Technische Universität Berlin
Fakultät IV - Elektrotechnik und Informatik
Lehrstuhl Informatik und Gesellschaft

Diplomarbeit

Spiele-Software und Open Source

Niels Weber

31. März 2006

Prüfer: Prof. Dr. iur Bernd Lutterbeck
Zweitprüfer: Prof. Dr. Hermann Krallmann

Danksagung

Ich danke allen, die mir beim Erstellen dieser Diplomarbeit zur Seite gestanden und mich unterstützt haben. Zunächst sind hier meine Prüfer Prof. Dr. iur Bernd Lutterbeck und Prof. Dr. Hermann Krallmann zu nennen, die mir die Bearbeitung des Themas ermöglicht haben. Ein besonderer Dank an Matthias Bärwolff, der mich während der Arbeit begleitet hat, für seine inhaltlichen und formalen Hinweise und Korrekturen.

Danke auch an Sinja, die mich in den letzten Monaten täglich zur Weiterarbeit ermuntert und ermahnt hat. Ohne Dich wäre ich wohl nie fertig geworden.

Als letztes, aber nicht zuletzt, möchte ich mich bei meinen Eltern bedanken, die mir durch ihre Unterstützung das Studium überhaupt ermöglicht haben. Danke für Eure Geduld und Eure Hilfe.

Berlin, 31. März 2006, Niels Weber

Kurzzusammenfassung

Die folgende Arbeit untersucht, warum Open-Source-Prinzipien im Bereich der Spiele-Software bislang nur sehr wenig eingesetzt werden und wie sich dies ändern kann. Hierfür werden zunächst die Grundlagen beider Gebiete dargestellt. Es wird gezeigt, dass die größte Schwierigkeit bei der Anwendung von Open Source auf Spiele darin besteht, dass Spiele weniger Software und mehr Filmen gleichen. Anhand von sechs Beispielen wird demonstriert, wie dennoch Open-Source-Prinzipien zur Umsetzung gelangen können und wie die Beispiele von dieser Offenheit profitieren konnten. Gesondert wird auf das neu entstandene Gebiet der Online-Spiele eingegangen, welches besondere Chancen für eine Umsetzung als Open Source bietet. Abschließend wird untersucht, welche Lizenzen und welche Finanzierungsmodelle in der Spieleentwicklung angewendet werden können, so dass das resultierende Spiel möglichst offen ist, ohne dabei die Wirtschaftlichkeit aus den Augen zu verlieren.

Inhaltsverzeichnis

1 Einleitung

> „We do not quit playing because we grow old, we grow old because we quit playing.“
>
> – *Benjamin Franklin*

1.1 Motivation

Open Source als Methode der Softwareentwicklung hat sich in den letzten Jahren zu einer ernsthaften Alternative zu den klassischen Modellen entwickelt. Während das Phänomen Open-Source-Software in fast allen Anwendungsbereichen eine wichtige Bedeutung erlangt hat, bleibt ein Bereich bisher fast völlig frei von kommerziell entwickelter Software mit offenem Quellcode. Dies ist der Bereich der Computerspiele. Warum dies so ist, ist bisher nur oberflächlich untersucht worden. Dabei ist gerade der Markt für Computerspiele sehr groß.

Der Gesamtumsatz im Spielebereich übersteigt den der Filmindustrie und ist damit der größte Teilbereich der Unterhaltungsindustrie. Auch die Wachstumsraten übersteigen die anderen Bereiche der Unterhaltungsindustrie. Zusätzlich nimmt die Verbreitung von Spielesoftware in Geräten wie Mobiltelefonen oder Satellitenreceivern ständig zu, so dass mit weiterem Wachstum zu rechnen ist. Eine Studie von Pratchett (2005) im Auftrag der BBC ergab, dass 59% der britischen Bevölkerung zwischen 6 und 65 Jahren zumindest gelegentlich Computer- oder Videospiele spielen. Dabei liegen die Anteile bei jüngeren Jahrgängen noch deutlich über diesen Werten. Auch dies verdeutlicht die aktuelle und zunehmende Bedeutung von elektronischen Spielen.

Derzeit ist der Betriebssystem-Markt im Anwenderbereich eine Monokultur. Weil Computerspiele eine der wichtigsten Anwendungen in der privaten Computernutzung sind, hat eine Verfügbarkeit von Spielen auch einen großen Einfluss auf die Verbreitung von alternativen Betriebssystemen im Heimbereich. Ein offener Quellcode erleichtert die Portierung auf andere Plattformen, denn nur wer im Besitz des Quellcodes ist, kann eine solche durchführen. Open-Source-Spiele und die Nutzung von Open-Source-Betriebssystemen und anderer Software stehen damit in einem di-

rekten Zusammenhang. Insofern trägt ein offener Quellcode von Spielesoftware zur Vielfalt auf dem Computermarkt bei.

1.2 Problemstellung

Computerspiele scheinen sich grundsätzlich von anderer Software zu unterscheiden. Dies scheint dazu zu führen, dass es besondere Schwierigkeiten bei der Entwicklung von Open Source Spielesoftware gibt. In dieser Arbeit wird herausgearbeitet, worin diese Schwierigkeiten bestehen. Zusätzlich werden Ansätze gezeigt, wie trotz der besonderen Hindernisse Open-Source-Prinzipien in der Spieleentwicklung angewendet werden können. Weiterhin wird untersucht, wie Spielefirmen davon profitieren können, Teile ihrer Produkte unter Open-Source-Lizenzen zu vertreiben.

Mit der zunehmenden Verbreitung von Breitband-Internetzugängen verändert sich der Spielemarkt und neue Arten von Spielen entstehen. Auch die Vertriebsformen der Spiele ändern sich. Wie sich dies auf die üblichen Geschäftsmodelle auswirkt und inwieweit sich dadurch neue Möglichkeiten für Open Source Spiele ergeben, soll ebenfalls betrachtet werden.

1.3 Zielsetzung

In der Arbeit sollen verschiedene Szenarien aufgezeigt werden, in denen es für kommerzielle Entwickler von Spielesoftware nützlich sein kann, auf offenen Quellcode zu setzen. Hierbei wird auch ausgearbeitet, ob dies in jedem Fall sinnvoll ist und wo geschlossene Lizenzen einen Vorteil bieten. Zu diesem Zweck werden zunächst die Grundlagen der beiden Teilbereiche, Freie Software einerseits und Spielesoftware andererseits, dargestellt. Anschließend wird anhand von Beispielen und durch Interviews mit Entwicklern herausgearbeitet, welche besonderen Probleme und Möglichkeiten sich durch die Offenlegung des Quellcodes ergeben.

Vielfach wurde die Vermutung geäußert, dass sich Computerspiele prinzipiell nicht unter Open-Source-Lizenzen entwickeln lassen. Ein Ziel dieser Arbeit ist die Untersuchung, ob dies tatsächlich so ist, oder ob nicht vielmehr eine Unterscheidung zwischen dem Quellcode der Spiele und ihrem Content getroffen werden muss. Dies sollte sich an dem erfolgreichen Einsatz von offenem Quellcode im Spielebereich zeigen, wenn der dazugehörige Content unter einer proprietären Lizenz vertrieben wird.

Auch die Fragestellung, ob Spiele grundsätzlich einen zu kurzen Lebenszyklus haben, um sich nach einem Open-Source-Modell entwickeln zu lassen, wird in dieser Arbeit betrachtet. Hierbei wird untersucht, ob diese These prinzipiell zutrifft oder ob

unterschiedliche Arten von Spielen auch unterschiedlich lange Zyklen zeigen und sich somit Teilbereiche der Spieleentwicklung doch für Open-Source-Modelle eignen.

1.4 Methodik

Die folgenden Ausführungen zur Methodik stützen sich primär auf Lamnek (1995, besonders Seiten 35 – 102).

Für die eigene Forschung im Rahmen dieser Arbeit wurde eine qualitative Herangehensweise gewählt. Hierfür werden sechs Beispiele aus unterschiedlichen Bereichen der Spielesoftware untersucht. Eine quantitative Herangehensweise war durch die Vielschichtigkeit und Verschiedenartigkeit der einzelnen Bereiche nicht möglich. Solch eine Untersuchung könnte sich nur auf einen Teilbereich konzentrieren, um vergleichbare Daten zu erhalten. Der qualitative Ansatz ermöglichte eine Betrachtung des gesamten Spektrums der Spielesoftware.

Ein quantitativer Ansatz würde eine Standardisierung der Datenerhebung erfordern, wodurch nicht mehr auf die Relevanzsysteme der untersuchten Projekte eingegangen werden könnte. Aufgrund der Unterschiedlichkeit der Bereiche würden aufgeworfene Fragen nur zu einem Teilbereich der Untersuchung passend sein. Hierdurch entstünden Antworten, die im Rahmen der Arbeit wertlos wären. Ferner können quantitative Untersuchungen nur etwas über messbare Merkmale aussagen, die dahinter stehende Motivation kann nur durch eine qualitative Untersuchung erfasst werden. Zusätzlich ermöglicht die offene Formulierung der Fragestellung bei einer qualitativen Untersuchung auch vollkommen unvorhergesehene Antworten.

Als Methode für die Erhebung der Daten wurde eine Form des Interviews gewählt. Das Interview wird als die inzwischen wichtigste qualitative Methode angesehen, weil die Beobachtung eines sozialen Feldes von außen immer schwieriger wird. Diese Problematik besteht besonders im untersuchten Bereich der Spieleentwicklung, weil hier ohne eine direkte Befragung der Entwickler nur die Veröffentlichungen der einzelnen Projekte betrachtet werden könnten. Bei diesen ist aber davon auszugehen, dass sie nicht notwendigerweise die tatsächlichen Hintergründe erfassen.

Auch wenn üblicherweise qualitative Befragungen mündlich durchgeführt werden, soll in dieser Arbeit überwiegend auf die schriftliche Form zurückgegriffen werden, um es überhaupt zu ermöglichen, weit entfernte Personen zu befragen. Durch die offene Formulierung der E-mails und die Möglichkeit für Rückfragen ergibt sich dennoch eine halb-standardisierte Befragung, von der zu erwarten ist, dass sie mit einer mündlichen vergleichbare Resultate zeigt. Auch wenn der fehlende visuelle Aspekt der E-mailbefragung die qualitative Arbeit erschwert, lässt sich durch die spezielle Gruppe der Befragten, die eng mit diesem Medium für den Zweck der persönlichen

Kommunikation vertraut sind, dennoch von der Vergleichbarkeit mit einem persönlichen Gespräch ausgehen.

Die Befragung zielt auf die Arbeit am Computer und mit dem Internet ab. Von daher ist die Methode des E-mail-Interviews konsistent mit der Anforderung, die Befragung im natürlichen Umfeld des Befragten durchzuführen. Es wird eine weiche Befragungsmethode verwendet, wodurch ein Vertrauensverhältnis aufgebaut wird, welches zu offeneren Antworten führen soll. Dabei wird aber vermieden, bereits in den Fragen eine Position aufzustellen, um den Befragten nicht zu beeinflussen. Der Vertrauensaufbau wird dadurch erleichtert, dass sowohl im Bereich der Spiele- als auch der Open-Source-Entwickler lockere und offene Umgangsformen, vor allem bei der elektronischen Kommunikation, vorherrschen.

Für qualitative Befragungen werden keine Zufallsstichproben gezogen, vielmehr werden nach den bestehenden Erkenntnisinteressen einzelne Fälle für die Befragung ausgesucht. Dabei muss auf eine unverzerrte, typische Auswahl geachtet werden. Abweichende Fälle sollten integriert werden, die Auswahl im Verlaufe der Forschung eventuell erweitert werden. Die inhaltliche Auswahl sollte im Verlauf der Forschung erweitert werden, wenn sich dies aus den Befragungen ergibt.

Die Befragten waren jeweils wichtige an der Entwicklung beteiligte Personen, dies wurde über ihre Beiträge auf Mailinglisten und Webforen zu den entsprechenden Spielen verifiziert. Hiermit sollte vermieden werden, dass Aussagen von Personen, die wenig Einfluss auf das eigentliche Projekt haben, einen zu hohen Stellenwert erhalten.

Die Auswahl der untersuchten Titel wurde durch den sehr großen Spielemarkt erschwert. Hinzu kommt, dass die Bedingungen jedes Spiels sehr unterschiedlich sein können. Es konnte somit keine repräsentative Auswahl an Spielen gefunden werden. Vielmehr wurden solche Titel als Beispiele ausgewählt, die bereits eine gewisse Lebensdauer aufweisen, um komplette Fehlschläge auszuschließen. Wichtigstes Kriterium war aber, dass Open-Source-Konzepte in der ursprünglichen Entwicklung oder im weiteren Lebenszyklus des Spiels eingesetzt wurden. Weiterhin wurde darauf geachtet, dass durch die Titelauswahl nicht nur eine bestimmte Nische des Marktes erfasst, sondern dass ein weites Spektrum an unterschiedlichen Bereichen einbezogen wird.

Eine besondere Problematik entsteht, wenn es zu einem bestimmten Titel nicht möglich ist, ein Interview zu erhalten. Bei der jeweils kleinen Anzahl an möglichen zu befragenden Personen ist es nicht unwahrscheinlich, dass solch ein Fall auftritt. Die betroffenen Titel müssen, sofern nicht genügend andere Informationen für eine Analyse zur Verfügung stehen, aus der Untersuchung herausfallen.

1.5 Überblick

Im Anschluss an das erste Kapitel, das zur Einleitung und Heranführung an die Problematik dient, werden im zweiten Kapitel kurz die Geschichte und Grundlagen von Open-Source-Software betrachtet. Es wird dargestellt, worin sich Closed Source und Open-Source-Software grundlegend unterscheiden, nicht nur in technischer, sondern auch in wirtschaftlicher Hinsicht. Dabei wird auch dargestellt, wie sich grundsätzlich mit Open-Source-Software auch Geld verdienen lässt. Weiterhin wird betrachtet, worin Schwierigkeiten und besondere Herausforderungen, auch besonders unter politischen Gesichtspunkten, bei der Entwicklung und dem Einsatz von OS Software bestehen.

Im dritten Kapitel wird das Thema Computerspiele im Allgemeinen untersucht. Hierbei werden neben der Geschichte der Computerspiele auch die Verbindungen gezeigt, die zu den Wurzeln der Freien Software bestehen. Weiterhin wird dargestellt, wie die Spieleindustrie aufgebaut ist und aus welchen grundlegenden Teilen ein Computerspiel besteht. Zusätzlich zeigt dieser Abschnitt auch die wirtschaftliche Bedeutung dieser Industrie. Anschließend werden die Besonderheiten von Computerspielen verglichen mit sonstiger Software untersucht.

Der Schwerpunkt des vierten Kapitels ist die Betrachtung der sechs Beispiele. Hierfür werden Spiele mit unterschiedlichem Hintergrund danach untersucht, auf welche Weise Open-Source-Prinzipien bei ihnen angewendet werden. Für diese Untersuchung wurden soweit, dies möglich war, an der Entwicklung des jeweiligen Spieles Beteiligte befragt.

Das fünfte Kapitel dient zur Analyse der Informationen, die in der qualitativen Untersuchung erarbeitet wurden. Hierzu werden die gefundenen Gemeinsamkeiten und Unterschiede der Beispiele herausgearbeitet. Weiterhin werden zusätzlich gewonnene Erkenntnisse in diesem Kapitel dargestellt.

Im sechsten Kapitel schließlich werden die Möglichkeiten aufgezeigt, die durch die Anwendung der Open-Source-Prinzipien auf Computerspiele entstehen. Hierbei wird auch untersucht, in welchen Teilen diese Anwendung sinnvoll ist und wann andere Möglichkeiten vorzuziehen sind. Verschiedene Lizenzmodelle werden auf ihre Anwendbarkeit auf Computerspiele geprüft. Besonders wird in diesem Abschnitt darauf eingegangen, welche Rolle die verschiedenen Open-Source-Finanzierungsmodelle in der Spieleentwicklung einnehmen könnten. Ferner wird ein Ausblick auf mögliche zukünftige Forschungsfelder gegeben.

Schließlich findet sich im abschließenden siebenten Kapitel das Fazit der Arbeit mit einer kurzen Übersicht der erarbeiteten Ergebnisse.

Zur Dokumentation der eigenen Recherche werden in Anhang A die geführten Interviews, soweit sie in schriftlicher Form vorliegen, abgedruckt. Sie sind nach den

jeweiligen Spielen, auf die sie sich beziehen, sortiert; spätere Ergänzungsfragen wurden in den Kontext eingeordnet und sind nicht weiter hervorgehoben. Ein Interview wurde mündlich durchgeführt, eine gekürzte Mitschrift davon wurde ebenfalls in den Anhang A eingeordnet.

Der Anhang B enthält das Verzeichnis der verwendeten Quellen in alphabetischer Sortierung. Sofern es sich um bloße Weblinks handelt, werden sie allerdings nicht in diesem Verzeichnis geführt, sondern ausschließlich in Form von Fußnoten an den entsprechenden Stellen referenziert.

1.6 Begriffsbestimmungen

Es bestehen größere Kontroversen um die richtige Verwendung der Begriffe „Open Source“ und „Freie Software“.[1] Diese Arbeit soll nicht der Fortsetzung dieser Diskussion dienen. Vielmehr soll die Auswirkung offener Quellen im Allgemeinen betrachtet werden, unabhängig von der verwendeten Lizenz. Die beiden Begriffe werden deshalb in dieser Arbeit austauschbar eingesetzt. Bei der Verwendung von „Freie Software“ im Sinne eines Eigennamens werden beide Worte groß geschrieben, um eine Abgrenzung zu „kostenlos“ herzustellen. Der Gegensatz zu Freier Software besteht darum auch nicht in kommerzieller Software, sondern in der proprietären Software. Mit diesem Begriff werden dann auch alle Programme bezeichnet, deren Quellen nicht für die Allgemeinheit einsehbar sind, ebenfalls unabhängig von eventuellen Kosten.

Weil es gerade im Bereich der Computerspiele nicht ausschließlich um Programmcode, sondern auch um Inhalte in Form von Grafiken, Sounds und so weiter geht, mussten die Begriffe „offen“ beziehungsweise „geschlossen“ auch auf diesen Bereich übertragen werden. Analog zum Programmcode soll auch beim Content nicht der Preis das Kriterium zur Unterscheidung sein, vielmehr sind es die Verwendungsmöglichkeiten, die die jeweilige Lizenz gestattet.

Mit „Content“ oder „Inhalten“ werden in dieser Arbeit alle diejenigen Bestandteile eines Programms bezeichnet, die nicht aus dem Programmcode bestehen. Dabei kann es sich um Soundeffekte, Musik, Grafiken, dreidimensionale Modelle, Texte und ähnliches handeln.

Von Bedeutung ist der Begriff des „Online-Spiels“. Hiermit werden in dieser Arbeit nur solche Spiele bezeichnet, deren primärer Spielmodus eine Verbindung mit dem Internet voraussetzt. Dies grenzt sie von solchen Spielen ab, die lediglich einen zusätzlichen Netzwerk-Modus bieten, aber hauptsächlich als Einzelspieler-Spiele ausgelegt sind.

[1] Ausführliche Betrachtungen dazu finden sich auf den Seiten der Free Software Foundation (URL: http://www.gnu.org/philosophy/ (12.12.2005)) beziehungsweise der Open-Source-Initiative (URL: http://www.opensource.org/ (12.12.2005)).

2 Grundlagen - Freie Software

Dieses Kapitel soll erklären, wie Freie beziehungsweise Open-Source-Software entstanden ist, welche Modelle dahinter stehen, welche Vorteile davon zu erwarten sind und wie sich diese Form der Softwareentwicklung finanzieren lässt.

2.1 Geschichte der Freien Software

Zunächst soll die Geschichte der Freien Software betrachtet werden. Diese hängt eng mit der Geschichte des Computers im Allgemeinen zusammen und verdeutlicht, warum Open Source die aktuelle Bedeutung erlangt hat.

2.1.1 1950-1970

Wie Levy (1984) über die Anfangszeit der Computergeschichte, also die 1950er bis 1960er Jahre, schreibt, war Software zu dieser Zeit inklusive Quellcode immer offen für alle. Dies lag sicher zum einen daran, dass es nur sehr wenige Computer gab, zum anderen auch daran, dass Computer meist nur im universitären Umfeld verwendet wurden. Es gab keine Standardsoftware, die vom Hersteller mitgeliefert wurde, jede Universität musste die benötigten Programme zum größten Teil selbst schreiben. Dies wurde an Universitäten, wie zum Beispiel dem Massachusets Institute of Technology (MIT) zum überwiegenden Teil von Studenten erledigt. Diese tauschten sich untereinander und mit den Computerbetreuern anderer Universitäten aus, um gemeinsam beste Lösungen zu entwickeln.

Die Entwicklung von Software wurde zu diesem Zeitpunkt allgemein als wissenschaftliches Arbeiten betrachtet schreibt Selig (2001). Somit wurden neue Erkenntnisse und Verbesserungen ganz selbstverständlich dem üblichen Peer Review Prozess ausgesetzt. Dies beinhaltet eine gegenseitige Überprüfung und Verbesserung von Lösungen, aber auch die Übernahme dieser Lösungen in eigene Arbeiten.

Levy (1984, Seiten 44 – 45) nennt als Beispiel eine Dezimalzahlenroutine[2], um die ein regelrechter Wettbewerb entstand, wer hier die kürzeste und beste Lösung hervorbringen könnte. Wurde als Ausgangsroutine noch eine Lösung verwendet, die 100

[2]Solch eine Routine wird benötigt, um den Binärcode des Computers in Dezimalzahlen zu übersetzen, die sich von einem Menschen besser verstehen lassen.

Zeilen Programmcode[3] brauchte, so war man durch gemeinschaftliche Verbesserungen bald auf 50 Zeilen gekommen. Als es schließlich jemand schaffte, die Routine sogar auf 46 Zeilen zu kürzen, veröffentlichte er sie, wie damals üblich, am Schwarzen Brett der Universität.

Auch über den Umgang der Gesellschaft mit Computern machten sich diese Studenten Gedanken. Da sie sich selber als „Hacker“ bezeichneten, formulierten sie ihre Gedanken in der sogenannten „Hacker Ethik“ aus. Hierin enthalten war eine Forderung nach freiem Zugang zu Informationen, und auch das gemeinschaftliche Streben nach besten Lösungen wurde festgehalten.

2.1.2 1970-1990

Mit der zunehmenden Verbreitung von Computern änderte sich die Situation. Es waren jetzt nicht mehr alle Nutzer von Computern gleichzeitig auch Programmierer. Diese reinen Nutzer waren bereit, für fertige Software zu bezahlen. Sie konnten oder wollten sich nicht weiter mit dem Programmieren beschäftigen, auch die Hacker Ethik war ihnen unbekannt. Gleichzeitig entwickelte sich der Beruf des Programmierers. Auch für diesen standen die Ziele der Hacker nicht im Vordergrund. An Stelle des gemeinschaftlichen Strebens nach immer besserem Code stand das Ziel, möglichst gut verkaufbare Programme zu produzieren. Eine freie Verbreitung der Quellcodes störte dabei. Levy berichtet in (1984, Seiten 228 – 230), dass einer der ersten, die anfingen Programme zu verkaufen, anstatt sie frei mit allen Interessierten zu teilen, Bill Gates mit seiner Basic Version war.[4]

Bis zum Jahr 1984 verschwand die ursprüngliche freie Software fast komplett. Software war keine Wissenschaft mehr, sondern zu einer Ware geworden. Bis Richard M. Stallman das GNU Projekt[5] gründete. Nach Husemann et al. (2002) liegt hier der Beginn der (modernen) Freien Software. Im GNU-Projekt und in der parallel entstandenen Free Software Foundation kam es zu den ersten systematischen Überlegungen, warum es sinnvoll sei, dass Software „Frei“[6] sein sollte. Eines von Stallmans erklärten Zielen war es, wie Ardal (2000) schreibt, ein komplett freies, UNIX-kompatibles System zu entwickeln.

[3] Hierbei entsprach eine Zeile Code jeweils einer Instruktion, da in Assembler programmiert wurde.

[4] Grassmuck (2000) schreibt in diesem Zusammenhang über Bill Gates berühmten „Open Letter to Fellow Hobbyists“, in dem er sich darüber beklagt, dass viele Computernutzer seine Software „stehlen“ würden womit sie die Entwicklung weiterer Software verhinderten.

[5] GNU ist eine rekursive Abkürzung, die für „GNU's not UNIX“ steht. Hiermit sollte ausgedrückt werden, dass ein komplett anderer Ansatz als in der mittlerweile zersplitterten und proprietären UNIX Welt gewählt wurde.

[6] Frei nicht im Sinne von kostenlos - „free beer“ - sondern im Sinne von Freiheit - „free speech“, mehr dazu im Abschnitt „Philosophie von Freier Software“

Im Gegensatz zu den ersten Jahren der Computergeschichte, als Software einfach frei war, weil es kein kommerzielles Interesse daran gab, musste jetzt eine rechtliche Absicherung dieser Freiheit geschaffen werden. Zu diesem Zweck entwickelte Stallman die GNU General Public License (GPL). Diese sorgte dafür, dass eine von ihr gedeckte Software nicht nur frei war sondern es auch blieb.

In der Öffentlichkeit spielte Freie Software zu diesem Zeitpunkt keine Rolle mehr, auch weil die Mehrheit der neu hinzugekommenen Nutzer nie etwas davon gehört hatte. Software war für diese Nutzer selbstverständlich eine Ware, genauso wie die Hardware, auf der sie lief. Einzelne kostenlose Programme änderten nichts daran, auch sie waren eine Ware, nur mit einem sehr niedrigen Preis.

2.1.3 seit 1990

Bis zum Jahre 1991 waren große Teile dieses freien UNIX kompatiblen Systems geschrieben. Der Systemkern fehlte aber noch - der im Jahre 1986 begonnene HURD-Kernel verzögerte sich immer wieder. In der Zwischenzeit fanden einige Veränderungen statt, die entscheidende Auswirkungen auf die Entwicklung der Freien Software hatten.

Das Internet, anfangs ein Zusammenschluss militärischer Netzwerke, wuchs immer weiter und schloss so auch die Netze der Universitäten ein. Grundlage der Infrastruktur dieses Netzwerkes war Freie Software, wie Ishii (1995) darstellt. Dies hatte zum einen den Vorteil, dass sich einheitliche Standards herausbildeten[7], zum anderen konnten Fehler auch deutlich schneller beseitigt werden, weil viel mehr fähige Personen den Quellcode nach solchen untersuchen konnten, als dies bei proprietärer Software möglich ist.

Diese Zusammenarbeit vieler mit dem gemeinsamen Ziel des verbesserten Programmes, veränderte wiederum die Art, wie Freie Software entwickelt wurde. Raymond (1999) bezeichnet die beiden unterschiedlichen Entwicklungsmodelle als „Kathedrale“ und „Basar“. Die Entwicklung nach dem neuen Basar Modell verhalf der Freien Software zu einer bisher unbekannten Dynamik.

Geprägt durch die Möglichkeiten, die das Internet bot und durch das neue Entwicklungsmodell, entwickelte der finnische Student Linus Torvalds seit 1991 einen UNIX kompatiblen Systemkern, den er Linux nannte. Durch die schnelle Entwicklung des Linux Kerns konnte dieser die Lücke füllen, die durch den fehlenden HURD Kern entstanden war. Mit GNU/Linux war ab 1994, wie Nüttgens und Tesei (2000a) schreiben, ein komplettes UNIX kompatibles System als freie Software vorhanden.

[7] Diese Standards wiederum trugen wesentlich zum Erfolg des Internets gegenüber proprietären Konkurrenten bei.

Open-Source-Software blieb auch mit der zunehmenden Verbreitung fast vollständig nichtkommerziell, Freizeitprogrammierer und Universitäten leisteten den größten Teil der Entwicklungsarbeit.[8] Dies änderte sich erst im Januar 1998, als mit Netscape erstmals eine bedeutende Softwarefirma ihr zentrales Produkt, den Webbrowser „Netscape Navigator“, unter eine Open-Source-Lizenz stellte. Andere Firmen, wie zum Beispiel Sun und IBM folgten und verhalfen Freier Software zu einem Durchbruch in den Bereich der geschäftlichen Anwendungen.

Zum jetzigen Zeitpunkt dominiert freie Software einen großen Teil des Servermarktes[9] und gewinnt in anderen Teilbereichen kontinuierlich an Marktanteilen. Auch die Desktop Arbeitsplätze in großen Firmen und Behörden[10] laufen zum Teil schon unter freier Software oder befinden sich im Prozess der Umstellung. Auch betreibt die EU-Kommission eine eigene Webseite, die der Förderung der Verbreitung von Open-Source-Software dienen soll.[11] Weiterhin ist zu beobachten, dass große Softwarefirmen einzelne Programme unter Open-Source-Lizenzen stellen und andere Programme von vorneherein unter freien Lizenzen entwickeln.

Im Bereich der privaten Computernutzung fristet freie Software zur Zeit aber noch ein Schattendasein. Ein wichtiger Grund, der viele Nutzer vom Umstieg auf freie Systeme abhält, ist häufig die fehlende Unterstützung für einzelne Software, die als benötigt betrachtet wird. Ein Bereich, in dem die fehlende Unterstützung besonders deutlich wird und in dem sehr wenig freie Alternativen vorhanden sind, ist der Bereich der Computerspiele. Capiluppi et al. (2003) erklären, dass der größte Teil der erfolgreichen Open-Source-Software noch immer von Experten für Experten entwickelt wird. Daraus resultiert, wie Crowston und Scozzi (2002) zeigen, dass sich freie Software häufig noch an den Bedürfnissen von Entwicklern und professionellen Administratoren orientiert, nicht an denen von reinen Anwendern.

2.2 Philosophie von Freier Software

Um die Grundidee Freier Software zu verstehen, ist es nötig festzustellen, wie Software, anders als klassische Güter, überhaupt als Allgemeingut funktionieren kann.

[8] Allerdings waren auch zu dieser Zeit schon Firmen an der Entwicklung beteiligt, wie David Lane unter http://openz.org/oshistory.php (08.12.2005) berichtet. Für diese war Software aber nur ein Nebenprodukt und nicht der eigentliche Geschäftsbereich.

[9] Zum Beispiel stellen Firmen ohne Computerbezug, wie der Otto-Versand, ihre Infrastruktur auf Linux um. (URL: http://www.heise.de/newsticker/meldung/49081 (17.10.2005))

[10] Hier ist als herausragendes Beispiel das Limux Projekt der Stadt München zu nennen. Im Zuge dieses Projektes sollen bis zum Jahr 2008 alle Computerarbeitsplätze der Stadtverwaltung auf Linux umgestellt werden.

[11] Zu finden ist diese Seite unter der URL: http://europa.eu.int/idabc/en/chapter/452 (17.10.2005)

Weiterhin werden in diesem Abschnitt die Vorteile der Nutzung dieser Software gegenüber den proprietären Gegenstücken dargestellt, und es wird untersucht, welche Ziele die Entwickler Freier Software damit verfolgen.

2.2.1 Software als Allgemeingut

Das prinzipielle Problem von Dingen, die frei für die Allgemeinheit zu nutzen sind, hat Hardin (1968) dargestellt. Wenn es für jemanden möglich ist, viel Nutzen aus einem Allgemeingut zu ziehen, er aber nur zu einem Bruchteil durch diese Nutzung mit belastet wird, dann wird er die Ressource so viel nutzen, wie nur irgend möglich. Weil sich fast alle Nutzer so verhalten werden, führt dies unweigerlich dazu, dass das Allgemeingut schließlich zerstört wird.

Osterloh et al. (2003) beschäftigen sich unter anderem mit der Frage, wie es dazu kommt, dass diese Tragödie der Allmende im Bereich der Freien Software nicht auftritt. Ein Grund dafür liegt darin, dass sich Software durch erhöhten Verbrauch nicht abnutzt. Es schadet also der Allgemeinheit nicht, wenn ein zusätzlicher Nutzer einer bestimmten Software neu hinzukommt. Weiterhin wird festgestellt, dass Hardin davon ausgeht, dass alle Nutzer prinzipiell extrinsisch[12] motiviert sind. Tatsächlich scheint es aber eine große Zahl intrinsisch motivierter Nutzer zu geben, die also ohne direkten Gewinn ihre Zeit in die Entwicklung Freier Software investieren.

Die Kombination aus diesen beiden Faktoren (also kein Verbrauch des Gutes und intrinsisch motivierte Produktion) führt dazu, dass Software als Allgemeingut bestehen kann. Zusätzliche Nutzer schaden nicht, je nach Art der Software bieten zusätzliche Nutzer sogar einen Vorteil, der durch positive Netzwerkeffekte bewirkt wird. Die Produktion weiterer Freier Software kann durch die intrinsische Motivation einiger Nutzer ausreichend gedeckt werden. Zusätzlich besteht auch eine extrinsische Motivation zur Entwicklung Freier Software zumindest für einen Teil der Entwickler.[13] Je höher die Anzahl produzierender Nutzer ist, desto mehr Nutzen bietet Freie Software insgesamt.

In einer Modellrechnung zeigt Khalak (2000), dass der Erfolg von Open-Source-Software zu einem großen Teil davon abhängt, dass eine kritische Masse an Nutzern erreicht wird. Dies liegt an den starken Netzwerkeffekten, die im Softwarebereich auftreten. Weiterhin macht er deutlich, dass sich Open-Source-Software in Märkten, in denen Werbung einen großen Einfluss auf den Markt hat, kaum verbreiten kann.

[12] Extrinsische Motivation beinhaltet eine Bezahlung in Form von Geld oder ähnlichem; intrinsische Motivation dagegen beruht auf einer Befriedigung von Bedürfnissen durch die Ausübung der Tätigkeit an sich.

[13] Mehr dazu im Abschnitt zu Finanzierungsmodellen.

2.2.2 Vorteile Freier Software

Die Vorteile, die Freie Software mit sich bringt, sind in den drei Bereichen Technik, Politik und Ökonomie zu finden.

Technische Vorteile sind vor allem für Firmen und technisch versierte Nutzer der Software vorhanden. Für diese besteht mit der Verfügbarkeit des Quellcodes die Möglichkeit, benötigte Änderungen schnell und ohne großen Aufwand durchzuführen.[14] Bei proprietärer Software dauern solche Anpassungen länger und sind in der Durchführung aufgrund der nötigen Kommunikation[15] mit dem Softwarehersteller komplizierter.

Außerdem ist die Fehlerrate bei Software mit frei verfügbarem Quellcode niedriger[16], weil eine größere Anzahl Personen den Quellcode nach diesen Fehlern durchsucht. Raymond (1999) stellt in diesem Zusammenhang fest, dass die Fehlerbehebung einfacher wird und formuliert dies als „Linus' Law": „Given enough eyeballs, all bugs are shallow". Zhao und Elbaum (2003) haben in einer Studie festgestellt, dass tatsächlich durch die Beteiligung der Anwender an der Fehlersuche viele Fehler gefunden wurden, die die Entwickler nicht entdeckt haben.

Durch das intensive Code Review, das durch viele unabhängige Entwickler stattfinden kann, erhöht sich im Allgemeinen sowohl die Stabilität als auch die Sicherheit von Open Source Software. Allerdings trifft dies nur auf Software zu, die auch einen hinreichend großen Anwenderkreis hat. Dies wird besonders dadurch bestärkt, wenn eine möglichst große Überschneidung zwischen Entwicklern und Anwendern vorhanden ist, denn - so bemerkt von Hippel (2001) - es ist deutlich einfacher für einen Nutzer einen selbst gefundenen Fehler zu reproduzieren, als für einen Entwickler, der nur aus zweiter Hand von diesem Fehler weiß. Wie MacCormack (2001) schreibt, können konventionelle Softwarefirmen diesen Effekt durch frühe öffentliche Beta-Tests nachahmen, doch können sie auch damit nicht an die Testintensität von großen Open Source Projekten herankommen.

Die politischen Vorteile der Nutzung freier Software ergeben sich aus der ursprünglichen „Hacker Ethik", die es verlangt, dass Informationen allen Interessierten zugänglich gemacht werden sollen. Die Weitergabe eines Programms als Freie Software wird damit zu einer moralischen Pflicht. Wie Zappe (2004) feststellt lässt sich unter bestimmten Umständen auch die Wohlfahrt eines Staates durch die freie Weitergabe maximieren.

Es ist zu beachten, dass sich das „Frei" bei Freier Software nicht auf einen bestimmten Preis bezieht, der für die Nutzung der Software zu bezahlen ist, sondern allein

[14] Aus diesem Grund stellt zum Beispiel die Deutsche Flugsicherung ihre Infrastruktur auf Linux um, wie die Firmenzeitschrift „transmission" in der Ausgabe Juli 2005 berichtet.

[15] Es muss ein Vertrag abgeschlossen werden, inklusive Pflichtenheft und Endabnahme.

[16] Dies stellten unter anderem Damien Challet und Yann Le Du in einer Untersuchung fest. URL: http://arxiv.org/abs/cond-mat/0306511 (29.06.2003)

auf die Möglichkeit, die Software nicht nur zu nutzen, sondern auch zu verändern und weiter zu geben Dieser Unterschied wird im Englischen in der Gegenüberstellung von „free speech“ und „free beer“ dargestellt. Kostenlos bedeutet also nicht automatisch frei und frei heißt nicht immer auch kostenlos.

Der Zusammenhang von Freier Software und Demokratie wird an dem folgenden Beispiel von Young und Rohm (2000, Seite 190) deutlich:

> Open Source passt außerdem von seinen Grundlagen her zu der Funktionsweise westlicher Demokratien. Stellen Sie sich vor, unser Rechtssystem würde auf einer geschlossenen Quellcodestruktur aufbauen, in der jeder Rechtsanwalt alle Rechtsausführungen neu erfinden müsste, weil es ihm nicht erlaubt wäre, sich auf bereits vorhandene Rechtsfälle zu stützen, ohne dafür Lizenzgebühren an andere Rechtsanwälte zu zahlen.

