Diplomarbeit

Herausforderungen bei der Entwicklung browserbasierter Online-Multiplayer-Spiele

ausgeführt zum Zweck der Erlangung des akademischen Grades eines
"*Diplom-Ingenieurs für technisch-wissenschaftliche Berufe*"
am Masterstudiengang Telekommunikation und Medien
der Fachhochschule St. Pölten

ausgeführt von
Manuel Fallmann, B.Sc.
tm071010

unter der Erstbetreuung von
Dipl. Ing. (FH) Fritz Grabo

Zweitbegutachtung von
Dipl. Ing. Grischa Schmiedl

Ort, Datum Unterschrift

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.
- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Diese Arbeit stimmt mit der von den Begutachtern beurteilten Arbeit überein.

Ort, Datum *Unterschrift*

Kurzfassung

Computerspiele sowie das Internet haben über die letzten Jahrzehnte kulturell an Einfluss gewonnen und konnten sich Dank stetigem technischen Fortschritt zunehmend verbreiten und weiterentwickeln. In Folge dessen stellen Online-Multiplayer-Spiele inzwischen eine populäre Freizeitbeschäftigung dar. Bei diesen handelt es sich in der Regel um Desktop-Applikationen. In den letzten Jahren gewann jedoch der Browser als Plattform an Bedeutung und browserbasierte Applikationen sind mittlerweile eine interessante Alternative zu herkömmlicher Software geworden. So ist es naheliegend, diese Plattform auch für Online-Multiplayer-Spiele zu nutzen.

Ziel der Arbeit ist es, die Herausforderungen mit denen man bei der Umsetzung eines browserbasierten Online-Multiplayer-Spiels konfrontiert wird, zu beleuchten und Lösungen dafür zu finden. Der erste Teil dieser Arbeit dient der Einführung in die Materie und erläutert die Entstehungsgeschichte von den Anfängen der Multiplayer-Spiele, über die zunehmende Vernetzung, bis hin zu den Vor- und Nachteilen eines browserbasierten Ansatzes. Danach folgt eine Betrachtung der Herausforderungen, deren Ursachen sowie bewährter Lösungsansätze, wie sie bereits bei desktopbasierten Multiplayer-Spielen angewandt werden. Mit diesem Wissen als Basis wird ein Blick auf die mögliche browserbasierte Realisierung dieser Verfahren geworfen. Die gewonnen Erkenntnisse werden abschließend in einer Proof of Concept Umsetzung mittels Flash, PHP und JSON exemplarisch realisiert. Diese zeigt wie ein browserbasiertes Online-Multiplayer-Spiel in der Praxis erfolgreich umgesetzt werden kann.

Abstract

Video games as well as the internet gained a lot of cultural influence during the last decades and were able to spread and develop further, thanks to technological progress. As a result, online multiplayer games have become a popular free time activity. These games are commonly implemented as desktop applications. During the course of the last years however, web browsers gained importance as platform and browser-based applications have become an interesting alternative to regular software. Therefore it seems natural to use this platform for online multiplayer games as well.

The main objective of this paper is to observe the challenges that come with realizing a browser-based online multiplayer game and to find fitting solutions. The purpose of the first part of this paper is to give an introduction on the matter. Starting with the history of early multiplayer games, this section explains how the first networked multiplayer games emerged and leads to a review of the benefits and drawbacks of browser-based solutions. A study of the confronting challenges follows, as well as a look at their causes and proven solutions as used by established multiplayer games. This knowledge is then used for the conception of feasible browser-based scenarios. The gained insight on the matter is finally employed for a proof of concept project using Flash, PHP and JSON. This project shows the successful realization of a browser-based online multiplayer game.

Danksagung

Mein Dank geht an meinen Diplomarbeitsbetreuer Dipl.-Ing. (FH) Fritz Grabo, welcher mir mit konstruktiver Kritik und Ratschlägen beim Schreiben dieser Arbeit zur Seite stand. An meine Freunde, welche den Arbeitsvorgang zwar nicht unbedingt beschleunigt, aber zumindest den Zeitraum in dem diese Arbeit entstand angenehmer gestaltet haben. Die Community meiner Website MINDistortion.tv, von denen mich manche Mitglieder seit Jahren begleiten. Und natürlich meinen Eltern, welche mich mein Leben lang unterstützt haben und von denen ich weiß, dass sie immer für mich da sein werden.

St. Pölten, Österreich — Manuel Fallmann

2. März 2009

Inhaltsverzeichnis

Kapitel 1

Einleitung

Videospiele sowie das Internet sind beides im Vergleich zu Print und Fernsehen relativ junge Medien, die erst im Laufe der letzten Jahrzehnte entstanden sind. Ähnlich dem Internet, sind auch Videospiele mittlerweile Teil der Mainstream Kultur geworden. Dies liegt auch am technischen Fortschritt und einer sich stetig verbessernden IT Infrastruktur. Beides Faktoren, welche sich positiv auf die Verbreitung des Internets, von Videospielen und in weiterer Folge von Online-Multiplayer-Spielen ausgewirkt haben. So gibt es eine große Anzahl an Spielen die nicht nur alleine, sondern auch mit anderen Menschen gemeinsam gespielt werden können. Weltweit treffen sich Millionen von Menschen um online miteinander zu spielen. [vgl. 1, Seite 1][vgl. 43][vgl. 42]

Betrachtet man den technischen Fortschritt der letzten Jahrzehnte, so stellen Online-Multiplayer-Spiele ein Fachgebiet dar, welches durchaus von Interesse ist. Während man sich vor 30 bis 40 Jahren noch dieselbe Maschine teilen musste, spielen nun bis zu hunderte von Menschen gemeinsam im selben Spiel. Eine Frage die sich hier nun stellt, ist wie derartige Multiplayer-Spiele aus technischer Sicht umgesetzt werden können, da es viele Aspekte gibt, die es zu beachten gilt. Redet man von lokalen Multiplayer-Spielen, bei welchen gemeinsam auf derselben Maschine gespielt wird, fällt es leicht beiden Spielern ein gleiches Abbild der gemeinsamen Spielwelt zu liefern. Sobald man die einzelnen Spieler jedoch vor jeweils eigene PCs oder Spielekonsolen setzt und über ein Netzwerk gespielt wird, sei es ein LAN oder das Internet, stösst man auf neue Herausforderungen. Latenzzeiten, Bandbreite und diverse andere Faktoren müssen hier von den Entwicklern beachtet werden, um den Spielern ein möglichst unbeeinträchtigtes Erlebnis bieten zu können. [vgl. 2, Seite 6ff und Seite 69ff][vgl. 3, Seite 7]

Online-Multiplayer-Spiele erfordern in der Regel eigene Software, die vom Benutzer auf der jeweiligen Maschine installiert werden muss. Möchte man die Installation eines Spieles jedoch vermeiden oder hat nicht die erforderlichen Benutzerrechte, bieten browserbasierte Spiele eine leicht zugängliche Alternative. Obwohl diesen nicht die technischen Möglichkeiten herkömmlicher Videospiele offen stehen, so bieten sie doch den Vorteil, dass man von jedem Computer mit

Internetzugang aus auf sie zugreifen kann, unabhängig von Ort und Zeit. Auch eine zentrale Speicherung von Spielständen oder ähnlichen Daten ist möglich, wodurch der Spieler auch von anderen Computern aus auf diese zugreifen kann. Hinzu kommt das hier in der Regel mit plattformunabhängigen Technologien gearbeitet wird, für das die meisten Browser nur ein entsprechendes Plugin oder einen entsprechenden Player brauchen. Deswegen benötigen derartige Applikationen keine Portierung zwischen Windows, MacOs oder Linux. Dadurch ergeben sich Vorteile welche nicht nur für Spiele, sondern in ähnlicher Form auch für andere Online-Applikationen interessant sind. Beispielsweise im Office Bereich, bei Web Desktops oder Rich Internet Applications im Allgemeinen. [vgl. 4, Seite 2][vgl. 41, Seite 7, Seite 13]

Trotzdem wirken sich diese technischen Einschränkungen auch auf die Vernetzbarkeit browserbasierter Spiele aus. Ähnlich dem Sprung zwischen lokalem und vernetztem Multiplayer-Spiel, bedeutet ein browserbasierter Ansatz wiederum neue Herausforderungen. Diese Arbeit befasst sich mit der Betrachtung und Bewertung ebenjener Faktoren und versucht Lösungswege anzubieten. Als Basis dienen bereits etablierte Algorithmen, Denkansätze und Vorgehensweisen aus dem allgemeinen Forschungsumfeld von Online-Multiplayer-Spielen. Durch eine genaue Untersuchung und Evaluierung dieser bereits vorhandenen Erkenntnisse soll es ermöglicht werden, darauf aufbauend ähnliche Lösungen für den browserbasierten Bereich zu finden.

Diese theoretische Auseinandersetzung mit den technischen Hintergründen von Online-Multiplayer-Spielen und deren browserbasierten Artverwandten findet in drei Teilabschnitten statt. Der erste Teil bietet einen Überblick über den Bereich der Multiplayer-Spiele im Allgemeinen. Hier wird sich nur peripher mit den technischen Aspekten befasst. Der Fokus liegt auf dem thematischen Umfeld und der geschichtlichen Entwicklung dieser Art von Applikationen. Es dient der Betrachtung einiger grundlegender Konzepte welche die Basis dieses Fachgebiets darstellen (sh. Kapitel 2).

Als nächstes werden die Grundlagen dieser angestrebten Vernetzung betrachtet. Dies umfasst unter Anderem grundlegende Netzwerkprobleme wie Latenz, Jitter und Paketverluste, das Dilemma zwischen Konsistenz und Responsiveness und eine Betrachtung des Aspekts der Skalierbarkeit derartiger Netzwerke. Daraus folgend werden verschiedene Lösungsansätze und Optimierungsmöglichkeiten für diese Problematiken vorgestellt und eine mögliche, browserbasierte Umsetzung dieser Konzepte betrachtet. Als Abschluß dieses Kapitels wird auf das Thema Sicherheit und Cheating Prevention eingegangen, da auch (bzw. vor allem) bei Spielen den Benutzern nie vollständig vertraut werden darf (sh. Kapitel 3). [vgl. 3, Seite 213]

Die praktische Umsetzung des erarbeiteten Wissens stellt den abschließenden Teil dieser Arbeit dar. Als Proof of Concept wird in diesem Kapitel ein browserbasiertes Online-Multiplayer-Spiel entwickelt. Hierbei liegt der Fokus weniger auf der Client-Seite des Spiels, sondern mehr auf der Server-Seite und vor allem der Client-Server Kommunikation im Sinne der betrachteten Konzepte für

Online-Multiplayer-Spiele. Bei dem Spiel selbst handelt es sich um ein rundenbasiertes Strategiespiel das allerdings auch verzögerungsarme Echtzeit-Elemente aufweist. Die dabei entstehende Software könnte als Grundlage für zukünftige, artverwandte Projekte dienen, welche außerhalb des Rahmens dieser Arbeit liegen. Auf die genauen Anforderungen und verwendeten Technologien wird im Laufe der Arbeit noch genauer eingegangen. Dasselbe gilt für die detaillierte Bewertung der Ergebnisse (sh. Kapitel 4).

Kapitel 2

Multiplayer-Spiele im Überblick

Als Einstieg in die Materie bietet das folgende Kapitel einen Einblick in die Entstehung von Online-Multiplayer-Spielen und diversen Umgebungsfaktoren. Der Aufbau erfolgt chronologisch und bewegt sich von Anfangs allgemeinen Konzepten zunehmend ins Detail. So werden zuerst die Anfänge von Multiplayer-Spielen beleuchtet und grundlegendes Wissen zu vernetzten Multiplayer-Spielen vermittelt um abschließend auf browserbasierte Multiplayer-Spiele im Speziellen einzugehen.

2.1 Die Anfänge der Multiplayer-Spiele

Rein per Definition müssen Multiplayer-Spiele nicht immer zwingend über das Internet oder andere Netzwerke gespielt werden. Vor allem die ersten Videospiele beschränkten sich auf lokale Multiplayer-Spiele, bei welchen beide Spieler sich dieselbe physische Maschine teilten. Während dies damals vor allem technische Hintergründe hatte, bieten auch heute noch viele aktuelle Spieletitel diese Möglichkeit an. Die Spieler bewegen sich hierbei entweder auf demselben Bildschirm und somit demselben Ausschnitt der Spielwelt, verwenden Split Screen wobei der Bildschirm geteilt wird oder spielen in Rotation und wechseln einander ab. [vgl. 2, Seite 5f]

Ein paar Jahre nach den ersten lokalen Multiplayer-Spielen, wurden auch bereits bestehende IT Infrastrukturen, Server und Mainframes, neben ihren bisherigen Aufgabengebieten, für spielerische Zwecke verwendet. Diese erste Generation von Online-Multiplayer-Spielen legten den Grundstein für viele Spiele die noch folgen sollten.[vgl. 2, Seite 8f]

2.1.1 Lokale Multiplayer-Spiele

Wenn auch kein Computerspiel im herkömmlichen Sinne, stellt das 1958 entstandene Tennis for Two den Urvater der Multiplayer-Spiele dar. Bei diesem Spiel wurde ein Oszilloskop benutzt, um es zwei Spielern zu ermöglichen virtuell Tennis zu spielen. Als Atari Anfang der 1970er Jahre das darauf basierende Spiel Pong veröffentlichte, wurde es zum ersten kommerziell stark erfolgreichen Videospiel. Auch wenn es dazwischen bereits andere Multiplayer-Spiele, wie beispielsweise das 1961 veröffentlichte Spacewar gegeben hatte, läutete Pong damit das Zeitalter der Arcade-Videospiele ein. [vgl. 5, Seite 46f] [vgl. 2, Seite 6ff]

Da bei einem lokalen Multiplayer-Spiel mehrere Personen mit derselben Maschine interagieren, ergibt sich der Vorteil, dass hier nur ein einziger Game State aufrecht erhalten werden muss. Das heißt, dass im Gegensatz zu vernetzten Multiplayer-Spielen, allen Mitspielern konsistente Abbilde der Spielrealität geliefert werden können, ohne mit den für Computernetzwerken üblichen Herausforderungen kämpfen zu müssen. Allerdings bringen lokale Multiplayer-Spiele auch Nachteile mit sich. So hat ein Spieler nie sämtliche Ein- und Ausgabegeräte für sich alleine zur Verfügung sondern muss diese immer mit seinen Mitspielern teilen. Sei es physisch (z.B. geteilte Tastatur oder Split Screen) oder zeitlich (abwechselndes Spielen). Dazu kommt die lokale Abhängigkeit [vgl. 3, Seite 7]. Im Gegensatz zu vernetzten Multiplayer-Spielen müssen sich bei ihren lokalen Artverwandten alle Spieler am selben Ort befinden. So traten, als es anfing technisch möglich zu werden, bald die ersten Multiplayer-Spiele auf, die über Netzwerke gespielt werden konnten.

2.1.2 Beginnende Vernetzung

Die ersten Spiele, welche Computernetze verwendeten, stellten noch keine Multiplayer-Spiele im herkömmlichen Sinne dar. Die damaligen Spieleserver ermöglichten keine Interaktion zwischen den Spielern, sondern stellten ihnen lediglich die Software zur Verfügung welche die Benutzer über ihre Terminals ausführen konnten. Eine dieser Plattformen stellte PLATO dar, welches neben Spielen vor allem auch Community Features wie Email und Chatfunktionen zur Verfügung stellte. [vgl. 29]

Kurz nach PLATO gewannen Anfang der 1980er Jahre Multi User Dungeons, auch bekannt als MUDs, an Bedeutung[vgl. 3, Seite 7]. Diese stellten ihren Benutzern nicht nur eine Art virtuelle Welt zur Verfügung, sondern gaben ihnen auch bereits die Möglichkeit miteinander in dieser Welt zu interagieren. MUDs funktionieren im Prinzip wie Online-Chats mit Spielelementen und spieleähnlicher Struktur. Ähnlich einem Adventure Spiel konnten sich Spieler zwischen verschiedenen Orten und Räumen bewegen, um dort mit der Spielwelt oder anderen Spielern zu interagieren. [vgl. 6, Seite 121ff]

Die ersten MUDs hatten eine rein textuelle Oberfläche, in welcher der Spie-

ler seine Befehle eingeben konnte (z.B. “go east”, “open door”). Es wurde eine Client-Server Architektur verwendet, bei welcher die Spiele von ihren Terminals aus per Telnet auf den Server zugreifen konnten. Da Telnet jedoch die Antworten der MUD Server nicht immer optimal darstellte, kam es bald zu diversen anderen, spezialisierten, Clients die auf MUDs zugeschnitten waren. Die damaligen MUDs inspirierten so manche Multiplayer-Spiele der folgenden Generationen, wie beispielsweise Everquest. Sie zählen zu den ersten Spielen, die das gemeinsame Online Spielen ermöglichten. [vgl. 2, Seite 8f]

2.2 Netzwerk und Online-Multiplayer-Spiele

Mit der zunehmenden Verbreitung von LANs und dem Internet, öffneten sich neue technische Perspektiven, welche das Spielen über miteinander verbundene Maschinen noch weiter erleichterten. Darüber hinaus gab der technische Fortschritt Spiele-Entwicklern neue Möglichkeiten den Spieler in den Bann zu ziehen. Die grafische Darstellung und der Sound konnten rapide verbessert werden, womit eine intensivere Immersion des Spielers erreicht wurde. Anfang der 1990er zeigte der First-Person Shooter Doom was die neuen Technologien alles möglich machten. Es folgten viele Spiele ähnlicher Bauart. [vgl. 2, Seite 12]

Aber auch andere Genres konnten das Potential hinter den neuen Technologien für sich entdecken. Auf die einzelnen Genres und deren jeweiligen Anforderungen an die Netzwerkkommunikation wird in Abschnitt 2.2.2 genauer eingegangen.

2.2.1 Doom und der Beginn einer Ära

Als Doom 1993 von id Software[1] veröffentlicht wurde, stellte es den ersten First-Person Shooter dar welcher auch über ein Netzwerk gespielt werden konnte. Bei Doom konnten bis zu vier Spieler gemeinsam im Kooperations-Modus oder gegeneinander in sogenannten Death-Matches spielen. Dies beschränkte sich damals jedoch noch auf LANs. [vgl. 7, Seite 5][vgl. 27]

Um das Spielen über LANs zu ermöglichen, bediente sich Doom eine Peer-to-Peer Architektur. Jede Maschine auf der gespielt wurde, stellte einen unabhängigen Peer dar, auf welchem eine eigene Instanz des Spiels lief. Die einzelnen Doom Peers schickten 35 mal pro Sekunde ein Update der Benutzerinputs aus. Gleichzeitig warteten sie bis die Datenpakete sämtlicher Mitspieler eingelangt waren. Anhand einer fortlaufenden Paketnummer konnte das Spiel erkennen ob alle Daten angekommen waren. Im Falle einer unerwarteten Paketnummer ging das Spiel davon aus, dass ein vorhergehendes Paket verloren gegangen war und forderte die vermissten Daten an. Allerdings schickte das Spiel sämtliche

[1] http://www.idsoftware.com

Updates per Ethernet Broadcasts aus, was zur Folge hatte, dass sämtliche Computer, welche sich im selben Netzwerk befanden, bei ihrer Arbeit unterbrochen wurden. Vor allem bei der damaligen Hardware wirkte sich dies sehr störend aus. So kam es in diversen Firmen und Universäten zu Regelungen, welche sich gezielt gegen Doom richteten. [vgl. 2, Seite 13f]

Als Protokoll wurden sowohl bei Doom als auch anderen Spielen IPX anstatt IP verwendet. Der Grund hierfür war die leichtere Implementierung. Dies hatte jedoch den Nachteil, dass bei Spielen über das Internet eigene Software verwendet werden musste, welche das Tunneln von IPX über das Internet ermöglichte. Darüber hinaus waren viele der damaligen vernetzten Multiplayer-Spiele nur auf LANs ausgelegt, wodurch geringe Bandbreite und hohe Latenzzeiten verstärkt zu Problemen führten. Das 1996 veröffentlichte Spiel Quake war eines der ersten Spiele welche ohne Tunneling auskamen und für das Spielen über Internet gedacht waren. Darüber hinaus bot es die Möglichkeit auf Spieleserver zuzugreifen, welche durchgehend erreichbar waren. Durch diese zentralen Treffpunkte konnten sich die Spieler erstmals besser koordinieren, um gegeneinander anzutreten. [vgl. 2, Seite 19f]

Mittlerweile stellen derartige Client-Server Architekturen, für kommerziell entwickelte Online-Multiplayer-Spiele, die am häufigsten verwendete Variante dar. Es werden jedoch oft auch Hybridformen eingesetzt. Diese nutzen in der Regel Peer-to-Peer Technologien für Teile des Spiels, welche sich gar nicht oder nur peripher auf das Gameplay auswirken (z.B. Chat)[vgl. 2, Seite 16f]. Diese zusätzlichen Aspekte sind heutzutage in diversen Plattformen integriert um das Spielerlebnis selber mit Community Features zu erweitern[sh. 44][sh. 45][sh. 46].

2.2.2 Genres und deren technische Anforderungen

Im Laufe der 1990er Jahre begannen zunehmend mehr Genres auch im Bereich der Online-Multiplayer-Spiele präsent zu sein. Jedes dieser Genres steht vor eigenen Herausforderungen, um ihren Spielern ein möglichst immersives Multiplayer-Erlebnis zu bieten. Stellvertretend werden hier die Genres First-Person Shooter, Echtzeit-Strategiespiele und MMORPG (Massively Multiplayer Online Role-Playing Game) vorgestellt und betrachtet.

First-Person Shooter

Bei First-Person Shootern handelt es sich um schnelle, actionlastige Spiele aus der Ich-Perspektive. Bekannte Vertreter dieses Genres sind die bereits erwähnten Spiele Doom und Quake. Counterstrike[2], sowie die Spiele der Unreal Tournament[3] und Call of Duty Serie[4], gehören ebenfalls zu diesem Genre. All diesen

[2] http://store.steampowered.com/app/240/
[3] http://www.unrealtournament3.com
[4] http://www.callofduty.com

Spielen ist gemein, dass sie vom Spieler hohes Geschick und schnelles Reaktionsvermögen erfordern. Dementsprechend muss der Zustand der Spielwelt besonders schnell und präzise übermittelt werden. Vor allem Mitte der 1990er Jahre gab es bei Spielen über das Internet deswegen noch oft Probleme mit den damals geringen Übertragungsraten und hohen Latenzzeiten. Änderungen in der Spielwelt wurden oft nur langsam propagiert, wodurch die Immersion und selbst die Spielbarkeit von First-Person Shootern maßgeblich beeinträchtigt wurde [vgl. 2, Seite 20]. In der Regel sind bei Echtzeit-Systemen, wie sie Online-Multiplayer-Spiele darstellen, Latenzzeiten zwischen 0.1 und 1.0 Sekunden akzeptabel. Bei First-Person Shootern und ähnlich schnellen Spielen sollte die Latenz allerdings im unteren Bereich liegen[vgl. 3, Seite 173f]. Laut eines Tests der anhand des Spiels Unreal Tournament 2003 durchgeführt wurde, wird bei Spielern ein Round-Trip Delay ab 60ms (was einer Latenzzeit von ca. 30ms entspricht) bereits als störend empfunden [vgl. 8, Seite 5]. Die Toleranzgrenze liegt laut ähnlichen Studien bei einer Latenz von 150ms-180ms (Quake 3) bzw. 225ms-250ms (Half-Life)[vgl. 9, Seite 2].

Wie man erkennen kann liegt die größte Herausforderung bei First-Person-Shootern in der Geschwindigkeit der Spiele. Es muss also in diesen Fällen, noch mehr als bei anderen Genres, extrem Wert auf schnelle Kommunikation zwischen Client und Server gelegt werden. Im Optimalfall bedeutet dies, hohe Latenzzeiten zu vermeiden. Diese zeitlichen Verzögerungen können jedoch nie vollkommen umgangen werden. Dementsprechend müssen Techniken eingesetzt werden, welche den Faktor Latenz so gut wie möglich kompensieren um den Spielfluss aufrecht zu erhalten.

So können beispielsweise auf der Client-Seite Algorithmen angewandt werden, welche den derzeitigen Game State aufgrund bisheriger Daten berechnen ohne ein aktuelles Update erhalten zu haben[vgl. 2, Seite 21]. Dies kann beispielsweise durch Dead Reckoning erreicht werden, eine der zahlreichen Techniken auf die in Kapitel 3 genauer eingegangen wird. Da diese verbesserte Responsiveness jedoch auf Kosten der kollektiven Konsistenz des Game States geht, muss hier ein Weg gewählt werden, welcher die Konsistenz nur innerhalb einer akzeptablen Grenze beeinträchtigt [vgl. 3, Seite 184]. Die Bedeutung von Responsiveness und Konsistenz, sowie der damit verbundene Zwispalt, wird in Kapitel 3.1.4 detailliert abgehandelt.

Fairness ist ein weiterer Faktor der von Latenz beeinträchtigt werden kann. Spieler mit hoher Latenz sind Spielern mit niedriger Latenz gegenüber im Nachteil [vgl. 8, Seite 5]. Dementsprechend müssen Latenzunterschiede im Optimalfall ausreichend kompensiert werden. Dieses Problem kann unter anderem mit verschiedenen Time Manipulation Techniken angesprochen werden [vgl. 2, Seite 93ff]. Eine weitere Möglichkeit bietet automatisiertes Latency Balancing, bei welchem unterschiedliche Latenzzeiten bei den Spielern durch den Einsatz eines künstlichen Delays ausgeglichen wird. Die hierbei eingesetzte Software kann unabhängig von der Applikation als zusätzliche Ebene zwischen Client und Server implementiert werden [vgl. 9, Seite 3ff].

