Entwicklung eines ortsbezogenen Objekt-Management-Systems für Peer-to-Peer-Onlinespiele

Ragnar Mogk

Prüfer: Prof. Alejandro Buchmann
Betreuer: Robert Rehner

Databases and Distributed Systems
Technische Universität Darmstadt

Bachelorarbeit
30. März 2013

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Die elektronische Fassung der Arbeit stimmt mit der gedruckten überein.

Darmstadt, den 30.03.2013 Ragnar Mogk

Abstract

Videospiele sind heutzutage fester Bestandteil unserer Gesellschaft. Die Branche wächst und sorgt mit Vielfalt und Innovationen für große Umsätze. Ein Aspekt vieler Spiele ist ein Multiplayermodus, darin verbinden sich mehrere Spieler über das Netzwerk zu einer gemeinsamen Spielwelt. Oft werden Client/Server Architekturen verwendet, um diese Netzwerke aufzubauen. Damit sind für den Betreiber Kosten und Aufwand verbunden, außerdem haben Ausfälle bei einem zentralen Server große Auswirkungen. Deshalb soll eine Peer-to-Peer Architektur für das Netzwerk verwendet werden. Dabei gibt es verschiedene Herausforderungen, die bewältigt werden müssen. Interagieren mehrere Spieler gleichzeitig mit Objekten in der Welt soll gewährleistet werden, dass dadurch keine Fehler im Spiel entstehen. Diese Arbeit nutzt ein bestehendes Interest Management Verfahren aus, um ein solches Objektmanagement zu entwickeln. Dazu werden Objekten dynamisch Besitzer zugeteilt, die dafür zuständig sind den Zustand des Objektes zu verwalten und allen anderen Spielern zugänglich zu machen. Um dies effizient zu erledigen, wird das Interesse von Spielern an Objekten geschätzt und die Objekte anhand dieses Interesses verteilt. Das Objektmanagement wird als Teil von *Planet PI4* implementiert und anschließend mit Hilfe von Simulationen ausgewertet.

Inhaltsverzeichnis

1 Einführung

In heutigen Videospielen übernehmen Spieler die Kontrolle über eine oder mehrere Spielfiguren in einer virtuellen Welt. Diese Form der Unterhaltung ist inzwischen in der Bevölkerung verbreitet und sorgt mit Vielfalt und Innovationen für große Umsätze.[1]

Viele Spiele stellen eine zwei- oder dreidimensionale Welt dar und erlauben es mehreren Spielern sich in der selben Welt aufzuhalten, indem sie sich über ein Netzwerk miteinander verbinden. Dadurch entstehen für die Spieler viele neue Interaktionsmöglichkeiten, die das jeweilige Spiel attraktiver machen können. Für den Betreiber des Spiels kann das zur Verfügung stellen der Netzwerkinfrastruktur jedoch hohe Kosten und Aufwand bedeuten. Der Grund dafür ist, dass oft leistungsstarke Server mit viel Bandbreite betrieben werden müssen, um einer hohen Anzahl an Spielern eine Verbindung zu ermöglichen. Auch kann es bei Engpässen oder Ausfällen dazu kommen, dass Spieler nicht mehr zum Netzwerk verbinden können. Die folgenden Teile dieser Arbeit beschäftigt sich mit Objektmanagement, einem typischen Problem das bei Netzwerkspielen auftritt, und beschreibt Lösungen, mit denen die Ressourcen der Spieler genutz werden, um eine bessere Skalierbarkeit und Ausfallsicherheit zu erbringen.

1.1 Objektmanagement

In vielen virtuellen Spielwelten können Spieler mit Objekten interagieren. Das ermöglicht ihnen Einfluss auf die Welt nehmen und diese zu verändern. In einem vernetzen Spiel sollten alle Spieler die Änderungen aller anderen Spielers sehen können. Ist die selbe Tür beim einen Spieler offen und beim anderen geschlossen, kommt es zu Inkonsistenzen, die den Spieler verwirren und zu weiteren Fehlern führen können.

Besonders wenn sich viele Spieler und Objekte in den Welt befinden, kann es schnell dazu kommen, dass sich die Spieler nicht über den Zustand der Objekte einig sind. Es muss also dafür gesorgt werden, dass sich die Spieler auf eine gemeinsamen Zustand einigen. Die Wahl dieses Zustandes sollte fair sein, es sollten also nicht bestimmte Spieler benachteiligt werden.

Andererseits können kleine Abweichungen des Zustandes häufig toleriert werden, wenn die Spieler selbst den Unterschied nicht merken. Für viele Objekte in der Welt gilt, dass ein Spieler sie gar nicht oder nur ungenau wahrnehmen kann, da sein Sichtbereich eingeschränkt ist. Im nächsten Abschnitt werden Ansätze vorgestellt wie dies ausgenutzt werden kann, um effizient einen gemeinsamen Zustand zu erreichen.

1.2 Lösungsansätze

Um einen gemeinsamen Zustand der Welt zu erlangen, senden sich alle Spieler in einem ersten Ansatz ihre Änderungen an der Umgebung gegenseitig zu. Dies sorgt zwar dafür, dass jeder den gemeinsamen Zustand kennt, aber die Bandbreite die jeder einzelne Spieler benötigt, steigt linear mit der Anzahl der Spieler insgesamt.

Eine gängige Lösung besteht darin einen zentralen Server zu bestimmen, der die Änderungen von allen Spielern erhält und diese dann an die anderen Spieler verteilt. Dieser Ansatz führt oft zu besseren Ergebnissen, da man den Server so ausstattet, dass er eine ausreichende Bandbreite für die gewünschte Anzahl an Spielern hat. Da der Server die Änderungen aller Spieler erhält und dann an jeden anderen Spieler weiterverteilt, steigt seine Belastung quadratisch mit der Anzahl der Spieler. Um die Anzahl der Spieler an die der Server Änderungen senden muss zu verringern, werden jedem Spieler nur Änderungen innerhalb seines *Sichtbereiches* (siehe Abschnitt 3.2) zugesendet. Ein zentraler Server bringt aber ein erhöhtes Ausfallrisiko mit sich. Fällt der eine Server aus, dann bricht das Netzwerk zusammen. Zudem kommt auch ein leistungsstarker Server bei vielen Benutzern an die Grenzen von Bandbreite und Rechenleistung. Ein Server der genug Kapazität hat, ist also möglicherweise zu kostspielig.

Stattdessen können die Ressourcen genutzt werden, die den Benutzern auf ihren Clienten zur Verfügung stehen. Hierzu wird ein Overlay Netzwerk zwischen den Benutzern auf eine solche Weise aufgebaut, dass der Aufwand nicht wie oben beschriebene linear wächst. Dafür wird wie beim Client/Server Verfahren ausgenutzt, dass Spieler nicht an allen Änderungen interessiert sind. Änderungen werden nur dann an Spieler verteilt, wenn die Änderung einen Teil des Sichtbereiches dieses Spielers betrifft. Schon das Aufrechterhalten dieses Overlay Netzwerkes ist ein Problem, mit dem sich unter anderen *VON*[8] und *pSense*[13] beschäftigen.

Eine der Schwierigkeiten in einem solchen verteilem System ist es dafür zu sorgen, dass der Zustand den jeder Spieler sieht konsistent ist, also keine größeren Abweichungen untereinander auftreten. Der Rest dieser Arbeit beschäftigt sich damit, wie die Objekte so gehandhabt werden können, dass der Zustand konsistent bleibt, auch wenn sich die Verbindung zwischen Spielern im Netzwerk ändern oder ganz ausfallen.

1.3 Gliederung der Arbeit

Zunächst werden in Kapitel 2 grundsätzliche Ansätze vorgestellt, die zum Objektmanagement verwendet werden können. Danach wird auf vorhandene Objektmanagementsysteme eingegangen und deren Funktionsweise sowie Limitierungen und Probleme dargelegt. Der Rest dieser Arbeit befasst sich dann mit anderen Lösungen für einige dieser Probleme.

Kapitel 3 führt grundlegende Begriffe ein und definiert welche Ziele genau erreicht werden sollen. Danach wird der Algorithmus zur Verwaltung der Objekte auf einer abstrakten Ebene beschrieben. Die technische Umsetzung dieser Ansätze wird danach in Kapitel 4

genauer dargestellt. Dazu gehören das Kommunikationsprotokoll zwischen Spielern, sowie das Umgehen mit spezifischen Problemen. Ob die gewünschten Ziele erreicht wurden, wird in Kapitel 5 betrachtet. Die gewünschten Ziele werden etwas genauer untersucht und es werden Metriken festgelegt, mit denen sich die Qualität des Ansatzes messen lässt. Zudem werden die Ergebnisse von Simulationen untersucht, um Eigenschaften des Systems festzustellen und das Erreichen der Ziele zu prüfen.

2 Stand der Forschung

Dieser Abschnitt stellt einige Verfahren vor um, Objekte in einem Netzwerk zu verwalten. Zuerst werden verteilte Hashtabellen beschrieben. Diese stellen ein sehr allgemeines Verfahren dar und sind für Echtzeitsysteme eher ungeeignet, da aber viele ihrer Konzepte relevant sind, ist es hilfreich wenigstens einen Überblick über ihre Funktionsweise zu geben. Die weiteren Teile dieses Abschnittes beschäftigen sich mit einzelnen Lösungen speziell für Echtzeitsysteme und Videospiele.

2.1 Distributed Hash Table (DHT)

Eine DHT nimmt eine Menge von eindeutigen Bezeichnern (*Schlüssel*) und weist sie einer Menge von Knoten in einem Netzwerk zu. Es ist damit möglich den Knoten zu finden, der für einen bestimmten Schlüssel zuständig ist, um bei diesem Knoten die zu dem Schlüssel gehörenden Daten abzufragen oder zu bearbeiten. Wie genau Schlüssel bestimmten Knoten zugeteilt werden, unterscheidet sich je nach Implementierung. Eine Möglichkeit besteht darin, die Schlüssel und Knoten in einem Ring anzuordnen und jedem Knoten die Schlüssel zuzuordnen, die dem Knoten am nächsten sind. Wichtig ist es, eine Distanz zwischen Knoten und Schlüsseln zu definieren. Bei *Chord*[14] übernimmt ein Knoten beispielsweise alle Schlüssel, die sich vor ihm auf dem Ring befinden. Die Last muss gleichmäßig auf die Knoten verteilt werden. Um diese gleichmäßige Verteilung der Knoten und Schlüssel im Ring zu gewährleisten, kann ein kryptographisches Hash verwendet werden. Damit werden die Werte generiert, die die Position für Schlüssel und Knoten im Ring bestimmen. Um einen Schlüssel zu finden, sendet ein Knoten eine Anfrage dichter an das gesuchte Ziel. Die Anfrage wird solange an einen Knoten gesendet, der dichter am Ziel ist, bis der gesuchte Schlüssel gefunden wurde. Dazu hat jeder Knoten eine Liste an bekannten Knoten, die meist so gewählt ist, dass er sehr viele Knoten in seiner eigenen Nähe kennt und nur wenige weiter entfernte Knoten. Die Wahl der Liste beeinflusst die Geschwindigkeit einer Suchanfrage. In der Regel wächst die Dauer einer Anfrage logarithmisch mit der Anzahl der Knoten im Netzwerk. Zusätzlich können DHTs eine Resistenz gegen den Ausfall von Knoten bieten, indem mehrere Knoten gleichzeitig für einen Schlüssel zuständig sind.

Es gibt verschieden Ausprägungen von DHTs und sie werden in vielen Arbeiten[14, 11, 12] ausführlich beschrieben. Alle basieren auf einem Prinzip, das dem obigen ähnlichen ist. Eine DHT lässt sich verwenden, um Objekte auf die im Netzwerk vorhandenen Knoten zu verteilen, indem jedem Objekt ein Schlüssel zugewiesen wird und dieses Objekt dann von dem Rechner verwaltet wird, der für den Schlüssel zuständig ist. Dies löst eines der

Probleme, die beim Objektmanagement auftreten und wird von den in der nächsten Sektion vorgestellten Systemen in dieser Form verwendet.

2.1.1 SimMud

SimMud[9] simuliert ein einfaches Spiel, das Spieler und Essen enthält. Spieler können sich bewegen und Essen kann zu Teilen gegessen werden.

Es benutzt *Pastry*[12] und das darauf aufbauende *Scribe*[5]. *Scribe* ist ein Overlay Netzwerk für Multicast auf Anwendungsebene, es benutzt die Pfade in *Pastry* um seinen Multicast Baum aufzubauen.

Das Spielfeld, auf dem sich die Spieler und das Essen befinden, wird in feste Regionen aufgeteilt. Jeder dieser Regionen wird eine feste ID zugewiesen. Will nun ein Spieler eine dieser Regionen betreten, tritt er der Multicast Gruppe dieser Region bei und sendet seine aktuelle Position zusammen mit einer fortlaufenden Sequenznummer an diese Gruppe. Durch die Sequenznummer ist sichergestellt, dass eventuelle verzögert gelieferte alte Positionen keine neuen überschreiben.

Neben den ständig gesendeten Positionsupdates werden noch zwei weitere Arten von Nachrichten vorgesehen. Zum einen können zwei Spieler direkt miteinander interagieren. Hierbei müssen sich die Spieler einigen und eventuell geänderte Zustände der Umgebung mitteilen. Zum anderen können Spieler mit Objekten interagieren.