Aus der Perspektive von staatlichen Einrichtungen ist die Nutzung Quelloffener Software mit mehreren Vorteilen verbunden. So wird eine Abhängigkeit gegenüber großen ausländischen Softwareunternehmen vermieden. Es kann bei verfügbarem Quellcode die verwendete Software auf Hintertüren untersucht werden, wodurch eine zusätzliche Absicherung gegen Spionage geschaffen wird. Aufträge für Programmierung, Wartung und sonstige Dienstleistungen rund um die verwendete Software können an lokale Unternehmen vergeben werden, unabhängig davon, wo das Programm ursprünglich geschrieben wurde. Dadurch lassen sich lokal Arbeitsplätze schaffen. Dies ist auch daran zu erkennen, dass laut Dempsey et al. (1999) ein größerer Anteil der Open-Source-Entwickler aus Europa kommt als dies bei den Entwicklern proprietärer Software der Fall ist.

Neben den ökonomischen Vorteilen für die Gesamtwirtschaft ergeben sich auch für einzelne Unternehmen Vorteile aus der Nutzung Freier Software. Diese beruhen zum einen auf den bereits genannten technischen Vorteilen, zum anderen darauf, dass sich unter Umständen hohe Lizenzgebühren vermeiden lassen. Weil sich die Software anpassen lässt und somit auf nicht benötigte Funktionen verzichtet werden kann, ergeben sich auch Einsparmöglichkeiten durch eine längere Verwendbarkeit älterer Hardware. Wenn andersherum eine bestimmte zusätzliche Funktion benötigt wird, so ist es am effizientesten, wie Bessen (2001) darlegt, wenn für alle anderen Funktionen bereits vorhandene Software genutzt werden kann und nur das eine zusätzliche Feature neu geschrieben werden muss. Dies ist am günstigsten mit Open Source Software möglich.

Eine große Gefahr für Unternehmen, die spezielle proprietäre Software einsetzen, liegt darin, dass die Softwarefirma aufhört zu existieren oder aus anderen Gründen weder Support noch Weiterentwicklung für das Programm anbieten kann. Dies kann für das nutzende Unternehmen existenzbedrohend sein. Durch eine Verfügbarkeit der

Quelltexte wird diese Gefahr beseitigt. Das nutzende Unternehmen kann im Bedarfsfall jederzeit eine alternative Softwarefirma mit der Pflege des Programms beauftragen.

2.2.3 Besondere Probleme

Ein Problem, das Freie Software besonders trifft sind Softwarepatente.[17] Auch Hersteller proprietärer Software sind durch Patente bedroht, sofern es sich nicht um große Unternehmen mit eigenen Patentanwälten und eigenen Patentportfolios handelt. Doch stellt diese Problematik aus drei Gründen für Freie Software eine besonders große Gefahr dar:

1. Patentverletzungen durch Quelloffene Software fallen leichter auf, weil der Quellcode für jeden frei einsehbar ist. Bei Closed Source Software kann eine Patentverletzung nur aufgrund der Funktionalität vermutet werden.

2. Lizenzgebühren für die Nutzung von Patenten werden oft anhand der verkauften Stückzahl der betroffenen Software festgelegt. Diese lässt sich bei Open Source Software prinzipiell nicht feststellen. Außerdem können diese Lizenzkosten prinzipbedingt nicht auf den Softwarepreis umgelegt werden, denn es ist niemand verpflichtet, die Software beim Hersteller zu kaufen.

3. Auch die Anmeldung eines Patentes ist teuer und, wie Weckenmann (2003) bemerkt, keine Garantie dafür, von aufwendigen Gerichtsverfahren verschont zu werden. Dies trifft besonders kleine Firmen, die sich die zusätzlichen Kosten einer eigenen Rechtsabteilung nicht leisten können.

Im Ergebnis führt dies dazu, dass Hersteller von Freier Software, soweit sie in Ländern operieren, die von Softwarepatenten betroffen sind, versuchen müssen, bestehende Patente zu umgehen. Dadurch wird häufig die Funktionalität der Programme beeinträchtigt.[18] Eine weitere Strategie einzelner Open Source Hersteller besteht darin, selber Patente anzusammeln und diese für die Nutzung durch Quelloffene Software freizugeben. Im Falle eines Patentverletzungsverfahrens lassen sich solche Patente nutzen, um mit der gegnerischen Firma einen Vergleich, zum Beispiel in Form eines Austauschabkommens, auszuhandeln.

Walsdorfer (2003) zeigt die grundlegende Problematik bei der Patentierung von Software auf:

[17] Die Firma Open Source Risk Management stellte in einer Studie fest, dass alleine der Linux Kernel durch 283 Patente bedroht sei. Nachzulesen unter der URL: http://www.osriskmanagement.com/linuxpatentpaper.pdf (02.08.2004).

[18] Als Beispiel seien hier Abspielprogramme für Musik- und Videodaten genannt, bei denen die verwendeten Algorithmen besonders häufig einem Patentschutz unterliegen.

> Darüber hinaus spricht gegen eine Softwarepatentierung aber auch, dass Patente grundlegende Ideen so lange und absolut schützen, dass sie eine gesunde Weiterentwicklung von Softwareprodukten in diesem schnellebigen Markt extrem behindern könnten.

Ein zweites Problem können Firmengeheimnisse darstellen. Dies betrifft besonders Software, die ursprünglich mit geheimem Quellcode entwickelt wurde und nun freigegeben werden soll. Wenn bei der Herstellung dieser Software Firmengeheimnisse von anderen Firmen verwendet wurden, müssen die betroffenen Codeteile vor einer Freigabe entfernt werden. Ein ähnliches Problem besteht, wenn in einer Firma sowohl Open Source als auch Closed Source Software entwickelt wird.

Als drittes ist schließlich das Problem zu nennen, dass Code aus Quelloffener Software unter Verstoß gegen die Lizenzbedingungen von anderen Firmen in Software mit geheimem Quellcode verwendet wird. Solche unlizensierte Verwendung von Code ist bei Freier Software offensichtlich besonders leicht durchführbar. In bisher bekannt gewordenen Fällen ließ sich solche Verwendung recht sicher nachweisen, obwohl die verletzende Software nicht im Quellcode vorlag. Nach dem Bekanntwerden solcher Urheberrechtsverletzungen gab es bisher meist schnell eine Einigung, zumal die rechtliche Situation in diesen Fällen eindeutig ist.

2.2.4 Das Entwicklungsmodell

Auch wenn traditionell Freie Software genauso entwickelt wurde wie solche mit geheimem Quellcode, so gibt es bei den meisten modernen Projekten doch deutliche Unterschiede zur traditionellen Art der Softwareentwicklung. Dieses andere Entwicklungsmodell ist mitentscheidend für den Erfolg von Open Source Software. Raymond hat dieses Entwicklungsmodell als „Basarmodell“ dem herkömmlichen „Kathedralenmodell“ gegenübergestellt. Die Kernpunkte dieser Gegenüberstellung sind in Tabelle 2.1 dargelegt.

Das Basarmodell scheint auf den ersten Blick einer Gesetzmäßigkeit zu widersprechen, die von Brooks (1975) aufgestellt wurde. Demnach lässt sich bei der Softwareentwicklung nicht in dem Sinne mit „Mann-Monaten“ rechnen, dass man durch eine Erhöhung der Anzahl Programmierer die Entwicklungsdauer verkürzt. Im Gegenteil wird durch ein Hinzufügen von weiteren Entwicklern ein Projekt später abgeschlossen werden. Die Tatsache, dass Linux und andere Open-Source-Projekte eine schnelle und erfolgreiche Entwicklung zeigen, obwohl sehr viele Programmierer daran beteiligt sind, belegt, dass dieses Gesetz hier nicht zutreffen kann. Raymond begründet dies mit den verbesserten Kommunikationsmöglichkeiten durch die Entstehung des Internets. Dies sorgt dafür, wie Ettrich (2004) zeigt, dass alle Entwickler mit demselben Code arbeiten und somit Ressourcen fressende Parallelentwicklungen verhindert werden.

	Kathedrale	Basar
Betatests	intern	offen
neue Versionen	selten	so oft wie möglich
Änderungen pro Release	umfangreich	minimal
Entwickler sind Anwender	nein	ja
Nutzer	passive Konsumenten	in Entwicklung eingebunden
Entwicklungsort	meist lokal	weltweit verteilt
Entwicklungsarbeit	möglichst zentral	möglichst weit verteilt

Tabelle 2.1: Gegenüberstellung von Kathedral- und Basarmodell, Quelle: Eigene Zusammenstellung nach Raymond (1999)

Nach Nüttgens und Tesei (2000c) ist die Open-Source-Organisation eine Mischung aus der klassischen hierarchischen Organisation und einer Parallel-Organisation. Zusätzlich kommen noch neue Elemente hinzu, die sich in keiner dieser Formen finden lassen. Wie Barr (2005) erklärt, ist die Open-Source-Entwicklung ab einer gewissen Projektgröße genau dann am erfolgreichsten, wenn die Community am meisten eingebunden ist, also so wenig wie möglich zentralisiert entwickelt wird.

Wenn Berechnungsmodelle, wie das von Boehm (2002) vorgestellte, für Softwarekosten in der traditionellen Entwicklung verwendet werden, so erhält man für komplexe Open-Source-Projekte, wie zum Beispiel GNU/Linux Distributionen, extrem hohe Werte. So schätzen Robles und Gonzalez-Barahona (2004) den Entwicklungsaufwand der Debian Distribution in der Version 3.0 auf 3,6 Milliarden US-Dollar und 26835 Mann-Jahre. Diese immensen Kosten können anscheinend durch die Verwendung des Basarmodells bei der Entwicklung umgangen werden, vorausgesetzt es ist ein effektives Kommunikationsmedium wie das Internet vorhanden.

Einen weiteren Vorteil, den dieses Entwicklungsmodell durch das Internet erhält, sehen Kogut und Turcanu (1999) in der Möglichkeit einer Entwicklung rund um die Uhr, durch die weltweite Verteilung der beteiligten Programmierer. Hierdurch lässt sich die Entwicklungsgeschwindigkeit erhöhen, vorausgesetzt, dass eine effektive Kommunikation zwischen den Beteiligten stattfindet.

2.3 Finanzierungsmodelle

Grundsätzlich gibt es mehrere Möglichkeiten, wie ein Open-Source-Projekt finanziert wird. Die konventionelle Art der Software-Finanzierung, nämlich über den Verkauf der fertigen Software in Form eines Produktes, lässt sich offensichtlich nicht problemlos auf Open-Source-Software übertragen. Schließlich hat ja jeder Käufer der Software

wiederum das Recht, Kopien anzufertigen und selbst zu verbreiten. Dementsprechend wird sich der Preis, der mit dem Verkauf der Software erzielt werden kann, recht schnell in Höhe der Selbstkosten einpendeln.

Über Finanzierungsmodelle von Open-Source-Software wurde bereits viel diskutiert. Seit den ersten Ideen zu dieser Thematik ist einige Zeit vergangen und so lassen sich erste Ergebnisse ablesen, welche Finanzierungsmodelle wirklich erfolgversprechend oder auch schon bewährt sind.

Leiteritz (2002) untersucht verschiedene Finanzierungsmodelle von Open-Source-Firmen. Demnach gibt es drei grundsätzliche Methoden, nach denen Geschäftsmodelle für Freie Software funktionieren, nämlich Produkt-Modelle, Dienstleistungs-Modelle und Mediator-Modelle. Dabei stellt er fest, dass das Mediator Modell wirtschaftlich keinen Erfolg zeigen konnte. Neben Dienstleistungs- und Produkt-Modell soll hier deshalb nur die nicht kommerzielle Produktion von Freier Software betrachtet werden.

2.3.1 Dienstleistungen

Ein verbreiteter Ansatz und wohl auch das älteste Modell der Finanzierung von Open-Source-Projekten ist der Verkauf von Support zur Software. Dieser Support kann zum Beispiel darin bestehen, dass ein fertiges Paket aus verschiedenen Open-Source-Projekten vorinstalliert angeboten wird, dass telefonische Hilfestellungen bei der Installation gegeben werden, oder dass bei eventuellen Problemen schnell geholfen wird. Auch gedruckte Handbücher oder Verbreitung der Software auf CD-ROM gehören in diesen Bereich.

Das bekannteste Beispiel in diesem Bereich sind die verschiedenen GNU/Linux Distributionen, wie zum Beispiel Red Hat und Suse. Bei diesen wird ein Produkt verkauft, welches aus CD-ROMs und Handbüchern besteht und zu dem ein Installationssupport dazugehört. Die Dienstleistung besteht hierbei also in der Zusammenstellung der Pakete, dem Erstellen der Handbücher und dem angebotenen Installationssupport. Zusätzlich lässt sich noch ein erweiterter Support erwerben. Die Software, die in diesem Paket mit verkauft wird, bleibt aber weiterhin Freie Software, es ist möglich die Software dieses Paketes auch ohne die zusätzlichen Dienstleistungen kostenlos zu erhalten.

Die Bedeutung dieses Geschäftsmodells zeigt sich darin, dass einige der nach diesem Modell arbeitenden Unternehmen börsennotiert sind. Es ist allerdings festzustellen, dass diese Anbieter inzwischen zum größten Teil auf Unternehmenskunden ausgerichtet sind. Der Verkauf der Dienstleistungen an Privatkunden hat anscheinend nicht den erwarteten Erfolg gezeigt. Byfield (2005) macht in diesem Zusammenhang deutlich, dass nicht viele Nutzer bereit sind, für Software zu zahlen, die sie auch kostenlos herunterladen können, wenn der einzige Mehrwert eine hübsche Verpackung ist. Dies zeigt sich auch daran, dass die Anzahl Distributionen, die in Geschäften verkauft

werden, deutlich zurückgegangen ist, während insgesamt heute mehr verschiedene Distributionen existieren als zuvor. Weiterhin stellen Lakhani und von Hippel (2002) heraus, dass im Bereich der Privatnutzer sehr viel Supportleistung kostenlos durch gegenseitige Hilfestellungen der Nutzer erfolgt, indem zum Beispiel Technologien wie Newsgroups verwendet werden.[19]

2.3.2 Auftragsarbeiten

Dies ist eine Variante des Dienstleistungsmodells, bei dem die Leistung direkt durch Programmierarbeit erfolgt und nicht durch zusätzliche Leistungen, wie bei der klassischen Variante.

Gerade bei schon lange bestehenden Open Source Projekten mit grundlegenden Funktionen kann es vorkommen, dass zum Beispiel eine Firma eine bestimmte Erweiterung des Projektes wünscht. Sie hat nun die Möglichkeit, den Autor des Projektes oder auch einen beliebigen anderen Programmierer dafür zu bezahlen, dass er diese gewünschten Erweiterungen in die Software einbaut. Die Software bleibt trotzdem Open Source, alle so bezahlten Erweiterungen werden wieder der Allgemeinheit zugänglich gemacht. Für den Auftraggeber liegt aber der Vorteil darin, die benötigten Erweiterungen zu erhalten, ohne eine komplette Software neu entwickeln lassen zu müssen.

Dieses Modell ist leicht abgewandelt auch im Privatkundengeschäft möglich, wie zum Beispiel die Firma Transgaming zeigt. Diese bietet eine speziell angepasste Wine Version an, die es erleichtert, Spiele für die Windows Plattform unter Linux auszuführen. Welche Funktionalitäten als nächstes eingebaut werden sollen, wird durch eine Abstimmung der Abonnenten dieser Firma bestimmt.

2.3.3 Hardwarefirmen

Für reine Hardwarefirmen ist Software ein Nebenprodukt, mit dem direkt prinzipiell kein Geld verdient wird. Die Vorteile, die durch die Software entstehen, liegen darin, dass sich die dazugehörige Hardware besser verkaufen lässt. Es ist also oft nicht von Bedeutung für diese Firmen, ob der Quellcode dieser Software geheim oder frei verfügbar ist. Eine Mithilfe von externen Entwicklern an Treibern kann dagegen Geldmittel im Unternehmen selbst einsparen. Ein Finanzierungsproblem besteht hier also insoweit nicht, als die Firma die Softwareentwicklung sowieso bezahlen müsste, die Software aber ohnehin kostenlos herausgeben muss, um die entsprechende Hardware zu verkaufen.

[19] Auch wenn vielfach dieser kostenlose Support auch durch Firmen genutzt wird, so ist doch zu erwarten, dass in unternehmenskritischen Bereichen ein garantierter und damit kostenpflichtiger Support in Anspruch genommen wird.

Ein Problem, welches für bestimmte Hardwarefirmen von Bedeutung ist, sind allerdings Firmengeheimnisse in der Hardware selbst, die sich bei einer Offenlegung des Quellcodes von Treibersoftware möglicherweise nicht mehr schützen lassen. Dennoch ist es auch in diesen Fällen oft vorteilhaft, die nicht betroffenen Bereiche zu öffnen, um von den Entwicklungen externer Programmierer zu profitieren. Zum Teil kann dieses Problem auch dadurch umgangen werden, dass Schnittstellen zur Hardware veröffentlicht werden, ohne die Treibersoftware selbst offen zu legen.

2.3.4 Softwarefirmen

Klassische Softwarefirmen besitzen oft mehrere Produkte, die miteinander verknüpft sind. Häufig ist dies der Fall bei Programmen die eine Client-Server-Struktur haben und über das Internet kommunizieren. Hier ist es oft üblich, einen Teil der Software kostenlos herauszugeben, während der andere Teil Geld kostet. In so einem Fall kann der kostenlose Teil als Open Source Software herausgegeben werden, wodurch zum Beispiel die Verfügbarkeit auf unterschiedlichen Plattformen verbessert und somit die Verbreitung des Programms erhöht werden kann. Dadurch wird der kostenpflichtige Teil interessanter. Somit finanziert sich der kostenlose Teil durch die Einnahmen aus dem kostenpflichtigen Teil, ebenso wie es auch mit einem proprietären kostenlosen Teil der Fall wäre.

Ein zweites Modell, welches von Softwarefirmen verwendet wird, ist das Modell einer doppelten Lizenz. Zum einen wird die Software unter einer freien Lizenz bereitgestellt, die ähnlich der GPL eine Weitergabe aller Modifikationen unter den selben Bedingungen erzwingt, zum anderen wird auch kostenpflichtig eine proprietäre Lizenz angeboten. Ein Unternehmen, welches basierend auf dieser Software ein eigenes Produkt verkaufen möchte, wird üblicherweise die proprietäre Lizenz wählen. Endkunden und nicht kommerzielle Projekte dagegen werden die Freie Lizenz bevorzugen.

2.3.5 Universitäten

Ein wichtiger Teil der Softwareentwicklung findet im Rahmen universitärer Forschung statt. Myers (1996) zeigt auf, dass grundlegende Dinge, wie die direkte Manipulation von graphischen Objekten und Fenstersysteme in Universitäten ihren Ursprung haben, ebenso ganze Programmkategorien, wie Textverarbeitungen und Zeichenprogramme. Nicht zuletzt stammt auch die Technologie für das Internet aus diesem Bereich.

Für diese Projekte ist eine Freigabe des Quellcodes ein natürlicher Vorgang, weil nur so über wissenschaftlich erprobtes „peer reviewing“ eine Verbesserung ermöglicht werden kann. Die Kosten für die Entwicklung dieser Programme werden also wie die Kosten anderer wissenschaftlicher Arbeit behandelt, somit ist für diesen Bereich der

Softwareentwicklung kein eigenes Finanzierungsmodell nötig. Die universitäre Entwicklung ist somit in diesem Aspekt vergleichbar mit der Entwicklung durch Freizeitentwickler.

Nach Kelty (2001) ist die gesamte Entwicklung von Open-Source-Software Teil der Wissenschaft. Analog zur „Währung“ der Zitate in der üblichen wissenschaftlichen Arbeit sind hier die Codebeiträge und die Erwähnung der Autoren einer Software Bestandteil eines Währungssystems. Im universitären Rahmen ist dieses System fest etabliert und lässt sich so auch auf die universitäre Softwareentwicklung anwenden.

2.3.6 Freizeitprogrammierer

Als Freizeitprogrammierer sollen hier diejenigen Open-Source-Entwickler bezeichnet werden, die an sich entweder einen anderen Beruf haben, oder in Ausbildung, Rente usw. sind. Ihre Beiträge zum Pool der Freien Software entstehen im Regelfall komplett freiwillig und unvergütet und aus dem persönlichen Wunsch heraus, sich an einem bestimmten Projekt zu beteiligen. Die Finanzierung ist bei ihnen also irrelevant, die Lebenshaltungskosten werden durch andere Einnahmen gedeckt, die Software-Programmierung ist ein Hobby. Hars und Ou (2001) geben ihren Anteil an der Gesamtmenge der Open-Source-Entwickler mit 28% an.

Die Motivation dieser Gruppe soll im Folgenden näher betrachtet werden, denn sie stellt eine große Anzahl Entwickler und zeichnet für einen großen Teil der existierenden Freien Software verantwortlich. Zudem nimmt diese Gruppe im Bereich der Spieleentwicklung ebenfalls eine wichtige Stellung ein.

2.4 Motivation

In einer Studie stellten Ye und Kishida (2003) fest, dass eine wichtige Motivation von Entwicklern Freier Software darin liegt, ihr bestehendes Wissen zu erweitern und neue Möglichkeiten zu finden, bestimmte Aufgaben zu erledigen. In einem geschlossenen System kann zwar auch ein Programmierer mit seiner Software neue Wege gehen, aber in einem offenen System besteht die Möglichkeit, weitere Anregungen und neues Wissen von anderen Mitgliedern der Entwicklergemeinschaft zu erhalten. Hierdurch kann ein Open-Source-Entwickler sein Wissen schneller vermehren, als dies einem Entwickler von proprietärer Software möglich ist. Wie Robles et al. (2001) in einer Befragung von mehr als 5000 Entwicklern Freier Software feststellten, erhoffen sich viele von ihnen, durch diese Weiterbildung bessere Möglichkeiten oder eine höhere Bezahlung im Beruf zu erhalten. Diese extrinsische Motivation kann jedoch nur zum Teil gelten, denn nicht alle Open-Source-Entwickler werden durch dieses erworbene

Wissen mehr Geld verdienen können. Für einige ist offensichtlich die Möglichkeit, Spaß am Lernen zu haben, ausreichend.

Osterloh et al. (2004) zeigen auf, dass eine Mischung aus intrinsischer und extrinsischer Motivation bei den Beteiligten vorhanden sein muss, damit die Entwicklung eines Open-Source-Projektes erfolgreich verlaufen kann. Ein Projekt wird demnach meist von intrinsisch motivierten Entwicklern gestartet. Wenn es einen gewissen Reifegrad erlangt hat, ergibt sich dadurch die Möglichkeit für extrinsisch motivierte Entwickler, in das Projekt einzusteigen.

Anders ausgedrückt, wird ein Open-Source-Projekt im Normallfall anfangs keinen wirtschaftlichen Nutzen bringen. Mit zunehmender Reife und Verbreitung ergeben sich jedoch durchaus Möglichkeiten, monetären Gewinn aus den meisten Projekten zu ziehen. Gehring (2001) vermutet allerdings, dass für kreativ tätige Menschen eine extrinsische Motivation sogar kontraproduktiv sein kann, während von Krogh (2003) davon ausgeht, dass Innovation am besten funktioniert, wenn eine Mischung aus intrinsischer und extrinsischer Motivation vorliegt.

3 Grundlagen - Computerspiele

Nachdem wir uns mit den Grundlagen der Freien Software beschäftigt haben, wollen wir jetzt die Grundlagen des zweiten Teilbereichs – der Spielesoftware – untersuchen. Neben der Geschichte wird auch der gegenwärtige Zustand der Industrie betrachtet. Weiterhin wird gezeigt, aus welchen Bestandteilen moderne Computerspiele aufgebaut sind. Im abschließenden Abschnitt dieses Kapitels wird betrachtet, worin die Unterschiede zwischen Computerspielen und anderer Software liegen und welche besonderen Schwierigkeiten dadurch entstehen.

3.1 Geschichte der Computerspiele

Es ist wichtig, die Geschichte der Computerspieleentwicklung zu kennen, um zu verstehen, welche Gründe es für den heutigen Zustand der Spieleindustrie gibt. Auch lassen sich aus der Geschichte die gemeinsamen Wurzeln der Spieleindustrie und der Open Source Bewegung in der Hacker Kultur erkennen. Bei der Betrachtung der Geschichte werden für die Zeit bis 1983 nicht nur reine Computerspiele sondern auch Videospiele mit einbezogen. In der folgenden Zeit soll sich die Sicht auf Spiele für echte Computer konzentrieren, weil der Konsolenmarkt ab diesem Zeitpunkt eine deutlich andere Struktur aufweist. Aus den jeweiligen Entwicklungen lassen sich Schlussfolgerungen für den Bereich der Spiele insgesamt ableiten.

3.1.1 Sechziger Jahre

Wie Levy (1984) über die Anfangszeit der Computernutzung berichtet, war bis 1961 die Idee mit Computern zu spielen überhaupt nicht vorhanden. Dies lag zum einen an der geringen Anzahl vorhandener Computer, zum anderen daran, wie Computer zu damals bedient wurden. Weil zu dieser Zeit nur speziell ausgebildete Personen einen direkten Zugriff auf die Computer hatten, waren Programme für die damaligen Großrechner nicht interaktiv. Ein Entwickler gab sein Programm an das Computerpersonal ab und nahm Stunden später das Ergebnis in Empfang. Unter diesen Bedingungen waren Computerspiele undenkbar. Es gab zwar vereinzelt Vorschläge und erste Versuche damit, eine Form von Videospielen zu bauen, doch können diese nicht als wirkliche

Computerspiele bezeichnet werden. Zum einen war bei diesen Videospielen kein Computer im eigentlichen Sinne vorhanden, sondern entweder spezielle Hardware, die an einen Fernseher angeschlossen wurde oder ein umgebautes Oszilloskop. Zum anderen waren diese Videospiele keine Software, sondern wurden in Hardware implementiert. Gleiches gilt für die zu dieser Zeit bereits verbreiteten elektromechanischen Flipper.

Im Jahr 1961 änderte sich diese Situation mit dem TX-0 Computer und besonders später im selben Jahr mit der PDP-1. Beide Computer ermöglichten den direkten Zugriff, so dass interaktive Programme möglich wurden. Um die neuen Möglichkeiten und die Leistungsfähigkeit der neuen Computer zu demonstrieren, wurden sogenannte „Demo“ Programme geschrieben, die keinen direkten Nutzen hatten. Unter diesen Programmen gab es bereits ein „Tic-Tac-Toe“ für den TX-0, wenn auch ohne graphische Ausgabe. Graetz (1981) berichtet, dass die Science Fiction begeisterten Hacker[20] ein Demo-Programm für die PDP-1 entwickeln wollten, welches die Möglichkeiten ausnutzte, die der eingebaute Bildschirm lieferte - „Spacewar“ entstand im Jahr 1962.

Computerspiele waren in der Anfangszeit nicht kommerziell motiviert, sondern, wie Pias (2003) zeigt, eher eine politische Aktion. Computer waren zu dieser Zeit ausschließlich in Forschungseinrichtungen und großen Unternehmen vorhanden. Es gab also keinen Markt für computerbasierte Unterhaltung im Allgemeinen und Spiele im Speziellen. Herz (1997) beschreibt, dass sich Spacewar unter denjenigen, die Zugriff auf eine PDP-1 hatten, sehr schnell verbreitete. Dies war sehr einfach, weil der Quellcode selbstverständlich jedem zur Verfügung stand, der danach fragte.[21]

In den nächsten Jahren entstanden einige weitere Spiele, die Myers (1996) zu den Urvätern verschiedener Spielegenres zählt. So gab es mit „Lunar Lander“ die erste Flugsimulation, mit „Hammurabi“ eine Wirtschaftssimulation und mit „Adventure“ das erste Rollenspiel.

Obwohl diese Spiele eine Begeisterung unter den Computernutzern auslösten, waren es insgesamt nur wenige tausend Personen, die sie nutzen konnten. Der Grund dafür liegt in der zu dieser Zeit noch geringen Verbreitung von Computern aufgrund ihres hohen Preises.[22] Diese hohen Kosten verhinderten auch, dass eine solche „unwissenschaftliche“ Nutzung des Computers von Anwendern, die nicht zu den Hackern gehörten, überhaupt als nutzbringend angesehen werden konnte. Brunvard (1996) erwähnt, dass selbst die Verwendung einer Textverarbeitung als Verschwendung wertvoller Ressourcen angesehen wurde. Eine kommerzielle Verwertung von Computerspielen war daher ausgeschlossen.

[20] Hacker war zu dieser Zeit die Selbstbezeichnung derjenigen, die Computer nicht nutzten, weil sie mussten, sondern weil sie Spaß daran hatten. Die negative Bedeutung, die heute oft mit diesem Begriff in Verbindung gebracht wird, ist erst später entstanden.

[21] Aufgrund des offenen Quellcodes gab es auch bald Portierungen auf andere Computersysteme, die mit einem Bildschirm ausgestattet waren.

[22] Hinzu kam, dass Rechnerzeit oft streng eingeteilt wurde und Wartelisten vorhanden waren. Spiele mussten in diesem Fall natürlich zurückstehen.

3.1.2 Siebziger Jahre

Ein Versuch, Spacewar 1971 in Form eines Spielautomaten einer breiteren Öffentlichkeit zugänglich zu machen, scheiterte am mangelnden wirtschaftlichen Erfolg. Darum war, wie Holmquist et al. (1999) erklären, Pong das erste kommerziell verwertete und für die Allgemeinheit wirklich erhältliche Computerspiel. Es wurde ab dem Jahr 1972 in Form von Spielautomaten verbreitet. Diese Automaten gesellten sich zu den bereits existierenden mechanischen Flippern, die in vielen Cafes zu finden waren. Obwohl sich Spiele durch solche Automaten ein größeres Publikum erschlossen, waren sie immer noch sehr teuer, für jedes weitere Spiel musste ein eigener Automat gefertigt werden, und für den Start jeder Spielrunde musste der Nutzer Geld bezahlen.

Auch diese Automaten waren nur in eingeschränkter Form wirkliche Computerspiele. Jedoch bedienten sie sich bereits der Standardtechnologie, wie zum Beispiel der Fernsehtechnik. Die benötigte Hardware wurde in Massenfertigung hergestellt, nicht mehr in Handarbeit als Einzelstück.

1977 wurde die erste Spielkonsole für den Heimmarkt veröffentlicht, bei der nicht mehr alle möglichen Spiele direkt in der Konsole eingebaut waren. Spiele für diese Form der Konsole wurden auf Steckmodulen ausgeliefert. Somit stand mit einem einzigen Gerät eine unbegrenzte Anzahl Spielemöglichkeiten offen. Zunächst entwickelten nur die jeweiligen Hardwarehersteller auch Spiele für ihre Konsolen, oftmals waren diese Portierungen der entsprechenden Spielautomaten. Durch die Trennung von Konsole und Spielmodul wurde es aber erstmals möglich, dass unabhängige Spielefirmen eigene Titel herausbrachten.[23]

Gleichzeitig erlebten auch die Arcade-Automaten eine große Verbreitung und sehr viele Neuerscheinungen. Mit dem Erscheinen von „Space Invaders“ 1978 begannen viele neue Firmen mit dem Eintritt in die Spieleentwicklung. Dieses „Goldene Zeitalter“ der Arcadespiele, wie es zum Beispiel Guins (2004) bezeichnet, hielt bis in die frühen Neunziger Jahre an.

An den Universitäten gab es in diesem Jahrzehnt eine umfangreiche Entwicklung neuer Spielkonzepte. Dies blieb zunächst völlig losgelöst von der kommerziellen Spieleentwicklung, in der einfache Spiele und häufige Neuauflagen erfolgreicher Titel vorherrschten. Auch die Hardware unterschied sich vom Arkade Markt. Während dort speziell für das jeweilige Spiel entwickelte Rechner verwendet wurden, fand die Entwicklung hier auf den sich immer mehr verbreitenden Großrechnern statt. Unter den Studenten von Universitäten mit zueinander kompatiblen Computersystemen fand ein reger Austausch der dort entwickelten Spiele statt.

Gegen Ende der Siebziger Jahre wurde im universitären Rahmen auch damit begonnen, Onlinespiele zu entwickeln. Hierbei läuft das eigentliche Spiel auf einem entfern-

[23] Die erste Firma dieser Art war die 1978 von ehemaligen Atari-Mitarbeitern gegründete Activision.

ten Rechner, der Nutzer bedient eine Konsole, die nur den Zugriff auf diesen Server bereitstellt.

Für die sich langsam verbreitenden Heimcomputer verlief die Spieleentwicklung zu dieser Zeit noch ähnlich der universitären. Die Verteilung der Titel erfolgte häufig über Bücher oder Zeitschriften, in denen der Sourcecode zum Selbstabtippen abgedruckt war. Viele dieser Titel waren ursprünglich für die Großrechner der Universitäten entwickelt worden und wurden jetzt an die Hardware der Heimcomputer angepasst.

3.1.3 Achtziger Jahre

Im Jahr 1983 kam es zu einem totalen Zusammenbruch auf dem Videospielemarkt. Gallagher und Park (2002) führen ihn auf zwei Ereignisse zurück: Zum einen gab es im Jahr 1982 eine Schwemme minderwertiger Spiele für den Atari 2600. Dies führte zu hohen Rückläuferzahlen, vor allem bei Atari selbst, deren Umsatz um 50% einbrach und die in diesem Jahr einen Verlust von 539 Millionen US-Dollar erlitten.

Der zweite und vielleicht noch wichtigere Grund lag in dem massiven Preisverfall, der in dieser Zeit bei den Heimcomputern auftrat.[24] Infolgedessen verschwand der Preisvorteil der Spielekonsolen gegenüber den Heimcomputern, was dazu führte, dass viele Kunden einen richtigen Computer statt einer Konsole kauften.

Im Jahr 1984 war der Markt für Videospiele fast vollständig verschwunden und die großen Konsolenhersteller hatten sich aus dem Markt zurückgezogen. Eine weitere Folge dieses Crashes besteht darin, dass bei allen modernen Konsolen sämtliche verfügbaren Spiele nur als Lizenz des Konsolenherstellers vertrieben werden können. Hierdurch änderte sich gleichzeitig das Finanzierungsmodell vieler Spielekonsolen, die eigentlichen Einnahmen wurden nicht mehr durch den Verkauf der Hardware erzielt, sondern über Lizenzgebühren für die Spiele. Eine Spieleentwicklung für Konsolen ist seitdem nicht mehr für jeden möglich, kostenlose Spiele kann es hier prinzipiell nicht mehr geben.

Neben den Problemen auf dem Konsolenmarkt führten auch die deutlich geringeren Kosten der Entwicklung für Heimcomputersysteme zu einem „Goldenen Zeitalter“ in diesem Bereich. Anstatt Module mit teuren Speicherbausteinen herzustellen, konnten Titel für Computer einfach auf Kassetten und später auf Disketten ausgeliefert werden. Gleichzeitig war die Attraktivität der Computer für den Privatnutzer deutlich höher als die der Konsolen. Neben der Möglichkeit auch „ernsthafte“ Anwendungen auf einem Computer auszuführen, trug auch die Möglichkeit, Software leicht kopieren zu können, zu dieser Attraktivität bei.

[24] Zwischen dem 1.1.1983 und dem 31.5.1983 fielen die Preise der gängigen Heimcomputer zwischen 33% und 73%, so berichten Gallagher und Park

Spieleentwickler aus dem universitären Umfeld starteten Firmen, die zum Teil Titel von den Großrechnern auf Heimcomputer portierten und damit hohe Gewinne erzielen konnten.[25]

Drastische Leistungssteigerungen der Hardware in diesem Jahrzehnt ermöglichten die Entwicklung einer Vielzahl neuer Techniken in Spielen. Besonders im graphischen Bereich gab es neue Ansätze, wie zum Beispiel die ersten 3D Grafiken und der Einsatz von Videosequenzen. Auch wurden die textbasierten Eingabemethoden vermehrt durch „point-and-click“ Schnittstellen abgelöst.

Mit dem Spiel „Snipes“ begann 1983 erstmals auch die Vernetzung von Heimcomputern, auch diese eine Umsetzung aus der universitären Welt der Großrechner. Auch die ersten Mailboxen[26] entstanden, von denen die meisten auch Spiele anboten. Größere Mailboxensysteme, die über mehrere Einwahlmöglichkeiten verfügten, ermöglichten die ersten Onlinespiele mit mehreren Nutzern außerhalb des wissenschaftlichen Bereichs. Die ersten dieser komplett auf die Mehrspieler-Nutzung über Netzwerk ausgelegten Spiele waren Multi-User-Dungeons (MUD), die so zu textbasierten Vorläufern der heutigen Massiv-Mehrspieler-Online-Spielen wurden. Neben kostenlosen Angeboten gab es auch hier schon kommerzielle Dienstleister, die eine regelmäßige Nutzungsgebühr verlangten.

Auch der Markt der tragbaren Spiele entstand in den Achtziger Jahren. Allerdings blieb es zunächst bei Geräten, die nur fest eingebaute Spiele besaßen. Auch von den Grafik- und Soundfähigkeiten blieben sie weit hinter den Konsolen- und Heimcomputerspielen zurück. Zumeist waren die eingebauten Spiele Portierungen erfolgreicher Arkadetitel der Siebziger Jahre. Bae (2002) berichtet, dass diese „Game & Watch“ genannten Geräte weltweit über 40 Millionen Mal verkauft wurden.