Echtzeit-Strategiespiele

Bei Echtzeit-Strategie Spielen handelt es sich, wie der Name schon sagt, um Strategiespiele, die im Gegensatz zu rundenbasierten Spielen in Echtzeit ablaufen. Bei diesen Spielen geht es darum, anhand gegebener Ressourcen ein bestimmtes Ziel zu erreichen (beispielsweise den Gegner zu besiegen). Diese werden genutzt um strategisch wertvolle Gebäude und Einheiten zu produzieren, welche bestmöglichst eingesetzt werden müssen. Beispielsweise um die Karte zu erkunden, den Gegner anzugreifen und die eigene Basis zu verteidigen [vgl. 10, Seite 7]. Nennenswerte Vertreter dieses Genres sind Dune II und die Command and Conquer Serie[5], sowie die Spiele der Warcraft Reihe[6]. Der erste Teil von Warcraft, welcher 1994 von Blizzard veröffentlicht wurde, gilt als das erste Echtzeit-Strategiespiel, welches auch über das Internet gespielt werden konnte. Obwohl sich der Erfolg in Grenzen hielt, ebnete es den Weg für Warcraft II, welches bis zu 8 Spieler gleichzeitig ermöglichte. Auch dieses Spiel basierte auf dem IPX Protokoll. Im Prinzip konnte es deshalb nur über LANs gespielt werden, aber wie den First-Person Shootern dieser Zeit bot spezielle Tunnelling Software die Möglichkeit online zu spielen. Der Erfolg dieses und auch anderer Online-Multiplayer-Spiele führte schließlich zu Blizzard's Battle.net, welches es erleichtert Mitspieler zu finden. Es bietet auch einige Community Features. [vgl. 2, Seite 23f]

Da es bei diesen Spielen mehr auf Strategie als auf schnelle Reaktionen ankommt, lässt sich erkennen, dass sie um einiges latenztoleranter sind, als beispielsweise First-Person Shooter. Im Allgemeinen kann man davon ausgehen, dass bei dieser Art von Spielen Latenzzeiten von bis zu 500ms akzeptabel sind [vgl. 3, Seite 173]. Es gibt allerdings auch eine Studie, die anhand von Warcraft III zeigt, dass sich in manchen Fällen selbst Latenzzeiten von mehreren Sekunden nur wenig bis gar nicht auf die Leistung des Spielers auswirken [vgl. 11, Seite 11].

Eine der technischen Herausforderungen dieses Genres sind die Vielzahl an Einheiten, die von den Spielern kontrolliert werden. So ist es durchaus möglich, dass Hunderte an Einheiten gleichzeitig im Spiel sind. Es liegt auf der Hand, dass das Propagieren des aktuellen Status jeder einzelnen Einheit keine optimale Lösung darstellt. Auch wenn in diesem Fall Game State Updates, zumindest theoretisch, seltener erfolgen müssen, wie beispielsweise bei First-Person Shootern. Eine mögliche Lösung dieses Problems ist synchrone Simulation. Hierbei wird zwischen bestimmbaren und unbestimmbaren Ereignissen unterschieden. Bestimmbare Ereignisse werden vom Spiel selber ausgelöst. Unbestimmbare Ereignisse sind jene, auf welche das Spiel keinen Einfluss hat, wie beispielsweise der Input der Spieler. Ist die Ausgangssituation diesselbe, bleibt der Game State bei allen Spielern solange synchron, bis ein unbestimmbares Ereignisse passiert. Dementsprechend reicht die Übertragung der unbestimmbaren Ereignisse. [vgl. 3, Seite 205]

[5] http://www.commandandconquer.com
[6] http://eu.blizzard.com/de/war3/

MMORPGs

MMORPGs sind Rollenspiele, die von hunderten Menschen gleichzeitig über das Internet gespielt werden. Die Weiterentwicklung des eigenen Avatars stellt bei diesen Spielen einen der Kernaspekte dar. Durch Kämpfe und spezielle Aufgaben bekommt der vom Spieler gesteuerte Charakter Erfahrungspunkte, welche sich wiederum auf dessen Fähigkeiten auswirken. Dazu kommen laufend neue Ausrüstungsgegenstände, welche sich der Spieler in der Regel erkämpfen oder erkaufen muss. Diese wiederum setzen oft eine bestimmte Anzahl an Erfahrungspunkten voraus. Eine weitere Charakteristik dieses Genres sind riesige Spielwelten, welche vom Spieler erforscht werden können. Dazu kommt bei MMORPGs die soziale Komponente, dass man mit einer Vielzahl von Menschen online interagieren kann. Zu nennenswerten Vertreter dieses Genres zählen Ultima Online[7] und EverQuest[8], welche beides als Urväter des Genres gelten und natürlich World of Warcraft[9], welches Dezember 2008 mehr als 11 Millionen Spieler verzeichnen konnte. [vgl. 10, Seite 7f][vgl. 2, Seite 21f][vgl. 43]

Die wesentlichen Herausforderungen, auf welche man bei MMORPGs stößt, lassen sich direkt auf deren Größe schließen. So muss die Latenz, trotz hunderten von Spielern, innerhalb eines akzeptablen Rahmens gehalten werden. Dazu kommen riesige virtuelle Welten, welche nicht von einem einzelnen Server gehostet werden können und so in verschiedene Untergebiete aufgeteilt werden müssen. [vgl. 12, Seite 1][vgl. 2, Seite 27]

Dieses sogenannte Zoning ist eine für MMORPGs markante Vorgehensweise. Die Spielwelt wird in mehrere Zonen aufgeteilt, welche sich auf jeweils eigenen Server befinden. Überschreitet ein Spieler die Grenze zwischen zwei Zonen, wechselt er somit auch den Server. Der Nachteil sind bei dieser Vorgehensweise allerdings die Ladezeiten, welche beim Wechsel zwischen den Zonen auftreten. Ein Problem welches die Illusion einer einzigen, großen Welt zerstört und somit die Immersion der Spieler negativ beeinflusst. [vgl. 12, Seite 1]

Ein anderer, ähnlicher Ansatz, versucht genau diese Grenzen zu vermeiden, oder zumindest verschwimmen zu lassen. In diesem Fall gibt es keine klaren Grenzen, sondern Grenzgebiete. Diese werden auf zwei verschiedenen Servern gehostet. Verlässt der Spieler die Zone auf Server A, kommt er in ein Grenzgebiet welches mit Server B abgestimmt ist. Der Spieler wechselt hierbei nicht sofort den Server, sondern erst nachdem er das Grenzgebiet vollkommen durchquert und somit die Zone auf Server B erreicht hat. Die Nachteile bei dieser Lösung sind unter anderem der hohe Entwicklungsaufwand und ein erhöhter Bedarf an Netzwerkressourcen für Ereignisse die in diesem Grenzgebiet stattfinden. [vgl. 12, Seite 2]

Für First-Person Shooter und Echtzeit-Strategiespiele ist Zoning eher unge-

[7] http://www.uoherald.com
[8] http://everquest.station.sony.com
[9] http://www.wow-europe.com

eignet, da deren Spielwelten in der Regel nicht groß genug sind, um wirklichen Nutzen daraus ziehen zu können. Dazu kommt, dass es bei solchen Spielen leicht passieren kann, dass sich das Kampfgeschehen auf ein bestimmtes Gebiet konzentriert, was wiederum zu einer erhöhten Belastung des Servers führen würde, der für die jeweilige Zone verantwortlich ist. [vgl. 13, Seite 1]

Man sieht, dass vor allem bei MMORPGs die Skalierbarkeit eine wesentliche Rolle spielt. Dazu kommt die Tatsache, dass komplexe IT Infrastrukturen benötigt werden, um sowohl die große Anzahl an Spielern, als auch die riesigen Spielwelten verwalten zu können.

Vergleich der Anforderungen

Anhand der verschiedenen Genres die vorgestellt wurden, lassen sich Aspekte von Online-Multiplayer-Spielen erkennen, die je nach Art des Spiels unterschiedliche Bedeutung haben. In den konkreten Fällen wurden die Faktoren Latenz, Skalierbarkeit und die Simulation einer Vielzahl von Spielobjekten betrachtet.

First-Person Shooter benötigen eine möglichst geringe Latenz, MMORPGs müssen leicht skalierbar sein und Echtzeit-Strategie Spiele brauchen eine Technik die es ihnen erlaubt möglichst viele Einheiten im kollektiven Game State zu halten. Dies spiegelt jedoch nur eine Priorisierung wieder. So handelt es sich hierbei um Faktoren die für jedes Online-Multiplayer-Spiel interessant sind, wenn auch mit einem jeweils unterschiedlichen Stellenwert.

2.3 Browserbasierte Multiplayer-Spiele und deren Vorteile und Einschränkungen

Nachdem auf den vorhergehenden Seiten wichtige Aspekte und grundlegende Faktoren von Online-Multiplayer-Spielen beleuchtet wurden, wird das Thema nun im Zusammenhang mit browserbasierten Technologien betrachtet. Obwohl viele der Konzepte klassischer Online-Multiplayer-Spiele übernommen werden können, müssen sie in diesem Fall dennoch aus einem anderen Blickwinkel betrachtet werden. Das Verwenden browserbasierter Technologien bringt zwar Vorteile mit sich, bedeutet allerdings auch, dass man sich mit technischen Einschränkungen abfinden muss. Diese Vor- und Nachteile, sowie auch andere wesentliche Aspekte, werden im Folgenden behandelt.

2.3.1 Vorteile browserbasierte Spiele

Wie bereits in der Einleitung erwähnt, benötigen Online-Multiplayer-Spiele eigene Client-Software, welche am Computer des jeweiligen Spieler installiert sein

muss. Browserbasierte Spiele laufen hingegen innerhalb des Browsers und müssen deswegen nicht extra installiert werden. Dies bringt gleich mehrere Vorteile mit sich. Erstens, der Benutzer muss keinen möglicherweise komplexen und langwierigen Installationsvorgang durchführen. Hierzu zählt nicht nur der Installationsvorgang des Spiels selber, sondern auch die Installation diverser Treiber und ähnlicher Software, die gegebenenfalls benötigt werden, um das Spiel optimal darzustellen. Ein weiterer Vorteil ist, dass ein Benutzer browserbasierte Spiele auch auf Maschinen spielen kann, auf denen er nicht den benötigten Zugriff für eine Installation hat, wie beispielsweise öffentlich zugänglichen Computern. Dies gilt auch für Maschinen, auf denen der Benutzer sie aus persönlichen Gründen nicht installieren möchte. [vgl. 4, Seite 2]

Weiters liegt es in der Natur browserbasierter Spiele, dass diese online verfügbar sind. Das heißt, dass man mit jedem Computer, der über einen Internetanschluss verfügt, auf jene zugreifen kann. Auf diese Weise können derartige Spiele nicht nur nahezu überall gespielt werden, sondern man kann sie auch leicht Freunden und Verwandten weiterempfehlen. Handelt es sich um ein Spiel wo der Benutzer seinen Fortschritt speichern möchte, besteht die Möglichkeit diesen Spielstand auf einen zentralen Server zu speichern. In diesem Fall kann nicht nur das Spiel ortsunabhängig aufgerufen werden, sondern auch der jeweilig persönliche Spielstand des Benutzers (z.B. Anhand eines Logins). [vgl. 14, Seite 2]

Diese Omnipräsenz browserbasierter Spiele wird durch einen weiteren Faktor verstärkt: Plattformunabhängigkeit. Im Gegensatz zu klassischer Software ist hier keine Portierung zwischen unterschiedlichen Betriebssystemen nötig. Computerspiele, die im Browser gestartet werden, brauchen meistens nur bestimmte Plugins. Und selbst auf die kann in manchen Fällen, wie beispielsweise bei Spielen die mit JavaScript geschrieben wurden, verzichtet werden. Einige dieser Plugins sind weit verbreitet und können deswegen als gegeben angesehen werden [vgl. 40]. Sie stellen eine eigene Meta-Ebene dar, auf der browser- und plattformunabhängig gearbeitet werden kann. Auf diese Weise kann man aus ein und derselben Quelldatei, Inhalte und Software für sämtliche Plattformen generieren, auf denen das entsprechende Plugin zur Darstellung installiert ist. [vgl. 41, Seite 7, Seite 13]

Dadurch, dass die Client-Software von einem zentralen Server geladen wird und nicht am Computer des jeweiligen Benutzers installiert ist, fällt es leichter Updates durchzuführen. Dies bedeutet, dass Updates nicht nur für die Server-Seite, sondern auch für die Client-Seite des Spiels an zentraler Stelle durchgeführt werden können. Dementsprechend kann davon ausgegangen werden, dass sämtliche Clients über die jeweils aktuelle Version verfügen. [vgl. 14, Seite 2]

2.3.2 Technische Einschränkungen und Gegebenheiten

Vorab sei zu sagen, dass die technischen Einschränkungen auch stark von der jeweilig verwendeten Technologie abhängt. Dies gilt sowohl für die vorhandenen

Möglichkeiten zur Netzwerkkommunikation, als auch für die graphische Darstellung des Spiels und die Interaktion. So können beispielsweise Animationen und anderere multimediale Inhalte, wie Sound, mittels Adobe Flash leichter und effizienter eingesetzt werden, als mit JavaScript[vgl. 41, Seite 7ff]. Ähnlich verhält es sich bei serverseitigen Technologien. Es folgt eine kurze Klassifizierung der möglichen clientseitigen Technologien, sowie eine Betrachtungen diverser technischer Herausforderungen die bei browserbasierten Online-Multiplayer-Spielen zu beachten sind. Dieses Wissen soll bei der späteren praktischen Umsetzung eines browserbasierten Online-Multiplayer-Spiels zum Tragen kommen.

Clientseitige Einschränkungen

Im wesentlichen können drei verschiedene Klassen von clientseitigen Technologien erkannt werden:

- Technologien, welche rein browserbasiert sind und deswegen abgesehen vom Browser selber keine weitere Laufzeitumgebung benötigen. Dieser Ansatz wird beispielsweise von AJAX verfolgt.
- Plugin-basierte Technologien, für welche eine Erweiterung des Browsers in Form einer einmaligen Installation erforderlich sind. Applikationen die auf diese Technologien aufbauen, verwenden das jeweilige Plugin als Laufzeitumgebung. In diese Kategorie fallen unter anderem Java und Flash.
- Stand-alone Technologien, welche eine vom Browser separate Laufzeitumgebung benötigen. Diese seien jetzt allerdings nur der Vollständigkeit halber erwähnt, da es sich hierbei um Technologien handelt, welche außerhalb des Browsers aktiv sind. Beispiele hierfür sind Java Web Start und die auf Flash basierende Adobe Integrated Environment (AIR). [vgl. 14, Seite4][vgl. 38, Seite 45f]

Browserbasierte Spiele und Applikationen sind von ihrer jeweiligen Laufzeitumgebung abhängig. Dies bedeutet, dass diese Art von Applikationen nicht mehr technische Möglichkeiten bieten können, als die Laufzeitumgebung in der sie sich befinden. Dazu kommt, dass die Handlungsfreiheit derartiger Software auch aus Sicherheitsgründen eingeschränkt wird. Da browserbasierte Applikationen standartmäßig ausgeführt werden sobald sich der Benutzer auf der entsprechenden Website befindet, könnte sonst schadhafter Code auf den Client geschleust werden, was das Betriebssystem kompromittieren würde. Desktop-Applikationen die unabhängig vom Browser agieren, bieten somit mehr Möglichkeiten und können Teile des Betriebssystems nutzen (Treiber, Ein- und Ausgabegeräte,...) welche den meisten browserbasierten Technologien nicht zur Verfügung stehen. [vgl. 39, Seite 4][vgl. 38, Seite 7ff und Seite 20f]

Ein weiterer Nachteil ist, dass browserbasierte Applikationen weniger effizient mit Systemressourcen umgehen können, als ihre am Desktop einheimischen Art-

verwandten. Dafür sind zwei wesentliche Faktoren ausschlaggebend: Die jeweilige Laufzeitumgebung stellt einen eigenen Hintergrundprozess dar, der wiederum Systemressourcen für sich selber in Anspruch nimmt. Auch wenn es sich in manchen Fällen nur um den Browser selber haltet. Zweitens, sind diese Art von Applikationen meist in Skriptsprachen geschrieben. Mit anderen Worten, im Gegensatz zu klassischen Programmiersprachen wie C++, wo der Quellcode zu Maschinencode kompiliert wird, werden die einzelnen Anweisungen vom Browser oder dem jeweiligen Plugin zur Laufzeit interpretiert und ausgeführt (sh. Abb. 2.1). Dementsprechend sind browserbasierte Applikationen in der Regel weniger performant als vergleichbare Programme welche in einer höheren Programmiersprache geschrieben wurden. [vgl. 39, Seite 4][vgl. 38, Seite 20f][vgl. 15, Seite 2]

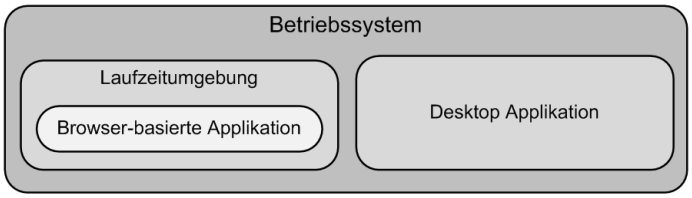

Abbildung 2.1: Browserbasierte Applikationen und Desktop Applikationen

Serverseitige Einschränkungen

Ein wesentlicher Unterschied zwischen der Server-Seite von browserbasierten und herkömmlichen Online-Multiplayer-Spielen, ist die Art der Serverumgebung. Für die serverseitige Lösung eines browserbasierten Online-Multiplayer-Spiels wird in der Regel ein Webserver oder eine ähnliche Laufzeitumgebung benötigt, in welcher die Applikation ausgeführt wird. Dies ist bei herkömmlichen Online-Multiplayer-Spielen nicht der Fall. Spieleserver stellen in diesen Fällen eigenständige Programme dar. Ähnlich dem Vergleich browser- und desktopbasierter Applikationen, kann hier in der Regel somit performanter gearbeitet werden. [vgl. 2, Seite 187ff][vgl. 24, Seite 2ff]

Entwicklern stehen auf der Server-Seite verschiedene webbasierte Alternativen zur Verfügung. So kann mittels CGI (Common Gateway Interface) am Server auf externe Software zugegriffen werden, welche in einer höheren Programmiersprache wie C++ geschrieben wurde. Die Software selber ist zwar in diesen Fällen performanter, verliert diesen Vorteil jedoch dadurch, dass für jede Anfrage ein eigener Prozess gestartet werden muss. Deswegen stellen für Webapplikationen Programmier- und Skriptsprachen wie JAVA, C# oder PHP und die damit verbundenen Frameworks den üblicheren Weg dar. Eine genaue Betrachtung der jeweiligen Vor- und Nachteile dieser und auch anderer serverseitiger Lösungen würde jedoch den Rahmen dieser Arbeit sprengen. [vgl. 24, Seite 3f]

Einschränkungen bei der Client-Server Kommunikation

Eine Herausforderung bei der Entwicklung browserbasierter Online-Multiplayer-Spiele stellt die Verbindungslosigkeit des HTTP Protokolls dar. Der Spieler kann somit keine bleibende Verbindung mit dem Server aufrecht erhalten und muss sich immer wieder von neuem mit ihm verbinden. Diese Lösung ist allerdings nicht besonders performant. Alternativ können jedoch Sockets verwendet werden, welche TCP benutzen. Bei diesen wird zwischen Client und Server eine persistente, bidirektionale Verbindung aufgebaut, welche zur Kommunikation genutzt werden kann. XML Sockets, beispielsweise, stellen eine spezielle Form von Sockets dar und werden oft für die Server-Seite von Multi-User Applikationen verwendet. Für die Umsetzung einer Client-Server Architektur mittels XML Sockets stehen einem sowohl auf Server- als auch auf Client-Seite viele verschiedene Technologien zur Verfügung. Dementsprechend stellen sie für browserbasierte Online-Multiplayer-Spiele einen sehr interessanten Ansatz dar. Bei sehr hohem Kommunikationsaufkommen kann der XML Parser jedoch zum Flaschenhals werden und die Performance beeinträchtigen. [vgl. 37, Seite 1f und Seite 5]

Für manche Spiele können Jabber Server verwendet werden, eine Technologie die normalerweise für Instant Messaging verwendet wird. Diese verwenden XMPP (Extensible Messaging and Presence Protocol), welches auf XML basiert und Kommunikation zwischen verschiedenen Plattformen ermöglicht. Neben dem Verschicken von Nachrichten hat diese Technologie auch Presence Management implementiert, was bedeutet, dass dem Server immer bekannt ist, welcher Benutzer gerade online ist und welcher nicht. Ein Aspekt der beispielsweise für die Implementierung einer Spiel-Lobby von großem Vorteil sein kann. [vgl. 16, Seite 1f]

Als nächstes stellt sich die Frage, in welcher Form die Daten zwischen Client und Server ausgetauscht werden können. Ein naheliegender Weg ist die Verwendung von XML, wie es bei den vorher erwähnten XML Sockets und Jabber Servern eingesetzt wird. Dadurch, dass es sich hierbei um ein weit verbreitetes Format handelt, gibt es sowohl auf Client- als auch auf Server-Seite viele verschiedene Möglichkeiten der technischen Umsetzung. Dies bringt wiederum eine hohe Plattformunabhängigkeit mit sich. XML Nachrichten sind außerdem auch vom Menschen leicht lesbar, was zumindest in der Entwicklungsphase das Debuggen erleichtern kann [vgl. 37, Seite 2]. Ein großer Nachteil ist jedoch der vergleichsweise große Overhead der allein für den Aufbau der XML Struktur benötigt wird. Dies kann sich vor allem bei sehr update-intensiven Applikationen, wie es Online-Multiplayer-Spiele darstellen, äußerst negativ auswirken.

Eine Alternative zu XML stellt JSON dar, welches einen weitaus geringeren Overhead hat. Auch dieses wird von vielen Plattformen unterstützt und hat eine klare, eindeutige Struktur. Derartige Objekte sind sehr simpel aufgebaut und einfach zu verarbeiten. Sie sind genau wie XML vom Menschen lesbar. Zu den Nachteilen gehört jedoch, dass JSON nicht so erweiterbar ist wie XML

und sich komplexere Datenstrukturen schwerer darstellen lassen. JSON weist im Vergleich zu XML noch weitere Einschränkungen auf, wie ein Mangel an Namespaces und einem Validator. Diese werden jedoch in der Praxis für die Umsetzung eines vernetzten Multiplayer-Spiels irrelevant sein. Namespaces werden benötigt, um die einzelnen Elemente eines XML Dokuments in Kontext zu setzen und somit eine mögliche Mehrdeutigkeit zu verhindert. Bei der Verwendung von JSON Datenobjekten zur Client-Server Kommunikation bei vernetzten Multiplayer-Spielen, können diese jedoch auch ohne Namespaces in einen eindeutigen Kontext gesetzt werden. Ein XML Validator gewährleistet, dass ein XML Dokument wohlgeformt ist, sprich der erwarteten Datenstruktur entspricht. Eine derartige Überprüfung der Datenstruktur wird vor allem für den Austausch von Daten zwischen verschiedenen Web-Applikationen benötigt, um hier eine standartkonforme Kommunikation zu ermöglichen. Da man jedoch im Falle eines Online-Multiplayer-Spiels nur selten die Schnittstelle zur Serverkommunikation als öffentlichen Service zur Verfügung stellt, kommt man ohne Validator aus. Eine Überprüfung des Inputs auf Client- und Server-Seite ist natürlich trotzdem zu empfehlen. [vgl. 36, Seite 5f und Seite 48ff]

Den geringsten Overhead bietet jedoch das Übertragen von Binärdaten. Dies kann mittels binärer Sockets gelöst werden. Wie XML Sockets ermöglichen auch diese eine persistente Verbindung zwischen Client und Server, jedoch mit dem Unterschied das die Daten hier nicht in Form von XML-Dokumenten, sondern als reine Binärinformation übertragen werden. Dies benötigt aufgrund des geringeren Overheads weniger Bandbreite als XML oder JSON. Die ausgetauschten Daten sind in diesem Fall nicht mehr vom Menschen lesbar. Die Implementierung dieser Lösung kann sich in manchen Fällen als komplex oder nur schwer umsetzbar erweisen. Allerdings sollte dieser Ansatz, aufgrund seiner Effizienz, bei der Entwicklung eines browserbasierten Online-Multiplayer-Spiels zumindest in Erwägung gezogen werden. [vgl. 35]

2.3.3 Usability und Interaktion

Die Aspekte Usability und Interaktion sind zwar nicht technischer Natur, sollten aber dennoch kurz behandelt werden da sie bei Videospielen wesentliche Faktoren darstellen. Sie sind einen wichtiger Teil des Game Designs. Dazu kommt, dass es auch eines der Ziele der eingesetzten Technologien ist, durch ihren richtigen Einsatz Usability und somit Spielbarkeit möglichst gut zu gewährleisten.