Um Objekte konsistent zu halten, existiert für jede Region ein Koordinator. Es ist der Spieler dessen ID in der DHT die niedrigste Distanz zu der ID der Region hat. Spieler senden ihre Änderungen an Objekten zum Koordinator, der sie entweder verwirft oder sie akzeptiert und an den Rest der Spieler per Multicast sendet. Ein Änderung wird verworfen, wenn ein eventuell mitgesendeter alter Wert oder eine Versionsnummer nicht mehr aktuell ist, um kollidierende Änderungen zu vermeiden. Jeder Koordinator hat ein oder mehrere Backups: diejenigen Spieler deren IDs die niedrigste Distanz zu der ID der Region haben. Fällt ein Koordinator aus, wird von der DHT automatisch ein Backup Koordinator gefunden.

Die Auswertung von *SimMud* hat eine gute Skalierung ergeben, solange die Anzahl der Spieler pro Region nicht steigt. Steigt die Anzahl der Spieler in der Region, so steigt die Anzahl der Nachrichten pro Spieler in gleichem Maß. Besonders die Wurzel des Multicast Baumes und der Koordinator einer Region werden stark belastet, wenn sich viele Spieler in dieser Region befinden. Erwähnenswert ist, dass beide belasteten Spieler aufgrund der Beschaffenheit der DHT meist selbst nicht Teil der Region sind und sie sich deshalb selbst nicht für Nachrichten in dieser Region interessieren.

2.1.2 A peer-to-peer architecture for massive multiplayer online games

In *A peer-to-peer architecture for massive multiplayer online games*[7] wird ein Ansatz beschrieben, der ähnlich zu *SimMud* ist. Auch hier wird die Spielwelt in Regionen aufgeteilt, innerhalb derer sich die Spieler mithilfe von *Pastry* und *Scribe* verbinden. Anders als bei

SimMud werden Objekte mithilfe von *Past* in der DHT redundant gespeichert. *Past*[6] ist eine zusätzliche Erweiterung zu *Pastry*, die sich für diesen Verwendungszweck eignet.

Jede Region hat neben einem Koordinator eine Reihe an Backup Koordinatoren. Das Spiel ist in einzelne Züge aufgeteilt, in jedem Zug senden die Spieler ihre Updates an alle Koordinatoren. Einer der Koordinatoren sendet den geänderten Zustand an die anderen Spieler, die restlichen Koordinatoren senden eine Prüfsumme ihres unabhängig berechneten Zustandes. Durch dieses System können Anomalien oder Betrugsversuche eines Koordinators erkannt werden. Es ist vorgesehen, dass der Koordinator bei jedem Zug wechselt, um wiederkehrend falsche Zustände eines Koordinators zu vermeiden. Gleichzeitig können Koordinatoren, die abweichende Zustände oder Prüfsummen senden, erkannt und eventuell ausgeschlossen werden. Auch soll durch das Abwechseln die Last verteilt werden, die beim Senden entsteht.

Eine explizite Evaluation ist zwar nicht vorhanden, aber diese erhöhte Resistenz gegen Angriffe oder Fehler lässt auf ein erhöhtes Nachrichtenaufkommen schließen, wodurch die bei *SimMud* beschriebenen Probleme verstärkt werden, wenn sich viele Spieler in einer Region befinden. Auch das Austauschen der jeweils sendenden Koordinatoren reduziert nur die mittlere Bandbreite, die Bandbreitenspitzen in einer Runde bleiben erhalten und können einen bestimmten Koordinator überfordern.

2.2 *Colyseus*

Colyseus[4] ist ein System, das bekannte Eigenschaften von Spielen für verschiedene Optimierungen ausnutzt. Zum einen ist es in Spielen nicht erforderlich, dass der Zustand zu jedem Zeitpunkt konsistent ist. Schlägt die Übertragung der Position eines Objektes fehl, ist es zum Beispiel nicht erforderlich die alte Position erneut zu senden, es reicht die neue Position zu übermitteln. Fehlende Daten können in der Zwischenzeit interpoliert oder einfach übersprungen werden, ohne das Spiel sonderlich zu beeinträchtigen. Zum anderen sind Zustandsänderungen in einem Spiel sehr gut vorhersehbar. Ein Spieler kann zum Beispiel meist keine unstetigen Positionsänderungen durchführen, jede neue Position ist immer in der Nähe der alten Position. Außerdem interessiert sich ein Objekt nur für andere Objekte in seiner Nähe, beziehungsweise kann vorhersagen für welche Objekte es sich interessieren wird.

Um Objekte konsistent zu halten, hat auch *Colyseus* Koordinatoren für die Objekte. Anders als bei den vorherigen Systeme ist ein Koordinator nicht mehr für eine ganze Region zuständig, sondern besitzt bestimmte Objekte, ohne dass diese in irgendeiner Form zusammenhängen müssen. Um den Besitzer eines Objektes zu finden, verwendet *Colyseus* ein System namens *Mercury*[3]. *Mercury* funktioniert ähnlich wie eine DHT (mit range query), erlaubt aber das Suchen nach einem ganzen Bereich von Attributen im Gegensatz zu Schlüsseln. Dadurch ermöglicht *Mercury* eine effiziente Suche nach einem zusammenhängenden Bereich eines Attributes, hier zum Beispiel nach einem Intervall der Position auf der x-Achse eines Objektes. Außerdem ist es mit *Mercury* möglich nach

mehreren Attributen gleichzeitig zu suchen, wodurch ein zusammenhängender Raum im Spiel durchsucht werden kann. Ein Knoten muss nun nur den Bereich bestimmen in dem sich Objekte befinden können für die er sich interessiert, und kann dann mit Hilfe von *Mercury* die Besitzer von Objekten in diesem Bereich finden. Kennt ein Knoten den Besitzer eines Objektes, lässt er sich von diesem eine Kopie des Objektes zusenden und erhält in Zukunft direkt Updates über Änderungen dieses Objektes. Die verschiedenen Kopien können zwar durch verzögerte Updates etwas veraltet sein, aber durch die zu Beginn beschriebene Toleranz gegenüber kleinen Inkonsistenzen, entstehen dadurch keine Probleme, da meist nach kurzer Zeit wieder ein Update mit dem konsistenten Zustand des Originals empfangen wird. Will ein Knoten ein Objekt ändern, das er nicht besitzt, sendet er seine Änderungen auch direkt an den Besitzer.

Colyseus nutzt die anfangs genannten Eigenschaften aus, um eine bessere Verteilung der Last zu erzielen, und kann eine messbare Steigerung der Leistungsfähigkeit gegenüber einem einzelnen Server vorweisen. Es bleiben aber immer noch Probleme übrig. Zum einen kann sich der Besitzer eines Objektes nicht ändern. Dadurch kann es dazu kommen, dass sich auf einem Knoten viele Objekte befinden, deren Positionen innerhalb der Spielwelt weit verteilt sind. Dadurch wird der Bereich in dem dieser Knoten Objekte kennen muss größer und damit auch die Anzahl an Kopien, die er erhalten muss. Die Eigenschaft, dass Objekte sich nur für andere Objekte in ihrer Nähe interessieren, wird also nicht mehr so gut ausgenutzt. Zum anderen dürfen keine Knoten ausfallen, da sonst die Objekte, die auf diesem Knoten gespeichert sind, nicht mehr aktualisiert werden können. Dies ist nur bei Spielen praktikabel deren Runden kurz genug sind, so dass der Ausfall eines Knotens sehr unwahrscheinlich ist, oder indem man nur sehr zuverlässige Rechner als Knoten nutzt, über die sich die Spieler indirekt verbinden.

2.3 Gemeinsamkeiten

Die vorgestellten Verfahren bauen ein Netzwerk auf, in dem sich die Spieler entsprechend einer künstlichen Verwaltungsstruktur verbinden. Innerhalb dieses Netzwerkes wird jedem Objekt ein Spieler zugewiesen, der dafür zuständig ist, das Objekt zu verwalten. Die Wahl dieses Spielers ist sehr wichtig, da dadurch bestimmt wird, wie Änderungen an einem Objekt im Netzwerk verteilt werden. In den obigen Verfahren wird um einen Besitzer zu finden eine zusätzliche Struktur verwendet, die unabhängig von dem eigentlichen Spiel ist. Dies erlaubt zwar ein recht einfaches Umgehen mit Besitzern, erfordert aber zusätzliche Verwaltungsarbeit. Der nächste Abschnitt stellt ein Verfahren vor, das ebenfalls mit Objekte und Besitzern arbeitet, zum Finden eines geeigneten Besitzers aber keine zusätzliche Struktur verwendet, sondern Eigenschaften von Spielern und Objekten im Spiel selber ausnutzt.

3 Algorithmus

Dieser Abschnitt befasst sich mit der Funktionsweise des in dieser Arbeit behandelten Objektmanagements auf einer abstrakteren Ebene. Dazu werden zunächst die Begriffe *Objekt*, *Sichtbereich* und *Interaktionsbereich* eingeführt, danach werden damit die allgemeinen Ziele beschrieben, die erreicht werden sollen.

Der zweite Teil dieses Abschnitts beschäftigt sich mit den Konzepten des hier vorgestellten Objektmanagements. Dazu wird ein Interest Management System als Voraussetzung vorgestellt, und auf dieser Basis *Interesse* als zusätzliche Information eingeführt. Das Interesse erlaubt es, jedem Objekt einen Besitzer zuzuweisen, der für dessen Verwaltung zuständig ist. Zuletzt wird betrachtet wie Spieler mit diesen Objekten interagieren können.

Kapitel 4 beschäftigt sich danach mit der konkreten Umsetzung dieses Algorithmus.

3.1 Objekt

Ein Objekt ist etwas, das ein Spieler in der Spielwelt wahrnehmen und mit dem er interagieren kann. Während es nicht jedem Spieler zu jedem Zeitpunkt möglich ist jedes Objekt wahrzunehmen, existieren dennoch für alle Spieler die gleichen Objekte innerhalb der Welt. Objekte, die nur für eine Teilmenge der Spieler existieren, werden hier nicht weiter betrachtet und Objekte, die nur für einen einzelnen Spieler existieren, müssen nicht im Netzwerk verteilt werden. Vom Objektmanagement werden nur Objekte behandelt deren Zustand sich während der Laufzeit verändert und die von den Spielern beeinflusst werden können. Alle anderen Objekte sind statisch und können bereits im Vorfeld ausgetauscht werden.

3.2 Sichtbereich und Interaktionsbereich

Jeder Spieler hat einen ihn umgebenden Bereich, in dem er andere Spieler und Objekte wahrnehmen kann. Die Form dieses *Sichtbereiches* (area of interest, AOI) kann unterschiedlich sein. Beispiele sind eine Kugel oder ein Quader um den Spieler herum, der Sektor in dem sich der Spieler befindet oder die Teile des Spielfeldes, die vom Spieler innerhalb einer bestimmten Zeit aus erreichbar sind. *A Spatial Model of Interaction in Large Virtual Environments*[2] beschäftigt sich genauer mit Sichtbereichen. Im Folgenden wird ein kugel- beziehungsweise kreisförmiger Sichtbereich angenommen.

Der *Interaktionsbereich* ist ein echter Teilbereich des Sichtbereiches, innerhalb von dem der Spieler Objekte beeinflussen kann. Dadurch ist sichergestellt, dass ein Spieler ein

Objekt sehen kann bevor er mit ihm interagiert, da das Objekt zuerst den Sichtbereich des Spielers betreten muss, bevor es den Interaktionsbereich betritt. Abbildung 3.1 zeigt eine schematische Darstellung von Sichtbereich und Interaktionsbereich.

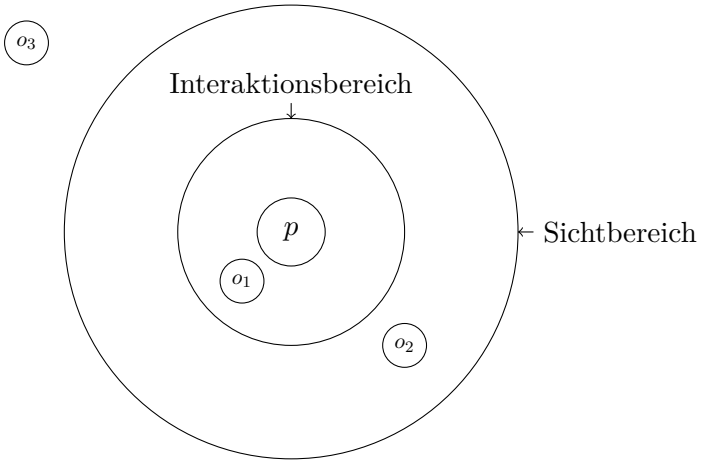


Abbildung 3.1: Schematische Darstellung des Sichtbereiches und Interaktionsbereiches eines Spielers p. Die Objekte o_1 und o_2 sind sichtbar und p kann mit o_1 interagieren. o_3 ist nicht sichtbar.

3.3 Ziele

Zu einem bestimmten Zeitpunkt sollte der Zustand eines Objektes für alle Spieler, die dieses Objekt sehen können, gleich sein. Und Änderungen sollten zeitnah bei Spielern ankommen. Um die Leistungsfähigkeit zu erhöhen, wird eine Abweichung des Zustandes der Objekte für verschiedene Spieler zugelassen. Es muss garantiert werden, dass diese Abweichungen nicht zunehmend größer werden, sondern immer gegen einen gemeinsamen Zustand konvergieren.

