Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Ausarbeitungarbeit

Andre Jeworutzki

Entwicklung kollaborativer Multiplayerspiele

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

**André Jeworutzki**

## Thema der Ausarbeitung

Entwicklung kollaborativer Multiplayerspiele

## Stichworte

Konsistenz, Synchronisation, Kollaboration, Multiplayer, Latenz, verteiltes System

## Kurzzusammenfassung

Diese Ausarbeitung handelt über die Entwicklung kollaborativer Multiplayerspiele. Hierzu werden spezielle Synchronisationsverfahren vorgestellt, um typische Probleme wie Konsistenz aus verteilten Systemen zu behandeln. Aus diesem Themen entsteht schließlich eine Masterarbeit.

**André Jeworutzki**

## Title of the paper

Developement of collaborative multiplayer games

## Keywords

Consistency, Synchronization, Collaboration, Multiplayer, Latency, Distributed system

## Abstract

This paper is about the developement of collaborative mulitplayer games. On this synchronization methods are presented to solve typical problems like consistency from distributed systems. Finally the master thesis will be based on described topics.

# Inhaltsverzeichnis

# 1. Einleitung

Multiplayerspiele erfreuen sich großer Beliebtheit, so zählt Spielehersteller Blizzard allein elf Millionen zahlende Kunden zu seinem Online-Rollenspiel "World of Warcraft Grund genug sich näher mit dem Thema zu beschäftigen. Infolgedessen bringt diese Ausarbeitung die Entwicklung eines einfachen Multiplayerspiels mit sich; ein Multiplayerspiel, das ehr die Bezeichnung "Tech-Demoäls Spiel verdient. Während das Spiel nur Mittel zum Zweck ist, glimmt der Brennpunkt auf den technischen Aspekten: Die wurzeln tief in den verteilten Systemen. Den Gipfel bildet schließlich die kollaborative Komponente, die Multiplayerspiele auf die nächste Ebene der Spielerfahrung hebt. Am Ende entsteht ein kollaboratives Multiplayerspiel und demonstriert, dass das zu entwickelnden Framework in der Praxis funktioniert.

## 1.1. Aufbau

Diese Ausarbeitung beginnt mit den allgemeinen Problemen verteilter Anwendungen 2 - Ein Multiplayerspiel ist eine spezielle verteilte Anwendung. Anschließend erläutert Kapitel 3 den Begriff Konsistenz und wie Multiplayerspiele eine konsistente Spielwelt erzeugen. Kapitel 4 behandelt daraufhin die Konflikte, die trotz konsistenter Spielwelt durch Latenzzeit entstehen - verschiedene Synchronisationsverfahren begrenzen diese Konflikte. Auf Grundlage der vorgestellten Verfahren beschreibt Kapitel 5 abschließend das Thema der Masterarbeit: die Entwicklung eines Frameworks für kollaborative Multiplayerspiele.

# 2. Grundlegende Probleme bei der Entwicklung von Multiplayerspielen

„Game experience may change during online playing.“ - diese Meldung sieht der Anwender häufig im Ladeschirm eines Multiplayerspiels. Der Multiplayerspieleentwickler hingegen erhält keine solche Nachricht von seiner Entwicklungsumgebung, obwohl seine Kommandos nicht länger atomar sind, sondern sich in drei Ereignisse spalten:

1. Nachrichten erteilen
2. Nachrichten empfangen
3. Nachrichten ausführen

In einem Multiplayerspiel vergeht zwischen allen drei Ereignissen Zeit, deren Dauer dem Entwickler zum Teil unbekannt ist. Weiterhin garantiert dem Entwickler niemand, dass jemand seine Nachricht empfängt und ausführt. Dieser Abschnitt befasst sich mit den Ursachen, die dazu führen, dass der Entwickler sich mit Problemen in Multiplayerspielen auseinandersetzt, die in Singleplayerspielen nicht auftreten.

## 2.1. Limitierte Größen

### 2.1.1. Übertragungsrate

Die Übertragungsrate (häufig auch als Bandbreite bezeichnet) ist die Kapazität der übertragenen Daten über einen Kanal [Schürmann (2004)]. Probleme für Multiplayerspiele entstehen aus folgenden Punkten:

1. Die Übertragungsrate ist begrenzt
2. Teilnehmer verfügen über unterschiedliche Internetzugänge und damit Übertragungsraten
3. Die Übertragungsrate kann von der Senderichtung abhängen („Download“ und „Upload“)

Die Begrenzung der Übertragungsrate schränkt die maximale Anzahl an Spielobjekten und die maximale Frequenz der Spielaktualisierungen (Nachrichten) ein [Pellegrino und Dovrolis (2003)]. Grundsätzlich gilt daher Sparsamkeit bei allen Nachrichten, um nicht an die Grenze der Übertragungsrate zu stoßen - sonst folgt Paketverlust.

#### Paketverlust

Ein unsicheres Netzwerk wie das Internet garantiert nicht, dass ein Empfänger alle Nachrichten erhält. Paketverlust entsteht aus folgenden Gründen [Schürmann (2004)]:

1. Kollision (zum Beispiel im WLAN)
2. Hardwarefehler (zum Beispiel durch „umgekipptes Bit“)
3. Überlastung (zum Beispiel beim Router)

Für Multiplayerspiele bewirkt Paketverlust, dass Nachrichten verloren gehen und die Aktion eines Teilnehmers für andere Teilnehmer niemals vorkommt.

### 2.1.2. Latenz

Die Latenz beschreibt die Verzögerung einer Nachricht von einem Teilnehmer zu einem anderen. Die Verzögerung hängt ab von:

1. Der Ausbreitungsgeschwindigkeit der Daten (durch Lichtgeschwindigkeit begrenzt)
2. Der Verarbeitungsgeschwindigkeit der beteiligten Endgeräte

In Multiplayerspielen stellt die Latenz die wichtigste Größe dar. Studien belegen, dass der Mensch Verzögerungen wahrnimmt, sobald sie jenseits der 100ms Grenze liegen [Bailey (1982); Chen u. a. (2006b)]. In [Beigbeder u. a. (2004)] stellen die Autoren fest, dass eine hohe Latenz das Spielerlebnis stärker beeinträchtigt, als ein hoher Paketverlust. Weitere Untersuchungen zeigen, dass die tolerierbare Obergrenze für die Latenz abhängig vom Genre des Multiplayerspieles ist [Claypool und Claypool (2006); Sheldon u. a. (2003); Fritsch u. a. (2005)].

In der Praxis ist es unmöglich, die Verzögerung exakt zu ermitteln [Lamport (1978). Stattdessen misst man die „Round Trip Time“ (RTT). Die RTT entspricht der Zeitspanne, die nach dem Versenden eines Pakets bis zum Empfang des Antwortpakets vergeht. Ausgehend von einer symmetrischen Verzögerung errechnet sich die Latenz durch die Formel: RTT / 2.

