Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Alexander Schnoor

Konzeption eines Protokolles zur Visualisierung von Simulationen in verteilten Multiagentensystemen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexander Schnoor

Konzeption eines Protokolles zur Visualisierung von Simulationen in verteilten Multiagentensystemen

Bachelorthesis eingereicht im Rahmen der Bachelorarbeit

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Sarstedt
Zweitgutachter: Prof. Dr. Jenke

Eingereicht am: 07. September 2015

Alexander Schnoor

Thema der Arbeit
Konzeption eines Protokolles zur Visualisierung von Simulationen in verteilten Multiagentensystemen

Stichworte
Multiagentensystem, Verteilte Systeme, Systemmodellierung, Visualisierung, Protokoll

Kurzzusammenfassung
In dieser Bachelorarbeit wird ein Konzept für ein Protokoll zur Darstellung von Simulationsergebnissen aus einem verteilten Multiagentensystem in einer 3D Visualisierung sowie die dazugehörigen Herausforderungen vorgestellt. Als konkretes Beispiel wird das Agentenbasierte Modell des Krüger-Nationalpark in Südafrika von Ulfia Lenfers herangezogen.
Alexander Schnoor

Title of the paper
Conception of a protocol for visualization of data from distributed simulation systems

Keywords
Multi-Agent-System, Distributed Systems, Systemmodelling, Visualization, Protocol

Abstract
This thesis will show the concept of a protocol to display data from distributed multi-agent-simulations in a 3D visualization. The thesis will refer the agent-based model of the Kruger National Park in South Africa developed by Ulfia Lenfers.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Motivation

Bei der Simulation einer großen Zahl von Agenten in einem verteilten System entsteht schnell eine große Menge an Daten. Bilder sind in der Regel schneller zugänglich als ein Text, der erst gelesen und verstanden werden muss. Eine Visualisierung kann in diesem Fall dabei helfen, das entworfene Modell schnell zu überprüfen und es für Betrachter zugänglich machen, die weniger mit der Materie vertraut sind. (Keim, 2002)

Um die Ergebnisse und Abläufe eines verteilten Simulationssystems wie MARS darstellen zu können, wird ein Kommunikationsprotokoll benötigt. Ein solches Protokoll zum Informationstransport wird im Rahmen dieser Arbeit entworfen.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist der Entwurf eines Protokolls zur Darstellung von verteilten Agentensimulationen. Neben der Quasi-Live-Ansicht soll es auch die Möglichkeit geben, die Simulation schrittweise zu betrachten oder auf einen früheren Status zurückzugehen. All dies geschieht losgelöst von der eigentlichen Simulation. So wird die Simulation nicht angehalten oder neu ausgeführt, wenn ein Betrachter sich einen Zeitpunkt genauer ansehen möchte. Eine zusätzliche Herausforderung besteht in der möglichst vollständigen Wiederverwendbarkeit des Protokolls. In Abb. 1.1 sind die beteiligten Komponenten und ein Konzept zur Einbindung zu sehen. Eine genaue Erläuterung der Komponenten und wie sie zusammenarbeiten ist im Abschnitt 4.1 zu finden. Für diese Arbeit sind die Komponenten VIS und VIEW von zentraler Bedeutung.