Ein Objekt ist nicht für jeden Spieler gleich wichtig und die Wichtigkeit kann sich im Laufe der Zeit ändern. Je wichtiger ein Objekt für einen Spieler ist, desto wichtiger ist es auch, dass dieser Benutzer den Zustand genau kennt. Dies gilt besonders für Objekte mit denen der Benutzer interagieren kann, da der Zustand eines Objektes die Interaktionsmöglichkeiten beeinflusst. In der anderen Richtung kann ein Benutzer so wenig an einem Objekt interessiert sein, dass er dessen Zustand überhaupt nicht mehr zu kennen braucht.

Diese Eigenschaften sollen auch erhalten bleiben, wenn es zu Fehlern innerhalb des Netzwerkes kommt und Spieler unerwartet die Verbindung trennen.

3.4 Wissen des Interest Managements ausnutzen

Der hier vorgestellte Algorithmus baut auf einer vorhandenen positionsbasierten Netzwerkstruktur (*Interest Management*) auf. Diese muss Spielern den Sichtbereich[1] von anderen Spielern innerhalb ihres Sichtbereiches zur Verfügung stellen und es Spielern innerhalb des Sichtbereiches erlauben, miteinander zu kommunizieren. Unterstützt das Netzwerk eine effiziente Methode, um Nachrichten an alle Spieler innerhalb des Sichtbereiches zu senden (*Multicast*), kann dies ausgenutzt werden, um Objektnachrichten zu verteilen, es ist aber nur notwendig Nachrichten an einzelne Spieler senden zu können. Die Basis ist also eine simple Peer-to-Peer Architektur in der jeder Spieler nur seine direkten Nachbarn kennt und kein globales Wissen über das Netzwerk existiert. Beispiele für geeignete Netzwerke sind *VON*[8] und *pSense*[13]. Beide Beispiele stellen eine direkte Verbindung zwischen allen Spielern innerhalb des Sichtbereiches her, halten das Netzwerk aber auch bei niedriger Spielerdichte verbunden, indem zusätzliche Verbindungen außerhalb des Sichtbereiches aufgebaut werden.

3.5 Interessensfunktion

Jedem Paar aus Spieler und Objekt wird ein Wert zugewiesen, der beschreibt wie sehr dieser Spieler an dem Objekt interessiert ist. Dieses Interesse kann unterschiedlich berechnet werden, muss aber einige Kriterien erfüllen. Zum einen muss das Interesse für ein Objekt innerhalb des Interaktionsbereiches größer sein als das Interesse eines Objektes, das sich nur innerhalb des Sichtbereiches befindet, das Interesse für dieses zweite Objekt muss aber auch wieder größer sein als für ein Objekt außerhalb des Sichtbereiches. Wenn also $I(p, o)$ das Interesse von Spieler p an Objekt o angibt und die drei Objekte o_1, o_2 und o_3 wie in Abbildung 3.1 angeordnet sind, dann muss gelten $I(p, o_1) > I(p, o_2) > I(p, o_3)$. Im Folgenden heißt es, dass sich ein Spieler für ein Objekt interessiert, wenn sich das Objekt innerhalb seines Sichtbereiches befindet und die Interessenten eines Objektes alle Spieler sind, die sich für dieses Objekt interessieren.

Neben diesen grundlegenden Bedingungen beeinflusst die Wahl des Interesses die Effizienz des Objektmanagements, da Nachrichten je nach Interesse priorisiert werden können. Dazu werden im nächsten Abschnitt einige Auswahlkriterien vorgestellt.

3.6 Kriterien zur Interessenbestimmung

Die Wahl des Interesses erfolgt nach folgenden Kriterien.

Für einen einzelnen Spieler sollte das Interesse hauptsächlich beschreiben, wie sehr der Spieler mit diesem Objekt interagieren möchte und wie wichtig Änderungen an dem Objekt für ihn sind. Tendenziell sollte ein Spieler schneller von Änderungen an Objekten,

1 Bei einem festen Sichtbereich reicht also die Spielerposition

für die er sich stärker interessiert, erfahren und er kann effizienter Änderungen an ihnen durchführen. Dafür nimmt er jedoch in Kauf, mehr für ihn unwichtige Informationen zu bekommen, da ein höheres Interesse häufigere Aktualisierungen bedeuten kann, wodurch die Geschwindigkeit sinken kann mit der dieser oder andere Spieler Änderungen für wichtige Objekte erhalten.

Für eine Gruppe von Spielern ist es wichtig, dass alle diejenigen, die ein großes Interesse an dem selben Objekt haben, effizient miteinander kommunizieren können. Ist dies nicht gegeben, kann es zu verzögerten Änderungen oder gar einem inkohärenten Zustand kommen.

Außerdem ist es vorteilhaft, wenn ein Spieler das Interesse eines beliebigen anderen Paares aus Spieler und Objekt berechnen oder zumindest schätzen kann, ohne zusätzliche Informationen mit diesem Spieler austauschen zu müssen.

Für eine zusammenhängende Spielwelt wie sie häufig vorkommt, bietet es sich an die Distanz, die zwischen dem Spieler und dem Objekt besteht, zu verwenden. Dies ist dadurch begründet, dass ein Spieler meist nur Objekte in seiner Nähe direkt beeinflussen kann. Daher müssen sich alle Spieler, die das Objekt beeinflussen könnten, über ihre Positionen austauschen. Ein Spieler kann somit anhand der Position eines Objektes sowie der Position des anderen Spielers das Interesse dieses Spielers schätzen.

3.7 Aufgaben des Besitzers

Um den Zustand eines Objektes zu verwalten, wird jedem Objekt ein Besitzer zugewiesen. Dieser ist dafür zuständig die Spiellogik für dieses Objekt auszuführen und den Zustand des Objektes entsprechend anzupassen.

Updates werden vom Besitzer an alle anderen Spieler in seinem Sichtbereich gesendet. Dies geschieht wie in Abschnitt 4.5 beschrieben. Ist die Bandbreite eines Spielers erschöpft, können Updates an Spieler, die sich nicht für das Objekt interessieren, eingestellt werden und Updates an Spieler, die sich nur wenig für das Objekt interessieren, verlangsamt werden. Dadurch erhalten die Spieler, die sich sehr für ein Objekt interessieren, weiterhin häufig Updates.

3.8 Interaktion mit Objekten

Will ein Spieler mit einem Objekt interagieren, so sendet er die gewünschte Aktion an den Besitzer des Objektes. Der Besitzer wendet die Aktion an und aktualisiert den Zustand des Objektes entsprechend. Den Zustand des Objektes direkt zu ändern, hätte einige Nachteile für den Spieler. Zum einen kann eine Aktion oft effizienter kodiert werden als der gesamte Zustand, was zu einem geringeren Datentransfer führt. Zum anderen entspricht eine Aktion einer bestimmten Änderung an dem Objekt. Senden also mehrere Spieler gleichzeitig Aktionen an den Besitzer, hat dieser die Chance die Aktionen in einer von ihm gewählten Reihenfolge auszuführen und einen neuen Zustand zu berechnen, der

alle Aktionen beinhaltet. Dem entgegen würde ein direktes Anpassen des Zustandes von anderen Spielern viel eher zu Konflikten führen.

3.9 Besitzerwahl

Der Besitzer eines Objektes muss im Spiel eindeutig gewählt werden, dafür gibt es verschiedene Möglichkeiten. Eine Alternative ist, dass ein Spieler alle Objekte besitzt. Dies ist sehr ähnlich zu einer Client/Server Architektur und würde deren Nachteile übernehmen (Ausfallrisiko, hohe Kosten für Serverbetreiber). Dann könnten Objekte fest einem bestimmten Spieler zugeordnet werden, zum Beispiel über die IDs von Spielern und Objekten oder über die Region in der sich ein Objekt befindet, was den in Kapitel 2 vorgestellten Systemen entspricht. Im Folgenden wird jedoch eine andere Strategie vorgeschlagen.

Änderungen an Objekten durchzuführen ist für den Besitzer des Objektes am günstigsten, es ist dazu keine zusätzliche Nachricht notwendig. Deshalb sollte der Besitzer eines Objektes der Spieler sein, der die meisten Änderungen an diesem durchführen möchte. Um vorherzusagen welcher Spieler dies sein wird, wird das Interesse eines Spielers an dem Objekt verwendet. Der Besitzer eines Objektes sollte also derjenige Spieler sein, der das größte Interesse an diesem Objekt hat[1].

Zu Beginn ist der Besitzer eines Objektes der Spieler, der es zur Welt hinzugefügt hat, da davon ausgegangen wird, dass ein Spieler die Welt nur in seiner näheren Umgebung beeinflusst, das Objekt also auch in seiner Nähe erzeugt wurde.

Findet der aktuelle Besitzer einen anderen Spieler mit einem größeres Interesse an dem Objekt, so übergibt er den Besitz des Objektes zusammen mit der nächsten Aktualisierung des Zustandes des Objektes.

Da jeder Spieler eine vollständige Kopie des Objektes verwaltet, kann beim Ausfall eines Besitzers jeder andere Spieler als möglicher Ersatz fungieren. Um den neuen Besitzer eindeutig zu bestimmen, kann wieder das geschätzte Interesse verwendet werden. Durch falsche Schätzungen kann es dabei zu mehreren neuen Besitzern kommen. Wie ein solcher Konflikt gehandhabt wird ist in Abschnitt 4.8 beschrieben.

1 Es ist ausreichend, wenn es keinen Spieler gibt, der ein viel größeres Interesse an dem Objekt hat.

4 Umsetzung in Planet PI4

Dieser Abschnitt befasst sich mit der Umsetzung des in Kapitel 3 beschriebenen Systems. Die Umsetzung erfolgte als Netzwerkmodul für *Planet PI4* [10], das einen geeigneten Rahmen in Form eines Mehrspieler Weltraumshooters bietet. Die Implementierungssprache ist *C++*.

In den folgenden Abschnitten werden zuerst Objekte eingeführt wie sie vom System verwaltet und verwendet werden, um dann in Abschnitt 4.3 einen Überblick über das gesamte System zu geben. Wie Objekte verwaltet werden wird in Abschnitt 4.4 und Abschnitt 4.5 erklärt. Die restlichen Unterabschnitte befassen sich dann mit dem Finden und Verwalten von Besitzern.

4.1 Objekt

Für das Netzwerk enthält ein Objekt drei wichtige Informationen, eine im Netzwerk eindeutige ObjectID um das Objekt zu identifizieren, den aktuellen Besitzer des Objektes und den Zustand (die Nutzlast) des Objektes, bestehend aus einer Liste von anwendungsabhängigen Einträgen. Die Anwendung sollte die Einträge so verwenden, dass Daten, die sich unabhängig voneinander ändern können, auch in verschiedenen Einträgen des Objektes gespeichert werden. Dies ist nicht nur eine Vereinfachung des Datenzugriffs für die Anwendung, sondern es erlaubt dem Objektmanagement auch zu erkennen, welche Teile des Zustandes sich geändert haben. Dies könnte zum Beispiel für partielle Updates genutzt werden.

4.2 Objekt ID

Die ID eines Objektes ist ein Bitmuster fixer Länge, anhand dessen dieses Objekt eindeutig identifiziert wird. Unabhängig vom etwaigen Zustand zweier Objekte werden diese genau dann als gleich betrachtet, wenn sich ihre IDs nicht unterscheiden. Im Folgenden wird von 64 Bit pro ID ausgegangen, aber solange es genügend IDs gibt, um alle möglichen Objekte eindeutig zu identifiziert, ist jede beliebige fixe Länge möglich. Dies kann ausgenutzt werden, um Informationen in der ID zu kodieren. Da zwei Objekte von unterschiedlichem Typ nicht gleich sind, lässt sich so der Typ eines Objektes direkt aus der ID ablesen und muss nicht zusätzlich kodiert werden. Auf gleiche Weise können andere statische Informationen über Objekte direkt aus der ID gewonnen werden.

4.3 Systemüberblick

Dieser Unterabschnitt gibt einen Überblick über das gesamte System. Er soll dazu dienen, die Erklärungen in den folgenden Abschnitten einordnen zu können und die Zusammenhänge zwischen den einzelnen Komponenten zu verstehen. In Abbildung 4.1 findet sich eine graphische Darstellung des Systems.

Aus Sicht eines Spiels hat das Objektmanagement zwei wichtige Schnittstellen. Zum einen `IActiveObjectManagement` das vom `ObjectManager` implementiert wird und es dem Spiel erlaubt neue Objekte vom Objektmanagement verwalten zu lassen. Zum anderen `IActiveObjectHandler` der vom Spiel selbst durch den `ActiveObjectHandler` implementiert wird und es dem Objektmanagement erlaubt das Spiel über neue Objekte zu informieren, so wie auf bestimmte Eigenschaften von Objekten (wichtig ist die Position des Objektes in der Spielwelt) zuzugreifen. Das Spiel kann ihm bekannte Objekte über die Schnittstelle `IActiveObject` untersuchen und verändern.

Der `ObjectManager` verwaltet die verteilten Objekte (`DistributedObject`) und delegiert verschieden Aufgaben an die anderen Teile des Objektmanagements wie es in den folgenden Unterabschnitten erklärt wird. Allgemein ist der `DistributionManager` dafür verantwortlich Nachrichten im Netzwerk zu verteilen, die dann vom `MessageLlistener` empfangen und bearbeitet werden, und der `InterestManager` verwaltet alles was mit dem *Interesse* und *Besitz* von Objekten zu tun hat. Da das Objektmanagement auf einer vorhandene Netzwerkstruktur aufbaut wie sie in Abschnitt 3.4 beschrieben wird, greifen die obigen Komponenten auf die Schnittstelle `ISpatielMulticast` zu, deren primäre Funktionen das Senden sowie Empfangen von Nachrichten von und an andere Spieler sind.