#### Jitter

Jitter beschreibt eine variierende Latenzzeit. Ein Eingangspuffers kompensiert Jitter, hat allerdings eine erhöhte Latenz zur Folge. Durch Jitter fällt es Spielern schwer, sich auf die Verzögerung durch die Latenz einzustellen: „Wie weit muss ich beim Zielen vorhalten?“. Hohe Latenz trübt das Spielgefühl laut [Dick u. a. (2005)] stärker als hoher Jitter.

## 2.2. Fazit

Anders als im Singleplayerspiel ist eine Nachricht im Multiplayerspiel nicht atomar, sondern teilt sich in drei Ereignisse: senden, empfangen und ausführen. Zwischen diesen Ereignissen vergeht Zeit, die nicht immer vorhersehbar ist. Zusätzlich gehen Nachrichten verloren. Ursachen für diese Probleme sind begrenzte Übertragungsraten, Paketverlust und Latenz. Latenz und Jitter beeinträchtigen das Spielgefühl am stärksten.

# 3. Konsistenz

Konsistenz in Multiplayerspielen bedeutet, dass genau ein Zustand der Spielwelt existiert, auf den alle Teilnehmer blicken. Um Konsistenz zu wahren, muss mindestens eine Entität existieren, die den letzten Zustand kennt, damit ein Teilnehmer den Zustand lokal wiederherstellen kann [Palant u. a. (2006)]. Nachfolgend erläutert dieses Kapitel die Konsistenzmodelle. Anhang A erläutert die Techniken.

## 3.1. Konsistenzmodelle

### 3.1.1. Verteiltes Konsistenzmodel

Jeder Teilnehmer berechnet das Model der Spielwelt selbst. Die Teilnehmer setzen jede Aktion sofort um. Damit das Model bei allen Teilnehmern konsistent ist, darf keine Aktion verloren gehen. In diesem Fall muss der Teilnehmer eine verlorene Aktion erkennen und erneut anfordern, da er sonst Gefahr läuft, nicht mehr konsistent mit den anderen Teilnehmern zu sein. Dieses Model beruht darauf, dass alle Teilnehmer vertrauenswürdig sind, da jeder Teilnehmer das Model zu seinem Vorteil ändern kann. Das US-Militär verwendet dieses Model für Simulationen [Standards Committee on Interactive Simulation (SCIS)].

### 3.1.2. Zentrales Konsistenzmodel

Bei dieser Technik existiert genau ein Model der Spielwelt: Kein Teilnehmer berechnet das Model, stattdessen existiert eine autoritäre Entität, die das Model für alle Teilnehmer verwaltet und verteilt. Ein Teilnehmer stellt das Model lediglich graphisch dar und leitet seine Aktionen an die autoritäre Entität weiter. Die Teilnehmer gleichen somit Terminals ("minimal client"). Kommerzielle Multiplayerspiele verwenden ausschließlich autoritäre Entitäten, die sicherstellen, dass jeder Teilnehmer das Model nur innerhalb der zulässigen Regeln ändert.

## 3.2. Fazit

Konsistenz besteht, wenn alle Teilnehmer auf denselben Zustand blicken. Um Konsistenz wiederherzustellen, muss eine autoritäre Entität existiert, die den letzten gültigen Zustand kennt. Im folgenden Kapitel 4 geht es um die Frage, wie sich unterschiedliche Latenzzeiten auf die Konsistenz bei dem einzelnen Teilnehmer auswirken.

# 4. Synchronisation

Das vorherige Kapitel 3 handelt über Konsistenz bei allen Teilnehmern - allerdings nicht zum selben Zeitpunkt: Ursache für dieses Phänomen ist die Latenz und Wirkung sind unterschiedliche Sichten in die Vergangenheit. Kein Teilnehmer sieht jemals den letzten konsistenten Zustand, außer er selbst ist die autoritäre Entität. Obwohl ein Multiplayerspiel konsistent ist, treten durch Latenz merkwürdige Effekte bei den einzelnen Teilnehmern auf, nachfolgend als Konflikt bezeichnet.

Zunächst demonstriert dieses Kapitel exemplarische Konflikte und erklärt anschließend verschiedene Synchronisationstechniken, um diese Konflikte zu reduzieren. Vorweg sei festgehalten, dass keine perfekte Lösung bekannt ist, um Konflikte zu verhindern.

## 4.1. Konsistenz-Konflikte

### 4.1.1. Kollision

Folgende Situation liege vor (siehe Bild 4.1): Spieler A und B stehen sich in einem Spielfeld gegenüber und zwischen Beiden befindet sich ein leeres Feld. Es darf nur ein Spieler auf einem Feld stehen. Spieler A bewegt nun seine Spielfigur vorwärts auf das leere Feld. Da Spieler B die Bewegung von Spieler A (noch) nicht erhalten hat, sieht B, dass das Feld vor ihm leer ist und versucht ebenfalls vorwärts auf das (für B freie) Feld zu springen. Da die Spielwelt von B veraltet ist, kommt es zu einer Konfliktsituation, da Spieler A bereits auf dem Feld steht. In diesem Fall muss B sich an eine Entität wenden, um den konsistenten Zustand lokal wiederherzustellen.

### 4.1.2. Zielen

Wie bei der Kollision sieht ein Spieler immer veraltete Positionen der anderen Spieler. Um einen anderen Spieler zu treffen, muss daher jeder Spieler beim Zielen immer soweit vorhalten, wie die Latenz seine Sicht auf die Spielwelt verzögert.

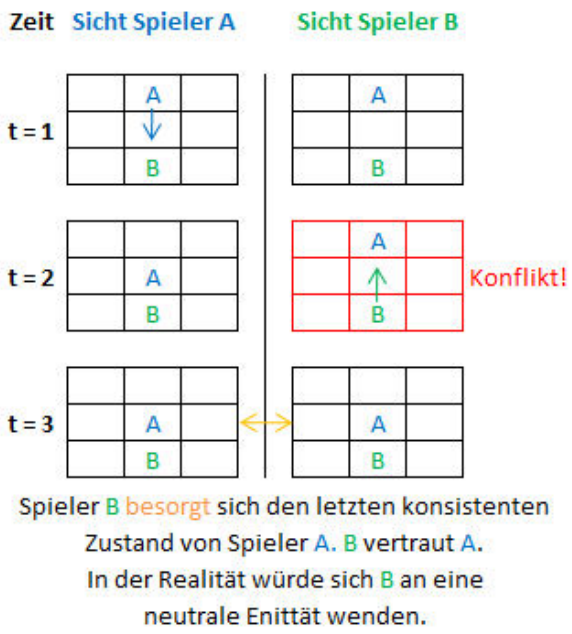


Abbildung 4.1.: Konfliktsituation Kollision

### 4.1.3. Dead man shooting

Spieler A und B schießen zur selben Zeit. A sieht, wie B umfällt. Durch die Latenz bricht jedoch einige Millisekunden später auch A zusammen. A wurde von einem Toten erschossen!