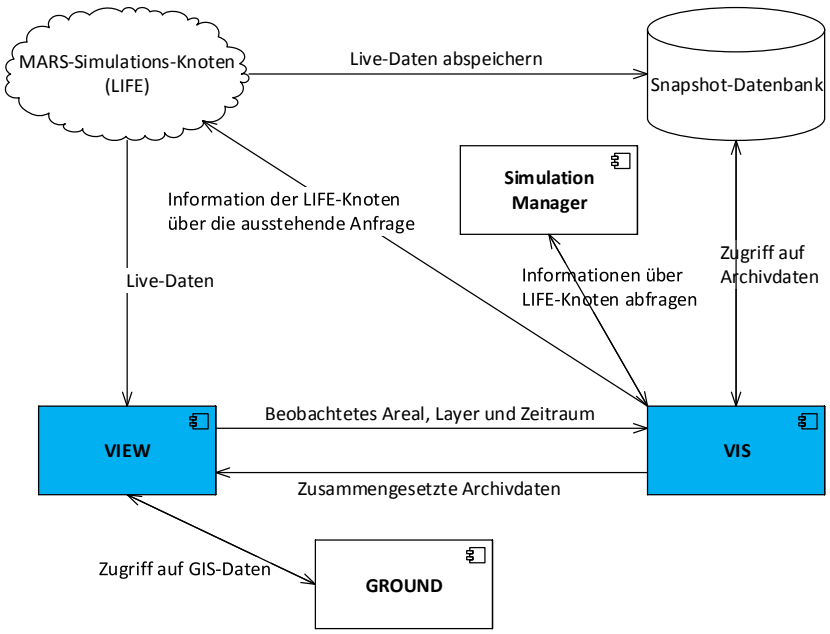


Abbildung 1.1: An der Visualisierung beteiligte Komponenten (Kernkomponenten der Bachelorarbeit blau)

1.3 Aufbau der Arbeit

Die Arbeit beginnt mit einer Einleitung zum Thema Multi-Agenten-Simulation und Visualisierung. In der folgenden Problemanalyse wird unter anderem betrachtet, wie das MARS-System aufgebaut ist und welche Herausforderungen daraus entstehen. Danach werden die konkreten Anforderungen an das System formuliert und im Anschluss Möglichkeiten zu deren Umsetzung diskutiert. Die Arbeit wird sich dabei auf ein Modell des Krüger-National-Parks beziehen, das den derzeitigen Showcase für das MARS-Projekt darstellt. Im abschließenden Kapitel wird der aktuelle Stand diskutiert und ein Ausblick auf mögliche Verbesserungen gegeben.

2 Grundlagen

2.1 Multi-Agenten-Simulationen

2.1.1 Agenten

Nach Russell und Norvig (2004) zeichnet sich ein Agent zunächst dadurch aus, die Umgebung über Sensoren wahrnehmen und mittels Aktuatoren handeln zu können. Sie skizzieren des weiteren vier grundlegende Agentenarten (siehe Abbildung 2.1).

Als einfachster Agent wird der *einfache Reflex-Agent* vorgestellt. Die Entscheidungen dieser Agenten basieren nur auf der aktuellen Wahrnehmung und beziehen den bisherigen Verlauf nicht mit ein. Ein solcher Agent folgt damit einem Regelbuch mit beliebiger Komplexität und ähnelt stark einem endlichen Automaten. Die Einfachheit dieses Agententyps hat zur Folge, dass seine Intelligenz als sehr eingeschränkt betrachtet werden kann.

Die nächste Entwicklungsstufe ist der *modellbasierte Reflex-Agent*. Dieser Agent führt einen Wahrnehmungsverlauf, der ihm erlaubt, auch Annahmen für Zustände machen zu können, die er nicht direkt wahrnehmen kann. Dazu braucht er Kenntnisse über das Modell der Simulation. Zum einen muss der Agent die Regeln kennen, nach denen sich für ihn relevante Aspekte des Modells außerhalb seines Wahrnehmungsradius weiterentwickeln, zum anderen muss er wissen, was seine Aktionen bewirken werden. Basierend auf diesem Wissen wählt er die auszuführende Aktion nach vordefinierten Regeln.

Zielbasierte Agenten beziehen die Auswirkungen ihrer Aktionen in die Wahl ihrer nächsten Aktion mit ein und überprüfen, welche der möglichen Aktionen sie ihrem Ziel näher bringen wird. Dieser Agententyp hat damit die Fähigkeit zu planen und reagiert nicht nur auf den aktuellen Zustand.

Bei *nutzenbasierten Agenten* wird eine Bewertungskomponente hinzugefügt, die es dem Agententyp ermöglicht, seine Planung zu optimieren.

Alle Agententypen können zu *lernenden Agenten* erweitert werden, die ihr Wissen mit der Zeit erweitern.