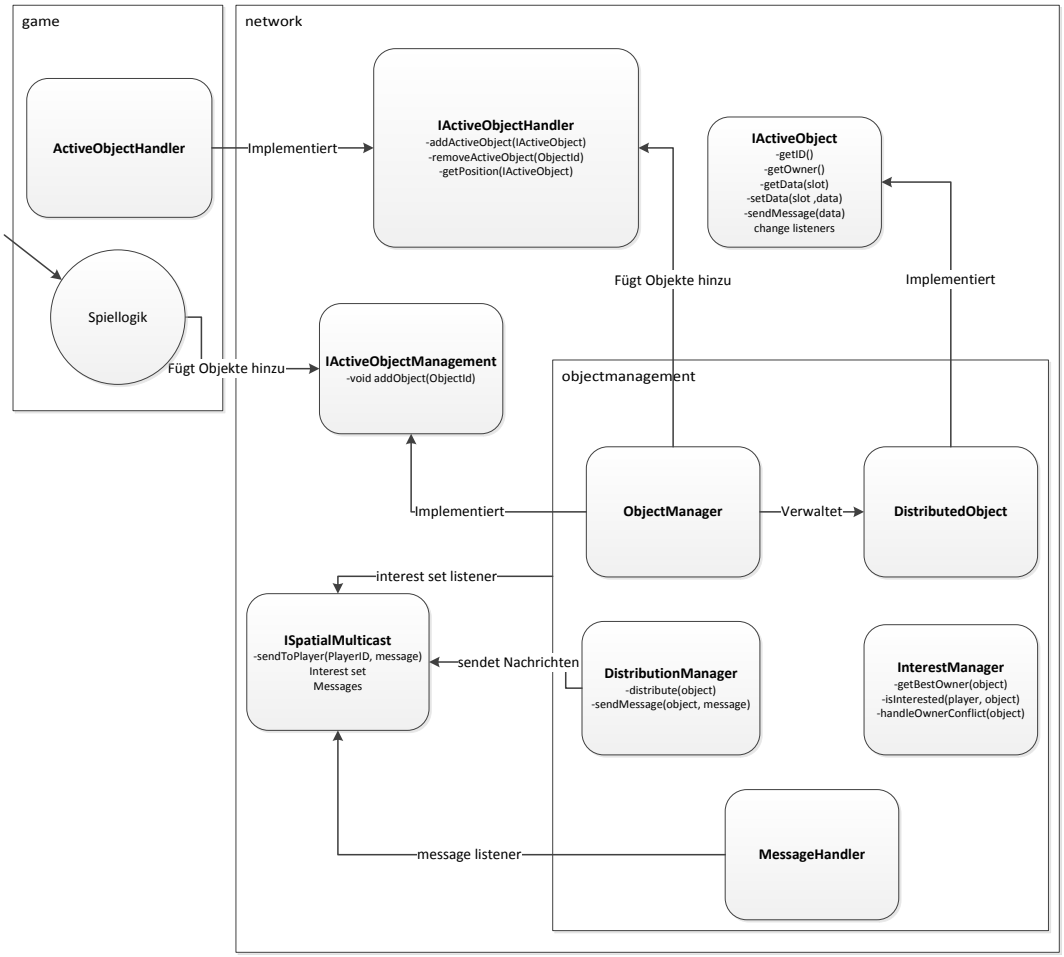


Abbildung 4.1: Schematische Darstellung des gesamten Systems. Das Spiel (*game*) wird über die beiden Schnittstellen `IActiveObjectHandler` und `IActiveObjectManagement` mit dem Objektmanagement verbunden. Das Objektmanagement selbst verwendet wieder eine Schittstelle `ISpatialMulticast` um auf die Basisnetzwerkfunktionen zuzugreifen. Eine textueller Überblick über das System wird in Abschnitt 4.3 gegeben. Abbildung 4.6 bietet einen vergrößerten Ausschnitt des Objektmanagements.

4.4 Erstellung von Objekten

Bevor ein Objekt verwaltet werden kann, muss es dem Netzwerk hinzugefügt werden. Dieser Vorgang wird zusätzlich zu der Beschreibung in diesem Unterabschnitt in Abbildung 4.2 veranschaulicht. Außerdem stellt Abbildung 4.6 die groben Zusammenhänge zwischen den

in diesem Unterabschnitt erwähnten Komponenten dar.

Das Spiel kann ein neues Objekt mit von ihm gewählter ID und Nutzdaten erstellen und an den `ObjectManager` übergeben, damit dieser es verwaltet. Der Besitzer des Objektes wird ausschließlich vom Objektmanagement verwaltet, im Gegensatz zur ID, die vom Spiel festgelegt wird. Nachdem das Spiel ein Objekt an den `ObjectManager` übergeben hat, behält es selbst keine Referenz auf dieses Objekt. Das Objektmanagementinterface trifft keine Unterscheidung zwischen dem ursprünglichen Ersteller eines Objektes und allen anderen Knoten. Der `ObjectManager` lässt dann ein normales Update vom `DistributionManager` verteilen. Dieses Update wird von den `MessageHandlern` aller Interessenten, einschließlich dem Sender, empfangen. Die Empfänger erkennen anhand der ID, dass sie ein neues Objekt erhalten haben und weisen den `ObjectManager` dazu an dieses Objekt zu verwalten. Ab diesem Zeitpunkt gilt das Objekt für das Objektmanagement als erfolgreich erstellt. Noch befindet sich das Objekt aber nur im Netzwerk, nicht innerhalb der Spielwelt (dies gilt auch für den Knoten auf dem das Objekt ursprünglich erstellt wurde). Um eine Repräsentation des Objektes innerhalb der Spielwelt kümmert sich der `ActiveObjectHandler`. Anhand der Daten des Objektes wird eine entsprechende Repräsentation im Spiel erstellt, die auf dem Zustand des verteilten Objektes basiert.

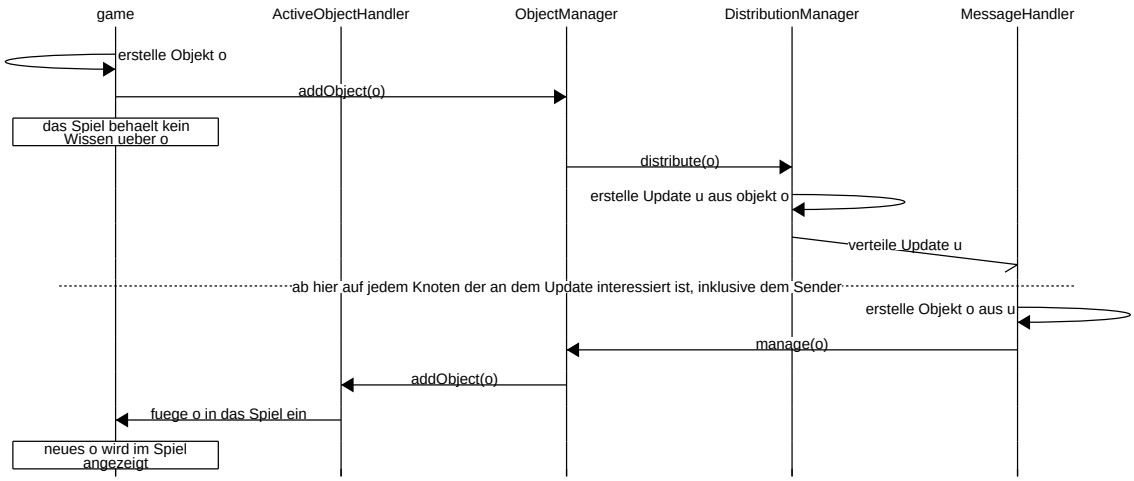


Abbildung 4.2: Ein neues Objekt wird vom Spiel erstellt und anschließend dem Netzwerk übergeben. Dort wird das Objekt an alle Interessenten verteilt. Bei diesen wird das Objekt lokal erstellt und dem Spiel hinzugefügt.

4.5 Aktualisierung von Objekten

Wie in Abschnitt 4.1 beschrieben, besteht der Zustand eines Objektes aus einer Liste von Daten. Diese können durch den Besitzer oder einen der Interessenten verändert werden. Zusätzlich erfolgt eine Aktualisierung wenn sich der Besitzer eines Objektes ändert, die versendete Nachricht ist für Änderungen von Zustand und Besitzer die selbe. Es ist einem Spieler nicht möglich ein Objekt zu ändern, an dem er nicht interessiert ist.

4.5.1 Aktualisierung durch den Besitzer

Wird einer dieser Einträge vom Besitzer des Objektes verändert, muss diese Änderung im Netzwerk verbreitet werden. Um die Anzahl der nötigen Updates zu verringern, löst nicht jede Änderung an einem Eintrag sofort ein Update aus. Stattdessen werden mehrere Änderungen gesammelt und zu einem einzelnen Update zusammengefasst. Wie in Abbildung 4.3 dargestellt, ist der ObjectManager dafür verantwortlich zu entscheiden, wann ein Update verteilt werden soll. Dieses Update wird an alle Interessenten außer den Besitzer verteilt. Der MessageHandler des jeweiligen Empfängers wendet die Änderungen auf das zugehörige lokale Objekt an. Erhält ein Knoten ein Update zu einem Objekt, welches ihm nicht bekannt ist, so behandelt er dieses wie in Abschnitt 4.4 beschrieben und erzeugt ein neues Objekt.

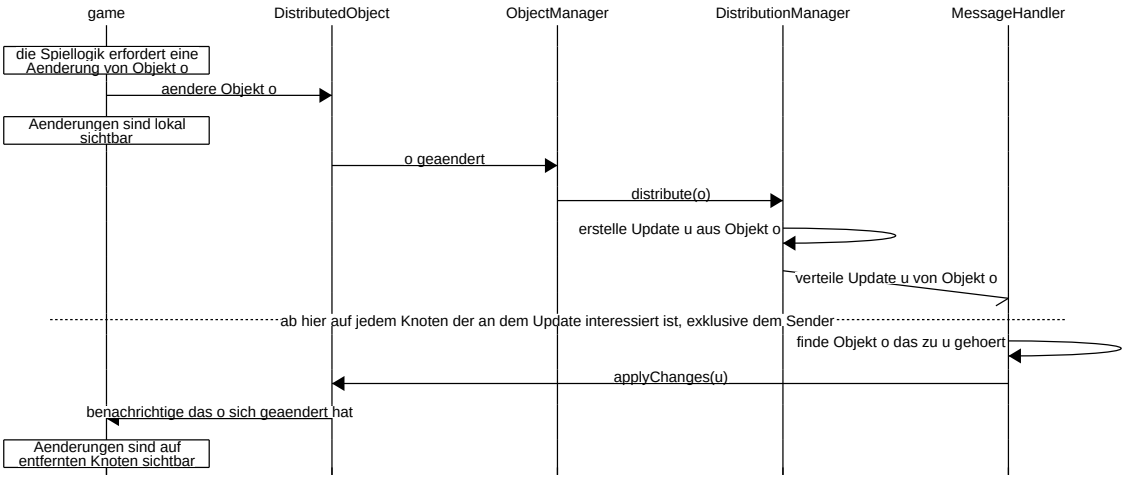


Abbildung 4.3: Die Spiellogik berechnet einen neuen Zustand für ein Objekt. Der `ObjektManager` wird über diese Änderung informiert und sorgt dafür, dass sie an alle Interessenten verteilt wird. Bei diesen werden die Änderungen auf die lokale Kopie des Objektes angewandt und das Spiel über die Änderung informiert.

4.5.2 Aktualisierung durch einen Interessenten

Für alle Interessenten außer dem Besitzer gilt, dass diese die Daten des Objektes nicht direkt verändern. Stattdessen senden sie Nachrichten an die Kopie des Objektes des Besitzers. Diese Nachrichten werden von der Spiellogik behandelt und können ihrerseits zu Änderungen des Objektes führen. Dieses Vorgehen garantiert, dass alle Änderungen durch den Besitzer geprüft sind, bevor sie durchgeführt werden, und erlaubt es mit unterschiedlichen Änderungen aus verschiedenen Quellen umzugehen. Eine Diskussion der Vor- und Nachteile findet sich in Abschnitt 3.8.

4.6 Besitzer und Interesse

Der Besitzer wird nach den in Abschnitt 3.9 genannten Kriterien gewählt. Das bedeutet, der Besitzer eines Objektes sollte derjenige Spieler sein, der das größte Interesse an einem Objekt hat. Das Interesse berechnet sich direkt aus dem Abstand zwischen dem Spieler und dem Objekt, oder genauer, das Interesse ist die negative euklidische Distanz zwischen Spieler und Objekt wie in Abbildung 4.4 dargestellt.

Eine Eigenschaft dieser Wahl des Interesses ist, dass der Besitzer eines Objektes immer in dessen Nähe ist. Dadurch kann die zugrundeliegende Netzwerkstruktur ausgenutzt werden, da die Verbindungen zwischen Knoten auch anhand der räumlichen Positionierung der Spieler aufgebaut wird, ein Besitzer also direkte Verbindungen zu anderen Spielern hat, die sich für das Objekt interessieren.