Pragmatisch betrachtet sind Online-Multiplayer-Spiele nichts anderes als eine Art von Echtzeit-Anwendungen, die je nach Genre einen Reaktionszeitraum von ca. 100ms bis einigen wenigen Sekunden zulassen können (sh. auch Kapitel 2.2.2). Ähnliche Felder in diesem Bereich sind DIS (Distributed Interactive Simulation) und CSCW (Computer Supported Cooperative Work). DIS ist ein IEEE Standard welcher im militärischen Bereich angewandt wird und sich mit der Kommunikationsarchitektur vernetzter Simulationen befasst. Er spezifiziert unter anderem, dass die Latenz bei derartigen Systemen unter 100ms sein soll,

da der Zusammenhang zwischen hoher Latenz und niedriger Leistung des Benutzers nichtlinear verläuft. Dies bedeutet, dass je nach Anwendung, ab einer gewissen Latenzzeit die Interaktion schwer bis unmöglich wird. CSCW ist ein Gebiet welches sich mit kollaborativer, vernetzter Software befasst. Ein Beispiel dafür wäre ein virtuelles Whiteboard, welches verschiedene Benutzer gleichzeitig lesen, beschriften oder auf sonstige Arten editieren können. [vgl. 3, Seite 173f][vgl. 2, Seite 39]

Eines der Kernelemente des Interaktionsdesigns bei Spielen, ist das Feedback welches der Spieler für seine Handlungen erhält. Nach Möglichkeit sollte das Spiel auf jeden Input reagieren, egal ob es sich um positives oder negatives Feedback handelt. Diese Antwort kann visuell, aural oder bei manchen Eingabegeräten auch taktil gegeben werden[10]. Reagiert das Spiel nicht auf den Input des Spielers, in welcher Form auch immer, kann dies zu Frustration führen. Selbst wenn der Input des Spielers unerwartet ist und nicht vorgesehen war, sollte das Spiel zumindest mit einer Art Signal antworten. Dadurch kann der Spieler verstehen, dass sein Input zwar zur Kenntniss genommen wurde, jedoch im derzeitigen Kontext des Spiels nicht verstanden oder durchgeführt werden kann. [vgl. 10, Seite 18]

Bei browserbasierten Online-Multiplayer-Spielen ergibt sich noch eine weitere Facette dieses Aspekts: Die Interaktion mit dem Server und anderen Spielern. Auch hierfür muss der Benutzer Feedback oder Informationen zum aktuellen Status des Spiels erhalten. Beispielsweise ob das Spiel Daten vom Server lädt oder ob der Gegenspieler noch online ist. Um anzuzeigen, dass gerade auf den Server zugegriffen wird und kurz gewartet werden muss, können beispielsweise sogenannte Throbber oder Ladebalken verwendet werden. Throbber sagen dem Benutzer, dass die Applikation eine Aktion unbekannter Dauer durchführt. Dies kann zum Beispiel der Fall sein wenn auf eine Antwort oder Daten unbekannter Länge von einem Server gewartet wird. Eine konventionelle Darstellung wäre ein sich drehender Kreis, wie er von diversen Apple Produkten und Ajax Applikationen bekannt ist. Ist die Ladedauer bestimmbar, kann ein Ladebalken eingesetzt werden. Dieser gibt Auskunft darüber, wie weit der Prozess prozentuell fortgeschritten ist (sh. Abb. 2.2.[vgl. 17, Seite 17ff]

Wichtig ist es auch, dem Spieler Informationen über den Onlinestatus seiner Mitspieler zu geben. Im einfachsten Fall kann dies über eine Kontaktliste oder Lobby erfolgen, in der sämtliche anderen Benutzer, die gerade online sind, aufgelistet werden. Was auch von Interesse sein kann, sind Änderungen des Onlinestatus. Dies kann beispielsweise durch ein kurzes Aufblinken des Benutzernamens erfolgen, wenn dieser online geht oder sich gerade ausloggt. Weitet man diesen Gedanken aus, erscheint es auch sinnvoll Informationen darüber zu liefern, was das Gegenüber gerade macht. Die Rede ist hierbei nicht von Feedback, dass durch eine primäre Aktion des Gegenübers verursacht wird, wie der Erhalt einer Nachricht oder ein Treffer durch den Gegner. Sondern von Inputs, welche

[10] Wobei bei browserbasierten Spiele aus technischen Gründen nur die ersten beiden Varianten von Interesse sind.

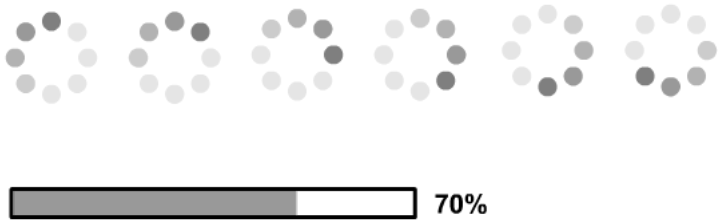

Abbildung 2.2: Bewegungsablauf eines Throbbers (oben), Ladebalken mit Prozentanzeige (unten)

diesen Aktionen vorausgehen. Dieses Prinzip wird zum Beispiel bei diversen Instant Messaging Clients angewandt [33][32]. So kann der Benutzer anhand eines kleinen Bereichs des Chatfensters erkennen ob das Gegenüber gerade an einer Nachricht schreibt. Im Bereich der Browser-Spiele wird diese Idee beispielsweise beim Spiel Quadradius angewandt, ein rundenbasiertes Strategiespiel welches Ähnlichkeiten mit dem Brettspiel Dame hat. Bei diesem Spiel werden die einzelnen Figuren per Drag-and-Drop auf dem Spielfeld bewegt. Sobald ein Spieler eine dieser Figuren virtuell aufhebt, erhält sein Gegner dies als visuelle Information. Dadurch können längere Interaktionspausen überbrückt werden und der Spieler erhält die Bestätigung, dass sein Gegenüber das Spiel nicht verlassen hat. Als Nebeneffekt wird hierbei das Gegenüber vermenschlicht, da man beispielsweise längere Denkpausen verfolgen kann bzw. sieht wenn sich der Gegenspieler doch für das Bewegen einer anderen Figur entscheidet [34].

Kapitel 3

Herausforderungen vernetzter Multiplayer-Spiele und deren Bewältigung

Am Anfang dieses Kapitel wird ein kurzer Überblick zu wesentlichen technischen Hintergründen gegeben. Darin finden sich einige Grundlagen zum Thema Computernetze und Netzwerkarchitektur, so wie die daraus resultierenden Implikationen welche für die Problemstellung dieser Arbeit interessant sind. Nach dieser Betrachtung der Grundsituation erfolgt eine Darlegung wesentlicher Lösungen und Ansätze, um die daraus resultierenden Herausforderungen zu bewältigen. Hierbei handelt es sich um bewährte Techniken aus dem Bereich der Online-Multiplayer-Spiele. Diese werden zunächst im Detail erläutert um anschließend auch aus Sicht browserbasierter Applikationen betrachtet zu werden. Anhand dessen wird die mögliche, technische Umsetzung mittels browserbasierter Technologien begutachtet und bewertet. Der letzte Teilabschnitt widmet sich dem Thema Cheating und Cheating Prevention, wobei zweiteres ein für Online-Multiplayer-Spiele markanter Aspekt ist.

3.1 Technische Hintergründe

Es folgt ein kurzer Exkurs in den Bereich der Netzwerk- und Kommunikationstechnik. Hierbei wird sich auf jene Aspekte konzentriert, die für das Thema Multiplayer-Spiele relevant sind. Nachdem die Grundlagen dieses Fachgebiets erläutert wurden, wird ein Blick auf die daraus folgenden Implikationen für Online-Multiplayer-Spiele geworfen. Dies ermöglicht ein besseres Verständnis der damit verbundenen Herausforderungen.

3.1.1 Herausforderungen auf der physischen Ebene von Computernetzen

Smed und Hakonen [3] teilen Computernetze auf drei Ebenen auf: Der physischen Plattform, der logischen Plattform und der vernetzten Applikation selber. Die physische Ebene ist verantwortlich für Ressourcen wie Bandbreite und Limitierungen wie Latenz. Diese Eigenschaften sind abhängig von der Infrastruktur welche auf dieser Ebene zu finden ist, beispielsweise Kabel und Hardware. Die logische Plattform setzt auf der physischen Plattform auf und verwaltet das Netzwerk. Dies betrifft das Steuern des Datenflusses, Routing, etc. Die vernetzte Applikation stellt die oberste Ebene dar. Diese verarbeitet und überprüft die übertragenen Daten um sie in Kontext zu setzen. Auf dieser Ebene liegen vernetzte Multiplayer-Spiele. [vgl. 3, Seite 172]

Die physische Plattform konfrontiert uns mit Faktoren wie limitierter Bandbreite, hoher Latenz und der beschränkten Leistungskapazität der einzelnen Knotenpunkte welche den Datenverkehr steuern. Diese stellen Einschränkungen der physischen Infrastruktur dar, auf welche die vernetzte Applikation keinen Einfluss hat. Dementsprechend müssen diese Einflüsse bei der Entwicklung einer derartigen Applikation berücksichtigt werden. Die Bandbreite bezeichnet die Übertragungskapazität einer Leitung innerhalb eines Netzwerks. An ihr kann man erkennen wieviele Daten innerhalb eines bestimmten Zeitraums übertragen werden können. Die Latenzzeit, ist die Dauer welche eine Nachricht bzw. ein Datenpacket vom Sender zum Empfänger benötigt. [vgl. 3, Seite 173]

Latenz ist unvermeidbar, da bei der Datenübertragung über ein Computernetz immer eine zeitliche Verzögerung auftritt. Die Serialisierung der Datenpakete wirkt sich ebenfalls auf die Latenz aus (Serialization Delay). Dies ist bedingt durch die Zeit welche benötigt wird, um ein einzelnes Datenpaket nach dem anderen über die Leitung zu schicken. Abhängig ist Serialization Delay vom Datendurchsatz und der Länge der einzelnen Datenpakete, wobei sich am zweiten Faktor erkennen lässt, dass auch ein Zusammenhang mit dem jeweils verwendeten Protokoll besteht. Eine weitere Quelle für höhere Latenzzeiten stellen Queuing Delays dar. Diese treten dann auf wenn mehrere Datenpakete gleichzeitig ankommen und somit nacheinander abgearbeitet werden müssen. [vgl. 2, Seite 70ff][vgl. 18, Seite 1f]

Die Änderungen der Latenz über einen bestimmten Zeitraum wird als Jitter bezeichnet und stellt eine zusätzliche Herausforderung da. Jitter kann beispielsweise durch Änderungen der Route geschehen, welche die Datenpakete über das Netzwerk nehmen. Aufgrund derartiger Routenwechsel kann es passieren, dass ein Pfad kürzer oder länger wird, was unterschiedliche Latenzzeiten zur Folge hat. Im Zusammenhang mit Serialization Delay kann Jitter aufgrund unterschiedlicher Paketgrößen auftreten. Selbiges gilt für Queuing Delay. [vgl. 2, Seite 73f]

Auch Paketverluste können Probleme verursachen. Diese können aus mehreren

Gründen auftreten. So gibt es bereits auf rein physischer Ebene eine geringe Rate an Datenkorruption. Dadurch treten Bitfehler auf. Diese können zwar in der Regel durch Redundanz und ähnliche Mechanismen erkannt und korrigiert werden, stellt sich jedoch heraus, dass ein Paket nachwievor beschädigt ist, so wird dieses verworfen. Weiters kann es passieren, dass ein Paket durch diverse Queues verloren geht. Gelangt der Buffer welcher die Queue hält an seine Kapazitätsgrenzen, müssen Pakete verworfen werden. Manche Systeme setzen dies pro-aktiv ein um von vornherein eine Überlastung der Queue zu vermeiden. Abgesehen davon kann ein Paketverlust auch durch Änderungen des Netzwerkpfades verursacht werden. Das ist dann der Fall, wenn nicht sofort eine voll funktionstüchtige Punkt-zu-Punkt-Verbindung hergestellt werden kann. [vgl. 2, Seite 74f]

Verzögerungsarme Echtzeit-Applikationen, wie sie Online-Multiplayer-Spiele oft darstellen, werden durch diese Faktoren besonders beeinträchtigt. Dementsprechend müssen Methoden angewandt werden, welche diesen Problemen entgegensteuern. Einige dieser möglichen Vorgehensweisen werden in Kapitel 3.2 genauer beleuchtet.

3.1.2 Protokolle

Zur Datenübertragung zwischen den Knotenpunkten müssen Protokolle verwendet werden. Diese besagen wie die Datenpakete aufgebaut sind, was sie bedeuten und wie im Fehlerfall reagiert wird. Neben IP sind die darauf aufbauenden Protokolle TCP und UDP grundlegend. TCP ermöglicht eine zuverlässige Punkt-zu-Punkt Verbindung. Die Daten werden in Pakete aufgeteilt, welche vom Empfänger in der richtigen Reihenfolge verarbeitet wird. Doppelte Pakete werden verworfen, fehlende nachgefordert. Diese Zuverlässigkeit geht jedoch auf Kosten von Verarbeitungszeit und benötigt darüber hinaus größere Datenpakete. UDP hingegen ist zwar schneller, jedoch auch weniger zuverlässig da verbindungslos. Die Daten kommen zwar in beliebiger Reihenfolge an, werden dafür aber schneller verarbeitet und in kleineren Paketen übertragen (minimaler Header). [vgl. 25][vgl. 26]

Man kann erkennen, dass diese beiden Protokolle im Bereich der Online-Multiplayer-Spiele jeweils unterschiedliche Anwendungsmöglichkeiten bieten. Eine Übertragung per TCP sollte dann erfolgen, wenn es sich um Daten handelt die für den Spielverlauf von großer Relevanz sind. Beispielsweise die Anzahl der Lebenspunkte, Munition, vorhandene Ressourcen und Sieg oder Niederlage. UDP ist vor allem für Spielinformation interessant, die möglichst schnell und ökonomisch übermittelt werden sollen, deren Übertragung jedoch nicht unbedingt verlässlich sein muss. Ein Beispiel hierfür wäre die Position der Spieler bei einem First-Person-Shooter oder Ähnlichem. Geht eine Positionsangabe zwischen Punkt A und Punkt B verloren, so kann dieser Datenmangel leicht durch Softwarelogik ausgeglichen werden. [vgl. 3, Seite 175]

HTTP stellt ein Protokoll dar, welches auf den vorher genannten Protokollen aufsetzt. Da es der Datenübertragung auf Applikationsebene im World Wide Web dient, liegt es Nahe auch die Kommunikation browserbasierter Spiele darauf aufzubauen. Von Nachteil ist allerdings, dass HTTP stateless ist, was heißt, dass die Verbindungen nicht persistent sind. Dementsprechend können in diesem Fall TCP Sockets eine sinnvolle Alternative darstellen (sh. Kapitel 2.3.2). [vgl. 31]

3.1.3 Netzwerkarchitekturen

Was den logischen Aufbau eines Computernetzwerks anbelangt, gibt es im wesentlichen Peer-to-Peer, Client-Server und Server-Netzwerk Architekturen.

Peer-to-Peer
Bei einer Peer-to-Peer Architektur sind alle Knotenpunkte des Netzwerks gleichgestellt. Jeder dieser Knotenpunkte ist mit jedem anderen Knotenpunkt des Netzwerks verbunden (sh. Abb. 3.1). Dieser Ansatz wurde vor allem bei vielen der ersten vernetzten Multiplayer-Spielen angewandt. Dies ist bei einer kleinen Anzahl an Spielern oder innerhalb eines LANs auch eine praktikable Lösung. Allerdings sind derartige Architekturen aufgrund der fehlenden Hierarchie nur schwer skalierbar. [vgl. 3, Seite 175][vlg. 19, Seite 3f]

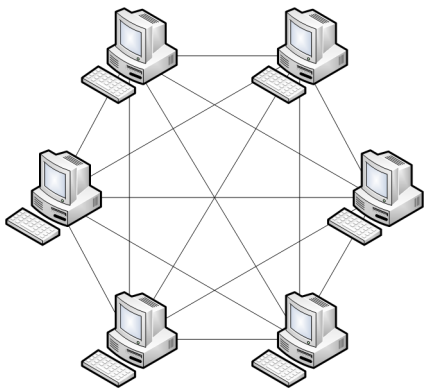

Abbildung 3.1: Peer-to-Peer Architektur

Client-Server
Bei Client-Server Architekturen stellt einer der Knotenpunkte den Server dar. Jegliche Kommunikation wird über diesen Server geleitet, während die anderen Knotenpunkte als Clients agieren (sh. Abb. 3.2). Dadurch, dass alle Pakete über den Server laufen wird die Übertragung zwar verlangsamt, bringt jedoch den Vorteil mit sich, dass der Datenfluss besser gesteuert werden kann. Dadurch muss nicht immer jede Nachricht an alle Clients ausgeschickt werden und mehrere Datenpakete können zu einem zusammengefasst werden um einen besseren Datenfluss zu gewährleisten. Dazu kommt, dass am Server eine Verwaltung implementiert werden kann. Allerdings stellt hier der einzelne Server einen Flaschenhals dar. [vgl. 3, Seite 176][vgl. 19, Seite 4]

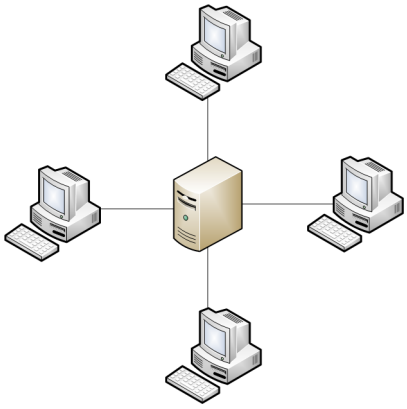

Abbildung 3.2: Client-Server Architektur

Server-Netzwerk
Server-Netzwerk Architekturen basieren auf mehreren zusammengefassten Servern. Diese interagieren untereinander wiederum in Form eines Peer-to-Peer Netzwerks. Durch diese Vorgehensweise kann die Belastung auf mehrere Server aufgeteilt werden, was wiederum bessere Skalierbarkeit ermöglicht. Als Nachteil zu sehen ist jedoch die vergleichsweise höhere Komplexität solcher Netzwerke. [vgl. 3, Seite 176]

3.1.4 Konsistenz vs. Responsiveness

Aufgrund der limitierten Ressourcen von Computernetzwerken ergibt sich die Dichotomie von Konsistenz und Responsiveness als wesentlicher Aspekt der beachtet werden muss. Im Bereich von Online-Multiplayer-Spielen bezeichnet Konsistenz die Ähnlichkeit der Game States auf den einzelnen Clients. Vollkommene Konsistenz würde heißen, dass bei allen Spielern der Game State absolut ident ist. Um dies zu erreichen müsste allerdings jeder Client auf die Updates der Anderen warten bevor er mit dem Spiel fortfahren kann. Responsiveness bezeichnet die zeitliche Verzögerung bis ein Ereignis bzw. eine Änderung des Game States registriert und verarbeitet wurde. Um möglichst hohe Responsiveness zu erreichen, muss das Spiel gegebenenfalls fortfahren bevor alle Mitspieler das Update erhalten haben. Man erkennt, dass diese beiden Aspekte einen Zusammenhang haben und einander entgegen wirken können. Für vernetzte Multiplayer-Spiele und ähnliche verzögerungsarme Echtzeit-Anwendungen ist hohe Responsiveness jedoch wichtiger als Konsistenz, weshalb letztere gegebenenfalls beeinträchtigt wird. [vgl. 3, Seite 184]

Um möglichst hohe Konsistenz zu erreichen, müssen die Prozesse welche auf den einzelnen Knotenpunkten eines Netzwerks laufen, eng verbunden sein und miteinander synchronisiert sein. Dies benötigt hohe Bandbreite und kurze Latenzzeiten, sowie eine niedrig gehaltene Anzahl an Knotenpunkten. Will man möglichst hohe Responsiveness erreichen, müssen die Knotenpunkte in der Lage

sein unabhängig vom Zustand der anderen Knotenpunkte zu agieren. Dementsprechend sollten die Knotenpunkte und die auf ihnen laufenden Prozesse nur lose miteinander vernetzt sein. Desweiteren ist zusätzliche Rechenkapazität erforderlich um die Anforderungen an Bandbreite und Latenz gering zu halten. In der Praxis lassen sich diese beiden Anforderungen nicht vereinbaren, weshalb je nach Priorität auf der einen oder anderen Seite Abstriche gemacht werden müssen. [vgl. 3, Seite 185]

Aus Sicht der Multiplayer-Spiele ergeben sich zwei mögliche Extreme: Eine auf allen Clients vollkommen konsistente Spielwelt mit langer Reaktionszeit und eine dynamische Spielwelt, die zwar schnell reagiert aber nicht auf allen Clients den gleichen Game State hat. Um die Konsistenz der kollektiven Spielwelt möglichst hoch zu halten, müssen nicht nur viele Updates geschickt, sondern auch auf Empfangsbestätigungen der Clients gewartet werden. Darüber hinaus müssen die verschickten Daten mit den Informationen der anderen Clients verglichen werden um Inkonsistenzen zu vermeide. Daraus ergibt sich ein zeitlicher Aufwand der zu Kosten der Responsiveness geht. Um die Responsiveness zu verbessern, sind wiederum Vorgehensweisen notwendig, welche der Konsistenz schaden. So besteht die Möglichkeit auf Empfangsbestätigungen zu verzichten. Die Applikation muss dadurch nicht warten und kann möglichst schnell weitergeführt werden. Die zweite ist es, die Anzahl der erforderlichen Updates zu reduzieren. Dies kann durch diverse Prediction Techniken erreicht werden. Das Spiel geht in diesem Fall weiter, auch wenn der Client aufgrund beschränkter Netzwerkkapazitäten nur wenige Updates erhält. Um in diesen Fällen eine ausreichende Konsistenz aufrecht zu erhalten, müssen verschiedene Methoden und Techniken angewandt werden um problematische Inkonsistenzen zu vermeiden oder zu korrigieren. [vgl. 3, Seite 184ff][vgl. 2, Seite 89]

3.2 Lösungsansätze, Optimierung und mögliche browserbasierte Umsetzung

All diese grundlegenden Herausforderungen beim Vernetzen von Multi-User Applikationen resultieren in einen Bedarf an entsprechenden Lösungen. Es gibt eine Vielzahl an Möglichkeiten zur Kompensation und Korrektur der aus der Übertragung per Computernetzen resultierenden Probleme, sowie zur Optimierung der Kommunikation um die gegebenen Netzwerke möglichst effizient zu nutzen. In weiterer Folge wird eine mögliche Umsetzung mit Hilfe browser- und web-basierter Technologien betrachtet.

3.2.1 Kompression und Aggregation von Nachrichten

Durch die Kompression der zu übermittelnden Daten kann bei vernetzten Applikationen Bandbreite gespart werden. Dies geht jedoch auf Kosten von Rechenkraft, da sowohl En- als auch Decoden der Nachrichten einen Aufwand

darstellen.

Verlustfreie und verlustbehaftete Kompression
Kompression kann sowohl verlustfrei als auch verlustbehaftet stattfinden. Bei verlustfreier Kompression geht keine Information verloren sondern kann vollständig wiederhergestellt werden. Will man eine bessere Kompressionsrate erreichen, kann man auf verlustbehaftete Methoden zurückgreifen. Bei dieser Technik geht jedoch Information verloren, da jene Daten verworfen werden die vergleichsweise irrelevant sind, sodass der Datenverlust möglichst wenig auffällt. [vgl. 3, Seite 190]

Interne und externe Kompression
Setzt man die zu komprimierenden Daten in Kontext mit den bereits davor erhaltenen Informationen, ergibt sich eine zusätzliche Klassifizierung in interne und externe Kompression. Interne Kompression geschieht ohne Rücksicht auf die anderen Daten, so dass nur die jeweils aktuelle Nachricht komprimiert wird. Bei externer Kompression wird jede Nachricht in Zusammenhang mit den Daten davor betrachtet, sodass eine bessere Kompressionsrate ermöglicht wird. Dies kann zum Beispiel dadurch erfolgen, dass man nicht den gesamten Game State überträgt, sondern nur die jeweiligen Änderungen. Da interne Kompression jede Nachricht unabhängig von den Anderen betrachtet, ist sie für nicht-zuverlässige Übertragungsdienste wie UDP geeignet. Externe Kompression hingegen benötigt ein verlässliches Protokoll wie TCP. [vgl. 3, Seite 190]

Aggregation von Nachrichten
Bei der Aggregation von Nachrichten reduziert man die Anzahl der Datenpakete, indem man die Informationen mehrerer aufeinanderfolgender Updates zusammenfasst. Dadurch, dass eine großes Datenpaket weniger Headerinformationen erfordert als eine Menge kurzer Datenpakete, kann so der Overhead reduziert werden. Dementsprechend spart dieser Ansatz Bandbreite. Allerdings muss hierfür Rechenkraft aufgewandt werden und die Responsiveness der Applikation wird dadurch beeinträchtigt. Die Akkumulierung der Nachrichten kann timeout-based und quorum-based erfolgen. Geht man timeout-based vor, werden alle Nachrichten zusammengefasst die sich innerhalb eines vordefinierten Zeitraums ansammeln. Dadurch kann garantiert werden, dass es zu keiner zu großen Verzögerung kommt. Im schlimmsten Fall erspart man sich keine Bandbreite, da es passieren kann, dass innerhalb der vorgegebenen Zeit nur ein einziges Update zustande kommt. Ist die Vorgehensweise quorum-based, fasst man jedesmal eine vordefinierte Anzahl an Nachrichten zusammen. Dadurch wird bei jedem Update eine bestimmbare Menge an Bandbreite gespart. Allerdings bringt dieser Ansatz den Nachteil, dass nicht garantiert werden kann wie oft die Updates erfolgen. So kann es passieren, dass die Verzögerung zwischen den Updates so groß ist, dass das Spiel oder die Echtzeit-Applikation beeinträchtigt wird. Dementsprechend kann jedoch ein hybrider Ansatz von Interesse sein. In diesem Fall werden die Informationen zusammengefasst und verschickt, sobald entweder ein bestimmter Zeitraum verstrichen ist oder sich genügend Updates angesammelt haben. [vgl. 3, Seite 191]

Möglichkeiten zur browserbasierten Umsetzung

Die soeben genannten Verfahren können mit gewissen Vorbehalten auch für browserbasierte Applikationen verwendet werden.

Verlustfreie und verlustbehaftete Kompression
Verlustfreie Kompression von Nachrichten kann beispielsweise unter Verwendung des LZW-Algorithmus erfolgen [20]. Dieser wird in Verbindung mit anderen Algorithmen von zahlreichen Programmen zur Komprimierung verwendet. Da der LZW-Algorithmus leicht implementiert werden kann, gibt es für viele client- sowie serverseitige Technologien eine entsprechende Umsetzung. Ähnlich verhält es sich mit den LZ77-Algorithmus und anderen Algorithmen der LZ Familie [30].