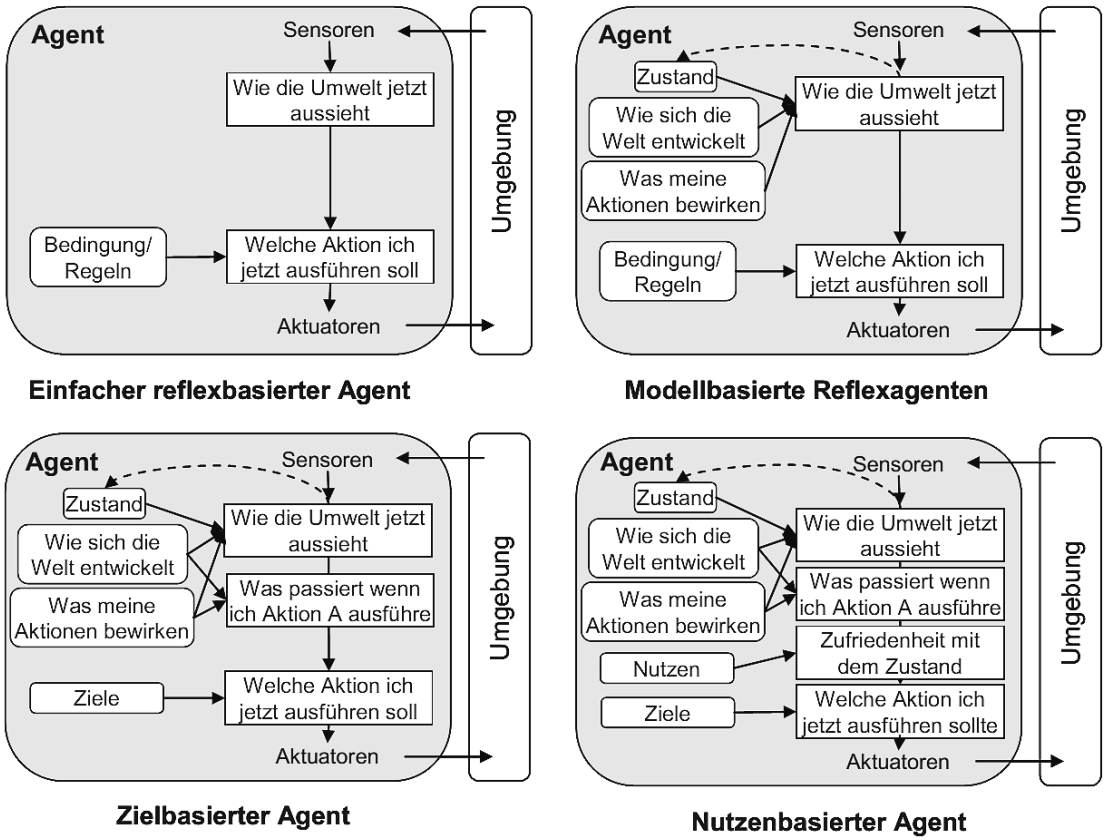


Abbildung 2.1: Verschiedene Entwicklungsstufen von Agenten (Quelle: Feddersen, 2010, S. 156)

2.1.2 Einsatzbereiche

Die Einsatzbereiche für Multiagentensysteme (MAS) sind breit gefächert. Es ist zu beachten, dass MAS nicht zwangsweise mit Simulationen im Zusammenhang stehen, sondern auch in anderen Bereichen zur Problemlösung verwendet werden. In diesem Abschnitt sollen einige Beispiele für die Nutzung von MAS gezeigt werden.