3.1.4 Neunziger Jahre

Die erste Hälfte dieses Jahrzehnts war vom Entstehen neuer Spielegenres geprägt. Neben den Echtzeit-Strategie-Spielen im Gefolge von „Dune 2“ und den Survival-Horror-Spielen wie „Alone in the Dark“, waren es vor allem die First-Person-Shooter, allen voran „Doom“, die in diesen Jahren erstmals auftraten und in der Folge einen großen Teil des Spielemarktes ausmachten. Doch auch klassische Genres wie die graphischen Adventures erlebten zu dieser Zeit einen Höhepunkt.

Auch das Vertriebsmodell veränderte sich. Unter dem Druck erfolgreicher Sharewaretitel[27] wie „Doom“ begannen die meisten Hersteller damit, spielbare Demoversionen

[25] Ein Beispiel ist die Firma Infocom, die das Spiel „Zork“ erfolgreich auf mehrere Heimcomputersysteme portierte.

[26] Der Name „Mailbox“ leitet sich daraus her, dass die eigentliche Bestimmung dieser Systeme im Austausch von elektronischer Post bestand.

[27] Das Sharewaremodell erlaubt ein Ausprobieren eines Programms vor der Kaufentscheidung.

ihrer Titel kostenlos zu verteilen. Diese wurden oft über Zeitschriften, die sich mit Computerspielen befassten, verbreitet.

Die Zeit seit der Mitte der Neunziger Jahre ist von einer starken Konzentration auf dem Spielemarkt geprägt. Viele Firmen verschwanden oder wurden von größeren gekauft. Gleichzeitig setzten die noch verbleibenden Spieleentwickler vermehrt auf bewährte Spielkonzepte. Im Gegensatz zu den Achtzigern, in denen Experimente an den grundlegenden Spielprinzipien im Vordergrund standen, wurden jetzt hauptsächlich technische Aspekte wie Grafik und Sound verbessert.

Unterstützt wurde dieses durch das Aufkommen spezieller 3D Graikkarten, die einen großen Teil der Grafikberechnungen übernehmen konnten und somit eine deutliche Verbesserung der Grafikqualität ermöglichten. Wie Vogel (2001) feststellt, wurde das unter anderem auch durch die zunehmende Verbreitung der preiswerten CD-ROM und der dadurch erfolgten starken Erhöhung des zur Verfügung stehenden Speicherplatzes möglich. Damit werden eine deutlich verbesserte Grafik und hochwertige Soundeffekte erst wirtschaftlich, da diese sehr viel Platz auf dem Datenträger belegen.

Ein weiterer Trend ist in der zunehmenden Nutzung von Netzwerken, besonders des Internets, durch Spiele erkennbar. Vorreiter waren hier die Universitäten, bedingt durch die vorhandenen UNIX Strukturen, die prinzipiell auf Vernetzung ausgerichtet waren, zeigen Stabell und Schouten (2001). Im kommerziellen Bereich waren es neben den neuen Genres der First-Person-Shooter und Echtzeit-Strategie-Spiele, bei denen sich die Netzwerkfähigkeit als eine Bedingung für den Erfolg herausstellte, vor allem die klassischen Multi-User-Dungeons, die von dieser Entwicklung profitierten. Eine weite Verbreitung erfuhren diese aber erst mit dem Aufkommen voll graphischer Versionen, in denen sie als Massive-Multiplayer-Online-Role-Playing-Games (MMORPG) entstanden. Der erste erfolgreiche Titel dieses neuen Genres war das 1997 erschienene Ultima Online.

Der Bereich der mobilen Spiele wurde in diesem Jahrzehnt durch den Gameboy von Nintendo beherrscht. Dieser ermöglichte den Austausch von Spielen durch Steckmodule, ähnlich wie die frühen Heimkonsolen. Auch von den Grafik- und Sound-Fähigkeiten entsprach er den frühen nicht portablen Konsolen.

Erstmals begannen auch Spieler, selbst größere Veränderungen an angebotenen Spielen vorzunehmen und diese Änderungen zunächst über Mailboxen, später über das Internet zu verbreiten. Diese Modifikationen (Mods) wurden von den Herstellern der Spiele zumeist begrüßt, weil sich durch sie die Lebenszeit ihrer Spiele verlängerte. Zur Unterstützung der Modder veröffentlichten einige Entwickler große Teile der Datenstrukturen ihrer Titel oder legten den Spielen Werkzeuge zur Veränderung bei. Durch Preisverleihungen für die besten Mods durch einige Spielefirmen wurde diese Entwicklung weiter gefördert.

3.1.5 Gegenwart

In den letzten Jahren ist eine Angleichung der Spieleindustrie an die Filmindustrie erkennbar. Moderne Computerspiele verfügen oft über photorealistische Grafiken und Videosequenzen. Professionelle Schauspieler und Sprecher sind an der Produktion von Spielen beteiligt. Etat, Personalaufwand und Erstellungszeit gleichen denen von Hollywoodproduktionen. Gleichzeitig wird beklagt, dass neue Spiele lediglich alte Konzepte mit verbesserten Grafiken enthalten, während die Entwicklung neuer Prinzipien vernachlässigt wird.

Mit der Verbreitung von Breitbandanschlüssen, setzt sich der Trend zur Vernetzung fort. Im Zusammenhang damit können MMORPG immer größere Nutzerzahlen vorweisen. Gold, Gegenstände und Accounts dieser MMORPG werden vermehrt außerhalb der Spiele, zum Beispiel über Internetauktionen, verkauft. Dies nutzen vereinzelt Anbieter von MMORPG auch als neues Geschäftsmodell. Die zunehmende Wichtigkeit der Mehrspielerfähigkeit eines Spieles schlägt sich auch in der Entstehung von teamorientierten Spielen nieder. Hierbei müssen mehrere Spieler zusammen eine Aufgabe lösen. Interessanterweise entstand diese Fokussierung zu einem großen Teil durch von Nutzern erstellte Mods.[28]

Mobile Spiele nehmen einen immer größeren Raum ein, nicht zuletzt durch die Möglichkeit, auch mobile Geräte mit Farbbildschirmen herzustellen. Mobiltelefone werden programmierbar und können somit als tragbare Spielkonsolen dienen. Gleichzeitig vergößerte die Vereinheitlichung der Systeme durch die Unterstützung von Java in den meisten Mobiltelefonen den Markt für jedes einzelne Spiel. Tragbare Konsolen erhalten weitere Funktionen, wie zum Beispiel die Möglichkeit, Filme in hoher Qualität auf ihnen abzuspielen.

Das Ziel, einen größeren Markt für Computerspiele zu erschließen, hat zur verstärkten Entwicklung sogenannter „Casual Games“ geführt. Diese Spiele richten sich nicht an die üblichen Spieler, sondern an solche, die ohne großen Lern- oder Zeitaufwand gelegentlich ein Spiel spielen wollen. Aus diesem Grund sind solche Spiele im Allgemeinen deutlich weniger aufwendig entwickelt, um sie auch preiswerter anbieten zu können.

Auch die ursprünglichen Spieler werden durch solche wenig zeitaufwendigen Titel vermehrt angesprochen, wie Chris Satchell[29] feststellt. Er führt aus, dass das Durchschnittsalter von Spielern in den Vereinigten Staaten und in Europa inzwischen bei 30 Jahren liegt und diese Gruppe weniger Zeit zum Spielen aufbringen kann, als dies bei jüngeren Altersgruppen der Fall ist.[30]

[28] Counterstrike, ein verbreitetes Mod des First-Person-Shooters Half-Life, legte wichtige Grundlagen im Bereich des auf Zusammenarbeit im Team basierten Computerspieles.

[29] Er ist der General Manager der Xbox Game Developer Group bei Microsoft.

[30] Auf der Game Developers Conference Europe 2005, nachzulesen unter http://www.gamasutra.com/php-bin/news_index.php?story=6392 (17.10.2005).

3.2 Zustand der Spieleindustrie

Während in früheren Jahren noch kleine Entwicklerstudios oder sogar einzelne Entwickler den Spielemarkt dominierten und durchaus Titel veröffentlichen konnten, die von der Qualität im höchsten Bereich lagen, beherrscht heute eine andere Struktur die Spieleindustrie.

Die großen Summen, Macedonia (2001) nennt eine bis fünf Millionen US Dollar, die für die Entwicklung eines Computerspiels der sogenannten AAA Kategorie[31] aufgewendet werden, können nur von wenigen großen Publishern vorfinanziert werden. Diese lassen die eigentliche Programmierung von kleineren Entwicklerstudios durchführen und übernehmen neben der Finanzierung auch Werbung, Verpackung und Vertrieb. Auch die Lokalisierung und Übersetzung in verschiedene Sprachen wird im Allgemeinen durch den Publisher durchgeführt.

Diese Abhängigkeit von einem finanzkräftigen Publisher ist ein großes Problem für die Entwickler. Vielfach können kreative Titel nicht veröffentlicht werden, weil sich kein Publisher findet, der bereit ist, solch ein Risiko einzugehen. In Konflikten zwischen Entwicklern und Publishern besteht das Problem für die Entwickler, dass es nur wenige finanzstarke Publisher gibt und sie so Gefahr laufen, zukünftig prinzipiell von diesen ausgeschlossen zu werden.

Viele Entwicklerstudios lagern selbst wiederum einen mehr oder weniger großen Teil der Produktion in verschiedener Form aus.[32] Das kann auf unterschiedliche Weise erfolgen. So können Codebibliotheken oder gleich eine ganze Engine zugekauft werden oder für die Sprachausgabe Schauspieler eingestellt werden. Auch die Inhalte der Spiele kommen häufig aus anderen Bereichen. Seit Beginn der kommerziellen Spieleentwicklung werden zum Beispiel Lizenzproduktionen erfolgreicher Kinofilme hergestellt.

Diese Lizenzproduktionen sind andererseits ein großes Problem der Spieleindustrie. Rocca (2005) erklärt, dass Lizenzprodukte fast nie zu herausragenden Spielen führen und dass die größten Fehlschläge fast immer auf externen Lizenzen basierten. Gleichzeitig wird durch die Verwendung von lizensiertem Content das finanzielle Risiko für die Publisher verringert. Im Ergebnis führt dies zu einer großen Zahl mittelmäßiger und mittelmäßig erfolgreicher Spiele.

Auch Myers (n.d.) sieht ein Problem in dieser Anpassung an die Filmindustrie. Er argumentiert, dass externe Hintergrundgeschichten bei Computerspielen keinen Nutzen bringen oder sogar schädlich sind. Computerspiele sind demnach ein komplett

[31] AAA bedeutet hierbei, dass es sich um einen Spitzentitel handelt, eine ähnliche Einteilung wie bei Hollywood Filmen, bei denen es neben den teuren Produktionen noch „Low Budget“ Filme oder B-Movies gibt.

[32] Dies zeigt eine Studie von Amritt Ventures, zitiert nach http://www.gamasutra.com/php-bin/news_index.php?story=5974 (17.10.2005).

eigener Bereich; der Versuch, Konzepte aus anderen Bereichen zu übernehmen, zum Beispiel um eine Vermarktung auch als Film zu erlauben, führt nicht zu besseren Spielen.

Ebenso ist die teilweise vorherrschende Konzentration auf rein technische Aspekte eines Spiels problematisch, wie Rouse (1998) herausstellt. An sich langweilige Spiele werden durch immer aufwendigere Grafiken vermarktet, interessantere Spiele mit einfacher Grafik haben Schwierigkeiten, einen Publisher zu finden.

Nicht nur die Kosten und der Entwicklungsaufwand haben für Computerspiele die Größenordnung von Hollywood Produktionen erreicht, auch die Einnahmen der Spieleindustrie lassen sich mit denen der Filmindustrie vergleichen, beziehungsweise übertreffen sie diese bereits, wie Abbildung 3.1 zeigt. So erwirtschafteten Spielefirmen im ersten Halbjahr 2005 in den Vereinigten Staaten Einnahmen in Höhe von 4,1 Milliarden US Dollar.[33] Weltweit erwirtschafteten Computerspiele im Jahr 2004 einen Umsatz von 25 Milliarden US Dollar. Vorhersagen gehen von einem Wachstum auf 55 Milliarden US Dollar bis zum Jahr 2009 aus, womit die Spieleindustrie das größte Wachstum im Unterhaltungsmarkt aufweist.[34]

Macedonia erklärt, dass der weitaus größte Teil dieser Einnahmen inzwischen auf dem Konsolenmarkt erwirtschaftet wird:

> These days, PCs are just a sideshow compared to consoles such as the PlayStation 2 and the newly launched Gamecube and Xbox. Analysts consider a PC game that sells 100,000 copies to be a major hit, yet console blockbusters such as Square's Final Fantasy X for the PS2 sold more than 2.4 million copies in Japan alone this year.

Während in den USA Computerspiele nur rund ein Zehntel des gesamten Spielemarktes ausmachen, liegen in Deutschland Konsolenspiele und Computerspiele vom Umsatz her ungefähr gleich auf. Wie die Frankfurter Allgemeine Zeitung am 14.07.2004 (Nr. 161 / Seite 16) berichtete[35], belief sich der Umsatz im Jahr 2003 bei Computerspielen auf 569 Millionen Euro, bei Videospielen auf 564,3 Millionen Euro.

Neben diesem sehr großen Markt gibt es weiterhin kleine Entwicklerstudios, sogenannte „Independent Developers“ oder auch „Indies“. Aufgrund der sehr viel geringeren Ressourcen dieser Firmen, vor allem der fehlenden Finanzreserven für eine

[33] Quelle: The NPD Group (URL: http://www.npd.com/ (17.10.2005)), zitiert nach Gamasutra (URL: http://www.gamasutra.com/php-bin/news_index.php?story=6053 (17.10.2005)).

[34] Zahlen nach einer Analyse von PriceWaterHouseCoopers (URL: http://www.pwcglobal.com/ (01.11.2005)), zitiert nach Gamasutra (URL: http://www.gamasutra.com/php-bin/news_index.php?story=6752 (01.11.2005)).

[35] URL des Artikels: http://www.faz.net/s/RubC9401175958F4DE28E143E68888825F6/Doc~E4D21E3FC3DB249AF8E0F5E3F92F6EAE3~ATpl~Ecommon~Scontent.html (15.12.2005)

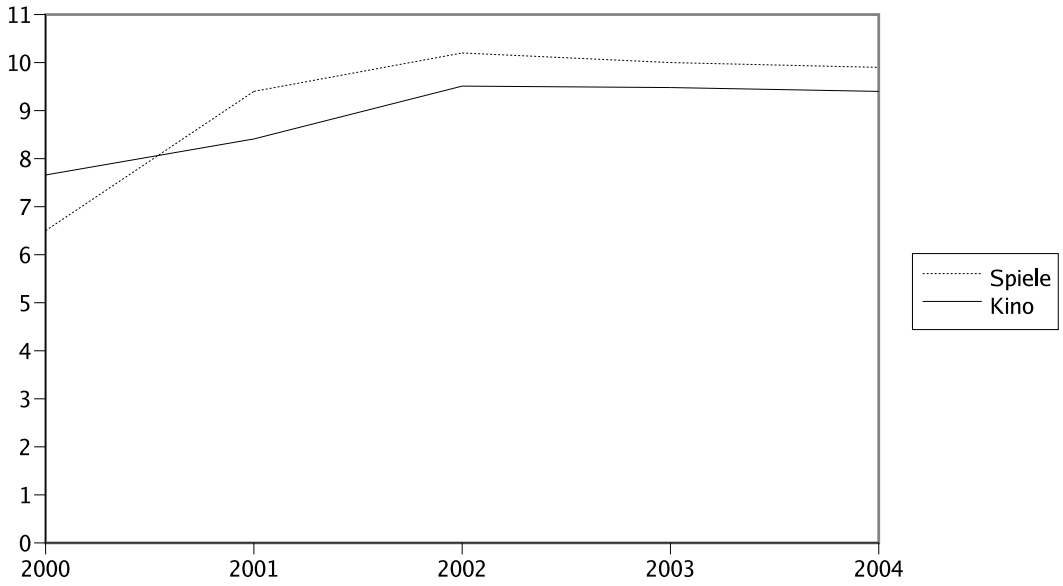

Abbildung 3.1: Vergleich der Einnahmen (in den USA) der Spieleindustrie mit denen der Kinokasse (in Milliarden US Dollar). Quelle: Eigene Grafik nach Brahe (2005)

mehrere Jahre dauernde Entwicklung eines neuen Titels, sind die dort entstehenden Spiele von einer anderen Art als die der großen Publisher. Während dort auf bewährte Konzepte gesetzt wird und viele Spiele unter Lizenz entwickelt werden, versuchen kleine Entwickler, mit ganz neuen Ideen auf dem Markt Fuß zu fassen.

Die von den Indies entwickelten Spiele sind meist von einem sehr viel geringeren Umfang als die Titel der großen Studios. Oft sind sie auch lediglich per Download von einer Webseite zu kaufen, weil ohne einen Publisher der Vertrieb an Ladengeschäfte nur in Ausnahmefällen möglich ist. Aus diesen beiden Tatsachen resultieren im Allgemeinen geringere Preise für diese Spiele. Damit bedienen diese Entwickler vermehrt den neuen Markt der Casual Gamer.

Tinney (2005) zeigt am Beispiel des Indies „Large Animal“ unter anderem auf, wie groß dieser Markt ist. Das Spiel „Rocket Bowl“ dieses Entwicklerstudios, das von ein bis fünf Leuten über 18 Monate entwickelt wurde, kostete 150.000 Dollar bis zur Fertigstellung. Das Spiel, das sich gezielt an Casual Gamer richtet und ausschließlich als Download vertrieben wird, erreichte mehr als 750.000 Anwender.

Vielfach bedienen Indies auch den Bereich des „Advergaming“, also der Entwicklung von Werbespielen für unterschiedliche Firmen. Hier ist der Umfang und Entwicklungsaufwand noch einmal deutlich geringer als es bei den Casual Games ohnehin schon der Fall ist. Der Etat wird bei diesen Spielen von der auftraggebenden Firma

bezahlt, die Spiele werden auf unterschiedliche Arten kostenlos verteilt, häufig als Download auf der Firmenwebseite oder auch als Beilage zum verkauften Produkt.

Der deutsche Spielemarkt zeigt mehrere Besonderheiten. Wie bereits erwähnt, nehmen gerade in Deutschland die Computerspiele im Vergleich zu den Videospielen eine deutlich wichtigere Position ein als in den meisten anderen Ländern. Eine zweite Besonderheit liegt in der Herkunft der in Deutschland verkauften Spiele. Wie der Bundesverband der deutschen Spieleentwickler G.A.M.E.[36] berichtet, werden 95% des in Deutschland mit Computer- und Videospielen erwirtschafteten Umsatzes durch internationale Titel abgedeckt. Deutsche Titel sind aber auch im Ausland wenig gefragt. Dies ist in den europäischen Nachbarländern anders, unter anderem aufgrund von Fördermaßnahmen durch die jeweiligen Regierungen. Durch diese Marktsituation stehen deutsche Spieleentwickler vor besonderen Problemen, die sie zwingen, besonders günstig zu produzieren.

3.3 Aufbau eines Computerspiels

Computerspiele werden im Allgemeinen unter Verwendung mehrerer Grundbausteine erstellt, die möglichst häufig wiederverwendbar sind und so die Entwicklung erleichtern. Die Grafik 3.2 auf Seite 33 stellt diesen Aufbau dar, der im Folgenden erläutert wird.

An unterster Stelle steht eine Hardware Abstraktionsebene (Hardware-Abstraction-Layer – HAL). Diese dient dazu, Einzelheiten der Hardwareansteuerung zu verbergen. Beispiele dafür sind DirectX unter Windows und SDL unter Windows, Linux, MacOS X und weiteren Systemen. Diese Abstraktionsschichten bleiben meist über längere Zeiträume stabil. Aus diesem Grund ist hier auch ein großes Interesse an einheitlichen Standards vorhanden. Das verwendete HAL ist ein entscheidender Faktor, wenn es um die Portierbarkeit eines Spiels geht.

Für diese unterste Ebene gilt dann auch, was Andreas Lange bereits 2000 in Telepolis[37] schrieb:

> Spiele, die Engines auf denen sie basieren und vor allen Dingen die Tools, die man zu ihrer Herstellung benötigt, werden mehr und mehr plattformunabhängig ausgelegt sein. So wird es in Zukunft keinen großen Aufwand mehr bedeuten, ein Spiel für mehrere Betriebssysteme gleichzeitig zu veröffentlichen. In diesem Zusammenhang spielt die Open-Source-Idee eine wichtige Rolle. Kann man doch nur so hoffen, Schnittstellen wie

[36] Webseite des G.A.M.E. Bundesverbandes unter der URL: http://www.game-bundesverband.de (18.01.2006).
[37] URL: http://www.heise.de/tp/r4/artikel/6/6678/1.html (17.10.2005).

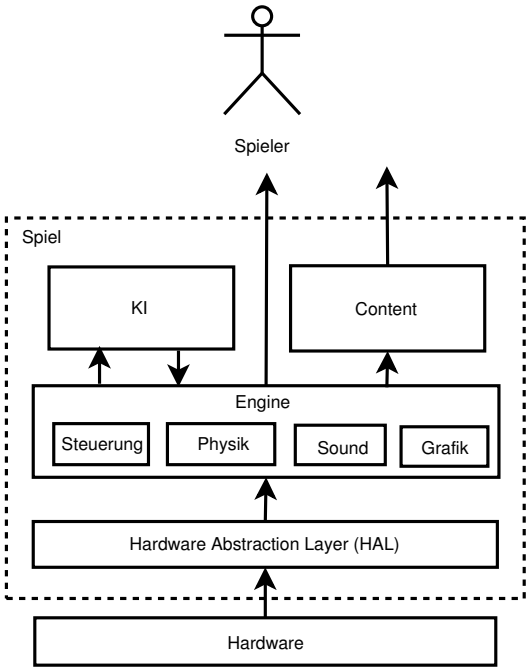

Abbildung 3.2: Der typische Aufbau eines Computerspiels, Quelle: Eigene Zusammenstellung

> OpenAL als Standard auf allen Betriebssystemen durchzusetzen. Nur was keiner einzelnen Firma gehört, und was alle weiterentwickeln können, wird in Zukunft eine Chance haben, als Standard anerkannt zu werden.

Eine Ebene höher findet sich die Engine des Spiels. Obwohl sie auf die unterste Schicht aufgesetzt sind, gibt es in einer Engine im Allgemeinen noch für das Betriebssystem spezifische Teile. Zu beachten ist, dass häufig nicht eine einzelne Engine zum Einsatz kommt, die alle grundlegenden Aufgaben erledigt, sondern oft eine Aufteilung stattfindet. So kann es eine spezielle Engine für die Physiksimulation geben, während die Soundausgabe von einer weiteren Engine übernommen wird. Diese verschiedenen in einem Spiel verwendeten Engines sind häufig von unterschiedlichen Herstellern, die sich jeweils auf einen bestimmten Bereich spezialisiert haben, lizenziert.

Die eingesetzte Engine bestimmt zu einem großen Teil das Erscheinungsbild des Spiels. Neue technische Entwicklungen, die im Spiel verwendet werden sollen, erfordern meistens umfangreiche Änderungen oder eine komplette Neuentwicklung dieser Bibliotheken. Aus diesem Grund ist der Zeitraum, über den eine bestimmte Engine eingesetzt wird, deutlich kürzer als bei der untersten Ebene. Dennoch findet eine Engine oft in mehr als einer Spielegeneration Verwendung.

Weil ein immer größerer Teil der Grafikfunktionen direkt in der Hardware der Grafikkarte implementiert wird, müssen zwangsweise die Engines an diese Entwicklungen angepasst werden. Sie greifen allerdings nicht direkt auf diese Kartenfunktionen zu, sondern über den Treiber, im Prinzip ein eigenes HAL der Grafikkarte, der im Normalfall direkt vom Hersteller des Grafikchips entwickelt wird. Vollmer (2003) stellt fest, dass aus dieser Verlagerung der Grafikdarstellung in die Hardware folgt, dass bestimmte Ansätze zur Berechnung von Grafik gegenüber Alternativen, die in Software implementiert werden müssen, bevorzugt werden. Dadurch sind sich viele Engines, die derselben Generation angehören, von ihren Eigenschaften her sehr ähnlich.

Ein wichtiger Teil des Spiels, der zumeist parallel zur eigentlichen Engine existiert, ist die eingesetzte Künstliche Intelligenz (KI). Hierbei existieren ganz unterschiedliche Ansätze, je nachdem für welchen Zweck die KI im Spiel verwendet werden soll. Stahl und Skibak (2002) beschreiben einige übliche Arten von in Spielen verwendeten Künstlichen Intelligenzen. Auch in diesem Bereich existieren generische Entwicklungspakete von Drittherstellern. Wenn mehr Flexibilität gewünscht wird, werden allerdings häufig vollständige Skriptsprachen eingesetzt, um das Verhalten von Computerspielern zu steuern. Je nach Ansatz lassen sich diese Skripte auch später noch verändern und ermöglichen so eine Anpassung, zum Beispiel für Mods oder verschiedene Schwierigkeitsgrade.

Auf der obersten Ebene und damit direkt für den Nutzer sichtbar, ist der Inhalt des Spiels, von Dammertz (2002) als Content bezeichnet. Darunter fallen die Grafik und alles was dazu gehört,[38] sowie Sound und Musik. Weiterhin sind der Aufbau der einzelnen Level und die Hintergrundgeschichte des Spieles ein wichtiger Bestandteil des Contents. Weil dieses der Bereich ist, der als ehestes als „das Spiel“ vom Nutzer verstanden wird, gibt es häufig mehrere Spiele, die sich fast nur im Content und kaum im Programmcode unterscheiden. Dies ist besonders oft bei Nachfolgern oder Erweiterungen zu erfolgreichen Titeln anzutreffen. Wichtig ist dieser Punkt, weil bei der Erstellung des Contents kaum Softwareentwickler benötigt werden, dafür hauptsächlich Grafiker, Musiker, Schauspieler und Autoren.

3.4 Unterschied zwischen Spielen und anderer Software

Neben Hargreaves (1999) und Husemann et al. (2002) hat sich zuletzt Geitgey (2004) damit beschäftigt, worin die Unterschiede zwischen Spielen und sonstiger Software bestehen und warum es so schwierig zu sein scheint, hochwertige und auf dem Markt erfolgreiche Open-Source-Spiele zu entwickeln. Abgesehen davon findet das Thema

[38] Bei 2D Spielen sind das hauptsächlich Bitmap Grafiken, bei 3D Spielen zum Beispiel Modelle und Texturen.

Spiele in Untersuchungen zu Open Source wenig Beachtung. Andersherum wird auch bei der Betrachtung von Spielen zumeist eine proprietäre Lizenz stillschweigend vorausgesetzt, ohne die Möglichkeit einer offenen Lizenz in Betracht zu ziehen.

Ein wichtiger Unterschied liegt in der Tatsache begründet, dass Spiele im Allgemeinen extrem kurzlebige Produkte sind. Das Open-Source-Entwicklungsmodell basiert aber gerade auf einer langfristigen und stetigen Verbesserung, nicht darauf, ein einmal abgeschlossenes Programm möglichst schnell durch einen neu entwickelten Nachfolger zu ersetzen. Während es bei sonstigen Programmen wünschenswert ist, dass die Benutzungsoberfläche möglichst lange gleich bleibt, um keine neue Einarbeitungszeit investieren zu müssen, ist bei Spielen eine ständige Neuerung gefragt. Sie lassen sich eher mit Filmen vergleichen, ein bereits „durchgespieltes“ Spiel ist uninteressant.[39]

Andererseits verschlingt die Spieleentwicklung immer mehr Ressourcen und ist auch hierbei mit der Filmindustrie vergleichbar. Singleton (2003) stellt heraus, dass diese investierten Ressourcen einem hohen Risiko ausgesetzt sind. Spieleentwicklung findet häufig an der Spitze der aktuellen Technologie statt, so dass es sehr schwer ist, Erfolg oder Misserfolg vorherzusagen.

Hargreaves erklärt, dass die Wichtigkeit des reinen Programmcodes bei Spielen sehr viel geringer ist als bei gewöhnlicher Software. Bei Spielen sind Grafik, Sound, Musik und Design sehr viel entscheidendere Faktoren. Der Programmcode ist nur dafür zuständig, diese Elemente dem Nutzer zur richtigen Zeit zu präsentieren. Torvalds (2001) stellt fest:

> Heute macht die Programmierung nur einen ziemlich kleinen Teil der Spiele aus. Was wirklich zählt, sind Musik und Handlung. Vergleicht man ein Computerspiel mit einem Film, ist die Programmierkomponente lediglich die Kameraarbeit.

Hinzu kommt, dass Programmcode wieder verwendet und dabei immer weiter verbessert werden kann, während Grafik und Musik für jedes Spiel komplett neu gestaltet werden müssen. Geitgey führt aus, dass sich Spiele zu einem großen Teil über diese Elemente definieren und voneinander absetzen und nicht über den verwendeten Code.[40]

Der Bundesverband der Entwickler von Computerspielen (G.A.M.E.) erklärt in Behrmann et al. (2005) zum Verhältnis von Software und Computerspielen folgendes:

> Die Software ist aber nur ein Teil des Computerspiels, quasi der Träger der Interaktivität. Die Spiele selbst sind multimediale Werke. In ihnen sind

[39] Eine Ausnahme hiervon sind Spiele, die aus verschiedenen Gründen besonders langlebig sind. Diese sollen später noch genauer betrachtet werden.

[40] Auch hierin ist eine deutliche Ähnlichkeit mit Filmen festzustellen.

> typischerweise zahlreiche Werkformen integriert und kombiniert. Zu nennen sind etwa Computeranimation, Bild-, Ton-, Laufbildkunst, graphische sowie technische Grundbausteine.
>
> Durch den Zusammenschluss der Werke entsteht eine neue Qualität, die ebenso wenig als Software beschrieben werden kann, wie ein Film als 1000 Photografien und Musik und Sprache bezeichnet wird. Man könnte ein Computerspiel als eine (interaktive) elektronische Publikation oder eben als interaktiven Film beschreiben. Der Begriff „Software“ ist in diesem Zusammenhang aber irreführend. Das gestalterische und narrative, das Gameplay all jenes, was das Computerspiel erst als solches wahrnehmbar macht, dass erzeugt nicht die Software autonom, sondern der oder die Designer des Spiels. Die Software ist nur eine Art Trägermedium der Vision des Spiels.

Die große Ähnlichkeit von Spiele Software mit Filmen hat zur Folge, dass bei der Frage, wie mit Urheberrechten umgegangen wird und welche Lizenzen sinnvoll sein können, weniger auf das geachtet wird, was sonst in der Software Industrie Anwendung findet. Inspiration sind hier eher Verfahrensweisen aus der Filmindustrie. Es ist in diesem Zusammenhang zu beobachten, dass Filmstudios selbst in die Entwicklung von Computerspielen einsteigen, auch bekannte Regisseure wie Steven Spielberg werden zunehmend darin eingebunden.[41] Auch die verwendeten Werkzeuge gleichen sich in beiden Bereichen aneinander an, wie Waugh (2005) am Beispiel Lucas Arts darstellt. Besonders deutlich wird dies bei vollständig im Computer animierten Filmen.

Interessant ist auch eine Betrachtung der Entwickler von Open-Source-Spielen und eine entsprechende Einordnung in die Finanzierungsmodelle aus dem zweiten Kapitel. Nach dem derzeitigen Stand sind Spiele, die vollständig, also inklusive Content, unter einer Freien Lizenz stehen, von der Gruppe der Freizeitprogrammierer geschrieben. Dies erklärt sich auch darin, dass die Entwicklung von Computerspielen durch intrinsische Faktoren motiviert werden kann, wie Gumhold und Weber (2004) feststellen. Diese besondere Motivation bei der Spieleentwicklung liegt darin begründet, dass die Benutzung von Spiele Software im Normallfall zur eigenen Unterhaltung erfolgt, im Gegensatz zu anderer Software, die oft aufgrund äußerer Zwänge erfolgt. Weil eine intrinsische Motivation für die Anwendung von Spielen vorliegt, wird dadurch auch eine Motivation zur Entwicklung dieser Spiele geschaffen. Die anderen Entwicklergruppen sind im Gegensatz zu anderer Software nicht im Bereich der vollständig Freien Spiele vertreten.

Ein bedeutendes Problem für Open-Source-Spiele liegt aber in etwas begründet, was Lichtl und Wurzer (2001) festgestellt haben: Wenn man ein Spiel produzieren will, das sich gut verkauft, muss man etwas herstellen, das exzellent ist. Es reicht

[41] Heise Newsticker dazu: http://www.heise.de/newsticker/meldung/64921 (18.10.2005).

nicht, etwas zu schaffen, was nur funktioniert, wie es bei Anwendungssoftware der Fall ist. Ähnliches stellt auch Laitinen (2005) heraus, wenn es um die Nutzerschnittstelle geht. Dies ist durch die größere Konkurrenz im Spielebereich und den Anspruch eines Spieles, Spaß zu machen, begründet. Während es zum Beispiel nicht allzu viele verschiedene Textverarbeitungen gibt, ist die Auswahl bei First-Person-Shootern deutlich größer[42]. Bei einer Büroanwendung muss sich der Anwender im Allgemeinen mit Unzulänglichkeiten abfinden, bei einem Spiel hat er die Wahl seine Freizeit anders zu gestalten.

Giusti (2006) stellt fest, dass die Schwierigkeiten von Open-Source-Projekten im Spielebereich auch in einem weiteren Punkt deutlich werden: Während die bekannten und erfolgreichen Open-Source-Projekte alle das Ergebnis der Zusammenarbeit von größeren Gruppen sind, weisen Freie Spieleprojekte meistens nur einen sehr kleinen Entwicklerkreis auf. Oftmals ist sogar nur ein einzelner Entwickler mit einem bestimmten Projekt beschäftigt. Ein einzelner Entwickler kann aber von den bereits aufgeführten Vorteilen der Open-Source-Entwicklung kaum profitieren.

[42] Es gibt Schätzungen, dass über 6000 First-Person-Shooter existieren.

4 Qualitative Untersuchung ausgewählter Computerspiele

Nachdem Geschichte und Grundlagen sowohl von Freier Software als auch von Spielen vorgestellt sind, wird im folgenden Abschnitt untersucht, wie sich die beiden Bereiche miteinander verknüpfen lassen, also wie sich Prinzipien aus der Open-Source-Entwicklung auf die Spieleentwicklung übertragen lassen. Zu diesem Zweck wird anhand von Beispielen aufgezeigt, wie sich Teile aus dem Open-Source-Prozess im Bereich der Spieleentwicklung wiederfinden.

Für die Betrachtung der Beispiele wurden neben allgemein verfügbaren Quellen auch Interviews mit Entwicklern der einzelnen Projekte durchgeführt, teilweise per E-mail, teilweise in mündlicher Form. Diese werden, soweit möglich und sinnvoll im Anhang abgedruckt. Für die Beispiele Neverweinter Nights und Quake war es nicht möglich, Interviews durchzuführen, es waren in diesen Fällen aber ausreichend andere Quellen vorhanden, um sie dennoch in der Untersuchung belassen zu können.

Weil der Spielebereich sehr schnellebig ist, sind für aktuelle Themen, die dieses Gebiet betreffen, oft nur wenige oder gar keine wissenschaftliche Quellen verfügbar. Aus diesem Grund wurde auch auf andere Veröffentlichungen, wie zum Beispiel Webseiten von Entwicklern, zurückgegriffen, sofern relevante Informationen nur aus solchen Quellen zu erhalten waren.

Die im Folgenden betrachteten sechs Beispiele von Spielen stammen aus verschiedenen Kategorien. Dies ist wichtig, weil zu erwarten ist, dass unterschiedliche Spielegenres auch verschiedene Herangehensweisen an offenen Quellcode bedingen. Auch die Hintergründe der Entwicklung sind sehr unterschiedlich, zum Teil besteht ein kommerzielles Interesse, zum Teil handelt es sich um nicht gewinnorientierte Projekte. Daher haben sie auf sehr unterschiedliche Weise mit der Thematik der Freien Software zu tun.

Wie aus Tabelle 4.1 zu ersehen ist, sind die Spiele alle bereits mehrere Jahre alt, es wurde aber darauf geachtet, dass alle Titel weiterhin in Benutzung sind, also nicht bereits das Ende ihres Lebenszyklusses erreicht haben. Damit haben alle betrachteten Titel eine Besonderheit, die sie von den meisten Spielen unterscheidet, denn der übliche Lebenszyklus eines Computerspiels ist nur wenige Monate lang. Dieses lange Leben hat bei den einzelnen Titeln unterschiedliche Gründe. Bei den Titeln Freeciv und Planeshift liegt dies daran, dass entsprechend dem Basarmodell bereits

Name des Spiels	Erstveröffentlichung	von Beginn an OS	kommerziell
Neverwinter Nights	2002	Nein	Ja
Freeciv	1995	Ja	Nein
Quake	1996	Nein	Ja
Planeshift	2000	Ja	Nein
Glest	2003	Nein	Nein
Nebula Engine	1998	Ja	Ja

Tabelle 4.1: Übersicht der Beispiele, Daten aus eigener Forschung

die frühe Entwicklung dieser Titel öffentlich stattfand, in der Lebenszeit ist hier also die Entwicklungszeit enthalten, die auch bei proprietären Titeln mehrere Jahre betragen kann. Bei Quake und Nebula handelt es sich primär um Engines, die ohnehin eine längere Lebensdauer aufweisen. Wieso Neverwinter Nights und Glest sich über längere Zeit halten konnten, muss in der Untersuchung geklärt werden, allerdings sind diese beiden Spiele auch erst deutlich später als die anderen Beispiele erschienen.

