Consistency in verteilten virtuellen Welten

Daniel Gritzner
dgritzne@rumms.uni-mannheim.de
Matrikelnummer: 1111782

Eingereicht bei:
Lehrstuhl für Praktische Informatik IV, Universität Mannheim
Prof. Dr.-Ing.Wolfgang Effelsberg, Tonio Triebel
A5, 6
68159 Mannheim

Die vorliegende Arbeit stellt ein Beitrag zum Seminar "Algorithmen zur Unterstützung von Immersive Gaming" dar.

Zusammenfassung In der vorliegenden Arbeit wird das Thema Consistency im Kontext von verteilten virtuellen Welten betrachtet. Consistency beschäftigt sich mit der Gleichheit von Zuständen der Instanzen mehrerer Teilnehmer einer solchen virtuellen Welt. Hierfür werden formale Methoden zur Betrachtung von Consistency gegeben und es werden anschließend drei Algorithmen vorgestellt, die Consistency gewährleisten sollen. Local Lag verzögert hierbei die Ausführungen von Handlungen der Teilnehmer. Dead Reckoning übermittelt Zustandsänderungen von Objekten, so dass jeder Teilnehmer den gleichen Zustand jeden Objekts sieht. Timewarp verwaltet Listen von Zuständen und Objekten um mit deren Hilfe bei verspätet eintreffenden Datenpaketen dennoch einen korrekten aktuellen Zustand berechnen zu können. Dies geschieht indem einfach im Schnelldurchlauf von einem alten, korrekten Zustand aus alle Zwischenzustände, die durch das verspätet eintreffende Paket beeinflusst werden, neu berechnet werden und somit am Ende wieder ein korrekter Zustand erreicht wird.

1 Einleitung

Die vorliegende Arbeit betrachtet das Thema Consistency im Kontext von verteilten virtuellen Welten. Dabei handelt es sich um fortlaufende Simulationen, die jeweils eine virtuelle Welt darstellen und an der mehrere Benutzer gleichzeitig teilnehmen. Dabei soll die simulierte virtuelle Welt bei allen Benutzer gleich sein, jedoch ohne dass es einen zentralen Server gibt, der die Verwaltung der Simulation übernimmt und eine Referenz der virtuellen Welt darstellt. Statt dessen läuft die Simulation parallel auf den teilnehmenden Peers und somit ohne eine Referenz dessen, wie der korrekte Zustand der virtuellen Welt ist. Ein Beispiel für eine solche verteilte virtuelle Welt wäre ein Multiplayer-Computerspiel, bei dem die Spieler über ein Peer-to-Peer-Netzwerk kommunizieren ([1]).
Im Verlauf der Arbeit wird dargestellt was Consistency ist, wie man diese formal betrachten kann und es werden drei Algorithmen vorgestellt, die versuchen Consistency in oben beschriebenen Welten zu gewährleisten.

1.1 Was ist Consistency?

Natürlich sollen die Handlungen der Teilnehmer innerhalb der virtuellen Welt auch Einfluss auf diese haben können. Beispielsweise könnte man sich ein Massively Multiplayer Online Role-Playing Game (MMORPG) in einer Science-Fiction-Umgebung vorstellen, in der Spieler gemeinsam als Crew ein Raumschiff steuern und damit das Weltall erforschen, Handel mit verschiedenen Planeten treiben oder ähnliches. Hierbei muss es den Spielern natürlich möglich sein den Kurs ihres Raumschiffs zu ändern und somit Einfluss auf den Verlauf der Simulation zu nehmen. Hierbei kann jedoch das Problem auftreten, dass nicht alle Teilnehmer den gleichen Zustand der virtuellen Welt zu dem Zeitpunkt haben, wenn von einem Teilnehmer eine Handlung ausgeführt wird. Z.B. kann einfach aufgrund von Verzögerungen bei der Übertragung über ein Netzwerk, die von Teilnehmer i ausgeführte Handlung bei den anderen Teilnehmern verspätet ankommt. Somit ist das Resultat der Handlung nicht zwangsweise identisch und es entstehen im Allgemeinen Unterschiede in den Zuständen der virtuellen Welt bei den einzelnen Teilnehmern.
Wieder in unserem Beispiel könnte man sich denken, dass die Spieler mit ihrem Raumschiff durch ein Asteroidenfeld navigieren. Wenn der Pilot eine Kursänderung vornimmt und diese Änderung verzögert bei den anderen Spielern ankommt, kann es passieren, dass das Raumschiff bei den anderen Spielern mit einem Asteroid kollidiert, weil die Kursänderung zu spät durchgeführt wurde, während der Pilot in seiner Instanz des Spiels allen Asteroiden ausgewichen ist. Die Kollision mit dem Asteroid könnte fatale Folgen für das Raumschiff haben, welche allerdings nicht bei allen Spielern registriert werden, da die Kollision nicht bei allen Spielern stattfindet. Somit unterscheiden sich die Instanzen, die die einzelnen Spieler sehen, voneinander unter Umständen erheblich.
Die Vermeidung dieses Problems ist Inhalt des Themas Consistency. Der Begriff lässt sich mit Konsistenz, also Gleichheit bzw. Übereinstimmung übersetzen.