Die technische Umsetzung ist simpel. Man implementiert sowohl am Server als auch am Client den Kompressionsalgorithmus seiner Wahl. Der Client encoded die Daten, schickt sie an den Server welcher sie dann wiederum mit dem jeweiligen Algorithmus decoded und verarbeitet. Allerdings muss man beachten, dass das en- und decoden der Daten Rechenaufwand darstellt. Auf Client-Seite ist es in diesem Fall nicht sonderlich relevant, aber auf Server-Seite kann dies zu Leistungseinbußen führen da sich für diesen der Rechenaufwand gemeinsam mit der Anzahl der verbundenen Clients erhöht. Desweiteren ist zu beachten das Komprimierung nicht immer sinnvoll eingesetzt werden kann. Handelt es sich beispielsweise um eine geringe Datenmenge, fällt die Kompressionsrate niedriger aus, als bei einerer größeren Menge an Daten.

Die Idee hinter verlustbehafteter Kompression ist ein Verringern der Daten durch Verwerfen von Information, die entweder keinen oder nur geringen Einfluss auf das Allgemeinbild hat. Im Falle eines browserbasierten Online-Multiplayer-Spiels kann dies zum Beispiel geschehen in dem bestimmte Werte (z.B. die Position des Spielers) innerhalb eines sinnvollen Rahmens gerundet werden. Dies betrifft nur den Client, da der Server hier einfach nur die ihm gegebenen Daten weiterverarbeitet. Ausnahmen würden verlustbehaftete Kompressionsalgorithmen darstellen, welche beidseitiges de- und encoden erfordern.

Interne und externe Kompression
Interne Kompression bedient sich eine der vorher erwähnten Verfahren. Die Umsetzung externer Kompression kann erfolgen indem nur die Änderung Delta der jeweilig interessanten Werte übertragen wird. Bleibt ein Spieler beispielsweise am selben Ort stehen, wird im nächsten Update die Positionsangabe ausgelassen. Erst wenn er sich wieder bewegt, wird die Veränderung zur zuletzt bekannten Position übermittelt. Wichtig ist hierbei das sämtliche Daten ankommen und in der richtigen Reihenfolge verarbeitet werden. Im simpelsten Fall kann der Client einfach eine persistente Verbindung mit einem TCP Socket am Server aufbauen und dieser vollkommen vertrauen. Allerdings ist es ratsam dennoch zusätzliche Schutzmechanismen einzusetzen. So kann man die Updates mit fortlaufenden Nummern markieren, welche dann vom Client und vom Server über-

prüft werden, um gegebenenfalls verloren gegangene Updates nachzufordern. Dementsprechend müssen dann allerdings auch ältere Updates zumindest eine Zeitlang gespeichert bleiben um gegebenfalls darauf zugreifen zu können. Wie weit eine solche Vorgehensweise Sinn macht, hängt auch von der Art des Spiels ab. Wenn sich im Spielverlauf nur wenige Sachen gleichzeitig verändern macht diese Technik weit mehr Sinn als wenn das Spiel höchstdynamisch ist und alle Spielobjekte und Spieler ständig in Bewegung sind.

Aggregation von Nachrichten
In der Umsetzung wird die Aggregation von Nachrichten je nach Vorgehensweise mittels eines Timers für die Zeit, einem Zähler für die Anzahl der Nachrichten oder beidem erfolgen. Um die einzelnen Updates aus den Daten auszulesen, müssen diese auch seperat auslesbar sein. Verwendet man JSON können die einzelnen Nachrichten jeweils ein Objekt darstellen auf welches gezielt zugegriffen werden kann. Arbeitet man mit XML Dokumenten können vordefinierte Nodes der Gesamtstruktur jeweils eine Nachricht darstellen. Wie sinnvoll diese Methode angewandt werden kann, hängt von der Art des Spiels ab.

3.2.2 Dead Reckoning

Dead Reckoning dient der Reduzierung der benötigten Bandbreite, indem man die Updates seltener schickt und der Game State zwischen den Updates vom Client berechnet wird. Verwendet man eine nicht-zuverlässige Verbindung wie beispielsweise UDP, kann Dead Reckoning helfen verloren gegangene Pakete auszugleichen. Wird diese Methode angewandt, können die Updates neben der derzeitigen Situation auch Informationen enthalten, welche die Berechnung zukünftiger Game States erleichtert. Da es vorkommen kann, dass der berechnete Game State und der tatsächliche Game State nicht vollkommen ident sind, müssen hier korrektive Maßnahmen ergriffen werden sobald dieser Unterschied durch ein Update bekannt wird. Eine sofortige Umänderung auf den tatsächlichen Game State kann sich negativ auf das Spielgefühl auswirken. Dementsprechend ist in der Regel eine schrittweise Annäherung des lokalen, berechneten Game State an den kollektiven, tatsächlichen Game State notwendig. [vgl. 3, Seite 191][vgl. 2, Seite 92]

Ein weiterer Grund für die Verwendung von Dead Reckoning ist die Steigerung der Responsiveness. Der Client muss nicht zwingend auf die Updates vom Server warten, sondern kann das Spiel weiterlaufen lassen als ob es sich um einen lokalen Multiplayer handeln würde. Hier zeigt sich wieder das Problem zwischen Responsiveness und Konsistenz. Die gewonnene Responsiveness geht auf Kosten der Konsistenz, da nicht mehr alle Spieler jederzeit denselben Game State haben. [vgl. 2, Seite 86f]

Im Wesentlichen besteht Dead Reckoning aus folgenden Schritten:

1. Der Client erhält ein Update das aktuellen kollektiven Game States

2. Der lokale Game State wird Schritt für Schritt an die im Update enthaltenen Informationen angeglichen
3. Die Informationen werden genutzt um bis zum nächsten Update die Positionen des Gegners zu berechnen
4. Zurück zu Schritt 1 [vgl. 3, Seite 191ff][vgl. 2, Seite 86ff]

Um den Game State im voraus zu berechnen wird auf einfache Physik zurückgegriffen. Weiß man beispielsweise die letzte Position eines Objektes und dessen Geschwindigkeit, inklusive Richtung, so können daraus die nächsten Positionen berechnet werden. Lässt man als zusätzlichen Faktor die Beschleunigung einfließen, wird die Berechnung noch genauer. Diese Vorgehensweise lässt sich auch bei anderen Eigenschaften des Objektes anwenden, wie beispielsweise der Neigungswinkel eines Flugzeugs. Je länger keine Updateinformationen zur Darstellung verwendet werden und der Game State nur berechnet wird, desto ungenauer wird die Vorhersage. Dementsprechend wird ein Schwellenwert für die mögliche Ungenauigkeit definiert, bei dessen Überschreiten ein Update eingeleitet wird (sh. Abb. 3.3. Je großzügiger dieser Schwellenwert definiert ist, desto mehr driftet der lokale vom kollektiven Game State ab, aber desto weniger Updates werden benötigt. Bei einem niedrigen Schwellenwert verhält es sich dementsprechend genau umgekehrt. [vgl. 2, Seite 90ff]

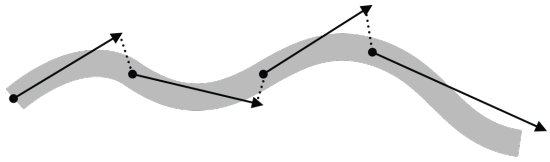

Abbildung 3.3: Dead Reckoning und korrektive Maßnahmen

Die mögliche Ungenauigkeit der Vorhersage hängt auch von der Art des Objektes ab. Die Bewegung eines Objektes mit hoher Trägheit lässt sich genauer bestimmen als die Bewegung eines unberechenbaren Objektes wie es beispielsweise ein Spieler in einem First-Person Shooter darstellt. Dementsprechend muss abhängig von der Art des Spiels und den jeweiligen Spielelementen abgewogen werden, in welcher Form Dead Reckoning eingesetzt wird. [vgl. 2, Seite 91f]

Da es mit hoher Wahrscheinlichkeit zu Differenzen zwischen dem lokalen und dem kollektiven Game State kommen wird, muss hier eine Angleichung stattfinden. Die simpelste Variante ist es, den lokalen Game State sofort nach Erhalt eines Updates an den kollektiven Game State anzupassen. Dies hat jedoch den Nachteil, dass es zu sprunghaften Änderungen kommen kann, welche die Immersion des Spiels beeinträchtigen würde. Dementsprechend ist eine schrittweise Annährung an den kollektiven Game State notwendig. Im Optimalfall findet diese schnell und unaufällig statt. Um dies zu gewährleisten, muss ein Zeitraum definiert werden, innerhalb dessen die Annäherung abgeschlossen wird. So wird für ein sich bewegendes Objekt nach jedem Update ein neuer Pfad berechnet, der die Position und Bewegungsrichtung der lokalen Instanz mit der Position

und Bewegungsrichtung der kollektiven Instanz gleichsetzt. Da sich dieser Pfad von Update zu Update ändert, ist es empfehlenswert Techniken zur Kurvenanpassung anzuwenden, welche unnatürliche Bewegungen und Richtungsänderungen vermeiden. In manchen Fällen lassen sich jedoch visuelle bzw. logische Störungen nur schwer bis gar nicht vermeiden. [vgl. 3, Seite 193ff]

Trotz der Vorteile welche diese Technik bietet, sollte nicht vergessen werden, dass diese auch sehr rechenintensiv sein kann. Natürlich hängt die benötigte Rechenkraft stark von der Komplexität der jeweiligen Vorgehensweise ab. Wobei die durchdachteren Techniken in der Regel genauere Ergebnisse liefern, was sich sowohl auf den Spielverlauf, als auch die benötigten Netzwerkressourcen positiv auswirkt. Im Endeffekt kommt es jedoch auf das Spiel und dessen Spielelemente selber an, welche Dead Reckoning Vorgehensweise Sinn machen und welche nicht.

Möglichkeiten zur browserbasierten Umsetzung

Da es sich hierbei um eine überwiegend mathematische Problemstellung handelt, fällt es nicht schwer diese Methoden bei browserbasierten Spielen anzuwenden. Da diese Technik die Rechenleistung der Clients benutzt um die Server- und Netzwerklast zu reduzieren, erscheint sie besonders empfehlenswert. Dazu kommt, dass sie dabei helfen kann Verbindungsschwächen gegebenfalls auszugleichen. Zu beachten ist jedoch, dass browserbasierte Technologien meist weniger performant sind als herkömmliche Applikationen (sh. Kapitel 2.3.2). Das Spiel muss dementsprechend ausgelegt werden und von vornherein Restriktionen setzen die eine übermäßige Belastung der vorhandenen Ressourcen verhindern (z.B. begrenzte Anzahl an Spielern).

Die Umsetzung mittels beispielsweise Flash könnte hier auf Basis des Events enterFrame passieren. Dieser Event wird, je nach der Framerate (Bilder pro Sekunde) der Flashapplikation, zwischen 0.01 und 120 Mal Pro Sekunde ausgelöst. Bei einem Flash Spiel kann man in der Regel mit einer Framerate von 25 bis 60 Bildern pro Sekunde rechnen. Das Einfachste wäre, bei jedem neuen Frame ein Update vom Server zu fordern. Das Spiel würde somit 25 bis 60 Mal pro Sekunde auf den Server zugreifen. Unter der Verwendung von Dead Reckoning kommt man jedoch mit einer geringeren Anzahl an Requests aus.

Die praktische Umsetzung von Dead Reckoning kann somit bei browserbasierten Spielen analog zu normalen Online-Multiplayer-Spielen erfolgen. Wie bei der Kompression von Nachrichten wird auch hier Bandbreite gespart, mit dem Vorteil, dass die zusätzliche Rechenlast nur den Client betrifft. Allerdings verschiebt Dead Reckoning das Gleichgewicht zwischen Responsiveness und Konsistenz in Richtung Responsiveness.

3.2.3 Time Manipulation

Time Manipulation Verfahren befassen sich mit dem Thema Latenz, bzw. mit den Latenzunterschieden zwischen den einzelnen Clients.

Aufgrund der unterschiedlichen Latenzzeiten verschiedener Spiele, kann es zu unfairen Situationen kommen. Da ein Spieler mit geringer Latenz den aktuellen Game State früher erfährt als einer mit hoher Latenz, kann dieser schneller reagieren. Ebenso wenn beide trotz der Unterschiede gleich schnell reagieren, hat der Spieler mit der geringeren Latenz den Vorteil, dass seine Aktionen früher beim Server ankommen und somit verarbeitet werden. Sinn und Zweck von Time Manipulation ist es hiermit, genau Latenzunterschiede auszugleichen. [vgl. 2, Seite 93]

Um diese Problematik zu entschärfen gibt es verschiedene Techniken, welche auch komplementär angewandt werden können. Eine wird als Time Delay bezeichnet und gibt den Spielern mit niedriger Latenz ein Verzögerungshandicap. Die Zweite ist Time Warp, mit der Spielern mit hoher Latenz entgegengekommen wird[vgl. 2, Seite 93]. Zusätzlich können auch Local Perception Filter eingesetzt werden. Diese Technik dient jedoch mehr der erwartungsgemäßen Darstellung des Spielgeschehens als der Fairness. [vgl. 3, Seite 196f]

Time Delay
Bei Time Delay wird versucht durch eine Verzögerung der Kommunikation von Spielern mit niedriger Latenz zu gewährleisten, dass bei allen Spielern dieselbe Verzögerung vorherrscht. Erhält der Server ein Update von einem Spieler mit geringer Latenzzeit, so zögert er die Verarbeitung des Spieler Inputs soweit hinaus, dass die Latenz der anderen Spieler ausgeglichen wird. Ähnlich schickt der Server seine Updates an Spieler mit niedriger Latenz später aus als an Spieler mit hoher Latenz. Diese Verzögerung kann allerdings auch von der Client-Seite initiiert werden, indem der Client die Updates welche er vom Server erhält nicht sofort, sondern zeitlich verzögert verarbeitet. Bei der Wahl des jeweiligen zeitlichen Buffers muss jedoch darauf geachtet werden, die Verzögerung konstant zu halten. Eine häufige Veränderung der zeitlichen Verzögerung wirkt sich negativer auf den Spielverlauf aus, als eine etwas längere, aber konstante Verzögerung. Eine Anpassung des Buffers an die jeweiligen Latenzverhältnisse sollte nur gelegentlich vorgenommen werden, mit einem Abstand von mindestens 10 Sekunden. Was bei Time Delay nicht vergessen werden darf, ist, dass die gewonnene Fairheit auf Kosten der allgemeinen Responsiveness geht. [vgl. 2, Seite 93f]

Time Warp
Der Time Warp arbeitet auf der Server-Seite mit einem Rollback Mechanismus. Ziel ist es, ohne eine zusätzliche Verzögerung einzubauen und die Responsiveness zu beeinträchtigen, die Ereignisse am Server so ablaufen zu lassen, als ob es keine Latenzunterschiede geben würde. Aufgrund der Latenz der einzelnen Spieler, wird nach Eintreffen der Updates am Server festgestellt welcher Input zuerst

erfolgt ist, unabhängig davon welcher Input den Server als erstes erreicht hat. Wenn also zwei Aktionen im Widerspruch zueinander stehen und eine bereits durchgeführt wurde, weil das Update aufgrund geringerer Latenz früher beim Server eingelangt ist, so wird diese Aktionen rückgängig gemacht. [vgl. 2, Seite 94f]

Die Funktionsweise von Time Warp kann anhand eines First-Person Shooters dargestellt werden. Spieler A erschießt Spieler B während dieser gerade um die Ecke rennt. Spieler B hat allerdings eine geringere Latenz, so dass er bereits an einer anderen Position in wenn der Input von Spieler A eingelangt. Ohne Time Warp würde dieser Schuss nicht gelten. Da sich der Server jedoch der Latenzzeit bewusst ist, rechnet er die Position des Spielers B um die entsprechende Verzögerung zurück und stellt fest, dass dieser getroffen wurde. Der Treffer wird gezählt und der Input des Spielers B als ungültig befunden. [vgl. 2, Seite 94f]

Um Time Warp anwenden zu können, muss die Latenzzeit sehr genau gemessen werden. Diese kann jedoch anhand der Client Requests festgestellt werden, welche bei Online-Multiplayer-Spielen zu Genüge passieren. Dementsprechend können im Laufe des Spiels Feinadjustierungen und eine Anpassung des Time Warps erfolgen. Diese Technik wird beispielsweise beim Spiel Half-Life 2 angewandt. Ein Aspekt der beachtet werden muss, ist, dass es beim Einsatz dieser Vorgehensweise zu Inkonsistenzen kommen kann. Beispielsweise wenn ein Treffer erst gezählt wird, nachdem der Gegenspieler um eine Ecke geflüchtet ist. Derartige Situation werden jedoch aufgrund der schnellen Natur dieser Spiele nur selten wahrgenommen. [vgl. 2, Seite 95f]

Local Perception Filter

Eine ähnliche Methode die zusätzlich angewandt werden kann, sind Local Perception Filter. Der Fokus liegt hier jedoch weniger auf Fairness, sondern mehr darauf, dass das Spiel auf eine Art und Weise dargestellt wird, welche die Latenzunterschiede nicht erkennen lässt. Hierbei werden zwei Arten von Objekten unterschieden: Spieler, deren Verhalten und Bewegung nicht bestimmbar sind und passive Entitäten, deren Verhalten bestimmt werden kann (beispielsweise da sie der Spielphysik folgen). Der Grundgedanke ist auch hier, dass es aufgrund unterschiedlicher Latenzzeiten zu Diskrepanzen in der Wahrnehmung der Spielwelt kommen kann. Jeder Spieler wird in seiner eigenen Zeitzone betrachtet. Passive Entitäten befinden sich somit immer in der Zeitzone des Spielers der am Nächsten ist, bzw. passen ihre Zeitzone am Weg zwischen zwei Spielern an. Sinn dahinter ist, dass der Spieler der sich einem Objekt am nächsten befindet, am ehesten damit interagiert. Somit kann es zwar passieren, dass Spieler eine unterschiedliche Wahrnehmung desselben Game States haben, diese jedoch mit der eigenen Erwartungshaltung übereinstimmt. [vgl. 3, Seite 196f]

Durch diese Einteilung in unterschiedliche Zeitzonen je Spieler kommt es zu einer Krümmung der Spielzeit. Diese muss die drei folgenden Faktoren erfüllen:

- Der Spieler muss mit Objekten in seinem direkten Umfeld interagieren können

- Der Spieler muss Ereignisse die in seinem weiteren Umfeld passieren in Echtzeit wahrnehmen, wobei diese auch leicht verzögert dargestellt werden können
- Die Krümmung der Spielzeit sollte nicht vom Spieler wahrgenommen werden [vgl. 3, Seite 198]

Auch hier müssen die Latenzzeiten der Spieler genau festgestellt werden können. Ebenso können auch bei dieser Methode Fehler in der Darstellung passieren, beispielsweise wenn ein Spieler das Spiel verlässt und somit die Krümmung der Spielzeit verändert wird. Der eigentliche Nachteil dieser Methode liegt jedoch darin, dass die Spieler nicht direkt miteinander interagieren können. Die Interaktion kann somit nur auf Basis passiver Entitäten erfolgen (beispielsweise Geschoße verschiedener Art). [vgl. 3, Seite 198]

Möglichkeiten zur browserbasierten Umsetzung

All diesen Methoden ist gemein, dass eine Messung der Latenzzeiten der einzelnen Spieler erforderlich ist. Hierbei können Zeitstempel von Nutzen sein. Schickt der Server dem Client einen Request den dieser unmittelbar beantworten muss, kann diese Nachricht mit einem Zeitstempel des Servers versehen werden. Dieser Zeitstempel wird mit der Antwort des Clients an den Server übertragen und der aktuellen Serverzeit verglichen. Die verstrichene Zeit entspricht in dem Fall der Roundtrip Time, von welcher wiederum die Latenzzeit abgeleitet werden kann. Da die Latenzzeit jedoch nicht immer exakt der Hälfte der Roundtrip Time entspricht, ist fraglich ob die daraus gewonnen Werte ausreichend genau sind. Um dies festzustellen, bedarf es jedoch einer Teststellung die außerhalb des Rahmens dieser Arbeit liegen würde. Davon ausgehend, dass die Latenzunterschiede ausreichend genau bestimmt werden können, ergeben sich die folgenden Möglichkeiten der Umsetzung.

Time Delay
Der Einsatz von Time Delay ist naheliegend, da dieser Ansatz im Vergleich zu den Anderen nicht nur simpel umsetzbar ist, sondern auch weniger Rechenkraft benötigt. Die Umsetzung kann auf Server-Seite mit Hilfe eines Timers erfolgen, welcher die jeweiligen Aktionen verzögert. Alternativ kann der Time Delay auf die Client-Seite verlagert werden. Der Server stellt hierbei dem Client die Latenzinformationen zur Verfügung, welcher die Verzögerungen lokal durchführt (z.B. bei eigener geringer Latenz die Updates später an den Server schickt).

Time Warp
Time Warp ist eine größere Herausforderung, da hier vor allem die Server-Seite einer komplexeren Umsetzung bedarf. Um mit Time Warp arbeiten zu können muss auf Server-Seite ein Buffer implementiert werden, in welchem die letzten Aktionen der Spieler gespeichert werden. Dieser Buffer wird benötigt um gegebenfalls ein Rollback durchzuführen, also eine ungültige Aktion rückgängig

zu machen. Unabhängig von der verwendeten Server-Technologie werden vom Server die folgenden Aktionen durchgeführt:

- Latenzzeiten werden gemessen und den jeweiligen Spielern zugeordnet
- Inputs der Spieler werden empfangen
- Inputs werden in einem Buffer gespeichert
- Inputs werden miteinander verglichen und anhand der Latenz in die richtige Reihenfolge gebracht
- Inputs werden ausgeführt und fließen in den Game State ein
- Oder werden im Fall eines Konflikts rückgängig gemacht und der Game State angepasst
- Updates (bzw. Korrekturen) des aktuellen Game States werden an die Spieler ausgeschickt

Da der Server in der Lage sein muss Inputkonflikte zu erkennen, müssen in ihm die entsprechenden Teile der Spiellogik implementiert sein. Während man in vielen Fällen den Server auf eine Vermittlerrolle zwischen den Clients reduzieren kann, bedarf es hier einer komplexeren Logik, die über ein einfaches Prüfen und Weiterleiten der Client-Requests hinausgeht. Dementsprechend mehr Rechenlast liegt auf Seiten des Servers. Natürlich hängt die Komplexität der Server-Seite in diesem Fall auch stark von der Art des Spiels ab und kann gegebenfalls geringer sein. Auch muss ein größerer Bedarf an Rechenleistung nicht immer ein Hindernis darstellen. Allerdings ist es ein Faktor dessen man sich bewusst sein muss.

Eine Verlagerung der zusätzlichen Rechenlast auf Client-Seite ist bei Time Warp wenig zielführend. Um dieses clientseitige Verfahren anzuwenden muss einer der Clients den Game State verwalten um somit als Server zu agieren. Da jedoch trotzdem noch ein Webserver benötigt wird, welcher die Updates zwischen den Clients vermittelt, kommt es hierbei zu einer höheren Latenz. Dies beeinträchtigt die Responsiveness, genau einen der Faktoren die von dieser Vorgehensweise unberührt bleiben soll.

Local Perception Filter
Aufgrund dessen, dass sich ein Local Perception Filter mit der Präsentation des Game States auseinandersetzt, kann dieser auf Client-Seite implementiert werden ohne den Server zusätzlich zu belasten. Die Berechnung der Latenzzeiten geht jedoch nachwievor zu Lasten des Servers, wobei dies im Vergleich zu den anderen beiden Methoden Time Delay und Time Warp einen geringeren Aufwand darstellt. Local Perception Filter können jedoch im Fall von browserbasierten Technologien auch auf Client-Seite rechenintensiv sein. Strebt man eine Implementierung dieser Technik mit Hilfe einer browserbasierten Technologie an, so sollte man diesen Aspekt im Auge behalten.

Egal welche dieser Methoden angewandt werden um Latenzunterschiede zwischen den Spielern auszugleichen - implementiert man die erforderliche Logik auf Client-Seite, können sich für den Benutzer eine neue Möglichkeiten zum Cheaten ergeben. Eine genau Betrachtung der Sicherheit von Spielen und des Themas Cheating Prevention findet im Kapitel 3.3 statt.

3.2.4 Area-of-Interest Filtering

Ziel dieses Verfahrens ist eine Reduzierung der benötigten Bandbreite. Es wird die Tatsache genutzt, dass bei vielen Spielen die Spieler nicht das gesamte Geschehen wahrnehmen.

Einfache Auras
In der simpelsten Variante wird ein Multiplayer-Spiel die Updates eines jeden im Spiel vorhandenen Objektes an sämtliche beteiligten Spieler ausschicken. Allerdings sind diese Updates nicht immer für alle Spieler relevant. So macht es wenig Sinn einem Spieler Updates über aurale Ereignisse zu schicken die außerhalb seiner Hörweite liegen oder über visuelle Informationen die außerhalb seines Blickwinkels sind. Dies bedeutet das mehr Netzwerkressourcen also nötig verwendet werden. Dazu kommt, dass der Client filtern muss welche Ereignisse für ihn relevant sind. Dementsprechend bietet sich eine Optimierungsmöglichkeit an, indem jeder Spieler nur die Updates erhält die für ihn von Interessen sind. Eine Technik die hierfür angewandt wird ist Area-of-Interest Filtering. [vgl. 21, Seite 4f]

Die sogenannten Auras entsprechen in diesem Fall dem räumlichen Umfeld des Spielers, innerhalb dessen er Updates erhält. Innerhalb dieses Bereichs findet Interaktion statt. Auras können für verschiedene Medien unterschiedlich ausgelegt sein. So kann beispielsweise die Aura für Umgebungsgeräusche einen größeren oder kleineren Raum erfassen als die Aura für die visuelle Darstellung. Auch für die geometrische Fläche einer Aura gibt es verschiedene Möglichkeiten. So kann sie kreisförmig sein, quadratisch oder zellenbezogen (sh. Abb. 3.4. Bei letzterem wird der Raum in beispielsweise quadratische Zellen geteilt, wobei jede Zelle die innerhalb eines bestimmten Umkreises des Spielers liegt, als Ganzes in die Aura gefasst wird. Berühren sich die Auras zweier oder mehrerer Spieler, erhalten diese untereinander die jeweiligen Updates. [vgl. 3, Seite 206ff][vgl. 22]

Fokus und Nimbus
Eine Weiterführung dieses Konzepts ist die Unterteilung der Aura in Fokus und Nimbus. Der Fokus ist die Zone innerhalb welcher der Spieler Reize wahrnimmt (beispielsweise das Sichtfeld), der Nimbus beschreibt die Zone innerhalb welcher der Spieler wahrgenommen wird. Ein Spieler erhält von seinem Mitspieler also nur dann Updates, wenn sein Fokus den Nimbus des Mitspielers schneidet. Dies hat zur Folge, dass Spieler sich nicht immer zwangsläufig gegenseitig wahrnehmen, sondern diese Wahrnehmung gegebenenfalls auch einseitig sein kann, beispielsweise wenn ein Spieler hinter dem Anderen steht. [vgl. 3, Seite 208][vgl.