4.1 Neverwinter Nights

Das von der kanadischen Firma BioWare entwickelte und im Jahr 2002 bei Atari erschienene Rollenspiel Neverwinter Nights[43] wurde mehr als zwei Millionen Mal verkauft und hat auch vier Jahre nach der Veröffentlichung noch eine sehr große Zahl von Spielern.[44]

Auf den ersten Blick sieht dieses Programm nach einem klassischen proprietären Produkt aus. Die Lizenz für das Spiel ist auch eine proprietäre mit den üblichen Verboten des Kopierens und Veränderns. Der Sourcecode steht ebenfalls nicht zur Verfügung. Bei diesem Programm kommen aber einige Besonderheiten zum Tragen, die Spiele von anderen Programmen unterscheiden. Dadurch werden in diesem Fall interessante Open-Source-Ansätze ermöglicht.

Neverwinter Nights ist sehr modular aufgebaut. Es gibt Spielmodule, ein Toolkit um solche Module zu erstellen und eine Gameengine, in der diese Module gespielt werden können. Interessant ist nun die Lizenz, soweit sie das Toolkit betrifft. Damit erstellte Module dürfen weitergegeben werden, soweit dies kostenlos geschieht. Für sich genommen ist das noch nichts Besonderes, auch andere Spiele, die ein Toolkit enthalten, erlauben die kostenlose Weitergabe der Modifikationen. Bemerkenswert

[43] Die offizielle Homepage des Spiels ist zu finden unter der URL: http://nwn.bioware.com/ (28.03.2006).

[44] Die Homepage des Spiels zeigt die jeweils aktuell online spielenden Spieler an. Diese betragen regelmäßig mehrere tausend, obwohl nur ein kleiner Teil der Spieler überhaupt online spielt.

ist aber die Tatsache, dass auch die Module, die mit dem Originalspiel mitgeliefert werden,[45] im Toolkit betrachtet und verändert werden können, wenn auch in diesem Fall die Weitergabe nicht gestattet ist. Jedes Modul bringt somit seinen Source Code mit sich. Dadurch wird ein Vorteil von Open-Source-Software ausgenutzt, denn das Programm ist an die eigenen Bedürfnisse anpassbar.

Weiterhin ist auffallend, dass ein großer Teil der Funktionalität des Spieles nicht in der Black Box der Gameengine versteckt ist, sondern in Form einer der Programmiersprache C ähnlichen Scriptsprache im Sourcecode vorliegt. Somit lassen sich grundlegende Funktionen des Spieles verändern und diese Veränderungen weitergeben. Die Lizenz, unter der Neverwinter Nights vertrieben wird, erlaubt auch ausdrücklich eine kostenlose Weitergabe von solchen selbst erstellten Skripten. Dies ermöglichte zum Beispiel die Verbreitung von veränderten KI-Skripten, wodurch sich das Verhalten einiger durch den Computer gesteuerten Figuren deutlich verbessern ließ.

Diese Veränderungsmöglichkeiten wurden bereits vielfältig genutzt. Modifizierte Versionen des Programms werden zum Beispiel für interaktiven Geschichtsunterricht an Schulen eingesetzt.[46] Tan et al. (2005) berichten, dass es durch die Verknüpfungsmöglichkeiten des Spiels mit externer Software sehr vereinfacht wird, Lehrsysteme einzubinden, die noch weiterreichende Möglichkeiten bieten. Ein anderes Beispiel für die vielfältigen Nutzungsmöglichkeiten sind Machinima Filme, die mit Hilfe von Neverwinter Nights erstellt werden. Während die meisten solcher Filme nur sehr kurz sind, befindet sich auch ein Film in der Entwicklung, der Spielfilmlänge erreichen soll.[47] Dieser Film wiederum entsteht unter einer Creative-Commons-Lizenz, wird also selbst unter einer Open Source ähnlichen Lizenz verbreitet.

In Repositories[48] gibt es nicht nur tausende von Modulen, sondern auch Codeschnipsel und Algorithmen, die frei getauscht und gegenseitig verbessert werden. Jeder hat die Möglichkeit, das Modul eines anderen im Toolkit zu öffnen und seinen Bedürfnissen anzupassen. Ein großer Teil der bereits über 4000 veröffentlichten Module greift nicht nur auf die Inhalte des Originalspiels zurück, sondern enthält wiederum von anderen Nutzern erstellten Content.[49] Um diese durch Nutzer erstellten Erweiterungen zu fördern hat Bioware umfangreiche Dokumentationen der API der Engine zur Verfügung gestellt.[50]

[45] Man könnte diese mitgelieferten Module auch als das eigentliche Spiel bezeichnen.

[46] Zum Beispiel zur Darstellung des Lebens und der Ereignisse zur Zeit des Amerikanischen Unabhängigkeitskrieges (URL: http://www.educationarcade.org/modules.php?op=modload&name=Sections&file=index&req=viewarticle&artid=9&page=1 (17.10.2005)).

[47] Es handelt sich um Bloodspell, zu finden unter http://www.bloodspell.com/ (17.10.2005).

[48] Das größte dieser Repositories ist das Neverwinter Vault des IGN Network, zu finden unter http://nwvault.ign.com/ (17.10.2005).

[49] Es ist allerdings festzustellen, dass bei vielen Nutzern die Beachtung von Urheberrechten nur eine geringe Rolle zu spielen scheint. Gerade bei den in den Repositories angebotenen Sound- und Musikdateien sind häufig offensichtliche Verletzungen von Urheberrechten zu beobachten.

[50] Diese sind unter der URL http://nwn.bioware.com/developers/ (17.10.2005) zu finden.

Sarvas et al. (2005) bemerken, dass BioWare durch diesen von Nutzern erstellten Content nur profitieren kann. Nicht nur wird das Spiel selbst attraktiver, weil mehr Content zur Verfügung steht, sondern zusätzlich hat BioWare durch die Lizenz des Toolkits auch das Recht, diesen Content selbst zu vermarkten und somit direkten Gewinn daraus zu ziehen. Diesen zweiten Schritt ist die Firma bis jetzt allerdings nicht direkt gegangen. Gründe dafür könnten darin liegen, dass sich die Ersteller des Contents betrogen fühlen könnten, wodurch eine negative Reputation entstehen würde. Weiterhin müsste BioWare selbst aufwendig prüfen, ob der Content frei von Rechten Dritter ist. Es wurden aber bereits einzelne Nutzer, die zuvor herausragenden Content produziert hatten, entweder fest eingestellt oder für die Erstellung von neuen Inhalten für das Spiel bezahlt. Zudem veranstaltet BioWare in unregelmäßigen Abständen Wettbewerbe, bei denen unter unterschiedlichen Bedingungen Content mit Hilfe des Toolkits produziert werden soll. Diese dienen ausdrücklich dazu, um neue Mitarbeiter aus der Community zu finden.[51]

Für BioWare ist ein auf die Community ausgerichteter Ansatz ein Kernbestandteil der Geschäftspolitik, erklärte der Community Manager der Firma, Jay Watamaniuk gegenüber DeMaria (2005). Offensichtlich wird dies an der intensiv betriebenen Kommunikation mit den Nutzern über mehrere Webforen und dem über Jahre bestehenden Support für publizierte Spiele. Dieser Ansatz nutzt gezielt Vorteile aus, von denen auch das Basarmodell der Softwareentwicklung profitiert. So werden dadurch die Nutzer motiviert, Fehlerberichte einzusenden und sich gegenseitig bei Problemen weiter zu helfen. Zusätzlich schafft er eine Identifikation der Spieler mit den Entwicklern, was eine dauerhafte Kundenbindung zur Folge hat und somit in der Zukunft die Verkaufschancen neuer Spiele erhöht. Allerdings ist dieses Vorgehen auch sehr aufwendig an Ressourcen für BioWare, denn die fortgesetzte Pflege älterer Spiele kostet Geld, ebenso wie die Kommunikation mit den Nutzern.

Andererseits ist dieses intensive Bemühen um die Community ein wichtiger Grund, warum sich das Spiel über einen so langen Zeitraum erfolgreich verkaufen lässt. Die positiven Netzwerkeffekte der Nutzergemeinschaft sorgen dafür, dass der Kauf des Spiels auch heute noch interessant ist, obwohl der ursprüngliche Erscheinungstermin bereits vier Jahre zurück liegt und die verwendete Technik somit nicht mehr dem aktuellen Stand entspricht. Die ursprüngliche Fassung wird deutlich reduziert angeboten, aber in einer aktualisierten Version mit beiden erschienenen Erweiterungen und mehreren zusätzlichen Modulen wird Neverwinter Nights noch immer zum Vollpreis verkauft.

Seit Ende 2004 vertreibt BioWare auch zusätzliche Spielmodule einzeln über einen Online-Shop.[52] Bei diesen wurde die Open-Source-Funktionalität aufgegeben, das heißt diese Module sind im Toolkit weder betrachtbar noch veränderbar, um einen

[51] Webseite eines solchen, von BioWare veranstalteten Wettbewerbs unter der URL: http://www.bioware.com/biozone/articles/2005_11_30_WritingContest/index.html (02.03.2006).

[52] Dieser ist zu finden unter der URL: http://store.bioware.com/products/ (27.03.2006).

Schutz vor nicht lizenzierten Kopien zu gewährleisten. Dieser Schritt wurde von der Community zum Teil heftig kritisiert, dient aber andererseits dazu, die fortgesetzte Pflege und Weiterentwicklung des Spiels zu finanzieren. Auch findet ein großer Teil des neuen Contents dieser Module später Eingang in den Bestand des eigentlichen Spiels, so dass sie wieder im Toolkit verwendet werden können.

Ein großes Hindernis bei einer eventuellen zukünftigen Freigabe des gesamten Quellcodes von Neverwinter Nights als Open Source ist die Lizenzierungsproblematik.[53] Zum einen sind Teile der Engine, zum Beispiel die Ausgabe von Videosequenzen im Bink Format und die Miles Sound Engine, von externen Firmen lizenziert und müssten somit vor einer Veröffentlichung der Quellen entfernt werden. Zum anderen ist der Content ebenfalls unter Lizenz hergestellt, denn Neverwinter Nights implementiert die AD&D Regeln, deren Rechte bei Wizards of the Coast liegen. Die Verwendung dieser Regeln unterliegt der d20 Lizenz[54], die ausdrücklich die Verwendung in Computerspielen ausschließt, weil diese Verwendung exklusiv an Atari lizenziert wurde.

4.2 Civilization und Freeciv

Im Jahr 1991 erschien bei Microprose mit dem Titel „Civilization“ ein rundenbasiertes Strategiespiel mit dem Ziel, die gesamte Geschichte der Menschheit zu simulieren.[55] Es handelt sich um ein reines Einzelspielerspiel. Da das Spiel als proprietäres Produkt entwickelt wurde ist der Quellcode nicht zugänglich.

Unter anderem dadurch bedingt, dass einzelne Aspekte des Spieles sich relativ leicht verändern ließen[56], war Civilization auch noch vier Jahre später auf dem Markt. Der fehlende Mehrspielermodus wurde allerdings als großer Mangel empfunden. Das 1995 erschienene CivNet sollte diesen Mangel beheben, war allerdings nicht sehr erfolgreich. Gleichzeitig wurde das Open-Source-Projekt „Freeciv“ gegründet, dessen erstes Ziel es war, einen Mehrspieler Klon des Spieles zu entwickeln.[57]

Weil bereits der kommerzielle Originaltitel nur sehr einfache Grafiken und wenig Soundeffekte aufwies, konnte das Freie Projekt in diesen Bereichen schnell einen ähnlichen Stand erreichen. Die konsequente Auslegung auf den Mehrspielermodus führte zu einem deutlichen Vorsprung in diesem Bereich. Dafür wurde die Entwicklung einer

[53] Eine Diskussion zu diesem Thema einschließlich eines Entwicklerkommentars ist unter http://nwn.bioware.com/forums/viewtopic.html?topic=444232&forum=42 (23.01.2006) zu finden.
[54] Der Text dieser Lizenz ist zu finden unter der URL: http://www.wizards.com/d20/files/d20stlv6.rtf (24.10.2005).
[55] Die Zeitangaben zu den Civilization und Freeciv Titeln stammen von der Freeciv Webseite unter der URL: http://www.freeciv.org/index.php/Timeline (28.03.2006).
[56] Viele Elemente waren in einfachen Textdateien, die leicht angepasst werden konnten, enthalten.
[57] Interview mit Vasco Alexandre Da Silva Costa (siehe Anhang A.1.2).

künstlichen Intelligenz als Spielgegner für den Einzelspielermodus erst mit späteren Versionen in Angriff genommen.[58]

Das kommerzielle Civilization erschien 1996 in einer zweiten Version. Neben leichten inhaltlichen Änderungen am Spiel wurde vor allem die Grafik deutlich verbessert und die Entwicklung eigener Modifikationen vereinfacht. Später brachte Microprose zwei Erweiterungen zu Civilization 2 auf den Markt, die auf diesen Veränderungsmöglichkeiten basierten und sie noch erweiterten. Aber auch die neue Version bot keine Möglichkeit, das Spiel mit mehreren menschlichen Spielern zu spielen.

Die Entwicklergemeinschaft von Freeciv reagierte auf den neuen Titel, indem sie die Möglichkeit anboten, zwischen den Spielmodi beider kommerziellen Versionen zu wählen.[59] Grafisch fiel der freie Titel deutlich hinter das proprietäre Vorbild zurück. Erst nach einigen Jahren konnten die Fortschritte in diesem Bereich auch in Freeciv genutzt werden. Der Hauptgrund für diese langsamere Entwicklung lag in der mangelnden Verfügbarkeit entsprechender Grafiken unter einer freien Lizenz.[60] Deshalb mussten Grafiken oft von den Programmierern selbst gezeichnet werden. Bei Soundeffekten und besonders Hintergrundmusik war der Mangel an verfügbarem freien Material noch stärker ausgeprägt.[61] Aus diesem Grund verzichteten die Freeciv-Entwickler zunächst auf die Verwendung von akustischen Elementen.

Als 2001 der dritte Teil des kommerziellen Civilization erschien, hatte sich die freie Variante dem Ziel der Kompatibilität mit der zweiten Version sehr genähert. Dies ging so weit, dass sich Grafiken und Sounds des Originals zum Teil im Open-Source-Programm verwenden ließen. Während die kommerzielle Variante weiterhin über eine aufwendigere Gestaltung verfügte, wurde das Interface von Freeciv im Allgemeinen als praktischer angesehen. Die KI von Civilization war vielseitiger als die des freien Titels, dieser konnte aber mit einer höheren Spielstärke aufwarten. Die Verfügbarkeit auf mehreren Plattformen und vor allem der Mehrspielermodus waren weitere Pluspunkte des freien Projekts.

Ab diesem Zeitpunkt ist zu beobachten, dass sich die Zielsetzung der Freeciv-Entwickler verändert hat. Es herrscht nicht mehr das Ziel vor, einen möglichst genauen Klon des kommerziellen Vorbilds zu erstellen.[62] Vielmehr findet eine eigenständige Weiterentwicklung statt. Dies liegt möglicherweise daran, dass viele Fans des ursprünglichen Civilization vom dritten Teil der Serie enttäuscht waren, zumal dieser

[58] Interview mit Vasco Alexandre Da Silva Costa (siehe Anhang A.1.2).
[59] Interviews mit Per Inge Mathisen und mit Vasco Alexandre Da Silva Costa (siehe Anhänge A.1.1 und A.1.2).
[60] Interview mit Per Inge Mathisen (siehe Anhang A.1.1).
[61] Interviews mit Per Inge Mathisen und mit Vasco Alexandre Da Silva Costa (siehe Anhänge A.1.1 und A.1.2).
[62] Interviews mit Per Inge Mathisen und mit Vasco Alexandre Da Silva Costa (siehe Anhänge A.1.1 und A.1.2).

in seiner ursprünglichen Version noch immer keine Mehrspieler-Möglichkeit aufwies.[63] Von der Entwicklerseite her ist seitdem eine größere Offenheit für neue Ideen vorhanden, die so bisher nicht in Spielen der Civilization Reihe verwendet wurden. Zudem werden andere „world-building“ rundenbasierte Strategiespiele, wie Master of Orion und Master of Magic dahingehend betrachtet, ob sich Konzepte von ihnen in Freeciv einbauen lassen.[64]

Für Schwachstellen im Spielkonzept[65] werden unabhängig vom „offiziellen“ Weg Lösungsansätze entwickelt. Zusätzlich werden Ideen aus anderen rundenbasierten Strategiespielen in das Civilization Konzept eingebaut. Der Versuch, die immer aufwendigeren Grafiken des Vorbilds nachzuahmen wurde zurückgestellt und zunächst mehr Wert auf eine Erhöhung der Funktionalität gelegt. Allerdings wird weiter versucht, die „Spielregeln“ der Vorbilder wahlweise zu emulieren.[66] Zumindest für die ersten beiden Civilization Titel ist dies auch recht genau möglich, wenn auch nicht mehr die Standardeinstellung von Freeciv.

Hierbei ist es den Entwicklern besonders wichtig, eine möglichst weite Veränderbarkeit des Spiels, ohne dass Änderungen am Quellcode nötig sind, zu ermöglichen. Dies wurde häufig von Spielern gefordert, die eigene Modifikationen des Spieles erstellen wollten. Hierbei folgt Freeciv den immer größeren Modifikationsmöglichkeiten des Originals. Interessant ist, dass es ja durch die Verfügbarkeit des Quellcodes auch ohne diese Entwicklungen bei Freeciv durchaus möglich wäre, umfangreiche Veränderungen am Spiel selbst durchzuführen. Offensichtlich gibt es eine Gruppe von Spielern, die nicht die Fähigkeit oder den Willen hat, sich mit der Programmierung des Spieles an sich auseinanderzusetzen, die andererseits aber bereit ist, einen hohen Zeitaufwand auf die Modifikation des Spiels zu verwenden.

Auch das Erscheinen der vierten Version der kommerziellen Civilization Reihe wird von den Freeciv-Entwicklern genau beobachtet. Primär werden dabei weiterhin geänderte Funktionalitäten betrachtet, weniger äußere Aspekte wie Grafik und Sound. Prinzipiell ist die Bereitschaft vorhanden, Neuerungen auch dieses Teils zu übernehmen, wenn sie sich in der praktischen Erfahrung bewähren sollten. Das Ziel, einen echten Klon des Vorbilds zu entwickeln, wird aber nicht mehr verfolgt. Vielmehr sehen sie sich in der Entwicklung eines eigenständigen Spiels mit eigenen Stärken.[67]

Unter Spielern der kommerziellen Civilization Version ist eine große Bereitschaft

[63] Interview mit Vasco Alexandre Da Silva Costa (siehe Anhang A.1.2).
[64] Interviews mit Per Inge Mathisen und mit Vasco Alexandre Da Silva Costa (siehe Anhänge A.1.1 und A.1.2).
[65] Die wohl bedeutendste Schwachstelle ist ICS – Infinite City Sprawl, also die Spielstrategie so viele Städte, wie nur möglich zu bauen, ohne diese zu entwickeln. Unter Freeciv Entwicklern wird diese Strategie auch als „smallpox“ bezeichnet.
[66] Interview mit Per Inge Mathisen (siehe Anhang A.1.1).
[67] Interviews mit Per Inge Mathisen und mit Vasco Alexandre Da Silva Costa (siehe Anhänge A.1.1 und A.1.2).

zu erkennen, eigenen Content für das Spiel zu erstellen, seien es eigene Szenarien oder veränderte Grafiken.[68] Dadurch ist eine große Menge von nicht kommerziell entstandenem Content verfügbar. Allerdings sind die Lizenzbedingungen dieses Contents meist nicht eindeutig geregelt, wohl auch weil an den rechtlichen Aspekten bei den Spielern, die ihn erstellen, wenig Wissen oder Interesse in diesem Bereich besteht. Deswegen ergibt sich für die Entwickler von Freeciv die Schwierigkeit, diesen Content nicht ohne weiteres verwenden zu können,[69]auch wenn die Ersteller im Allgemeinen mit dieser Verwendung keine Schwierigkeiten haben. Zunächst müssen sie über die Auswirkungen einer Freien Lizenz informiert werden und dann ihren Content unter eine solche Lizenz stellen.

Ein weiteres Problem besteht darin, dass diese Modder von kommerziellen Spielen, gerade auch von den kommerziellen Civilization Titeln, teilweise Grafiken und Sounds verwenden, ohne die entsprechenden Rechte dazu zu besitzen. Um zu vermeiden, dass solcher Content versehentlich Einzug in Freeciv erhält, müssen die Entwickler zunächst zu jedem Teil des Contents den echten Urheber ausfindig machen.[70] Diese Kontaktaufnahme ist zum Teil sehr zeitaufwendig, weil Modder oft nur unter Pseudonymen in Foren auftreten und sie diese nach einiger Zeit zum Teil nicht mehr besuchen.

Eine große Schwierigkeit bei größeren Neuerungen im grafischen Bereich liegt in der nötigen Koordination mehrerer Bereiche. Es muss neuer Code geschrieben werden, um die neuen grafischen Funktionen zu ermöglichen, gleichzeitig müssen aber die neuen Grafiken vorhanden sein, um den neuen Code verwenden zu können. Hierfür müssen sich Grafiker und Programmierer trotz unterschiedlicher Arbeitsweisen eng miteinander abstimmen.

Während die Entwickler von Freeciv das Spiel auch spielen, sind nicht alle Spieler auch an der Entwicklung des Programms beteiligt. Dies führt zum Teil dazu, dass die Entwickler Wünsche und Probleme der reinen Spieler nicht erkennen. Die Basar-Methode der Softwareentwicklung fordert darum auch als einen der Kernpunkte, dass die reinen Nutzer eines Programms in die Entwicklung einbezogen werden. Die ursprünglichen Methoden der Kommunikation mittels Mailinglisten und Bug-Tracking-System sind für viele Spieler, die technisch weniger erfahren sind, abschreckend und ungewohnt. Um auch von diesen Nutzern Feedback zu erhalten versuchen die Entwickler mittels eines Webforums den Austausch zu verbessern. Die Einführung dieses Webforums hatte den Nebeneffekt, dass sich vermehrt auch Grafiker an der Entwicklung von Freeciv beteiligten.[71] Diese Form der Kommunikation über Foren ist in den Gemeinschaften von Grafikern offenbar üblich, so dass hierdurch Schranken abgebaut

[68] Ausgetauscht werden diese über unabhängige Foren, wie Apolyton, zu finden unter der URL: http://apolyton.net/ (27.03.2006).
[69] Interview mit Per Inge Mathisen (siehe Anhang A.1.1).
[70] Interview mit Per Inge Mathisen (siehe Anhang A.1.1).
[71] Interview mit Per Inge Mathisen (siehe Anhang A.1.1).

werden konnten.

Zu Beginn bestand das Entwicklerteam von Freeciv nur aus Programmierern. Erst zum jetzigen Zeitpunkt, nach einigen Jahren der Entwicklung, gelang es, vermehrt Grafiker mit einzubinden. Auffällig ist, dass ein gewisser Bestand an Grafikern im Webforum dazu führte, dass sich weitere Grafiker leichter einbringen konnten. Der Entwickler Da Silva Costa führt dies darauf zurück, dass zusätzlich zu den unterschiedlichen technischen Gewohnheiten von Programmierern und Grafikern, auch die verwendete Sprache in den beiden Gruppen unterschiedlich ist.[72] Um die zunächst nicht vertretene Gruppe von Grafikern in die Entwicklung einzubinden, mussten zunächst diese Barrieren überbrückt werden. Dazu waren Entwickler notwendig, die sich in beiden Gruppen heimisch fühlten.

Aus wissenschaftlicher Sicht ist Civilization und besonders Freeciv aus mehreren Gründen interessant. Zum einen besteht aus den Sozialwissenschaften ein großes Interesse an diesem Spiel, da es ja den Anspruch erhebt, eine Simulation der Menschheitsgeschichte zu sein. Es wird jedoch vielfach die Ungenauigkeit und Verfälschung in der Geschichtsdarstellung kritisiert. Zum Beispiel sieht Poblocki (2002) in Civilization eine Fortschreibung einer imperialistischen Weltsicht des 19. Jahrhunderts. Nun ist Civilization in erster Linie ein Spiel und muss nicht unbedingt wissenschaftlich korrekt sein. Gerade die Open-Source-Variante Freeciv könnte aber an dieser Stelle ansetzen. Mit einer entsprechend offenen Struktur, wie sie zur Zeit ein Schwerpunkt der Freeciv Entwicklung ist, wäre es für Wissenschaftler möglich, einzelne Parameter des Spieles, wie zum Beispiel den Forschungsbaum, so zu verändern, dass das Spiel zusätzlich als wissenschaftliches Instrument nutzbar wäre. Schon jetzt ist Freeciv in vielen Punkten deutlich flexibler als es die Closed-Source-Variante sein kann.

Zum anderen ist Freeciv für die Forschung im Bereich der Künstlichen Intelligenz von Interesse. Wie Laird und van Lent (2001) darlegen, besteht zur Zeit noch ein großer Forschungsbedarf in der Entwicklung von strategisch vorgehenden Computergegnern. Das taktische Vorgehen im Spiel ist relativ leicht zu erfassen, weil die Möglichkeiten jeweils begrenzt sind. Eine längerfristige Strategie, ohne dass die KI „schummeln“ muss ist dagegen der größte Schwachpunkt in aktuellen Computergegnern. Der offene Quellcode von Freeciv ermöglicht es der Forschung, an dieser Stelle anzusetzen. Houk (2004) stellt einen solchen möglichen Ansatz vor, für den zumindest offene Schnittstellen zum Spiel benötigt werden. Er stellt fest, dass durch den offenen Quellcode die Forschungsarbeit deutlich vereinfacht wird, was mit ein Grund für die Auswahl von Freeciv als Forschungsgegenstand war.

[72] Interview mit Vasco Alexandre Da Silva Costa (siehe Anhang A.1.2).

4.3 Doom und Quake

Mit dem 1992 erschienen Wolfenstein 3D und der nachfolgenden Reihe der Quake und Doom Spiele gehört id-Soft zu den erfolgreichsten Spieleentwicklern. Diese Spiele haben das Genre der heutigen First-Person-Shooter[73] mit begründet und definiert. Zusätzlich waren diese Spiele seit Erscheinen der ersten Doom Version Pioniere im Bereich der Mehrspieler-Spiele.

Die Quake-Reihe ist ein klassisches Beispiel, wenn es um Open Source im Spielebereich geht. ID-Soft hat bei diesem Spiel wie auch bei seinen Vorgängern „Wolfenstein 3D“ (1992) und „Doom“ (1993) und seinen Nachfolgern „Quake 2“ (1997) und „Quake 3 Arena“ (1999) die Gameengine unter die GPL gestellt[74], nachdem die Nachfolgeengine auf dem Markt war. Sämtliche Grafik-, Ton-, und Leveldaten, also der Content des Spiels, bleiben aber weiterhin unter der ursprünglichen proprietären Lizenz des Spieles, nur die Engine selbst wird Open Source.

Diese Verfahrensweise kann somit zwar dazu beitragen, dass hochwertige Spiele, die auf dieser Engine aufsetzen, als Open Source verfügbar werden, allerdings wird die Engine natürlich immer erst dann Open Source, wenn sie veraltet und somit nicht mehr wirtschaftlich nutzbar ist. Ein erstes Ergebnis der Veröffentlichung des Engine Quellcodes waren aber zum Teil weitgehende Veränderungen und Verbesserungen. Durch diese wurde die Lebenszeit der entsprechenden Spiele deutlich verlängert. Im Endeffekt ergibt sich aus dieser Verfahrensweise also ein finanzieller Zugewinn für die Herstellerfirma, in diesem Fall also ID-Soft. Schließlich muss jeder, der das verbesserte Spiel spielen will, immer auch das ursprüngliche Spiel gekauft haben.

ID-Soft stellt sich somit als ein Spieleentwickler dar, der das typische Open-Source-Finanzierungsmodell von Softwarefirmen nutzt. Ein Teil der Software wird freigegeben, wodurch sich der andere Teil besser verkaufen lässt. Dies gilt nicht nur für das Endprodukt des Spiels an sich, sondern besonders auch für die Engine. Externe Spieleentwickler, die vor der Auswahl einer Engine für ein eigenes Spiel stehen, können mit der älteren Open-Source-Version Erfahrungen sammeln und entscheiden sich somit wahrscheinlicher auch für die Verwendung der aktuellen Closed-Source-Variante. Auch kann ID-Soft auf diese Art den Markt, auf dem sie ihre Engine absetzen können, vergrößern. Neue, kleine Entwicklerstudios, die sich die hohen Kosten für die Lizenzierung der aktuellen Engine nicht leisten können, haben die Möglichkeit, einen ersten Titel mit der alten Engine herauszugeben. Für weitere Titel sind dann durch die

[73] Mit First-Person-Shooter werden Aktion Spiele bezeichnet, die der Spieler aus der „Ich“-Perspektive spielt. Bei Spielen dieser Art bewegt sich der Spieler frei durch eine dreidimensionale Spielwelt, in der er durch Menschen oder Computer gesteuerte Gegner ausschalten muss.

[74] Genauer gesagt steht die Engine der Quake-Titel unter einer Duallizenz, entweder der GPL oder gegen Bezahlung unter einer proprietären Lizenz, die es dem Lizenznehmer erlaubt, seine Modifikationen geheim zu halten. Genauere Informationen dazu unter der URL: http://www.idsoftware.com/business/technology/ (06.03.2006).

Gewinne des ersten Spiels die Gelder vorhanden, die jeweils aktuelle Engine zu lizenzieren.

Inzwischen gibt es aber vermehrt auch Spiele, die komplett Open Source sind und die auf eine verbesserte Quake 1 oder Quake 2 Engine aufsetzen. Zumeist ist bei diesen Spielen der Content frei erhältlich, selbst aber nicht frei im Sinne von Freier Software. Es gibt aber auch hier Ansätze, freien und nicht nur kostenlosen Content zu schaffen. Diese sind jedoch von der Entwicklung noch deutlicher hinter dem Stand der Technik zurück, als es die freigegebenen Engines ohnehin schon sind. So wird mit Freedoom[75] Content für die Doom Engine entwickelt und mit Open Quartz[76] solcher für die Quake 1 Engine. Währenddessen gibt es bereits erste Projekte, die kostenlosen, aber nicht freien Content für die Quake 3 Engine erstellen.[77]

Quake kann als ein Spiel angesehen werden, bei dem die Entwickler von Beginn an ein Open Source ähnliches Modell im Hinterkopf hatten, wie Cummins (2002) feststellt. Als Folge davon waren dieselben Level Editoren, die auch die ursprünglichen Designer bei der Entwicklung verwendeten, von Beginn an für die Nutzer verfügbar. Dies führte zu einer raschen Entwicklung durch die Nutzer nicht nur von neuen Leveln, sondern auch von ganz anderen Anwendungsbereichen. Zum Beispiel ist es möglich, mit der Quake Engine einen virtuellen Treffpunkt aufzubauen, der als dreidimensionaler Internetchat verstanden werden kann.

Cummins sieht hierin die Möglichkeit für ein ganz neues Geschäftsmodell, indem die Umgebung des Spieles für E-Commerce Anwendungen benutzt wird. Nutzer könnten so dreidimensional durch virtuelle Geschäfte gehen und Modelle von realen Waren betrachten. Der Verkauf dieser Waren könnte ebenso innerhalb dieses virtuellen Ladens stattfinden. Eine andere Möglichkeit besteht für Scheiblauer (2004) darin, dass die Engine in der Architektur eingesetzt wird, um so eine bessere Vorstellung eines geplanten Bauwerks zu ermöglichen, als dies mit einem konventionellen Modell möglich ist.

Im Rahmen von Quake und anderen First-Person-Shootern mit ähnlich offener Architektur, also vorhandenen umfangreichen Editoren und öffentlichen Schnittstellen für Mod-Entwickler, haben sich umfangreiche Communities gebildet. Guettler und Johansson (2003) sehen diese Communities als vergleichbar mit Open-Source-Gemeinschaften. Sie führen dies am Beispiel von Counter Strike aus, bei dem zu erkennen ist, dass selbst erstellte Karten für das Spiel nicht nur zur freien Verfügung gestellt werden, sondern auch eine Art Peer Review durchlaufen. Gemeinsam werden eventuelle Probleme der Karten besprochen und Fehler behoben. Die Frage des

[75] Webseite zu Freedoom unter der URL: http://freedoom.sourceforge.net/ (29.12.2005).

[76] Webseite des Open-Quartz-Projekts unter der URL: http://openquartz.sourceforge.net/ (06.03.2006).

[77] Als Beispiel sei hier Tremulous genannt, auch wenn zum jetzigen Zeitpunkt noch keine funktionierende Version veröffentlicht wurde, die lediglich den Quake 3 Quellcode benötigt. URL: http://tremulous.net/ (06.03.2006).

Urheberrechts spielt in diesen Gemeinschaften keine große Rolle, das Ergebnis der gemeinschaftlichen Entwicklung steht an vorderster Stelle.

Auch Richards (2003) sieht die Übernahme von Open-Source-Prinzipien und die Veröffentlichung des Engine-Quellcodes als Basis für diese Gemeinschaften an. Noch wichtiger war dies aber für das Online-Spielen überhaupt, weil nur so zahlreiche beliebte Mehrspieler-Mods entwickelt werden konnten. Auch das Genre der Machinima Filme konnte sich nur durch die umfangreichen Veränderungsmöglichkeiten an diesen Engines wirklich entwickeln. So zeigt Wehn (2004), dass es genau die Veränderungsmöglichkeiten am ersten Titel der Quake-Reihe waren, die zur Entstehung der Machinima führten. Während zunächst noch mittels einer integrierten Aufzeichnungsfunktion Szenen aus dem Spiel mitgeschnitten wurden (sogenannte Demos), folgte bald darauf die Entwicklung einer Software, mit der solche Demos verändert werden konnten. Dies führte im August 1997 zum ersten Machinima Film „Diary of a Camper“, der auf der Quake Engine basierte. Die Freigabe des Quake Quellcodes eröffnete schließlich ganz neue Möglichkeiten bei der Gestaltung von Machinima.

Eine weitere wichtige Eigenschaft von Quake und anderen First-Person-Shootern ist die zunehmende Wahrnehmung als Sportart. Es gibt Weltmeisterschaften für verschiedene Computerspiele, bei denen zum Teil hohe Geldpreise ausgesetzt sind. In Südkorea werden bereits Turniere verschiedener Computerspiele live im Fernsehen übertragen. Um eine bessere Präsentation für den Zuschauer zu erreichen, lassen sich bei quelloffenen Spielen auf diese Anwendung ausgelegte Clients entwickeln, wie Richards darlegt.

Es ist sicherlich von Vorteil, wenn ein Spiel, das in solchen Turnieren verwendet werden soll, Open-Source-Software ist. Zum einen erhöht sich dadurch die mögliche Anzahl Plattformen, auf denen das Spiel zur Verfügung steht[78], zum anderen lässt sich nur so der Quellcode des Spieles daraufhin untersuchen, dass ein Turnier wirklich fair abläuft.[79] Bisherige Verfahren, wie zum Beispiel die Software „Aequitas“ der Electronic Sports League[80], setzen dagegen nur auf zufällige Screenshots der Spielerbildschirme und eine Untersuchung von Logfiles. Zudem sind sie von ihrer Anwendbarkeit sehr eingeschränkt. Mit zunehmendem Interesse in der Bevölkerung an diesem elektronischen Sport und damit zunehmenden wirtschaftlichen Interessen, wird eine Forderung nach offenen Quellen für solche Turniersoftware immer wichtiger.

Gleichzeitig muss für den regulären Ablauf solcher Turniere aber auch gewährleistet sein, dass alle Spieler eine unveränderte Version des Spiels verwenden. Dies wird offensichtlich durch die Verwendung des Open-Source-Modells erschwert. Schließlich ist es für einen Spieler sehr viel komplizierter, ein Spiel zu manipulieren, von dem er

[78] Ein Spiel, das nur auf einer Plattform läuft, wäre im Sportkontext vergleichbar mit einer Sportart, bei der nur eine ganz bestimmte Schuhmarke verwendet werden darf.
[79] Dies könnte man mit Dopingtests im konventionellen Sport vergleichen.
[80] Webseite der ESL: http://www.esl-europe.net (22.10.2005).

nur den Binärcode besitzt, als wenn er auch Zugriff auf den Quellcode hat. Andererseits muss auf jeden Fall die Gleichheit der jeweils verwendeten Software garantiert werden, was auch bei Binärcode eine Überprüfung erfordert.

Ähnlich wie bei Freeciv für den Bereich der strategischen KI, bietet sich Quake für die Erforschung von taktischer KI an. Ebenso sind neben den üblichen Computergegnern auch KI gesteuerte Partner des Spielers denkbar. Laird und van Lent (1999) entwickelten einen umfangreichen Bot für Quake II, der in verschiedenen Bereichen deutliche Fortschritte gegenüber klassischen Computergegnern zeigt. Diese Arbeit ist nur durch offene Schnittstellen in Computerspielen möglich, am besten durch einen komplett offenen Quellcode.

Auch für andere Forschungsarbeiten, wie die Umwandlung von Wolfenstein 3D in ein Web-basiertes Spiel, die Wong (2000) beschreibt, ist ein offener Quellcode unabdingbar.