### 4.1.4. Manipulation verteilter Spielobjekte

Mehrere Spieler greifen gleichzeitig auf dasselbe Spielobjekt zu: zwei Spieler heben gleichzeitig eine Kiste auf. Mehrere Möglichkeiten sind denkbar: Nur einer erhält die Kontrolle, beide erhalten Kontrolle oder jeder erhält eine Kopie der Kiste.

## 4.2. Synchronisationsalgorithmen

### 4.2.1. Konservative Synchronisation

Die konservativen Synchronisationsalgorithmen in verteilten Systemen stellen eine kausale Ordnung der Ereignisse durch logische Zeitstempel her. Allerdings ignoriert kausale Ord-

nung den zeitlichen Zusammenhang der Ereignisse, wodurch Konflikte ungehindert auftreten.

### 4.2.2. Pessimistische Synchronisation

Pessimistische Verfahren versuchen durch Verzögerung Konflikte zu vermeiden und hoffen, dass bis zum Ablauf der Verzögerungszeit sich der Konflikt auflöst.

#### Local Lag

Local Lag verzögert die graphische Darstellung bei allen Teilnehmern um eine Verzögerungszeit [Mauve u. a. (2004)]. Solange alle Nachrichten vor Ablauf der Verzögerungszeit eintreffen, kann jeder Teilnehmer die Nachricht lokal zur selben Zeit ausführen - vorausgesetzt alle Teilnehmer haben ihre lokalen Uhren synchronisiert. Im Optimalfall stimmt die Darstellung des Spielgeschehens bei allen überein. Die Verzögerungszeit lässt sich jedoch nicht beliebig erhöhen, da sie den flüssigen Spieleindruck trübt: die Zeit zwischen Tastendruck und Darstellen der Aktion steigt merkbar an. Laut Arbeit [Shneiderman (1984)] nimmt ein Benutzer bereits eine Verzögerungszeit zwischen 80 und 100 ms wahr.

#### Local Rollback

Ähnlich wie beim Local Lag verzögert Local Rollback die Verarbeitung eingegangenen Nachrichten, in der Hoffnung dass bis zum Ablauf der Verzögerungszeit alle Nachrichten eintreffen, die älter als die zuletzt ausgeführte sind („straggler message“). Ein Rollback lässt sich ausschließen, wenn alle Teilnehmer bereits neuere Nachrichten als die zuletzt ausgeführte gesendet haben.

### 4.2.3. Optimistische Synchronisation

Optimistische Verfahren betreiben keinen Aufwand, um Konflikte zu verhindern. Stattdessen stellen sie Methoden vor, um die konfliktauslösende Nachricht rückgängig zu machen.

#### Time Warp

Das Time Warp Verfahren führt einen Rollback aus, sobald ein Konflikt auftritt [Mauve u. a. (2004)]. Um einen Konflikt zu erkennen, überträgt Time Warp mit jeder Nachricht einen Zeitstempel. Das Verfahren prüft die kausale Ordnung aller Nachrichten: Trifft eine Straggler-Nachricht ein, lieft ein Konflikt vor. Time Warp leitet daraufhin ein Rollback ein und korrigiert die zuletzt ausgeführten Zustandsänderungen durch „incremental state saving“ oder „copy state saving“.

Zwar löst Time Warp einen Konflikt auf, allerdings treten dabei graphische Anomalien auf beispielsweise Teleportation eines Spielobjekts; Dieser Effekt lässt sich abmildern, indem sich das Spielobjekt allmählich auf die richtige Position verschiebt (abhängig von der Anwendung).

### 4.2.4. Verteilte Synchronisation

#### Bucket Synchronisation

Die Bucket Synchronisation teilt die Spielzeit in Intervalle gleicher Länge ein ("Buckets") [Gautier und Diot (1998)]. Die Teilnehmer verarbeiten jedes Bucket nach einer festen Verzögerung, um die Spielwelt darzustellen. Jeder Teilnehmer sammelt Datenpakete (ÄDUs") von anderen Teilnehmern, sodass in jedem Bucket zum Zeitpunkt der Darstellung je eine ADU von jedem Teilnehmer existiert. Erhält ein Teilnehmer eine verspätet ADU, verwirft er sie - stattdessen verwendet er die letzte erhaltene ADU, um den entsprechenden Teilnehmer darzustellen. Der Schwerpunkt bei der Entwicklung der Bucket Synchronisation liegt auf Peer-to-Peer Systeme. Aus diesem Grund existiert keine autoritäre Entität, die Konflikte auflösen kann.

#### Rendezvous Synchronisation

Im Artikel [Chandler und Finney (2005)] verfolgen die Autoren einen Ansatz mit vielen Zuständen: Jeder Teilnehmer besitzt seine eigene Sicht auf die Spielwelt. Die Teilnehmer tauschen ihre lokalen Zustände aus, um sie abzugleichen. Es existiert also nie eine Entität, die jemals den einzig „wahren“ Zustand kennt. Allerdings bleibt der Artikel einer Antwort schuldig, ob der Ansatz in der Praxis funktioniert und mit welcher Anwendung. Der Leser stelle sich hierzu ein Online-Schachspiel vor, bei dem beide Spieler unterschiedliche Spielfiguren besitzen.

### 4.2.5. Spezielle Synchronisation

#### Client-side-prediction

Ein Teilnehmer sendet eine Nachricht an eine autoritäre Entität. Die autoritäre Entität schätzt dann die Latenzzeit zum Teilnehmer ab und berechnet die Spielwelt zum Zeitpunkt zurück, in dem der Teilnehmer die Nachricht losgeschickt hat. Dadurch muss ein Spieler zum Beispiel beim Zielen nicht länger vorhalten, sondern kann direkt auf die gegnerische Spielfigur schießen [Valve (2005); Bernier (2006)].

## 4.3. Fazit

Liegt Konsistenz in Multiplayerspielen vor, können durch Latenz dennoch unterschiedliche Konflikte auftreten. Pessimistische Synchronisation versucht durch Warten, diese Konflikte zu verhindern. Time Warp stellt den konsistenten Zustand wieder her, wenn es bereits zu spät ist und ein Konflikt auftrat. Die Bucket Synchronisation kommt ohne autoritäre Entität aus, während Rendezvous Synchronisation versucht mehrere asynchrone Zustände anzugleichen. Bei Client-side-prediction berechnet die autoritäre Entität den Zustand des Clients zur Zeit der Aktion zurück und erzeugt dadurch ein direktes Spielgefühl beim Spieler.