Ökonomie

In der Ökonomie kommen MAS zum Einsatz, um Lieferketten zu organisieren; dies kann mittlerweile fast selbstständig geschehen. Es gibt bereits Prototypen von agenten-basierten Kontrollsystemen, die die Auslastung einer computergesteuerte Fließbandfertigung maximieren. Der Prototyp *Production 2000+*, entwickelt bei *DaimlerChrysler*, konnte den Durchsatz mit dieser Methode von 75% auf 99,7% der theoretischen Maximalleistung erhöhen. Zudem kann ein solches System flexibel auf Bedarfsanforderungen reagieren und teilweise verschiedene Va-

rianten von Werkstücken gleichzeitig fertigen. Aufgrund deutlich höherer Kosten im Vergleich zu konventionellen Fertigungsstraßen hat sich dieser Ansatz jedoch bisher nicht durchgesetzt. (Schild und Bussmann, 2007; Klügl, 2006; Sundermeyer und Bussmann, 2001)

Auch das Marketing bedient sich zunehmend der Möglichkeiten von Multi-Agenten-Simulationen. Als Beispiel kann hier die Einkaufsstätten- oder Markenwahl von Konsumenten genannt werden. (Feddersen, 2010)

Demographie und Landschaftsplanung

Im weitläufigen Bereich der Sozialwissenschaften ist die Multi-Agenten-Simulation mittlerweile die meistverwendete Simulationsart. Zu den zahlreichen Szenarien zählen Simulationen der Land- und Wassernutzung oder demographischer Entwicklungen. (Klügl, 2006)

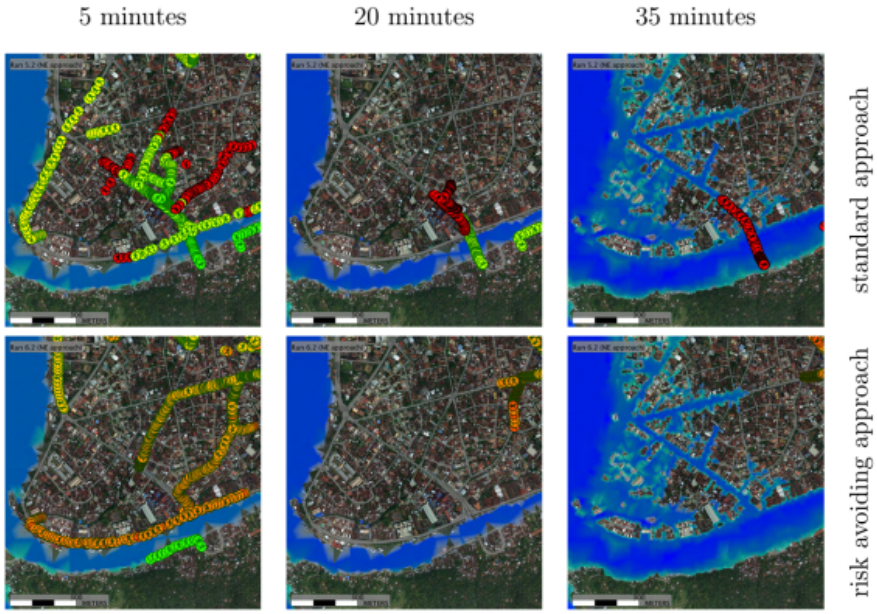


Abbildung 2.2: Simulation einer Stadtevakuierung mit und ohne Risikovermeidung in MATSim[1](Quelle: Lämmel (2011))

Stadtplanung und Katastrophenmanagement

Im Bereich der Stadtplanung lässt sich MAS nutzen, um die Städte in verschiedenster Hinsicht zu planen oder zu optimieren. Huttner u. a. (2009) beschreiben zum Beispiel das *KLIMES-*

[1]http://www.matsim.org/

Projekt. Dessen Ziel ist es, das Wohlbefinden der Bevölkerung während Hitzeperioden, auch unter Berücksichtigung des Klimawandels, zu gewährleisten.

Hernández, Ossowski und Garcia-Serrano (2002) stellen MAS vor, die die Organisation der Verkehrsleitung in der Umgebung von Barcelona zur Laufzeit organisiert.

