichtigung der verfügbaren Systeme gestaltete sich angesichts der jungen Technologie als schwierig. Hier sind die einzelnen Systeme nicht so richtig ausgereift, was die Tests auf verschiedenen Emulatoren und Systemen zeigten. So ließ sich auf dem Emulator von Nokia unter JBuilder nicht mit Haltepunkten debuggen, „MIDP for PalmOS 1.0“ verschickte die „KeyReleased“-Events nicht sauber sobald mehrer Knöpfe gedrückt waren und der Siemens SL45i Emulator konnte keine „runnable“-Objekte mit `callSerially(runnable Object)` starten, wie es in MIDP eingentlich vorgesehen ist, um Operationen nach `paint()` einzuleiten.

56 Für die Java Standard Edition allgemein gibt es zwar Werke zur Spieleprogrammierung, doch sind diese einfach oder rudimentär gehalten - auch wenn sie ein Tiling, wie in [Steinke00], oder *Quadtrees* zur Speicherung von Spielobjekten, wie in [Petchel01], verwenden.

Hinzu kam das Gefühl, das einen beschleicht, wenn man Software einsetzt, die noch in der Beta-Phase steckt. Hier und da tauchten immer wieder Fehler auf, die sich kaum reproduzieren ließen und somit den Entwicklungsprozess erschwerten.

Das Problem, das sich hier für diese Diplomarbeit ergibt, ist, dass man mit junger Technologie entwickelt, deren Unausgereiftheit dazu führen kann, dass ein falsche Beuurteilung entsteht. Dazu gehört vor allem, dass man die meissten Spezifikationen als vorläufig ansehen muss, solange nur eine Handvoll Geräte auf dem Markt sind. So ist anzunehmen, dass sowohl schnellere, wie auch langsamere Geräte auf dem Markt erscheinen werden. Bereits jetzt ist die Leistungskluft sehr gross: Palm-Geräte sind im Vergleich zu den StrongARM 32Bit RISCs in den PocketPCs mit ihren Motorola 68k Prozessoren schwach ausgestattet.

Dazu kommen einzelne Betriebssysteme, die laut [Symbian00] starke Performance-Unterschiede in der Java-Verarbeitung aufweisen. Ein HP Jornada mit Windows CE und 200Mhz StrongARM wird hier beispielsweise von EPOC mit einfachem 20Mhz ARM geschlagen. Hier ist die Entwicklung der Voraussetzungen für diese Arbeit leider auch in technischer Hinsicht[57] noch nicht abgeschlossen.

6.1 Implementierung des Models

Das Model bildet die wichtigsten Spieldaten ab. Das sind einmal die Spielobjekte `WObj` und eine Hintergrund-Map, die mittels eines einfachen 2D-Integer-Arrays abgebildet wird.

6.1.1 Spielewelt: WWorld

Die Spielewelt `WWorld` stellt Funktionen und Methoden zur Verfügung, um Zugriff auf die Objekte zu erhalten und um diese zu verwalten.
Ähnlich wie bei den Sprites der Grafikkomponente, werden die `WObjs` in ein einfaches Array eingehängt, um eine möglichst effiziente Verarbeitung garantieren zu können. Arrays bieten prinzipbedingt eine Typsicherheit, bzw. erhalten diese,

57 neben den gestalterischen Grundlagen, wie sich in *3.1-3.5* gezeigt hat

während die langsamere und speicherhungrige Implementierungsalternative über einen Java-Vektor dieses nicht gewährleistet, bzw. das über eine zusätzlich zu implementierende `Object.instanceOf`-Abfrage sicherstellen muss.

In `WWorld` können ähnlich wie bei den `WSprites` und der `GFX`-Klasse, `WObjs` über `addWObjs(WObj newWObj)` und `removeWObjs(int index)` verwaltet werden. Durch die Managementfunktion wird theoretisch ein Pooling (vgl. [Symbian00]) ermöglicht, welches aber ebenfalls wie bei `GFX` erst noch implementiert werden muss. Dazu müsste auch noch eine Builder-Funktionalität für `WObj`-Objekte in `WWorld` implementiert werden.

Aus Sichtweise der Robustheit ist die Tatsache problematisch, dass `WWorld` das Array zur Verarbeitung an die Kontroller über `getWObjects()` herausgibt. Hier war aus Performancegründen leider keine andere Lösung möglich. Das Problem wäre auch nicht unbedingt das Anfertigen einer Kopie des Arrays gewesen, um diese in den Controllern verarbeiten zu können. Dies hätte `System.arraycopy(...)` erledigt, welches in der Regel nativ implemtiert ist. Problematisch wäre das Durchiterieren, um Änderungen an den WObjs zu übernehmen. Das wäre nach jedem Controller-Zugriff nötig gewesen, um die Daten konsistent zu halten, was den Rechenaufwand nahezu verdoppelt hätte. Würden dann noch `WObjs` hinzugefügt oder gelöscht, wären ganz neue Kopien des `WObjs`-Arrays notwendig. Diese Maßnahmen würden den Code nicht nur langsamer machen, sondern auch verkomplizieren und damit schwerer wartbar machen. Das KISS-Prinzip („Keep it simple, stupid!“), nach dem wGame implementiert ist, trägt im Gegensatz zur obigen Methode zu einer indirekten Robustheit bei.

Die Implemetierung von *Accessormethoden* für den Zugriff auf einzelne Objekte scheidet ebenfalls aus. Hier würde die Performance bei den ständigen Zugriffen auf die einzelnen Objekte und ihre Attribute viel zu sehr leiden, da durch die Methoden-Aufrufe ein signifikanter Overhead innerhalb der Iterationen durch die Controller entstehen würde, wie [Symbian00] oder [Shirazi00] belegen.

6.1.2 Spielobjekte: WObj

`WObj`-Instanzen dienen als Repräsentation der beweglichen Spielobjekte. Sie beinhalten für das Objekt relevante Daten wie Position, Bewegungsvektor, Bewegungsverzögerung, Zustand, Kollisionsparameter, Animationsnummer und Clippingbreite.

Implementiert sind sie als einfache Datenkontainer, d.h. die Klassen-Attribute sind `public` um eine ausreichende Performance gewährleisten zu können, wenn in den Controllern eine Vielzahl dieser Objekte durchgearbeitet wird. Dabei werden innerhalb der `WObjs` keine komplexen Daten gespeichert, sondern lediglich Parameter, auf die die Kontroller zugreifen. Die Infos, die hier also nach außen gegeben werden, sind bis auf die Bewegungsverzögerung atomar. Das heißt in diesem Zusammenhang, dass sie sich nicht gegenseitig beeinflussen und die Manipulation eines Parameters nicht die Korrektur eines anderen nach sich ziehen muss. Dadurch fällt das Offenlegen in dieser Hinsicht nicht sehr nachteilig ins Gewicht.

Für den Fall der Bewegungsverzögerung ist es leider so, dass hier eine Kapselung in Accessorfunktionen vom Standpunnkt des Information-Hidings (vgl. [Walter99]) besser wäre. Dann müsste das Bewegungsystem allerdings komplett in `WObj` wandern, um eine wirklich sauberes Information-Hiding zu erreichen. Das würde wiederum die Flexibiltät und Komfortabilität bei der Spieleentwicklung reduzieren, da nun auch in anderen Klassen als `WLogic` Veränderungen vorgenommen werden müssten.

6.2 Implementierung der View

Die View-Komponente innerhalb eines Systems für Computerspiele ist traditionell als kritischer Teil anzusehen. Nirgendwo sonst rächen sich inperformantes Design und ebensolche Programmierung so sichtbar wie hier. Ziel ist es daher einerseits eine möglichst saubere objektortientierte Architektur zu erhalten, andererseits die hohen Performanceanforderungen an das Programm zu erfüllen.

Normalerweise wird bei Spielen eine Framerate von mindestens 25 Bildern pro Sekunde angestrebt. Auf dem Palm Vx waren diese nicht zu erreichen, jedoch immerhin ein eingermaßen flüssiges Spielgefühl mit ca. 10-15 Bildern pro Sekunde. Dabei kommt dem Entwickler das langsame Display des Palm entgegen, das bei schnellen Bewegungen träge zeichnet und die Animationen flüssiger erscheinen lässt.

Bei der Entwicklung wurden die gemachten Entwürfe immer wieder in einer Implementierung getestet, ob sie den Anforderungen entsprechen, bzw. schnell genug sind. Dabei entpuppte sich nicht etwa der Speicher wie in [Wolf00] als Problem, sondern überwiegend die Geschwindigkeit. Allerdings muss dazu auch gesagt werden, dass sehr speichersparend programmiert wurde, d.h. auf viele Speicherallokationen verzichtet und der Code kompakt gehalten wurde. Das führte dazu, dass das *Memory-Footprint* von wGame auf dem Palm bei gerade ca. 5K liegt.