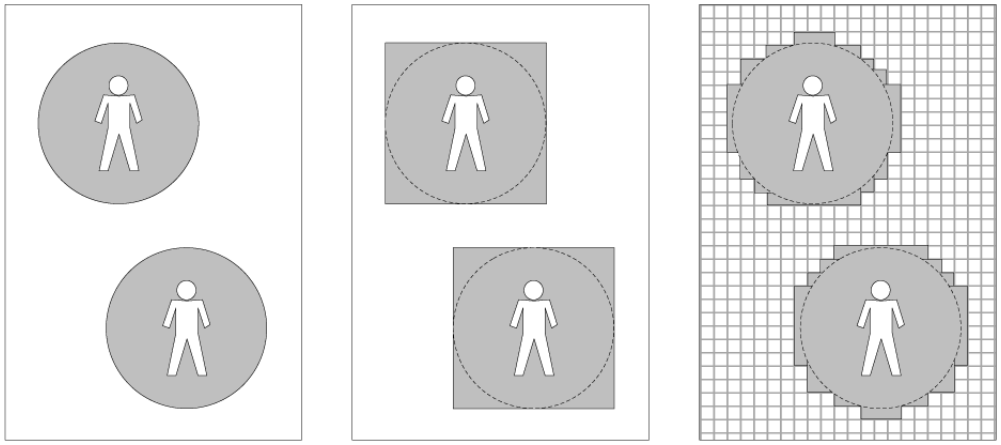

Abbildung 3.4: Auras: (v. links n. rechts) kreisförmig, quadratisch, zellenbezogen

2, Seite 96]

Möglichkeiten zur browserbasierten Umsetzung

Da hier das Einsparen von Bandbreite einen der wesentlichen Beweggründe darstellt, werden derartige Filter am sinnvollsten auf der Server-Seite implementiert. Dies erfordert wiederum entsprechende Rechenleistung, je nach Vorgehensweise.

Einfache Auras
Einfache Auras können relativ simpel implementiert werden. Diese können anhand der dem Server bekannten Positionen der Spieler berechnet werden und somit auch die Überschneidungen derselben. Der Server schickt den Spielern anschließend das jeweilig relevante Subset an Daten. Wie aufwendig die Aura-Berechnungen sind hängt neben der Anzahl der Spieler auch von der jeweilig verwendeten geometrischen Form der Aura ab bzw. ob mehrere Auras berechnet werden (z.B. Aural und visuell) oder nur eine.

Fokus und Nimbus
Verwendet man Fokus und Nimbus wird die Sache wiederum komplexer. Eine Überlegung die hier getroffen werden kann, ist ob sich die Geometrie der Spielwelt auf diese beiden Bereiche der Aura auswirkt oder nicht. Lässt man die Spielwelt außen vor, muss für jede mögliche Überschneidung zwischen Fokus und Nimbus eine Berechnung stattfinden. Das allein macht bereits die doppelte Anzahl an Berechnungen notwendig. Dazu kommt, dass die umfasste Fläche des Fokus in der Regel eine andere ist als die des Nimbus und diese somit separat berechnet werden muss. Bezieht man die Spielwelt mit ein, nimmt der Rechenaufwand weiter zu. Hierbei muss nämlich auch ein Teil der Spiellogik auf der Server-Seite implementiert werden, um zumindest den Einfluss des räumlichen

Umfelds in die Berechnungen miteinfließen zu lassen.

Aufgrund des hohen Leistungsaufwandes sind für browserbasierte Online-Multiplayer-Spiele in den meisten Fällen wahrscheinlich nur simple Auras interessant. Eine Implementierung dieser Technik bietet für bereits geringe zusätzliche Serverbelastung eine Einsparung an Netzwerkressourcen. Wie effizient diese Technik angewandt werden kann hängt auch von der Art des Spiels ab. Ist die Spielwelt so klein, dass sowieso immer alle Spieler miteinander interagieren, wird diese Methode nicht sinnvoll sein.

Ähnlich wie bei Time Manipulation kann auch hier die Auslagerung der Serverlast auf einen der Clients angedacht werden. Allerdings unter denselben Vorbehalten.

3.2.5 Synchrone Simulation

Ziel der synchronen Simulation ist es, mittels einer Reduzierung der benötigten Updates Bandbreite zu sparen. Sie geht hierbei einen ähnlichen Weg wie Dead Reckoning, jedoch mit dem Unterschied, dass synchrone Simulation das gesamte Spielgeschehen als solches betrifft und nicht auf eine Verbesserung der Responsiveness abzielt.

Die Grundlage einer synchronen Simulation sind Spielverläufe die auf allen beteiligten Clients vollkommen gleich verlaufen und somit konsistente Game States aufrecht erhalten. Um dies zu erreichen benötigen alle Spieler diesselben Grundvoraussetzung ab Spielstart. Die Konsistenz wird aufrecht erhalten in dem die Ereignisse des Spiels in bestimmbare und unbestimmbare Ereignisse unterschieden werden. [vgl. 3, Seite 205]

Bestimmbare Ereignisse werden vom Spiel selber ausgelöst und können somit berechnet werden. Beispiele hierfür wären das physikalische Verhalten von Objekten oder von Einheiten welche von Punkt A zu Punkt B einem vordefinierten Pfad folgen. Die Entscheidungen eines künstlichen Spielers sind dann vorhersehbar, wenn bei allen Clients zur Entscheidungsfindung diesselbe pseudo-zufällige Zahl verwendet wird. Da all diese Ereignisse von Beginn an gleich verlaufen und auf Änderungen gleich reagieren, handelt es sich hierbei um Game State Informationen die nicht übertragen werden müssen. [vgl. 3, Seite 205]

Unbestimmbare Ereignisse sind solche, welche vom Spieler initiiert werden und somit nicht berechnet werden können. Da dies die einzigen Ereignisse sind welche ein Spiel in unbestimmbare Bahnen leiten kann, sind sie auch die einzigen die übertragen werden. [vgl. 3, Seite 205]

Ziel dieser Vorgehensweise ist es Netzwerkressourcen zu sparen. Außer der Ausgangssituation, welche an alle Spieler propagiert werden muss, sind nachher nur Updates der Spieler Inputs von Nöten. Da diese Updates jedoch verloren gehen können, sollte die kollektive Konsistenz der Game States in regelmäßigen

Abständen überwacht werden. Um gegebenfalls eine Wiederherstellung der Game States zu gewährleisten, müssen entsprechende Mechanismen implementiert werden. [vgl. 3, Seite 205]

Möglichkeiten zur browserbasierten Umsetzung

Diese Technik kann direkt für browserbasierte Spiele übernommen werden. Abgesehen von den Kontrollmechanismen auf Server-Seite, findet hier der zusätzliche Leistungsaufwand nur am Client statt. Geht man davon aus, dass die Spiellogik zur Gänze am Client implementiert ist, fällt dieser zusätzliche Aufwand noch dazu sehr gering aus. Der Einsatz von synchroner Simulation ist also auch bei browserbasierten Spielen zu empfehlen, vorausgesetzt es handelt sich um ein Spiel wo sie sinnvoll eingesetzt werden kann. Dies betrifft sämtliche Spiele wo neben den Spielern selber auch andere Entitäten existieren, die im Verlauf des Spiels ihre Position oder andere Eigenschaften auf berechnbare Weise verändern.

3.2.6 Zusammenfassung

Wie man sieht gibt es eine Vielzahl an Techniken die angewandt werden können um die Herausforderungen vernetzter Multiplayer-Spiele zu bezwingen. Auch wenn manche Vorgehensweisen nur schwer oder wenig effizient browserbasiert umgesetzt werden können, so lassen sich dennoch alle dieser Ansätze zumindest theoretisch implementieren. Viele der erwähnten Methoden können auch im browserbasierten Bereich sinnvoll eingesetzt werden und sollten somit gegebenfalls in Erwägung gezogen werden. Auch eine Kombination verschiedener Ansätze kann je nach Spiel zielführend sein.

3.3 Cheating und Cheating-Prevention

Auch bei Online-Multiplayer-Spielen muss Sicherheit beachtet werden. Im Gegensatz zu anderen Applikation stellt Cheating hierbei einen wesentlichen Aspekt dar. Vor allem bei Spielen über das Internet ergibt sich hier ein großes Problem. Als Cheats zählen Methoden welche einem Spieler einen unfairen Vorteil verschaffen, der weder von den Entwicklern des Spiels noch von den Anbietern des Spieleservers vorgesehen wurden. [vgl. 23, Seite 1]

Rein technisch gesehen können Cheats auf der Server-Seite, Client-Seite oder dem dazwischen liegenden Netzwerk erfolgen. Darüber hinaus gibt es jedoch auch andere Möglichkeiten sich unfaire Vorteile zu verschaffen. Im nächsten Abschnitt werden verschiedene Arten von Cheats sowie mögliche Sicherheitsvorkehrungen betrachtet. Letztere werden vor Allem aus Sicht browserbasierter Technologien betrachtet.

3.3.1 Serverseitige Cheats

Die naheliegendste und einfachste Art von Cheats auf Server-Seite, ist der Mißbrauch von Administrationsrechten. Der Betreiber des Spieleservers oder eine Person die für jenen verantwortlich ist, kann beispielsweise Einträge in der Datenbank verändern, um sich selber im Spiel einen Vorteil zu verschaffen. Aus rein technischer Sicht lässt sich diese Möglichkeit nicht ausschließen ohne die Wartbarkeit des Spieleservers maßgeblich zu beeinträchtigen. Man kann diesem Problem jedoch vorbeugen, indem man darauf achtet welche Zugriffsrechte jeder interne Person zur Verfügung gestellt wird [vgl. 23, Seite 4]. Hat ein Angreifer sich auf nicht legitime Weise Zugang zum Server verschafft, kann dieser ebenfalls die im zugänglichen Rechte missbrauchen. Um dies zu vermeiden muss der Server als solches ausreichend abgesichert sein.

Mangelhafte Authentifizierung von Client-Requests aber auch Spieleservern stellen eine Sicherheitslücke dar. Fehlt ein Mechanismus zur Authentifizierung von vertrauenswürdigen Spieleservern, so kann ein Angreifer dies Nutzen um mit einem eigenen Spieleserver Zugriffsdaten von Spielerkonten zu stehlen. Bei browserbasierten Spielen kann man dies vermeiden, indem der Spieleserver von derselben Website zu Verfügung gestellt wird wie das Spiel selber. Alternativ kann am Client eine Liste vertrauenswürdiger Server gespeichert werden, welche gegebenenfalls ein Update erfährt. [vgl. 23, Seite 4]

Werden Client-Requests nicht hinreichend authentifiziert, kann auch dies zu einem Spielerkonto-Diebstahl führen. Um dies zu vermeiden sollte ein Benutzer nach dem aktuellen Passwort gefragt werden bevor es durch ein neues ersetzt wird. Selbiges gilt für ähnliche, kritische Situationen. [vgl. 23, Seite 4]

Bei Spielen in welchen sich ein Spieler in ein eigenes Konto einloggt, ist es auch wichtig das Passwort System selber zu schützen. So könnte ein Dictionary Attack durchgeführt werden, bei welchem verschiedene Passwörter automatisch durchprobiert werden. Derartigen Attacken kann entgegengesteuert werden, in dem der Server zwischen den einzelnen Login Versuchen eine kurze Verzögerung einbaut. Weiters denkbar wäre ein Einsatz von Captchas um automatische Skripts von vornherein abzuwehren. Das automatische Einfrieren eines Benutzerkontos ist zwar auch eine Möglickeit, allerdings mit Vorsicht zu genießen da dies wiederum einem Angreifer ermöglicht Benutzerkonten gezielt einzufrieren. [vgl. 2, Seite 110]

3.3.2 Clientseitige Cheats

Viele Cheats welche von der Client-Seite ausgehen, haben ein zu hohes Vertrauen in den Client als Ursprung. Dadurch, dass ein Spieler vollen Zugriff auf sein eigenes System hat, kann er hier Modifikationen vornehmen, welche das Spiel zu seinen Gunsten verändern. Der Grundsatz der hier befolgt werden muss ist der, dass dem Client nicht vertraut werden kann.

So kann er die eigene Software bzw. sein eigenes System so manipulieren, das es zusätzliche Informationen über den Game State preisgibt. Beispielsweise kann ein Cheater bei einem Echtzeit-Strategiespiel das Programm so verändern, dass er Gebiete sehen kann, welche er noch nicht erkundet hat. Oder die Grafiktreiber können bei einem First-Person Shooter so manipuliert werden, dass der Cheater durch Wände hindurch sehen kann. Dies ist eine schwer nachzuweisende Methode, da nicht das Spiel selber verändert wird, sondern das darunter liegende System. Um gegen derartige Cheats vorzugehen, würde es sich anbieten zu vermeiden, dass der Spieler diese Art von Informationen überhaupt erhält. Allerdings steht dies in Konflikt mit möglichen Optimierungstechniken (beispielsweise synchrone Simulation). Dementsprechend ist dies schwer umzusetzen ohne die Serverlast rapide zu steigern und das Spielgefühl zu beeinträchtigen. [vgl. 23, Seite 2f]

Wenn auch kein absolut zuverlässiger Ansatz, so ist es auf alle Fälle empfehlenswert derartige Spielinformationen zu Verschlüsseln und gut im Arbeitsspeicher des Clients zu verstecken. Dazu können vom Server die Inputs der Spieler hinsichtlich verdächtigen Verhaltens überprüft werden. Beispielsweise wenn er eine Basis angreifen möchte, deren Position ihm noch nicht bekannt sein sollte da er das entsprechende Gebiet noch nicht erkundet hat. Auf ähnliche Art und Weise können auch die Game States selber auf Diskrepanzen überprüft werden [vgl. 3, Seite 221].

Auch bei browserbasierten Spielen können derartige Cheating Prevention Maßnahmen getroffen werden. Game State Informationen können beispielsweise mit einem eigenen Schlüssel chiffriert werden, mit welchem das Programm die Daten erst dann entschlüsselt, wenn sie benötigt werden. Auch das Überprüfen des Spieler Inputs hinsichtlich verdächtigem Verhaltens kann innerhalb eines gewissen Rahmens umgesetzt werden. Allerdings gilt es hier zu beachten, das gegebenfalls Teile der Spiellogik auch am Server implementiert sein müssen, was wiederum die Rechenlast erhöht.

Automatisiertungsprogramme, auch Bots genannt, können einem Cheater helfen mit besseren Fähigkeiten zu spielen, als er eigentlich hat. Bots gibt es in verschiedenen Variationen. Bei MMORPGs beispielsweise, können Bots statt dem Spieler einfache, jedoch langwierige Aufgaben ausführen und so dabei helfen den Charakter bzw. seine Ausstattung zu verbessern ohne das der Spieler etwas dazu tun muss. Bots können auch in Form von Macros auftreten, die genutzt werden um bestimmte Bewegungsabläufe (z.B. ausweichen, drehen, schießen) oder Tastenkombinationen (z.B. bei Kampfspielen) automatisch ablaufen zu lassen. Eine andere Art von Bots die überwiegend bei First-Person Shootern vorkommt sind Aimbots. Diese helfen dem Cheater beim Zielen und Schießen. Hierbei ändert der Bot den Input des Spielers so um, dass jeder Schuss trifft. [vgl. 2, Seite 112]

Bots stellen auch für browserbasierte Spiele eine Gefahr dar. In der einfachsten Variante werden sie in Form von Macros auftreten, welche dem Spieler bestimmte Tastenkombinationen erleichtern. Das Erkennen einer derartigen Manipula-

tion des Spieler Inputs ist bei einer rein browserbasierten Lösung in der Regel für das Programm selber unmöglich.

Browserbasierte Multiplayer-Spiele haben den Vorteil, dass der Code zwar am Client ausgeführt wird, jedoch am Server liegt. Dementsprechend kann das Spiel selber nicht verändert werden. Allerdings könnte man beispielsweise die jeweilige Laufzeitumgebung so manipulieren, dass das Verhalten des Spiels on-the-fly verändert wird. Eine derartige Manipulation ist für die Applikation nicht umgehbar, da sie innerhalb ihrer Laufzeitumgebung gefangen ist. In manchen Fällen haben browserbasierte Applikationen jedoch die Möglichkeit Informationen über die eigene Laufzeitumgebung auszulesen. Gegebenenfalls kann dadurch eine Manipulation erkannt werden und die Applikation sich selber sperren.

Eine weitere Angriffsfläche bietet der Arbeitsspeicher. Hilfsprogramme können verwendet werden um auf die Daten im Arbeitsspeicher zuzugreifen, Werte zu ändern und somit den Game State zu manipulieren. Eine Verschlüsselung bzw. ein Verstecken der Daten ist hier nicht gezielt möglich, da die Verwaltung des Arbeitsspeicher von der jeweiligen Laufzeitumgebung und nicht der Applikation selber übernommen wird. Allerdings kann die Manipulation des Game States durch regelmäßiges überprüfen auf verdächtige Veränderungen erschwert werden. Darüber hinaus kann man kritische Werte wie Geld oder die Punkteanzahl mit einem zusätzlichen Schlüssel in Form eines Hashwertes oder ähnlichem absichern.

Eine simple Art des Cheaten, ist das Verlassen des Spiels sobald man merkt, dass man am Verlieren ist. Hierbei wird das Spiel frühzeitig beendet oder die Netzwerkverbindung unterbrochen. Diese Praxis wird angewandt um die eigene Spielstatistik zu schönigen, da es bei Beendigung des Spiels weder einen Gewinner noch einen Verlierer gab. Allerdings verschafft sich ein Cheater damit nicht nur selber einen Vorteil, sondern verweigert seinem Gegenspieler den Sieg. Je mehr Leute innerhalb einer Spielercommunity diese Praxis anwenden, desto mehr verliert das Spiel an Reiz, da man davon ausgehen kann das viele Runden nie beendet werden. Dementsprechend sollte ein Server nicht nur die gewonnenen und verlorenen Spiele aufzeichnen, sondern auch jene die außertürlich beendet wurden. Für den Umgang mit diesen Daten gibt es mehrere Alternativen. Eine Möglichkeit besteht darin, den Spieler dessen Verbindung getrennt wurde automatisch als Verlierer zu werten. Dies hat allerdings jenen Nachteil, dass ein Cheater somit eine Denial of Service Attacke auf seinen Gegenspieler ausüben könnte, sodass dieser die Verbindung und somit das Spiel verliert. Die andere Möglichkeit ist es, das Verhältnis zwischen zu Ende gespielten Spielen und abgebrochenen Spielen öffentlich zugänglich zu machen oder in eine Art Spielerwertung fließen zu lassen. Dadurch wird ein gezieltes Abbrechen der Verbindung gleich weniger attraktiv. [vgl. 23, Seite 3] [vgl. 2, Seite 110]

Da bei browserbasierten Spielen in der Regel alles über einen zentralen Server abgehandelt wird, muss man sich hier wenig Sorgen machen, dass einem Cheater die IP seines Gegenspielers bekannt ist (außer er ist auch über ein anderes Programm mit seinem Gegenspieler verbunden). In diesem Fall lässt sich ge-

trost ein Verbindungsabbruch als verlorenes Spiel betrachten, wobei es trotzdem erstrebenswert ist abgebrochene Spiele seperat zur erfassen.

3.3.3 Cheats auf Netzwerkebene

Viele Cheats auf Netzwerkebene arbeiten damit, dass sie die Datenpakete welche vom Client ausgeschickt werden verändern. So gibt es Aimbots die auf dieser Ebene arbeiten und die Pakete entsprechend manipulieren oder Scripts die bestimmte Requests mehrmals schicken, um dem Cheater eine Aktion öfter und schneller ausführen zu lassen als eigentlich vorgesehen (beispielsweise Schießen). Umgekehrt kann ein vom Cheater eingesetzter Proxy auch bestimmte Pakete vom Server verwerfen lassen. Auf diese Weise kann er beispielsweise verhindern das Schaden den er nimmt auch gezählt wird. [vgl. 2, Seite 116] [vgl. 3, Seite 214f]

Um die Manipulation der Datenpakete zu erschweren können Schlüssel in Form einer Checksum mitgeschickt werden. Dieser Ansatz lässt sich auch direkt für browserbasierte Spiele übernehmen. Bei dieser Methode schickt das Programm auf der Client-Seite seine Updates in Verbindung mit einem Hashwert des Inhalts. Damit dieser Hashwert schwerer nachvollzogen werden kann, wird gemeinsam mit dem Inhalt ein versteckter Wert gehasht, welcher nur am Server und Client bekannt ist. Der Server kann somit erkennen ob das Update manipuliert wurde [vgl. 3, Seite 215]. Die Algorithmen MD5 und SHA11 bieten sich für so eine Vorgehensweise auch bei browserbasierten Spielen an.

Allerdings gibt es einige Wege wie ein Cheater dieser Sicherheitsvorkehrung entgegensteuern kann. So kann er nachwievor dieselben Pakete mehrmals schicken, weil die Checksum in diesem Fall dieselbe bleibt. Er kann den Verschlüsselungsalgorithmus reverse engineeren und so beliebige, gültige Datenpakte bauen. Zusätzlich besteht je nach Algorithmus die Chance auf eine Hashwert-Kollision. Dies ist dann der Fall, wenn zwei unterschiedliche Datenpakte dieselbe Checksum haben. [vgl. 3, Seite 215]

Um die Checksum der Datenpakete sicherer zu machen, kann zusätzlich ein zufälliger Wert eingestreut werden, sodass auch Pakete mit demselben Inhalt unterschiedliche Hashwerte haben. Zusätzlich kann die Analyse der Datenpakete erschwert werden indem auch ein kleiner Anteil an nutzlosen Daten eingebaut wird. [vgl. 3, Seite 215]

Desweiteren kann am Server eine logische Überprüfung des Spieler-Inputs sowie dessen Game States stattfinden. Auf diese Weise können verdächtige Inputs des Spielers überprüft und gegebenfalls verworfen werden (beispielsweise wenn er schneller schießt als vorgesehen). Auch dem Problem der Paketfilterung auf Client-Seite kann hiermit entgegengesteuert werden. Wenn ein Cheater beispielsweise Schadenspakete ausfiltert, kann der Server seinen Status trotzdem mitverfolgen und Inputs des Spielers für ungültig befinden, sobald dieser ge-

storben ist. Analog ist eine browserbasierte Umsetzung dieser Vorgehensweisen möglich.

3.3.4 Sonstige Arten von Cheats

Es gibt auch Cheats die nicht unbedingt eine gezielte Veränderung der technischen Umgebung benötigen. Eine Vorgehensweise die hierzu zählt ist das Ausnutzen von Bugs oder sonstigen Schwächen der Software. Wird ein derartiger Bug entdeckt, kann ein Cheater dadurch einen erheblichen Vorteil gewinnen [vgl. 23, Seite 4]. Selbst wenn der Exploit auch anderen Spielern bekannt ist, die ihn theoretisch zum Ausgleich ebenfalls benutzen könnten, verändert dies das Gleichgewicht des Spiels oft auf eine Art welche das Spielgefühl insgesamt beeinträchtigen kann. Ein Beispiel aus der näheren Vergangenheit ist der Shield Exploit des Third-Person Shooters Gears of War 2. Bei Gears of War 2 stehen dem Spieler in manchen Levels Schilder zu Verfügung, welche dieser aufheben kann um dahinter in Deckung zu gehen. Damit das Schild jedoch keinen unfairen Vorteil bietet, kann ein Spieler der dieses verwendet nur seine Pistole benutzen. Bevor dieser Bug gepatched wurde, konnten Cheater, welche den Shield Exploit benutzten, statt der Pistole eine zwei-händigen Schusswaffe (z.B. Maschinengewehr, Schrotflinte) verwenden [vgl. 28].