4.4 Planeshift

Ein recht neuer Bereich der Computerspiele sind die Massiv-Mehrspieler-Online-Rollenspiele (Massive-Multiplayer-Online-Role-Playing-Game – MMORPG). Obwohl es schon länger Spiele gibt, die Online gespielt werden, hat die Anzahl Spieler, die bei diesen Spielen möglich war, erst jetzt Bereiche erreicht, bei denen man von Massiv-Mehrspieler-Spielen sprechen kann. Während konventionelle Netzwerkspiele das Zusammenspiel von einer kleinen Gruppe von Spielern (im ein- oder zweistelligen Bereich) erlauben, bieten Massiv-Multiplayer-Spiele eine Möglichkeit, zu hunderten oder zu tausenden gleichzeitig in derselben Spielwelt zu spielen.

Die ersten Spiele, die eine solch große Spielerzahl erlaubten, waren die Mehrspielerdungeons (Multi-User-Dungeons – MUD). Während sich hier zwar theoretisch hunderte Spieler gleichzeitig in derselben Spielwelt bewegen konnten, erreichten doch die wenigsten tatsächlich diese Spielerzahlen. Bedingt war dies vor allem durch die noch recht geringe Verbreitung von günstigen Internetzugängen und dadurch, dass MUD rein textorientiert waren.

Graphische Mehrspielerspiele waren zunächst Action Titel. Diese erreichen zwar ein großes Publikum, sind aber im Unterschied zu einem MUD nicht persistent. Das bedeutet, die Spielwelt bleibt nur so lange erhalten, wie eine Spielrunde läuft; wenn sich ein Spieler neu anmeldet, beginnt er wieder von vorne. Im Unterschied dazu bieten persistente Spiele eine dauerhaft bestehende Spielwelt, die auch dann weiter besteht, wenn gerade kein Spieler angemeldet ist. MUD boten diese Persistenz und mit MMORPG wurde sie auch auf graphische Titel übertragen.

Um die Besonderheit des Spielprinzips der MMORPG zu erkennen, ist es wichtig festzuhalten, dass Spiele dieses Genres grundsätzlich ein offenes Ende haben. Im Ge-

gensatz zu üblichen Spielen, die irgendwann einmal durchgespielt und dann weniger interessant sind, werden MMORPG meist dauerhaft, oft über mehrere Jahre gespielt. Dabei gibt es unterschiedliche Typen von Spielern, die auf unterschiedliche Bereiche Wert legen. Ein erfolgreiches Spiel muss allen Spielertypen gleichermaßen ein interessantes Spiel bieten, um diese Spieler dauerhaft zu binden. Tveit et al. (2003) und Thawonmas et al. (2003) haben Untersuchungen zu diesen Spielertypen angestellt. Demnach gibt es vier Arten von Spielern:

Achiever Sie wollen vor allem ihren Spielercharakter verbessern und ausbauen.

Socializer Sie legen primär Wert auf Unterhaltungen mit anderen Spielern.

Killer Sie sehen die Anzahl getöteter Gegner, seien es andere Spieler oder Nicht Spieler Charaktere (NPC), im Spiel als das Wichtigste.

Explorer Sie möchten neue Orte, Gegenstände und NPC entdecken.

Um die Bedürfnisse der unterschiedlichen Typen, besonders des Explorers, zu befriedigen, sind Anbieter solcher Spiele gezwungen, kontinuierlich neuen Content für das Spiel zu produzieren. Diese kontinuierliche Weiterentwicklung ist aber genau eine der Stärken des Open-Source-Modells.

Dieses Spielegenre ist verschieden von allen anderen Arten von Computerspielen, weil es zwingend eine Client-Server-Struktur voraussetzt. Dadurch ergibt sich bei der Frage nach dem Quellcode auch eine zusätzliche Möglichkeit. Es kann nicht nur der Spieleclient offen oder geschlossen sein, dasselbe ist auch für den Server möglich. Weiterhin ist dieselbe Unterscheidung auch für den Content möglich. Schließlich gibt es hier noch unterschiedliche Finanzierungsmodelle. So können Spiele werbefinanziert werden, eine Einmalzahlung verlangen oder einen monatlichen Betrag kosten. Zona (2002) schätzt, dass der weltweite Umsatz durch diese monatlichen Zahlungen im Jahr 2006 einen Betrag von 2,7 Milliarden US Dollar erreichen wird; der Anteil der MMORPG am Gesamtumsatz der Computerspieleindustrie ist also beträchtlich. Ausführlichere Informationen zur Bedeutung dieses Spielebereichs sind im Abschnitt Online-Spiele auf Seite 63 zu finden.

Das Planeshift Projekt[81] ist nicht kommerziell ausgerichtet und folgt dementsprechend zur Zeit keinem dieser Finanzierungsmodelle. Der Quellcode sowohl des Clients als auch des Servers stehen unter der GPL, also einer freien Lizenz, der Content selbst ist zwar kostenlos, aber nur unter einer proprietären Lizenz verfügbar. Der rechtliche Träger dieses Projektes ist eine non profit Corporation namens Atomic Blue.[82]

[81] Das Projekt ist zu finden unter der URL: http://www.planeshift.it (07.10.2005).
[82] Zu finden unter http://www.atomicblue.org/ (07.10.2005).

Diese unterschiedlichen Lizenzierungen von Code und Content sind für den reinen Spieler zunächst von untergeordneter Bedeutung. Er erhält das Spiel als Gesamtpaket zum Download kostenlos von der Projekt Webseite, muss keine monatlichen Gebühren bezahlen und kann das Spiel auch beliebig oft installieren. Sobald ein Nutzer jedoch mehr mit dem Programm anfangen möchte, als es nur zu spielen, stößt er schnell an die Grenzen der Lizenz. So ist es rechtlich nicht möglich, einen eigenen Server für das Spiel aufzusetzen, auch darf er keine Inhalte abwandeln oder eine veränderte Version weiter verbreiten. Mit dem Quellcode selbst wäre dies zwar alles gestattet, doch ist dafür der komplette Content auszutauschen. Damit entspricht die rechtliche Situation bei Planeshift ungefähr derjenigen von Quake nach der Veröffentlichung des Quellcodes unter der GPL.

Im Unterschied zu Quake war bei Planeshift der Quellcode von Anfang an offen und unter einer Freien Lizenz verfügbar. Dies dient ausdrücklich dazu, externe Entwickler anzuziehen und in die Entwicklung des Projektes einzubinden. Ein weiterer Unterschied liegt darin, dass Planeshift als Grundlage die Crystal Space Engine verwendet[83], die ihrerseits unter einer Open-Source-Lizenz steht. Modifikationen, die für das Spiel an der zu Grunde liegenden Engine vorgenommen werden, müssten also auch bei einem ansonsten geschlossenen Quellcode offen gelegt werden.

Die Überlegungen, die zu dieser Art der Lizenzierung führten, werden von den Entwicklern ausführlich auf der Webseite des Projektes dargelegt.[84] Primärer Beweggrund für eine proprietäre Lizenzierung des Contents war die Angst vor möglichen Spaltungen des Projekts, wenn auch die Inhalte unter einer Freien Lizenz stehen. Interessanterweise wird als ein zweiter Grund angegeben, dass Künstler, die gute Grafiken und Sounds erstellen können, mit einer freien Lizenz nicht einverstanden wären. Dies würde zu einem Verlust an Beiträgen durch diese Künstler führen, sollte eine freie Lizenz für den Content verwendet werden. Verbreitet scheint die Befürchtung zu sein, dass die Beiträge zu Zwecken verwendet würden, mit denen der beitragende Künstler nicht einverstanden sein könnte. Für potentielle Mitentwickler am Code des Spiels selbst ist es dagegen wichtig, sich den Code ansehen zu können, bevor sie fest in die Entwicklung einsteigen. Wie Entwickler Andrew Craig im Interview sagt, haben Programmierer so die Möglichkeit zu sehen, dass eine Substanz besteht und nicht nur leere Worte.[85]

Wenn Künstler unter der gegenwertigen proprietären Lizenz Content zu Planeshift beitragen wollen, müssen sie damit alle Rechte an die Atomic Blue Corporation übertragen. Es ist ihnen damit nicht mehr möglich, später die Rechte dem Projekt wieder zu entziehen oder ihren Content einem anderen Projekt zur Verfügung zu stellen. Somit hat ein Künstler unter dieser Lizenz weniger Einflussmöglichkeiten darauf, was mit seinem Werk passiert, als er dies mit einer freien Lizenz hätte.

[83] Zu finden unter der URL: http://www.crystalspace3d.org/ (10.10.2005).
[84] Unter der URL: http://www.planeshift.it/pslicense.html (12.10.2005).
[85] Interview mit Andrew Craig (siehe Anhang A.2.1).

Weiterhin ist zu beachten, dass jeder Entwickler, der Code zu Planeshift beiträgt, diesen zwar zu den Bedingungen der GPL einbringt, aber gleichzeitig verpflichtet wird, alle Rechte an die Atomic Blue Corporation zu übertragen.

Aus dieser Lizenzvereinbarung wird für jeden Beitragenden deutlich, dass er damit sehr viele Rechte aufgibt. Die Gegenleistung, ein freies MMORPG zu erhalten ist zwar zur Zeit gegeben, kann aber nicht garantiert werden. Weil Atomic Blue alle Rechte an den beigetragenen Teilen erhält[86], haben sie die Möglichkeit, zu einem späteren Zeitpunkt die Lizenz zu ändern und das Spiel in ein Closed-Source-Produkt umzuwandeln.

Atomic Blue geht mit dieser Lizenzpolitik das Risiko ein, soziales Kapital bei den Open-Source-Entwicklern zu verschenken, so wie es Osterloh et al. (2002) für den Fall Netscape dargestellt haben. Während Netscape allerdings durch eine restriktive Lizenz[87] von vorneherein Misstrauen bei den Entwicklern verursachte, ist die GPL, die für Planeshift verwendet wird, an sich unproblematisch. Die Übertragung des Copyrights von Entwicklerbeiträgen an Atomic Blue ist zwar durchaus nicht unüblich, ähnlich verfährt auch die FSF selbst, doch kann durch die proprietäre Lizenz für den Content leicht die Befürchtung entstehen, dass der Code für Planeshift in der Zukunft ebenfalls einmal proprietär werden könnte. Das kann potentielle Mitentwickler abschrecken, so dass deren Vetrauen durch andere Maßnahmen gewonnen werden muss.

Nach Auffassung der Kernentwickler halten sich positive und negative Folgen der Lizenzentscheidung bisher zumindest die Waage.[88]Je länger die Entwicklung unter dem aktuellen Modell fortgeführt wird, desto mehr Vertrauen in den zukünftigen Bestand des Modells dürfte bei potentiellen Beitragenden entstehen.

Der Versuch, ein komplett freies Gegenstück zu Planeshift zu entwickeln, ist bisher gescheitert. Spieler und Beitragende scheinen weniger Wert auf ein komplett Freies Spiel zu legen, als darauf, dass die Qualität des vorhandenen Spiels maximiert wird. Sollte sich die Atomic Blue Corporation jedoch dazu entschließen, Planeshift in ein proprietäres Spiel umzuwandeln, könnte sich diese Einstellung allerdings schnell ändern.

Die Frage, die bestehen bleibt, ist die letztliche Finanzierung des Spiels. Während normale Software an sich keine Kosten verursacht, findet bei einem MMORPG weniger der Verkauf eines Produktes statt, auch wenn vielfach zum Spielen ein Produkt erworben werden muss, sondern es wird vielmehr ein Service angeboten. Der Service besteht zum einen darin, dass eine Reihe Server betrieben werden, auf denen das

[86]Fraglich ist, ob diese Übertragung der Rechte überhaupt in diesem Maße zulässig ist. Die Gesetze dazu unterscheiden sich zwischen den Ländern zum Teil erheblich.

[87]Diese Lizenz (NPL) erlaubte es Netscape, bestimmte Änderungen proprietär zu halten, räumte ihnen also Sonderrechte am Code ein.

[88]Interview mit Andrew Craig (siehe Anhang A.2.1).

Spiel läuft, zum anderen in der kontinuierlichen Weiterentwicklung des Spiels. Durch diese beiden Faktoren entstehen laufende Kosten für den Betreiber, selbst wenn dieser nicht gewinnorientiert arbeitet.

Üblicherweise findet die Finanzierung durch den ursprünglichen Verkauf des Spiels, eine monatliche Gebühr oder durch beides statt. Es gibt aber neue Ansätze der Finanzierung, die eventuell auch für Planeshift denkbar wären. So könnte spielinterne Währung gegen reales Geld verkauft werden, wie es das Finanzierungsmodell von Roma Victor[89] vorsieht und wie es auch andere Anbieter bereits in Erwägung ziehen oder schon durchführen. Eine andere Möglichkeit wäre Werbung, die in das Spiel selbst eingebaut wird, so wie es bei Anarchy-Online[90] bereits als Alternative zur monatlichen Gebühr möglich ist. Die Idee, für bestimmte Spielfunktionen per Micropayment zu bezahlen, findet laut Cornett (2004) keine große Zustimmung unter Spielern, denn wer bezahlt, möchte auch Zugriff auf das gesamte Spiel haben.

Das Finanzierungsmodell, dass von Atomic Blue bisher angestrebt wird, zielt auf freiwillige Spenden ab. Diese Spenden haben bisher für die Deckung der Kosten ausgereicht. Aufgrund der bereits mehr als 150.000 registrierten Spieler gehen die Entwickler von einem für die nächste Zeit ausreichenden Spendenaufkommen aus. Um eventuell steigende Ausgaben in der Zukunft zu finanzieren, hält Andrew Craig den Verkauf von Merchandising Artikeln mit einem Bezug zu Planeshift für denkbar. Eine Gewinnorientierung ist durch den Status von Atomic Blue als non profit Firma nicht möglich.[91]

4.5 Glest

Glest ist ein freies Echtzeit-Strategie-Spiel (Real Time Strategie – RTS). Dieses Spielegenre ist für den Bereich der Open-Source-Spiele aus einem ähnlichen Grund wie das der rundenbasierten Spiele interessant. Ein wichtiger Bereich in der Entwicklung von Spielen dieses Typs ist die Künstliche Intelligenz. Buro und Furtak (2003) stellen fest, dass die KI in Echtzeitstrategie Spielen noch sehr mangelhaft ist, obwohl dieses Genre bereits seit mehr als zehn Jahren existiert. Wie sie erklären liegt dieser Mangel zu einem großen Teil daran, dass aus Marketinggründen sehr viel Wert auf immer bessere Spielegrafiken gelegt wurde[92], wodurch Entwicklungszeit und Systemressourcen für die Entwicklung einer guten KI fehlten.

[89] Roma Victor befindet sich zur Zeit im Betatest. Homepage unter der URL: http://www.roma-victor.com (18.10.2005).
[90] Homepage von Anarchy-Online unter der URL: http://www.anarchy-online.com/ (18.10.2005).
[91] Interview mit Andrew Craig (siehe Anhang A.2.1).
[92] Verbesserte Spielegrafik lässt sich leicht an Bildschirmfotos in Spieletests erkennen, die Qualität der KI zeigt sich erst bei einem ausführlichen Test.

Weiterhin ist es gerade die Forderung nach Echtzeitfähigkeit der KI in diesem Genre, die zu besonderen Schwierigkeiten bei der Entwicklung im Vergleich zu rundenbasierten Spielen führt. Viele herkömmliche KI Techniken versagen in dieser Situation, weil sie auf optimalen Lösungen basieren, die die Zeitkomponente nicht in Betracht ziehen.

Ein weiterer Faktor ist die fehlende Vergleichsmöglichkeit verschiedener Künstlicher Intelligenzen durch die Geschlossenheit der Spiele. Im Gegensatz zum Computerschach, bei dem die Programme unterschiedlicher Firmen leicht gegeneinander antreten und somit verglichen werden können, bringt ein RTS Spiel exakt eine KI mit, die fest mit dem Spiel verzahnt ist und nicht ausgetauscht werden kann. Dadurch kann kein Wettbewerb zwischen verschiedenen Ansätzen und Techniken stattfinden, der zu einer Weiterentwicklung der KI führen würde. Genau an diesem Punkt wird es durch Open-Source-Software möglich, einen wichtigen Schritt zur Verbesserung zu vollziehen. Definierte Schnittstellen für die KI und Offenlegung der Quellen ermöglichen einen Wettbewerb verschiedener Verfahren und die jeweils bestmögliche Implementation.

Es ist aber zu beachten, dass bei Echtzeit-Strategie-Spielen die Bedeutung von Grafik und Sound weitaus größer ist als bei rundenbasierten Spielen. Dies drückt sich auch darin aus, dass die klassische isometrische Perspektive bei diesen Spielen zum größten Teil bereits aufgegeben wurde und nun 3D Grafiken eingesetzt werden. Die dadurch ermöglichte freie Wahl der Perspektive und des Zooms erfordert einen höheren Aufwand bei der Entwicklung der Grafiken. Bisherige Open-Source-RTS-Spiele haben gerade in diesen Bereichen noch große Mängel, wie Wen (2004) bei der Betrachtung der freien RTS Engine Stratagus feststellt. Glest dagegen kann besonders mit seiner guten Grafik überzeugen.

Glest ist zwar ein nicht kommerzielles Projekt, kommt aber von der Entwicklerseite her aus einer anderen Richtung als die meisten Open-Source-Programme. Es war ursprünglich ein reines Windows Spiel, das zwar kostenlos, aber nur mit geschlossenem Quellcode erhältlich war. Nach der Herausgabe einer 1.0 Version beschlossen die Entwickler ein Ende des Projektes. Anstatt es einfach verschwinden zu lassen, öffneten sie stattdessen den Quellcode, um anderen interessierten Entwicklern eine Weiterentwicklung zu ermöglichen, beziehungsweise um neue Entwickler in das Projekt hineinzubringen.[93]

Zunächst änderte dieser Schritt nicht viel, auch weil er kaum bekannt wurde. Nachdem allerdings die Informationen auf der Linux-Spiele-Seite „The Linux Game Tome“[94] veröffentlicht wurde, lag innerhalb weniger Tage eine Portierung des Spiels auf das Linux System vor. Dieser Port ist vollständig durch externe Entwickler durchgeführt worden.

[93] Interview mit Martiño Figueroa (siehe Anhang A.3.1).
[94] Zu finden unter der URL: http://www.happypenguin.org (16.02.2006).

Als Resultat aus der Freigabe des Codes und der Portierung nach Linux wurde auch die allgemeine Entwicklung des Spieles neu belebt. Es kamen nicht nur viele neue Spieler und auch neue Entwickler hinzu, auch die ursprünglichen Entwickler wurden neu motiviert, das Projekt fortzusetzen. Der Hauptentwickler von Glest beschreibt im Interview[95]die große Öffentlichkeitswirkung als die wichtigste Folge der Offenlegung des Quellcodes.

Seit diesem Schritt wurden mehrere neue Versionen herausgegeben, die eine Mischung aus interner Weiterentwicklung und externen Beiträgen enthalten. So wurde die Codebasis der bisherigen Windows Version mit der neuen Linux Version vereinigt, um eine einheitliche Weiterentwicklung zu ermöglichen. Dies führte unter anderem zu einem saubereren Code, der leichter auf weitere Plattformen portiert werden kann als der ursprüngliche. Somit konnte auch die wegen der anderen Prozessor Architektur schwierigere Portierung auf MacOS X gestartet werden. Auch wurde verstärkt die Entwicklung von neuem Content in Angriff genommen, der in Form einer Erweiterung herausgebracht werden soll. Selbst die Entwicklung eines bisher fehlenden und ursprünglich nicht geplanten Mehrspielermodus wird nicht mehr ausgeschlossen.

Die Entwickler des Spieles waren von vorneherein an einer möglichen Portierung auf andere Systeme interessiert. Nur durch eine entsprechende Wahl von verwendeten Bibliotheken war die Portierung nach Linux in diesem kurzen Zeitraum möglich. Und nur weil die verwendeten Bibliotheken auch selbst unter Open-Source-Lizenzen stehen, ergaben sich auch von der rechtlichen Seite keine Schwierigkeiten bei der Portierung.

Auch im Fall von Glest wurde es durch die Freigabe des Codes möglich, umfangreiche Modifikationen des Spiels zu erstellen und diese als neue, eigenständige Spiele zu veröffentlichen. Ein Beispiel dafür ist das in der Entwicklung befindliche „Gods Dawn“.[96]

4.6 The Nebula Device

The Nebula Device ist eine Spiele Engine, die seit 1998 als Open Source von der Berliner Firma Radon Labs[97] entwickelt wird. Auf dieser Engine basieren inzwischen mehr als zwanzig kommerzielle Spiele[98], darunter zehn Titel von Radon Labs selbst, die weiteren von anderen Firmen, die auf der ganzen Welt verteilt sind. Die Engine

[95]Interview mit Martiño Figueroa (siehe Anhang A.3.1).
[96]Homepage von „Gods Dawn“ unter der URL: http://www.3dtowns.at/ (01.11.2005).
[97]Homepage von Radon Labs: http://www.radonlabs.de (28.03.2006).
[98]Eine Liste der in Deutschland erschienen Nebula-Titel findet man unter der URL: http://radonlabs.de/nebulagames.html (28.03.2006)

selbst wird unter einer BSD ähnlichen Lizenz verbreitet,[99] zusätzlich bietet Radon Labs seit einiger Zeit Werkzeuge für Nebula an, die unter einer proprietären Lizenz stehen. Im Unterschied zu Engines, wie der von Quake, setzt Nebula auf einer niedrigeren Abstraktionsebene an. Dadurch findet keine Festlegung auf ein bestimmtes Genre von Spielen statt, andererseits ist für die Nutzung dieser Engine aber auch mehr Programmieraufwand nötig, als dies bei vollständigen Frameworks der Fall ist. Von Vorteil ist dies unter anderem für die Multi-Plattform Fähigkeit der Engine, wenn auch nur für die Windows Plattform ein 3D Renderer mitgeliefert wird.

Der Grund für die Entwicklung dieser Engine nach dem Open-Source-Modell liegt darin, dass keine zu diesem Zeitpunkt existierende Engine die Funktionen aufwies, die Radon Labs für ihr neuestes Projekt[100] benötigte. Die Firma versprach sich von der Freigabe des Quellcodes sechs wichtige Vorteile:[101]

Zunächst einmal war die Freigabe der Engine eine Marketingaktion, die die Aufmerksamkeit der Presse unter anderem auf die mit dieser Engine entwickelten Spiele richten sollte. Als zweites versprach man sich Vorteile von der Zusammenarbeit mit anderen Entwicklerstudios, die diese Engine ebenfalls verwenden wollten. Weiterhin konnte man durch die freie Lizenz ohne weitere Kosten von den Weiterentwicklungen durch externe Entwickler profitieren, wodurch sich die Fähigkeiten der eigenen Spiele verbessern ließen. Zusätzlich wurde die Engine auf diese Art über einen langen Zeitraum auf vielen unterschiedlichen Rechnerkonfigurationen getestet, so dass Fehler frühzeitig behoben werden konnten. Eine Veröffentlichung das Quellcodes zwang dazu, bei der internen Programmierung sauber vorzugehen und sinnvoll zu dokumentieren. Zuletzt konnten sich Nachwuchsentwickler bereits mit der Nebula Engine vertraut machen, bevor sie bei Radon Labs (oder anderen beteiligten Firmen) eingestellt wurden, was die Einarbeitungszeit verringerte.

Als ein Nebeneffekt wurde diese Engine, die eigentlich für 3D Computerspiele gedacht war, bald auch zu gänzlich anderen Zwecken eingesetzt. Zum Beispiel konnte die Nasa sie zur Datenvisualisierung verwenden, während das Oregon Research Institute einen Rollstuhlsimulator entwickelte. Für nicht kommerzielle Projekte ist eine Open-Source-Engine oft die einzige Möglichkeit, weil für proprietäre Engines im Allgemeinen Kosten für die Lizenzierung in Höhe von mehreren hunderttausend Dollar anfallen.[102]

[99] Es handelt sich hierbei um die TCL Lizenz. Genauer Text unter der URL: http://radonlabs.de/license.html (10.02.2006).

[100] Dies war „Project Nomads“, das 2002 von CDV publiziert wurde und auf der Spielemesse ECTS 2001 den Preis für das beste PC Spiel der Messe erhielt.

[101] Interview mit Bernd Beyreuther (siehe Anhang A.4.1).

[102] Als Beispiel sei hier die „Unreal 2“ Engine angeführt, deren Lizenzierungskosten sich auf 350.000 US Dollar plus eine Gewinnbeteiligung belaufen. Die genauen Bedingungen sind nachzulesen unter der URL: http://www.unrealtechnology.com/html/licensing/terms.shtml (24.03.2006).

Die Nebula Engine war ursprünglich ausschließlich für die Windows Plattform vorgesehen. Durch die Entwicklung nach dem Open-Source-Modell war es externen Programmierern möglich, Ports auf andere Plattformen, wie zum Beispiel Linux, vorzunehmen. Der für diesen Zweck der Portierung geschriebene Code kommt auch den anderen Plattformen zu Gute. Die ursprüngliche Version bot lediglich einen Direct3D Renderer; um eine Version für Linux zu ermöglichen, war die Entwicklung eines OpenGL Renderers nötig. Dieser ist schließlich wieder unter Windows nutzbar, wodurch dort zwei alternative Rendering Methoden zur Auswahl stehen. Das wiederum erleichtert das Finden und Beheben von Fehlern. Die üblicherweise beträchtlichen Kosten für die Entwicklung eines zweiten Renderers entfielen für Radon Labs, weil die Entwicklung durch externe Entwickler durchgeführt wurde, die Vorteile daraus sind für die Firma aber voll nutzbar.

Für die neue Version der Engine, Nebula 2, steht eine vergleichbare Entwicklung noch aus. Hier ist ebenso ursprünglich nur ein DirectX Renderer verfügbar gewesen. Aufgrund der zu der Zeit noch fehlenden Funktionalität in OpenGL blieb es bisher dabei. Inzwischen wäre es möglich, diesen zweiten Renderer zu entwickeln, eine Entwicklung, die Radon Labs wieder externen Programmierern überlassen wird. Dies liegt daran, dass sich der finanzielle Aufwand einer internen Entwicklung nicht durch die relativ geringe Zahl an möglichen zusätzlichen Kunden ausgleichen lässt. Eine Anpassung der restlichen Engine an andere Betriebssysteme als Windows wurde bereits durch externe Nutzer durchgeführt.

Radon Labs konnte also durch das Entwickeln der Engine als Open-Source-Software die von Gabriel und Goldman (2003) als „Innovation Happens Elsewhere“ bezeichnete Strategie nutzen. Dies ermöglicht nicht nur die Innovationen externer Entwickler mitzunutzen und somit schneller bessere Ergebnisse zu erzielen als alleine, es schafft auch eine positive Grundlage für die Kommunikation mit anderen Firmen und den Nutzern. Das äußerte sich, wie Bernd Beyreuther im Interview[103] erklärt, ganz konkret darin, dass ein anderes deutsches Entwicklerstudio einen Auftrag für ein Spiel an Radon Labs abgab, da es diesen selber nicht mehr bewältigen konnte. Möglich wurde dies nur, weil das andere Studio ebenfalls die Nebula Engine verwendete.

Weil Radon Labs einen eigenen Codebaum der Nebula Engine verwendet, der nicht direkt für externe Entwickler zugänglich ist, entstehen Differenzen zwischen der Offenen und der internen Variante. Diese Forks werden in unregelmäßigen Abständen wieder zusammengeführt.[104] Hierdurch werden gewisse potentielle Vorteile der Open-Source-Entwicklung der Engine wieder aufgegeben. Zum Beispiel könnten Doppelentwicklungen vorkommen. Der Effekt ist für die offene Variante stärker als für die interne, weil Radon Labs ja Zugriff auf den Quellcode der offenen Version hat und somit Entwicklungen von dort leichter portieren kann. Praktisch findet dies aber kaum

[103] Interview mit Bernd Beyreuther (siehe Anhang A.4.1).
[104] Interview mit Bernd Beyreuther (siehe Anhang A.4.1).

statt, weil auch diese Vorgehensweise einen hohen Aufwand bedeutet.[105] Den Nutzern der offenen Variante bleibt nur die Möglichkeit, auf die nächste Zusammenführung der Codebäume zu warten, wollen sie nicht eine existierende Funktion ein zweites Mal entwickeln.

Dieses ist auch das schwerwiegendste Problem in der Weiterentwicklung von Nebula. Wie der externe Entwickler Megan Fox im Interview[106] erklärt, führen diese Management-Probleme dazu, dass nützliche Funktionen zum Teil nicht übernommen werden. Dadurch werden externe Nutzer demotiviert, etwas zur Fortentwicklung der Engine beizutragen. Als Resultat ergibt sich eine gewisse Stagnation – etwas, das eigentlich durch die Verwendung eines Open-Source-Modells verhindert werden sollte.

Bei solch einem Projekt wie einer Spiele-Engine, ist zu erwarten, dass die Zielgruppe zum allergrößten Teil selbst aus Entwicklern besteht. Unter diesen Bedingungen sollten sich die Vorteile des Basarmodells verstärken, weil fast jeder Nutzer die nötigen Fähigkeiten hat, um an der Entwicklung mitzuwirken. Mitteilungen von Nebula Kernentwicklern auf der offiziellen Mailingliste deuten darauf hin, dass tatsächlich der größte Bedarf an externen Beiträgen nicht in der direkten Codeentwicklung besteht, sondern vielmehr in Arbeiten wie der besseren Dokumentation und dem Administrieren von Webseite und Forum. Besonders problematisch ist der Mangel an offizieller Dokumentation und guten Tutorials, wie M. A. Garcias im Interview[107] erklärt. Dies liegt unter anderem in der geringen Kommunikation zwischen internen und externen Entwicklern begründet. Dieses Problem wurde auch von Radon Labs erkannt, die Firma bemüht sich seitdem um eine Verbesserung in jenem Bereich.[108]

Ein großer Teil der externen Entwicklung ist nicht direkt im Kern der Engine angesiedelt. Stattdessen gibt es diverse „Contrib“ Projekte, die Funktionen ergänzen, ohne in den eigentlichen Code-Baum eingebunden zu sein. Die Nützlichkeit dieser nur zum Teil eingebundenen Ergänzungen ist sehr beschränkt. Sie entspringen oft der Entwicklung eines einzelnen externen Projekts und sind deshalb zu spezialisiert, um von anderen Nutzern auch angewendet zu werden.[109] Dazu kommt das Problem, dass ihre Qualität oft nicht dem Standard der Nebula Engine selbst entspricht. Weiterhin werden diese Teile oftmals nicht aktiv gepflegt, so dass ihre Funktionalität nach größeren Änderungen im Kernprojekt oft nicht mehr gegeben ist.[110] Als Resultat daraus wird ein großer Teil der externen Entwicklungen nicht in ernsthaften Projekten eingesetzt. Stattdessen werden die entsprechenden Funktionen von jedem Projekt erneut implementiert.

Eine weitere Schwierigkeit der Entwicklung liegt darin, dass andere Firmen neben

[105] Interview mit Bernd Beyreuther (siehe Anhang A.4.1).
[106] Interview mit Megan Fox (siehe Anhang A.4.2).
[107] Interview mit M. A. Garcias (siehe Anhang A.4.3).
[108] Interview mit Bernd Beyreuther (siehe Anhang A.4.1).
[109] Interview mit M. A. Garcias (siehe Anhang A.4.3).
[110] Interview mit Megan Fox (siehe Anhang A.4.2).

Radon Labs, die die Nebula Engine ebenfalls verwenden, für ihre Projekte eine stabile Version benötigen. Das führt dazu, dass sie sich zu einem bestimmten Zeitpunkt vom Hauptbaum abspalten und intern eine eigene Version pflegen, die als Grundlage für das eigene Spiel dient.[111] Zumeist wird zwar versucht, Änderungen aus diesen Abspaltungen wieder auf den Hauptbaum zurück zu portieren, doch gelingt dies nicht immer problemlos. Dadurch gehen nützliche externe Entwicklungen teilweise für die Open-Source-Version verloren. Ebenso sind diese Erweiterungen meist für Radon Labs selbst nicht nutzbar, weil der Integrationsaufwand zu hoch wäre.[112]

Um zusätzliche Einnahmen mit dem Nebula Device zu erwirtschaften, versuchen Radon Labs Werkzeuge zu verkaufen, die die Nutzung der Engine erleichtern sollen. Sie entstammen ursprünglich aus Entwicklungen für den internen Bedarf. Ob dieses Modell erfolgreich ist, lässt sich noch nicht abschließend feststellen. Die befragten externen Entwickler ziehen selbst erstellte Tools der „offiziellen“ Variante vor.[113] Andererseits berichtet Bernd Beyreuther davon, dass Radon Labs durch den Verkauf der Werkzeuge bereits Gewinne erwirtschaftet.[114]

[111] Interview mit M. A. Garcias (siehe Anhang A.4.3).
[112] Interview mit Bernd Beyreuther (siehe Anhang A.4.1).
[113] Interview mit M. A. Garcias (siehe Anhang A.4.3).
[114] Interview mit Bernd Beyreuther (siehe Anhang A.4.1).

5 Analyse

Dieses Kapitel dient zur Untersuchung der Informationen, die durch die Betrachtung der Beispiele erhalten wurden. Zunächst werden die hierbei gefundenen Gemeinsamkeiten und Unterschiede dargestellt. Anschließend wird ausführlich das Gebiet der Online-Spiele behandelt, das sich als besonders wichtig für die Anwendungsmöglichkeiten von Open Source auf Computerspiele gezeigt hat. Schließlich wird auf die besondere Problematik der Middleware im Spielebereich eingegangen.

5.1 Gemeinsamkeiten und Unterschiede der Beispiele

Bei der Betrachtung der Beispiele ergeben sich einige Gemeinsamkeiten, die unabhängig von der Art des Spiels bestehen. Andererseits gibt es auch Unterschiede, die zum größten Teil durch die Verschiedenheit der Genres bestimmt sind. Beides wird im Folgenden zusammenfassend dargestellt.

Unabhängig vom Genre des Spiels, wird es für viele Spieler immer wichtiger, dass sie eigene Modifikationen am Spiel vornehmen können. Die Art, wie diese Modifikationen aussehen, sind von der Art des Spieles abhängig. Bei Rollen- und Strategiespielen, aber teilweise auch in anderen Genres, ist die Möglichkeit, das Benutzungsinterface des Spiels den eigenen Wünschen anzupassen, sehr von Spielern gewünscht. In allen Genres bis auf MMORPG ist aber die häufigste von Nutzern vorgenommene Veränderung die Erstellung neuer Karten oder Level. Bei den MMORPG sind diese serverabhängig und damit prinzipbedingt nicht durch Spieler veränderbar. Hier erwarten Spieler aber regelmäßige Neuerungen von der betreibenden Firma. Weiterhin ist es besonders bei Mehrspieler-Spielen wichtig, dass die Nutzer das Aussehen ihrer Avatare anpassen können.

Neue Karten verlängern die Zeit, die ein Spiel attraktiv bleibt, weil hierdurch wieder etwas Unbekanntes erforscht werden kann. Spieler erhalten die Möglichkeit, ihre Kreativität umzusetzen. Für Spielefirmen ergibt sich nicht nur ein längerer Zeitraum, in dem das Spiel verkauft werden kann, sondern auch die Chance, fähige Leveldesigner unter den Spielern zu erkennen und als Mitarbeiter für die eigene Firma zu gewinnen. Firmen, die Spieleengines an andere Entwickler lizenzieren, können hochwertige Mods verwenden, um die Leistungsfähigkeit ihrer Engine zu demonstrieren und so weitere Lizenznehmer zu gewinnen.

Wenn ein Spiel so geschrieben ist, dass Modifikationen leicht eingebracht werden können, so ist es nicht unbedingt erforderlich, dass auch der Quellcode offen ist, um die Fähigkeiten der Spieler zu nutzen. Lediglich ein Karteneditor reicht dafür aber meistens nicht aus, weil der Wunsch der Nutzer nach weitergehenden Änderungen besteht. Das Beispiel Neverwinter Nights zeigt, wie weit eine Veränderbarkeit eines Spiels gehen kann, ohne dass der eigentliche Quellcode offen ist. Allerdings liegt hier nicht nur eine sehr umfangreichen API vor, sondern es sind zusätzlich viele Interna, wie zum Beispiel die KI, komplett im Quellcode verfügbar und durch den Spieler veränderbar. Hierfür ist es wichtig, dass die entsprechenden Teile des Spiels in Form von Skriptsprachen implementiert werden, weil die Spieler das Spiel nicht selbst neu kompilieren können. Die Einbindung einer Skriptsprache in die Spieleengine ermöglicht sehr umfangreiche Änderungen durch die Nutzer. Es scheint sogar so zu sein, dass dieses für die meisten Modder wichtiger ist als der Zugriff auf den eigentlichen Quellcode. Das Beispiel Freeciv zeigt, dass für die meisten Anwender eine Einarbeitung in den Quellcode eines Spiels zu aufwendig ist, auch wenn dadurch noch deutlich umfangreichere Modifikationen möglich würden.

Allerdings lässt sich an den Beispielen sehen, dass durch eine Offenlegung des Quellcodes zumindest die Lebenszeit der Engine deutlich verlängert werden kann. Die Nutzung der Engine durch verschiedene Spiele bringt normalerweise Verbesserungen mit sich, von denen wiederum alle diese Engine nutzende Spiele profitieren können. Teilweise ermöglicht ein offener Quellcode auch überraschende neue Verwendungsmöglichkeiten, wie die Beispiele der Nebula und Quake Engines zeigen.