6.2.1 Double Buffering: GFX

Die `GFX`-Klasse ist, wie schon im Architektur-Teil erwähnt, das zentrale Element der View. Hier werden die wesentlichen Grafikobjekte, wie Hintergrund und Sprites, referenziert. Kritische Punkte sind hierbei die Datenhaltung der Sprites und ihrer Animationen, sowie die Transformation, bzw. das Clipping, der `WObjs` in Sprites. Bei jedem Bildshirmaufbau müssen die Spielobjekte auf Darstellbarkeit überprüft werden, gegebenenfalls Sprites mit den Animationen erzeugt und gezeichnet werden.

Der erste Ansatz ist hierbei, die Datenstruktur, die die Sprites hält, möglichst einfach und damit schnell zu halten. Wie bei fast allen anderen Container-Strukturen wird ein simples Array verwendet, das in Java auch Funktionen wie `Exceptions` bei unerlaubten Zugriffen zur Verfügung stellt. Arrays haben den Vorteil, dass sie ohne Accessorfunktionen zu bearbeiten sind, was wertvolle Taktzyklen spart (vgl. [Schatzmann01]) und im Vergleich zu einem Java-Vektor auch weniger Speicher beanspruchen. Damit erfüllen sie als Datencontainer für diesen Einsatzzweck die wichtigsten Kriterien. Hinzu kommt, dass das Sprite-Array meist sequentiell

abgearbeitet werden muss, sodass hier die Verarbeitung der einzelnen Objekte durch eine einfache Schleife geschehen kann.

Hier wird auch eine zentrale Optimierungsstrategie während der Erstellung deutlich, die auch in Werken wie [Shirazi00] zu finden ist. Operationen, die während einer Schleife verarbeitet werden und damit häufig auftreten, machen eine Optimierung des Prozesses lohnend. Die notwendigen Operationen werden möglichst nicht unnötig oft wiederholt und auf die zugrundeliegenden Strukturen ist einfach zuzugreifen, was der verwendete Implementierungsansatz gewährleistet.

Als zentraler Bestandteil der Grafikfunktionalität übernimmt `GFX` auch ein Double-Buffering. Sämtliche Grafiken werden zuerst in den Puffer gezeichnet, dann in Ihrer Gesamtheit auf den Screen. Das spart Zeit beim Bildschirm-Aufbau[58] und verhindert, dass ein unangenehmes Flackern erzeugt wird, falls das Bild nicht schnell genug gezeichnet werden konnte.

Die eigentliche Zeichenoperation erfolgt mit der Methode `drawBuffer()`, in der zuerst der seperate Puffer des Hintergrundes gezeichnet wird, dann alle Sprites. Von einem Setzen von Clipping-Regionen des `Canvas` wird abgesehen, da dies nur bei niedriger Füllrate des Screens mit Sprites eine Verbesserung bringen würde. Nicht jedoch wenn auf dem Screen viele Sprites gezeichnet würden, und dadurch ein nicht zu vernachlässigender Overhead erzeugt würde. Dazu müsste entweder eine große Clippingregion, die alle Sprites abdeckt berechnet werden oder das Clipping für jedes einzelne Sprite gesetzt werden. Dabei müssten natürlich auch die letzte Position der Sprites berücksichtigt werden, um die vorherige Darstellung überzeichnen zu können. Vielleicht wäre dies jedoch eine Möglichkeit in Spezialfällen noch eine erträgliche Geschwindigkeit auf sehr langsamen Geräten zu erreichen, die auch mit wenigen Sprites schon an ihrer Leistungsgrenze angelangt sind. Hier (und leider nur in diesem Fall) würde wirklich ein signifikanter Performancezuwachs erreicht werden.

58 Der Zeichenvorgang muss nicht bei jedem Zeichnen mit den Zugriffen auf den Videospeicher synchronisiert werden, sondern nur einmal.

Der Vollständigkeit halber sei erwähnt, dass das zum Zeichnen verwendete `Displayable Canvas`, von dem GFX abgeleitet wird, auch die Tastatur-Events fängt. Diese lassen sich auch nicht durch Listener, wie in AWT oder Swing, umhängen, sonder müssen von hier aus weitergeleitet werden – in diesem Fall an eine Instanz des wGame-eigenen Input-Handlers `WInput`.

6.2.2 Hintergründe und Maps: WBkgr

Instanzen von `WBkgr` übernehmen das Zeichnen eines Hintergrundes. Hintergründe werden, wie schon erwähnt, per Tiling aus Maps generiert. So vorteilhaft die Maps in speichersparender Hinsicht sind, umso komplexer machen sie das Handling des Hintergrundes. Beim Zeichnen des Hintergrundes muss bestimmt werden, welcher Teil der Map mit welchen Tiles gezeichnet werden muss.

Zu den Strategien um diesen Umstand zu entschärfen gehört, nur bei wirklich neuen Positionen den Puffer neu zu zeichnen. Beim Verschieben des Hintergrundes (Scrollen) werden nur jene Bildbereiche gezeichnet, die tatsächlich neu sind. Dies ist etwas umständlich und damit auch nicht so einfach zu implementieren. wGame realisierte zunächst eine Implementierung, bei der der Hintergrund, bei einem Scrolling, ganz neu gezeichnet wird. Doch selbst dies war für den Palm zu langsam. Mittlerweile wird auf Grund des noch relativ üppig verfügbaren Speichers (s. Memory-Footprint in *6.2*) der komplette Hintergrund vorberechnet und mit einem entsprechenden Offset gezeichnet, was die schnellstmögliche Lösung darstellt.
Es sei jedoch angemerkt, dass dies dazu führt, dass sehr große Hintergründe zu viel Speicher beanspruchen. Hier wäre es notwendig einen anderen Weg zu gehen und den Hintergrund spalten- und zeilenweise aufzubauen.

Bei allen beschriebenen Methoden zeichnen Instanzen von `WBkgr` die Tiles erst in einen Puffer, der dann beim Zeichnen des `GFX`-Buffers gezeichnet wird.
Für zukünfitge Implementierungen soll `WBkgr` zu einer abstrakten Klasse werden. Das heißt, dass es für `WBkgr` unterschiedliche Implementierungen geben soll, die unterschiedlichen Optimierungsbedürfnissen gerecht werden. Also z.B. eine Rea-

lisierung für kleine, komplett gepufferte Hintergründe und eine andere, um große Hintergründe verarbeiten zu können.

6.2.3 Sprites: WSprite

Die Klasse `WSprite` realisiert unabhängige graphische Objekte mit Animationen, die von `GFX` erzeugt und geladen werden. `GFX` übernimmt für die Sprites eine Builder- und Verwaltungs-Funktionalität. Dies hat neben dem Prinzip des Information-Hidings den weiteren Vorteil, dass für die Sprite-Objekte ein Pooling implementiert werden kann, um das Clipping der `WObj` und Sprites zu beschleunigen. Dieses ist noch nicht implementiert, wird aber bei großen Welten und schnellen Spielen als unerlässlich angesehen, um bei häufig auftretendem Entfernen und Hinzufügen von Sprites, eine performante Ausführung zu ermöglichen, da *Heap-Allokationen* bekanntlich viel Zeit verschlingen.

Hierbei soll beim Erstellen der Sprites zunächst ein Pool auf bereits freie verfügbare Objekte überprüft und soweit vorhanden, mit den richtigen Daten gefüllt werden. Beim Löschen eines Sprites wird dieses wieder in den Pool eingehängt. Die Poolgröße ist dabei natürlich variabel und wird durch die Größe des Pool-Arrays bestimmt.