Ein Grenzgebiet des Cheaten ist es Eigenschaften der Spielmechanik auf unfaire Art und Weise zu benutzen. Da es sich weder um das Ausnutzen von Bugs oder um eine Manipulation des Verhaltens der Applikation handelt, sind dies keine Cheats im eigentlichen Sinne sondern eher als unsportliches Verhalten zu verstehen. Ein Beispiel aus dem Bereich der First-Person Shooter ist Spawncamping. Spawns sind jene Punkte eines Levels, auf denen Spieler die gestorben sind wieder ins Spiel einsteigen und starten. Ein Spawncamper lauert an ebenjenen Punkten auf gegnerische Spieler, um ihnen in den Rücken zu fallen bevor sie ihm gefährlich werden können (beispielsweise durch bessere Waffen oder Rüstung). Da derartiges Verhalten ein Ungleichgewicht der Chancen verursacht, ist es wichtig diesen Aspekt beim Game Design zu beachten. Oft fallen diese Probleme erst nach Veröffentlichung des Spiels auf, was bedeutet das Updates entwickelt werden müssen um das Spiel ausgeglichen zu halten und so den Spielspass auf lange Sicht zu sichern. Um Spawncamping entgegenzusteuern kann man das Spiel beispielsweise so designen, dass neu gespawnte Spieler für kurze Zeit weder Schaden erhalten noch verursachen können. Dieser Zeitraum sollte für den Spieler ausreichend sein um sich gegebenfalls in Sicherheit zu bringen. [vgl. 23, Seite 7]

3.3.5 Allgemeine Gedanken zum Thema Cheating Prevention

Cheating Prevention und ausgeglichenes Game Play gehören zu den wichtigsten Faktoren die beachtet werden müssen wenn ein Spiel über längere Zeit interessant bleiben soll. Bei Cheating Prevention ist es wichtig ein Gleichgewicht

zwischen Sicherheit und Spielbarkeit zu schaffen. Da die Daten nur über eine kurze Zeitspanne sicher bleiben müssen, werden bei Online-Multiplayer-Spielen nicht so strikte Sicherheitsvorkehrungen benötigt wie bei anderen, kritischeren Applikationen. Gleichzeitig jedoch müssen Verfahren angewandt welche die vorhandenen Ressourcen schonen. [vgl. 3, Seite 224]

Vor allem gilt bei Online-Multiplayer-Spielen, wie auch bei Online-Applikationen im Allgemeinen, dass dem Client nicht vertraut werden darf. Dementsprechend muss darauf geachtet werden, welche für das Game Play kritische Aspekte am Client ablaufen. Auch der Input der Clients sollte nicht ohne entsprechende Überprüfung verarbeitet werden. Natürlich gilt auch hier wieder das Gleichgewicht zwischen Sicherheit und Spielbarkeit aufrecht zu erhalten. [vgl. 23, Seite 2f] [vgl. 2, Seite 116]

Kapitel 4

Konzeption und Umsetzung eines browserbasierten Online-Multiplayer-Spiels

Dieses Kapitel widmet sich der praktischen Umsetzung der in dieser Arbeit gewonnenen Erkenntnisse. Mittels der in den vorigen Kapiteln betrachteten Techniken wird ein browserbasiertes Online-Multiplayer-Spiel erstellt. Dabei handelt es sich um ein rundenbasiertes Strategiespiel bei welchem verzögerungsarme Echtzeit-Elemente verwendet werden um die Immersion des Spiels zu verbessern. Als erstes wird das Spiel und dessen technischen Anforderungen vorgestellt, sowie die verwendeten Technologien. Die technische Umsetzung wird clientseitig und serverseitig betrachtet, ebenso die Kommunikation dazwischen. Der Schwerpunkt liegt auf jenen Elementen, die für Multiplayer-Spiele wesentlich sind und nicht auf der Spiellogik selber. Abschließend erfolgt eine Bewertung des Projekts und ein Ausblick auf zukünftige Erweiterungsmöglichkeiten.

4.1 Ziele

Anhand des Beispielprojektes soll betrachtet werden wie Online-Multiplayer Spiele mittels browser- und web-basierter Technologien entwickelt werden können. Die in Kapitel 3 betrachteten Vorgehensweisen sind allgemein gültige Konzepte, wodurch eine praktische Umsetzung nahezu unabhängig von der verwendeten Technologie möglich sein sollte.

Die durch den Praxisbezug gewonnenen Erkenntnisse sollen jedoch auch bei der browserbasierten Umsetzung anderer Vorgehensweisen behilflich sein. Das hier entwickelte Beispielprojekt könnte somit als Basis für zukünftige, ähnliche Projekte dienen.

4.2 QuaRkZ, ein browserbasiertes Flash-Spiel

Das Flash-Spiel QuaRkZ wurde 2005 vom Autor dieser Arbeit veröffentlicht und ermöglicht das Spielen gegen einen Computergegner oder lokalen Gegenspieler. Ziel des Fallbeispiels ist die Implementierung eines Multiplayer Modus über das Internet.

Bei QuaRkZ übernimmt der Spieler die Rolle eines Quarks, ein subatomares Partikel welches anhand eines geistähnlichen Avatars dargestellt wird. Das Spiel findet auf einem quadratischen Spielfeld statt, welches in Felder verschiedener Ladungen unterteilt ist. Die Spieler setzen abwechselnd ein Teilchen in ihrer jeweiligen Farbe. Je nachdem welcher Spieler das letzte Teilchen platziert hat, ist das Feld Blau oder Rot geladen und hat je nach Anzahl der platzierten Teilchen einen Wert zwischen 0 und 8. Überschreitet eines der Felder einen bestimmten Schwellenwert (in den Ecken entspricht dieser Wert 3, am Rand 5 und auf allen anderen Feldern 8) kommt es zu einer nuklearen Reaktion. Hierbei übernehmen die angrenzenden Felder die farbliche Ladung des überladenen Feldes und gehen somit in den Besitz des jeweiligen Spielers über. Gewonnen hat jener Spieler, welcher als Erster eine bestimmte Anzahl an Teilchen unter seiner Kontrolle hat.

Abbildung 4.1 Zeigt die Spielansicht. Zu erkennen sind die Avatare der Spieler, deren Punktezahl sowie das Spielfeld selbst. Die Ladungswerte der Felder werden zusätzlich zu den sichtbaren Teilchen auch numerisch dargestellt. Die Felder, auf welchen Blitze zu sehen sind, stehen kurz vor einer nuklearen Reaktion. Sie haben ihren von der Position am Spielfeld abhängigen Schwellenwert erreicht.

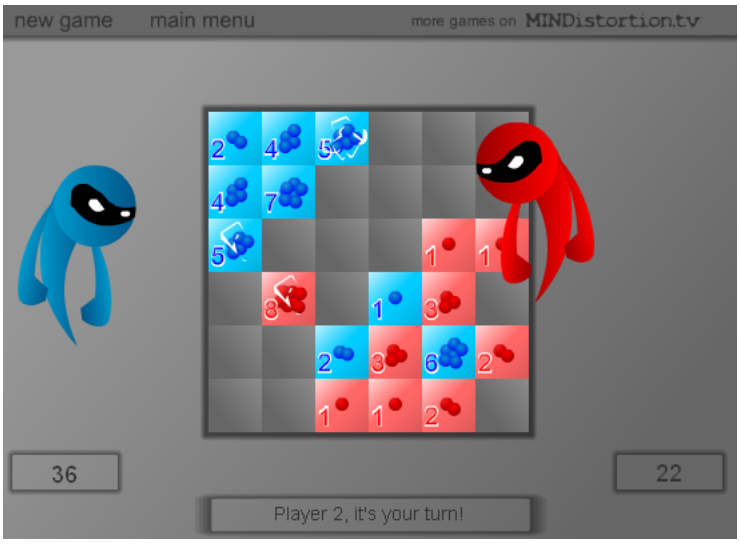

Abbildung 4.1: QuaRkZ Spielansicht

Jener Spieler der an der Reihe ist, steuert mit seiner Maus die Bewegung seines

Avatars. Diese Bewegung wird im Online-Multiplayer Modus in verzögerungsarmer Echtzeit an den jeweiligen Gegner übertragen.

4.3 Anforderungen

Um Spielern zu ermöglichen miteinander in Kontakt zu treten und gegeneinander anzutreten, muss ihnen eine Möglichkeit gegeben werden zu sehen welche potentiellen Spieler gerade online sind. Dementsprechend ist eine Art Presence Management System von Nöten, welches die Online Benutzer und deren derzeitigen Status verwaltet. Benutzer die sich in das Spiel einloggen, müssen von dem System erkannt und erfasst werden. Ebenso muss ein Spieler aus der Online Liste entfernt werden, sobald er sich ausloggt oder die Verbindung verliert.

Die Mausbewegung des Benutzers muss während des Spiels zeitnah übertragen werden. Der Spielverlauf wird zwar nicht wesentlich beeinflusst, wodurch Zeit kein kritisches Element darstellt (wie beispielsweise bei First Person Shootern). Dennoch sollte die Übertragung schnell geschehen um die Immersion zu verbessern. Denn dadurch, dass der Spieler die Bewegungen des Gegenspielers beobachten kann, wird er daran erinnert das er gegen einen realen Menschen spielt. Auch eine Steigerung des Gefühls der Interaktivität wird erreicht. Somit handelt es sich um Information, die zwar in hoher Frequenz propagiert werden muss, deren kollektive Konsistenz jedoch vernachlässigbar ist.

Einzelne Spielzüge müssen seltener übertragen werden als Mausbewegungen. Dafür müssen sie verlässlich übertragen werden um die Spielsituation für beide Spieler konsistent zu halten. Derartige Datenpakete dürfen weder verloren gehen noch manipuliert werden.

4.4 Die verwendeten Technologien

Auf der Client-Seite wird Flash eingesetzt. Diese Technologie bietet zwei wesentliche Vorteile:

- Sie ist stark auf die Entwicklung von Applikationen mit multimedialen Inhalten und dynamischer grafischer Oberfläche spezialisiert. Ein Vorteil, der vor allem bei der Entwicklung von Spielen zur Geltung kommt.
- Flash ermöglicht die Verwendung von Sockets wodurch eine persistente Verbindung mit einem Spieleserver hergestellt werden kann. Der Server kann dem Client somit aktiv Updates zur Verfügung stellen, ohne dass dieser in regelmäßigen Abständen den Server abfragen muss. Dadurch ist schnellere Kommunikation gegeben und Bandbreite wird gespart.

Da der zu erweiterende Quellcode noch in ActionScript 2.0 und nicht in der aktuellen Version 3.0 geschrieben wurde, wird die Client-Seite dieses Projekts in ActionScript 2.0 programmiert und läuft ab Version 7 des Flash Plugins.

Die Server-Seite basiert auf PHP 5.2.x. Ebenso wie bei anderen serverseitigen Skript- und Programmiersprachen können in PHP Sockets implementiert werden, mit welchen sich die clientseitige Software verbinden kann. Die Wahl dieser Technologie erfolgt aus persönlicher Präferenz des Autors.

Für die Kommunikation zwischen Client und Server bieten sich XML, JSON und ein eigenes, binäres Format an. XML wird zwar von allen gängigen Technologien unterstützt und bietet somit plattformübergreifende Erweiterungsmöglichkeiten, hat allerdings auf Grund seiner Struktur einen größeren Overhead als die beiden Alternativen. Dadurch wird in Summe mehr Bandbreite benötigt. Den geringsten Overhead bringt die Verwendung von Binärdaten mit sich. Hier ist allerdings das Problem, dass es auf Client-Seite nur beschränkte Implementierungsmöglichkeiten gibt und sich diese als übermäßig komplex herausstellen können. Dementsprechend fällt die Wahl auf JSON. Dieses Format ist kompakt und lässt sich plattformübergreifend implementieren. Da Flash JSON nicht nativ beherrscht, wird eine entsprechende Open Source ActionScript Klasse verwendet.

Zur Cheating Prevention werden MD5 Hashwerte verwendet. Auch dafür wird Flash um eine externe ActionScript 2.0 Klasse erweitert.

4.5 Technische Konzeption und angewandte Verfahren

Die erste Überlegung ist es, welche Aktionen einem Spieler möglich sein müssen um Gegenspieler zu finden, mit ihnen ein Spiel anzufangen und dieses zu spielen. Ebenso müssen Ereignisse beachtet werden, welche außerhalb dieser regulären Verhaltensweisen passieren können.

Aktionen und Ereignisse des regulären Spielverlaufs:

1. Spieler loggt sich ein
2. Spieler betritt die Lobby[1] und sieht wer online und für ein Spiel verfügbar ist
3. Spieler lädt einen anderen Spieler ein oder wird eingeladen
4. Neues Spiel wird gestartet
5. Mausbewegung des jeweiligen Spielers wird übertragen

[1] Eine Lobby dient bei Online-Multiplayer-Spielen als Einstiegsbereich in welchem eine Liste potentieller Gegenspieler dargestellt wird.

6. Spielzug wird übertragen
7. Ein Spieler gewinnt, beide Spieler kehren zurück in die Lobby

Irreguläre Ereignisse:

- Spieler kann sich nicht mit dem Server verbinden
- Spieler nimmt die Einladung eines Spielers an, der bereits die Lobby verlassen hat
- Spieler verlässt vorzeitig das Spiel
- Spieler versucht zu cheaten

Für die Herangehensweise an diese Aufgabenstellung bieten sich mehrere der in Kapitel 3 erwähnten Verfahren an. In diesem konkreten Fall werden Folgende aus dem Bereich der Online-Multiplayer Spiele bekannten Konzepte angewandt:

Dead Reckoning
Um die Mausbewegung des Gegners trotz seltener Updates immer noch flüssig darzustellen, wird eine simplifizierte Art von Dead Reckoning auf Client-Seite durchgeführt. Da es sich bei der Mausbewegung um kein kritisches Spielelement handelt, kann ein Verlust an Konsistenz durchaus in Kauf genommen werden.

Kompression
Diese wird nicht in Form eines Kompressionsalgorithmuses eingesetzt, sondern in Form einer verlustbehafteten Filterung der zu übertragenen Daten. So wird nicht jede Bewegung der Maus propagiert, sondern nur jene bei welchen die Maus ihre Position merklich verändert hat. Ab wann eine Position als "merklich verändert"gilt, hängt von einem vordefinierten Schwellenwert ab. Bei den einzelnen Spielzügen wird nur die jeweilige Veränderung und nicht der gesamte Game State übertragen. Dies findet in direktem Einklang mit einer synchronen Simulation statt.

Synchrone Simulation
Nachdem der Spielzug in Form der Position des neu platzierten Teilchen auf dem Spielfeld übertragen wurde, werden am Client jene Berechnungen ausgeführt, welche das Spielfeld aktualisieren. Dies betrifft die Farbe des Feldes, die Anzahl der Teilchen pro Feld sowie gegebenfalls nukleare Reaktionen welche mehrere Felder gleichzeitig verändern.

Cheating Prevention
Um eine Manipulation der einzelnen Spielzüge zu erschweren, werden diese mit einem Hashwert versehen. Spieler, welche ein Spiel vorzeitig verlassen, werden automatisch als Verlierer gewertet, sodass derartiges Verhalten nicht zum eigenen Vorteil verwendet werden kann. Die Gefahr einer Denial of Service Attacke

um die Verbindung des Gegenspielers zu unterbrechen besteht nicht, da jegliche Kommunikation über einen zentralen Server geführt wird und zwischen den Spielern untereinander keine direkte Verbindung besteht.

Auf den Einsatz von Time Manipulation und Area-of-Interest Filtering wird verzichtet. Time Manipulation wird nicht benötigt da Latenz und Latenzunterschiede bei diesem Spiel nur eine geringe Rolle spielen. Area-of-Interest Filtering ist uninteressant da jedem Spieler jederzeit die gesamte Spielinformation zur Verfügung stehen muss.

4.6 Technische Umsetzung

Im folgenden Abschnitt wird betrachtet, wie diese Konzepte in die Praxis umgesetzt werden. Die gezeigten Codebeispiele sind nicht immer vollständig ausgeführt und zeigen nur die wesentlichen Bestandteile, welche für das Verständnis der Logik ausschlaggebend sind. Der vollständige Quelltext ist auf der beiliegenden CD-ROM zu finden.

Auf der Client-Seite werden nur jene Teile betrachtet, welche für die Implementierung der Online-Multiplayer Funktionalitäten von Interesse sind. Die technische Umsetzung der Spiellogik wird außen vor gelassen, abgesehen von jenen Ausschnitten, welche sich direkt auf die Umsetzung des Online-Multiplayers auswirken.

Ein Grundgedanke hinter der Entwicklung der Server-Seite ist eine spätere Weiterverwendung für ähnliche Projekte. Dementsprechend muss eine ausreichende Erweiterbarkeit gegeben sein. Dieser Anforderung wird die meiste Aufmerksamkeit gewidmet.

Anschließend wird bei der Betrachtung der Client-Server Kommunikation auf die praktische Anwendung des Übertragungsformats JSON eingegangen. Ebenso erfolgt eine Darstellung des Kommunikationsflusses zwischen Client und Server.

4.6.1 Client-Server Kommunikation

Wie bereits erwähnt findet die Client-Server Kommunikation mittels JSON Objekten statt. Diese können sowohl von PHP als auch Flash simpel verarbeitet werden. So können aus den Nachrichten, welche zwischen Client und Server ausgetauscht werden, bestimmte Datenobjekte nach dem Parsen direkt übernommen und verwendet werden.

Die Nachrichtenobjekte, welche verschickt werden, haben immer den gleichen Grundaufbau: Ein Property namens `type`, die eigentlichen Daten in Form eines Objektes oder Arrays und gegebenfalls ein Property `hash`, welches bei `rdelta`

Updates einen Hashwert zur Überprüfung beinhaltet. Darauf aufbauend gibt es verschiedene Arten von Nachrichtenobjekten, mit jeweils unterschiedlichen Nutzdaten. Diese werden sind im Folgenden aufgelistet:

- `login`: Nachrichtenobjekte dieses Typs beinhalten als Nutzlast eine simplifizierte Variante des `User` Datenobjektes.
- `userlist`: Enthält ein Array mit `User` Objekten.
- `invite`: Beinhaltet ein `invite` Datenobjekt mit den Properties `sender_id` und `recipient_id`.
- `joingame`: Beinhaltet ein `Game` Objekt, wird vom Client verschickt.
- `startgame`: Beinhaltet ein `Game` Objekt, wird vom Server als Bestätigung verschickt.
- `error`: Beinhaltet ein `error` Datenobjekt, mit den Properties `code` und `msg` welche den Fehlercode und gegebenfalls eine textuelle Fehlermeldung beinhalten.
- `udelta`: Enthält ein `Game` Objekt mit einem `udelta` Datenobjekt.
- `rdelta`: Enthält das Property `hash` sowie ein `Game` Objekt mit einem `rdelta` Datenobjekt.
- `disconnect`: Enthält ein Array mit `User` Objekten.
- `switchlocation`: Beinhaltet nur ein Property welches die neue `location` definiert zu der gewechselt werden soll (beispielsweise die Lobby oder ein Spiel).

Abbildung 4.2 zeigt in welcher Form ein Spielverlauf, wie er in Kapitel 4.5 beschrieben wird, ablaufen kann. Der Server agiert hierbei als Schnittstelle zwischen den beiden Clients.

4.6.2 Clientseitig

Das bereits bestehende Spiel wird um drei Bereiche erweitert: Ein Login Fenster, die Lobby wo andere Spieler gefunden werden können und eine modifizierte Variante des Spiels selber. Die Verbindung zum Server wird über eine Socket Verbindung hergestellt, welche JSON Objekte verschickt und entgegennimmt.

Um sich einzuloggen muss der Spieler bei der derzeitigen Proof of Concept Variante nicht registriert sein. Passwort wird dementsprechend keines benötigt, der Benutzername muss dennoch eindeutig sein. Um diese Eindeutigkeit zu gewährleisten, wird der Name auf der Server-Seite verglichen und gegebenenfalls mit einer Fehlermeldung geantwortet. Sobald sich der Spieler einloggt wird lokal

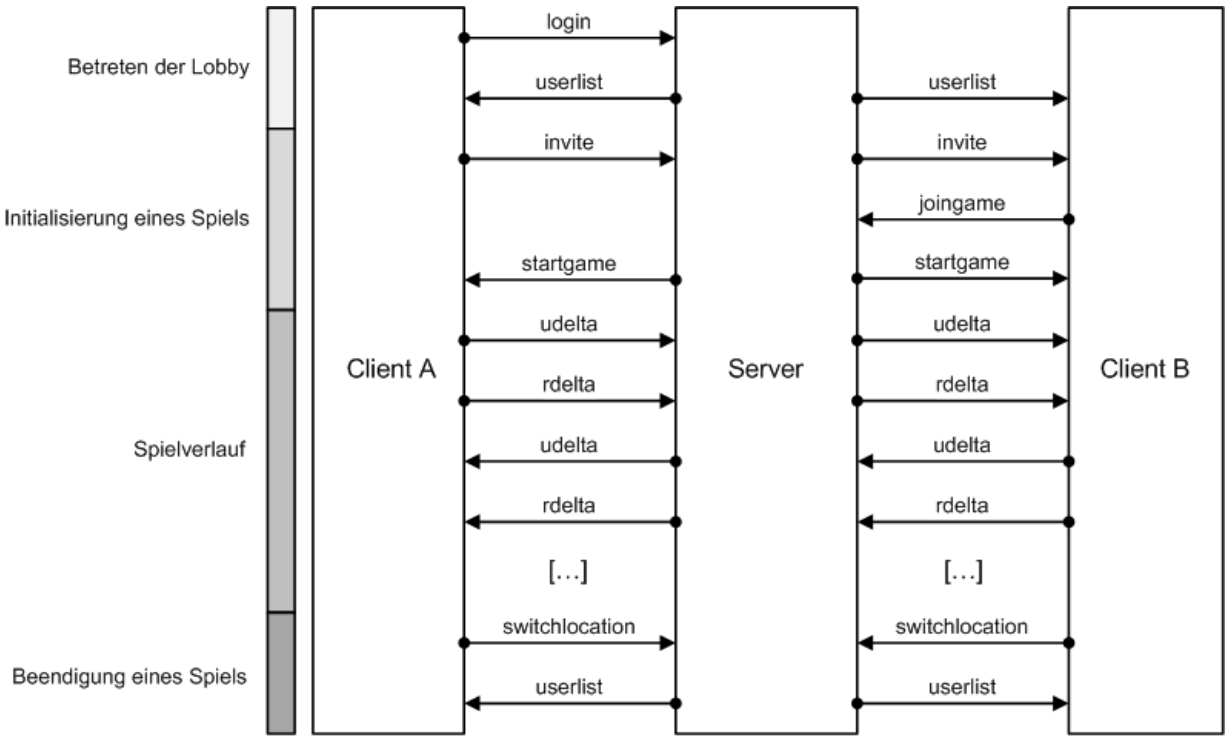

Abbildung 4.2: Client-Server-Client Kommunikationsverlauf

ein Objekt namens `localPlayer` erstellt. Dieses entspricht vom grundlegenden Aufbau her der Klasse `User` auf Server-Seite. Anschließend wird eine `XMLSocket` Instanz erstellt welche versucht eine Verbindung zum Server herzustellen. Obwohl die Klasse `XMLSocket` eigentlich für Kommunikation per XML vorgesehen ist, kann sie durchaus auch für andere Übertragungsformate verwendet werden[2]. Listing 4.1 zeigt den Verbindungsaufbau.

Listing 4.1: Clientseitiger Verbindungsaufbau

```
GameServer = new XMLSocket();

// Aufruf bei der Verbindungsaufbau oder Verbindungsfehlschlag
GameServer.onConnect = function(success) {
    if (success) {
        status.text = "connected";
        welcome.text = "Welcome, "+localPlayer.username;
        login(localPlayer.username); // Login Anfrage an Server
    } else {
        status.text = "connection failed";
    }
};

// Aufruf sobald Verbindung geschlossen wird
GameServer.onClose = function() {
    quitGame(); // Spiel beenden
    gotoAndStop("onlinelobby"); // Zurueck in die Lobby
    status.text = "connection lost";
};

// Verbindungsaufbau mit socket.example.net auf Port 1337
GameServer.connect("socket.example.net",1337);
```

[2]Seit ActionScript 3.0 gibt es eine zusätzliche Socket Klasse die nicht speziell auf XML ausgelegt ist und auch binäre Sockets ermöglicht.

Wie sich erkennen lässt kann durch Eventhandling zuverlässig auf Änderungen des Sockets reagiert werden. Die `TextField` Instanzen `status` und `welcome` dienen der Information des Benutzers über den Zustand der Applikation. Bekommt der Client ein Update vom Server, wird der Event `onData` ausgelöst. Dieser gibt das Datenpaket direkt an die Funktion `process` weiter, welches die eingelangten Daten als JSON parst. Es ensteht ein `msg` Objekt, welches anhand des Properties `type` an die jeweilig zuständigen Funktionen zur Weiterverarbeitung übergeben wird (sh. Listing 4.2).

Listing 4.2: Clientseitige Datenverarbeitung

```
// Aufruf bei Ankunft eines Updates vom Server
GameServer.onData = function(update) {
    process(update);
};

// Sortiert und verarbeitet die eingelangten Daten
function process(update) {
    var msg = json.parse(update);
    switch (msg.type) {
        case "userlist" :
            [...]
        case "invite" :
            [...]
        case "error" :
            [...]
        case "startgame" :
            [...]
        case "udelta" :
            [...]
        case "rdelta" :
            [...]
        case "disconnect" :
            [...]
        case "resend" :
            [...]
        default :
            break;
    }
}
```

Wie die Funktion `process` zeigt, werden die Serverupdates in verschiedene Typen unterteilt, welche jeweils unterschiedlich verarbeitet werden.

userlist
Hierin enthalten ist ein Array von `User` Objekten. Der Client erhält hiermit die Information welche User sich beispielsweise in der Lobby befinden und bekommt Zugriff auf jene Daten, die notwendig sind um über den Server Kontakt mit dem jeweiligen Client aufzunehmen. Die grafische Darstellung der `userlist` erfolgt durch die Flash Komponente `List`. Wird einer der in dieser Liste aufgezählten Spieler angeklickt, wird eine `invite` Anfrage an den Server geschickt, welcher diese an den jeweiligen Client weiterleitet. Durch das Verarbeiten der `userlist` erfährt der Client desweiteren über welchen Socket er selber mit dem Server verbunden ist. Updates dieser Art werden vom Server ausgeschickt sobald ein Benutzer einen bestimmten Bereich betritt (beispielsweise das Betreten der Lobby durch einloggen) oder verlässt (beispielsweise in dem er einem Spiel beitritt und die Lobby verlässt). Diese Daten werden gezielt nur an jene Benutzer geschickt, welche sich im jeweiligen Bereich befinden.

invite
Dies ist die Einladung zum Spiel durch einen anderen Benutzer. Anhand dieser Updates wird eine Liste von Spielern dargestellt, welche auf einen Gegenspieler

warten. Das Klicken auf die jeweilige Einladung gilt als Bestätigung und schickt eine Anfrage an den Server, welcher ein neues Spiel initialisiert. Bei Annehmen einer Einladung wird die Lobby verlassen und das jeweilige Spiel betreten. Das vom Client verschickte `invite` Objekt, welches vom Server weitergeleitet wird, enthält den Sender und den Empfänger der Nachricht in Form einer Id, durch welche vom Server die Sockets der jeweiligen Benutzer bestimmt werden können (sh. Listing 4.3). Diese Ids werden auch bei anderen Nachrichten, welche sich an einen spezifischen Benutzer richten, an den Server mitgeschickt.