Das Evakuierungsmanagement bei Katastrophen profitiert ebenso vom Einsatz von MAS. So können kostengünstig Evakuierungen von Schiffen, Flugzeugen oder Gebäuden schon während des Konstruktionsprozesses simuliert werden. Ebenso kann die Evakuierung von Städten und entsprechend die Organisation von Auffanglagern geplant werden. (Kagaya u. a., 2005; Lämmel, 2011)

Militär

Militärische Einrichtungen nutzen MAS häufig für Trainingseinheiten. Dabei reicht das Spektrum von diplomatischen Missionen über Konvoi-Begleitung (siehe Abb. 2.3) bis hin zu Kampfeinsätzen. Gründe dafür sind unter anderem die Flexibilität der Systeme, vereinfachte Analysemöglichkeiten und auf Dauer deutlich geringere Kosten als reale Manöver. Zu den am weitesten verbreiteten Systemen gehört das *Virtual Battlespace*-System von *Bohemia Interactive Simulations*. [2] In vielen Fällen wird zusätzlich Hardware verwendet, um die Immersion zu steigern.

Abbildung 2.3: Beispiel für eine interaktive Agentensimulation. (Quelle: Wikipedia[3])

[2] https://bisimulations.com/

[3] https://en.wikipedia.org/wiki/DARWARS (03.08.2015)

Spiele

Mit steigender Rechenleistung wuchs auch das Interesse der Spielebranche am Einsatz von Agenten. Die Einsatzbereiche in der Simulation spiegeln sich auch in den Genres der Spiele wider. So werden in der Städtebau-Simulation *Cities: Skylines* Agenten in Kombination mit Statistiken eingesetzt, um auch bei großen Städten gewährleisten zu können, dass das Spiel flüssig läuft. (Paradox Interactive, 2015) Würde man in diesem Szenario rein auf Agenten setzen, würde das System voraussichtlich zusammenbrechen.

In Rollenspielen, Ego-Shootern und Militärsimulationen wird die KI zunehmend mit Agenten modelliert, die in größerem Maße auf das Verhalten des Spielers eingehen, ein eigenständiges Handeln simuliert und Aktionen selbst plant. Ein besonderes Augenmerk ist dabei auf die Kommunikation zwischen den Agenten zu legen. (Westra u. a., 2009) In vielen Fällen wird die Qualität der KI jedoch durch die verfügbare Rechenleistung eingeschränkt oder absichtlich durch die Entwickler begrenzt, da sie dem Spieler sonst überlegen wäre.

Der Übergang zwischen der Simulation realer Szenarien und Spielen wird bei der Spieleserie *ArmA* besonders deutlich, da diese auf der gleichen Engine basiert wie das *Virtual Battlespace*-System.

Film

Insbesondere Filmszenen mit einer großen Zahl von Charakteren, wie zum Beispiel eine große Versammlung oder Massenschlachten, wären äußerst kostenintensiv, wenn man sie mit Statisten umsetzen würde. Mittlerweile existiert Software, die einen Großteil der Akteure durch computergenerierte Agenten ersetzt und so Kosten reduziert. Bekanntheit erlangte dieses Verfahren unter anderem durch den Einsatz in der *Herr der Ringe*-Triologie, bei dem *MASSIVE*[4] zum Einsatz kam. (Klügl, 2006)

2.1.3 Unterschiede zu anderen Simulationsarten

Wie in Abbildung 2.4 zu sehen, gibt es diverse Möglichkeiten, Szenarien zu simulieren. Dabei sind jedoch nicht alle Simulationsansätze für jedes Szenario tauglich.

Makrosimulationen, wie z.B. Differentialgleichungsmodelle, gehen davon aus, dass die Eingabedaten homogen sind. Einzelne Bestandteile des simulierten Problems werden bei dieser Simulationsart nicht betrachtet. Im Gegensatz dazu wird das zu simulierende Problem bei Mikrosimulationen in seine Bestandteile zerlegt.