`WSprite` besitzt, um seine Animation zu realisieren, eine Referenz auf ein einfaches Array aus `Images`. Das Timing wird dabei über einen einfachen `int`-Zähler realisiert, der durch die Animationsfunktion heruntergezählt und der von `WProcess` übergebenen verstrichenen Zeit bestimmt wird. Ist er kleiner 0, wird er einfach wieder auf den Initialwert `Delay` zurückgesetzt. Ähnlich verhält es sich beim Animationsarray. Ist die letzte Animationsphase erreicht, wird auf die erste zurückgesprungen. Der Knackpunkt liegt hier, wie bei der restlichen Implementierung von wGame auch, in der möglichst einfachen Realisation.

```
public void doAnim(int deltaTime)
{
  // tempDelay schon 0?
  if (tempDelay <= 0)
  {
```

```
      // checken ob noch nicht letzte AnimPhase
      if (animCount < (animLength-1))
      {
        // animCount hochzählen
        animCount++;
        // neues animDelayTemp laden
        tempDelay = animDelay;
        currentImage = anim[animCount];
      }
      // letzte AnimPhase erreicht
      else
      {
        // animCount zurücksetzen
        animCount = 0;
        // tempDelay auf animDelay-Wert rücksetzen
        tempDelay = animDelay;
        currentImage = anim[animCount];
      }
    }
    else
    {
      // tempDelay runterzählen
      tempDelay -= deltaTime;
    }
  }// Ende doAnim()
```

Listing 6-1: `WSprite.doAnim(int deltaTime)` führt Animationen durch

6.2.4 Clipping: WClip

Clipping bedeutet in der Grafikprogrammierung allgemein, dass nicht-relevante Daten nicht berücksichtigt werden, um Rechenzeit zu sparen. Für wGame realisiert `WClip` eine Routine, um alle `WObjs` in der Spielewelt dahingehend zu untersuchen, ob sie angezeigt und als Sprites geladen werden müssen. Ebenso wird umgekehrt überprüft, ob es nötig ist, die als Sprites geladenen Objekte anzuzeigen oder zu löschen. Durch dieses Verfahren werden vor allem bei größeren Spielwelten Ressourcen und Rechenzeit gespart.

Zur Realisierung werden alle Objekte in der Spielewelt nacheinander in einer Schleife geprüft, ob sie sich innerhalb des auf dem Bildschirm dargestellten Bereichs befinden. Entsprechend werden neue Sprites geladen oder gelöscht.

6.2.5 Die MIDP Implementierung von Siemens und ihre Gameerweiterung

Der Vollständigkeit wegen möchte ich an dieser Stelle hinzufügen, dass während meiner Diplomarbeit eine Implementierung des MIDP von Siemens für das SL45i erschienen ist, die ähnliche Fähigkeiten wie wGame besitzt. Hierzu zählen ein tilebasierter Hintergrund, Sprites und eine rechteckbasierte Kollsionserkennung. Jedoch implementiert [Siemens01] kein MVC-Pattern und die Methoden sind nativ implementiert[59]. Die Möglichkeiten sind sehr auf die des Siemens SL45i zugeschnitten. Sprites und Hintergründe sind auf monochrome Grafiken[60] beschränkt und Transparenzen werden nicht über das PNG-Format geladen, sondern über Bitmasken realisiert. Tiles müssen 8*8 Pixel groß und die Spritebreite durch 8 teilbar sein. Hier ist wGame mit beliebig großen Sprites und Hintergrundtiles flexibler. Einzige Einschränkung ist die Tatsache, dass Hintergrund-Tiles quadratisch sein müssen.

Es lässt sich also sagen, dass die MIDP-Gameerweiterung auf Grund ihrer Hardwareabhängigkeit und ihrer sehr spezifischen Funktionen eher einen zweifelhaften Nutzen hat. Zukunftsträchtiger sieht hier der Ansatz von Nokia aus (vgl. [Nokia01a]), der direkten Zugriff auf die Pixeldaten innerhalb von MIDP ermöglicht, eine Vielzahl von Farbtiefen unterstützt und dem Spielerogrammierer so auch genug Freiraum für eigene Implementierungsansätze bietet. Damit lässt er sich leichter in plattformübergreifende Frameworks wie wGame implementieren.

6.3 Implementierung der Controller

Controller werden in der vorhandenen Implementierung von wGame hart codiert aufgerufen. Es ist keine Plug-and-Play-Möglichkeit vorhanden, um Controller nachträglich auf unterster Ebene einzuhängen. Grund hierfür ist, neben dem Overhead, die Frage der Notwendigkeit. Zusätzliche Funktionen können über die `WLogic`-Schnittstelle implementiert werden und es sind Callbacks aus der Game-Loop heraus vorhanden. Das hat allgemein den Vorteil hat, dass der Spielepro-

59 und damit nicht ohne weiteres einsehbar und auch nicht auf anderen MIDP-Plattformen zu benutzen

60 1Bit Farbtiefe (schwarz-weiss)

grammierer nur Wissen über das `WLogic`-Interface und nicht noch zusätzlich über ein Plug-and-Play-System benötigt. MIDP-Plattformen sind momentan sehr eingeschränkt, so dass die Frage nach dem Nutzen berechtigt erscheint.

Mit einer forschreitenden Entwicklung und steigender Verarbeitungsgeschwindigkeit der Endgeräte würde es jedoch Sinn machen, hier ein Plug-and-Play-fähiges System einzuführen. Aus der momentanen Situation heraus kann man jedoch froh sein, wenn Spiele unter MIDP einigermassen ruckelfrei laufen. Zusätzliche Grafikeffekte oder ähnliches bleiben hier erst einmal eine Wunschvorstellung.

Trotzdem ist für die nächsten Iterationsstufen der Entwicklung zumindest eine einheitliche abstrakte Klasse oder ein Interface angedacht, um die Voraussetzungen zu schaffen, Controller über ein einheitliches Basisinterface ansprechen zu können.

6.3.1 Prozesssteuerung: WProcess

`WProcess` implementiert die Game-Loop von wGame. Traditionell werden in dieser Schleife alle wichtigen wiederkehrenden Operationen innerhalb der Spielewelt angestoßen – Bewegen der Objekte, Kollisionserkennung, Berechnung der KI der Gegner und das Neuzeichnen des Spieles.

In wGame ist die Game-Loop aber keine echte Schleife, sondern ein Methode die aufgerufen werden muss. Dies hat den Grund, dass in Java das Painting und das System-Eventhandling in einem eigenen Thread laufen und dieser zu verhungern droht, wenn eine Schleife ständig durchiteriert wird (so geschehen bei einem vorhergehenden Spieleprojekt). Deshalb müssen die Spielberechnungen mit dem `paint()`-Aufruf der Java-Laufzeitumgebung in Einklang gebracht werden. Das heißt für die Implementierung, dass `paint()` den Spieleprozess startet und dieser wiederum ein `repaint()` anfrängt.

```
public void run()
  {
    // Timing setzen
```

```
    long currentTime = System.currentTimeMillis();
    deltaTime = (int)(currentTime - lastTime);
    lastTime = currentTime;
    //FPS ausgeben?
    if(WDebug.FPS)
    {
      int fps = 1000/deltaTime;
      System.err.println("FPS: " + fps);
    }
    //Tasteneingaben verarbeiten
    gameLogic.processInput(myInputHandler);
    //Objekte bewegen;
    gameLogic.doWObjects(deltaTime);
    //Objektkollisionen erkennen
    collision.checkColl();
    //run KI
    brain.doKi();
    //Clipping
    clipper.doClip();
    //Backbuffer zeichnen
    myGFX.drawBuffer(deltaTime);
    //Screen zeichnen
    myGFX.repaint();
    //Custom Code aus WLogic aufrufen
    gameLogic.customLogic();
  }//Ende run()
```

Listing 6-2: `run`-Methode von `WProcess` mit Aufruf der einzelnen Spielkomponenten

6.3.2 Inputhandling: WInput

`GFX` setzt in `WInput` entsprechende Booleans, um die Steuerungsevents zu puffern, die es nach MIDP-Spezifikation fängt. Diese können dann innerhalb der Game-Loop abgefragt werden. Somit wird eine Funtionalität implementiert, die der des Rendezvous-Patterns aus [Douglass99] gleicht. Zwei Prozesse die asynchron stattfinden werden synchronisiert: das Feuern der Eingabeevents und das Abarbeiten der Spieleloop.

6.3.3 Kollisionserkennung: WCollision

`WCollision` bietet zwei Arten der Kollisionserkennung an. Einmal, um die Objekte `WObjs` untereinander auf Kollisionen zu testen und darüber hinaus einen optimierten Satz von Methoden, um Kollisionen der Objekte mit der Hintergund-Map festzustellen.

Die Kollisionserkennung arbeitet in beiden Fällen direkt auf Modelebene, d.h. es werden alle Objekte berücksichtigt. Hier besteht die Überlegung, ob für große Spielwelten nicht eine Unterteilung der Spielobjekte in einen *Quadtree* sinnvoll wäre, um nur Objekte vergleichen zu müssen, die im selben Node registriert sind. Dies ist allerdings noch nicht implementiert.