Listing 4.3: Invite versenden

```
// Wird aufgerufen wenn ein Spieler in der Lobby eingeladen wird
function inviteToGame(socketId) {
    msg = new Object();
    msg.type = "invite";
    msg.invite = new Object();
    msg.invite.sender_id = localPlayer.socketId;
    msg.invite.recipient_id = socketId;
    GameServer.send(json.stringify(msg));
}
```

error
Es handelt sich hierbei um Fehlermeldungen des Servers, beispielsweise bei invalidem Input. Die Behandlung erfolgt auf Basis eines der in diesen Updates enthaltenen Fehlercodes. Ein Fehler wird beispielsweise dann ausgelöst, wenn der verwendete Benutzername bereits existiert.

startgame
Diese Art von Update initialisiert ein Spiel sobald ein `invite` angenommen wurde. Hierbei wird ein `game` Objekt übertragen, welches Informationen über die Ausgangssituation und die Rahmenbedingungen des Spiels enthält. Dieses wird wie in Listing 4.4 ersichtlich als lokaler Game State übernommen.

Listing 4.4: Spiel starten

```
// msg wird von process uebergeben
function handleStartGame(msg) {
    game = msg.game;
    // Startet das lokale Spiel
    gotoAndStop("humanvsonline");
}
```

udelta
Derartige Updates enthalten jene Daten, welche zwar in hoher Frequenz, aber nicht unbedingt zuverlässig, übertragen werden müssen. Daten, welche über diese Art von Updates übertragen werden, sind in diesem konkreten Fall die Mausbewegungen des jeweiligen Gegners. Diese werden wie in Listing 4.5 ersichtlich nur jede 1/4 Sekunde ausgeschickt. Dieser zeitliche Wert ergibt sich daraus, dass nur jedes 5te Frame ein Mausupdate erfolgt und das Spiel mit 20 Frames pro Sekunde ausgeführt wird.

Listing 4.5: Übertragen der Mausbewegungen

```
// Werden zu Begin des Spiels initialisiert
mouseDelay = 4;
frameCounter = 0;

// Wird bei jedem Frame aufgerufen, also 20 Mal pro Sekunde
function sendMouseUpdate() {
    frameCounter++;
    // Durchfuehrung sobald ein Mindeszeitraum von 1/4 Sekunde verstrichen ist
```

```
    if (frameCounter>=mouseDelay) {
        // Berechnung der Positionsdifferenz
        diffX = targetX-oldX;
        diffY = targetY-oldY;

        // Update wird nur geschickt wenn die Differenz gross genug ist
        if (diffX*diffX+diffY*diffY>5) {
            // Generieren eines msg Objektes und versenden als JSON
            var msg = new Object();
            msg.type = "udelta";
            msg.game = new Object();
            msg.game.udelta = new Object();
            [...]

            // targetX und targetY entsprechen der derzeitigen Mausposition
            msg.game.udelta.xmouse = Math.round(targetX);
            msg.game.udelta.ymouse = Math.round(targetY);
            _root.GameServer.send(_root.json.stringify(msg));
        }
        oldX = targetX;
        oldY = targetY;
        frameCounter = 0;
    }
}
```

Anhand dieser Updates wird die Bewegung des gegnerischen Avatars berechnet. Hierbei wird ein Verfahren zur schrittweisen Annäherung an die tatsächliche Mausposition angewandt. Dies ermöglicht eine flüssige Bewegung trotz unvollständiger Informationen und entspricht einer simplifizierteren Variante von Dead Reckoning (sh. Kapitel 3.2.2). Listing 4.6 zeigt den Grundlegenden Aufbau dieser Vorgehensweise.

Listing 4.6: Berechnung der Position des gegnerischen Avatars

```
// Mausposition des Gegners
targetX = _root.game.udelta.xmouse;
targetY = _root.game.udelta.ymouse;

// Asymptotische Zwischenschritte
xstep = (targetX-_x);
ystep = (targetY-_y);

// Position des Avatars
_x += xstep;
_y += ystep;
```

rdelta

Im Gegensatz zu `udelta` Updates, sind hier spielkritische Elemente enthalten. Beispielsweise die Spielzüge der einzelnen Spieler. Derartige Updates sind mit einem Hashwert versehen, um eine Manipulation durch Cheater zu erschweren. Ein Spielzug wird propagiert sobald ein Spieler ihn durch die Auswahl eines Feldes getätigt hat. Listing 4.7 zeigt in Grundzügen wie eine derartige Nachricht aufgebaut wird.

Listing 4.7: Übertragen eines Spielzugs

```
// Initialisieren eines neuen msg Objektes
var msg = new Object();
msg.type = "rdelta";
msg.game = new Object();

// Der Spielzug
msg.game.rdelta = new Object();
msg.game.rdelta.xclick = x;
msg.game.rdelta.yclick = y;
[...]

// Generieren eines Hashswerts anhand eines JSON Strings des msg.game
// Objektes, einer zufaelligen Zahl sowie einem nicht oeffentlichen vordefinierten
//  Schluesselwert der auf Client und Server ident ist
msg.r = Math.ceil(random(100));
msg.hash = Hash.MD5(_root.json.stringify(msg.game)+_root.md5salt+String(msg.r));
_root.GameServer.send(_root.json.stringify(msg));
```

Diese Nachrichten werden anhand des Hashwertes am Server überprüft. Im Falle eines Fehlers wird dieses Update einige Male erneut angefordert. Danach erhält der Benutzer eine Fehlermeldung, wird aus dem Spiel entfernt und zurück in die Lobby verschoben. Wird die Nachricht vom Server anerkannt, leitet dieser die Nachricht an den jeweiligen Spieler weiter. Die Verarbeitung und Darstellung des Spielzugs eines entfernten Spielers erfolgt auf Basis von Code, der bei QuaRkZ bereits für die Züge des Computergegners eingesetzt wird. Die Entscheidungen der KI werden hierbei durch die vom gegnerischen Spieler empfangenen Spielzüge ersetzt. Diese Art von Updates stellen die unbestimmbaren Ereignisse der synchronen Simulation dar.

disconnect
Diese Nachricht informiert den Client darüber, dass der gegnerische Spieler das Spiel verlassen hat womit der verbleibende Spieler zum Gewinner erklärt wird.

resend
Stellt eine Anfrage des Servers dar, ein fehlerhaftes `rdelta` Datenpaket nochmal zu schicken. Um dies zu ermöglichen wird der jeweils letzte lokale Spielzug gespeichert. Derartige Nachrichten werden dann vom Server verschickt, wenn der Inhalt eines Updates nicht mit dem damit verbundenen Hashwert übereinstimmt.

4.6.3 Serverseitig

Auf der Server-Seite wird ein Socketserver gestartet, welcher die von den Clients eingehenden Inputs verarbeitet und gegebenenfalls weiterleitet. Die Logik ist mit dem Ziel aufgebaut möglichst vielseitig verwendbar zu sein. Es folgt eine Betrachtung der einzelnen Klassen der Server-Seite und der jeweils für Online-Multiplayer relevanten Methoden.

User
Die Klasse `User` stellt eine reines Datenobjekt dar. Als Properties enthält sie die Variablen `socket`, `socket_id`, `name`, `password`, `location` und `invalid_messages`. Als `socket` findet sich eine systeminterne Referenz zum Socket über welchen der Benutzer verbunden ist. Die Variable `socket_id` stellt einen numerischen Wert dar, welcher mit der Socket Referenz assoziert ist, da der Socket selber nicht als JSON Objekt geparst werden kann. Durch diese Id kann der Spieler eindeutig identifiziert und einem Socket zugeordnet werden, ohne dass auf das Socket Objekt selber zugegriffen werden muss. Der Name des Benutzers wird als `name` gespeichert, das Passwort als `password`. Letzteres Attribut ist derzeit nur als Platzhalter für die Zukunft zu sehen, da in diesem Beispiel keine Benutzerkonten implementiert werden. Als `location` wird der Ort verstanden, an dem sich der Benutzer momentan befindet. Dies ist entweder die Lobby oder ein Spiel. Dadurch kann nicht nur verfolgt werden welche Spieler gerade verfügbar sind, sondern es ermöglicht auch das Versenden von Nachrichten zu allen Benutzern einer bestimmten Gruppe. Hinter `invalid_messages`

verbirgt sich ein Counter welcher im Falle eines ungültigen `rdelta` Updates des Clients verwendet wird. Erreicht dieser Counter einen vordefinierten Schwellenwert, gilt der Benutzer als Cheater und die Verbindung wird getrennt.

Game
Instanzen der Klasse `Game` stellen ebenfalls reine Datenobjekte dar. Sie enthalten die Properties `name`, `players`, `reliable`, `unreliable` und `currplayer`. Die Variable `name` korrespondiert mit dem Wert `location` der Klasse `User`. Die beteiligten Spieler werden als `User` im Array `players` gespeichert. Die Objekte `reliable` und `unreliable` enthalten die jeweils zu übertragenen Daten. Der Aufbau dieser Objekte ist nicht weiter definiert und somit frei verwend- und erweiterbar. Im Falle eines rundenbasierten Spiels wird in `currplayer` jener Spieler gespeichert, welcher an der Reihe ist. Dies geschieht in Form eines Indexes, mittels den auf das entsprechende `User` Objekt in `players` zugegriffen werden kann.

ConnectedUsers
Diese Klasse kümmert sich um die Verwaltung der verbundenen Benutzer und ist somit für Presence Management zuständig. Sie enthält die drei Arrays `users`, `user_locations` und `user_names`. Das Array `users` enthält sämtliche Benutzer die derzeit verbunden sind. Um die Verwaltung der jeweiligen `location` einer `User` Instanz zu erleichtern, stellt `user_locations` ein zweidimensionals Array dar, in welchem die `User` zusätzlich in Kontext mit ihrer jeweiligen `location` abgespeichert werden. So kann beispielsweise durch `user_locations["lobby"]` auf alle `User` zugegriffen werden welche sich derzeit in der Lobby befindet. Im Array `user_names` werden die Benutzernamen der verbundenen `User` gespeichert. Dies erleichtert das Überprüfen auf doppelte Benutzernamen.

Die Methoden dieser Klasse beschränken sich auf diverse Verwaltungsaufgaben. Durch `add_location` wird das Array `user_locations` um einen neuen Ort erweitert, wie es zum Beispiel für ein neues Spiel benötigt wird. Die Methoden `add_user`, `remove_user`, `remove_user_by_socket` und `move_user_to_location` agieren ihrem Namen entsprechend und sind notwendig um die Integrität der Daten aufrecht zu erhalten, da die Benutzerinformationen in mehreren Objekten gespeichert werden. Als Getter stehen `get_user_by_socket`, `get_user_by_socket_id` und `get_users_by_location` zur Verfügung. Die letzten beiden Methoden sind jedoch als redundant anzusehen, da sowohl `socket_id` und `location` als Index für die Arrays `users` und `user_locations` verwendet wird.

SocketServer
Diese Klasse stellt den Grundstein dar. Sie kümmert sich um die grundlegende Verwaltung der Sockets und ist sehr allgemein aufgebaut um Wiederverwendbarkeit zu gewährleisten. Listing 4.8 zeigt einen Umriss des Constructors dieser Klasse.

Listing 4.8: Constructor der Klasse SocketServer

```
// Bis auf $address und $port sind alle Parameter optional
public function __construct($address = 0, $port = 0,
$domain = AF_INET, $type = SOCK_STREAM, $protocol = SOL_TCP)
{
```

```
    // Diverse Variablen werden initialisiert
    $this->address = $address;
    $this->port = $port;
    [...]

    // Socket wird erstellt und konfiguriert, gegebenfalls Fehlerbehandlung
    if (($this->parent_socket = socket_create($domain, $type, $protocol)) === false)
        [...]

    // Unbegrenzte Ausfuehrungszeit um ein Timeout zu vermeiden. In der php.ini muss
    // gegebenfalls ebenfalls eine unbegrenzte Ausfuehrungszeit definiert werden
    set_time_limit(0);

    // Diverse andere Einstellungen
    [...]
}
```

Bei der Initialisierung einer `SocketServer` Instanz wird ein PHP Socket erstellt. Die Methode `run` startet den Server. Der Socket wird an den angegebenen Port gebunden und fängt die eingehenden Inputs der Clients zu lesen an. Es wird eine Endlosschleife gestartet, über welche die eingehenden Verbindungen verwaltet und die ankommenden Daten gelesen werden (sh. Listing 4.9). Damit diese Schleife nicht frühzeitig beendet wird, muss eine unbegrenzte Ausführungszeit definiert sein (sh. Listing 4.8)

Listing 4.9: Starten des SocketServers

```
public function run()
{
    $this->server_running = true;

    // Socket wird gebunden
    if (!@socket_bind($this->parent_socket, $this->address, $this->port))
    {
        $this->quit(); // Socket kann nicht gebunden werden, Server wird gestoppt
        [...]
    }

    // Socket Schleife wird abgehoert
    if (!@socket_listen($this->parent_socket, $this->call_limit))
    {
        $this->quit(); // Socket kann nicht abgeh-ort werden, Server wird gestoppt
        [...]
    }

    $this->read_sockets = array($this->parent_socket);
    [...]

    // Socket Schleife wird gestartet
    // Kann gestoppt werden indem server_running auf false gesetzt wird
    while ($this->server_running)
    {
        [...]

        // Sockets deren Zustand sich veraendert hat werden behandelt
        foreach($this->changed_sockets as $socket)
        {

            // Ueberpruefung ob es sich um parent_socket handelt
            if($socket == $this->parent_socket)
            {
                // Neuer Client?
                if (($client = socket_accept($this->parent_socket)) < 0)
                {

                    // Client kann nicht verbinden, Fehlerbehandlung starten
                    $this->on_connect_error($client);
                    continue;
                }
                else
                {

                    // Neuer Client wird in die Liste verbundener Sockets aufgenommen
                    array_push($this->read_sockets, $client);
                    $this->on_connect($client);
                }
            }
            else
            {
                // Neuer Input wird ausgelesen
                $this->read_socket($socket);
            }
```

```
        }
    }
}
```

Für das Auffangen der über die jeweiligen Sockets ankommenden Daten ist die Methode `read_socket` aus Listing 4.10 zuständig. Diese Methode kümmert sich nicht nur um die Verarbeitung der Daten, sondern überwacht auch welche Sockets ihre Verbindung trennen.

Listing 4.10: Auslesen der ankommenden Daten

```
public function read_socket($socket, $length = 16384)
{
    $bytes = socket_recv($socket, $buffer, $length, 0);
    if ($bytes == 0)
    {
        $this->on_disconnect($socket);
    }
    else
    {
        $this->process_buffer($socket, $buffer);
    }
}
```

Die Methode `quit` stoppt die Serverschleife und schließt die vorhandenen Sockets. Die Methoden on_connect_error, on_connect und on_disconnect dienen dem Abfangen und der Behandlung der jeweiligen Ereignisse. In der Basisklasse `SocketServer` enthalten diese nur wenig Funktionalität. Das Verschicken von Updates an Clients geschieht mittels `send_update`. Diese Methode arbeitet ein Array von Clients ab und schreibt auf die korrespondierenden Sockets. Die Methode `process_buffer` ist in dieser Klasse lediglich ein Platzhalter. Sie ist für die logische Weiterverarbeitung der ankommenden Daten vorgesehen.

GameServer
Es handelt sich hier um eine Erweiterung der Klasse `SocketServer`. Der Constructur ist um zusätzliche Einstellungen erweitert. Dazu gehört eine Instanz der Klasse `ConnectedUsers`, welche die Verwaltung der verbundenen Spieler übernimmt. Die Methoden `run`, `quit` und `on_connect` werden direkt von der Elternklasse übernommen.

Die Methode `send_update` ist nur geringfügig erweitert. Im wesentlichen arbeitet sie wie die gleichnamige Methode der Klasse `SocketServer`, allerdings mit den Unterschieden, dass ihr nicht Sockets sonder `User` übergeben werden und dass an die zu verschickenden Nachrichten ein NULL-Charakter angehängt wird. Letzteres ist notwendig damit Flash das Ende einer Datenpakets richtig erkennen kann.

Vollständig durch eigene Logik ersetzt wird die Methode `process_buffer` und stellt das Herzstück der Verwaltung der einkommenden Daten dar. Sie ist das serverseitige Equivalent zur clientseitigen `process` Methode und ist in Listing 4.11 zu sehen. Wie sich erkennen lässt werden abhängig von der Art des Updates verschiedene Subroutinen aufgerufen.

Listing 4.11: Weiterverarbeiten der ankommenden Daten

```
public function process_buffer($socket, $buffer)
{
    // Alternativ kann eine aehnliche Loesung fuer XML Daten implementiert werden
```

```
    if ($this->encoding == "json")
    {
        $buffer = trim($buffer);

        if ($buffer == "<policy-file-request/>")
        {
            // Rueckgabe des von Flash benoetigten Policy Files
            $msg = '<?xml version="1.0"?><cross-domain-policy><allow-access-from
            domain="*" to-ports="*"/></cross-domain-policy>';
            $msg .= chr(0x00); // Abschliessender NULL-Charakter, von Flash benoetigt
            socket_write($socket, $msg);
        }
        else
        {
            // Bestimmen der weiteren Datenverarbeitung
            $data = json_decode($buffer);
            if ($data->type == "login")
            {
                $this->login($socket, $data->user->name);
            }
            elseif ($data->type == "invite")
            {
                $this->invite($data->invite->sender_id, $data->invite->recipient_id);
            }
            elseif ($data->type == "joingame")
            {
                $this->join_game($data->game->p1, $data->game->p2);
            }
                [...]
        }
    }
}
```

Sobald sich der Client mit dem Server verbindet, wird vom Client eine `login` Anfrage geschickt. Diese wird von `process_buffer` an die Methode `login` weitergeleitet. Diese erstellt ein neues `User` Objekt und überprüft mittels der `ConnectedUsers` Instanz ob der Benutzername bereits vorhanden ist. Handelt es sich um einen noch nicht verwendeten Benutzernamen, wird der neue `User` zur Lobby hinzugefügt und ein Update der in der Lobby wartenden Spieler ausgeschickt (sh. Listing 4.12). Dieses erhalten nur jene Clients, welche als `location` Property ebenfalls den Wert `"lobby"` haben. Dieses Update erfüllt desweiteren den Zweck, dass der neu eingeloggte Benutzer erfährt, über welchen Socket er selber mit dem Server verbunden ist.

Listing 4.12: Verarbeiten einer Login Anfrage

```
public function login($socket, $user_name, $password = "")
{
    $user = new User($socket, $user_name, $password);

    // Ueberprueft ob Benutzername bereits vorhanden
    if (!in_array($user_name, $this->connected_users->user_names))
    {
        // User wird in die Lobby verschoben
        $user->location = "lobby";
        $this->connected_users->add_user($user);

        // Lobby Update wird propagiert
        $msg->type = "userlist";
        $msg->users = $this->connected_users->user_locations['lobby'];
        $this->send_update($this->connected_users->user_locations['lobby'], $this->
            encode($msg));
    }
    else
    {
        // Fehlerbehandlung
        [...]
    }
}
```

Die Methode `invite` ist für das Verwalten und Weiterleiten der von den Clients verschickten Einladungen verantwortlich. Sie überprüft ob beide Benutzer noch verbunden und verfügbar sind und leitet die Einladung weiter.

Nimmt ein Spieler eine Einladung an, wird die Methode `join_game` aufgerufen. Diese initialisiert eine neue `Game` Instanz, speichert sie im Array `games` und fügt in `ConnectedUsers` das neue Spiel als `location` hinzu. Befinden sich beide Spieler noch in der Lobby wird ihre `location` auf das jeweilige Spiel geändert. Sie erhalten zur Bestätigung ein Update vom Typ `startgame` und jene Spieler, die sich noch in der Lobby befinden, bekommen eine aktualisierte Benutzerliste (sh. Listing 4.13).

Listing 4.13: Initialisierung eines neuen Spiels

```
public function join_game($socket_id1, $socket_id2)
{
    // User Objekte auslesen
    $user1 = $this->connected_users->get_user_by_socket_id($socket_id1);
    $user2 = $this->connected_users->get_user_by_socket_id($socket_id2);

    // Ueberpruefen ob sich beide User noch in der Lobby befinden
    if ($user1->location == "lobby" && $user2->location == "lobby")
    {
        // Vorerst ein Platzhalter Game Objekt initalisieren
        $game = new Game();
        $index = (array_push($this->games, $game))-1;

        // Neue location hinzufuegen
        $location = "game_".$index;
        $this->connected_users->add_location($location);

        // Versuchen die User zur neuen location zu verschieben
        if ($this->connected_users->move_user_to_location($user1, $location)
        && $this->connected_users->move_user_to_location($user2, $location))
        {
            // User erfolgreich verschoben, game kann befuellt werden
            $clients = array($user1, $user2);
            $this->games[$index]->name = $location;
            $this->games[$index]->players = $clients;

            // Spiel wird gestartet
            $msg->type = "startgame";
            $msg->game = $this->games[$index];
            $this->send_update($clients, $this->encode($msg));

            // Aktueller Stand der Lobby wird propagiert
            $msg->type = "userlist";
            $msg->users = $this->connected_users->user_locations['lobby'];
            $this->send_update($this->connected_users->user_locations['lobby'],
            $this->encode($msg));
        }
        else
        {
            // User konnten nicht verschoben werden, game wird geloescht
            [...]
        }
    }
    else
    {
        // Fehlermeldung wenn User nicht mehr verfuegbar
        [...]
    }
}
```

Die Methode `send_rdelta` wird für die Verarbeitung von `rdelta` Nachrichten verwendet. Da diese Daten kritisch für den Verlauf des Spiels sind, stellt hier Cheating Prevention einen wichtigen Aspekt dar. Dementsprechend findet eine Überprüfung des Hashwertes statt, welcher vom Client gemeinsam mit dem Update verschickt worden ist. Die Variable `md5salt` hat den selben Wert wie die gleichnamige Variable auf Client-Seite, ist jedoch nicht öffentlich bekannt. Kann kein Fehler festgestellt werden, wird das Update an den jeweils anderen Spieler weitergeleitet. Besteht Verdacht auf Manipulation, wird eine `resend` Anfrage an den Client geschickt und das `User` Attribut `invalid_messages` inkrementiert. Nach einer bestimmten Anzahl an ungültigen `rdelta` Datenpaketen wird die Verbindung zum Benutzer unterbrochen. Listing 4.14 gibt einen Überblick.

Listing 4.14: Verarbeiten einer rdelta Nachricht

```
public function send_rdelta($data, $sender_socket_id)
{
    // Ueberpruefung des Hashwertes
    if ($data->hash == md5($this->encode($data->game).$this->md5salt.$data->r)
    && $data->hash != "")
    {
        [...]

        // Hashwert korrekt, Update wird weitergeleitet
        $this->send_update($clients, $this->encode($msg));

        [...]
    }
    else
    {
        [...]

        // Verbindung des Benutzers wird nach einer bestimmten Anzahl
        // an Fehlversuchen getrennt
        if ($this->connected_users->users[$sender_socket_id])->
            increment_invalid_messages() < 4)
        {
            $msg->type = "resend";
            $this->send_update($clients, $this->encode($msg));
        }
        else
        {
            $this->handle_error(4, $clients);
        }
            [...]
    }
}
```

Da hingegen die Methode `send_udelta` Daten verwaltet, welche von geringerer Wichtigkeit sind, stellt sich diese weniger komplex dar. Sie leitet lediglich die empfangenen Daten an den entsprechenden Client weiter.

Trennt ein Client die Verbindung zum Server, wird die Methode `on_disconnect` aufgerufen. Diese erweitert die Methode der Elternklasse `SocketServer` um grundlegende Elemente der Benutzerverwaltung. Sie verwendet die `ConnectedUsers` Instanz um den entsprechenden `User` aus allen Datenobjekten zu löschen, gegebenfalls das `Game` Objekt zu entfernen, sowie diese Änderungen zu propagieren.

Weitere Methoden dieser Klasse sind `switch_location`, `remove_game` und `handle_error`. Diese dienen den vorher erwähnten Methoden und werden mit Ausname von `switch_location` nie vom Client direkt ausgelöst. Um eine spätere Erweiterung der Klasse `GameServer` auf beispielsweise XML Daten zu ermöglichen, findet die Umwandlung der Update-Objekte nach JSON in der Methode `encode` statt. Alle Daten werden, bevor sie propagiert werden, durch diese Methode bearbeitet. Dadurch können andere Umwandlungsroutinen später leichter implementiert werden.

4.7 Bewertung

Als clientseitige Technologie erweist sich Flash als äußerst praktikabel. Abgesehen von den Vorteilen welche Flash auf multimedialer sowie grafischer Ebene bietet, ist die Verfügbarkeit von Sockets jener Aspekt der diese Technologie für browserbasierte Online-Multiplayer-Spiele besonders interessant macht. Dadurch, dass aufgrund der persistenten Verbindung aktuelle Daten auch ohne

ständiges Abrufen eines serverseitigen Skripts erhalten werden, können die gegebenen Resourcen besser genutzt werden. Allerdings bringt das Verwenden von Sockets in Flash auch einen Nachteil mit sich: Es können nur Verbindungen über Ports ab 1024 aufgebaut werden. Dementsprechend kann es zu Problemen mit Firewalls, Routern oder Proxies kommen.