# 5. Masterarbeit

Bisher handelte diese Ausarbeitung über die Probleme (siehe 2 und 3) und Verfahren (siehe 4) in Multiplayerspielen. In Hinblick auf die Masterarbeit ist jetzt ein konkreter Anwendungsfall zu entwickeln, der eine Kombination der beschriebenen Verfahren nutzt: Die Spielidee ist ein kollaboratives Multiplayerspiel. Eine Marktanalyse zeigt, dass der kollaborative Bereich in kommerziellen Multiplayerspielen bisher wenig Aufmerksamkeit erhalten hat. Mögliche Lösungsansätze, Risiken und eine Zusammenfassung schließen dieses Kapitel und diese Ausarbeitung ab.

## 5.1. Marktanalyse

Betrachtet man den Spielemarkt der letzten Jahre, so lassen sich unter anderen folgende Trends ausmachen:

### 5.1.1. Trend 1: Physikrätsel

Das Spiel Half Life 2 hat mit Hilfe der Physik-Engine "Havok"physikalische Rätsel salonfähig gemacht. Seitdem erscheinen immer weitere Spiele, die Havok nicht nur für graphische Effekte einsetzen, sondern um größere Interaktion mit der Spielwelt zu ermöglichen. In Multiplayerspielen verzichten Spieleentwickler bisher jedoch auf eine realistische Interaktion mit der Spielwelt. Die Gründe sind Skalierbarkeit und Latenz.

### 5.1.2. Trend 2: Kooperativer Multiplayer

Die Teilnehmer spielen gemeinsam gegen computergesteuerte Gegner, Beispiele aus verschiedenen Genres sind: „Left 4 Dead“, „Command & Conquer - Red Alert 3“, „Splinter Cell 4“ und „Diablo 2“.

Beide Trends bilden die grundlegende Idee für ein kollaboratives Multiplayerspiel.

## 5.2. Anwendungsfälle

### 5.2.1. Anwendungsfall I: Kollaboratives Mikado

Zwei oder mehr Spieler spielen gemeinsam Mikado. Das 3D-Spielfeld besteht aus übereinander gestapelten Mikadostäben. Jeder Spieler besitzt lediglich eine "Hand". Der Spieler kann mit dieser Hand einen Stab an jede beliebige Stelle anfassen und daran ziehen. Um erfolgreich einen Stab aus einem Mikadostapel zu entfernen, ohne dass der Stapel umfällt, müssen die Spieler gleichzeitig an beiden Enden eines Stabes fassen und ziehen.

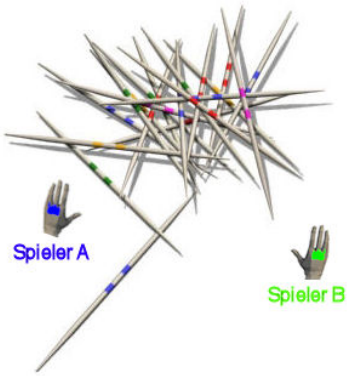


Abbildung 5.1.: Kollaboratives Mikado

### 5.2.2. Anwendungsfall II: Kollaboratives Tetris

Zwei oder mehr Spieler spielen gemeinsam Tetris. Auf einem zwei dimensionalen Spielfeld fallen nach und nach unterschiedlich geformte Steine von oben runter. Die Steine sammeln sich unten auf den Boden. Ziel des Spiels ist es, eine oder mehrere durchgängige horizontale Linien herzustellen. Diese Linien verschwinden daraufhin und schaffen Platz für neue Steine. Das Spiel ist verloren, wenn kein Platz mehr für weitere Steine auf dem Spielfeld ist.

Wie beim kollaborativen Mikado besitzen wieder alle Spieler eine Hand. Um einen Stein drehen zu können, muss zunächst ein Spieler eine Seite des Steins festhalten, während der andere Spieler den Stein von einer anderen Seite zieht. Zieht ein Spieler alleine an einen Stein, kann er den Stein horizontal verschieben oder die Fallgeschwindigkeit erhöhen.

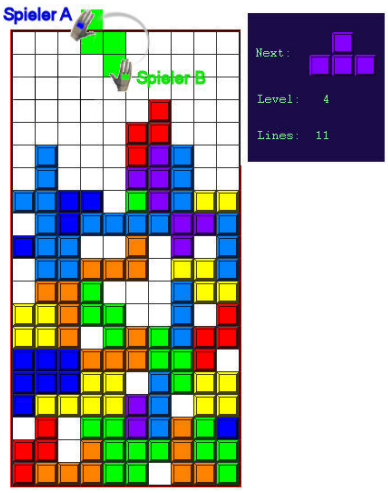


Abbildung 5.2.: Kollaboratives Tetris

## 5.3. Umsetzung

Ein ungefähre Lösungsansatz sehe wie folgt aus: Durch die kollaborativen Komponente ist das Ziel eine möglichst synchrone Darstellung unter den Teilnehmern. Hierzu bietet sich Local Lag an. Um die Avatare darzustellen, eignet sich Dead Reckoning (siehe Anhang D). Damit die Spieler möglichst viele Spielobjekte manipulieren können, sind die Spielobjekte durch Berechnungvorschrift zum Beispiel mit Bucket Synchronisation zu übertragen. Die Skalierbarkeit sollte durch eine replizierte Architektur siehe Anhang B.2.3 sichergestellt sein. Weiterhin ist der Einsatz graphischer Effekte (siehe Anhang B.2.4) denkbar, um Latenzzeiten darzustellen und den Spieler ein Gefühl für eine kurze Wartezeit zu vermitteln.

## 5.4. Ziele

Die Masterarbeit soll feststellen, ob kollaborative Multiplayerspiele trotz Latenzzeiten möglich sind. Weiterhin ist die Skalierbarkeit zu untersuchen: Wie viele manipulierbare Spielobjekte sind möglich? Wie viele Spieler kann das Spiel unterstützen?

Weiterhin ist zu klären, welche Kombination der vorgestellten Verfahren für ein kollaboratives Multiplayerspiel sich am besten eignet. Hierzu ist ein Netzwerkframework zu entwickeln, dass beliebige kollaborative Multiplayerspiele unterstützt; also unabhängig von den Spielobjekten ist. Zur Evaluation der Anwendungsfälle ist eine schlichte 3D-Umgebung erforderlich: Diese ließe sich mit XNA [Microsoft (2008)] umsetzen. Zusätzlich wäre eine Physik-Engine wünschenswert, da die Grundidee auf das Lösen physikalischer Rätsel basiert.