6.3.3.1 Objekt-Objekt-Kollisionen

Aber auch so arbeitet die Kollisionserkennung sehr effizient. Es wird etwa nicht jedes Objekt mit jedem auf Kollision, aus Sicht des einzelnen Objekts, geprüft, sondern die Objekte werden global betrachtet. Dadurch müssen nur die Objekte überprüft werden, welche bei den vorigen Überprüfungen noch nicht berücksichtigt wurden. Implementiert wird diese Lösung mittels zwei ineinander geschachtelter Schleifen, was einfach und effizient ist.

6.3.3.1.1 Rechteckbasierte Kollisionserkennung

Die Kollision selbst wird mittels eines Vergleiches der in `WObj` definierten Kollisionsrechtecke erkannt. Überschneiden sich diese bei der gegenwärtigen Position der Objekte, findet eine Kollision zwischen beiden statt. Dabei ist es grundsätzlich möglich, dass das Kollisionsrechteck andere Ausmaße hat als die Anzeige durch das Sprite. Der Grund hierfür liegt darin, dass visuell sichtbare Eigenschaften, wie z.B. gezeichneter Dampf eines Objekts, dieses nicht mit einem anderen kollidieren können lassen sollen.

Die Rechteckü berprüfung selbst kann einfach und schnell über eine Art achsenbasierte Überprüfung ausgeführt werden. Dabei werden die Rechteckpunkte durch größer-als- und kleiner-als-Vergleiche auf eine mögliche Kollision überprüft.

6.3.3.1.2 Alternative Algorithmen zur Kollisionserkennung

Alternative Algorithmen zur Kollisionserkennung stellen im Bereich der 2D-Grafik vor allem pixelbasierte Algorithmen dar. Sie bieten den Vorteil, dass sie wesentlich genauer sind und sich damit genau die gewünschten Bereiche eines Objektes auf Kollisionen testen lassen. Systeme, die speziell auf Spiele ausgelegt sind, wie z.B. der Commodore Amiga, bieten spezielle Hardwarefunktionen, um

Kollisionen über einen Vergleich von Bitmasken festzustellen. Mit konventionellen Mitteln, also in Software implementiert, lassen sich solche pixelbasierten Verfahren am besten über Bit-Shifts realisieren. Die Masken-Bytes werden solange verschoben, bis sie der tatsächlichen relativen Position der Objekte zueinander entsprechen. Dann werden sie über logische Verknüpfungen miteinander verglichen. Da die pixelbasierte Kollisionserkennung wesentlich rechenintensiver ist als eine rechteckbasierte, wird in wGame bisher nur eine rechteckbasierte implementiert. Bei leistungsfähigeren Systemen wäre es aber sicher lohnenswert über eine Implementierung nachzudenken. Ausgeführt würde die pixelbasierte Erkennung ohnehin nur dann, wenn ein Rechtecktest positiv ausgefallen wäre, so dass der Performancebedarf im Rahmen des Realisierbaren bleibt.

6.3.3.1.3 Event-Behandlung bei Kollisionen zwischen Objekten

Tritt eine Kollision zwischen zwei Objekten auf, so wird die Eventmethode `objColl(int srcObj, int trgtObj)` in `WLogic` aufgerufen, um das Ereignis entsprechend zu behandeln.

6.3.3.2 Objekt-Hintergrund-Kollision

Um Hintergrund-Kollisionen festzustellen, wird für jede Bewegung in Richtung der 2D-Raumesachsen (Hoch, Runter, Rechts, Links) eine Überprüfungsmethode zur Verfügung gestellt. Diese untersuchen das übergebene Objekt dahingehend, ob es sich auf dem aktuellen Punkt aufhalten darf. Wenn nicht, wird es in die Richtung zurückgesetzt, aus der es sich herbewegt hat.

Dabei arbeitet die Hintergrund-Kollisionserkennung tilebasiert. Das heißt, dass das Kollsionsrechteck des Objekts und die darunterliegenden Werte, bzw. Felder, in der Hintergrund-Map verglichen werden.
Diese `moveCheck...(...)`-Methoden müssen allerdings nach der unmittelbaren Bewegung des Objekts in die jeweilige Richtung aufgerufen werden, da es sonst zu Fehlberechnungen kommt.

6.3.3.2.1 Event-Behandlung bei Hintergrund-Kollisionen

Trifft das Objekt auf Felder, die ein bestimmtes Ereignis auslösen, wird die Eventmethode `doTileAction(...)` in WLogic aufgerufen.Dabei wird das auslösende Objekt, die Tileposition in der Map, sowie die Tilenummer übergeben.

```
public void doWObjects(int deltaTime)
  {
    //wObjects holen
    WObj[] wObjects = gameWorld.getWObjects();

    //wObject reference cache
    WObj wObject = null;
    //wObjects bewegen
    for(int i=0; i < wObjects.length; i++)
    {
      //cache wObject[i]
      wObject = wObjects[i];
      //is "null"?
      if(wObject != null)
      {
        //verstrichene Zeit abziehen;
        wObject.tempDelay -= deltaTime;

        //tempDelay <= 0?
        if(wObject.tempDelay <= 0)
        {
          //Objekte bewegen
          wObject.x += wObject.vecX;
          if(wObject.vecX > 0)
            collision.moveCheckRight(wObject);
          if(wObject.vecX < 0)
            collision.moveCheckLeft(wObject);
          wObject.y += wObject.vecY;
          if(wObject.vecY > 0)
            collision.moveCheckDown(wObject);
          if(wObject.vecY < 0)
            collision.moveCheckUp(wObject);
          //Überlauf verstrichener Zeit von delay
                  //"abziehen"
          wObject.tempDelay += wObject.delay;
        }
      }
    }//Ende for(int i=0; i < wObjects.length; i++)
  }//Ende doWObjects()
```

Listing 6-3: Bewegen der Objekte in `WLogic.doWObjs(int deltaTime)` mit unmittelbarem Aufruf von `WCollision.moveCheck...(WObj wObject)`

6.3.4 Die Gegnersteuerung WKi

WKi täuscht dem menschlichen Spieler ein mehr oder weniger intelligentes Verhalten der Computergegner vor. Es werden nicht eine „echte“ künstliche Intelligenz, wie z.B. ein neuronales Netzwerk implementiert, sondern Funktionen, die ein typisches Verhalten eines Gegners innerhalb eines Computespieles ermöglichen.

Hierzu gehören:

Annäherung der Position des Computergegners (Non-Player-Character: NPC*)* an den menschlichen Spieler durch Manipulation des NPC-Bewegungsvektors (verfolgen)

Flucht vor dem Spieler

pseudo-zufällige Bewegungsmuster und Bemerken von Hintergrundkollisionen.

Das reicht aus, um Gegnern die Möglichkeit zu geben, den Spieler zu verfolgen, bzw. anzugreifen.

Realisiert wird diese Funktionalität mittels einer *ObjectID* und einem *State* in `WObj`. Entsprechend dieser Parameter startet die KI ihre verschiedenen Routinen, die noch durch weitere Parameter wie KI-Verzögerung und Bewegungslisten genauer spezifiziert werden können.

Für die Zukunft besteht die Überlegung, dieses Verhalten über einzelne, parametriesierbare KI-Objekte zu realisieren. Zusammen mit den `WObjs` würden sie dann kleine Statemachines bilden und ein State-Pattern aus [Gamma00] realisieren.

7 Technologie-Demonstration

Um die Tauglichkeit von wGame einschätzen und verifizieren zu können, entstand während der Entwicklung der API eine Demonstration der Fähigkeiten, die gleichzeitig auch ein Testing darstellte. Darüber hinaus stellt die Umsetzung zumindest teilweise eine Verifikation der in den theoretischen Teilen dieser Diplomarbeit gemachten Überlegungen dar.

7.1 Spielkonzept

Als Spielkonzept für einen umzusetzenden Titel wurde zunächst eine Variante des Spieles „Pac Man“/Namco gewählt, welche zeigt, dass gängige Spiele mit wGame umsetzbar sind.

Das Spielprinzip ist schnell erklärt: Der Spieler steuert einen Block-schluckenden Vielfraß durch ein Labyrinth und wird dabei von Geistern gejagt. Berührt er einen Geist verliert er ein Leben. Frißt er hingegen besondere Blöcke, wird er für kurze Zeit unbesiegbar und kann auch die Geister fressen. Eine Spielstufe ist abgeschlossen, sobald alle Blöcke im Level weggefressen sind. Verliert ein Spieler alle seine Leben ist das Spiel zu Ende.