Ziel bei der Betrachtung von Consistency ist daher zu erreichen, dass alle Teilnehmer einer oben beschriebenen verteilten virtuellen Welt zu bestenfalls jedem Zeitpunkt den gleichen Zustand der Simulation der Welt haben. Dabei ist mit Zeitpunkt nicht ein Zeitpunkt in der realen Welt gemeint, sondern ein Zeitpunkt innerhalb der simulierten virtuellen Welt.

1.2 Warum ist Consistency in virtuellen Welten wichtig?

Wie bereits im vorherigen Abschnitt beschrieben können Handlungen eines Teilnehmers unterschiedliche Auswirkungen haben, je nachdem in welchem Zustand sich die Simulation gerade befindet. Dies kann natürlich auch zur Folge haben, dass die Entscheidungsfindung der Teilnehmer bezüglich ihrer Handlungen beeinflusst wird. Wieder im selben Beispiel würden die Spieler, bei denen die Kollision mit dem Asteroid stattfand, eventuell versuchen in Rettungskapseln zu fliehen. Währen dessen würden Spieler, bei denen allen Asteroiden ausgewichen wurde, normal ihrem Plan folgen (z.B. eine Raumstationen zwecks Auffüllen von Treibstoff anfliegen) und weiterfliegen. Für diese Spieler wäre es unverständlich warum ein Teil der Crew plötzlich in Rettungskapseln das Schiff verlässt. Um also zu gewährleisten, dass alle Teilnehmer innerhalb einer *gemeinsamen* Welt agieren und gegebenenfalls interagieren, müssen also beim Design solcher verteilten virtuellen Welten Consistency-Betrachtungen durchgeführt werden.

2 Formale Beschreibung

In diesem Abschnitt wird gezeigt, wie man Consistency, also Gleichheit der Zustände der Simulationen der einzelnen Teilnehmer, formal beschreiben kann. Dazu werden erst einige wichtige bzw. nützliche Konzepte zur Betrachtung von Consistency eingeführt, anschließend wird eine mögliche Definition von Consistency geben. Danach werden noch wichtige Begriffe, die im Zusammenhang mit dem Thema stehen, erklärt. Wenn von einem Teilnehmer gesprochen wird, kann damit der Spieler selbst oder der Rechner, an dem dieser Spieler sitzt, gemeint sein. Die ist aus dem Kontext ersichtlich.

2.1 Wichtige Konzepte

Es gibt zwei zentrale Konzepte, die besonders wichtig sind, wenn man über Consistency reden will. Dabei handelt es sich zum einen um Zustände (states) und Handlungen (operations). Als Notation wird im weiteren Verlauf dieser Arbeit $s_{i,t}$ verwendet um den Zustand der Simulation von Teilnehmer i zum Zeitpunkt t der virtuellen Welt zu beschreiben. Analog dazu wird o_{i,t^0,t^*} verwendet um Handlungen zu beschreiben, die von Teilnehmer i zum Zeitpunkt t^0 erteilt wurde, jedoch erst zum Zeitpunkt t^* ausgeführt werden sollen. Beide Zeitpunkte t^0 und t^* sind dabei Zeitpunkte der virtuellen Welt und nicht der realen Welt. Dabei muss natürlich $t^* \geq t^0$ gelten, ansonsten wäre es eine Handlung, die in

der Vergangenheit des Teilnehmers ausgeführt werden müsste, der die Handlung überhaupt erst erteilt. Dies macht natürlich kein Sinn. Desweiteren wird die Menge aller Handlungen, die im Verlauf einer Simulationen erteilt und ausgeführt werden, als O bezeichnet.
Zwei weniger zentrale, aber dennoch wichtige Konzepte sind die Funktion R_i und der fiktive Teilnehmer P. R_i wird benutzt um zu beschreiben, ob eine Handlung vor einem bestimmten Zeitpunkt t bei Teilnehmer i ankam. Sie ist definiert durch:

$$R_i(t, o_{j,t^0,t^*}) = \begin{cases} \textit{wahr} & o_{j,t^0,t^*} \text{ kam bei } i \text{ vor oder zu Zeitpunkt } t \text{ an} \\ \textit{falsch} & \text{sonst} \end{cases}$$

Dabei sind die Zeitpunkte t^0 und t^* der Handlung irrelevant für das Ergebnis von R_i. Der fiktive Teilnehmer P kann als eine Art Teilnehmer mit einem Rechner angesehen werden, auf dem die Simulationen immer den perfekten Zustand hat. D.h. zu P besteht keine Netzwerkverzögerung und der Rechner von P benötigt keine Zeit für die Berechnung der Simulation. Da also immer alle Handlungen vor t^* empfangen werden und zum korrekten Zeitpunkt berechnet werden, sieht P immer den korrekten Zustand der Simulation. P erteilt keine Handlungen.