## 5.5. Risiken

- Gleichzeitige Kontrolle eines Spielobjekts lässt sich durch Latenzzeit nicht gewährleisten
- Limitierte Latenzzeit oder Bandbreite schränken die Zahl der Spielobjekte ein
- Interaktion mit der Spielwelt schränkt die Zahl der Teilnehmer oder Spielobjekte zu sehr ein
- Frei verfügbare Physik-Engine erforderlich
- Kenntnisse in Graphikprogrammierung erforderlich

## 5.6. Zusammenfassung

Das Thema Multiplayerspiele beschäftigt sich mit den Problemen aus den verteilten Systeme. Um eine virtuelle Spielwelt zu erschaffen, muss die Sicht aller Teilnehmer konsistent sein. Trotz Konsistenz treten durch Latenz bei den Spielobjekten merkwürde Effekte auf, wie plötzliche Teleportation oder fehlende Kollision. Synchronisationsverfahren versuchen diese Effekte zu behandeln.Die Masterarbeit beschreibt die Idee von kollaborativen Multiplayerspielen. Das zu entwickelnde Netzwerk-Framework besteht aus einer Kombination der vorgestellten Verfahren, um interaktive, dynamische Spielwelten zu erschaffen.

# Literaturverzeichnis


[Aggarwal u. a. 2004] AGGARWAL, Sudhir ; BANAVAR, Hemant ; KHANDELWAL, Amit ; MUKHERJEE, Sarit ; RANGARAJAN, Sampath: Accuracy in dead-reckoning based distributed multi-player games. In: *NETGAMES*, 2004, S. 161–165

[Aggarwal u. a. 2005] AGGARWAL, Sudhir ; BANAVAR, Hemant ; MUKHERJEE, Sarit ; RANGARAJAN, Sampath: Fairness in dead-reckoning based distributed multi-player games. In: *NETGAMES*, 2005, S. 1–10

[Bailey 1982] BAILEY, Robert W.: *Human Performance Engineering: A Guide for System Designers*. Prentice Hall Professional Technical Reference, 1982. – 672 S. – ISBN 0-1344-5320-4

[Beigbeder u. a. 2004] BEIGBEDER, Tom ; COUGHLAN, Rory ; LUSHER, Corey ; PLUNKETT, John ; AGU, Emmanuel ; CLAYPOOL, Mark: The effects of loss and latency on user performance in unreal tournament 2003. In: *NETGAMES*, 2004, S. 144–151

[Bernier 2006] BERNIER, Yahn W.: *Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization*. 2006. – URL http://developer.valvesoftware.com/wiki/Lag_Compensation

[Bettner 2001] BETTNER, Paul: *1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond*. (Game Developer's Conference 2001. 2001. – URL http://zoo.cs.yale.edu/classes/cs538/readings/papers/terrano_1500arch.pdf

[Cai u. a. 1999] CAI, Wentong ; LEE, Francis B. S. ; CHEN, L.: An Auto-Adaptive Dead Reckoning Algorithm for Distributed Interactive Simulation. In: *Workshop on Parallel and Distributed Simulation*, 1999, S. 82–89

[Chandler und Finney 2005] CHANDLER, Angie ; FINNEY, Joe: Rendezvous: An Alternative Approach to Conflict Resolution for Real Time Multi-User Applications. In: *PDP*, 2005, S. 160–167

[Chen u. a. 2006a] CHEN, Kuan-Ta ; HUANG, Chun-Ying ; HUANG, Polly ; LEI, Chin-Laung: An empirical evaluation of TCP performance in online games. In: *Advances in Computer Entertainment Technology*, 2006, S. 5

[Chen u. a. 2006b] CHEN, Kuan-Ta ; HUANG, Polly ; LEI, Chin-Laung: How sensitive are online gamers to network quality? In: *Commun. ACM* 49 (2006), Nr. 11, S. 34–38

[Claypool und Claypool 2006] CLAYPOOL, Mark ; CLAYPOOL, Kajal T.: Latency and player actions in online games. In: *Commun. ACM* 49 (2006), Nr. 11, S. 40–45

[Conner und Holden 1997] CONNER, Brook ; HOLDEN, Loring: Providing a Low Latency User Experience in a High Latency Application. In: *SI3D*, 1997, S. 45–48, 184

[Cronin u. a. 2002] CRONIN, Eric ; FILSTRUP, Burton ; KURC, Anthony R. ; JAMIN, Sugih: An efficient synchronization mechanism for mirrored game architectures. In: *NETGAMES*, 2002, S. 67–73

[Dick u. a. 2005] DICK, Matthias ; WELLNITZ, Oliver ; WOLF, Lars C.: Analysis of factors affecting players' performance and perception in multiplayer games. In: *NETGAMES*, 2005, S. 1–7

[Fritsch u. a. 2005] FRITSCH, Tobias ; RITTER, Hartmut ; SCHILLER, Jochen H.: The effect of latency and network limitations on MMORPGs: a field study of everquest2. In: *NETGAMES*, 2005, S. 1–9

[Gautier und Diot 1998] GAUTIER, Laurent ; DIOT, Christophe: Design and Evaluation of MiMaze, a Multi-Player Game on the Internet. In: *ICMCS*, 1998, S. 233–236

[Gray und Lamport 2006] GRAY, Jim ; LAMPORT, Leslie: Consensus on transaction commit. In: *ACM Trans. Database Syst.* 31 (2006), Nr. 1, S. 133–160

[Guo u. a. 2003] GUO, Katherine ; MUKHERJEE, Sarit ; RANGARAJAN, Sampath ; PAUL, Sanjoy: A fair message exchange framework for distributed multi-player games. In: *NETGAMES*, 2003, S. 29–41

[Harcsik u. a. 2007] HARCSIK, Szabolcs ; PETLUND, Andreas ; GRIWODZ, Carsten ; HALVORSEN, Pål: Latency evaluation of networking mechanisms for game traffic. In: *NETGAMES*, 2007, S. 129–134

[Standards Committee on Interactive Simulation (SCIS) of the IEEE Computer Society 1996] IEEE COMPUTER SOCIETY, USA Standards Committee on Interactive Simulation (SCIS) of the: *IEEE standard for distributed interactive simulation communication services and profiles*. 1996

[Lamport 1978] LAMPORT, Leslie: Time, Clocks, and the Ordering of Events in a Distributed System. In: *Commun. ACM* 21 (1978), Nr. 7, S. 558–565

[Lamport 2004] LAMPORT, Leslie: Recent Discoveries from Paxos. In: *DSN*, 2004, S. 3

[Lee u. a. 2002] LEE, Ho ; KOZLOWSKI, Eric ; LENKER, Scott ; JAMIN, Sugih: Multiplayer game cheating prevention with pipelined lockstep protocol. In: *IWEC*, 2002, S. 31–39

[Li u. a. 2007] LI, Shaolong ; CHEN, Changjia ; LI, Lei: A new method for path prediction in network games. In: *Computers in Entertainment* 5 (2007), Nr. 4

[Lincroft 1999] LINCROFT, Peter: *The Internet Sucks: Or, What I Learned Coding X-Wing vs. TIE Fighter*. (Game Developer's Conference 1999. 1999. – URL http://www.gamasutra.com/features/19990903/lincroft_01.htm