Um eine der mobilen und drahtlosen Unterhaltung gerechtwerdenden Komponente hinzuzufügen, wurden einige Erweiterungen angedacht. So soll es in Weiterentwicklungen möglich sein, Highscores zu einem Server hinaufladen zu können. Es sollen Kooperative-Mehrspieler-Spielmodi verfügbar sein, in denen mehrere Spieler (bis ca. 4) vor den Geistern flüchten und Chatfunktionen sollen einen Community-Wert erzeugen.

Des Weiteren soll es möglich sein, eigene Labyrinthe zu bauen, die als Herausforderung anderen Spielern vorgesetzt werden könnten. Theoretisch ließe sich so auch das eigentliche Spielprinzip erweitern. Beispielsweise könnte dem Spieler die Möglichkeit eingeräumt werden, in Notfällen ein Labyrinth verlassen zu können. Dabei weiß er aber nicht, in welches Labyrinth er geworfen wird und verliert auch alle Punkte, die er im vorherigen gesammelt hat.

Nach obigem Muster ließen sich auch weitere Klassiker der Videospielgeschichte durch Netzwerkkomponenten und damit um einen Mehrwert erweitern. So wären Autorennen durch eigene Strecken erweiterbar, die dann mit anderen Spielern ausgetauscht werden könnten und in denen man auch gegeneinander antreten könnte.

7.2 Spielumsetzung mit wGame

Zur Umsetzung war auf Programmierseite dank der Funktionalitäten von wGame lediglich das Editieren der Callback-Rountinen in `WLogic` nötig, was die Entwicklung schnell und einfach machte.

Grafiken wurden im PNG-Format in der Software „Paintshop Pro“ erzeugt, und die Map für den Hintergrund im Tool „Tile Studio“ erstellt.

7.2.1 Implementierungsschritte

In *`WLogic.gameInit()`* werden die notwendigen Initialisierungen gemacht. Hierzu werden zunächst die Animationen der Spielobjekte mit *`WResourceMan.createImgSeq(...)`* geladen, eine Verzögerungstabelle für die Animationen mit einem simplen `new int[] {500, 500, 500, 500, 500}` erzeugt und mit *`myGFX.setAnims(...)`* ins Grafiksubsystem eingehängt. Mit Aufrufen wie *`new WObj(50, 50, 0, PLAYERID, 0)`*, werden die Spielobjekte erzeugt und mit *`WWorld.addWObj(...)`* in die Spielewelt eingehängt.

Schließlich wird mit *`WResourceMan.createMap(MAPWIDTH, MAPHEIGHT, "demo1map.txt")`*, die Hintergrund-Map geladen und mit *`myGFX.createWBkgr(bkgrmap[0].length, bkgrmap.length, TILESIZE, bkgrmap, bkgrTiles)`* schließlich der Hintergrund erzeugt und in GFX eingehängt. Mit dem Setzen der Map-Referenzen in der Kollisionserkennung mit *`collision.setMap(bkgrmap)`* und der Spielewelt mit *`gameWorld.setMap(bkgrmap)`* und einer Timerinitialisierung ist das Spiel auch schon theoretisch bereit zum Spielstart.

Es müssen aber noch die für den eigentlichen Spielverlauf und die Interaktivität relevanten Callbacks gesetzt werden. Die Methode `objColl(int srcObj, int trgtObj)` dient zur Behandlung von Kollisionen der Spielobjekte untereinander, also beispielsweise Gegner und Spieler. Sie ist aber bisher noch nicht ausprogrammiert und erzeugt lediglich Debug-Output. `doTileAction(...)` verarbeitet hingegen Ereignisse, die auftreten, wenn ein Spielobjekt mit einem Hintergrundelement (*Tile*) kollidiert. Hier wird momentan nur mit `if(tileAction[tileNum] == PILL)` für das Spielerobjekt überprüft, ob es sich um eine Pille handelt, entsprechend wird dann der Hintergrund mit `tempMap[tileY][tileX] = NOP` neu gesetzt und dieses Tile mit `myGFX.renderTile(tileX, tileY, NOP)` gezeichnet.

In `WLogic.customLogic()` wird die Spielelogik implementiert, die zyklisch, also einmal pro Spielberechnung und Bildaufbau, durchgeführt wird. Hier ist momentan nur das Scrolling eingebettet, das immer dann ausgelöst wird, wenn sich der Spieler den Randbereichen des Bildschirms nähert. In `processInput(WInput myInputHandler)` wird der Inputhandler auf seinen Zustand hin abgefragt und der Spieler entsprechend bewegt. In `doWObjects(int deltaTime)` werden schließlich die Spielobjekte in Abhängigkeit der verstrichenen Zeit und ihres Bewegungsvektors bewegt.

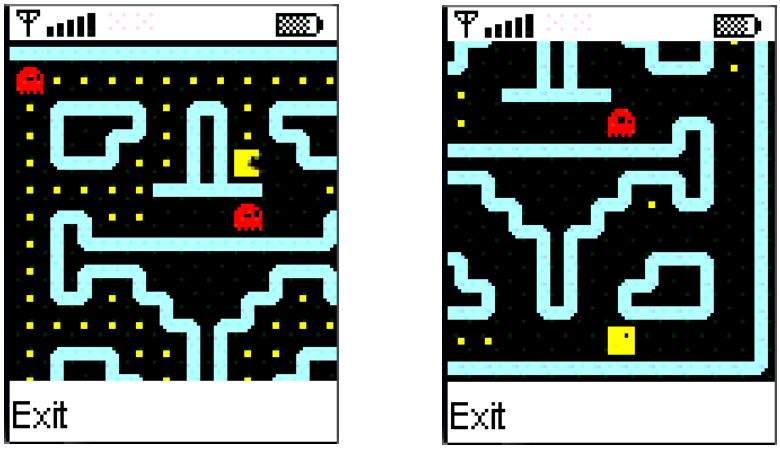

Abbildung 7-1: Screenshots des Spieleprototypen

Mit dieser Implementierung ist der Prototyp des Spieles schon komplett. Wie die Umsetzung gezeigt hat, sind zentrale Spielfuntkionen mit wGame einfach zu rea-

lisieren. Kapitel 8 geht detailiert auf die Erfahrungen bei der Entwicklung von wGame und der Technologie-Demonstration, bzw. des Spieleprototypen, ein.

8 Auswertung der praktischen Arbeit

Das folgende Kapitel behandelt die Erfahrungen, die während der praktischen Entwicklung gesammelt wurden. Dazu gehören neben Implementierungsstrategien und -vorgehensweisen, auch die Umsetzbarkeit der in Kapitel 3 erstellten Thesen zur Spielgestaltung.

8.1 Ergebnisse der Implementierung von wGame

Die Implementierung der Spiele-API zeichnete sich zunächst durch zahlreiche Hürden ab, die zu überwinden waren. Hierzu gehörten die wenig ausgereiften Entwicklungstools, von denen einige z.B. kein vernünftiges Debugging ermöglichten. Außerdem fielen die verfügbaren Basisinformationen sehr mager aus. Bis auf allgemeine Werke zur MIDP-Implementierung, war eigentlich nur die Diplomarbeit [Wolf00] wirklich hilfreich, da sie die anfängliche Vermutung, auf mobilen Devices möglichst minimalistisch zu entwickeln, bestätigte.

Mit der Implementierungsstrategie, eine möglichst einfache Architektur zu schaffen und ebenso einfach zu realisieren, lief der Entwicklungsprozess relativ reibungslos ab. Dazu trug auch die hohe Iterationsrate im spiralförmig verlaufenden Entwicklungsprozess bei. Bis auf kleinere und wohl auch übliche Zwischenfälle in der Softwareentwicklung, gab es keine grösseren Pannen oder Rückschläge. Das iterative Vorgehen erwies sich vor allem bei den performancekritischen Teilen als ebenso effizient wie notwendig. Wären die gemachten Optimierungen in der Viewkomponente nicht gemacht worden, würde die Darstellung auf dem PalmVx wohl immer noch zu stark ruckeln und damit entwickelte Spiele wären praktisch unspielbar.

Die Realisation als MVC-Pattern erwies sich auch in Hinblick auf die Performance und Optimierungsfähigkeit des Systems als sehr nützlich. Dadurch konnten die rechenintensiven Teile besser und einfacher optimiert werden. Dies belegt in diesem Zusammenhang auch klar, dass performanz-kritische Systeme durch Objektorientierung effizienter entwickelt werden können.

Der performanceorientierte, wenn auch relativ aufwändige Entwicklungsprozess zahlte sich jedoch aus und ist für ähnlich zeitkritische Projekte ebenfalls zu empfehlen, um die gestellten Anforderungen erfüllen zu können.