2.2 Definition

Man kann Consistency folgendermaßen mathematisch formulieren:

$$\begin{aligned} &\forall\ t, i, j : \\ &\forall\ o_{k,t^0,t^*} \in O \text{ mit } t^* \leq t : \\ &R_i(t, o_{k,t^0,t^*}) \wedge R_j(t, o_{k,t^0,t^*}) \Rightarrow (s_{i,t} = s_{j,t}) \end{aligned}$$

In Worten heißt das, man betrachtet jeden Zeitpunkt t der virtuellen Welt. Für jeden dieser Zeitpunkte betrachtet man alle Paare (i, j) aus Teilnehmern, die alle Handlungen, welche bis einschließlich t ausgeführt werden sollen, erhalten haben. Wenn für jedes dieser Paare (i, j) die Gleichung $s_{i,t} = s_{j,t}$ erfüllt ist, also deren Zustände unter der Vorraussetzung des vollständigen Wissens (heißt alle nötigen Handlungen wurden empfangen) gleich sind, dann erfüllt die betrachtete verteilte virtuelle Welt das Consistency-Kriterium. Somit gewährleistet die Simulation also, dass die Teilnehmer eine gemeinsame Welt sehen, sofern jeder Teilnehmer die hierfür nötigen Informationen über die Handlungen erhalten hat. Diese Vorraussetzung hat zur Folge, dass kurzzeitige Unterschiede in den Zuständen zugelassen werden können, also so genannte Short-term Inconsistencies (kurzzeitig/vorrübergehende Inkonsistenzen) auftreten können. Auf diese wird später noch genauer eingegangen.
Die Abbildung 1 veranschaulicht das Kriterium. Die blauen Linien stellen dabei die Zustände bei den einzelnen Teilnehmern im Verlauf der Zeit dar. Die grünen Pfeile stellen eine Handlung o_{2,t^0,t^1} dar, welche zu unterschiedlichen Zeitpunkten bei den Teilnehmern 1 und 3 ankommt. Damit das Consistency-Kriterium erfüllt ist, müssen zu jedem Zeitpunkt t (entspricht einer gedachten vertikalen

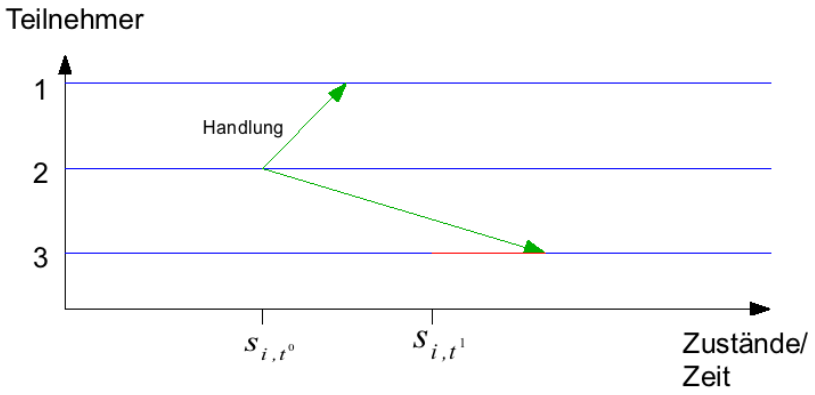

Abbildung 1. Veranschaulichung des Consistency-Kriteriums

Linie) die Zustände der Teilnehmer identisch sein. Eine Ausnahme stellt der rot markierte Bereich bei Teilnehmer 3 dar. Dessen Zustand darf innerhalb dieses Bereichs abweichen, da das Datenpaket mit der Handlung von Teilnehmer 2 bis dahin noch nicht angekommen ist, obwohl es bereits benötigt wird. Sobald auch Teilnehmer 3 das Datenpaket erhalten hat, muss dessen Zustand wieder identisch mit dem der anderen Teilnehmer sein.
Würde man solche Phasen, in denen Abweichungen der Zustände erlaubt sind, nicht zulassen, wäre es quasi unmöglich eine verteilte virtuelle Welt zu realisieren, die das so modifizierte Consistency- Kriterium erfüllt. Aufgrund von physikalischen Bedingungen müssen eben auch Fälle berücksichtigt werden, bei denen Datenpakete zu spät oder gar nicht ankommen.

2.3 Correctness

Das Consistency-Kriterium erfordert nur, dass die Zustände der einzelnen Teilnehmer identisch sind. Es wird jedoch keine Aussage darüber gemacht, ob die Zustände (bzw. im Falle von erfüllter Consistency der Zustand) auch korrekt im Sinne der Regeln der virtuellen Welt sein müssen. Es könnte z.B. in unserem Sci-Fi-MMORPG ein Fall auftreten, dass eine Spielercrew eine Schmugglerbande sucht, welche sich versucht mit Hilfe einer getarnten Selbstschussanlage zu schützen. Dabei könnte es passieren das die Spielercrew durch den Bereich fliegt, der durch die Reichweite der Selbstschussanlage abgedeckt wird. Geht allerdings das Datenpaket, welches die Kursänderung enthält, die die Spieler in Nähe der Anlage bringt, verloren, sieht der Rechner, welcher die Anlage kontrolliert, nicht das diese schießen müsste. Bei erfüllter Consistency wird der Punkt an dem sich das Raumschiff der Spieler befindet wird und die Richtung in die es sich bewegt irgendwann korrigiert. Dies kann jedoch zu einem Zeitpunkt geschehen, an dem das Raumschiff bereits wieder außerhalb der Reichweite der Selbstschussanlage ist. Somit schießt die Anlage nie, obwohl dies hätte passieren müssen bei