[Mauve u. a. 2002] MAUVE, Martin ; FISCHER, Stefan ; WIDMER, Jörg: A generic proxy system for networked computer games. In: *NETGAMES*, 2002, S. 25–28

[Mauve u. a. 2004] MAUVE, Martin ; VOGEL, Jürgen ; HILT, Volker ; EFFELSBERG, Wolfgang: Local-lag and timewarp: providing consistency for replicated continuous applications. In: *IEEE Transactions on Multimedia* 6 (2004), Nr. 1, S. 47–57

[Microsoft 2008] MICROSOFT: *Game Framework XNA*. 2008. – URL http://msdn.microsoft.com/en-us/xna/default.aspx

[MINDCONTROL ] MINDCONTROL: *How to communicate peer-to-peer through NAT (Network Address Translation) firewalls*. – URL http://www.mindcontrol.org/~hplus/nat-punch.html

[Palant u. a. 2006] PALANT, Wladimir ; GRIWODZ, Carsten ; HALVORSEN, Pål: Consistency requirements in multiplayer online games. In: *NETGAMES*, 2006, S. 51

[Pantel und Wolf 2002] PANTEL, Lothar ; WOLF, Lars C.: On the suitability of dead reckoning schemes for games. In: *NETGAMES*, 2002, S. 79–84

[Pellegrino und Dovrolis 2003] PELLEGRINO, Joseph D. ; DOVROLIS, Constantinos: Bandwidth requirement and state consistency in three multiplayer game architectures. In: *NETGAMES*, 2003, S. 52–59

[Schürmann 2004] SCHÜRMANN, Bernd: *Grundlagen der Rechnerkommunikation*. 1. Vieweg + Teubner, 2004. – 65 S. – ISBN 3-5281-5562-0

[Sheldon u. a. 2003] SHELDON, Nathan ; GIRARD, Eric ; BORG, Seth ; CLAYPOOL, Mark ; AGU, Emmanuel: The effect of latency on user performance in Warcraft III. In: *NETGAMES*, 2003, S. 3–14

[Shneiderman 1984] SHNEIDERMAN, Ben: Response Time and Display Rate in Human Performance with Computers. In: *ACM Comput. Surv.* 16 (1984), Nr. 3, S. 265–285

[Valve 2005] VALVE: *Source Multiplayer Networking*. 2005. – URL http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

[Webb u. a. 2007] WEBB, Steven D. ; SOH, Sieteng ; LAU, William: Enhanced mirrored servers for network games. In: *NETGAMES*, 2007, S. 117–122

[Yan und Randell 2005] YAN, Jeff J. ; RANDELL, Brian: A systematic classification of cheating in online games. In: *NETGAMES*, 2005, S. 1–9

[Zhang u. a. 2006] ZHANG, Yi ; CHEN, Ling ; CHEN, Gencai: Globally synchronized dead-reckoning with local lag for continuous distributed multiplayer games. In: *NETGAMES*, 2006, S. 7

# A. Erhalten der Konsistenz

Ein Multiplayerspiel muss drei Probleme lösen, um die Konsistenzbedingung zu erfüllen:

1. Verlorene Nachrichten
2. Duplizierte Nachrichten
3. Ordnung aller Nachrichten

Hinweis: Das TCP-Protokoll löst zwar die drei Probleme, allerdings nur für Punkt-zu-Punkt-Verbindungen. Multiplayerspiele benötigen hingegen eine globale Ordnung aller Nachrichten.

## A.1. Problem 1: Verlorene Nachrichten

Nachrichten gehen durch Paketverlust verloren. Je nach Art der Nachricht, sind unterschiedliche Vorkehrungen zu treffen:

### A.1.1. Möglichkeit 1: Jede Nachricht enthält den absoluten Zustand

Jeder Teilnehmer sendet regelmäßig seinen kompletten Zustand. Geht ein Zustand verloren, wartet ein anderer Teilnehmer auf den nächsten und stellt damit den konsistenten Zustand wieder her. Veraltete Nachrichten kann der Teilnehmer ignorieren.

- Geringe Latenz durch regelmäßiges Einwegsenden des Zustandes
- Kein Teilnehmer muss Nachrichten neu anfordern
- Jeder Spielobjekt sendet seinen kompletten Zustand: Je mehr Spielobjekte, desto höher das Nachrichtenaufkommen

### A.1.2. Möglichkeit 2: Jede Nachricht enthält die Berechnungsvorschrift für den nächsten Zustand

Solange jeder Teilnehmer einen konsistenten Zustand besitzt, kann er mit der Berechnungsvorschrift den nächsten konsistenten Zustand berechnen. Dies erfordert allerdings, dass keine Nachricht verloren geht und die Teilnehmer alle verlorenen Nachrichten erkennen - zum Beispiel durch eine Sequenznummer in jeder Nachricht.

- Hohe Latenz bei Nachrichtenverlust: Teilnehmer muss warten, bis die verlorene Nachricht eintrifft
- Teilnehmer müssen verlorene Nachrichten erkennen
- Durch die Berechnungsvorschrift lassen sich komplexe Zustandsänderungen mit nur einer Nachricht beschreiben: Veränderung vieler Spielobjekte möglich

## A.2. Problem 2: Duplizierte Nachrichten

Duplizierte Nachrichten sind in Multiplayerspielen nicht gravierend, da die Aktionen in der Regel idempotent sind zum Beispiel: „Bewege Einheit A nach B“. Allerdings kann ein Duplikat zu einem späteren Zeitpunkt eintreffen und damit eine neuere Nachricht aufheben. Um Duplikate zu vermeiden, erhält jede Nachricht einen Zeitstempel oder eine eindeutige Sequenznummer.

## A.3. Problem 3: Ordnung aller Nachrichten

Ordnung aller Nachrichten lässt sich durch folgende Strategien erreichen:

### A.3.1. Strategie 1: Ordnung durch autoritäre Entität

Jeder Teilnehmer sendet seine Nachrichten an eine autoritäre Entität, die alle Nachrichten in eine kausale Ordnung bringt und an die Teilnehmer weiterleitet. Die autoritäre Entität muss für alle Teilnehmer erreichbar sein und trägt den Hauptteil der Rechenlast und des Kommunikationsaufwands.

### A.3.2. Strategie 2: Ordnung durch Zeitstempel

Alle Teilnehmer synchronisieren regelmäßig ihre Uhren und versehen jede Nachricht mit einem Zeitstempel. Statt Zeitstempel sind auch Sequenznummern möglich (logische Uhr) [Lamport (1978)]. Diese Strategie bevorzugt bestimmte Teilnehmer, da die Uhren nicht exakt synchron laufen. Außerdem können Teilnehmer die Zeitstempel manipulieren.

### A.3.3. Strategie 3: Ordnung durch Consensus

Die Teilnehmer handeln die Ordnung untereinander aus. Da mehrere Verhandlungsrunden notwendig sind, ist die Kommunikation umfangreich und langwierig [Gray und Lamport (2006)].