[4] http://http://www.massivesoftware.com

Name	Makrosimulation (System Dynamics)	Mikroanalytische Simulation	Ereignisbasierte Simulation	Mehr-Ebenen Simulation	Zelluläre Automaten	Agentenbasierte Simulation
Zeitliche Stabilität und Zeitfortschritt	dynamisch, diskret bis kontinuierlich	dynamisch, diskret	dynamisch, diskret	dynamisch, diskret bis kontinuierlich	dynamisch, diskret	dynamisch, diskret
Lösungsverfahren	Simulation	Simulation	Simulation	Simulation	Simulation	Simulation
Folgerungsmethode	deterministisch	stochastisch	deterministisch, stochastisch	deterministisch, stochastisch	deterministisch, stochastisch	deterministisch, stochastisch
Operationalisierung	quantitativ	qualitativ, quantitativ	qualitativ, quantitativ	qualitativ, quantitativ	qualitativ, quantitativ	qualitativ, quantitativ
Funktionen	nichtlinear	nichtlinear	nichtlinear	nichtlinear	regelbasiert	regelbasiert
Abbildungsebene	Makroebene	Mikro- und Makroebene	Mikroebene	Mikro- und Makroebene	Mikro- und Makroebene	Mikro- und Makroebene
Modellzweck	Vorhersage	Vorhersage	Vorhersage	Erklärung	Erklärung, Vorhersage	Erklärung, Vorhersage
Möglichkeit der Darstellung von Interaktionen	Nein	Nein	Nein	Manchmal	Ja	Ja
Möglichkeit der Darstellung von komplexem Markenwahlverhalten	Nein	Ja	Nein	Nein	Nein	Ja
Möglichkeit der Darstellung von Konsumenten-heterogenität	Nein	Ja	Nein	Nein	bedingt	Ja

Abbildung 2.4: Verschiedene Simulationsarten (Quelle: Feddersen, 2010, S. 150)

Wollte man zum Beispiel einen Bahnhof simulieren, so würden in einer Makrosimulation Variablen, wie die Zahl der wartenden Fahrgäste oder Fußgängerdichte, homogen simuliert. In einer Multiagentensimulation, die in die Kategorie der Mikrosimulationen fällt, kann jeder Fußgänger mit individuellem Ziel und unterschiedlicher Geschwindigkeit simuliert werden. Agenten müssen jedoch nicht zwangsweise einzelne Objekte, wie Personen, Tiere oder Pflanzen, abbilden. Sie können auch komplexere Einheiten mit Zustandsvariablen darstellen.

Zelluläre Automaten ähneln Agentensimulationen in vielen Punkten und werden manchmal mit diesen gleichgesetzt. Bei zellulären Automaten wird die Basiseinheit als Zelle räumlich definiert, deren Wert nach für den gesamten Automaten gültigen Regeln aktualisiert wird. Multiagentensimulationen sind hier deutlich flexibler, da sich viele unterschiedliche Agenten mit komplett unterschiedlichen Regelsätzen definieren lassen. (Klügl, 2006; Feddersen, 2010)

2.2 Visualisierung

Nach Keim (2002) kann eine Visualisierung von Daten aus vielen Gründen hilfreich sein. Bei einer Simulation kann sie zur Überprüfung des Modells, zur Suche nach bisher unbekannten

Zusammenhängen oder auch zur Veranschaulichung für weniger fachkundige Betrachter dienlich sein. Daten in einer Visualisierung sind wesentlich leichter zugänglich, als eine große Menge von Zahlen. Komplexe Zusammenhänge lassen sich durch eine Visualisierung häufig wesentlich leichter entdecken und nachvollziehen, als durch das Analysieren von Zahlensätzen. Die detaillierte Analyse der Daten bleibt im Endeffekt zwar nicht erspart, allerdings kann der Aufwand meist deutlich reduziert werden, wenn aus der Visualisierung bereits zu entnehmen war, an welcher Stelle aussagekräftige oder relevante Datensätze liegen. Shneiderman (1996) nennt als zentrale Richtlinie für das Design von Visualisierungen ein "Visual Information Seeking Mantra". Es besteht aus:

- Übersicht
- Zoomen und Filtern
- Details bei Bedarf

Speziell im Bereich der 3D-Visualisierungen listet er Funktionen auf, die der Orientierung im Raum dienen:

- Gesamtübersichten
- Landmarken
- Perspektiven
- Transparenz
- Farbkodierung

Eines der beliebtesten Beispiele für den Nutzen von Visualisierungen zur Analyse ist die *Ghost Map* des britischen Arztes John Snow (siehe Abbildung 2.5). Dieser legte während der Choleraepidemie 1854 in London eine Karte an, auf der alle Choleraopfer verzeichnet wurden. Es wurde deutlich, dass sich die Zahl der Opfer in der Umgebung einer Pumpe deutlich häufte. Durch die Stilllegung der Pumpe wurden die Einwohner dazu gezwungen, andere Pumpen aufzusuchen. Diese Maßnahme führte binnen weniger Tage zu einem deutlichen Rückgang der Opferzahlen. (Lehmann u. a., 2010)

Abbildung 2.5: Die *Ghost Map* zur Suche nach der Cholera-Quelle. (Quelle: Wikipedia[5])

Als weiteres Beispiel für die Visualisierung mehrdimensionaler Datensätze kann ein Diagramm über Napoleons Russlandfeldzug zwischen 1812 und 1813 von Charles Joseph Minard angeführt werden (siehe Abb. 2.6). Auf dieser Grafik werden verschiedene Daten wie Temperatur, Position und Soldatenzahl gleichzeitig abgebildet. Trotz der großen Menge an vermittelten Daten erhält man bereits auf den ersten Blick eine Vorstellung von dem, was vermittelt werden soll. Ähnliche Karten wurden von Minard zu Emigrantenzahlen sowie zu Im- und Exporten erstellt.

[5] https://de.wikipedia.org/wiki/Cholera#Forschungsgeschichte (03.08.2015)