korrekter Simulation. An diesem Beispiel wird deutlich, dass man den Begriff Consistency noch weiter verschärfen kann. Man spricht dann von Correctness (Korrektheit). Bei erfüllter Correctness hätte die Selbstschussanlage im Beispiel also schießen müssen.
Ein mathematisches Kriterium für Correctness ist:

$$\begin{aligned}&\forall\ t, i:\\ &\forall\ o_{k,t^0,t^*} \in O \text{ mit } t^* \leq t:\\ &R_i(t, o_{k,t^0,t^*}) \Rightarrow (s_{i,t} = s_{P,t})\end{aligned}$$

Dieses Kriterium macht eine Aussage, die sehr ähnlich zu der Aussage des Consistency-Kriteriums ist. Jedoch wird für jeden Zeitpunkt und jeden Teilnehmer der jeweilige Zustand mit dem Zustand der Welt bei dem fiktiven perfekten Teilnhemer P verglichen. Hat also ein Teilnehmer i zu einem Zeitpunkt t alle nötigen Handlungen empfangen, muss $s_{i,t}$ gleich $s_{P,t}$ sein. Da $s_{P,t}$ per Definition den korrekten Zustand zu einem bestimmten Zeitpunkt darstellt, macht dieses Kriterium die Aussage, dass alle Teilnehmer, unter Vorraussetzung des vollständigen Wissens über die nötigen Handlungen, den korrekten Zustand erreichen müssen. Auch hier dürfen wieder aufgrund der Vorraussetzung kurzzeitig Abweichungen zum korrekten Zustand auftreten. Correctness impliziert Consistency, da die Zustände der Teilnehmer natürlich paarweise gleich sind, wenn jeder Zustand jeweils dem korrekten Zustand entspricht.
Abbildung 2 zeigt das Correctness-Kriterium. Die Abbildung ist analog zu Ab-

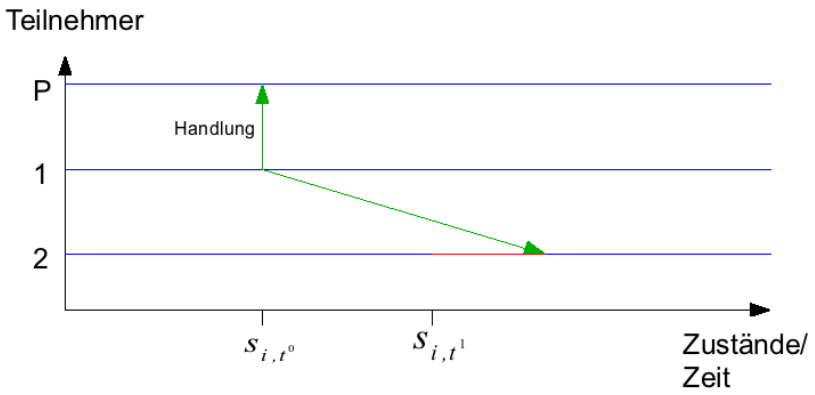

Abbildung 2. Veranschaulichung des Correctness-Kriteriums

bildung 1 zu lesen. Jedoch müssen hier zu jedem Zeitpunkt die Zustände der Teilnehmer 1 und 2 mit dem Zustand von P (zu welchem keine Verzögerung besteht) übereinstimmen, um das Correctness-Kriterium zu erfüllen. Auch hier stellt der rot markierte Bereich bei Teilnehmer 2 eine Ausnahme dar, da Teilnehmer 2 noch nicht alle Informationen hat um den korrekten Zustand zu be-

rechnen. D.h. innerhalb des roten Bereichs darf der Zustand von Teilnehmer 2 vom Zustand von Teilnehmer P abweichen.