8.2 Ergebnisse der Implementierung der Technologie-Demonstration

Die Realisationsmöglichkeit einer Variante eines Klassikers wie „Pac Man“ weist die Fähigkeit nach, dass relativ einfache Spiele mit Java und dem MIDP umsetzbar sind. Damit wird wGame den im theoretischen Teil gestellten Anforderungen gerecht, einfache, leicht zugängliche Spiele für mobile drahtlose Plattformen entwickeln zu können.

Zumindest von umsetzungstechnischer Seite konnten die im ersten Teil der Diplomarbeit gemachten Aussagen zur Spielgestaltung bestätigt werden. Bei den Grafiken ließen sich die behandelten Gestaltungsprinzipien umsetzen – Reduktion der Charaktere auf wesentliche Merkmale und einfache pixelweise Erstellung. Die Spielobjekte wirken damit der geringen Auflösung in der Stärke ihrer Ausdrucksweise entgegen.

Das einfache Spielprinzip ermöglicht einen einfachen Zugang zum Titel, auch wenn es sich dabei um eine Neuumsetzung eines bestehenden Spielprinzips mit Erweiterungen auf ein neues Format handelt.

Gleichzeitig lässt sich mit der Technologie-Demonstration sagen, dass wGame in der Lage ist diese Anforderungen dahingehend zu erfüllen, als dass es dem Software-Engineer die Entwicklung eines Spieletitels erleichtert. Damit ist auch nachgewiesen, dass eine Entwicklung nach professionellen Kriterien zumindest in der Hinsicht möglich ist, als dass durch eine einheitliche API für verschiedene Spielgenre, ein effizientes Verfahren im Entwicklungsprozess möglich wird.

8.3 Geplante Verbesserungen

Was sich im Nachhinein als etwas fragwürdig herausstellte, ist die Tatsache, dass der Zugriff auf Spielobjekte und Sprites meist über den Index des Arrays, in dem sie gespeichert sind, gemacht wird. Dies führt zwar dazu, dass Referenzen auf einzelne Objekte so etwas unwahrscheinlicher durch die Software verstreut gespeichert werden, aber andererseits ist dieser Ansatz wenig objektorientiert. Hier müssen sicherlich erst noch einige Praxiserfahrungen mit der API gesammelt werden, bevor diese Frage konkreter Beantwortet werden kann.

Ebenfalls wäre eine einheitliche abstrakte Klasse für die Controller und ein dynamisches Management derselben wünschenswert. Dadurch könnten zusätzliche Effekte oder Funktionalitäten dynamisch eingehängt werden. Tendenziell sollte deshalb in Zukunft allgemein etwas objektorientierter entwickelt werden, ohne dabei jedoch über das Ziel hinauszuschießen. So macht es sicherlich nur wenig Sinn, die Datenhaltung in Java-Vektoren zu organisieren, vielmehr ist hier vermehrt der Einsatz von abstrakten Klassen oder Interfaces gemeint, um eine einheitliche Implementierung zu gewährleisten.

Des Weiteren fehlen noch ein paar Kleinigkeiten, wie eine Utility-Klasse, um Spieldaten, wie die Anzahl der Punkte und der Leben, darstellen zu können. Eine Highscore-Liste wäre sicherlich ebenso für eine Vielzahl von Spielen sinnvoll, wie die Möglichkeit ein spezielles Menüsystem darstellen zu können. Dieses wäre grafisch anpassungsfähiger als die vom MIDP gebotenen Möglichkeiten und würde damit die Atmosphäre für den Spieler verdichten.

Prinzipiell wurde jedoch das Ziel der Diplomarbeit „Mobile und Wireles Entertainment“ erreicht. Zukünftige Entwicklungen auf dem mobilen Markt werden weitere Erkenntnisse über die Tauglichkeit von wGame liefern. Eine Neuentwicklung mit den in der Implementierung von wGame bisher fehlenden Features, wie Netzwerk- und Soundunterstützung, wird angestrebt. Trotz der zahlreichen Verbesserungsmöglichkeiten lässt sich resumieren, dass sich unter den einge-

schränkten Voraussetzungen der mobilen Endgeräte, Spiele nach den in Kapitel 3 erstellten Kriterien entwickeln lassen.

9 Ausblick

Die gemachten Betrachtungen und Erfahrungen, in theoretischer wie in praktischer Hinsicht, konnten einen ersten Eindruck vermitteln, in welche Richtung sich Mobile und Wireless Entertainment bewegt. Auch wenn sich der Markt noch sehr dynamisch verhält, lässt sich doch etwas über den grundlegenden Charakter der interaktiven Unterhaltung auf mobilen Clients sagen: Die Einfachheit der Endgeräte und deren eingeschränkte Möglichkeiten, werden wohl noch einige Zeit bestehen bleiben, vor allem im Vergleich zu Desktop-Systemen. Die Spielinhalte an sich leiden darunter aber nicht unbedingt, da auf Grund des Mobilitätscharakters der Plattform tendenziell eher einfache und leicht zugängliche Spiele interessant sind, die nicht den selben Ressourcenhunger haben, wie die neusten 3D-Shooter auf PCs oder Videospielkonsolen.

9.1 Zukunftsperspektiven von wGame

wGame oder Spiele-APIs für mobile Clients allgemein, sehen sich einerseits damit konfrontiert, hohe Leistung auf minimaler Hardware zur Verfügung zu stellen, andererseits müssen komplexe Funktionen zur Abstraktion von proprietären Erweiterungen der Geräte-Hersteller und eine leistungsfähige Netzwerkunterstüzung zur Verfügung gestellt werden. Eine Spiele-API wie wGame wird auch weiterhin einen Großteil der Entwicklungsressourcen darauf verwenden, diese Funktionen bei hoher Performanz zur Verfügung stellen zu können.

9.2 Zukunftsperspektiven von Mobile und Wireless Entertainment

Technisch stehen die mobilen Clients zwar noch hinter dem GameBoy zurück, die dynamische Entwicklung lässt jedoch darauf hoffen, dass sich dieser Rückstand mittelfristig verringert – oder sogar umkehrt, vergleicht man die Leistungsfähigkeit eines „High-End“-Clients, wie die des iPAQs mit der des GameBoy Advance.

Der Spieleentwickler und die Anwender dürfen sich also auf eine steigende Performance freuen, die die momentanen spartanischen Möglichkeiten verbessern

dürfte. Dadurch ist es wiederum möglich, die Spiele attraktiver zu machen und damit einen größeren Teil an Nutzern anzulocken.

Ebenso dürften sich mittelfristig die mangelnden Sound- und Netzwerkfähigkeiten verbessern. Entweder durch eine entsprechende MIDP-Weiterentwicklung und herstellerspezifische Erweiterungen, oder durch native Entwicklungen auf einer dominanten Plattform – sei es nun PalmOS, Windows CE oder Symbian OS. Hier machen es die unterschiedlichen Implementierungs-Voraussetzungen sogar noch sinnvoller, eine einheitliche abstrakte Schnittstelle in Form einer Spiele-API zu schaffen, um mehrere Plattformen unterstützen zu können.

Doch auch inhaltlich wartet auf Spieleentwickler noch einiges an Arbeit. So wollen die neuen Möglichkeiten, die sich durch die Wireless-Komponente ergeben, erst noch ausgeschöpft werden.

Abschließend lässt sich sagen, dass sich die Welt des Mobile und Wireless Entertainment einerseits noch stark entwickelt, sich andererseits ein Grundcharakter abzeichnet, der sicherlich noch einige Zeit bestehen bleiben wird. Die ständige Weiterentwicklung in diesem Bereich lässt die Zukunft für mobile und drahtlose Spiele weiterhin spannend erscheinen.

9.3 Danksagung

An dieser Stelle möchte ich meinen Dank all denen aussprechen, die mich während meiner Diplomarbeit tatkräftig unterstützt haben. Insbesondere gilt er meinen Betreuern an der FH Furtwangen, Prof. Wilhelm Walter und Prof. Dr. Ullrich Dittler, sowie Peter Rudolph von Wearix. Gedankt sei an dieser Stelle auch meinen Eltern, Lovorka und Peter Schmidt, und meiner Freundin Mandy für die Unterstützung während des gesamten Studiums. Auch allen anderen vielen Dank!