Außerdem kennt ein Spieler die Positionen seiner Nachbarn und kann deren Interesse an einem Objekt abschätzen. Ein Besitzer kann direkt erkennen, wenn es einen anderen

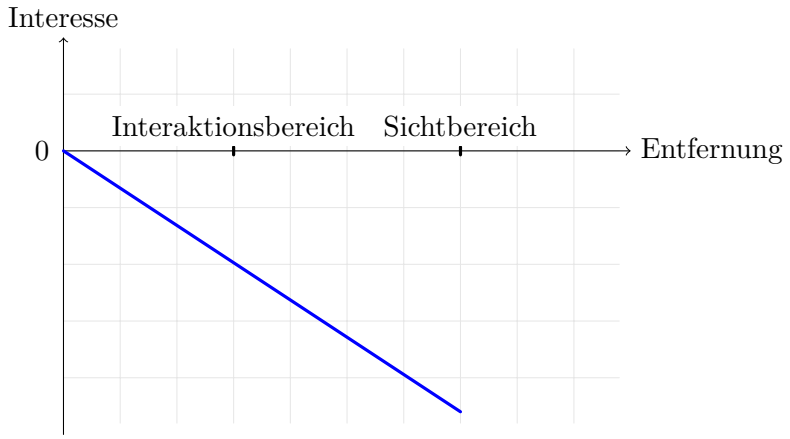


Abbildung 4.4: Darstellung der verwendeten Interessensfunktion. Das Interesse ist proportional zur negativen Entfernung. Mit steigender Entfernung sinkt also das Interesse das ein Spieler an einem Objekt hat.

Spieler gibt, dessen Interesse größer ist. In diesem Fall bietet es sich an, den Besitz des Objektes abzugeben. Da sich jedoch die Positionen und damit das Interesse von Spielern an Objekten ständig verändern, wird ein Objekt erst abgegeben, wenn das Interesse des neuen Besitzer deutlich größer ist als das Interesse des alten. Dies verhindert ein zu häufiges Wechseln des Besitzers, wenn zwei Spieler ein sehr ähnliches Interesse an einem Objekt haben.

4.7 Ausfall eines Besitzers

Fällt ein Spieler im Netzwerk aus, müssen die Objekte, die er besitzt, neu verteilt werden. Da jeder Spieler, der sich für ein Objekt interessiert, eine Kopie dieses Objektes hat, ist es jedem dieser Spieler möglich der neue Besitzer zu werden. Jeder interessierte Spieler ändert den Besitzer des Objektes lokal zu demjenigen Spieler, der das höchste geschätzte Interesse an dem Objekt hat. Dabei kann es vorkommen, dass zwei verschiedene Spieler den Besitz über ein Objekt übernehmen. Wie der dadurch entstehende Konflikt behandelt wird, ist im nächsten Unterabschnitt beschrieben.

4.8 Besitzerkonflikte

Es kann dazu kommen, dass zwei Spieler zur gleichen Zeit glauben, der Besitzer eines Objektes zu sein. Dies kann geschehen, wenn ein neuer Knoten das Netzwerk betritt oder sich mehrere Spieler gleichzeitig zu einem neuen Teil der Welt bewegen, da in beiden Fällen jeder Spieler die Objekte, die sich in dem betretenen Teil der Welt befinden, lokal neu erstellt. Außerdem kann es vorkommen, dass zwei andere Spieler gleichzeitig den Besitz über eines der Objekte übernehmen, wenn ein Knoten das Netzwerk verlässt.

In jedem der Fälle wird ein Konflikt von Besitzern gleich gehandhabt. Dies ist in Abbildung 4.5 dargestellt. Empfängt Spieler B ein Update zu einem Objekt, das er besitzt, geht er davon aus, dass der sendende Spieler A das Interesse von B falsch schätzt. Deshalb sendet B sein aktuelles Interesse an A. A kennt nun beide Werte und kann entscheiden, wer der beste Besitzer ist. Wird B als neuer Besitzer akzeptiert, sendet A keine weiteren Nachrichten, ansonsten sendet A sein Interesse an B, damit dieser den Besitz abgeben kann.

Je nach Aufbau des Interesses könnte es dazu kommen, dass zwei Spieler das gleiche Interesse an einem Objekt haben. In diesem Fall wird zusätzlich ein zufälliger Wert ausgetauscht, der den gleichen Interessenswerten wieder eine eindeutige Ordnung zuweist. Da die verwendete Interessensfunktion jedoch reelle Werte liefert, ist ein solcher Konflikt unwahrscheinlich. Selbst wenn die Distanz von zwei Spielern zu einem Objekt zu einem Zeitpunkt gleich ist, löst sich dieses Problem durch die Bewegung von Spielern und Objekten meist schnell wieder. Bewegen sich die Spieler nicht mit exakt gleicher Geschwindigkeit relativ zu dem Objekt, ändert sich das Interesse ständig. Dass beide Spieler das gleiche Interesse haben, wird dabei immer unwahrscheinlicher.

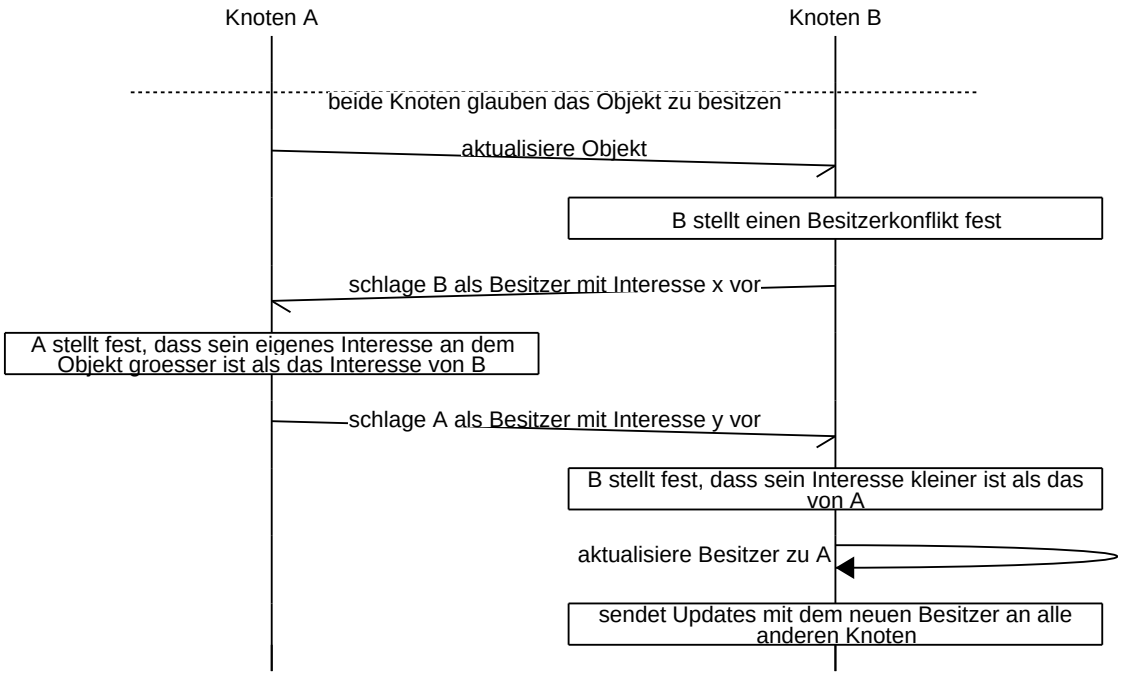


Abbildung 4.5: Ein Besitzerkonflikt tritt auf, wenn zwei Knoten gleichzeitig glauben ein Objekt zu besitzen. Der Konflikt wird gelöst, indem beide Knoten ihr Interesse an dem jeweiligen Objekt austauschen und damit einen eindeutigen Besizter ermitteln.

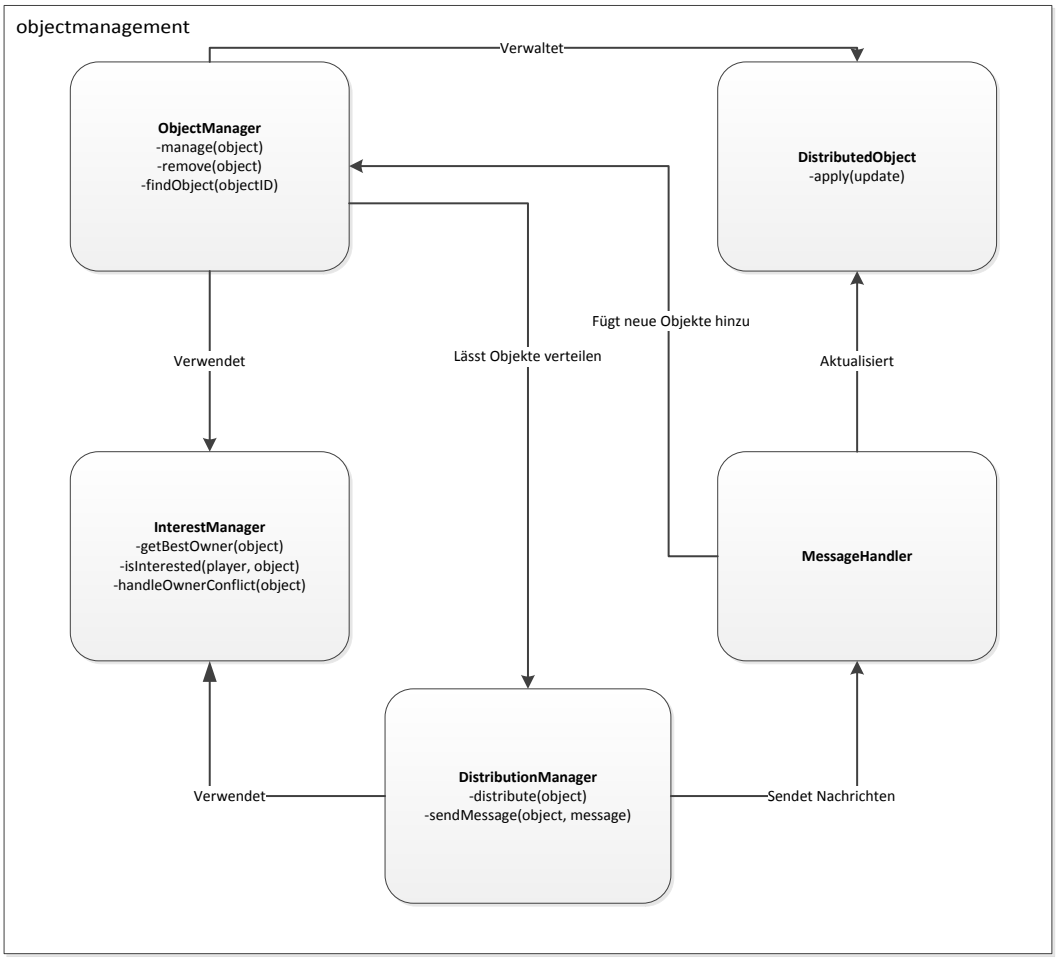


Abbildung 4.6: Schematische Darstellung der Komponenten des Objektmanagements. Vergrößerter Ausschnitt aus Abbildung 4.1. Zeigt eine vereinfachte Übersicht der inneren Zusammenhänge der einzelnen Komponenten des Objektmanagements.

5 Evaluation

Dieser Abschnitt beschäftigt sich damit, ob und wie weit die in den vorherigen Abschnitten erwähnten und in Abschnitt 5.1 rekapitulierten Ziele erreicht wurden. Dazu werden Metriken festgelegt, die es erlauben das Erreichen der Ziele experimentell zu untersuchen. Abschnitt 5.2 führt Kohärenz ein, den zentralen Wert um zu beschreiben, dass die Spielwelt für alle Spieler gleich aussieht. Abschnitt 5.3 beschäftigt sich damit, welche Ressourcen benötigt und wie diese gemessen werden.

Abschnitt 5.4 und Abschnitt 5.5 stellen Prognosen auf wie sich das System unter verschiedenen Lasten verhalten wird und erklären, welche Designentscheidungen und äußere Ereignisse Einfluss auf die Qualität haben.

Danach wird in Abschnitt 5.6 der Aufbau des zum Evaluieren verwendeten Simulators beschrieben. Abschnitt 5.7 beschreibt schließlich einige Tests und präsentiert die daraus gewonnenen Ergebnisse.

5.1 Qualitätskriterien

Das Ziel des hier vorgestellten Objektmanagements ist es, möglichst vielen Spielern gleichzeitig zu erlauben mit einer für alle einheitlichen Welt zu interagieren, dies wird als *Kohärenz* bezeichnet. Dazu muss der lokale Zustand jedes Spielers synchronisiert werden, um Abweichungen zu vermeiden. Es ist jedoch nicht nötig sie vollständig zu verhindern, solange die Spieler diese Abweichungen nicht deutlich wahrnehmen können. Dies erlaubt es Optimierungen durchzuführen, die dabei helfen zwei andere Ziele zu erreichen. Zum einen eine niedrige *Latenz*, Änderungen sollen also möglichst schnell bei anderen Spielern ankommen, zum anderen ein niedriger Netzwerkoverhead, dass heißt, je weniger *Bandbreite* durch Buchführung verbraucht wird, desto mehr steht für das eigentliche Spiel zur Verfügung. Dies erlaubt es auch Spielern mit einer schlechten Verbindung am Spiel teilzunehmen, beziehungsweise ermöglicht mehr Spieler und Objekte innerhalb der Spielwelt.