Es ist anscheinend so, dass der Content und der Code eines Spiels unterschiedlich von einer Öffnung profitieren. Besonders wenn Musik und Soundeffekte betrachtet werden, scheint es große Schwierigkeiten bei der Erstellung von freiem Material zu geben. In allen Genres zeigt es sich, dass dieser Bereich entweder nur lückenhaft abgedeckt wird oder das vorhandene Material nicht von einer vergleichbaren Qualität mit kommerziell produziertem ist. Während Open-Source-Projekte sich im Allgemeinen von rechtlich zweifelhaftem Content fern halten, nutzen Modifikationen von proprietären Spielen teilweise solchen. Dadurch erfolgende Urheberrechts-Verletzungen werden durch die Rechteinhaber oft über längere Zeit ignoriert. Das führt dazu, dass viele Mods in einer rechtlichen Grauzone existieren.

Am meisten scheinen Dinge wie der eigentliche Code, Karten und KI Skripte von einer Öffnung und der damit einhergehenden inkrementellen Verbesserung zu profitieren. Grafik und Sound sind wichtiger für die Identität eines Spiels, hier ist anscheinend eine Produktion aus einer Hand gewünscht. Gleiches gilt auch für andere Teile des Contents, die für das jeweilige Spiel charakterisierend sind.

Der Anbieter des bekannten Rollenspiels Dungeons&Dragons, Wizards of the Coast,[115] hat speziell wegen dieser besonderen Problematik von Spielen eine eigene Lizenz ge-

[115] Homepage von Wizards of the Coast: http://www.wizards.com/ (27.03.2006).

schaffen, die Open Gaming License (OGL).[116] Diese erlaubt eine Trennung von Open Game Content und Product Identity und somit eine Koexistenz von Offenen und Geschlossenen Teilen im selben Produkt. Die OGL hat erklärtermaßen die Open Source Prinzipien als Vorbild und das Ziel, diese Prinzipien auf die Ansprüche von Spielen umzusetzen.

Andererseits gibt es gewisse Schwierigkeiten bei der Umsetzung eines OGL basierten Spiels als Software. Die OGL fordert nämlich, dass aller Content, der unter dieser Lizenz steht, in einer menschenlesbaren Form zur Verfügung steht. Sourcecode zählt aber ausdrücklich nicht zu dieser Form. Damit stellt die OGL hohe Ansprüche an die Entwicklung eines Computerspiels, denn um die Lizenz einzuhalten, müssten alle Regeln des Spiels im Klartext in Textdateien gespeichert und beim Start des Spiels geparst werden.

5.2 Online-Spiele

Im Verlaufe der Arbeit hat sich ergeben, dass die Gruppe der Online-Spiele eine Sonderrolle einnimmt. Aus diesem Grund werden Spiele dieser Art im Folgenden genauer betrachtet.

Der Trend zu den Online-Spielen hat im Genre der Rollenspiele begonnen. Dieses Spielegenre hat als Grundprinzip die Kommunikation, etwas was auch moderne KI nicht leisten kann. Aus diesem Grund war ein starker Bedarf nach vernetztem Spielen vorhanden. Andere Genres folgen diesem Trend unterschiedlich stark. So wird in modernen First-Person-Shootern und Echtzeit-Strategie-Spielen im Allgemeinen ein Mehrspielermodus eingebaut, den Schritt zu einem echten Online-Spiel haben sie aber noch nicht getan.

Ein fundamentaler Unterschied zwischen Offline- und Online-Spielen besteht darin, dass Online-Spiele persistent sind. Dies bedeutet, dass die Spielwelt auch dann bestehen bleibt, wenn der Spieler die Software geschlossen hat. Veränderungen, die ein Spieler während des aktiven Spielens durchgeführt hat, bleiben erhalten und werden von ihm beim nächsten Anmelden genauso wieder vorgefunden. Es kann sich dabei um Erfahrungspunkte, Gold, Gegenstände, aber auch dauerhaft sichtbare Bestandteile der Welt, wie zum Beispiel ein durch den Spieler gebautes virtuelles Haus handeln.

Cypra (2005) sieht dann auch die Tatsache, dass im Online-Spiel eine Parallelwelt existiert, die ohne Unterbrechung lebt und sich fortentwickelt, auch wenn ein Spieler nicht anwesend ist, als das entscheidende Kriterium, das es von allen anderen Spielen unterscheidet.

[116] Der Text der OGL ist erhältlich unter der URL: http://www.wizards.com/default.asp?x=d20/article/20040121a (27.03.2006).

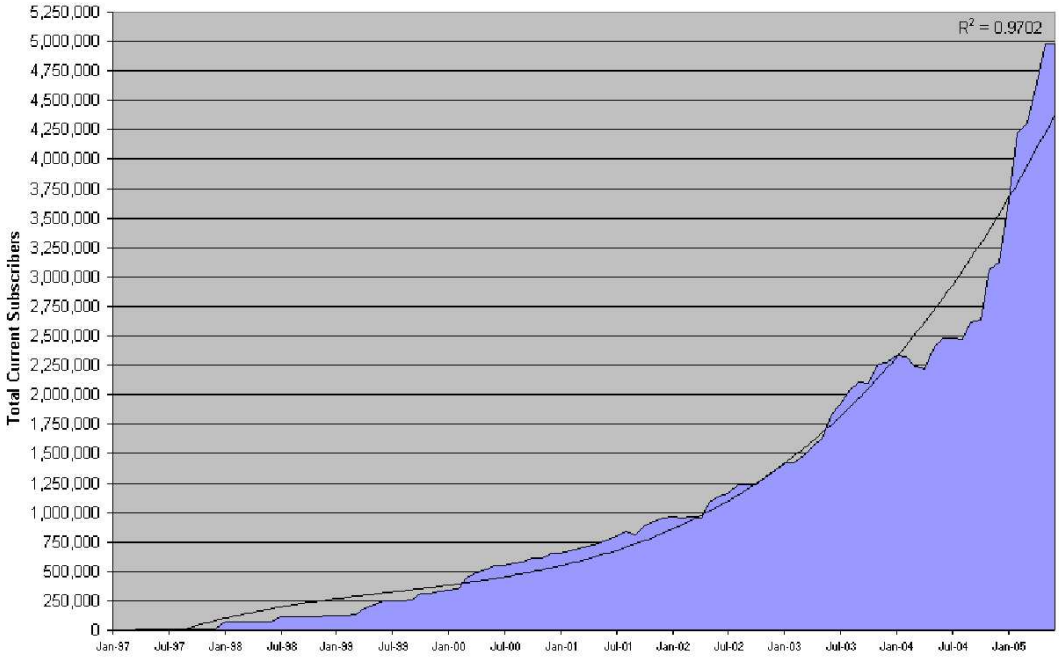

Abbildung 5.1: Entwicklung der Abonnentenzahlen bei MMOG (ohne Lineage, Lineage II und Ragnarok Online), Quelle: Woodcock (2005)

Es ist festzustellen, dass der Anteil an Spielen mit der Eigenschaft, das Zusammenspiel mehrerer Spieler über das Internet zu erlauben, stetig zunimmt. Dies wird auch an den untersuchten Beispielen deutlich. Lediglich Glest verfügt über keinen Mehrspielermodus und auch bei diesem Spiel ist ein solcher geplant. Die Nebula und Quake Engine sind beide auf eine Möglichkeit der Nutzung für Mehrspieler-Spiele ausgelegt. Bei Freeciv und Neverwinter Nights stehen Einzel- und Mehrspieler-Modi gleichwertig nebeneinander, bei beiden Spielen bietet die Mehrspieler-Variante zusätzliche Funktionen. Planeshift schließlich ist ein reines Online-Spiel, das keine Möglichkeit bietet, es alleine zu spielen.

Abbildung 5.1 zeigt, wie stark die Nutzerzahlen der reinen Online-Spiele in den letzten Jahren zugenommen haben. Es ist zu erwarten, dass sich dieser Trend in den nächsten Jahren noch verstärken wird. Die Gründe liegen zum einen darin, dass zunehmend größere Spielefirmen mit einem entsprechenden Etat auch für Werbung in den Markt eintreten. Vor allem aber ist mit der weiteren Verbreitung von Breitbandzugängen ein starker Zuwachs an Online-Spielern zu erwarten. Dies lässt sich vor allem aus Erfahrungen im asiatischen Raum ableiten, der eine deutliche höhere Dichte an Breitbandzugängen aufweist als Europa und Nordamerika. Weiterhin wird bislang nur ein sehr kleiner Teil der Spielegenres durch Online-Spiele erfasst. Dies wird beim Vergleich der Abbildungen 5.2 und 5.3 deutlich.

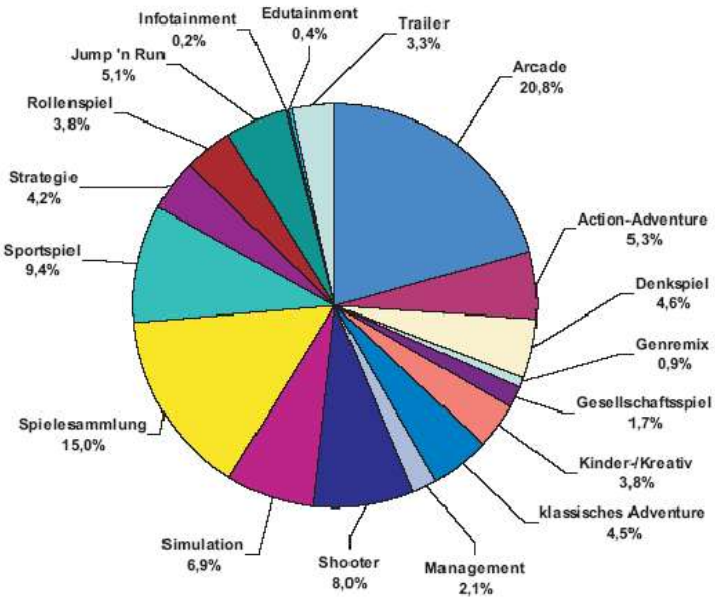

Abbildung 5.2: Genreverteilung bei Offline-Spielen, Quelle: USK, zitiert nach Behrmann et al. (2005)

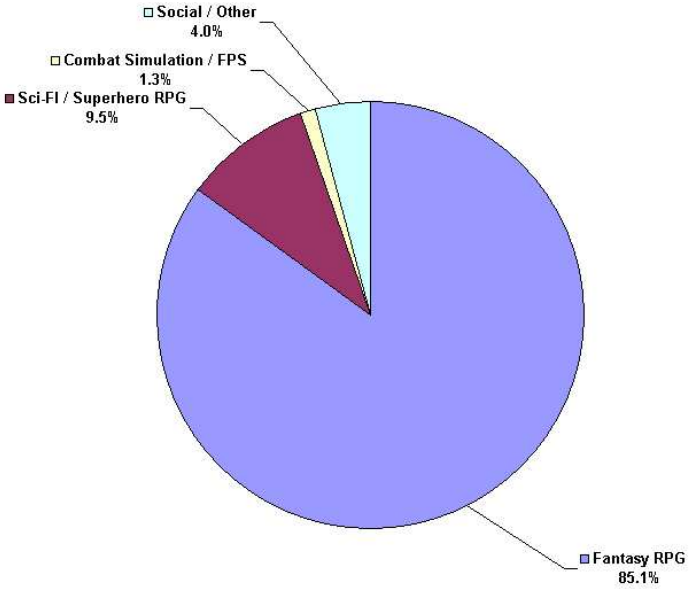

Abbildung 5.3: Genre-Verteilung bei Online-Spielen, Quelle: Woodcock (2005)

Sehr gut sichtbar wird aus dieser Gegenüberstellung, dass im Online Bereich die Rollenspiele, besonders aus dem Bereich der Fantasy Spiele, dominierend sind. Während im allgemeinen Spielemarkt, der durch Offline-Spiele dominiert wird, der Bereich der Rollenspiele nur einen Anteil von 3,8% erreicht, liegt dieser für Online-Spiele bei 94,6%. Demzufolge ist zu erwarten, dass eine Umsetzung von anderen Spielprinzipien auf den Online Bereich den Markt für Online-Spiele weiter deutlich vergrößern wird, auch wenn sich vermutlich nicht alle Genre als Online-Varianten realisieren lassen.

Warum von einer Sonderrolle der Online-Spiele gesprochen werden kann wird deutlich, wenn man die Geschäftsmodelle der verbreiteten MMORPG betrachtet, wie sie sich aus den in Tabelle 5.1 aufgeführten Preisen ergeben. Wie daraus ersichtlich wird, erfordern Online-Spiele, anders als konventionelle Computerspiele, für ihre Nutzung zumeist die Entrichtung einer monatlichen Gebühr, die zusätzlich zum einmaligen Kaufpreis des Spieles anfällt.

Spiel	Kaufpreis	monatliche Kosten	jährliche Kosten
Anarchy Online	34,44 Euro	17,19 Euro	109,71 Euro
City of Heroes	19,99 Euro	12,99 Euro	129,98 Euro
Everquest 2	16,66 Euro	12,49 Euro	119,99 Euro
Star Wars Galaxies	24,99 Euro	12,49 Euro	119,99 Euro
World of Warcraft	39,95 Euro	12,99 Euro	131,88 Euro
Die 4. Offenbarung	0,00 Euro	8,99 Euro	90,00 Euro
EVE Online	0,00 Euro	14,95 Euro	131,40 Euro
Puzzle Piraten	0,00 Euro	8,88 Euro	66,60 Euro
Runescape 2	0,00 Euro	4,17 Euro	50,00 Euro
Second Life	0,00 Euro	8,29 Euro	60,00 Euro
Tibia	0,00 Euro	6,65 Euro	59,95 Euro
Guild Wars	39,99 Euro	0,00 Euro	0,00 Euro
KAL Online	0,00 Euro	0,00 Euro	0,00 Euro

Tabelle 5.1: Einmalige und regelmäßige Kosten großer MMORPG, Quelle: Eigene Zusammenstellung des Autors. [117]

Aus dieser Aufstellung wird deutlich, dass es vier verschiedene Modelle der Finanzierung von MMORPG gibt. Die klassische Variante sieht einen Kaufpreis für das Spiel vor, wie es bei einem Offline-Spiel auch üblich ist. Zusätzlich wird eine monatliche Abogebühr verlangt, die sowohl für die laufenden Kosten des Anbieters, als auch

[117] Preise sind den offiziellen Homepages der Spiele entnommen und zu einem Dollarkurs von 1:1,2 in Euro umgerechnet.

für Erweiterungen des Contents bestimmt ist.[118]

Daneben gibt es inzwischen eine größere Zahl von MMORPG, bei denen kein Kaufpreis, sondern ausschließlich die monatliche Gebühr verlangt wird. Diese Spiele stammen hauptsächlich von kleineren Anbietern, die einen deutlich geringeren Werbeaufwand betreiben und auch keinen Zugriff auf die üblichen Retailmärkte haben. Die Clientsoftware für diese Titel wird normalerweise zum Download über die Webseite des Anbieters verteilt.

Das dritte Modell ist das von Guild Wars. Hier sollen die laufenden Kosten lediglich durch den Verkauf von späteren Erweiterungen gedeckt werden. Allerdings müssen Spieler diese Erweiterungen nicht kaufen, wenn sie nur das ursprüngliche Spiel weiter nutzen möchten. Ob dieses Modell langfristig tragfähig ist, lässt sich noch nicht beurteilen.

Ein viertes Modell wird von KAL Online, aber auch von einem Großteil der kleineren asiatischen MMORPG betrieben. Hierbei wird weder für das Spiel Geld verlangt, noch fallen monatliche Gebühren an. Stattdessen werden virtuelle Gegenstände, die im Spiel selbst nützlich sind, gegen reales Geld verkauft. Auch größere Anbieter, wie zum Beispiel Sony, experimentieren mit diesem Modell.[119] Bei Second Life ist der Verkauf virtueller Gegenstände eine zusätzliche Einnahmequelle zu den monatlichen Gebühren.

Eine Variante des vierten Modells besteht darin, das Spiel anstatt über den Verkauf virtueller Waren durch den Verkauf von realen Gütern zu finanzieren. Das Spiel dient hierbei als Werbeplattform, es wäre aber auch denkbar, die eigentliche Verkaufsplattform ebenfalls in die Spielumgebung zu integrieren. „Anarchy Online“ bietet neben der aufgeführten Finanzierung nach dem ersten Modell auch die Möglichkeit, unter diesem abgewandelten vierten Modell zu spielen. Dabei fallen für die Spieler keine Kosten an, dafür wird Werbung in die Spielumgebung eingeblendet.

Es werden also beim zweiten und vierten Modell keine Produkte mehr verkauft, beim ersten Modell ist der Produktverkauf nur eine Nebensache. Spielehersteller, die eines dieser Modelle für ihr MMORPG verwenden, sind damit als Dienstleister einzuordnen. Um möglichst viele Kunden für diese Dienstleistung zu gewinnen, ist es für sie nötig, die Zugangssoftware so weit wie möglich zu verbreiten.

Diese größtmögliche Verbreitung ist durch Open-Source-Software möglicherweise einfacher zu erreichen. Wenn die Quellen des Clients offen liegen, kann die Portierung des Spiels auf andere Plattformen von interessierten Nutzern selbst durchgeführt werden. Dies ist für den Anbieter der Spielwelt eine kostenlose Vergrößerung des Marktes.

[118] Umfangreiche Erweiterungen werden aber häufig als gesonderte Produkte verkauft.

[119] Sony hat die Veröffentlichung eines auf diese Art finanzierten Titels angekündigt. URL: http://www.gamasutra.com/php-bin/news_index.php?sssdmh=dm4.157076&story=7002 (04.11.2005).

Anbieter, die das erste Modell verwenden, also zusätzlich zu den monatlichen Gebühren auch eine einmalige Zahlung für die Clientsoftware verlangen, könnten eventuell auf das zweite Modell wechseln und somit ebenfalls für eine Öffnung der Software in Frage kommen. Dafür ist zu kalkulieren, ob die mögliche Vergrößerung des Marktes und die damit einhergehenden höheren Einnahmen durch Abonnement-Gebühren die Mindereinnahmen ausgleichen, die durch den Wegfall des Softwareverkaufs entstehen.

Zusammenfassend scheint der Bereich der Online-Spiele ideal zu den Open-Source-Prinzipien zu passen.

5.3 Middleware

Im Verlauf der Arbeit hat sich als neues Problem die immer häufiger in Spielen verwendete Middleware gezeigt. Weil zunehmende Bestandteile moderner Spiele nicht mehr vom Entwickler selbst erstellt, sondern dafür von anderen Firmen lizenzierte Codebibliotheken verwendet werden, hat das Entwicklungsstudio nicht mehr die Freiheit, den Quellcode des Spiels unter einer offenen Lizenz herauszugeben. Im Extremfall betrifft dies die gesamte Engine des Spiels, wenn selbige komplett von einem externen Anbieter lizenziert wurde.

Dieses Problem wird um so deutlicher, wenn man berücksichtigt, dass häufig die Kosten für solche Middleware-Lizenzen abhängig von der Stückzahl des jeweiligen Spiels sind. Dadurch werden Finanzierungsmodelle, die von der Grundannahme, dass ein Spiel ein Produkt ist, abweichen, erschwert oder sogar unmöglich gemacht. Sie verhindern auch, dass ein Spiel nach Beendigung seines Lebenszyklusses freigegeben werden kann. Zumeist wird sogar eine nur kostenlose Verteilung unmöglich.

Open-Source-Middleware könnte die Lösung dieses Problems darstellen. Dadurch behält der Spieleentwickler die Kontrolle über seine Software, kann also selber über eine Weitergabe des Quellcodes und sein Finnazierungsmodell entscheiden. Einzelne Bereiche der üblicherweise verwendeten Middleware werden zwar bereits durch entsprechende Open-Source-Bibliotheken abgedeckt,[120] doch gilt dies längst nicht für alle. Hinzu kommt, dass auch die vorhandenen Bibliotheken zum Teil nicht alle Anforderungen der Spieleentwickler abdecken.

Dieses Problem kann jedoch zu einer neuen Möglichkeit für Open-Source-Middleware werden. Ähnlich wie die Nebula Engine unter einer Open-Source-Lizenz vertrieben wird, während zusätzliche Werkzeuge und Portierungen auf andere Plattformen – zum Beispiel Spielekonsolen – kostenpflichtig angeboten werden, scheint es durchus denkbar, dass sich Firmen auf die Erweiterung und Pflege von Open-Source-Middleware

[120] Als Beispiel ist hier die Physik-Engine ODE zu nennen, URL: http://www.ode.org/ (29.03.2006).

spezialisieren. Dadurch könnte ein Modell entstehen, das eine Mischung aus einem Service- und einem Auftragsmodell darstellt, wobei sich der Anbieter nicht an Endkunden, sondern an andere Softwareentwickler richtet.

6 Diskussion

In diesem Kapitel wird gezeigt, wie sich die zuvor vorgestellten Open-Source-Prinzipien mit dem Bereich der Computerspiele zusammenführen lassen. Dabei liegt der Schwerpunkt darauf, welche Lizenzen in diesem Gebiet sinnvoll nutzbar sind und welche Geschäftsmodelle wie angewendet werden können. Dafür werden die Ergebnisse aus dem fünften Kapitel herangezogen und in einen Zusammenhang mit den Kapiteln zwei und drei gebracht. Anschließend wird unter Berücksichtigung der erarbeiteten Ergebnisse betrachtet, inwieweit Computerspiele und Open Source insgesamt zusammen passen. Zuletzt wird noch ein Ausblick auf weitere Forschungsfelder im behandelten Themenkomplex gegeben.

6.1 Lizenzmodelle für Computerspiele

Hawkins (2004) stellt vier grundsätzliche Möglichkeiten für Unternehmen dar, wie sie ihre Software lizenzieren können. Eine proprietäre Lizenz bedeutet einen geschlossenen Quellcode und damit die in der kommerziellen Spieleentwicklung vorherrschende Situation. Bei der Verwendung einer Lizenz mit einer „Giftpille“ ist zwar der Quellcode offen, der Hersteller behält sich aber Sonderrechte vor.[121] Dies ermöglicht das Einbinden von Nutzern, schließt im Ergebnis aber Firmen mit kommerziellem Interesse an der Software aus. Die Variante einer BSD ähnlichen Lizenz bedeutet für den Entwickler, dass eine Konkurrenzfirma zwar vom Quellcode profitiert, aber ihrerseits nichts zurückgeben muss. Mit einer viralen Lizenz profitieren alle Entwickler gleichermaßen, ohne dass einer der Beteiligten seinen Anteil am Code zurückhalten kann.

Im Rahmen der kommerziellen Spieleentwicklung scheint zunächst neben einer proprietären Lizenz nur die virale Variante erfolgsversprechend, weil die anderen beiden Möglichkeiten entweder zum eigenen Schaden führen könnten oder eine kommerzielle Mitentwicklung ausschließen. Allerdings macht das Beispiel der Nebula Engine von Radon Labs deutlich, dass auch eine BSD ähnliche Lizenz aus einem kommerziellen

[121] Eine übliche Form dieser Lizenz beinhaltet, dass der ursprüngliche Entwickler den gesamten Code, inklusive externer Beiträge, auch unter einer proprietären Lizenz anbieten darf, während alle anderen Entwickler lediglich eine Open Source Lizenz verwenden dürfen. Eine bekannte Lizenz mit Giftpille war die NPL von Netscape.

Standpunkt heraus nützlich sein kann. Dies liegt daran, dass sich mit einer BSD ähnlichen Lizenz Standards schaffen lassen. Der Erfinder eines Standards profitiert aber von dessen Durchsetzung.

Neben der Lizenzwahl für den Code des Spiels, gibt es auch die Frage, welche Lizenzen für den Content sinnvoll sind. Die klassischen Softwarelizenzen sind für Content nicht angemessen und wie anhand der Beispiele festgestellt wurde, haben Künstler häufig Probleme mit diesen Lizenzen. Als Alternative gibt es in diesem Bereich neben den proprietären Lizenzen die Möglichkeit, auf eine Creative-Commons-Lizenz auszuweichen. Weiterhin ist die gerade für den Bereich des Spielecontents entwickelte Open-Gaming-Lizenz zu betrachten.

Vom Standpunkt der Offenheit her, wäre eine proprietäre Lizenz für den Content die schlechtest mögliche Variante, auch wenn der Code des Spieles unter einer freien Lizenz steht. Ein Spiel mit offenem Code und proprietärem Content ist einem komplett proprietären Spiel trotzdem vorzuziehen, denn die wichtigsten Vorteile von Open Source sind durch einen offenen Quellcode bereits erreicht.

Die Open-Gaming-Lizenz ermöglicht dem Entwickler eines Spiels, Derivate seines Contents zu erlauben, dabei aber weiterhin eine Möglichkeit zu haben, eine gewisse Kontrolle über die Inhalte seines Titels auszuüben. Dies ist vor allem für Spieleheresteller nützlich, die bereits etablierten Content besitzen, durch den sich ihre Titel besser verkaufen lassen. Diese Teile des Contents könnten als Product Identity geschützt werden, während die restlichen Teile als Open-Game-Content freigegeben werden. Beispielsweise könnte eine Firma wie Nintendo die Charaktere der Mario-Reihe als Product Identity schützen, aber Teile der Spiele, die keinen Wiedererkennungswert haben, als Open-Game-Content freigeben. Wenn noch kein Content vorhanden ist, der solche einen Verkaufsanreiz bietet, ist die Verwendung der OGL im kommerziellen Bereich nicht ratsam, weil kein deutliches Unterscheidungsmerkmal zu Derivaten gegeben ist.

Das Creative-Commons-Projekt bietet mehrere Lizenzen an.[122] Diese können als Komplementäre zu den existierenden Softwarelizenzen betrachtet werden. Daraus ergibt sich auch das mögliche Einsatzgebiet der jeweiligen Lizenz in Kombination mit dem verwendeten Finanzierungsmodell. Die CC-BY Lizenz entspricht in etwa der BSD Lizenz, die CC-BY-SA Lizenz entspricht der GPL. Ein CC-BY-ND lizenzierter Content wäre passend zur klassischen Freeware, die zwar weitergegeben, aber nicht verändert werden darf.

Problematisch sind die NC-Varianten der Creative-Commons-Lizenzen, wie Möller (2006) zeigt. Diese an sich einsichtige Variante, die die kommerzielle Verwendung des darunter fallenden Contents ausschließt, verhindert tatsächlich fast alle Verbreitungsformen. So kann der Content weder über die Heft-CD einer Zeitschrift, noch über eine

[122] Eine Auflistung und Erklärung dieser Lizenzen findet sich auf der Webseite des Creative-Commons-Projekts, URL: http://creativecommons.org/ (02.01.2005).

werbefinanzierte Webseite verbreitet werden, weil beides streng genommen kommerzielle Nutzungsformen sind. Auch die Verwendung von NC-lizenziertem Content für ein Open-Source-Spiel ist ausgeschlossen, weil Open-Source-Lizenzen eine kommerzielle Verbreitung ausdrücklich erlauben.

Für reine Open-Source-Projekte ausgeschlossen ist auch Content, der unter die ND Varianten der Creative-Commons-Lizenz fällt, weil solcher Content nur unverändert weitergegeben werden darf. Es ist allerdings denkbar, wenn auch vermutlich unpraktisch, zwar eine Veränderung des Quellcodes zu erlauben, aber eine unveränderte Verbreitung des Contents zu fordern. Allenfalls für bestimmte Elemente, wie zum Beispiel Logos oder herausragende Spielfiguren sind ND Lizenzen in diesem Gebiet praktisch sinnvoll einsetzbar.

Die Wahl der Lizenzen für Code und Content eines Spieles hängen also hauptsächlich vom gewünschten Finanzierungsmodell ab. Daneben spielen aber auch weniger greifbare Aspekte eine Rolle, die zur Wahl einer Lizenz mit stärkeren Schutzmechanismen, als für das Finanzierungsmodell nötig, führen können. Im Interesse einer möglichst hohen Qualität und Quantität des für Freie Software zur Verfügung stehenden Contents, sollten diese Aspekte angegangen werden. Dies kann durch Aufklärungsarbeit unter den Erstellern solches Contents geschehen, um ihnen zu verdeutlichen, dass sie mit einer offenen Lizenz nicht etwa alle ihre Rechte abgeben.

6.2 Finanzierungsmodelle für Computerspiele

Im Folgenden wird die Übertragbarkeit der Open-Source-Finanzierungsmodelle auf die Spieleindustrie betrachtet. Dafür wird für jedes der im zweiten Kapitel vorgestellten Modelle untersucht, inwieweit es anhand der Ergebnisse aus dem vierten und fünften Kapitel mit der Entwicklung von Computerspielen vereinbar ist. Weiterhin wird gezeigt welche Chancen und Risiken bei den einzelnen Modellen bestehen.

6.2.1 Spielesoftware als Dienstleistung

Während in früheren Jahren Computerspiele einen sehr ausgeprägten Produktcharakter hatten, mehr noch als Anwendungssoftware, gibt es in letzter Zeit einen Wechsel hin zu einem Service- beziehungsweise Dienstleistungsmodell. Dieser Wechsel ist bedingt durch die zunehmende Vernetzung von Computern und auch von Spielkonsolen. Ein Mehrspielermodus wird immer mehr zur Grundvoraussetzung für ein erfolgreiches Spiel.

Der Dienstleistungsaspekt dominiert dann bei der nächsten Stufe der Vernetzung, den Massiv-Mehrspieler-Spielen (MMOG). Hierbei ist kein Einspielermodus mehr vorhanden, das „Produkt Spiel“ dient nur noch als Zugangssoftware zum „Service Spiel“.

Weil das Produkt nur noch eine Anzeige- und Steuerungssoftware für den Service ist, wird auch seine Bedeutung geringer. Besonders deutlich wird die Unwichtigkeit des Produktes bei Browserspielen, die als Client einen Webbrowser einsetzen.

Auch die Rolle des Spieleanbieters hat sich dadurch gewandelt. Obwohl die Clientsoftware weiterhin als Produkt verstanden werden kann, ist der Anbieter in erster Linie ein Dienstleister geworden. Das eigentliche am Spiel ist der Zugang zu den Spieldaten, die auf dem Server des Anbieters liegen. Dies ist vergleichbar mit der Rolle eines Anbieters von Bezahlfernsehen. Dieser verkauft zwar auch ein Produkt, nämlich den benötigten Dekoder, seine eigentliche Rolle besteht aber in der zur Verfügungstellung des Fernsehprogramms.

Das Verhältnis von Produkt und Dienstleistung wird im Allgemeinen auch durch das Geschäftsmodell verdeutlicht. Während zwar zumeist auch die Clientsoftware wie eine herkömmliche Software verkauft wird, fallen zusätzlich monatliche Gebühren für die Nutzung des Spiels an.[123] Diese monatlichen Gebühren übersteigen meistens den ursprünglichen Kaufpreis bereits nach kurzer Zeit, wie der Tabelle 5.1 auf Seite 66 zu entnehmen ist.

Der eigentliche Wert des Spieles liegt bei diesem Modell nicht mehr im Besitz der Clientsoftware, sondern im Zugang zum offiziellen Spieleserver. Aus diesem Grund ist beim Dienstleistungsmodell das Problem von nicht legitimen Programmkopien an sich nicht mehr vorhanden. Es besteht aber die Herausforderung, sämtlichen Content, der nicht frei verfügbar sein soll, vor unbefugter Nutzung zu schützen. Das kann auf verschiedene Weise geschehen. Zum einen ist es möglich, eine proprietäre Lizenz für den Content zu verwenden, auch wenn der Quellcode frei ist, wie es im Beispiel Planeshift gehandhabt wird. Zum anderen sind auch technische Sicherungen möglich. Diese sind zwar nie vollkommen sicher, doch gleiches trifft auch für Schutzmechanismen von Spielen zu, die als Produkte verkauft werden.

Ein anderes Problem ist die Absicherung des Spiels vor schädlichen Veränderungen am Open-Source-Client. Dieses Problem ließe sich möglicherweise durch Signierung von Clients umgehen. Dadurch könnten nur vom Hersteller genehmigte Clients auf die offiziellen Server zugreifen. Um damit die Vorteile des offenen Codes nicht wieder zu beseitigen, sollten spezielle Entwicklerserver zur Verfügung gestellt werden, die auch nicht signierten Clients den Zugriff erlauben.

Auf diese Art lassen sich die Vorteile von Open Source bei allen Spielen nutzen, die sich in Form einer Dienstleistung anbieten lassen. Dies sind neben MMORPG prinzipiell alle Spiele bei denen der Online-Modus den Kernbereich darstellt. Es ist zu erwarten, dass diese anderen Spiele in der Zukunft einen deutlich größeren Stellenwert erlangen.

[123] Meistens ist aber ein Monat Spielzeit bereits im Kaufpreis enthalten.

6.2.2 Spielesoftware als Auftragsarbeiten

Dieses Finanzierungsmodell wäre vor allem für den Bereich des Advergaming vorstellbar. Entwickler, die für diesen Bereich Spiele herstellen sind meist sehr klein und können somit deutlich von der Möglichkeit profitieren, Ressourcen aus dem Open-Source-Bereich für ihre Projekte zu verwenden. Die Auftraggeber sind im Allgemeinen branchenfremde Firmen, für die die Entwicklungskosten dieser Spiele eine Form der Werbung sind. Deswegen werden diese Titel im Normalfall kostenlos verteilt. Ob das Spiel einen offenen oder einen geschlossenen Quellcode hat, ist für die Auftraggeber vermutlich uninteressant.

Zu beachten ist beim Bereich des Advergaming allerdings das Markenrecht. Selbst wenn das komplette Spiel inklusive des Content frei verteilt wird, wird dennoch ein großer Teil dieses Contents markenrechtlich geschützt sein. Damit kann dieser Teil nicht unter eine freie Lizenz gestellt werden. Auch in diesem Bereich ist vermutlich die Lösung einer getrennten Lizenz für Quellcode und Content nötig.

Ein weiterer Bereich, in dem dieses Modell denkbar wäre, sind Open-Source-Spiele, bei denen ein Zusammenschluss mehrerer Spieler einen Entwickler mit der Implementierung von neuen Funktionen beauftragt und ihn dafür bezahlt. Fraglich ist bei einem solchen Modell aber, ob genügend solcher Aufträge regelmäßig zustande kommen, um ein Geschäftsmodell darauf zu gründen. In diesem Zusammenhang ist auf das bereits im zweiten Kapitel auf Seite 18 erwähnte Beispiel der Firma Transgaming zu verweisen.

6.2.3 Spiele von Hardwarefirmen

Zunehmend enthalten hoch integrierte Geräte, so genannte „Appliances“[124], auch Spielesoftware. Die Hersteller dieser Geräte versuchen, mittels dieser enthaltenen Software die Attraktivität der Hardware zu erhöhen. Oft ist die Hardware dieser Geräte zwar ähnlich der üblichen PC Hardware, weicht aber in einzelnen Bestandteilen von dieser ab. Dementsprechend ist es für diese Hersteller von Vorteil, wenn sie auf Spiele mit offenem Quellcode zurückgreifen können, die sich leicht an die geänderten Bedingungen anpassen lassen. Hardwarefirmen werden also meist keine kompletten Spiele selbst schreiben, sondern vielmehr auf bestehende Open Source Spiele zurückgreifen und diese sowohl verbessern, als auch an ihre spezielle Plattform anpassen. Damit können sie zur Weiterentwicklung bestehender Spiele beitragen und die Bekanntheit dieser Spiele erhöhen.

Weil die Hardware der Appliances meist sehr einfach ist, sind keine grafisch aufwendigen Spiele auf diesen möglich. Zudem sind, anders als bei Spielekonsolen, die

[124] Dies sind häufig Satelliten-Receiver, aber auch andere elektronische Geräte, soweit sie Zugriff auf einen Bildschirm haben, sind in diesem Zusammenhang möglich.

Bedienelemente zumeist nicht besonders gut für viele Spiele geeignet. Als Resultat werden im Allgemeinen nur einfache Puzzlespiele auf solchen Geräten eingesetzt. Gerade in diesem Genre sind Open Source Spiele bereits stark vertreten. Für Hersteller solcher Hardware ist der Einsatz von Open Source Spielen also die optimale Lösung, sie müssen aber darauf achten, Lizenzbestimmungen – vor allem von viralen Lizenzen – bezüglich der Veröffentlichung von Veränderungen einzuhalten.

6.2.4 Spiele von Softwarefirmen

Bei der Spieleentwicklung durch Softwarefirmen entsteht am Ende ein klassisches Produkt, welches verkauft werden soll. Während üblicherweise in diesem Finanzierungsmodell ein Serverprodukt verkauft werden soll, während die Clientsoftware kostenlos verteilt wird, ist dies bei Spielen umgekehrt, die Serversoftware ist hier meist kostenlos, der Client wird als Produkt verkauft. Eine Umkehrung dieses Modells erscheint nur in Ausnahmefällen möglich, da für den Betreiber eines Spieleservers kein finanzieller Anreiz darin besteht, den Server für Spieler nutzbar zu machen. Es wäre zwar ein Modell denkbar, bei dem Spieler für die Nutzung des Servers wiederum eine Gebühr bezahlen müssen, doch wäre dieses lediglich eine leichte Abwandlung des Dienstleistungsmodells.

Welche Modelle für reine Softwarefirmen denkbar sind, lässt sich an den Ansätzen von Bioware, id-Soft und Radon Labs feststellen. In allen drei Fällen bleiben die Spiele im Ergebnis klassische, proprietäre Produkte. Für den reinen Nutzer ändert sich also fast nichts im Vergleich zum konventionellen Modell, er kann aber mit kostenlos zusätzlich verfügbarem Content rechnen.

Für andere Entwickler, vor allem solche von kleineren Firmen und nicht kommerziellen Gruppen, ergeben sich durch diese Form der Offenheit aber weitergehende Möglichkeiten. Für sie wird es einfacher, selbst die Entwicklung eines Spieles durchzuführen, weil sie auf bestehende Teile zurückgreifen können. Dies ist bei dem Modell, welches Bioware bei Neverwinter Nights anwendet, allerdings nur indirekt möglich, weil ein Spieler immer auch das Originalspiel benötigt und nur ein nicht kommerzieller Vertrieb gestattet ist.