A Literaturverzeichnis

[IDA01] Institut für Demoskopie Allensbach; Allensbacher Computer- und Telekommunikations-Analyse 2001 – Sonderzielgruppen Kernzielgruppe Mobilfunk; erhältlich unter http://www.acta-online.de/; Allensbach 2001

[Bates01] Bates, Bob; Game Design: The Art & Business of Creating Games; Roseville, Kalifornien 2001

[Bewersdorff01] Bewersdorff, Jörg; Glück, Logik und Bluff – Mathematik im Spiel – Methoden, Ergebnisse und Grenzen; Braunschweig/Wiesbaden 2001

[DeLoura00] DeLoura, Mark; et al; Game Programming Gems; Hingham, Massachusetts, USA 2000

[DeLoura01] DeLoura, Mark; et al; Game Programming Gems 2; Hingham, Massachusetts, USA 2001

[Douglass99] Douglass, Bruce Powel; Real-Time UML Second Edition – Developing Efficient Objects for Embedded Systems; MA, USA; 1999

[EDGE01a] EDGE; Portable Pleasure: Handeld Generations, erschienen in EDGE #104, S.52; Bath, UK 2001

[EDGE01b] EDGE; Cash-U cashes in; erschienen in EDGE#101, S.115; Bath, UK 2001

[EDGE01c] EDGE; FAQ: Martin Chudley – MD, Bizarre Creations; erschienen in EDGE#101, S.124; Bath, UK 2001

[EDGE01d] EDGE; inbox; erschienen in EDGE#105, S.121; Bath, UK 2001

[EngineSoftwareXX] Engine Software; Gameboy Advance Music Replayer; http://www.engine-software.nl/Replayer/; Doetinchem, Niederlande o.J.

[Fox01] Fox, David; Creating Games using J2ME; http://www.gamasutra.com/resource_guide/20010917/fox_pfv.htm; San Francisco, Kalifornien 2001

[Fritz97] Fritz, Jürgen; Fehr, Wolfgang (bd. Hrsg.); Handbuch Medien: Computerspiele; Bonn; 1997

[FritzXX] Fritz, Jürgen; Fehr, Wolfgang; Computerspiele auf dem Prüfstand – Netzwerk-Spiele; Bonn, o.J.

[Frohwein02] Frohwein, Jeff; devrs.com; www.devrs.com; o.O. 2002

[Gamma00] Gamma, Erich et al; Design Patterns – Elements of Reusable Object Oriented Software; Boston, San Francisco, New York, Toronto, Montreal, London, Munich, Paris, Madrid, Capetown, Sidney, Tokyo, Singapore, Mexiko City 2000

[Giguère00] Giguère, Eric; Java 2 Micro Edition – The Ultimate Guide to Programming Handheld and Embedded Devices; New York, Chichester, Weinheim, Brisbane, Singapore, Toronto 2000

[Harbour01] Harbour, Jonathan S.; Pocket PC Game Programming: Using the Windows CE Game API; Roseville, California 2001

[Hentrich97] Hentrich, Günter; Skript zur Vorlesung Medienpsychologie; S.11; Furtwangen 1997

[Hübner01] Hübner, Katja; Mobiles Internet; erschienen in screen business online 2001, Ausgabe 03, S.34-39; Hamburg 2001

[Kauert01] Kauert, Oliver; „Game Forge“ – Entwicklung eines Game SDK; Furtwangen 2001

[Kelland01] Kelland, Matt; Postmortem: Ngame's Chop Suey Kung Fu; http://www.gamasutra.com/resource_guide/20010917/kelland_01.htm; San Francisco, Kalifornien 2001

[Loki01] Loki Software, Inc.; Hall, John R.; Programming Linux Games; San Francisco 2001

[MPEGXX] Moving Picture Experts Group; Startseite; http://mpeg.telecomitalialab.com/; Turin, Italien o.J

[Nokia01a] Nokia Mobile Phones; Nokia UI API v0.9 (draft); http://www.forum.nokia.com/javaforum/main/1,6668,1_0_10,00.html; Espoo, Finnland 2001

[Nokia01b] Nokia Mobile Phones; Imaging Phones; http://www.forum.nokia.com/main/1,35452,1_48_70,00.html; Espoo, Finnland 2001

[NokiaXX] Nokia Deutschland; Bluetooth, Weltsprache für drahtlose Kommunikation; http://www.nokia.de/firmen/technik/bluetooth/; o.O. 2001

[Petchel01] Petchel, Thomas; Java 2 Game Programming; Roseville, California 2001

[Puha01] Puha, Thomas; Wireless Entertainment: The State of Play; http://www.gamasutra.com/resource_guide/20010917/puha_pfv.htm; San Francisco, Kalifornien 2001

[Renner97] Renner, Michael; Spieltheorie und Spielpraxis – Eine Einführung für pädagogische Berufe; Freiburg 1997

[Riggs01] Riggs, Roger; et al; Programming Wireless Devices with the Java 2 Platform, Micro Edition; Massachusetts, USA 2001

[Rink02] Rink, Jürgen; PDA-Markt in Europa bleibt auf niedrigem Niveau; erschienen in c't 2002, Heft 4, S. 24; Hannover 2002

[Rollings00] Rollings, Andrews; Morris, Dave; Game Architecture and Design; Scottsdale, Arizona 2000

[Seppänen01] Seppänen, Lasse; Designing Mobile Games for WAP; http://www.gamasutra.com/resource_guide/20010917/seppanen_03.htm; o.O. 2001

[Shirazi00] Shirazi, Jack; Java Performance Tuning; Sebastopol, Kaliforniern 2001

[Siemens01] Siemens Mobile; http://www.siemens-mobile.de/btob/CDA/presentation/ap_btob_cda_presentation_frontdoor/0,,28,FF.html; München 2001

[Steinke00] Steinke, Lennart; Spiel mit mir! Java Arcade Spiele und wie man sie erstellt; erschienen in Java Magazin 9/2000-12/2000; Frankfurt a.M. 2000

[Schatzmann01] Schatzmann, James; Donehower, Roy; "Hindernisse auf der Überholspur – Teil 1: Performanzprobleme" in: Java Spektrum 5/2001, S.38; Troisdorf 2001

[Sun01] Sun Microsystems Inc.; Java Wireless Developer Homepage; www.java-soft.com/wireless; o.O. 2002

[Sun02] Sun Microsystems Inc.; JSR 118 – Mobile Information Device Next Generation; http://www.jcp.org/jsr/detail/118.jsp; o.O. 2002

[Symbian00] Symbian Ltd.; Allin, Jonathan; Writing Optimised Code for Constrained Devices; Teil der „Crystal C++ 6.0 SDK“-Online-Dokumentation; London 2000

[VUD01] Verband der Unterhaltungssoftware Deutschland e.V.; Der Markt der Unterhaltungssoftware (Konsole) – Januar bis September 2001; Paderborn 2001

[Walter97] Walter, Wilhelm; Arbeitsmaterial zur Vorlesung Informatik 2; Furtwangen 1997

[Walter99] Walter, Wilhelm; Arbeitsmaterial zur Vorlesung Programmieren 2; Furtwangen 1999

[Wolf00] Wolf, Christian; Programmierung und GUI Design eines auf XML basierenden Clients für die Java K Virtual Machine eines Palm Connected Organizers; Furtwangen 2000

B Glossar

2D-Shooter: Schießspiel mit zweidimensionaler Grafik.

3D-Engines: siehe *Spiele-Engine*

Accessormethoden: Methoden um auf einzelne Daten des Objektes zugreifen zu können.

API: Application Programming Interface – Funktionen die zur Erstellung einer Software benötigt werden.

Arcade: Englische Bezeichnung für Spielhalle

Billing: Abrechnung von an den Kunden verkaufter Inhalte.

Buttonabstraktion: *Buttons* werden für den Spieleprogrammierer abstrakt dargestellt, d.h. er frägt nicht spezielle Codes für eine bestimmte Taste der Hardware ab, sondern programmiert auf einer abstrakten Zwischenebene.

Button: Steuerknopf an einem Spieleingabegerät, wie z.B. Steuerkreuz oder Tastatur.

Client: siehe Client/Server

Client/Server: Das Client-Server-Prinzip beschreibt Netzwerke, in denen ein als Server bezeichneter Computer die Daten zur Verfügung stellt, die vom als Client bezeichneten Rechner benötigt werden. Die so genannte Peer-to-Peer-Vernetzung bildet dieses Verhältnis beidseitig nach. Das heißt, ein Client ist gleichzeitig auch Server für etwaige andere Rechner im Netz und umgekehrt.
Für die Spieleentwicklung sind Peer-to-Peer und Client/Server die zentralen Modelle für vernetztes Spielen, wobei auch Mischformen möglich sind, in denen

z.B. ein Server die Netzwerk-Adressen vermittelt, die eigentliche Kommunikation der Rechner untereinander dann aber direkt, also Peer-to-Peer, stattfindet.