Die nächsten Abschnitte befassen sich damit, wie sich das Erreichen dieser drei Ziele messen lässt.

5.2 Kohärenz

Kohärenz dient als Maß dafür wie sehr sich die Spieler über den Zustand der Spielwelt einig sind.

Ein System wird als vollständig kohärent bezeichnet, wenn jeder Spieler zu einem bestimmten Zeitpunkt den gleichen Zustand sieht. Oder genauer, wenn ein Spieler jedes Objekt innerhalb seines Sichtbereiches kennt und der Zustand eines Objektes für zwei verschiedene Spieler identisch ist, solange sich das Objekt innerhalb beider Sichtbereiche befindet. Abweichungen sind also nur relevant, wenn sich das Objekt innerhalb des Sichtbereiches befindet. Da kleine Abweichungen toleriert werden sollen, ist es notwendig, dass ein Kohärenzmaß die Abweichungen im System stetig beschreibt. Es soll also nicht nur gemessen werden ob eine Abweichung vorliegt, sondern auch wie groß diese ist.

Um ein geeignetes Maß für die Kohärenz zu erhalten, berechnet man den Unterschied des Zustandes eines Objektes für das sich mehrere Spieler interessieren. Bei diskreten Eigenschaften[1] gibt es keine kleinen Abweichungen, entweder sind sich zwei Teilnehmer über den Zustand einig oder nicht einig. Deshalb wird die größte Gruppe bestimmt, die sich über den diskreten Zustand des Objektes einig ist. Typischerweise sollte es zwei Gruppen geben wenn sich der Zustand eines Objektes ändert, diejenigen, die bereits den neuen Zustand kennen und jene, die noch vom alten Zustand ausgehen.

Für kontinuierliche Eigenschaften sind kleine Abweichungen des Zustandes akzeptabel, daher wird die Differenz von je zwei Zuständen betrachtet. Gemessen wird die Differenz durch eine Funktion $\delta : objekt \times objekt \rightarrow float$, die je nach Eigenschaft geeignet gewählt werden kann, konkret wird für die Position eines Objektes die euklidische Distanz verwendet. Für ein einzelnes Objekt b werden alle seine Kopien b_k jedes der n interessierten Spieler betrachtet. Für alle unterschiedlichen Paare von b_k wird einzeln die Differenz $\delta(b_i,b_j)$ mit $0 \leq i < j < n$ berechnet. All diese $\delta(b_i,b_j)$ werden gemittelt, um einen durchschnittlichen Fehler pro Objekt $\Delta_b = \mathrm{AVG}_{0 \leq i < j < n}(\delta(b_i,b_j))$ zu bestimmen. Da jedoch verschiedene Objekte unterschiedlich wichtig für unterschiedliche Spieler sind, müssen diese Fehler noch gewichtet werden. Dazu wird pro Spieler eine Gesamtabweichung Δ_s aller Objekte, für die sich der Spieler s interessiert, berechnet. Um Δ_s zu berechnen, wird jedem Objekt ein Gewicht γ_b in Abhängigkeit des Interesses zugewiesen. Das gewichtete Mittel ergibt dann die Abweichung für einen einzelnen Spieler $\Delta_s = \mathrm{AVG}(\gamma_b \Delta_b)$, Objekte, die sich innerhalb des Interaktionsbereiches des Objektes befinden, sind für den Spieler sehr wichtig und werden deshalb vollständig gewertet. Zwischen dem Sichtbereich und dem Interaktionsbereich wird die Gewichtung linear interpoliert, wie in Abbildung 5.1 dargestellt.

1 Hier werden nur solche Eigenschaften als diskret betrachtet auf denen sich kein vernünftiger Abstand berechnen lässt, wie zum Beispiel das Team das eine Basis besitzt. Andere Werte, wie zum Beispiel die Trefferpunkte eines Spielers, sind zwar diskrete Werte zwischen 0 und 100, werden hier aber als kontinuierlich behandelt.

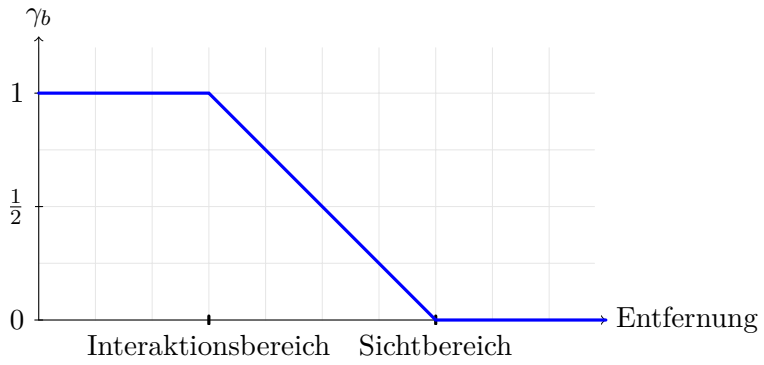


Abbildung 5.1: γ_b ist die Gewichtung eines Objektes b bei der Berechnung des Kohärenzwertes Δ_s eines Spielers s. Außerhalb des Sichtbereiches eines Spielers fließt das Objekt nicht in die Gewichtung mit ein, zwischen dem Sichtbereich und dem Interaktionsbereich wird linear interpoliert und innerhalb des Interaktionsbereiches wird die Abweichung vollständig gewertet.

5.3 Latenz und Bandbreite

Während Kohärenz in Abschnitt 5.2 sich damit beschäftigt, ob das System seine Funktion erfüllt, geht es in diesem Abschnitt darum, welche Ressourcen dafür benötigt werden.

Da Updates direkt vom Besitzer an die ihm bekannten Spieler gesendet werden, muss jeder Besitzer über genug Bandbreite verfügen, um diese Updates senden zu können. Andernfalls kommt es zu Verzögerungen, welche die Kohärenz verschlechtern. Die Latenz wird also nicht eigenständig betrachtet, sondern nur ihr Einfluss auf die Qualität der Simulation. Dies wird zusätzlich dadurch gerechtfertigt, dass eine hohe Latenz oft von äußeren Faktoren abhängt und absolute Werte dadurch viel Aussagekraft verlieren.

Die Bandbreite, die vom Objektmanagement benötigt wird, ist durch die Anzahl und Größe der versendeten Nachrichten bestimmt. Die Größe einer Nachricht setzt sich aus drei Teilen zusammen, dem Overhead des zugrundeliegenden *Netzwerkprotokolls*, dem Overhead für *Verwaltungsinformation* des Objektmanagements und der *Nutzlast*. Das verwendete Netzwerkprotokoll ist *CUSP*[15] und dadurch benötigte Bandbreite wird hier nicht betrachtet. Die Anzahl und Größe der Nachrichten wird gemessen und daraus ein Bandbreitenverbrauch bestimmt. Dieser Wert ist wichtig, um zu bestimmen, welche Anforderungen das System an die Verbindung eines einzelnen Spielers hat.

Da der Anspruch an die Bandbreite je nach Spiel und Situation variiert, werden im Nachfolgenden verschiedene Anordnungen miteinander verglichen. Der nächste Abschnitt befasst sich damit, welche Anordnungen gewählt werden, um die Skalierung des Systems mit verschiedenen Parametern zu untersuchen.

5.4 Sichtweitenverhältnisse

Grundlegend wird angenommen, dass bei vielen Größen nur ihr Verhältnis zur Sichtweite eines Spielers relevant ist. Vergrößert man die Welt bei gleicher Anzahl an Objekten und Spielern, befinden sich durchschnittlich[1] weniger Objekte und Spieler innerhalb der Sichtweite. Diese Annahme legt nahe, die Größe der Welt und die Geschwindigkeit mit der sich Objekte bewegen in Abhängigkeit von den Sichtradien anzugeben. Das erwartete Verhalten sollte sich bei Tests zeigen, in denen verschiedene Größen verändert werden ohne ihre Verhältnisse zu verschieben, zum Beispiel mehr Spieler und Objekte in einer größeren Welt.

Dementsprechend wird angenommen, dass bei variierender Objektdichte, also einem veränderten Verhältnis zwischen Sichtweite, Objektanzahl und Weltgröße, eine Veränderung der Messergebnisse deutlich wird. Genauer gesagt, sollte eine Erhöhung der Dichte für ein erhöhtes Datenaufkommen sorgen, solange bis die Bandbreite eines einzelnen Spielers überfordert ist und es zu Einbrüchen in der Konsistenz kommt.

5.5 Kohärenzbeeinflussung

Ist die Spielermenge stabil, also verlassen keine Knoten das Netzwerk oder kommen neu hinzu, sollte sich die Kohärenz nur ändern, wenn sich der Zustand eines Objektes ändert. Also auch dann, wenn der Besitzer eines Objektes wechselt. Interessanter ist der Fall bei dem über einen längeren Zeitraum einzelne Knoten ausfallen und neue beitreten. Neue Spieler befinden sich in einem abweichenden Zustand, bis sie Verbindung zu ihren Nachbarn aufgenommen haben, in diesem Zeitraum ist eine niedrigere Kohärenz zu erwarten. Besonders problematisch sind wegfallende Knoten, die Objekte besitzen. Ein willentlicher Verbindungsabbruch wird nicht simuliert, ein Besitzer versucht also nie den Besitz eines Objektes abzugeben, bevor er das Netzwerk verlässt. Solange bis ein potentieller neuer Besitzer das Fehlen des alten Besitzers bemerkt, wird der Zustand des Objektes nicht weiter aktualisiert. Bis alle Spieler den neuen Besitzer kennen und akzeptiert haben, kann es vorkommen, dass das Objekt von verschiedenen Spielern aktualisiert wird, dadurch also mehr Unterschiede im Zustand entstehen und die Kohärenz sinkt. Als Test eignet sich hierfür ein gleichmäßiges Wegfallen und Hinzukommen von Knoten, da dies die Gesamtanzahl an Spielern zu jedem Zeitpunkt nicht verändert. Ein zweiter interessanter Test ist das Ausfallen von einem großen Teil der Spielermenge zum gleichen Zeitpunkt. Dadurch könnte es vorkommen, dass alle Spieler wegfallen, die ein bestimmtes Objekt kennen. Dies sollte aber keinen Einfluss auf die Kohärenz haben, da damit auch alle Spieler weggefallen sind, die sich für dieses Objekt interessieren, der Zustand also nicht mehr relevant ist.

1 Falls man von einer Gleichverteilung der Objekte und Spieler ausgeht.

5.6 Simulation

Um das hier vorgestellte Objektmanagement zu testen, werden Tests in einem simulierten Netzwerk durchgeführt. Dazu werden *Planet PI4* und der darin enthaltene Simulator[16] verwendet. *Planet PI4* ist ein Mehrspielerspiel, in dem die Spieler Raumschiffe kontrollieren, die sich im Weltall bewegen und sich gegenseitig beschießen. Die Spieler sind dabei in Teams eingeteilt und versuchen, sich bewegende Basen zu erobern. Das Objektmanagement ist dafür zuständig für jede Basis zu verwalten, wo sich diese befindet und wo sie sich hinbewegt, sowie welches Team sie erobert hat und welche Teams dabei sind sie zu erobern. Die Spieler werden dabei durch Bots simuliert, während die Bewegung der Basen zufällig durch die Spiellogik entschieden wird. Die Spielwelt ist dabei in mehrere Regionen aufgeteilt, in der sich die Spieler frei bewegen können. In jeder dieser Regionen befindet sich die gleiche Anzahl an Basen, innerhalb einer Region ist die Verteilung der Basen zufällig.

Jeder Spieler hat eine begrenzte Bandbreite, die einer üblichen 16MBit Anbindung entspricht. Verzögerungen bei der Übertragung werden vom Simulator nach zuvor erhobenen Daten modelliert.[16]

5.7 Ergebnisse der Simulation

Dieser Abschnitt wertet die Ergebnisse einiger Simulationen aus und versucht, die experimentellen Daten mit dem erwarteten Verhalten des Systems in Verbindung zu bringen.

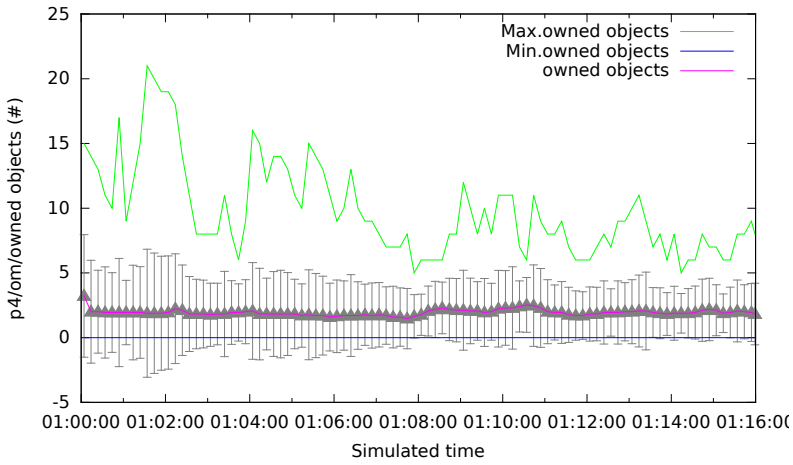
Zu Beginn der Simulation treten die einzelnen Spieler in schneller Abfolge dem Netzwerk bei. Nach etwa sechs Minuten ist dieser Vorgang abgeschlossen und das System kann in einem stabileren Zustand arbeiten. Die Gesamtdauer der meisten Simulationen beträgt deshalb sechzehn Minuten, um nach der initialen Phase das System in einem stabilen Zustand zu zeigen.