2.4 Short-term Inconsistencies

Die beiden Kriterien für Consistency und Correctness erlauben bei jedem Teilnehmer einen abweichenden Zustand, wenn dieser noch nicht alle nötigen Handlungen empfangen hat. Somit kann es also für einen kurzen Zeitraum zu so genannten Short-term Inconsistencies kommen. Bei erfülltem Consistency-Kriterium dauern diese maximal so lange an, bis alle nötigen Informationen empfangen wurden. Wieder am Beispiel des Sci-Fi-MMORPGs erklärt könnte dies im Fall der Navigation durch das Asteroidenfeld (siehe 1.1) bedeuten, dass ein Teil der Spieler kurzzeitig die Kollision mit einem Asteroiden sieht, jedoch sobald die Consistency wieder hergestellt ist das Resultat des erfolgreichen Ausweichmanövers sieht (das Schiff ist nicht mehr beschädigt). Das Auftreten solcher Short-term Inconsistencies lässt sich mit Hilfe folgender Formel bestimmen:

$$I_j(o_{i,t^0,t^*}) = d_{i,j} + T_{i,j} - (t^* - t^0)$$

Hierbei ist $d_{i,j}$ die Verzögerung, die durch die Übertragung von Daten von i zu j über das Netzwerk entsteht. $T_{i,j}$ ist der Zeitunterschied der Simulation von Teilnehmer i und j. Betrachtet man einen festen Zeitpunkt in der realen Welt, kann es bei fehlender Synchronisierung sein, dass die virtuelle Welt von Teilnehmer i gerade an einem anderen Zeitpunkt ist als die virtuelle Welt von Teilnehmer j. Dabei gilt: ist $T_{i,j}$ positiv, dann ist die Simulation von j der Simulation i um $|T_{i,j}|$ voraus ist. Für einen fixierten Zeitpunkt in der realen Welt gilt also: ist die Simulation von i gerade an dem Zeitpunkt t, dann ist die Simulation von j am Zeitpunkt $t + T_{i,j}$. Bei negativem Vorzeichen hängt j um den entsprechenden Betrag hinterher. Somit lässt sich die rechte Seite der Gleichung in zwei Terme gruppieren. Zum einen $d_{i,j} + T_{i,j}$ und zum anderen $t^* - t^0$. Der erste Term drückt dabei aus, wie viel später als t^0 die Handlung bei Teilnehmer j ankommt. Der zweite Term drück aus, wie viel später die entsprechende Handlung ausgeführt werden soll. Somit gilt also offensichtlich, dass genau dann Short-term Inconsistencies auftreten, wenn $I_j(o_{i,t^0,t^*}) > 0$ erfüllt ist. Also genau dann wenn die Informationen über die Handlung den Zielteilnehmer erst dann erreichen, wenn der Zeitpunkt für die Ausführung der Handlung beim Zielteilnehmer bereits vergangen ist. Für Simulationen, die das Consistency-Kriterium nicht erfüllen, ist die Funktion I_j ein Indikator dafür ab wann die Zustände der Teilnehmer sich unterscheiden.

3 Local Lag

Wie eingangs erwähnt stellt die vorliegende Arbeit drei Algorithmen vor, deren Ziel es ist das Consistency-Kriterium zu erfüllen. Local Lag ist der erste dieser Algorithmen. Kern des Algorithmus ist es die Ausführung von Handlungen zu verzögern, um eben Faktoren wie die Netzwerkverzögerung zu kompensieren. In [1] wird dieser Algorithmus eingesetzt.

3.1 Algorithmus

Der Algorithmus basiert darauf, dass die einzelnen Teilnehmer i den jeweils anderen Teilnehmern alle nötigen Informationen über alle o_{i,t^0,t^*} senden, damit diese ihre Simulation der virtuellen Welt jeweils anpassen können. Local Lag setzt hier an. Es wird dabei t^* einfach auf $t^0 + x$ gesetzt, wobei die Antwortzeit x einen Wert echt größer als 0 hat. Ist x ausreichend groß, nimmt die Funktion I_j, deren Term in der Klammer sich zu x vereinfacht, negative Werte an und es treten keine Inconsistencies auf. Unabhängig von den übertragenen Handlungen simuliert jeder Teilnehmer eine eigene Instanz der virtuellen Welt anhand der festgelegten Regeln der virtuellen Welt.

3.2 Eigenschaften

Die wesentliche Eigenschaft von Local Lag ist die Antwortzeit x. Je größer man x wählt, desto größere Netzwerkverzögerungen und Unterschiede der Uhren der virtuellen Welten können kompensiert werden. Allerdings hat die Antwortzeit auch gleichzeitig zur Folge, dass alle Handlungen, die ein Teilnehmer erteilt, für den Teilnehmer selbst verzögert ausgeführt werden. Wählt man also eine zu hohe Antwortzeit, kann sich diese Verzögerung für die Teilnehmer negativ bemerkbar machen. Im Hinblick auf Immersive Gaming stellt dies ein Problem dar, da der Spieler durch die wahrgenommene Verzögerung unter Umständen daran erinnert wird, dass es sich bei dem vorliegenden Spiel nur um ein solches handelt und somit gedanklich wieder in die reale Welt zurück kehrt. Somit gibt es eine Obergrenze für akzeptable Antwortzeiten. Es macht Sinn für das Finden eines guten Werts für die Antwortzeit psychologische Experimente durch zu führen oder Ergebnisse solcher Experimente wie z.B. in [5], [6] und [3] direkt zu verwenden, ohne diese selbst durch zu führen.
Ein weiteres Problem sind Schwankungen in den Verzögerungen, die durch die Netzwerkübertragung auftreten. Z.B. kann sich im Laufe einer Simulation einer virtuellen Welt die Zeit, die Datenpakete von einem Teilnehmer zu einem anderen Teilnehmer benötigen, erhöhen, wenn die Gesamtbelastung des zu Grunde liegenden Netzwerks steigt. Auch beachtet der Algorithmus nicht den Fall, dass Datenpakete verloren gehen können. Somit kann das Consistency-Kriterium doch wieder verletzt werden, da kein Versuch unternommen wird solche Fälle abzufangen oder zu korrigieren.