Das Modell von id-Soft ermöglicht unter bestimmten Einschränkungen auch eine kommerzielle Nutzung. Weil aber stets nur eine Engine als Freie Software zur Verfügung steht, die zu diesem Zeitpunkt technisch eine Generation im Rückstand ist, gibt es Begrenzungen in den Möglichkeiten. Weiterhin hinderlich für eine kommerzielle Nutzung ist die virale Lizenz, die es erfordert, eigene Verbesserungen an der Engine offen zu legen. Da dies aber nicht für den Content gilt, kann eine kleine Spielefirma, deren Content sich von den kostenlos verfügbaren Mods abhebt, durchaus auch auf eine kommerzielle Nutzung setzen. Entwickler, die auf diese Weise Erfolg

haben, werden wiederum für ein zukünftiges Projekt eine kostenpflichtige, proprietäre Engine von id-Soft bevorzugen, wodurch sich das Modell auch für diese lohnt.

Radon Labs geht noch einen Schritt weiter, indem eine kommerzielle Nutzung der Nebula Engine gestattet wird, ohne dass eine Verpflichtung zur Offenlegung des geänderten Quellcodes besteht. Dieses Modell garantiert eine höchst mögliche Verbreitung der eigenen Engine. Die Nutzer dieser Engine profitieren optimal von dieser Form der Lizenz. Aber auch für Radon Labs lohnt sich dieses Modell: Zum einen geben Nutzer Teile ihrer eigenen Änderungen wieder frei[125], wodurch Radon Labs günstig Verbesserungen erhält, zum anderen ermöglicht dies ein Zusatzgeschäft, indem Werkzeuge verkauft werden, die die Benutzung der Engine vereinfachen. Auch ist es denkbar, dass bestimmte Sonderversionen der Engine mit geschlossenem Quellcode verkauft werden können.

Firmen, die zwar selbst keine komplette Engine herstellen, aber stattdessen andere Middleware für Spiele vertreiben, können auf eine ähnliche Weise verfahren. Je nach dem, wie die Software genau aufgebaut ist und wie die Marktposition der eigenen Software ist, kann entweder eine ältere Version als Open Source freigegeben werden, während die aktuelle geschlossen bleibt, oder es wird die aktuelle Version freigegeben und Zusatzleistungen werden verkauft. Hierbei ist auch ein Übergang zwischen den Modellen möglich, wenn zum Beispiel der Einstieg in einen Markt geschaffen werden soll oder der Punkt erreicht ist, an dem die Position gefestigt ist.

6.2.5 Universitäre Entwicklungen

Anders als im Bereich der Infrastruktur-Software, zum Beispiel bei Web- und Mailservern, gibt es in der Spieleentwicklung kein direktes Eigeninteresse der Universitäten für eine Beteiligung.

Trotzdem sind universitäre Projekte wichtig für die Spieleentwicklung. So finden Ergebnisse aus den Gebieten der Künstlichen Intelligenz und der 3D Grafik Eingang in Spielesoftware. Umgekehrt werden wiederum Spiele für Forschungsarbeiten in diesen Bereichen verwendet. Dies ist dann besonders einfach möglich, wenn der Quellcode der Spiele vorliegt. Sowohl für die wissenschaftliche Forschung, als auch für Entwickler von Spielen ist es also von Vorteil, wenn der Zugang zu neuen Ergebnissen in grundlegenden Bereichen möglichst offen ist.

Universitäten können also Programmcode zur Spieleentwicklung beitragen, wenn es sich um allgemein verwendbare Bausteine handelt. Der Content von Computerspielen ist aber jeweils auf ein spezielles Projekt ausgerichtet, und somit ist es nur

[125] Hierzu sind sie zwar nicht verpflichtet, dennoch lohnt es sich oft, dafür zu sorgen, dass eigene Änderungen in den offenen Baum des Codes integriert werden. Andernfalls müssten diese Verbesserungen mit jeder Änderung der offenen Version neu an diese angepasst werden.

in Ausnahmefällen zu erwarten, dass dieser im Rahmen der universitären Arbeit entsteht.[126]

Die Entwicklung von kompletten Spielen im Rahmen von Universitätsprojekten kann zum einen aus der Richtung der Computerwissenschaften kommen. Hier könnte im Rahmen von Gruppenprojekten die Entwicklung eines Computerspiels stattfinden. Diese Projekte könnten aus unterschiedlichen Bereichen wie zum Beispiel Softwaretechnik oder Computergrafik stammen.

Weiterhin sind Spieleprojekte in künstlerischen Studiengängen denkbar. Hierfür könnten frei verfügbare Spielebaukästen verwendet werden, um den von Studenten entwickelten Content in Spieleform zu bringen. Solche Projekte sind für die kommerzielle Spieleentwicklung aber vermutlich eher uninteressant, wenn nicht neue Spielprinzipien dadurch entwickelt werden.

6.2.6 Spieleentwicklung als Freizeitprojekt

Wenn die Entwicklung eines Spiels als nicht kommerzielles Projekt unternommen wird, so gibt es wenig, was gegen eine Veröffentlichung unter einer freien Lizenz spricht. Weil ein Spiel bei solch einem Projekt ohnehin zum kostenlosen Download angeboten wird, entstehen keine Verluste dadurch, dass andere dieses Spiel ebenfalls weiterverbreiten können. Die durch die freie Lizenz gestattete kommerzielle Nutzung erlaubt es, das Spiel über die Heft-CD von Computerzeitschriften weit zu streuen.

Problematisch können Freie Lizenzen allerdings dann werden, wenn externe Teile in das Spiel übernommen werden. So verwenden einige Spieleprojekte Code, den sie von Firmen lizenziert haben. Diese Codeteile dürfen nicht mit veröffentlicht werden, wodurch eine quelloffene Version nicht funktionsfähig wäre. Ein weiteres Problem sind Grafiken und Sounds, die von externen Quellen stammen. Häufig sind diese zwar für eine nicht kommerzielle Nutzung freigegeben, dies reicht aber nicht für ein Open-Source-Projekt.

Um diese Probleme zu beseitigen ist es erforderlich, dass die bisher weitgehend getrennten Gemeinschaften der Programmierer auf der einen Seite und der Grafiker und Musiker auf der anderen Seite zusammengeführt werden. Dabei ist zu beachten, dass in beiden Gruppen ganz unterschiedlich gearbeitet wird. Diese unterschiedlichen Herangehensweisen müssen abgeglichen werden, damit die Zusammenarbeit funktionieren kann.

Künstler veröffentlichen in entsprechenden Communities oft viel Content, machen sich aber selten über die Lizenzen Gedanken. Dadurch sind diese Content-Teile nicht für Freie Software nutzbar, weil eine kommerzielle Nutzung fast immer ausgeschlossen

[126] Denkbar wären zum Beispiel Schauspielstudenten, die die Sprachausgabe für ein Spiel im Rahmen eines Projektes erstellen.

wird. Der Grund für die Wahl einer solchen Lizenz liegt im Allgemeinen nicht darin, dass die Verwendung in Freier Software ausgeschlossen werden soll. Zumeist wird die Verwendung des Contents in Projekten, die für den Nutzer kostenlos sind begrüßt. Es soll lediglich verhindert werden, dass sich eine Firma für ein kommerzielles Spiel des Contents bedienen kann.

Um es Künstlern, die sich bisher wenig mit Lizenzen beschäftigt haben, zu erleichtern, die richtige Lizenz zu wählen und eine möglichst große Freiheit des entstehenden Contents zu gewährleisten, wurde das Creative-Commons-Projekt[127] gegründet. Dieses bietet unterschiedliche Lizenzen an, die zum Teil kompatibel mit den Lizenzen freier Software sind. Dieses Projekt müssten Grafiker und Musiker kennenlernen und die angebotenen Lizenzen nutzen, um das Problem des knappen Contents für freie Spieleprojekte zu beseitigen.

Von einer anderen Richtung her kommen Spieleprojekte, die kommerzielle Spiele modifizieren oder ergänzen. Auch bei diesen Projekten ist es unproblematisch, den selbst erstellten Quellcode unter einer Freien Lizenz zu veröffentlichen. Es ist aber zu beachten, dass dieser Quellcode an sich nutzlos ist, weil zur Ausführung immer das proprietäre Originalspiel benötigt wird. Trotzdem bietet sich eine Veröffentlichug der Quellen an, wie es beim Beispiel Neverwinter Nights bei den meisten von Nutzern erstellten Modulen praktiziert wird. Dies führt nämlich dazu, dass die Nutzer voneinander anhand des Codes lernen können, wodurch sowohl Quantität als auch Qualität der Module steigen.

Viele Spieler, die eigene Modifikationen erstellen, sind sich über die Rechtslage kaum im Klaren. Häufig werden für solche Modifikationen urheberrechtlich geschützte Materialien ohne Erlaubnis verwendet. Dies führt zum Teil dazu, dass nach Monaten oder Jahren Arbeit an dieser Modifikation der Rechteinhaber davon erfährt und gegen die Urheberrechtsverletzung vorgeht.[128]

Um solche Vorfälle zu vermeiden ist es wichtig, dass Spieler, die Content auf irgendeine Art erstellen oder bearbeiten, über das Urheberrecht aufgeklärt werden. Dies beinhaltet eine Aufklärung über die Rechte der ursprünglichen Künstler und welche Konsequenzen diese haben, also unter welchen Umständen die Beiträge anderer verwendet werden dürfen. Zudem müssen sie auch über ihre eigenen Rechte an ihrem Content aufgeklärt werden. Dies sollte eine Erklärung der möglichen Lizenzen beinhalten, damit sie sowohl den Schutz bekommen, den sie wünschen, als auch die freie Verfügbarkeit von möglichst viel hochwertigem Content gewährleistet wird.

[127] Homepage URL: http://www.creativecommons.org (11.11.2005).

[128] Beispielhaft sei hier das „Ultima 1 – A Legend Reborn“ Projekt genannt, das nach einigen Techdemos schließlich eingestellt werden musste. URL: http://www.peroxide.dk/ultima.shtml (14.03.2006).

6.3 Wie weit passen Open Source und Computerspiele zusammen?

Nach der Betrachtung der Beispiele lässt sich erkennen, dass es durchaus möglich ist, kommerziell erfolgreiche Spiele mit einem offenen Quellcode und unter Verwendung des Basarmodells zu entwickeln. Voraussetzung dafür ist die Anwendung eines der betrachteten Finanzierungsmodelle (mit Ausnahme der nicht kommerziellen Entwicklung als Freizeit-Aktivität beziehungsweise der universitären Entwicklung). Diese Modelle implizieren aber eine Einschränkung der Freiheit des in den Spielen enthaltenen Contents, wobei der Grad dieser Einschränkung variieren kann.

Es zeigt sich auch, dass die Offenheit des Contents nicht nur von der Wahl der Finanzierung abhängt, sondern auch von Entscheidungen, die aufgrund von Gefühlen getroffen werden. In diesen Fällen könnte eine größere Freiheit erreicht werden, ohne dass für die Entwickler des Spiels negative Konsequenzen zu befürchten sind. Voraussetzung hierfür ist aber, dass den Künstlern und auch den Entscheidern die Vorteile von offenerem Content deutlich gemacht werden können. Die Lizenzen, die für die einzelnen Komponenten eines Spiels gewählt werden, sollten sorgfältig auf ihre Folgen überprüft werden, um eine so große Offenheit wie möglich zu erlangen, ohne das Finanzierungsmodell des Spiels dabei zu zerstören.

Je offener ein Spiel insgesamt aufgebaut ist, desto länger kann es auf dem schnellebigen Spielemarkt bestehen und desto mehr zusätzliche Anwendungsmöglichkeiten und damit Märkte ergeben sich. Es ist zu erwarten, dass sich dies positiv auf das finanzielle Ergebnis des Spiels auswirken wird. Dementsprechend ist ein Ausgleich zu finden, zwischen dem Risiko der größeren Offenheit und den sich daraus ergebenden Chancen.

Diese bestehen zu großen Teilen in den Netzwerkeffekten, denen eine Software beziehungsweise ihr Entwickler ausgesetzt ist, wenn er sie unter einer Open-Source-Lizenz veröffentlicht. Ein Gewinn durch größere Offenheit entsteht laut Nüttgens und Tesei (2000b, Seite 20) dann, wenn zum einen durch die Nutzung des resultierenden Netzwerkes günstiger produziert werden kann, zum anderen wenn die Investitionen in das Netzwerk geringer sind als der daraus entstehende Gewinn. Eine finanziell sinnvolle Nutzung von Open-Source-Prinzipien in der Spieleentwicklung hängt also von diesen Bedingungen ab.

Um den größtmöglichen Vorteil aus dem Netzwerk zu ziehen, muss das Basarmodell von Raymond (1999) umgesetzt werden. Wichtig ist dafür, dass die (zukünftigen) Nutzer eines Programms in die Entwicklung mit einbezogen werden. Der erste Schritt dazu besteht darin, dass die Anwender zu ihren Wünschen befragt werden. Dies wird in der Spieleentwicklung bereits vermehrt getan, indem Spieler schon in der Planungsphase zu Designentscheidungen befragt und möglichst frühe öffentliche Beta-Tests durchgeführt werden. So wurde zum Beispiel bereits bei Beginn der

Entwicklungsarbeiten an „Neverwinter Nights 2“ ein Internet-Forum eingerichtet, in dem die Entwickler verschiedene Bereiche des Spiels mit potentiellen Spielern diskutieren.[129] Auch bei der Entwicklung des vierten Teils der Civilization-Reihe wurde auf eine frühe Beteiligung der Nutzer gesetzt, indem Spieler, die bereits Erfahrungen mit vorhergehenden Teilen der Reihe hatten, zu frühen Beta-Tests eingeladen wurden.

Ein besonderes Feld, in dem eine möglichst große Offenheit einer Software durch eine Vergrößerung des Marktes von Vorteil ist, ist das Anpassen eines Spieles für die Zugänglichkeit durch körperlich eingeschränkte Nutzer. Dies können zum Beispiel Änderungen sein, die ein Spiel für Farbenblinde oder Schwerhörige[130] überhaupt erst nutzbar machen. Bei Open-Source-Lösungen fällt dies am leichtesten, Lösungen mit geschlossenem Quellcode sind nur dann sinnvoll realisierbar, wenn sie offene Schnittstellen für entsprechende Standardbibliotheken bieten.

Das Portieren einer Software auf andere Plattformen und Betriebssysteme ist ohne Zugriff auf den Quellcode so gut wie unmöglich. Durch die Verfügbarkeit auf mehr Plattformen wird der Markt vergrößert, sowohl in der Breite (es können mehr Nutzer zu einem bestimmten Zeitpunkt etwas mit dem Programm anfangen) als auch in der Zeit (die Lebensdauer eines Spiels verlängert sich, wenn es als Klassiker auf eine neue Plattform gebracht werden kann). Bei Closed-Source-Spielen kann der Port nur durch die Firma selbst oder einen Lizenznehmer erfolgen. Open-Source-Spiele erlauben der Community, das Spiel selbst auf die Plattform ihrer Wahl zu portieren. Wenn das Spiel so interessant ist, dass sich freiwillige Entwickler der Portierung annehmen, bedeutet dies eine für den Hersteller kostenlose Erweiterung des Marktes.

In der Bedingung, dass weiterhin Gewinn durch das Spiel erzielt werden soll, besteht eine besondere Herausforderung. Dafür muss ein Anreiz geschaffen werden, das Spiel zu kaufen, auch wenn der Quellcode frei verfügbar ist. Martin (2005) schlägt vor, dass es sich zumindest für unabhängige Entwicklerstudios durchaus lohnen kann, wenn das Spiel frei verbreitet wird – solange es einen interessanten Mehrspielermodus gibt, der sich nur mit einer gekauften Version verwenden lässt. Dies liegt daran, dass sich der Bekanntheitsgrad durch die freie Verbreitung deutlich erhöhen lässt, was durch Netzwerkeffekte zu mehr Spielern führt. Große Studios, die durch Werbung bereits einen genügenden Bekanntheitsgrad erreichen, können hiervon allerdings nicht unbedingt profitieren. Trotzdem kann diese Möglichkeit auch von diesen Entwicklern angewendet werden, unter der Voraussetzung, dass sich das Spiel über den Mehrspielermodus verkauft. Die Fortsetzung dieser Überlegung führt zu Spielen, die fast nur noch auf das Online-Spielen ausgerichtet sind.

Ähnlich, wie es bei anderen Bereichen der Software-Entwicklung bereits passiert ist, ist auch im Bereich der Computerspiele eine Wandlung vom Produktmarkt zu

[129] Das Forum befindet sich unter der URL: http://nwn2forums.bioware.com/forums/viewforum.html?forum=95 (30.03.2006).
[130] Untertitel für Computerspiele, URL: http://gamescc.rbkdesign.com/ (02.11.2005).

einem Servicemarkt zu erkennen. Der Bereich, in dem diese Veränderung am weitesten fortgeschritten ist, sind die MMORPG, also reine Online-Spiele; aber auch bei anderen Spielen sind Ansätze dazu zu erkennen, besonders wenn sie über eine Online-Anbindung verfügen. Infolgedessen sind in diesem Bereich die besten Möglichkeiten für eine Entwicklung nach einem Open-Source-Modell vorhanden. Zu klären bleibt dabei die Frage nach einem Schutz vor Manipulationen, die das Gleichgewicht beeinträchtigen können. Dies ist aber ein technisches Problem, welches in ähnlicher Form auch bei Closed-Source-MMORPG auftritt.

Eine zusätzliche Problematik für Closed-Source Spielehersteller liegt im Kopierschutz. Wie Walsdorfer (2003) feststellt, sind die üblichen Kopierschutzverfahren bei Spielen für den Nutzer oft sehr kompliziert. Er erklärt, dass die Kaufbereitschaft eines Spielers sinkt, wenn der Kopierschutz für ihn zu aufwendig wird. Die Frage, die sich dadurch stellt, ist nun die nach einem Ausgleich zwischen den Interessen des Herstellers und denen des Kunden. Die bisherigen Versuche scheinen an diesem Ausgleich zu scheitern. Ein Open-Source-Spiel mit einer Finanzierung nach einem Dienstleistungsmodell umgeht dieses Problem.

Das Beispiel der Quake Engine und in anderer Form auch der Nebula Engine zeigt, dass es unter bestimmten Umständen sinnvoll ist, eine Spieleengine als Open-Source-Produkt bereitzustellen. Dies gilt anscheinend aber nur für solche Engines, die sich – aus welchen Gründen auch immer – nicht an andere Entwicklerstudios gegen Entgelt lizenzieren lassen. Die dadurch entstehende weitere Verbreitung der Engine ermöglicht auf drei Arten Vorteile für ihre Entwickler. Zum einen kann sie als Werbung für eine modernere Variante derselben Engine dienen, die kostenpflichtig lizenziert wird. Als zweites wird dadurch ein Markt für zusätzliche Leistungen zu dieser Engine, wie zum Beispiel Support oder Toolkits geschaffen. Und nicht zuletzt können die ursprünglichen Entwickler von den Beiträgen anderer Programmierer profitieren.

Abschließend lässt sich festhalten, dass Open-Source-Software auch vor dem Bereich der Computerspiele nicht halt macht. Die großen Unterschiede zwischen Spielen und sonstiger Software verlangen aber Lösungen, die sich von den bisher im Open-Source-Bereich üblichen unterscheiden.

6.4 Ausblick

Eine Übertragung dieser Ergebnisse auf andere Einsatzbereiche von Spielen als den PC Markt wäre interessant. Es wurde bereits erwähnt, dass der weltweite Markt für Konsolenspiele denjenigen für Computerspiele deutlich übertrifft. Inwieweit sich Open-Source-Prinzipien auch hierhin übertragen lassen ist fraglich. Dies ist zum einen dadurch bedingt, dass Konsolen im Unterschied zu Computern keine Entwicklungsplattformen sondern reine Anwendungsplattformen sind. Zum anderen kann nicht

jeder Spiele für Konsolen herstellen und verkaufen, dazu ist zunächst eine Lizenz des Konsolenherstellers zu erwerben. Diese ist mit hohen Kosten verbunden. Zusätzlich wird pro verkauftem Spiel eine weitere Lizenzgebühr berechnet. Dies schließt eine klassische Open-Source-Verbreitung aus. Diese Problematik lässt sich schon daran erkennen, dass eine umfangreiche Modding-Kultur wie beim PC im Konsolenbereich nicht existiert.

Die tragbaren Konsolen, wie Nintendo DS und Sony PSP stellen Entwickler vor ähnliche Schwierigkeiten. Es gibt allerdings auch eine tragbare Konsole, für die frei entwickelt werden kann, die GPX2 von Gamepark Holdings.[131] Diese ist interessanterweise auf einem Linux System aufgebaut und unterstützt Open-Source-Bibliotheken wie SDL. Somit steht einer Entwicklung von Open-Source-Spielen für dieses Gerät wenig im Weg. Auch bei diesem System ist die Entwicklunsplattform aber ein Computer, nicht das Gerät selbst. Eine Betrachtung der kommerziell entwickelten Spiele für diese Konsole könnte interessant sein.

Die zunehmende Anbindung von Spielekonsolen an das Internet könnte dafür sorgen, dass es möglich wird, zumindest kleinere kostenlose Spieleprojekte auch für Konsolen zu veröffentlichen. Bei diesen wäre es dann denkbar, dass auf Open Source gesetzt wird. Hierfür bleibt aber abzuwarten, wie sich die Online-Dienste der Konsolenanbieter entwickeln und inwieweit sie eine solche Nutzung zulassen werden. Im Moment gibt es jedenfalls noch keine einzige Open-Source-Software über die bestehenden Onlinedienste der Konsolen.

Ein weiterer interessanter Bereich sind die mobilen Endgeräte. Im Unterschied zu tragbaren Spielekonsolen ist bei Mobiltelefonen der Markt noch offen und eine Open-Source-Entwicklung durchaus denkbar. Auch in diesem Bereich zeichnet es sich allerdings ab, dass eine Lizenzpflicht für Spiele kommen wird. Die Gründe dafür liegen, ähnlich wie bei den klassichen Spielekonsolen, darin, dass minderwertige Produkte drohen, den Markt zu zerstören. Trotzdem ist dies ein interessanter Bereich, zum einen aufgrund der Vereinheitlichung der Entwicklungsumgebung durch J2ME[132] und BREW[133], zum anderen wegen seiner hohen wirtschaftlichen Bedeutung – der Spielemarkt für mobile Endgeräte soll laut Datamonitor[134] im Jahre 2006 weltweit 17,5 Milliarden US-Dollar Umsatz erwirtschaften.

Auch im Bereich der Computerspiele sind weitere Untersuchungen denkbar, die zur Klärung der Thematik beitragen können. Mögliche Fragestellungen sind: Wie hoch ist der Anteil an professionellen Spieleentwicklern in der Open-Source-Spieleentwicklung, und wie sieht dies in anderen Bereichen der Softwareentwicklung aus? Wie lassen sich Künstler und Entwickler effizienter zusammenbringen? Welchen Einfluss haben Ko-

[131] Offizielle Webseite zu dieser Konsole: http://www.gpx2.com/ (29.12.2005).
[132] Webseite von SUN zu J2ME: http://java.sun.com/j2me/index.jsp (27.12.2005).
[133] Webseite von Qualcomm zu BREW: http://brew.qualcomm.com/brew/en/ (17.12.2005).
[134] Zitiert nach Leavitt (2003).

pierschutz beziehungsweise freie Verbreitungsmöglichkeiten auf die Verkaufszahlen eines Spiels? Wie lässt sich eine Client-Server basierte Struktur, zum Beispiel für MMORPG gegen Manipulationen sichern? Eine Beantwortung dieser Fragen kann dabei helfen, tragfähige Konzepte für Spiele unter Open Source Lizenzen zu entwickeln.

7 Fazit

Am Anfang dieser Arbeit stand die Fragestellung, warum sich bislang so wenig Open-Source-Software im Bereich der Spiele-Software findet. Die Antwort darauf ist die grundsätzliche Verschiedenheit von Spielen und anderer Software. Bei dieser Antwort sollte es aber nicht belassen werden, war doch die Zielsetzung, Möglichkeiten aufzuzeigen, wie diese Situation verändert werden kann.

Die Untersuchung der Beispiele zeigte, dass Open Source durchaus einen Platz in der Spieleentwicklung einnehmen kann. Das gilt auch für kommerzielle Spiele, deren Entwickler von einer Öffnung finanziell profitieren können. Es wurde aber auch deutlich, dass sich die üblichen Finanzierungsmodelle von Open-Source-Software nicht einfach auf Computerspiele umsetzen lassen, sondern an diese angepasst werden müssen. Eine getrennte Lizenzierung von Code und Content sind das wichtigste Resultat dieser Anpassung.

Eine Überraschung ergab sich im Verlaufe der Arbeit durch das Aufkommen reiner Online-Spiele. Diese verändern den Charakter der klassischen Computerspiele vollkommen. Weil bei diesen Spielen kein Produkt verkauft, sondern eine Dienstleistung vermietet wird, ergeben sich große Chancen für Open Source, über diesen Weg eine größere Wichtigkeit im Spielebereich zu erlangen.

In der Hoffnung, dass diese Arbeit etwas zu diesem Ziel beitragen kann, möchte ich mich dem anschließen, was Bernd Beyreuther, der Managing Director der Spielefirma Radon Labs im Interview sagte: „Ich denke, dass es langfristig gar keine Alternative zu Open Source geben kann.“

A Interviews mit Entwicklern

In diesem Anhang sind die geführten Interviews, soweit sie in schriftlicher Form vorhanden sind, abgedruckt. Weil die Entwickler weltweit verteilt sind, wurden die Interviews überwiegend in englischer Sprache durchgeführt. Auch wenn diese zum Teil nicht die Muttersprache der Interviewten ist, ergaben sich daraus keine größeren Schwierigkeiten, weil die Entwicklung und Kommunikation in allen untersuchten internationalen Projekten ebenfalls zumindest teilweise auf Englisch durchgeführt wird.

A.1 Freeciv Entwickler

Die Hauptentwickler von Freeciv wurden über die Spieleforen, die Entwickler Mailingliste und die CVS Beiträge ermittelt. Außerdem wurden sie selbst befragt, wer zu den wichtigen Entwicklern gehöre.

A.1.1 Per Inge Mathisen

How and when did you get involved with Freeciv?

According to http://www.freeciv.org/index.php/Special:People my first contributions were to version 1.7.1, which was released in 1998. I started hacking on the server, the Norwegian translation (a little bit) and rulesets.

Was this your first involvement in a free software project?

Yes.

What is your main task in Freeciv development?

Server and AI development. Got involved a lot in ruleset development (game design).

What is the biggest problem for Freeciv?

Lack of artists. Lack of time to do all exciting things we want to do ...

What is the relation to the commercial Civs? How does Freeciv compare to them?

We have no relation, except that Freeciv can emulate civ1 and civ2 rules, and started out looking a lot at what civ1 and civ2 were doing in the beginning.

Did this relation change over the course of Freecivs history?

Rather recently (last year? two?) we started caring a lot less about "compatibility" with civ2 rules, and began developing rules independently, by being inspired by the best of various world-building TBS games out there (the civs, master of orion, master of magic, etc.).

How do you think this relation will change in the future?

The drift away from being inspired by civ1/2 alone will continue. I expect Freeciv (default rules) to become quite distinct from any other game out there eventually. And the best, of course.

This seems to be the case with other Free Software clones of popular games too. They start out as clones and when the base is there, new ideas are built in.

Yes, cooperative design work is very hard in free software projects.

Is the player base different between Freeciv and commercial Civs?

Who knows.

What is your goal with Freeciv?

Become the greatest "civilization builder" game ever.

What is the biggest problem in getting content (graphics, sounds) for Freeciv?

Sounds. There seem to be almost zero good quality, free sounds on the internet that are verifiably free and that we can use. There are tons and tons of "public domain" sounds with very dubious credentials and no authorship history - but they are to us completely useless, since we take copyright issues seriously.

We had problems getting graphics too, but this has been improving markedly lately.

Would you think this because the communities of programmers and artists are quite different when thinking about licenses? Or is it just that there is not enough contact between both groups?

Different mentality, different history, and more fear of being ripped off.

Most artists seem to instinctively go for "non-commercial" licenses, which is unacceptable for us.

How do you think was it possible to get those artists involved?

The web forum helped a lot. IRC helps a bit, too. The artists seem to dislike mailing lists.

What are the relations to those who mod the commercial civs?

They are generally very positive, generally subject to the same limitations in license mentality as described above, and the added problem of often a very limited care and understanding of copyright law. We have had to be extremely careful and always double-check anything coming from that direction to ensure we do not unintentionally get copyright violations into the game.

A.1.2 Vasco Alexandre Da Silva Costa

How and when did you get involved with Freeciv?

It was summer, classes were over, and I had already played all the games I owned to death. So I started rummaging through my Linux CD collection searching for games to play. I got Freeciv from an Infomagic CD which had a mirror of the sunsite.unc.edu[135] FTP site in it. I thought the game was very interesting, but found the user interface somewhat lacking. So I tried to help with that.

My first write to Freeciv CVS as an annointed developer, so to speak, was in April 13th 1999.

Before that, I posted my wares at the mailing-list. My first post to the development mailing-list was probably around the summer of 1998. I remember Per being around at the time.

Was this your first involvement in a free software project? (If not, what was your first?)

This was my first involvement in a free software project. I had however contributed to a non-commercial software project before (a MUD[136] called Ultra-Envy), which has source code available.

What is your main task in Freeciv development?

GTK+ client development. Although lately I have worked a bit more on the game engine, like effects, and have dabbled elsewhere. Including the server networking code.

[135] URL: http://www.ibiblio.org/metalab/old_html/history.html
[136] URL: http://en.wikipedia.org/wiki/MUD

What is the biggest problem for Freeciv?

Lack of artists and system administrators. There is also some lack of code developers, but less pronounced.

Do you think the lack of artists is only to the mentioned licensing issues or are there other reasons why artists are less present in projects like freeciv?

I think licensing is actually a minor issue. The problem is how to attract a certain critical mass of persistent talent. Most projects are started by programmers for rather obvious reasons. Often these sorts of environments are not the place an artist would fit in. There is a language barrier between a programmer's discussion forum and an artist's. You do find the rare person who knows how to bridge both, but that person is very, very hard to find.

What is the relation to the commercial Civs? How does Freeciv compare to them?

Did this relation change over the course of Freecivs history?

Freeciv used to be modeled on the original Civilization rules, but for multiplayer only (it had no single-player mode). Later optional Civilization II like rules were supported as well, slightly tweaked for multiplayer. Then it got single-player mode and computer controlled players.

How do you think this relation will change in the future?

Lately the developers have agreed to break from the Civilization set of rules in several aspects.

Speaking personally, I always disliked fiddling with micromanaging citizens, riots, changing a unit's homecity etc. I have played Civilization III and while I enjoyed some aspects of it, I found it less interesting than Civilization II in others. To be honest, I think the best game in this veign I ever played was Master of Magic from Simtex, with Master of Orion II second, and Civilization I & II tied for third place.

These features are ingrained quite deep in the game engine, so they may take some time to change.

Do you think this is common, that free software games tend to copy existing proprietary games first and later start to develop in different directions?
What do you think are the reasons for this?

I think free software games are much like proprietary games in this respect. You have clones, improved clones, and the rare oddball.

Clones:
proprietary: Herzog Zwei, Dune 2, Command and Conquer, Warcraft
free: Stratagus

Oddballs:
proprietary: Tetris, PacMan, Black & White
free: Nethack, Holotz's Castle

The one difference in free software games is that they endure. Code is reused for decades, and progress is steady and non-relentant. Proprietary games are often one-hit. With some expections (Quake, Unreal Tournament, etc).

Is the player base different between Freeciv and commercial Civs?

I have talked with several Freeciv players which claimed to never have played a commercial Civilization game. Speaking personally I have played both.

What is the biggest problem in getting content (graphics, sounds) for Freeciv?

Sound is the biggest problem. Licensing art in general is a real headache and we do our best to handle licensing issues to ensure we are not infringing someone else's copyright. Many artists are unfortunately quite oblivious to the licensing aspect, this means extra work for the developers.

How can one teach the artists about copyright and licenses?

By being frank and honest, explaining the licenses the best we can (we are not lawyers though!) and having some sort of filter between the world outside and the release versions. Even so, every so often something passes between the nets, but often someone notices that it is using someone elses copyright, and we always remove the infring piece in those cases.

What is your goal with Freeciv?

To have fun. :-)

A.2 Planeshift Entwickler

Die anzusprechenden Entwickler wurden über den Development Teil des Spieleforums ermittelt.

A.2.1 Andrew Craig

How and when did you get involved with Planeshift?

I responded to a post on gamedev.net. I've actually found my post:

http://gamedev.net/community/forums/topic.asp?topic_id=65980&whichpage=1&#322459

So around Nov, 2001

Basically at that point I had graduated from University and was looking for something to do as I was job searching.

Was this your first involvement in an (open source) project? (If not, what was your first?)

No, I was working on my own wrappers for DirectX. It never really gained any strength though.

What is your main task in Planeshift development?

My main tasks are:

1) Team leadership. I am responsibly for the overall development of the project from the engine/client point of view. I work with the team director (Luca/Talad) to develop the master plan for the development of future versions. From there I coordinate with all our other developers to give them tasks that they want/can do.

2) Server maintenance. Since the server we are using is in linux and the other main programmer uses windows I handle a lot of the server setup/build. So that means doing patches, checking that it's still up, debugging core dumps.

3) Overall programming. Basically I try to make sure all our code is designed the best way it can be. So refactoring some areas, redesigning etc.

4) Generally a jack of all trades. So I support the forums/bugtracker and linux build guides, database backups that sort of thing.

What is the biggest problem for Planeshift?

I think it is the chaotic way of design and development. We try to develop master plans but they are more of guidelines for development. So much of the work is done as we think of it. And with several people doing that sometimes things can clash (code wise :))

What is your goal with Planeshift?

To be honest I don't really have a goal. This is a hobby of mine and it's something I like doing. It's like asking a coin collector or a bird watcher what their goal is. For me it's to have fun, learn, teach and meet new people.

What where the reasons for choosing the licenses?

For the source it only made sense because offering a free and free to play MMORPG would be very hard to do with a closed source system. Right now we can attract new developers because they can see we are not just somebody talking big.

Did the project profit from choosing these licenses? How?

Where potential contributors more or less willing when learning about the license?

Is that different between the groups of contributors (code, graphics, sounds)?

I will answer these 3 together because they are related. We chose a different set of licenses for the content because it really made sense to do so. There is a seperation from the code (the how) and the content (the why). We want to give everybody a chance to do their own thing using our engine but we want to protect our 'PlaneShift' brand and that means not releasing the content as GPL.

So that means that PS comes with 2 different licenses, one for the code and one for the content. In most cases this was at the insistance of the content people. Artists wanted to make sure that their work is only being used in ways that they approved. Also much of the work is uniquely defined for PlaneShift and to see that work in other games would dimish our impact.

This has turned some contributors off but has turned some on. So it's a balance between the two.

How do you plan on financing the game? (Server costs etc)

Right now we are using donations of hardware and bandwidth. It has not been a problem in the entire time I have been part of the project. When our first game server was lost we had new offers and were up and running again in a matter of days. I think we can provide good exposure to potential donators.

We've had over 150,000 people register accounts so that is no small number of people. Maybe at some point we will dive into the realm of financing though donations or by selling game related merchandise but no plans yet.

PlaneShift is owned by Atomic Blue which is the non-profit organization we set up so we would have to meet any legal rules defined by that related to money.

A.3 Glest Entwickler

Die befragten Glest Entwickler wurden über den „Credits“ Abschnitt der Webseite des Spiels herausgesucht. Die Befragung fand per E-mail und auf Englisch statt.

A.3.1 Martiño Figueroa

Who are you and what is your relation to the Glest project?

I was the orinigal creator and programmer of the project, later the other members joined to work on the other aspects of the game.

Was this your first involvement in an (open source) project? (If not, what was your first?)

Yes it is.

What where the reasons for opening up the source code?

Since it would be impossible to get any money out of it we thought that it would be cool to get the comunity involved into it.

What where the reasons for not opening up the content?

The same reasons I think.

Did the project profit from choosing these licenses? How?

No, we don't get any profit from the project, just the publicity.

What is the biggest problem for the Glest project?

From the programming point of view it is a very large system, and it needs to be engineered very carefully, also to integrate the work of all the different people is quite tough.

What is your goal with Glest?

We don't have a proper goal, just learn and make a good and popular game.

A.4 Nebula Engine Entwickler

Bei der Betrachtung der Nebula Engine gibt es zwei unterschiedliche Perspektiven. Zunächst existiert eine interne Sicht auf die Engine und die Lizenzentscheidung, sowie die Folgen daraus. Dieses Sicht wird repräsentiert durch Bernd Beyreuther, den

Managing Director der Firma Radon Labs, der mit an diesen Entscheidungen beteiligt war. Als Informationsquellen dienten hier neben dem E-mail-Kontakt auch ein persönliches Gespräch am 7. Februar 2006.

Zusätzlich existiert eine externe Sichtweise, die durch Entwickler, die an von Radon Labs unabhängigen Projekten arbeiten, repräsentiert wird. Aufgrund der weltweiten Verteilung dieser Entwickler wurde hier per E-mail kommuniziert.

A.4.1 Bernd Beyreuther

Könntest Du Dich kurz vorstellen und sagen, was Deine Aufgabe in der Nebula Entwicklung war, besonders im Zusammenhang mit der Entscheidung für Open Source?