Die Angaben der Spieleranzahl beziehen sich immer auf die Gesamtanzahl der simulierten Spieler, die Anzahl der Objekte ist immer ein Vielfaches der Anzahl der Regionen. Alle Simulationen werden in einer Welt mit 9 Regionen Durchgeführt. Spieler können sich ohne weiteres für Objekte aus mehreren Regionen interessieren, so dass ein Spieler auch bei einer Simulation mit 8 Objekten pro Region dennoch über den Zustand von mehr als 8 Objekten informiert werden kann.

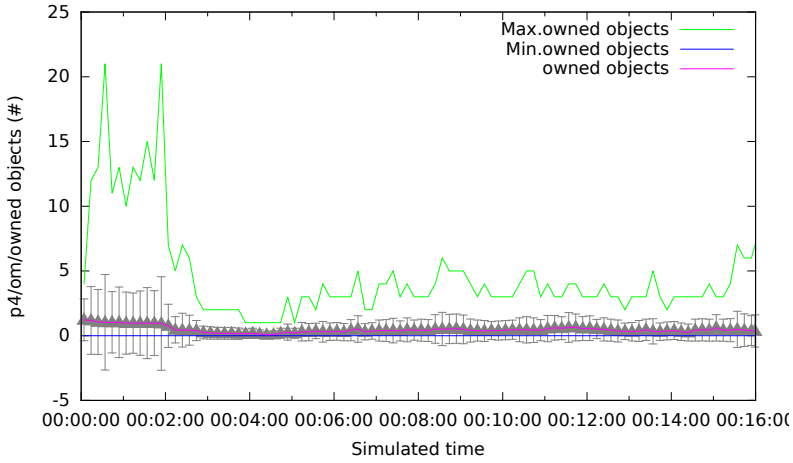
5.7.1 Objekte pro Besitzer

Die erste Messung betrachtet die Anzahl der Objekte, die ein Spieler durchschnittlich besitzt. Abbildung 5.2 zeigt dabei wie diese Anzahl sich verringert, wenn statt 16 Spielern die doppelte Menge vorhanden ist. Da die Anzahl der Objekte konstant bei 8 pro Region (72 insgesamt) bleibt, reduziert sich die durchschnittliche Anzahl der Objekte pro Spieler auf die Hälfte wie man es erwarten würde. Auch die maximale Anzahl an Objekten, die ein Spieler besitzt, reduziert sich, dies ist auf die recht gleichmäßige Verteilung an Spielern und Objekten zurückzuführen. Gleichzeitig ist die Zeit, die benötigt wird, um die Objekte

auf die Spieler zu verteilen, verringert. Die zusätzlichen Ressourcen, die jeder Spieler zur Verfügung stellt, können also für eine Verbesserung insgesamt genutzt werden.



(a) 16 Spieler



(b) 32 Spieler

Abbildung 5.2: Maximale und Durchschnittliche Anzahl der besessenen Objekte pro Spieler. Bei maximal 72 Objekten insgesamt. Besonders bei vielen Spielern dauert es zu Beginn der Simulation eine Weile, bis die Objekte auf die Spieler verteilt sind.

5.7.2 Erhaltene Nachrichten

Eine weitere betrachtete Statistik ist die Anzahl der erhaltenen Nachrichten. Es wird deshalb die Anzahl der Nachrichten und nicht deren Größe betrachtet, da die Größe maßgeblich von der Art des Spiels und des Zustands eines Objektes beeinflusst wird. Die Anzahl der Nachrichten ist weniger abhängig von dem jeweiligen Aufbau des Spiels,

sondern wird durch die Anzahl der Objekte bestimmt. Abbildung 5.3 zeigt zwei Graphen für 16 beziehungsweise 32 Spieler bei jeweils 8 Objekten pro Region, insgesamt also bis zu 72 Objekte. Da sich die Anzahl der Objekte nicht verändert, ist zu erwarten, dass auch die Anzahl der Nachrichten gleich bleibt. Wie die Abbildung aber zeigt, sinkt die Anzahl der Nachrichten bei steigender Spielerzahl leicht ab. Um das Netzwerk bei niedriger Spielerzahl verbunden zu halten, werden vom Interest Management zusätzliche Verbindungen aufgebaut. Dadurch erhalten Spieler mehr Nachrichten als für ihr eigenes Interesse unbedingt nötig wären. Eine höhere Anzahl an Spielern sorgt damit für einen geringeren Overhead bei der Anzahl der unnötig erhaltenen Nachrichten. Während der Zeit in dem das System stabil läuft, also nachdem der initiale Beitritte vorüber ist, liegt die maximale Bandbreite, die ein Spieler benötigt, bei 2000 Bytes pro Sekunde. Eine Ausnahme davon entsteht dann, wenn sich ein ein Besitzerkonflikt ergibt und ein Spieler von allen seinen Nachbarn darauf hingewiesen wird. Dabei kann der Grenzwert auf bis zu 4000 Bytes pro Sekunde ansteigen.

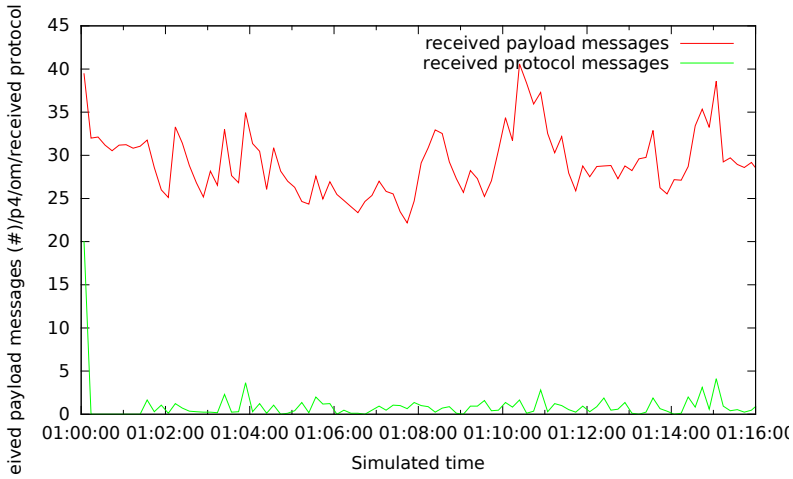
5.7.3 Kohärenz

Kohärenz wie in Abschnitt 5.2 beschrieben, misst wie sehr die Zustände der Spieler voneinander abweichen. Je größer der Kohärenzwert, desto stärker ist der Unterschied. Um die Werte anschaulicher zu gestalten, ist in diesem Unterabschnitt immer die Kohärenz der Position in Vielfachen der Sichtweite angegeben. Eine Kohärenz von 0.1 kann folglich als etwa 10% Abweichung in der Positionen veranschaulicht werden. Abbildung 5.4 zeigt die Kohärenz für zwei Simulationen mit 8 Objekten je Region und 16 beziehungsweise 32 Spielern. Die Kohärenz verschlechtert sich mit einer höhere Anzahl an Spielern. Da in diesem Beispiel keine Ausfälle betrachtet werden, sind alle Abweichungen im Zustand durch das wechselnde Interesse der Spieler verursacht. Je mehr Spieler und Objekte es gibt, desto öfter kommt es dabei auch zu Abweichungen.

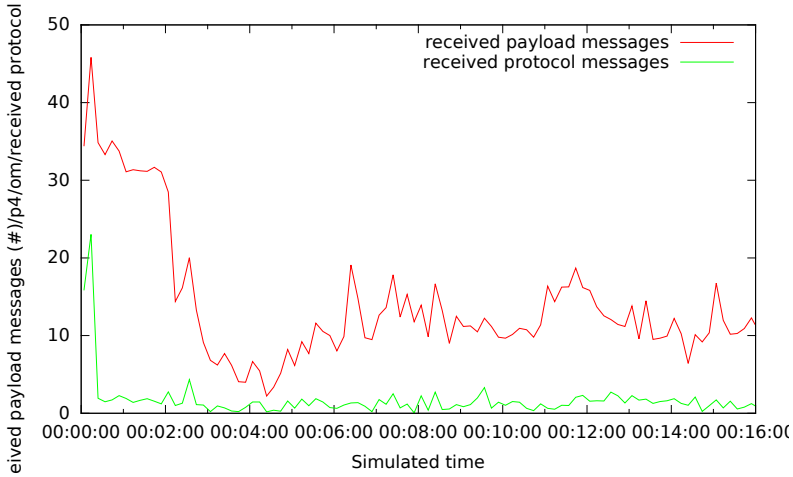
Besonders bei vielen Spielern und Objekten steigt die Kohärenz zu Beginn langsam an. Der Grund findet sich in der Gestaltung des Spiels, alle Objekte starten für alle Spieler im gleichen Zustand und ändern diesen dann langsam. Es dauert also eine Weile bis überhaupt ein großer Zustandsunterschied möglich ist.

5.7.4 Ausfall von Spielern

Dieser Unterabschnitt betrachtet eine längere Simulation von 48 Minuten, in der sich zu Beginn 32 Spieler und 8 Objekte je Region (72 insgesamt) befinden. Nach 30 Minuten fällt die Hälfte der Spieler plötzlich aus, ihnen wird keine Zeit gelassen sich auf den Absturz vorzubereiten. Abbildung 5.5 markiert den Zeitpunkt des Absturzes mit einer Linie. Deutlich zu erkennen ist der Ausfall der Spieler an der gestiegenen Anzahl der Objekte, die ein Spieler im Durchschnitt besitzt. Das Maximum an Objekten, das ein einzelner Spieler besitzt, steigt jedoch nicht an, die Objekte verteilen sich also wieder gleichmäßig auf die verbliebenen Spieler.



(a) 16 Spieler

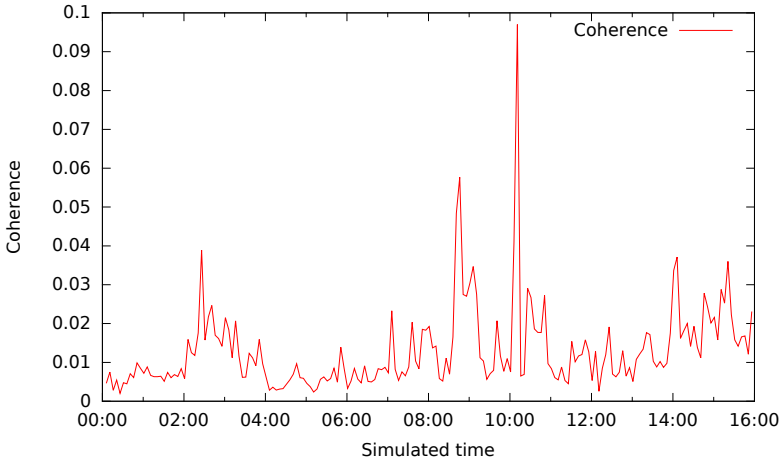


(b) 32 Spieler

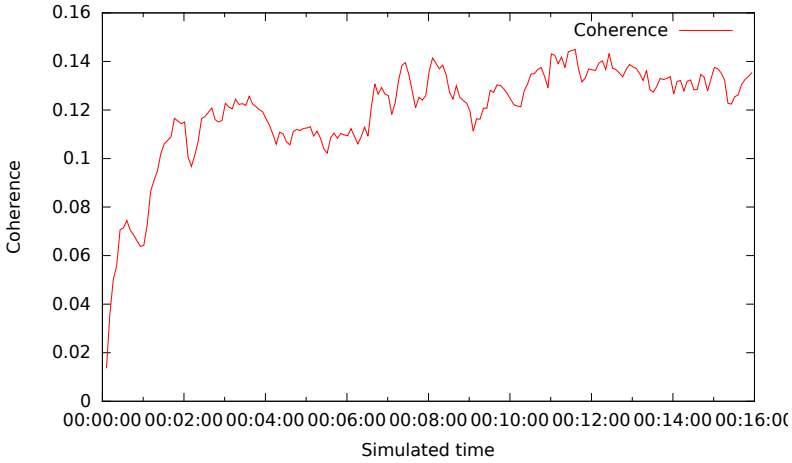
Abbildung 5.3: Durchschnittliche Anzahl der Nachrichten, die pro Spieler erhalten werden. Bei maximal 72 Objekten insgesamt. Payload Nachrichten sind die Zustandsupdates der jeweiligen Objekte, während Protokoll Nachrichten keine für das Spiel relevanten Informationen enthalten.

Bei den Nachrichten sieht man ein leichtes Absinken der Protokollnachrichten, da es nun weniger Spieler gibt, die Konflikte um den Besitzt eines Objektes beheben müssen. Kurz nach dem Ausfall steigt die Anzahl dieser Nachrichten für eine kurze Zeit. Dies sind die Nachrichten, die ausgetauscht werden, um die durch den Ausfall entstehenden Konflikte zu lösen.