4 Dead Reckoning

Der zweite Algorithmus, welcher vorgestellt wird, ist Dead Reckoning. Hierbei handelt es sich um einen Algorithmus, bei dem nicht die Handlungen übertragen werden, sondern Zustände. Jedes Objekt der verteilt simulierten virtuellen Welt hat einen Besitzer. Dieser ist dafür verantwortlich den anderen Teilnehmern im Falle einer Änderung den aktualisierten Zustand der betroffenen Objekte zu schicken. Dieser Algorithmus wird z.B. in [4] behandelt.

4.1 Algorithmus

Wie bereits erwähnt setzt dieser Algorithmus voraus, dass für jedes Objekt in der virtuellen Welt ein Besitzer definiert wird. Nur der Besitzer eines Objektes darf Handlungen erteilen, die den Zustand dieses Objekts verändern. Sobald eine solche Handlung ausgeführt wird, teilt der Besitzer allen anderen Teilnehmer mit wie der aktualisierte Zustand des Objektes aussieht. Solange kein aktualisierter Zustand übertragen wird, berechnet jeder Teilnehmer wie sich das Objekt anhand der Regeln der virtuellen Welt verhalten würde, wenn keine Handlungen ausgeführt werden, die das Objekt beeinflussen.
Um den Verlust von Datenpaketen zu kompensieren, sendet der Besitzer eines Objekts regelmäßig dessen Zustand an die anderen Teilnehmer. Dies geschieht auch der Zustand des Objekts sich nicht ändert.

4.2 Eigenschaften

Dieser Algorithmus erfüllt das Consistency-Kriterium. Zwar werden keine richtigen Handlungen übertragen, aber man kann die übertragenen Zustände der einzelnen Objekte jeweils als eine Handlung auffassen. Sobald eine echte Handlung ausgeführt wird, die eine Änderung am Zustand eines Objektes und somit am Zustand der virtuellen Welt bewirkt, wird der geänderte Zustand an die anderen Teilnehmer geschickt. Mit dieser Information kann jeder Teilnehmer seine Simulation entsprechend anpassen, so dass alle Teilnehmer wieder den gleichen Zustand der virtuellen Welt in ihrer Instanz haben. Da, nachdem ein solches Update durchgeführt wurde, alle Teilnehmer wieder nach den selben Regeln weiter simulieren, wie sich sämtliche Objekte der virtuellen Welt verhalten, bleiben die Zustände der Teilnehmer auch weiterhin gleich.
Jedoch kann Dead Reckoning keine Correctness gewährleisten. Bei Dead Reckoning könnte durch verlorene Datenpakete genau so ein Fall eintreten wie im Beispiel zu Correctness (2.3) beschrieben wurde.
Interessant ist noch, dass der Rechenaufwand für einen Teilnehmer für Objekte, die er nicht kontrolliert, vergleichsweise gering ist. Zum Beispiel müssen keine Kollisionen überprüft werden, da der Besitzer in solchen Fall, da sich eine Zustandsänderung ergibt, sowieso ein Update des Zustands verschickt.

5 Timewarp

Beim letzten Algorithmus in dieser Arbeit handelt es sich um Timewarp ([2]). Dieser Algorithmus verwendet zwei Listen. Eine Liste für Zustände und eine Liste für Handlungen. Anhand dieser Listen kann bei verspätet eintreffenden Handlungen der korrekte Zustand immer noch berechnet werden.

5.1 Algorithmus

Es wird angenommen, dass jede Instanz der verteilten virtuellen Welt anfangs mit einem, zu dem Zeitpunkt in der virtuellen Welt bei dem die Simulation

beginnt, korrekten Zustand $s_{i,t}$ initialisiert wird. Wie bereits erwähnt verwendet der Algorithmus zwei Listen $L_{i,s}$ und $L_{i,o}$. Jeder Teilnehmer i hat seine eigene Listen. Dabei enthält $L_{i,s}$ alle Zustände seit Beginn der Simulation. Anfangs enthält diese Liste nur den oben genannten Zustand $s_{i,t}$. $L_{i,s}$ soll nach den Zeitpunkten t der Zustände $s_{i,t} \in L_{i,s}$ sortiert sein. $L_{i,o}$ enthält alle erteilten Handlungen seit Beginn der Simulation und ist somit anfangs leer. Diese Liste soll nach den Zeitpunkten t^* der Handlungen $o_{k,t^0,t^*} \in L_{i,o}$ sortiert sein.
Timewarp führt nach der Initialisierung bis zum Ende der Simulation immer wieder folgende Schritte aus:

1. Warte eine konstante Zeit T_{TW}. Während dieser Zeit füge alle lokal erteilten und alle von anderen Teilnehmern empfangenen Handlungen in $L_{i,o}$ ein. Merke dir dabei das kleinste t^* der eingefügten Handlungen in der Variablen t_x. Nach der Wartezeit merke dir aktuelle Zeit t_C und gehe zum nächsten Schritt.
2. Wähle aus allen $s_{i,t} \in L_{i,s}$ mit $t \leq t_x$ den Zustand mit dem größten t aus. D.h. den letzten korrekten Zustand. Die nachfolgenden Zustände sind nicht mehr korrekt, da Handlungen empfangen wurden, die vor dem Zeitpunkt dieser Zustände durchgeführt werden sollen.
3. Beginne mit $s_{i,t'}$ gleich dem letzten korrekten Zustand, welcher im vorherigen Schritt ermittelt wurde und führe aus:
   (a) Nimm von allen Handlungen $o_{k,t^0,t^*} \in L_{i,o}$ mit $t^* \geq t'$ die Handlung, die das kleinste t^* hat.
   (b) Berechne s_{i,t^*} anhand von $s_{i,t'}$ und den Regeln der virtuellen Welt. Ersetze dabei alle alle $s_{i,t} \in L_{i,s}$ mit $t' \leq t < t^*$ durch jeweils aktualisierte Versionen.
   (c) Führe die in 3. (a) ausgewählte Handlung o_{k,t^0,t^*} auf s_{i,t^*} aus. Der so entstehende neue Zustand wird zu $s_{i,t'}$.
   (d) Führe die nächste Iteration (ab 3. (a)) durch.
   In dieser Schleife wurden jetzt alle Zustände $s_{i,t} \in L_{i,s}$ mit $t \geq t_x$ im Schnelldurchlauf durch korrekte Versionen ersetzt. Somit enthält $L_{i,s}$ jetzt wieder nur noch korrekte Zustände.
4. Berechne den Zustand s_{i,t_C} anhand der Regeln der virtuellen Welt aus dem Zustand $s_{i,t'}$, welcher das Resultat der letzten Iteration der Schleife in Schritt 3 war.
5. Füge den Zustand s_{i,t_C} am Ende der Liste $L_{i,s}$ hinzu.

Nachdem der Algorithmus einmal durchgelaufen ist, enthält $L_{i,s}$ einen zusätzlichen Zustand und alle Zustände sind mit Rücksicht auf das vorhandene Wissen über die Handlungen korrekt. Mit Hilfe der konstanten Wartezeit T_{TW} lässt sich festlegen wie oft pro Zeiteinheit ein neuer Zustand der virtuellen Welt berechnet wird. Z.B. entspricht $T_{TW} = 50$ms einer Rate 20 zusätzlichen Zuständen in $L_{i,s}$ pro Sekunde.

5.2 Eigenschaften

Wie bereits im Algorithmus vermerkt wurde, wird die Liste mit Zuständen in jedem Durchlauf korrigiert, so dass nur korrekte Zustände darin enthalten sind.

Auch wenn Handlungen zu spät empfangen werden, werden alle betroffenen Zustände neu berechnet. Somit ist unter der Vorraussetzung das der Empfang der Datenpakete sichergestellt wird, das Correctness- Kriterium erfüllt. Sobald alle Handlungen bekannt sind, gewährleistet Timewarp das alle Zustände $s_{i,t} \in L_{i,s}$ die Gleichung $s_{i,t} = s_{P,t}$ erfüllen. Wie in 2.3 erwähnt folgt daraus, dass Timewarp auch das Consistency-Kriterium erfüllt. Der Nachteil dieses Algorithmuses gegenüber den anderen sind jedoch die vergleichsweise hohen Kosten. Es müssen alte Zustände im Schnelldurchlauf nochmals bearbeitet werden, was zusätzlich Zeit kostet. Außerdem muss Speicherplatz für die Listen $L_{i,s}$ und $L_{i,o}$ zur Verfügung stehen und die Listen müssen korrekt verwaltet werden, was auch wieder Zeit kostet. Im Allgemeinen kostet die Berechnung eines neuen Zustands am meisten Zeit, da hierfür sämtliche Objekte, die durch die Teilnehmer in die virtuelle Welt eingeführt wurden, miteinander verglichen werden müssen. Dies ist z.B. für Kollisionsabfragen nötig. Also entstehen hier Kosten von $O(n^2)$ mit n Anzahl der Teilnehmer. Da die wesentlichen Schritte des Algorithmuses die Listenverwaltung und die Berechnung von Zuständen sind und für Listenverwaltung Algorithmen mit besserer Laufzeit als $O(n^2)$ bekannt sind, ist klar das, wie bereits erwähnt, die Zustandsberechnung am meisten Kosten verursacht.
Unabhängig von diesen Beobachtung ist noch erwähnenswert, dass Timewarp durch seine Funktionsweise eine gewisse Form von dynamischen Local Lag einführt. Dadurch, dass die Berechnung von neuen Zuständen in festen Zeitintervallen stattfindet, werden erteilte Handlungen mindestens bis zum Ablauf des nächsten Intervalls verzögert.
Man kann Timewarp auch bewußt mit Local Lag kombinieren, um so seltener ein Fall zu haben, indem Zustände korrigiert werden müssen. Ohne Local Lag verursachen z.B. Handlungen, die kurz vor Ablauf eines Timewarp- Zeitintervalls erteilt werden, bei den anderen Teilnehmern Inconsistencies, da sie die Handlungen durch die Netzwerkverzögerung erst nach Ablauf des Intervalls erhalten. Somit findet nach Ablauf das nächsten Zeitintervalls eine Korrektur statt. Local Lag kompensiert eben genau diese Netzwerkverzögerung. Die Wahrscheinlichkeit, dass erteilte Handlungen innerhalb des Zeitintervalls bei allen Teilnehmern ankommen, indem sie ausgeführt werden sollen, erhöht sich durch Local Lag. Somit wird die Anzahl der nötigen Korrekturen minimiert.