# B. Minimierung der Latenz

Dieser Abschnitt befasstsich mit Möglichkeiten zur Minimierung der Latenz: Hierzu zählen die Wahl des Transportprotokolls und der Netzwerktopologie, sowie die gefühlte Latenz.

## B.1. Transportprotokoll

Die Wahl des Transportprotokolls wirkt sich umgehend auf die Latenz aus. Neben den klassischen TCP und UDP Protokoll existieren spezialisierte Transportprotokolle wie das Real-Time Transport Protokoll (RTS) oder das Game Transport Protokoll (GTS), auf die diese Ausarbeitung aber nicht eingeht.

Ferner wäre Multicast-Übertragung wünschenswert, da ein Multiplayerspiel in der Regel eine Gruppe von Teilnehmern adressiert, jedoch verbieten viele Router im Internet Multicast. Aus demselben Grund stellt Broadcast keine Option dar.

### B.1.1. TCP

TCP liefert für Punkt-zu-Punkt-Verbindungen Fluss- und Staukontrolle, Ordnung der Nachrichten und Sicherheit vor verlorenen Nachrichten. Besonders die erneute Übertragung verlorener Nachrichten wirkt sich negativ auf die Latenz in Multiplayerspielen aus [Chen u. a. (2006a)]. Grundsätzlich besteht der Netzwerkverkehr in Multiplayerspielen aus vielen, winzigen Paketen, die für die Übertragung mit TCP ungeeignet sind, weil TCP für byteorientierte Übertragung großer Datenmengen ausgelegt ist. Trotz der Nachteile setzen viele Multiplayerspiele wie zum Beispiel "World of Warcraft"TCP ein, um hauptsächlich Schwierigkeiten mit Firewalls zu umgehen.

### B.1.2. UDP

UDP ist ein unsicheres, verbindungsloses Übertragungsprotokoll. Im Gegensatz zu TCP beherrscht UDP keinerlei Kontrolle, um den Datenverkehr zu regeln. Weiterhin erkennt UDP

keine verlorenen Nachrichten und stellt auch keine Ordnung her. UDP eignet sich besonders für Multiplayerspiele, die regelmäßig kleine Updatepakete senden, weil beim Senden kein zusätzlicher Aufwand entsteht. Desweiteren existieren UDP-Middlewares, die UDP um die fehlenden Kontroll- und Sicherheitsfunktionen von TCP erweitern.

## B.2. Netzwerktopologie

Die Wahl der Netzwerktopologie entscheidet darüber, ob die Kommunikation direkt oder indirekt durch eine autoritäre Entität läuft. Je direkter der Übertragungsweg, desto geringer die Latenz. Neben der Latenz sind aber weitere Aspekte zu beachten.

### B.2.1. Client/Server

Die Client/Server Architektur ist die am häufigsten verwendete. Sie lässt sich in Vergleich zu anderen Architekturen leicht implementieren: Die Clients tauschen ihre Daten immer über den Server aus.

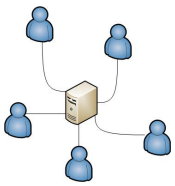

Abbildung B.1.: Client/Server Architektur

Für die Clients gilt das „minimal client“-Prinzip: Sie gleichen Terminals, die lediglich Benutzereingaben weiterleiten und die Spielwelt darstellen. Rechenkapazitäten der Clients bleiben daher ungenutzt.

Allein der Server ist für die Berechnung der Spielwelt zuständig. Der Hauptgrund ist, Cheating C.1 durch die Clients entgegenzuwirken. Zusätzlich kann ein Hersteller bezahlbare Dienste besser überwachen. Weiterhin übernimmt der Server die Rolle der autoritären Entität und verfügt somit über den letzten gültigen Zustand.

Da sämtliche Kommunikation über den Server läuft, muss er über hohe Netzwerk- und Rechenkapazität verfügen - dabei fallen jedoch laufende Kosten für die Infrastruktur an, besonders was die Ausfallsicherheit und Bandbreite betrifft. Schließlich ist die die Entfernung zwischen Client und Server ein wichtiges Kriterium, da sie die Latenz bestimmt.

### B.2.2. Peer-to-Peer

In einer Peer-to-Peer Architektur kommunizieren alle Teilnehmer ohne Umweg, wodurch die Latenz gering ausfällt, sofern die Entfernung untereinander nicht zu groß ist. Da jeder Teilnehmer alle anderen Teilnehmer über seine Aktionen informiert, ist der Kommunikationsbedarf weitaus höher als in der Client/Server Architektur. Außerdem führen die direkten Verbindungen in der Praxis häufig zu NAT-Problemen [MINDCONTROL].

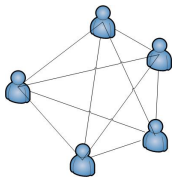

Abbildung B.2.: Peer to Peer Architektur

In Konfliktsituationen gibt es keine neutrale autoritäre Entität, die eine schnelle Entscheidung trifft. Stattdessen müssen die Teilnehmer langwierige Verhandlungen führen [Lamport (2004)]. Da der Ausfall eines Teilnehmers nicht so schwer wiegt wie der Ausfall eines zentralen Servers, ist die Peer-to-Peer Architektur sehr robust. Cheating ist das größte Problem in Peer-to-Peer, da ein Cheater die Spielwelt unfair manipulieren kann oder seine Aktionen solange verzögert, bis die Aktionen der anderen bekannt sind. Das Lockstep Protokoll [Lee u. a. (2002)] bietet sich hierzu als Lösung an.

In der Praxis gibt es nur wenige Multiplayerspiele, die Peer-to-Peer einsetzen und stammen ausschließlich aus dem Genre der Echtzeit-Strategie wie beispielsweise Age of Empires [Bettner (2001)].

### B.2.3. Replizierte Architektur (mit oder ohne Peer-to-Peer)

Sowohl Client/Server als auch Peer-to-Peer skalieren nicht mit zunehmender Teilnehmerzahl. Daher erweitert man die Client/Server Architektur um gespiegelte Server („mirror“ oder „proxy“) [Mauve u. a. (2002); Cronin u. a. (2002); Webb u. a. (2007)]. Alle Mirrors tauschen sich ständig aus und besitzen zu jedem Zeitpunkt ein Replikat der Spielwelt. Fällt ein Mirror aus, springt ein anderer für ihn ein.

Durch eine weiträumige Verteilung der Mirrors, kann sich jeder Teilnehmer mit den nächstgelegenen Mirror verbinden und so seine Latenz verkleinern. Da Mirrors als vertrauenswürdig gelten, können sie effizient untereinander kommunizieren, da zum Beispiel Prüfungen gegen Cheating C.1 entfallen.