Auf die Kohärenz hat der Ausfall selbst keine besonderen Auswirkungen. Fällt ein Spieler aus, der keine Objekte besitzt, so wird sein Zustand für die Kohärenz nicht weiter



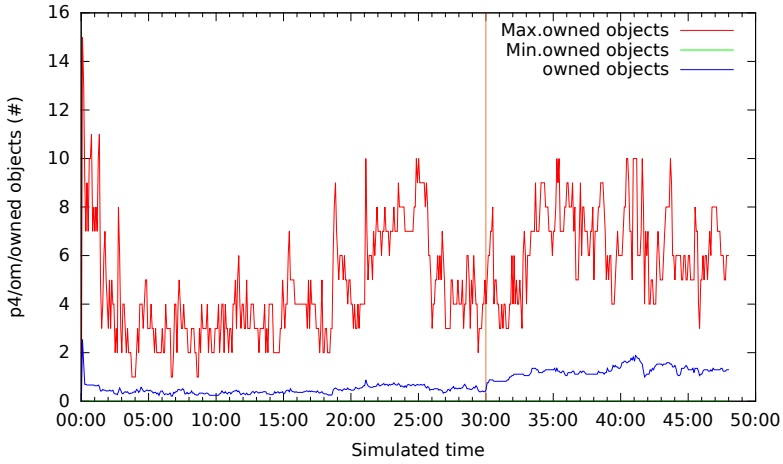
(a) 16 Spieler, 8 Objekte pro Gebiet, 72 Objekte insgesamt



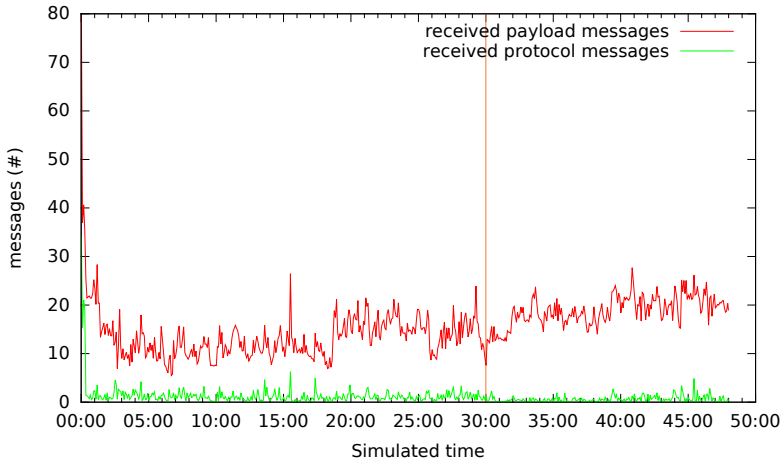
(b) 128 Spieler, 64 Objekte pro Gebiet, 576 Objekte insgesamt

Abbildung 5.4: Kohärenzabweichung der Spieler relativ zum Sichtradius. Größere Werte sind schlechter, ein Wert von 0.05 bedeutet in etwa eine durchschnittliche Abweichung von 5% über den Zustand aller Spieler.

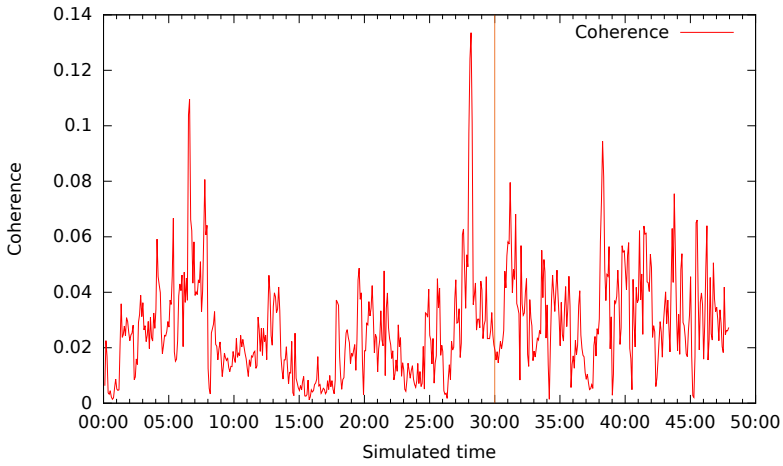
betrachtet. Fällt der Besitzer eines Objektes aus, kann der Zustand dieses Objektes sich erst wieder ändern, sobald es einen neuen Besitzer gibt. Auch wenn es zu einem Besitzerkonflikt kommt, starten beide neuen Besitzer bei dem bisherigen Zustand. Da sich das Objekt kontinuierlich ändert, wird auch hier der Unterschied nicht groß, bevor der Besitzerkonflikt behoben ist.



(a) Anzahl der besessenen Objekte. Nach dem Ausfall verdoppelt sich die Anzahl der Objekte die ein Spieler im Durchschnitt besitzt.



(b) Anzahl der eingehenden Nachrichten.



(c) Kohärenz

Abbildung 5.5: Darstellung von verschiedenen Metriken bei einem Ausfall von 50% der Spieler. Die Linie markiert den Zeitpunkt zu dem die Spieler wegfallen. Den Spielern wird keine Zeit gelassen sich auf das Verlassen des Netzwerkes vorzubereiten.

6 Zusammenfassung

In dieser Arbeit haben wir uns mit dem Objektmanagement in Mehrspielerspielen beschäftigt. Wir sind davon ausgegangen, dass klassische Client/Server Architekturen unvermeidbare Nachteile haben. Wir haben stattdessen ein System vorgeschlagen, das die Ressourcen der Spieler in einem Peer-to-Peer Verfahren nutzt, um mit niedrigem Aufwand viele Spieler und Objekte in der Spielwelt verwalten zu können.

Wir haben für das Objektmanagement ein Interest Management System ausgenutzt, das es Spielern erlaubt mit anderen Spielern in ihrer Nähe zu kommunizieren und Informationen über deren Positionen bereitstellt. Auf dieser Basis haben wir ein Verfahren eingeführt, das jedem Objekt einen Besitzer zuweist, der dafür zuständig ist, das Objekt zu verwalten. Um den Besitzer festzulegen, haben wir eine Interessensfunktion eingeführt und denjenigen Spieler als Besitzer bestimmt, der das größte Interesse an einem Objekt hat. Die Interessensfunktion ist so gewählt, dass Spieler diese aus vorhandenem Wissen für andere Spieler schätzen können. Dadurch ist es möglich ohne zusätzliche Kommunikation einen Besitzer festzulegen, der in vielen Fällen bereits eindeutig ist. Für Fälle in denen der Besitzer nicht eindeutig ist, wurde ein explizites Protokoll vorgestellt, mit dem ein Konflikt gelöst werden kann, indem die Spieler ihr tatsächliches Interesse austauschen. Nachdem ein Besitzer gefunden ist, können Objekte verändert werden, indem Spieler mit dem Besitzer kommunizieren. Der Besitzer ist dann dafür zuständig diese Nachrichten zu verarbeiten, Änderungen auf dem Objekt anzuwenden und an die interessierten Spieler zu verteilen.

Um das System zu evaluieren, wurden verschiedene Kriterien festgelegt, die erfüllt werden sollen. Zum einen ist es wichtig zu wissen wie viel Bandbreite verwendet wird und zum anderen soll sich der Zustand der Objekte sich nicht zu stark voneinander unterscheiden. Um den Unterschied zu messen, haben wir die Kohärenz als globales Maß für die Abweichung des Zustandes eingeführt. Wir haben dann Simulationen mit verschiedenen Parametern durchgeführt, um die Skalierbarkeit des Systems bei verschieden Mengen an Spielern und Objekten zu testen. Die Ergebnisse zeigen, dass das System durchaus mit vielen Objekten und Spielern umgehen kann, solange kleine Abweichungen im Zustand der Spieler akzeptabel sind. Besonders bei Abstürzen von 50% der Spieler zeigt das System, dass Peer-to-Peer Ansätze gut geeignet sind, um auch bei großen Ausfällen weiterhin funktionsfähig zu bleiben.

6.1 Ausblick

Das entwickelte System versucht möglichst effizient im Umgang mit den Ressourcen zu sein. Aber in einem inhomogenen Netzwerk kann es vorkommen, dass ein einzelner Spieler mit der Größe der zu sendenden Nachrichten überfordert ist. Dies sorgt dafür, dass einige Nachrichten erst mit Verzögerungen versendet werden. Um das zu vermeiden, könnten Nachrichten so priorisiert werden, dass Spieler mit einem höheren Interesse die Nachrichten zuerst erhalten und diese dann an Spieler weiterleiten, die ein niedrigeres Interesse. Für diese Fälle können auch weiter angepasste Interessensfunktionen untersucht werden.

Es gibt verschiedene weitere Eigenschaften, die für eine Interessensfunktion verwendet werden könnten, und andere Interessensfunktionen könnten zusätzliche Aufgaben erfüllen. Zum Beispiel könnten bestimmte vertrauenswürdige Spieler ein höheres Interesse bekommen, um möglichen Betrug zu erschweren und gleichzeitig ein Weiterspielen zu erlauben, wenn keine vertrauenswürdigen Spieler vorhanden sind.

Ein weiteres Problem, das nicht behandelt wurde, sind Objekte, für die sich niemand interessiert. Es wurde davon ausgegangen, dass diese für das Spiel nicht mehr relevant sind und sie deshalb entfernt und eventuell später neu generiert werden können. Aber für dünn besiedelte persistente Welten könnte es wichtig sein, Informationen über solche Objekte weiterhin zu speichern.

Literaturverzeichnis

[1] BITKOM (Hrsg.): *Gaming*. http://www.bitkom.org/de/themen/54697.aspx, Abruf: 31. März. 2013 (Zitiert auf Seite 1)

[2] Benford, Steve ; Fahlén, Lennart: *A Spatial Model of Interaction in Large Virtual Environments*. 1993 (Zitiert auf Seite 9)

[3] Bharambe, Ashwin R. ; Agrawal, Mukesh ; Seshan, Srinivasan: Mercury: Supporting scalable multi-attribute range queries. In: *SIGCOMM*, 2004, S. 353–366 (Zitiert auf Seite 7)

[4] Bharambe, Ashwin R. ; Pang, Jeff ; Seshan, Srinivasan: *A Distributed Architecture for Interactive Multiplayer Games*. 2005 (Zitiert auf Seite 7)

[5] Castro, Miguel ; Jones, Michael B. ; Kermarrec, Anne-Marie ; Rowstron, Antony ; Theimer, Marvin ; Wang, Helen ; Wolman, Alec: An Evaluation of Scalable Application-Level Multicast Built Using Peer-to-Peer Overlays. In: *Infocom'03*, 2003 (Zitiert auf Seite 6)

[6] Druschel, Peter ; Rowstron, Antony: PAST: A large-scale, persistent peer-to-peer storage utility. In: *HotOS VIII*, 2001, S. 75–80 (Zitiert auf Seite 7)

[7] Hampel, Thorsten: A peer-to-peer architecture for massive multiplayer online games. In: *Proceedings of NetGames '06*, 2006, S. 48 (Zitiert auf Seite 6)

[8] Hu, Shun-Yun ; Chen, Jui-Fa ; Chen, Tsu-Han: VON: a scalable peer-to-peer network for virtual environments. In: *Netwrk. Mag. of Global Internetwkg.* 20 (2006), Juli, Nr. 4, 22–31. http://dx.doi.org/10.1109/MNET.2006.1668400. – DOI 10.1109/MNET.2006.1668400. – ISSN 0890–8044 (Zitiert auf Seiten 2 und 11)

[9] Knutsson, Björn ; Lu, Honghui ; Xu, Wei ; Hopkins, Bryan: *Peer-to-Peer Support for Massively Multiplayer Games*. 2004 (Zitiert auf Seite 6)

[10] Lehn, Max ; Leng, Christof ; Rehner, Robert ; Triebel, Tonio ; Buchmann, Alejandro: An Online Gaming Testbed for Peer-to-Peer Architectures. In: *Proceedings of ACM SIGCOMMÍ1*, ACM, August 2011. – Demo (Zitiert auf Seite 15)

[11] Maymounkov, Petar ; Mazières, David: *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*. 2002 (Zitiert auf Seite 5)

[12] Rowstron, Antony ; Druschel, Peter: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: *MIDDLEWARE* (2001), S. 329–350 (Zitiert auf Seiten 5 und 6)

[13] Schmieg, Arne ; Stieler, Michael ; Jeckel, Sebastian ; Kabus, Patric ; Kemme, Bettina ; Buchmann, Alejandro: pSense - Maintaining a Dynamic Localized Peer-to-Peer Structure for Position Based Multicast in Games. In: *Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing*. Washington, DC, USA : IEEE Computer Society, 2008 (P2P '08). – ISBN 978–0–7695–3318–6, 247–256 (Zitiert auf Seiten 2 und 11)

[14] Stoica, Ion ; Morris, Robert ; Liben-Nowell, David ; Karger, David R. ; Kaashoek, M. F. ; Dabek, Frank ; Balakrishnan, Hari: Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. In: *ACM SIGCOMM*, 2001, S. 149–160 (Zitiert auf Seite 5)

[15] Terpstra, Wesley W. ; Leng, Christof ; Lehn, Max ; Buchmann, Alejandro P.: Channel-based Unidirectional Stream Protocol (CUSP). In: *Proceedings of the IEEE INFOCOM Mini Conference*, 2010 (Zitiert auf Seite 27)

[16] Triebel, Tonio ; Lehn, Max ; Rehner, Robert ; Guthier, Benjamin ; Kopf, Stephan ; Effelsberg, Wolfgang: Generation of Synthetic Workloads for Multiplayer Online Gaming Benchmarks. In: *International Workshop on Network and Systems Support for Games (NetGamesÍ2)*, IEEE, November 2012 (Zitiert auf Seite 29)