6 Zusammenfassung

Die vorliegende Arbeit hat gezeigt was Consistency ist und wie man diese formal beschreiben kann. Alle Betrachtungen wurden dabei im Kontext von verteilten virtuellen Welten gemacht. Für diese wurden auch die drei Algorithmen Local Lag, Dead Reckoning und Timewarp vorgestellt, welche Consistency oder gar Correctness im betrachteten Kontext gewährleisten können. Dabei wurden die verschiedenen Eigenschaften dieser Algorithmen aufgezeigt. Vergleicht man diese Algorithmen wird klar, dass Local Lag keine Consistency gewährleisten kann, sich jedoch dazu einsetzen lässt um die Performance von Timewarp zu verbessern. Timewarp gewährleistet nicht nur Consistency, sondern sogar Correctness.

Allerdings hat dieser Algorithmus vergleichsweise hohe Kosten. Bei Dead Reckoning wurde erkannt, dass Teile der Simulationen nicht von allen Teilnehmern ausgeführt werden mussten. Dies hat zur Folge das eine höhere Anzahl von Teilnehmern beim Einsatz von Dead Reckoning ein geringeres Problem darstellt als es dies beim Einsatz von Timewarp tut. Somit ist insgesamt ersichtlich, dass es keinen besten Algorithmus gibt und je nach Anwendungsgebiet Timewarp zusammen mit Local Lag oder eben Dead Reckoning besser ist. Als Faustregel lässt sich formulieren, dass sich bei sehr hohen Teilnehmerzahlen Dead Reckoning besser eignet, während Timewarp für Anwendungen, die Correctness erfordern, besser geeignet ist.
Ein Problem aller Algorithmen ist jedoch, dass nicht versucht wird durchgeführte Korrekturen zu verstecken. Wenn man sich wieder die Navigation durch das Asteroidenfeld im Sci-Fi-MMORPG vorstellt, würde keiner der Algorithmen versuchen die fälschlicherweise angezeigte Kollision mit dem Asteroiden zu verstecken. Das Raumschiff wäre einfach plötzlich wieder vollkommen unbeschädigt bei den Teilnehmern, die vorher die Kollision sahen. In Anbetracht von Immersive Gaming stellt dies ein Problem dar, da solche plötzlichen Änderungen am Zustand der virtuellen Welt als störend empfunden werden können, wenn diese zu groß sind. Eine Möglichkeit wäre es bestimmte schwer wiegende Ereignisse wie die Zerstörung eines Raumschiffs verzögert zu rendern, jedoch hat man hier wieder zwei neue Probleme. Eine zu hohe Verzögerung macht sich unter Umständen negativ bemerkbar (siehe Antwortzeit beim Local Lag Algorithmus). Auch erfordert es zusätzlichen Aufwand zu bestimmten bei welchen Ereignissen Verzögerungen eingesetzt werden müssen und bei welchen nicht.
Abschließend kann man also sagen, dass es bereits Algorithmen gibt, die Consistency bzw. die verschärfte Variante Correctness gewährleisten, es aber keinen optimalen Algorithmus gibt, der auf alle Anwendungsgebiete passt. Außerdem ist keiner der Algorithmen frei von Problemen was im unmittelbar vorangehenden Absatz gezeigt wurde.

Literatur

1. C. Diot et al. A distributed architecture for multiplayer interactive applications on the internet. *IEEE Network Magazine*, vol. 13, no. 4, July/August, 1999.
2. M. Mauve et al. Local-lag and timewarp: providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia*, vol. 6, no. 1, February:47–57, 2004.
3. S. Card et al. *The psychology of human-computer interaction.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.
4. S. Singhal et al. *Networked virtual environments design and implementation.* ACM press, New York, 1999.
5. S. Teal et al. A performance model of system delay and user strategy selection. In *Proc. of Human factors in computing systems 1992*, pages 295–305, 1992.
6. B. Schneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys*, vol. 16, no. 3, 1994.