Communities: Virtuelle Gemeinschaft, in der Mitglieder mittels Benutzung elektronischer Medien kommunizieren können.

FPU: Floating Point Unit, Einheit zur Durchführung von Gleitkommaarithmetik.

First-Person-Shooter: Schießspiel mit 3D-Darstellung aus der Ich-Perspektive.

Framework: Stellt über die Funktonalität einer *API* hinausgehende Voraussetzungen für die Softwareentwicklung. Neben der Bereitstellung von Funktionen, bietet ein Framework auch die Grundlagen für die Struktur der Software und somit eine weitreichendere Unterstützung bei der Softwareerstellung.

Game-Balancing: Bezeichnet den Vorgang bei der Spielentwicklung, in dem die einzelnen Spielbestandteile in ihrer Auswirkung auf das Spiel gegeneinander ausgewogen werden. Das sind neben grundlegenden Dingen, wie Geschwindigkeit des Spielers und der Gegner und zur Verfügung stehende Zeit, auch Gegnerverhalten- und/oder intelligenz, sowie evtl. verfügbare Ressourcen. Ziel des Game-Balancings ist eine aus Entwicklersicht angemessene Schwierigkeit des Spiels entsprechend der Zielgruppe und *Level/Spielstufe*.

Game-Loop: Schleife, die zur Berechnung des Spieles kontinuierlich abgearbeitet wird.

Gameplay: Spielprinzip

Handheld: Bezeichnet Computergeräte, die in der Hand gehalten werden. Im Kontext dieser Diplomarbeit bezeichnet Handheld, wenn nicht anders erwähnt, mobile Spielkonsolen.
Heap-Allokation: Speicher für Objekte wird in Java grundsätzlich über den so genannten Heap reserviert, anders als in C/C++, in dem auch Stack-Speicher

verwendet werden kann. Diese Reservierung beansprucht Zeit. Stack-Speicher ist schneller zu Reservieren als Heap-Speicher.

Highscore: Bestenliste eines Computerspiels. Spiele, wie z.B. „Pulleralarm – das Spiel zu tv total“ von westka interactive, nutzen Highscores im Internet, um Kommunikationsanreize innerhalb einer *Community* zu bieten und die Spielmotivation zu steigern.

Jump'n'Runs: Auch als Hüpfspiel bezeichnetes Geschicklichkeitsspiel, bei dem es darum geht, beliebig große Levels zu durchlaufen und dabei verschiedene Gegner und Hindernisse zu umgehen.

LCD: Liquid Crystal Displays (LCDs) sind in der mobilen Welt weit verbreitet, da sie im Gegensatz zu Kathodenstrahl-Röhren, wie bei konventionellen Fernsehern oder Monitoren, kleiner und flacher sind und weniger Strom verbrauchen.

Line-Up: Produkt-Portfolio eines Herstellers, im Spielebereich gewisse Anzahl von Spielen, die bestimmte Spielkategorien abdecken.

Listener: Softwareteil der aufgerufen wird, wenn bestimmte Events/Ereignisse erzeugt wurden.

Level/Spielstufe: Spielstufe in einem Spiel, d.h. Phase im Verlauf des Spieles. Normalerweise nimmt die Schwierigkeit eines Spieles während seines Verlaufes und mit steigender Spielstufe zu. Spielstufen bilden eine Unterteilung des Spieles in einzelne Abschnitte.

Map: Ein 2D-Array, das zur Speicherung von Hintergrundattributen wie Darstellung und Eigenschaften dient.

Memory Footprint: Größe des belegten Speichers zur Laufzeit eines Programms.
Multiplay: Mehrspieler-Möglichkeit

Multiplayer: s. Multiplay.

Organizer: s. PDA

Pad: Eingabegerät oder -feld. Zum Beispiel Game- und Joypad (Joystick-ähnliches Spielgerät, mit Steuerkreuz statt -knüppel) oder numerisches Pad (numerisches Tastenfeld auf Handies oder der Computer-Tastatur).

PDA: Personal Digital Assistant, Mobilgerät um Termine, Aufgaben usw. zu verwalten. Name durch Apples „Newton“ geprägt, aber erst durch Palm wurden PDAs zu einem weit verbreiteten Produkt. Mit Microsofts PocketPC werden auch vermehrt Multimedia-Funktionen möglich.

Peer-to-Peer: s. Client/Server

Polling: Im Gegensatz zum *Listener* wird beim Polling selbständig überprüft, ob Events vorliegen.

Pooling: Verfahren um die Allokation von Objekten zu beschleunigen. Dazu werden nicht mehr benötigte Objekte vorerst nicht wieder freigegeben, sondern gepuffert, um, falls Objekte des selben Typs wieder benötigt werden, diese herausgeben zu können.

Quadtree: Im Gegensatz zu einem Binärbaum mit zwei Nachfolgern pro Knoten, besitzt ein so genannter Quadtree vier Nachfolger.

Screen: Im Bezug auf Spiele gleichbedeutend mit „Anzeige“ oder „Bildschirm“.

Scrolling: Also Scrolling wird generell das Bewegen eines Bildbereichs bezeichnet. In Spielen wird in der Regel meist der Hintergrund „gescrollt“. Dadurch werden Spielflächen möglich, die größer sind als der darstellbare Bereich der Anzeige.

Server: sieher Client/Server

Spiele-Engine: Eine Spiele-Engine stellt eine Art Basissoftware für die Programmierung von Computerspielen dar. Je nach Ausprägung ist sie am ehesten mit einer *API* oder einem *Framework* vergleichbar.

Splitscreen: Bei Mehrspieler-Spielen Möglichkeit mehrere Spieler an einem Spiel simultan zu beteiligen. Dabei wird der Bildschirm in zwei oder mehr eigenständige spielerabhängige Bereiche unterteilt.

Sprites: Kleine grafische Objekte, die sich unabhängig vom Hintergrund bewegen lassen.

Smartphone: Mobilgerät, das sowohl Fähigkeiten eines Mobiltelefons als auch eines *PDAs* besitzt.

Tile: Grafisches (meist quadratförmiges) Element, aus dessen Vielzahl andere Grafiken, wie z.B. Hintergründe, gebaut werden.

Touchscreen: Berührungsempfindliche Anzeige, über die eine mausähnliche Steuerung vorgenommen werden kann.

C Verwendete Software-Werkzeuge

Für den Klassenentwurf und das Design der Architektur wurde das Tool „Together“ von TogetherSoft eingesetzt. Für die weitere Programmierung wurde zunächst „JBuilder 5“ von Borland und das „MobileSet Nokia Edition“ verwendet. Da das Debugging mit dem mitgelieferten MIDP-Emulator jedoch nicht richtig funktionierte, wurde auf „Forte for Java 3.0“ und „J2ME Wireless Toolkit 1.0.3“ von Sun umgestellt, womit nun endlich Breakpoints richtig funktionierten. Für die Versionsverwaltung wurde das nach GNU-Lizenz erhältliche „CVS“ eingesetzt.

Für die grafische Erstellung des Spiele-Prototypen kamen das Werkzeug „TileStudio“ von Wiering Software und „Paintshop Pro“ von JASC zum Einsatz. Während PaintShop Pro vor allem bei der Gestaltung der einzelnen Sprites und Hintergrund-Tiles Anwendung fand, wurde mit TileStudio der Hintergrund aus den einzelnen Tiles zusammengebaut.

Zum Schreiben der Diplomarbeit, und schließlich auch für den Satz, wurde „StarOffice 5.2“ von Sun verwendet, als Betriebssystem „Windows 2000 Professional“ von Microsoft.

D wGame-API und Technologie-Demonstration

E Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich, Boris Schmidt, geboren am 25.02.1977 in Heilbronn:

meine vorliegende Diplomarbeit mit dem Titel „Mobile und Wireless Entertainment – Eine Implementierung am Beispiel von Java 2 ME und MIDP“ unter Betreuung von Prof. Dipl.-Inf. Wilhelm Walter, Prof. Dr. Ullrich Dittler und Dipl.-Inf. Peter Rudolph selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als die in der Abhandlung aufgeführten Hilfen benutzt habe.

die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben kann.

Heilbronn, den 28.02.2002

Die Informationen in dieser Diplomarbeit wurden mit größter Sorgfalt aufbereitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Die FH Furtwangen, Wearix GmbH und der Autor übernehmen keine juristische Verantwortung oder irgendeine Haftung für eventuell verbliebene Fehler und deren Folgen. Alle Warenzeichen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen. Der Autor richtet sich im Wesentlichen nach den Schreibweisen der Hersteller.