Ich bin Bernd Beyreuther, 35 Jahre alt, habe Zeichentrick studiert, bin also ein Diplom-Animator. Ich habe an der Filmschule Babelsberg studiert. Ich habe also mit Programmierung erstmal so nichts am Hut, habe aber früher selber als Programmierer gearbeitet, bevor ich Zeichentrick studiert habe. Dieses Zeichentrick auf der einen Seite, das Programmieren auf der anderen Seite, diese Mischung war immer meine Spezialität.

Du hast das schon zu DDR Zeiten angefangen?

Richtig. Mein Starterlebnis war so: In irgendeinem Jugendclub wurde damals der Z1013[137] vorgeführt, also der Computerclub gegründet. Da war ich mit meinen Freunden und hab dann gemeint, ich werde mich mit den Dingern so lange beschäftigen, bis ich damit einen Trickfilm gemacht habe. Das war damals so mein Ding, damit kann man Bilder machen, damit beschäftige ich mich jetzt. Das hat dazu geführt, dass ich einige Jahre auch als Programmierer gearbeitet habe damals. Und dann wieder in's Studium gegangen bin und klassische Zeichentrickanimation gelernt habe, dann mich auf Computeranimation spezialisiert habe. In der Computerspieleentwicklung gehen meine ganzen Interessen zusammen: Bewegte Grafik, Welten schaffen, aber auch dieses Technische, was mir sehr liegt, was ich sehr mag.

Bei Nebula selber ist meine Rolle so: Dadurch, dass ich dieses „Mischwesen" bin, habe ich immer ein gutes Verständnis für beide Seiten gehabt, also für die Programmierung und für die Grafik. Das ist auch das was ich in die Entwicklung mit eingebracht habe. Am Anfang war es das Skripting, später war es die Beachtung, die Maya als Tool bekommen hat, die es jetzt auch in Form des Nebula Toolkits hat. Das man gute Tools hat, das ist mein wesentlicher Teil.

[137] Der Z1013 war ein Rechnerbausatz, der ab 1985 in der DDR verkauft wurde.

Die Tatsache, dass es diese völlig freie Open-Source-Lizenz ist, ist auch hauptsächlich auf meinem Mist gewachsen. Im Vorfeld haben wir da schon hin- und herüberlegt: „Machen wir eine GNU-Lizenz?“ Naja, mit einer GNU-Lizenz kann man dann ja keine kommerziellen Spiele entwickeln. „Nehmen wir eine Lizenz, wo wir Geld kriegen?“ Ich war der Meinung, am besten profitieren wir davon, dass wir es völlig frei machen.

Also habt ihr eine BSD-Lizenz gewählt?

Genau, also die TCL-Lizenz haben wir genommen.

Du warst also der Hauptverantwortliche für die Entscheidung?

Ich war nicht der allein Entscheidende, aber ein treibender Keil.

Was waren die Gründe, eine Open Source Lizenz zu verwenden?

Die erste Frage, die man sich dabei stellen muss, ist die Frage warum man überhaupt eine eigene Engine entwickelt. Diese Frage müsste man 2006 anders beantworten, als wir das 1998 gemacht haben. Wir wollten damals, 1998, ein Spiel machen, das bestimmte Eigenschaften aufweist, halt volumetrische Wolken hat, eine sehr große Sichtweite hat - Project Nomads. Und es gab damals keine fertige Engine, die so etwas gemacht hätte und dazu noch den Kriterien vom Workflow entsprach, die uns wichtig erschienen. Deswegen haben wir eine eigene Engine programmiert. Das macht natürlich jedes junge Team, seine eigene Engine zu programmieren, aber das ist an sich Unsinn. Wir haben es aber damals gemacht, aus bestimmten Gründen.

Wenn man dann aber anfängt, eigene Technologie zu entwickeln, hat man verschiedene Möglichkeiten: Man kann sie verkaufen, geheimhalten oder verschenken. Wenn man sie verkauft, kommt man ganz schnell in die Situation, dass man als Firma ein Produkt anbietet, was Support braucht, was Pflege braucht, wo Gewährleistungsrecht dranhängt etc. Und das wollten wir damals nicht. Das war auch eine Überlegung, als Nebula im Jahr 2000 dann publik wurde. In diesem Jahr hat jede deutsche Firma ihre eigene Engine verkaufen wollen, zum Beispiel Massive und Phenomedia usw. Die ganzen Spieleentwickler haben Engine-zentriert gedacht und wollten ihre eigene Engine verkaufen.

Das lag vermutlich am Erfolg von id-Soft?

Ja, das war das Modell, von dem man dachte: „Lass uns das mal kopieren.“ Jedenfalls sind die alle gescheitert. Wir sind heute die erfolgreichste Engine am deutschen Markt, also die mit den meisten kommerziellen Titeln, weil wir die Engine eben verschenkt haben.

Also verkaufen wollten wir damals nicht, wir wollten ja Spiele entwickeln und waren viel zu klein, um dann noch Supportfirma zu sein. Heute, wo wir das

Toolkit verkaufen, selbst mit unseren 15 Programmierern jetzt, ist es schwierig einen professionellen Level zu halten, der heutzutage gefordert ist bei Software; also ständig, regelmäßig Updates zu bringen, automatisch über das Web, einen bestimmten Dokumentationsgrad zu halten, Weiterentwicklung, also die neusten Features zu bringen, was wir beim Toolkit jetzt machen, als kommerzieller Anbieter. Das hätten wir damals nie so leisten können. Wollten wir auch nicht, denn wir haben gesagt: „Wir sind Spieleentwickler, die Engine ist ein Abfallprodukt."

Die zweite Möglichkeit, die Engine nicht zu verschenken, also geheim zu halten, ist die schlechteste. Weil dann niemand einen Nutzen davon hat. Und es ist eine Illusion, wenn man denkt, man behält sich einen Vorteil damit vor. In dem Moment, in dem man etwas veröffentlicht, ist man immer einen Schritt weiter als derjenige, der das dann liest. Wenn jemand ein Buch schreibt, ist er Meielen weiter als derjenige, der das Buch dann liest. Und die selbe Situation ist es auch beim Sourcecode. Wir haben einen Haufen Vorteile von diesen Open-Source-Sachen gehabt. Das haben wir nicht alles im Vorfeld gesehen. Wie schon gesagt, hat man keinen Nachteil davon, dass man Wissen weitergibt.

Die Vorteile sind in verschiedenen Bereichen zu sehen. Zum einen gibt's da draußen eine Community, die macht Bugfixing und Testing. Nebula war immer sehr stabil. Heute hatte ich gerade eine Computerbild Spiele, wo ein Nebula Spiel drin war. Die haben ja immer eine Hardwaretabelle, wo alles getestet wird und da steht bei uns eine „1,0", weil das Ding auf allen Systemen, für die es spezifiziert ist, einwandfrei läuft.

Weil es auch auf mehr Systemen getestet wird, als ihr selber besitzt?

Richtig, das hängt damit zusammen. Wobei dieser ganze Rückfluß der Open-Source-Community gar nicht so groß ist, wie man denken sollte. Also wir haben schon ein paar hundert Entwickler draußen, aber das ist nicht so groß.

Das zweite wichtige Kriterium ist das socialising, also „Netzwerkeln", dass also über die Tatsache, dass man es verschenkt, natürlich viele Connections entstehen. Davon haben wir sehr profitiert. Wir haben dadurch sogar neue Aufträge aquirieren können, weil zum Beispiel eine befreundete Firma, die auch Nebula verwendet, bestimmte Aufträge nicht mehr abwickeln konnte und sie an uns weitergegeben hat, einfach nur aus Freundlichkeit.

Ihr habt also finanzielle Vorteile davon gehabt, dass ihr die Engine verschenkt?

Richtig. Ausserdem ist es natürlich Werbung, über Open Source wird, vor allem in Deutschland, gerne berichtet. Man hat da grundsätzlich eine positive Berichterstattung drüber. Wenn man sagt: „Ich bin ein Open-Source-Projekt",

dann gibt es eine sehr positive Einstellung in Europa dazu. In Amerika ist das durchaus anders.

Das entscheidende Kriterium ist aber, dass wenn Code offen gemacht wird, dann ist der per Definition sauberer als Code, der nicht offen ist. Die Tatsache, dass wir fast die Hälfte unserer Codebase, also mit Mangalore ist ja der ganze Bereich „Game Framework“ dazugekommen, offenlegen, führt dazu, dass wir sehr cleanen Code haben, weil die Programmierer wissen, da werden noch ein paar tausend andere Leute reingucken. Und das ist für mich heute das wichtigste Kriterium, große Codestücke in der Firma offen zu haben.

Du meintest vorhin, dass von den externen Entwicklern relativ wenig zurück gekommen ist. Externe Entwickler haben das von der anderen Seite her als Problem gesehen, durch eine bestehende Kommunikationsbarriere. Die Mailingliste wird wenig von Radon Labs selbst benutzt, externe fühlen sich, als würden sie gegen eine Wand schreiben.

Gut, die Zusammenarbeit mit der Community ist noch einmal ein Thema für sich. Das ist in vielen Dingen von unserer Seite her nicht optimal gelaufen, weil wir die Philosophie hatten, dass die Engine erstmal ein Abfallprodukt ist. Wir sind nicht eine Gruppe von Studenten, die eine Hobbyengine basteln und dann abends immer die Foren pflegen. Sondern, wir haben einen Code, der bei uns ständig in Benutzung ist und in ein paar dutzend Firmen weltweit auch. Wo harte Deadlines gehalten werden müssen, wo immer zu wenig Leute da sind, wo immer zu wenig Zeit da ist, wo Überstunden gemacht werden müssen. So dass wir zu keinem Zeitpunkt in den sieben Jahren, die Nebula jetzt draußen ist, eine Webseite hatten, wie zum Beispiel OGRE[138], wo wir uns so richtig der Community widmen konnten.

Wir haben die vergangenen drei Jahre den Zustand gehabt, dass ein paar Leute, also Bruce Mitchener und Leif Garland, die Open-Source-Community eigentlich weitgehend eigenständig betreut haben. Durch den Workload bei uns oder durch Projekte sind dann Situationen entstanden, dass wir ein Jahr lang nicht mehr synchronisiert haben. Dabei sind wir dann in großen Schritten in eine ganz andere Richtung gegangen als die Community. Die Community macht ganz oft so experimentelle Sachen, wie mal eine neue Skriptsprache einzubauen. Oder sie bearbeitet halt Sachen, die ganz offensichtlich fehlen, wie Game Framework oder User Interface. So lange von uns dann nicht synchronisiert wird, gehen die Entwicklungen auseinander.

Das ist auch schwer, das zusammenzubringen. Denn wir machen das ja nicht als Hauptprodukt, bei dem wir die Community mit pflegen, es ist erstmal ein

[138]Eine freie 3D Engine, erhältlich unter der URL http://ogre3d.org/ (10.02.2006).

Abfallprodukt. Es ist natürlich schlecht, dass sich die Community dann – berechtigterweise – beklagt.

Es ist aber so, dass wir jetzt kommerzielle Produkte auf Nebula aufsetzen, das Toolkit war der erste Schritt und es kommt bald ein Toolkit 2. Wir setzen ausserdem Portierungen darauf auf. Die PC Version bleibt weiter Open Source, aber wir bringen XBOX und Nintendo DS als kommerzielle Portierungen. Das ist für Startup-Teams und kleine Firmen ein attraktives Modell. Sie bauen sich ihr Spiel preiswert auf dem PC und können dann zum Publisher gehen und sagen: „Das setzt auf die Nebula-Engine auf, für welche Plattform möchtet ihr es denn gerne?“ Wir werden diese Portierungen natürlich günstiger anbieten, als es zum Beispiel Renderware und andere große Anbieter machen.

Wir haben in der Beschäftigung mit diesen kommerziellen Modellen gemerkt, dass wir das nicht mehr so machen können, dass wir Nebula als Abfallprodukt behandeln. Denn schon jutzt ist es ein eigenes Produkt, was Geld abwirft und was natürlich eine dementsprechende Aufmerksamkeit erfordern muss. Und die Community gehört mit dazu zu diesem Produkt. Sie ist sozusagen der Köder. Und wir sehen schon, dass da ganz viel Verbesserungsbedarf in dem Bereich ist, zum Beispiel Dokumentation, Zusammenarbeit mit der Community, Information.

Wie funktioniert der Code-Austausch zwischen eurer internen und der offenen Nebula Variante?

Wir haben das jahrelang so gemacht, dass wir beide Varianten regelmäßig synchronisiert haben. Anfangs hat das unserer Lead-Programmer selber gemacht, dann ein anderer Mitarbeiter. Inzwischen sind bei uns zwei oder drei Leute damit beschäftigt das zu pflegen.

Bei manchen Entwicklungen müssen wir In-House bestimmte Desgin-Entscheidungen treffen, die uns von der offenen Variante wegführen. Ein Beispiel dafür ist das Game-Framework, für das wir bestimmte Änderungen einführen müssen, obwohl wir wissen, dass die Community in diesem Bereich aus bestimmten Gründen einen anderen Weg geht. Wir stellen dann unsere Version vor und sagen, dass die Community das übernehmen kann. Andersherum gibt es draußen auch immer mal Sachen, bei denen wir einen guten Ansatz sehen, den wir dann weiter entwickeln.

Es gab auch Anfragen von der Community, welche Features wir brauchen oder planen, damit die Arbeit zwischen interner und externer Variante besser synchronisiert werden kann. Das Problem dabei ist, dass wir als Firma zum Teil sehr kurzfristig auf bestimmte Kundenanfragen reagieren müssen. Es wird also aus wirtschaftlichen Gründen einiges anders gemacht, als vielleicht sonst. Dies liegt auch an den Kleinstprojekten, die wir zum Beispiel im letzten Jahr hatten,

bei einem mehrjährigen Projekt ist das anders. Größere Blöcke von externem Code in die interne Version einzubauen, ist bei solchen kurzfristigen Projekten schon aus Sicherheitsgründen nicht möglich.

Wie siehst Du die Chancen für Open Source im Computerspiele-Bereich allgemein?

Ich denke, dass es langfristig gar keine Alternative zu Open Source geben kann. Wir sehen das besonders im Bereich Online-Spiele, also Massive Multiplayer. Wir wollen uns selber in diesen Bereich reinbewegen. Es gibt dort zwar viele Firmen, die eigene Spiele anbieten, aber es gibt keinen Anbieter, bei dem man eine Massive-Multiplayer-Engine kaufen kann. Auch im Open-Source-Bereich haben wir da nichts verwendbares gefunden.

Für uns ergab sich die Frage, warum es nicht, wie es einen Kolab Server oder einen Bugzilla gibt auch eine Massive Multiplayer Engine gibt, mit der man zumindest starten kann. Das gilt natürlich genauso für andere Bereiche. In jedem technologischen Bereich ist der Bedarf vorhanden, dass es da Open-Source-Versionen gibt. Es existiert auch der Bedarf für kommerzielle Varianten, aber es ist immer auch der Raum für eine Open-Source-Variante vorhanden und es gibt immer auch den Bereich um so eine Open Source Lösung herum, von dem kommerzielle Firmen leben können, sei es im Support oder in der Entwicklung von zusätzlichen Tools.

Zum Beispiel gibt es die Open-Source-Physik-Engine ODE. Hierfür fehlen Tools, es fehlt ein Entwickler, der sich da sozusagen andockt und das in eine richtige Richtung bringt. Ich denke, dort ist ein Bedarf, der aber nicht gleich offen auf der Hand liegt.

Ich denke, es ist auch vielfach eine Kopfsache, dass man einfach Dinge festhalten will und sie nicht rausgeben möchte. Aber ich denke auch, dass in diesem Bereich noch einiges passieren wird.

A.4.2 Megan Fox

What is the project you are using the Nebula engine in and what is your role in that project?

We're using Nebula Device 2 in Elium (http://www.elium.tk), and I'm the lead developer / co-designer / co-manager on the project.

What where the reasons for choosing Nebula for that project?

It's one of the best offerings out there in terms of open source engines. It doesn't have the community or toolset that other opensource engines do, but the engine is actually commercially viable.

What it lacks is worth adding, while other open source engines typically have too much crap piled on top to make it worth using at all.

Did you have any experiences with Open Source software before?

Yes. I used Crystal Space before switching to Nebula Device, I'll be using OpenSteer in our AI, etc.

Did your project profit from the license of Nebula?

Of course. It means we can actually write a commercially-viable game, even if we don't actually intend to sell anything for the moment.
Most other open source projects like to infect their projects with a viral license that serves no other purpose than to help other open-source developers.

Where you able to get any changes back into the main Nebula version?

A few minor ones, yes. Nebula lacks the management necessary to really drive the development, and so patches tend to stagnate rather than being encouraged. It's a great engine, but with a very small community (and a less than ideal manager).

What do you think is the biggest problem that users of Nebula face?

Poor management that drives the community in the wrong direction. Secondarily, Nebula is not a "off the shelf" engine... it has great potential, but it requires a lot of personal work to get it to do everything you'll need it to do.

Nebula Device is, rather than a game engine, a good framework on which to build your own game engine. If there were an actually-decent open source game engine, the usefulness of Nebula Device would drop, but Crystal Space is still "it" in terms of finished engines – and Crystal Space is awful.

Have you used any of the "Contrib" projects that expand Nebula?

Only one, OpenAL, and it was awfully and broken.

Do you use any of Radon Labs commercial tools?

Nope – no money for it, and the tools don't impress me anyways.

A.4.3 M. A. Garcias

What is the project you are using the Nebula engine in and what is your role in that project?

The game project is called Renaissance (http://www.tragnarion.com/projects.php) and we're developing it along with a game development set of tools called Conjurer. I'm the lead graphics programmer of the project.

What where the reasons for choosing Nebula for that project?

We at tragnarion had the goal of having our own development environment, so we needed a platform flexible enough. Starting our own platform from scratch was too ambitious at that moment, and we looked for an open source project with enough flexibility to develop a complex system on top. At that moment, nebula2 was still in its first stage, and lacked many features, but it had a complete object system (including scripting), an extensible way of managing new modules that allowed creating many new game structures, and most importantly, a working, last generation graphics renderer. All of it could be used as a base for future developments (as we have done, extensively). The fact that the first nebula engine had been used successfully in a commercial game was also an implicit guarantee of good choice.
Also, the nebula2 open source community was very active at that moment, there were (and there still are) some really clever people there, and Radon Labs was often available for cooperation. And the free license was an important reason for the decision too.

Did you have any experiences with Open Source software before?

Not me personally, but other members of my team did. web-based development, mainly. And we're all for open source, free software.

Did your project profit from the license of Nebula?

Yes, as I said before, it was really nice to be able to choose and experiment with nebula for free. Of course, it also meant that we couldn't get any support or guarantee from the original author, which also worked in our favor, because we wanted to develop our engine and our tools based on inner requirements.
We plan to give back to the nebula open source community all code we've developed (when it is more stable and documented), so the fact of being it LGPL was just a plus, not a key point. We would have released our code anyway.

Where you able to get any changes back into the main Nebula version?

At the moment, we've released many improvements and fixes, but the modules that we have completely developed in-house are still pending (they would be useless unless they come with some support and/or documentation). The nebula open source community maintainers has been extremely cooperative in that task. Radon Labs, on the the other hand, is a little behind in integrating open

source fixes and improvements into its own code base (due to schedule reasons, we have been said).

What do you think is the biggest problem that users of Nebula face?

Development of features isn't organized nor steady. Everyone develops what they need at the moment for their own projects, and so there are lacking features that everyone hopes someone else implements. There is not much activity in the discuss list, nor the IRC channel. Documentation is a good start, but it lacks many important tutorials, code samples, and most importantly, design docs describing the intention and requirements upon which existing code was based.
Also, the communication with Radon Labs is very limited: "official" radon labs code is released once in a while, and is rapidly integrated with the main open source code, but there is little or no guarantee (nor support) for the changes introduced, so users need to work on the changes themselves, and code merges are not easy (I have done one such merge personally and it was a real hell, with many unexpected side effects; we're not planning to do another one until we release our own code as open source).
Finally, the learning curve for nebula users is not easy: it gets some time to get accustomed to the "nebula way of things", and available docs and tutorials don't help much. And some very important modules not being still available (like an entity framework, opengl renderer, or support for linux makefiles) isn't helping much either.

Have you used any of the "Contrib" projects that expand Nebula?

Only as a basis, but not completely. The terrain renderer and spatial database have been useful in developing our own, but once we had the basics, we branched away to fit our own needs. The problems described above multiply by 10 for the contrib modules- many of them are so outdated that are in fact useless. And most of their authors don't seem to be at all available for improvements or fixes.

Do you use any of Radon Labs commercial tools?

No. In that case, we needed a toolkit completely in-house developed, to fit the specific needs of the art team, and of the project itself. And the existing tools from Radon Labs weren't exactly what we needed - too limited. We needed a lot of freedom in our project, so when we needed something specific, we'd prefer doing it ourselves. And we did.

B Literaturverzeichnis

Ardal, A. (2000). Open Source - das Beispiel Linux, URL: http://ig.cs.tu-berlin.de/oldstatic/da/059/ardal-opensource.pdf (17.10.2005).

Bae, S. H. (2002). *Video Game Illusions*, Master's thesis, Parsons School of Design.

Barr, J. (2005). The paradox of free/open source project management, URL: http://software.newsforge.com/article.pl?sid=05/02/10/1539242 (17.10.2005).

Behrmann, M., Oehler, C., Winter, S., Dlugeizcyk, T., Reichart, S., Rode, M., Ullmann, T., Blechschmidt, A., v. Maydell, A., Friedmann, T., Putzki, T., Riegler, H. und Krell, A. L. P. C. (2005). Antworten zu 30 Fragen zur vorgeschlagenen Förderung von Computerspielen, URL: http://www.game-bundesverband.de/downloads/foerderung.zip (22.03.2006). G.A.M.E. – Bundesverband der Entwickler von Computerspielen.

Bessen, J. (2001). Open Source Software: Free Provision of Complex Public Goods, URL: http://www.researchoninnovation.org/opensrc.pdf (17.10.2005).

Boehm, B. W. (2002). Software engineering economics, *Software pioneers: contributions to software engineering*, Springer-Verlag New York, Inc., New York, NY, USA, pp. 641–686.

Brahe, T. (2005). The Mainstream is coming!, *The Escapist* **1**: 4.

Brooks, F. P. (1975). *The mythical man-month: essays on software engineering*, Addison-Wesley Pub. Co.

Brunvard, E. (1996). The Heroic Hacker: Legends of the Computer Age, URL: http://www.cs.utah.edu/~elb/folklore/afs-paper/afs-paper.html (17.10.2005).

Buro, M. und Furtak, T. (2003). RTS Games as Test-Bed for Real-Time AI Research, URL: http://www.cs.ualberta.ca/~mburo/ps/ORTS-JCIS03.pdf (17.10.2005). Submitted to Information Sciences.

Byfield, B. (2005). Seven tips to help FOSS companies succeed, URL: http://business.newsforge.com/article.pl?sid=05/03/01/2159227 (17.10.2005).

Capiluppi, A., Lago, P. und Morisio, M. (2003). Characteristics of Open Source Projects, *CSMR '03: Proceedings of the Seventh European Conference on Software Maintenance and Reengineering*, IEEE Computer Society, Washington, DC, USA, p. 317.

Cornett, S. (2004). The Usability of Massive Multiplayer Online Roleplaying Games: Desining for New Users, *CHI* **6**: 703–710.

Crowston, K. und Scozzi, B. (2002). Open source software projects as virtual organizations: competency rallying for software development, *IEE Proceedings Software*, Vol. 149.

Cummins, N. (2002). Integrating E-Commerce and Games, *Personal Ubiquitous Comput.* **6**(5-6): 362–370.

Cypra, O. (2005). Warum spielen Menschen in virtuellen Welten? – Eine empirische Untersuchung zu Online-Rollenspielen und ihren Nutzern, Diplomarbeit an der Johannes Gutenberg Universität Mainz.

Dammertz, H. (2002). Spiele unter Linux (SDL), URL: http://medien.informatik.uni-ulm.de/lehre/courses/ss02/linux/16_proseminar_linux_260602.pdf (17.10.2005).

DeMaria, R. (2005). How BioWare Makes Game Communities Work, URL: http://www.gamasutra.com/features/20051201/demaria_01.shtml (30.12.2005).

Dempsey, B., Weiss, D., Jones, P. und Greenberg, J. (1999). A Quantitative Profile of a Community of Open Source Linux Developers, *Technical Report TR-1999-05*, School of Information and Library Science University of North Carolina at Chapel Hill.

Ettrich, M. (2004). Koordination und Kommunikation in Open-Source-Projekten, Open Source Jahrbuch 2004.

Gabriel, R. P. und Goldman, R. (2003). Open Source: Beyond the Fairy Tales, URL: http://dess.gl.free.fr/2003/SLIDES/OPEN/Articles/e-zine-cgey.040105.pdf (17.10.2005).

Gallagher, S. und Park, S. H. (2002). Innovation and Competition in Standard-Based Industries: A Historical Analysis of the U.S. Home Video Game Market, *IEEE Transactions on Engineering Management* **49**(1): 67–82.

Gehring, R. A. (2001). „Virtuelles“ Eigentum?, Referat auf der Konferenz „Rechts-Links“ der Vereinigung Demokratischer Juristinnen und Juristen.

Geitgey, A. (2004). Where are the Good Open Source Games?, URL: http://www.osnews.com/story.php?news_id=8146\&page=1 (17.10.2005).

Giusti, A. (2006). Open Source Software and games: current trends and some gems, erscheint 2006 auf Newsforge.

Graetz, J. M. (1981). The origin of spacewar, *Creative Computing* **1**: 60.

Grassmuck, V. (2000). Freie Software – Geschichte, Dynamiken und gesellschaftliche Bezüge, URL: http://mikro.org/Events/OS/text/freie-sw.pdf (28.03.2006).

Guettler, C. und Johansson, T. D. (2003). Spatial principles of level-design in multiplayer first-person shooters, *NETGAMES '03: Proceedings of the 2nd workshop on Network and system support for games*, ACM Press, New York, NY, USA, pp. 158–170.

Guins, R. (2004). 'Intruder Alert! Intruder Alert!' Video Games in Space, *journal of visual culture* **3**: 195–211.

Gumhold, M. und Weber, M. (2004). Motivating CS students with game programming, URL: http://medien.informatik.uni-ulm.de/~mag/ICNEE04/ICNEE04.pdf.

Hardin, G. (1968). The Tragedy of the Commons, *Science* **162**: 1243–1248.

Hargreaves, S. (1999). Playing the Open Source Game, URL: http://www.talula.demon.co.uk/games.html.

Hars, A. und Ou, S. (2001). Working for Free? - Motivations of Participating in Open Source Projects, *Proceedings of the 34th Hawaii International Conference on System Sciences*.

Hawkins, R. E. (2004). The economics of open source software for a competitive firm, *Netnomics* **6**(2): 103–117.

Herz, J. C. (1997). *Joystick Nation*, Little Brown & Co. ISBN: 0316360074.

Holmquist, L. E., Björk, S., Falk, J., Jaksetic, P., Ljungstrand, P. und Redström, J. (1999). What about Fun and Games?, URL: http://www.viktoria.informatik.gu.se/groups/play/.

Houk, P. A. (2004). A Strategic Game Playing Agent for FreeCiv, *Technical Report NWU-CS-04-29*, Northwestern University - Computer Science Department.

Husemann, P., Martin, M., Ruhnke, I. und Sudek, H. (2002). Freie Software Kultur und Technik, URL: http://pingus.seul.org/~grumbel/tmp/FreieSoftwareKulturUndTechnik.pdf.

Ishii, K. (1995). Regularien im Internet, Diplomarbeit TU-Berlin.

Kelty, C. M. (2001). Free Software / Free Science, *First Monday* **6**(12): 0.

Khalak, A. (2000). Economic model for impact of open source software, http://opensource.mit.edu/papers/osseconomics.pdf.

Kogut, B. und Turcanu, A. (1999). The Emergence of E-Innovation: Insights from Open Source Development, URL: http://jonescenter.wharton.upenn.edu/papers/opensource.pdf (14.07.2004).

Laird, J. E. und van Lent, M. (1999). Developing an Artificial Intelligence Engine, *Proceedings of the Game Developers Conference*.

Laird, J. E. und van Lent, M. (2001). Human-Level AI's Killer Application: Interactive Computer Games, *AI Magazine* **22**: 0.

Laitinen, S. (2005). Better Games Through Usability Evaluation and Testing, URL: http://www.gamasutra.com/features/20050623/laitinen_01.shtml (23.06.2005).

Lakhani, K. R. und von Hippel, E. (2002). How open source software works: "free" user-to-user assistance, *Research Policy* **32**: 923–943.

Lamnek, S. (1995). *Qualitative Sozialforschung, Band 2 Methoden und Techniken*, 3. edn, Psychologie Verlags Union, Weinheim.

Leavitt, N. (2003). Will Wireless Gaming Be a Winner?, *Computer* **36**: 24–27.

Leiteritz, R. (2002). Der kommerzielle Einsatz von Open Source Software und kommerzielle Open Source-Geschäftsmodelle, URL: http://www.leiteritz.com/docs/Kommerzieller_Einsatz_von_OSS_und_OSS_Geschaeftsmodelle_2003.pdf (17.10.2005).

Levy, S. (1984). *Hackers - Heroes of the computer revolution*, Dell Publishing.

Lichtl, B. und Wurzer, G. (2001). Software Engineering in Games, Seminar Lecture, Institute of Computer Graphics, Technical University Vienna.

MacCormack, A. (2001). How Internet Companies Build Software, *MIT Sloan Management Review* **Winter**: 75–84.

Macedonia, M. (2001). Will Linux Be Computer Games' Dark Horse OS?, *Computer* **34**(12): 161–162.

Martin, A. (2005). Soapbox: Pirated P2P Games: Free Electronic Distribution for Independent Studios, URL: http://www.gamasautra.com/features/20051005/martin_01.shtml (02.11.2005).

Myers, B. A. (1996). A Brief History of Human Computer Interaction Technology, URL: http://www.comp.nus.edu.sg/~cs3283/ftp/BriefHistoryOfHCI.pdf (17.10.2005).

Myers, D. (n.d.). The attack of the backstories (and why they won't win), http://www.loyno.edu/~dmyers/F99%2520classes/AttackOfTheBackstories.pdf.

Möller, E. (2006). Freiheit mit Fallstricken: Die Creative Commons NC-Lizenzen und ihre Folgen, erscheint im Open Source Jahrbuch 2006.

Nüttgens, M. und Tesei, E. (2000a). Open Source - Konzept, Communities und Institutionen, *Veröffentlichungen des Instituts für Wirtschaftsinformatik* **156**: 1.

Nüttgens, M. und Tesei, E. (2000b). Open Source - Marktmodelle und Netzwerke, *Veröffentlichungen des Instituts für Wirtschaftsinformatik* **158**: 1.

Nüttgens, M. und Tesei, E. (2000c). Open Source - Produktion, Organisation und Lizenzen, *Veröffentlichungen des Instituts für Wirtschaftsinformatik* **157**: 1.

Osterloh, M., Rota, S. und Kuster, B. (2002). Die kommerzielle Nutzung von Open Source Software: Der Einfluss von sozialem Kapital, *Zeitschrift Führung und Organisation (ZFO)* **4**: 211–217.

Osterloh, M., Rota, S. und Kuster, B. (2003). Open Source Software Production: Climbing on the Shoulders of Giants, URL: http://opensource.mit.edu/papers/osterlohrotakuster.pdf.

Osterloh, M., Rota, S. und Kuster, B. (2004). *Open Source Jahrbuch*, Lehmanns Media - LOB.de, chapter Open-Source-Softwareproduktion: Ein neues Innvoationsmodell?, pp. 121–133.

Pias, C. (2003). Appropriation Art & Games. Spiele der Verschwendung und der Langeweile, URL: http://www.uni-essen.de/~bj0063/texte/spielekunst.pdf (17.10.2005).

Poblocki, K. (2002). Becoming-state The bio-cultural imperialism of Sid Meier's Civilization, *Focaal - European Journal of Anthropology* **39**: 163–177.

Pratchett, R. (2005). Gamers in the UK – Digital play, digital lifestyles., BBC New Media & Technology: Creative Research & Development.

Raymond, E. S. (1999). *The Cathedral & The Bazaar*, O'Reilly.

Richards, M. (2003). Multiplayer Games: A Spectator's View, Thesis Proposal Document.

Robles, G., Scheider, H., Tretkowski, I. und Weber, N. (2001). Who is doing it?, URL: http://widi.berlios.de/paper/study.pdf.

Robles, G. und Gonzalez-Barahona, J. M. (2004). Toy Story: an analysis of the evolution of Debian GNU/Linux, URL: http://libresoft.dat.escet.urjc.es/debian-counting/.

Rocca, J. D. (2005). Invasion of the IP Snatchers: Exploring Licensed versus Original Games, URL: http://www.igda.org/articles/dellarocca_ipsnatchers.php (19.10.2005).

Rouse, R. (1998). Do Computer Games Need to be 3D?, *Computer Graphics* **32**(2): 0.

Sarvas, R., Turpeinen, M., Viratnen, P., Hietanen, H. und Herrera, F. (2005). Legal and Organizational Issues in Collaborative User-Created Content, *Proceedings of DIGRA 2005 Conference: Changing Views Worlds in Play.*

Scheiblauer, T. (2004). Anwendung von Game Engines für kollaborative virtuelle Umgebungen in der Architektur, Diplomarbeit an der Technischen Universität Wien.

Selig, C. (2001). Freie Software und Linux - Geschichte, Philosophie, Technologie, Wirtschaft und Gesellschaft, URL: http://www.bnv-bamberg.de/home/ba1005/pics/2001-schm.pdf.

Singleton, S. (2003). „FreeCiv" and its Discontents: Policy Lessons from Open Source Games: A Case Study, URL: http://www.cei.org/pdf/3755.pdf (17.10.2005).

Stabell, B. und Schouten, K. R. (2001). The Story of XPilot, *ACM Crossroads* **3**(2): 0.

Stahl, M. und Skibak, S. (2002). Spiele-KI: Überblick, URL: http://medien.informatik.uni-ulm.de/lehre/courses/ws0304/odc/Berichte/KI-StateOfTheArt.pdf (26.10.2005).

Tan, J., Beers, C., Gupta, R. und Biswas, G. (2005). Computer Games as Intelligent Learning Environments: A River Ecosystem Adventure, *Artifical Intelligence in Education*, pp. 646–653.

Thawonmas, R., Ho, J.-Y. und Matsumoto, Y. (2003). Identification of Player Types in Massively Multiplayer Online Games, *Proceedings of the 34th Annual Conference of the International Simulation and Gaming Association (ISAGA).*

Tinney, W. (2005). Indie Postmortem: Large Animal's RocketBowl, URL: http://www.gamasutra.com/features/20050624/tinney_01.shtml.

Torvalds, L. (2001). *Just for fun*, Carl Hanser Verlag München.

Tveit, A., Rein, O., Iversen, J. und Matskin, M. (2003). Scalable Agent-Based Simulation of Players in Massively Multiplayer Online Games, *Proceedings of the 8th Scandinavian Conference on Artificial Intelligence*.

Vogel, H. L. (2001). *Entertainment Industry Economics*, Cambridge University Press.

Vollmer, D. R. (2003). Does it have to be 3D? Überlegungen zur Beziehung von 3D-Hardware, -Software und User, *playability* **1**: 0.

von Hippel, E. (2001). Learning from Open-Source Software, *MIT Sloan Management Review* **42**(4): 82–86.

von Krogh, G. (2003). Open-Source Software Development, *MIT Sloan Management Review* **44**(3): 14–18.

Walsdorfer, K. (2003). Urheberschutz im Softwaremarkt, Universität Freiburg.

Waugh, E.-J. R. (2005). Worlds Are Colliding!: The Convergence of Film and Games, http://www.gamasutra.com/features/20051212/waugh_01.shtml (07.03.2006).

Weckenmann, A. (2003). Patentierung von Software? - Die „one-click-Bestellung" von Amazon.com, URL: http://www.vwl.uni-freiburg.de/fakultaet/wi/ss03/folien/thema4.pdf (20.10.2005).

Wehn, K. (2004). Machinima - Was Ego-Shooter und Puppentheater gemeinsam haben, URL: http://www.telepolis.de/deutsch/special/game/17818/1.html (20.10.2005).

Wen, H. (2004). Stratagus: Open Source Strategy Games, URL: http://www.linuxdevcenter.com/pub/a/linux/2004/07/15/stratagus.html (17.10.2005).

Wong, A. (2000). Analyzing and Reengineering Wolfenstein3D, Project Report, URL: http://www.cs.ualberta.ca/~kenw/courses/2003/cmput664/project/examples/andy.pdf (08.11.2005).

Woodcock, B. (2005). An Analysis of MMOG Subscription Growth – Version 18.0, URL: http://www.mmogchart.com (13.03.2006).

Ye, Y. und Kishida, K. (2003). Toward an Understanding of the Motivation of Open Source Software Developers, *Proceedings of 2003 International Conference on Software Engineering (ICSE2003)*.

Young, R. und Rohm, W. G. (2000). *Der Red Hat Coup*, MITP-Verlag GmbH, Bonn.

Zappe, U. (2004). Vom spielerischen Ernst des Programmierens, Open-Source-Jahrbuch 2004.

Zhao, L. und Elbaum, S. (2003). Quality assurance under the open source development model, *The Journal of Systems and Software* **66**: 65–75.

Zona, I. (2002). State of Massive Multiplayer Online Games 2002: A New World in Electronic Gaming, URL: http://www.zona.net/news/2002mmogreport.html (11.10.2005).