PHP bietet sämtliche Möglichkeiten welche für die Entwicklung dieser Applikation notwendig sind. Da jedoch auch andere Technologien dieselben oder mehr Möglichkeiten bieten, stellt sich die Frage ob eine andere Entwicklungsumgebung, beispielsweise aufgrund besser Performance oder Skalierbarkeit, die bessere Wahl dargestellt hätte. Allerdings erweist sich PHP zumindest im Rahmen dieses Projekts als zweckmäßig.

JSON Objekte bieten eine effiziente Möglichkeit der Kommunikation. Die technischen Einschränkungen, welche JSON Objekte im Vergleich zu XML Objekten haben, sind für die Entwicklung dieser Applikation irrelevant. Dieses Datenformat kann jedoch nicht nur durch einen geringen Overhead punkten. So kann die Verarbeitung der JSON Objekte sowohl auf Client- als auch Server-Seite unkompliziert implementiert werden.

Ein wesentlicher Aspekt der bei der Umsetzung browserbasierter Multi-User Applikationen zu beachten ist, sind die unterschiedlichen Technologien welche zum Einsatz kommen. Einen Datenfluss zwischen Client und Server zu erreichen kann sich je nach den jeweils verwendeten Technologien als mehr oder weniger komplex erweisen. Die Kombination Flash-JSON-PHP erwies sich als unproblematisch. Dennoch müssen die unterschiedlichen Eigenheiten der jeweiligen Technologien beachtet werden. So akzeptiert Flash nur Servernachrichten welche mit einem NULL-Charakter abgeschlossen sind. Ebenso wird bei der ersten Anfrage ein Policy File angefordert, welches von der Server-Seite zur Verfügung gestellt werden muss. Ein wesentlicher Nachteil in der Entwicklung dieser Applikation war jedoch, dass es sich um zwei vollkommen unterschiedliche Skriptsprachen handelt. So müssen manche Elemente doppelt entwickelt werden. Desweiteren muss man beachten, dass manche Vorgehensweisen die vielleicht in PHP sinnvoll sind, mit ActionScript vollkommen anders gelöst werden können und umgekehrt.

Die eingesetzten Verfahren aus dem Bereich der Online-Multiplayer erweisen sich als sinnvoll und leicht implementierbar. Synchrone Simulation ergibt sich oft von selbst anhand der Spiellogik und hilft den Datenfluss gering zu halten. Kompressionsverfahren lassen sich auf viele verschiedene Arten implementieren. Das Verwerfen vernachlässigbarer Daten, wie es bei diesem Projekt der Fall war, ist simpel in der Durchführung und benötigt wenig Rechenaufwand. Dead Reckoning wurde zwar nicht in der herkömmlichen Art und Weise umgesetzt, aber es zeigt sich, dass Verfahren zum Ausgleich fehlender Informationen praktikabel sind und auch im browserbasierten Bereich ohne Probleme eingesetzt werden können.

Der Aufbau der Applikation ist zwar praktikabel, kann jedoch in vielen Punk-

ten noch verbessert werden. Die Erweiterbarkeit und Wiederverwendbarkeit des Quelltextes ist gegeben, kann allerdings noch durch weitere Abstrahierung gesteigert werden. `GameServer` stellt eine Klasse dar, die modularer aufgebaut werden könnte, beispielsweise in dem unterschiedliche Typen von Anfragen von unterschiedlichen Klassen behandelt werden. Das Verwenden von simplen Datenobjekten ohne Logik, ähnlich den Model Objekten der Model-View-Controller Architektur, erleichtert die spätere Erweiterbarkeit. Allerdings sind in der derzeitigen Architektur viele dieser Objekte nicht als Klassen implementiert sondern werden erst vor dem Versenden einer Nachricht definiert. Diese Datenobjekte als Klassen umzusetzen wäre eine klügere Wahl gewesen um die Wiederverwertbarkeit sowie zukünftige Erweiterungen zu erleichtern.

Das Spiel QuaRkZ kann durch die eingesetzten Technologien problemlos über das Internet gespielt werden. Die Mausbewegung des Gegners wird flüssig genug dargestellt um das Spielgefühl eines lokalen Spielers zu vermitteln. Die synchrone Simulation funktioniert, die einzelnen Spielzüge sind durch den mit ihnen verbundenen Hashwert nur schwer manipulierbar. Spieler deren Verbindung geschlossen wird, werden von der Benutzerverwaltung erkannt und die entsprechenden Maßnahmen eingeleitet. Es zeigt sich, dass zumindest simplere Spiele auch browserbasiert als Online-Multiplayer-Spiele umgesetzt werden können.

Allerdings ist zu beachten, dass sich die Tests auf die Funktionalität des Spiels beschränkten. Die Skalierbarkeit und Belastbarkeit der Applikation wurden hierbei außer Acht gelassen, da es nicht Ziel dieser Arbeit ist die technischen Grenzen browserbasierter Online-Multiplayer-Spiele in Zahlen zu fassen, sondern zu erfahren, welche Möglichkeiten innerhalb ebenjener Grenzen bestehen.

4.8 Ausblick

Da das Projekt gezeigt hat, dass sich ein browserbasiertes Multiplayer-Spiel mit den richtigen Technologien und Techniken ohne größere Probleme umsetzen lässt, bietet es sich an diese Idee weiter zu verfolgen. Die für das Projekt entwickelten PHP Klassen können hierbei als Grundlage dienen. Einige der erstrebenswerten Verbesserungen wurden bereits in der Bewertung erwähnt. So empfiehlt es sich die Applikation weiter zu abstrahieren und die Klasse `GameServer` modularer zu gestalten, um somit Logik auszulagern. Ebenso sollten mittels Klassen eindeutige Datenobjekte definiert werden.

Fehlerbehandlung ist ein weiterer Aspekt der an der derzeitigen Applikation noch verbessert werden kann. Das Implementieren zusätzlicher Methoden zur Fehlerbehandlung würde hierbei der Stabilität des Systems dienen. Dazu zählen auch zusätzliche Mechanismen zur Aufrechterhaltung der kollektiven Konsistenz des Game States.

Die Kommunikation zwischen Client und Server kann noch weiter optimiert

werden. Beispielsweise durch eine genauere Betrachtung dessen, welche Teile eines Datenobjektes für die weitere Verarbeitung tatsächlich übertragen werden müssen. Der Overhead der verschickten JSON Nachrichten kann reduziert werden, indem den Objekten und deren Properties kürzere Namen gegeben werden (beispielsweise `g` anstatt `game`).

Auch von Interesse sind weiterführende Tests hinsichtlich der Skalierbarkeit und Stabilität des Servers. Dadurch können Flaschenhälse erkannt und eine gezielte Optimierung bestimmter Teile der Gesamtapplikation durchgeführt werden.

Abgesehen von der Optimierung der Basisapplikation gibt es diverse zusätzliche Funktionen, welche für den Produktivbetrieb interessant sind. So kann die Applikation um Benutzerkonten erweitert werden. Dies ermöglicht auch das Aufzeichnen persönlicher Spielestatistiken sowie die Verwaltung einer Liste der gewonnenen und verlorenen Spiele. Ein weiteres Element welches die Immersion des Spiels verbessern kann, ist die Implementierung einer Chatfunktion sowie von Avatargesten. Dies erleichtert auch das Bilden einer Community. Zusätzlich bieten sich auch Erweiterungen an, welche den Spielablauf selber verbessern. Im Falle von QuaRkZ beispielsweise, das Einführen von Zeitlimits pro Zug.

Da die bisherigen Ergebnisse des Projekts zufriedenstellend sind, wird mit großer Wahrscheinlichkeit die Entwicklung fortgesetzt werden. Dementsprechend wird die entstandene Server-Seite auch für andere Spiele Anwendung finden. Darüber hinaus ist auch ein Einsatz bei vernetzten Multi-User Applikationen im Allgemeinen denkbar.

Kapitel 5

Conclusio

Die technische Weiterentwicklung und zunehmende Vernetzung von Computern eröffnet neue Möglichkeiten bei der Entwicklung von Computerspielen. Dies betrifft sowohl die sich stetig verbessernde Immersion mittels Audio und Video, als auch den Sprung vom lokalen zum Online-Multiplayer. Analog bietet der Browser als Plattform zunehmend mehr Möglichkeiten derartige Applikationen auch webbasiert umzusetzen. Dadurch erschließt sich ein neues Feld, welches zwar mit technischen Einschränkungen zu kämpfen hat, jedoch auch markante Vorteile bietet.

Die Ressourcenlimitationen, welche die Übertragung von Daten mittels Computernetzen mit sich bringen, betreffen unabhängig von der verwendeten Technologie sämtliche Online-Multiplayer. Beschränkte Bandbreite, hohe Latenz, begrenzte Rechenkraft der Hardware sowie die Dichotomie zwischen Konsistenz und Responsiveness sind Aspekte die bedacht werden müssen. Die Defizite, welche diese Faktoren mit sich bringen, lassen sich nie vollständig vermeiden. Ein Gewinn in einem dieser Bereiche geht immer auf Kosten eines anderen Aspekts. Dies führt, abhängig von den verwendeten Technologien, sowie vor allem von den Anforderungen des jeweiligen Spiels, zu einer Priorisierung der einzelnen Faktoren. Jedes Genre bringt eigene Herausforderungen mit sich, die es beim Festlegen der Prioritäten zu beachten gilt.

Je nachdem, welches Defizit man ausgleichen möchte und welche Aspekte vernachlässigbar sind, ergeben sich verschiedene Vorgehensweisen und Konzepte, die bei der technischen Umsetzung angewandt werden können. So steigert beispielsweise Dead Reckoning die Responsiveness auf Kosten der Konsistenz und eine Kompression der übertragenen Daten verringert die benötigte Bandbreite auf Kosten von Rechenkraft. Je nach den Anforderungen des Spiels, sowie der Möglichkeiten der verwendeten Technologien, sind jeweils andere Lösungsansätze aus dem Bereich der Online-Multiplayer-Spiele geeignet. Vor allem die Einschränkungen der jeweils eingesetzten Technologien ist für browserbasierte Ansätze ein wichtiges Thema. Die im Vergleich zu Desktop-Applikationen geringere Performanz browserbasierter Lösungen, stellt hierbei eine wesentliche

Einschränkung dar, sowie dass in den meisten Fällen die Notwendigkeit besteht, eine Client-Server Architektur einzusetzen.

Cheating und Cheating Prevention stellen einen eigenen Teilbereich der Konzeption und Umsetzung dar. Als Grundregel gilt dem Client nicht zuviel Vertrauen zu schenken und somit möglichst wenige kritische Prozesse auf Client-Seite ablaufen zu lassen. Ebenso resultiert daraus die Notwendigket den Input des Clients zu überprüfen. Dies darf allerdings nicht auf Kosten der Spielbarkeit gehen, wodurch Maßnahmen zur Cheating Prevention ressourcen- und spielgerecht eingesetzt werden müssen.

Bei der Konzeption und Umsetzung eines Online-Multiplayer-Spiels gibt es viele Aspekte und Ressourcenlimitationen die beachtet werden müssen. Selbiges gilt für Spiele dieser Art, welche den Browser als Plattform verwenden, jedoch mit dem Unterschied, dass hier weitere technische Einschränkungen in Kauf genommen werden. Allerdings bieten mittlerweile auch web- und browserbasierte Technologien genug Möglichkeiten um derartige Projekte zu entwickeln. Somit lassen sich viele der etablierten Techniken des Gebiets der Online-Multiplayer-Spiele auch im browserbasierten Bereich sinnvoll einsetzen.

Das abschließende Proof of Concept Projekt zeigt wie eine derartige praktische Umsetzung durchgeführt werden kann. Die Auswahl der verwendeten Technologien in Form von Flash, JSON und PHP erweist sich als zielführend, wobei andere Technologien gegebenfalls eine bessere Alternative darstellen können. Die angewandten Methoden, welche sich bereits bei desktopbasierten Online-Multiplayer-Spielen etabliert haben, bieten eine gute Grundlage für die Konzeption und Umsetzung gleichartiger browserbasierter Lösungen. Die Konzepte bleiben dieselben, einzig die Durchführung muss an die äußeren Gegebenheiten und die jeweiligen Technologien angepasst werden.

Das Thema Online-Multiplayer-Spiele stellt ein facettenreiches Fachgebiet dar. Auch wenn sich diese Arbeit auf den technischen Hintergrund dieses Bereichs konzentriert, so sollte nicht vergessen werden, dass es sich hierbei um eine Thematik handelt die auch aus Sicht des Interaktionsdesigns, der Wirtschaft und der Soziologie von Interesse ist. Aus technischer Sicht ist dieses Gebiet allerdings insofern interessant, als dass sich viele Konzepte aus dem Bereich der Online-Multiplayer-Spiele auch auf andere Applikationen anwenden lassen. So ist es naheliegend, dass die aus dieser Arbeit gewonnenen Erkenntnisse auch für andere browserbasierte Applikationen nützlich sein können, welche möglichst schnelle Interaktion zwischen mehreren Benutzern gleichzeitig erfordern.

Glossar

A

ActionScript Die Skriptsprache, welche zur Programmierung in der Flash Entwicklungsumgebung verwendet wird.

Arcade-Videospiel Kostenpflichtige Videospiele, welche in Form von Automaten in Spielhallen oder an anderen Orten der Freizeitgestaltung zu finden sind.

Area-of-Interest Filtering Eine Methode zur Reduzierung der benötigten Bandbreite, indem ein Spieler nur jene Daten erhält, welche für ihn relevant sind.

Aura Der Wahrnehmungsbereich eines Spielers bei Area-of-Interest Filtering.

B

Bot Ein automatisiertes Programm, welches verwendet wird um einen Menschen bei präzise definierbaren, sich wiederholenden Prozessen zu unterstützen.

C

Captcha Ein Test, welcher zur Unterscheidung zwischen automatisierten Programmen und Menschen dient.

Cheater Eine Person die Cheating ausnutzt um den Spielverlauf zu den eigenen Gunsten zu beeinflussen.

Cheating Das Ausnutzen von Bugs, Sicherheitslücken und schlechten Game Design Entscheidungen um sich einen unfairen Vorteil gegenüber anderen Spielern zu verschaffen.

Cheating Prevention Der Einsatz von Maßnahmen gegen Cheating.

CSCW Abkürzung für Computer Supported Cooperative Work. Ein Forschungsgebiet welches sich mit kollaborativer, vernetzter Software auseinandersetzt.

D

Dead Reckoning Ein Verfahren, welches der Erhöhung der Responsiveness eines Spiels dient. Dies wird erreicht durch die Vorausberechnung bestimmter Eigenschaften eines Spielobjektes, wie beispielsweise dessen Position.

Death-Match Spielmodus eines First-Person Shooters, in welchem Jeder gegen Jeden kämpft oder in Teams gegeneinander angetreten wird. Ziel ist die Eliminierung der jeweils feindlichen Spieler.

DIS Abkürzung für Distributed Interactive Simulation. Ein IEEE Standard welcher im militärischen Bereich angewandt wird und sich mit der Kommunikationsarchitektur vernetzter Simulationen befasst.

E

Exploit Ein Bug, welcher für Cheating verwendet werden kann.

F

Fokus Jener Bereich bei Area-of-Interest Filtering, innerhalb dessen ein Spieler Ereignisse und andere Spielobjekte wahrnimmt. Gegenstück zu Nimbus.

Framerate Die Anzahl der Bilder pro Sekunde. Ursprünglich aus dem Bereich Video, kann diese Größe auch bei Spielen und Applikationen angewandt werden.

G

Game State Der momentane Zustand eines Spiels sowie der darin enthaltenen Entitäten.

I

Immersion Jener Zustand der erreicht wird, wenn ein Spieler vollkommen in eine Spielwelt eintaucht und äußere Einflüsse ausgeblendet werden.

J

Jabber Eine auf XMPP (Extensible Messaging and Presence Protocol) basierende Technologie die überlicherweise für Instant Messaging und ähnliche Applikationen verwendet wird.

JSON Abkürzung für JavaScript Object Notation. Ein Format zur plattformunabhängigen Datenübertragung, wobei JavaScript hierbei nicht zwingend eingesetzt werden muss.

L

Latency Balancing Ein Verfahren zum Ausgleichen unterschiedlicher Latenzzeiten verschiedener Spieler mittels einer zwischen Client und Server eingesetzten, zusätzlichen Soft- bzw. Hardwareschicht.

Lobby Der Bereich eines Online-Multiplayer-Spiels, in welchem die verfügbaren Spieler aufgelistet sind und somit die Suche nach potentiellen Gegnern erleichtert wird.

Local Perception Filter Ein Verfahren zum Kaschieren unterschiedlicher Latenzzeiten verschiedener Spieler.

LZW-Algorithmus Abkürzung für Lempel-Ziv-Welch-Algorithmus. Ein lexikonbasiertes, verlustfreies Kompressionsverfahren.

M

MMORPG Abkürzung für Massively Multiplayer Online Role-Playing Game.

MUD Die Abkürzung für Multi User Dungeon. Ein textbasiertes Online-Rollenspiel, welches ähnlich einem Chat funktioniert und Telnet verwendet.

N

Nimbus Dieser Bereich umgibt einen Spieler bei Area-of-Interest Filtering und macht diesen, je nach Größe des Bereichs, leichter oder schwerer wahrnehmbar. Gegenstück zu Fokus.

Q

Queuing Delay Eine durch das serielle Empfangen von Datenpaketen verursachte Verzögerung, welche sich auf die Latenz auswirkt.

R

Responsiveness Die Reaktionszeit, welche eine Applikation oder ein Spiel benötigt um auf Eingaben eines Benutzers zu reagieren.

Rich Internet Application Abgekürzt RIA. Browserbasierte Applikationen, welche gegenüber herkömmlichen Webapplikationen eine höhere Interaktivität und komplexere Logik aufweisen.

S

Serialization Delay Eine durch das serielle Verschicken von Datenpaketen verursachte Verzögerung, welche sich auf die Latenz auswirkt.

Spawn Jene Punkte eines Levels, auf denen Spieler die gestorben sind wieder ins Spiel einsteigen.

Spawncamping In der Nähe eines Spawns auf Spieler lauern die gerade ins Spiel kommen.

Split Screen Eine Methode die angewandt wird, um das Bild das auf einem Monitor oder Fernseher dargestellt wird in zwei oder mehrere Bereich aufzuteilen. Dadurch können verschiedene Blickwinkel gleichzeitig gezeigt werden.

Synchrone Simulation Eine Methode zur Reduzierung der benötigten Bandbreite durch die lokale, mit anderen Clients synchrone, Berechnung zukünftiger Game States, wodurch weniger Daten verschickt werden müssen.

T

Throbber Grafisches Element der Benutzeroberfläche einer Applikation, welches in Form einer animierten Grafik Hintergrundaktivität von unbekannter Dauer anzeigt.

Time Delay Ein Verfahren zum Ausgleich unterschiedlicher Latenzzeiten verschiedener Spieler, welches mit künstlichen zeitlichen Verzögerungen der Kommunikation arbeitet.

Time Manipulation Ein Überbegriff für client- oder serverseitige Verfahren, welche die Problematik unterschiedlicher Latenzzeiten verschiedener Spieler behandeln.

Time Warp Ein Verfahren zum Ausgleich unterschiedlicher Latenzzeiten verschiedener Spieler, welches einen Rollback Mechanismus verwendet.

W

Web Desktop Eine browserbasierte Applikation, welche einen virtuellen Desktop zur Verfügung stellt.

Z

Zoning Das Aufteilen einer Spielwelt in mehrere Teilbereiche um die Rechenlast auf mehrere Server zu verteilen. Üblicherweise angewandt bei MMORPGs.

Literaturverzeichnis

[1] Williams, D. *The International Journal on Media Management* **4**(1), 41–54 (2002).

[2] Armitage, G., Claypool, M., and Branch, P. *Networking and Online Games*. John Wiley & Sons, Ltd, (2006).

[3] Smed, J. and Hakonen, H. *Algorithms and Networking for Computer Games*. John Wiley & Sons, Ltd, (2006).

[4] Bozzon, A., Comai, S., Fraternali, P., and Carughi, G. T. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, 353–360 (ACM, New York, NY, USA, 2006).

[5] Thomas, W. and Stammermann, L. *In-Game Advertising, Werbung in Computerspielen Strategien und Konzepte*. Gabler, (2007).

[6] Kiesler, S. and Curtis, P. *Culture of the Internet: Research Milestones from the Social Sciences*. Mallory International, (1997).

[7] Bryce, J. and Rutter, J. *Spectacle of the Deathmatch: Producing Character and Narrative in Multiplayer Gaming*, 66–80. Wallflower Press, London (2002).

[8] Quax, P., Monsieurs, P., Lamotte, W., De Vleeschauwer, D., and Degrande, N. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, 152–156 (ACM, New York, NY, USA, 2004).

[9] Zander, S., Leeder, I., and Armitage, G. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, 117–124 (ACM, New York, NY, USA, 2005).

[10] Bates, B. *Game Design, Second Edition*. The Premier Press, (2004).

[11] Sheldon, N., Girard, E., Borg, S., Claypool, M., and Agu, E. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, 3–14 (ACM, New York, NY, USA, 2003).

[12] Fritsch, T., Ritter, H., and Schiller, J. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, 1–9 (ACM, New York, NY, USA, 2005).

[13] Müller, J. and GORLATCH, S. *Comput. Entertain.* **4**(3), 11 (2006).

[14] Hartmann, L. *HAW Hamburg, Seminarausarbeitung* (2008).

[15] Dawson, M. *Beginning C++ Game Programming.* Premier Press, (2004).

[16] Lee, N. *Comput. Entertain.* **2**(4), 13–13 (2004).

[17] Lauriat, S. M. *Advanced AJAX, Architecture and Best Practices.* Pearson Education, Inc, (2008).

[18] Jehaes, T., De Vleeschauwer, D., Coppens, T., Van Doorselaer, B., Deckers, E., Naudts, W., Spruyt, K., and Smets, R. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, 63–71 (ACM, New York, NY, USA, 2003).

[19] Funkhouser, T. A. 222–228, (1996).

[20] Nelson, M. *Dr. Dobb's Journal, October* (1989).

[21] Morse, K. L., Bic, L., and Dillencourt, M. *Presence: Teleoper. Virtual Environ.* **9**(1), 52–68 (2000).

[22] Benford, S., Greenhalgh, C., Rodden, T., and Pycock, J. *Commun. ACM* **44**(7), 79–85 (2001).

[23] Yan, J. and Randell, B. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, 1–9 (ACM, New York, NY, USA, 2005).

Online Quellen

[24] Schmidt, A. http://herakles.fzi.de/iiwp/WS0506/03_6.pdf, Abgerufen am 18.02.2009.

[25] Postel, J. http://tools.ietf.org/html/rfc768, Abgerufen am 02.02.2009.

[26] http://tools.ietf.org/html/rfc761. http://tools.ietf.org/html/rfc761, Abgerufen am 02.02.2009.

[27] James, S. R. and Gillam, B. D. http://www.freepatentsonline.com/5964660.html, Abgerufen am 02.02.2009.

[28] http://gearsforums.epicgames.com. http://gearsforums.epicgames.com/showthread.php?t=650739, Abgerufen am 30.01.2009.

[29] Woolley, D. R. http://thinkofit.com/plato/dwplato.htm, Abgerufen am 23.02.2009.

[30] Geier, D. http://www.davidgeier.de/studies/prosem_dictcompr_ss07_foils.pdf, Abgerufen am 24.01.2009.

[31] http://www.w3.org. http://www.w3.org/Protocols/rfc2616/rfc2616.html, Abgerufen am 23.01.2009.

[32] http://www.icq.com. http://www.icq.com, Abgerufen am 21.01.2009.

[33] http://mail.google.com. http://mail.google.com/mail/help/intl/en/about.html, Abgerufen am 21.01.2009.

[34] http://www.quadradius.com. http://www.quadradius.com/quadradius/index.html, Abgerufen am 20.02.2009.

[35] http://livedocs.adobe.com. http://livedocs.adobe.com/flex/3/langref/flash/net/Socket.html, Abgerufen am 21.01.2009.

[36] Crockford, D. http://www.json.org/json.pdf, Abgerufen am 21.01.2009.

[37] Nicula, T. http://portal.acm.org/citation.cfm?id=1357910.1358091&coll=Portal&dl=ACM&CFID=12974505&CFTOKEN=41105632, Abgerufen am 27.11.2008.

[38] http://www.adobe.com/devnet/. http://www.adobe.com/devnet/flashplayer/articles/flash_player_9_security.pdf, Abgerufen am 20.01.2009.

[39] Klammer, M. and Silberhorn, H. http://www8.informatik.uni-erlangen.de/IMMD8/Lectures/WebEng/WS2001-2002/sources/31102001/Vorlesung2.pdf, Abgerufen am 20.01.2009.

[40] http://www.adobe.com/products/. http://www.adobe.com/products/player_census/flashplayer/version_penetration.html, Abgerufen am 19.01.2009.

[41] Alaire, J. http://store1.adobe.com/devnet/flash/whitepapers/richclient.pdf, Abgerufen am 19.01.2009.

[42] http://www.xbox.com/en US/. http://www.xbox.com/en-US/community/news/2007/0305-xboxlivereaches6million.htm, Abgerufen am 15.01.2009.

[43] http://www.blizzard.com. http://eu.blizzard.com/en/press/081223.html, Abgerufen am 15.01.2009.

[44] http://www.xbox.com/de DE/. http://www.xbox.com/de-DE/live/play.htm, Abgerufen am 16.01.2009.

[45] http://store.steampowered.com. http://store.steampowered.com/about/, Abgerufen am 16.01.2009.

[46] http://www.battle.net. http://www.battle.net/intro.shtml, Abgerufen am 16.01.2009.