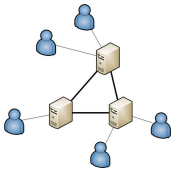

Abbildung B.3.: Mirror Architektur

Um die Latenz weiter zu minimieren, können die Teilnehmer wie in Peer-to-Peer direkt Nachrichten austauschen. Die Mirrors erhalten dann Kopien dieser Nachrichten und prüfen, ob die Aktionen den Regeln entsprechen. In der Praxis setzen besonders Massive Online Multiplayerspiele wie „World of Warcraft“ die replizierte Architektur ein, da sie mit vielen Teilnehmern skaliert. Nachteile sind die aufwendige Entwicklung und die hohen Kosten der Infrastruktur.

### B.2.4. Gefühlte Latenz

Graphische Tricks minimieren die gefühlte Latenz, indem das Spiel die Aktion eines Spielers sofort auf den Bildschirm bestätigt, hierzu zwei Beispiele:

- Feuert ein Spieler eine Waffe, so sieht er in der Regel nur „Platzpatronen“: Jede Patrone fliegt physikalisch korrekt und schlägt irgendwo ein, jedoch wertet erst die autoritäre Entität den tatsächlichen Flugverlauf und den zugefügten Schaden aus.
- Befiehlt ein Spieler einer Einheit von A nach B zu laufen, so taucht sofort eine Markierung an Stelle B auf - die Einheit selbst läuft allerdings erst Millisekunden später los.

[Conner und Holden (1997)] beschreibt eine interessante Möglichkeit, die Auswirkungen von Latenz durch grafische Effekte darzustellen. Die Autoren schlagen als grafische Effekte Bewegungsunschärfe, Unschärfe und Transparenz vor. Dieser Ansatz unterscheidet sich mit üblichen, weil Multiplayerspiele sonst versuchen, Latenz zu kaschieren.

## B.3. Fazit

Die Wahl des Transportprotokolls und der Netzwerktopologie beeinfluss unmittelbar die Latenz. Jedoch sind weitere Aspekte zu beachten: Das Transportprotokoll UDP bietet gegenüber TCP keine Funktionen, um den Datenfluss zu kontrollieren. Bei der Netzwerktopologie stellt die replizierte Architektur einen Kompromiss zwischen Client/Server und Peer-to-Peer

dar. Die gefühlte Latenz lässt sich durch graphische Effekte minimieren. Alternativ machen graphische Effekte die Latenz kenntlich.

# C. Cheating und Fairness

## C.1. Cheating

„Ein Cheater kann das Spiel für 1000 andere Spieler zerstören.“, so lautet das Ergebnis für viele Multiplayerspiele. Kommerzielle Multiplayerspiele betreiben einen großen Aufwand, um Cheating zu verhindern. Die Möglichkeiten zu schummeln, sind in Multiplayerspielen äußerst vielfältig [Yan und Randell (2005)].

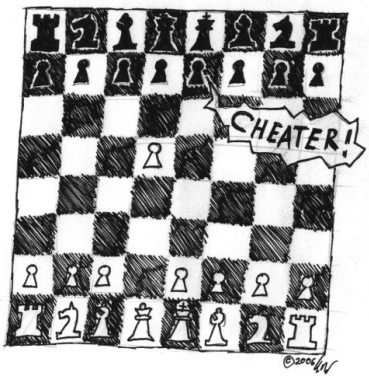


Abbildung C.1.: Cheater Beispiel

Grundlegend sind für Multiplayerspiele folgende Punkte zu beachten:

- Erkennen von nichtregelentsprechendes Verhalten
- Identifizierung des Teilnehmers, der sich nicht nach den Regeln richtet
- Maßnahmen ergreifen, die zukünftige Regelbrüche ausschließen

In Hinblick auf die Latenz wirken sich alle Anti-Cheat-Techniken negativ aus, da eine autoritäre Entität jede Nachricht zuvor prüft, ob keine unfaire Manipulation stattgefunden hat. Könnte man allen Teilnehmern vertrauen, ließe sich die Latenz halbieren.

## C.2. Fairness

Bei Fairness geht es darum, dass alle Teilnehmer trotz unterschiedlichen Latenzzeiten die gleiche Gewinnaussicht haben. Nachfolgend beschriebene Techniken manipulieren die Latenz oder Datenrate der Teilnehmer, um zwischen diesen ein ausgewogenes Latenzverhältnis zu erreichen [Aggarwal u. a. (2005); Guo u. a. (2003)]:

### C.2.1. scheduling algorithm

Dieser Algorithmus schätzt die Latenz zwischen den Spielern ab und verzögert die Nachrichten zu jeden einzelnen Teilnehmern in Abhängigkeit zur Schätzung.

### C.2.2. budget base algorithm

Ein Budget für jeden Teilnehmer steuert, wie viele Nachrichten er enthält: weit entfernte Teilnehmer erhalten häufiger Nachrichten als nah gelegene.

# D. Dead Reckoning

In einer idealen Umgebung mit uneingeschränkter Bandbreite, uneingeschränkter Prozessorleistung und vernachlässigbarer Latenz wäre es möglich, die Zustandsaktualisierungen entfernter Spielobjekte in der gleichen Frequenz zu senden, in der der Empfänger die Darstellung der Spielwelt aktualisiert. Dead Reconing beschäftig sich mit folgenden Fragen [Pantel und Wolf (2002); Aggarwal u. a. (2004); Cai u. a. (1999); Zhang u. a. (2006); Li u. a. (2007)]:

1. Welche Zustandsinformationen tauschen die Teilnehmer untereinander aus?
2. Wann schickt ein Sender Zustandsupdates eines Spielobjekts?
3. Wo werden entfernte Spielobjekte im aktuellen Frame dargestellt, wenn kein aktuelles Zustandsupdate vorliegt?

## D.1. Prinzip

Das generelle Prinzip bei Dead Reckoning ist es, Netzwerkverkehr durch lokale Berechnungen zu ersetzen. Die beiden grundsätzlichen Prinzipien zur Darstellung entfernter Objekte sind Extrapolation und Interpolation. Extrapolationstechniken basieren auf dem Prinzip, von vergangenen Zustandsangaben eines Objekts auf dessen aktuellen Zustand zu schließen. Diese Techniken werden allgemein unter dem Begriff Dead Reckoning zusammengefasst. Interpolation basiert auf der Übertragung reiner Positionsangaben. Um Dead Reckoning zu implementieren, muss jeder lokale Teilnehmer das Bewegungsmodell aller entfernten Spielobjekte kennen. Bei Paketverlust kann Dead Reckoning den zuletzt bekannten Zustand als Extrapolationsgrundlage nutzen, ohne das Paket neu anfordern zu müssen.

## D.2. Fazit

Dead Reckoning beschreibt Techniken zur Interpolation und Extrapolation, um entfernte Spielobjekte flüssig darzustellen. Es sendet für jedes Spielobjekt den absoluten Zustand. Paketverlust lässt sich somit ignorieren.