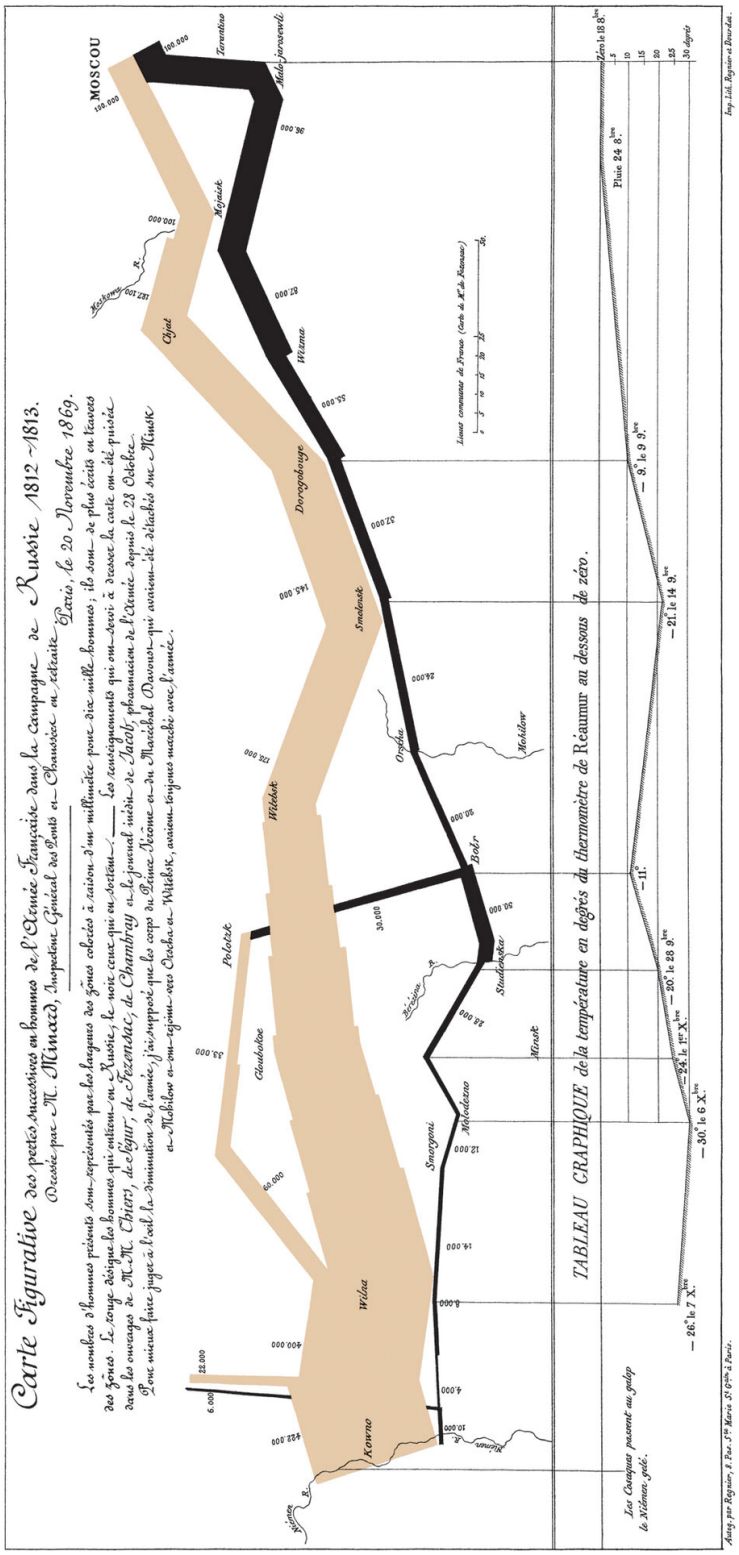


Abbildung 2.6: Visualisierung von Napoleons Russlandfeldzug 1812/13 von Charles Joseph Minard. Der Hinweg ist beige, der Rückweg schwarz. (Quelle: Wikipedia[6])

Im Falle einer Simulation erfüllt die Visualisierung, zusätzlich zur Ergebnisanalyse, einen weiteren Zweck. Schon während der Entwicklung kann der Entwickler feststellen, ob das Modell korrekt arbeitet und wo eventuelle Probleme zu suchen sind. Schon mit einer sehr simplen Visualisierung lässt sich zum Beispiel bei einem Jäger-Beute-Szenario nachverfolgen, ob sich die Agenten plausibel verhalten.

Komplexere Visualisierungen können zusätzlich Informationen über die einzelnen Agenten darstellen. Dazu können aktuelle Motive oder Attribute gehören. Einige Zustände eines Agenten kann man über ein individuelles Modell oder unterschiedliche Färbung darstellen. Dabei ist darauf zu achten, dass die Unterschiede schnell zu erkennen sind, indem eine ausreichend grobe Unterteilung gewählt wird. Färbt man zum Beispiel das Modell eines Agenten um dessen Hungergrad darzustellen, ist es sinnvoller, dies in Schritten von 10% statt 1% zu tun. Möchte der Betrachter genauere Informationen zum Hungergrad, kann er den betreffenden Agenten einzeln inspizieren.

2.3 Streaming und Netzwerkkommunikation

Da die Betrachtung der Simulation zur Laufzeit möglich sein soll, muss Streaming zum Einsatz kommen. Es gibt verschiedene Arten des Streamings, die hier kurz vorgestellt werden sollen.

2.3.1 Videostreams

Bei Videostreams erhält der Client eine fertige Videodatei (z.B. YouTube) oder die Perspektive einer Kamera bei einem Live-Event. Da diese Art des Streamings nicht interaktiv ist, fällt sie für den konkreten Anwendungsfall weg, da zum Beispiel keine detaillierte Analyse einzelner Agenten möglich wäre.

2.3.2 Dedizierte Berechnung

Mittlerweile gibt es auch Dienste, die Spiele und ähnliche 3D-Anwendungen im Rechenzentrum rendern und die generierten Bilder zurücksenden. *OnLive Games* (2014), Gaikai Inc. (2015) und Sony Computer Entertainment (2015) Damit könnte für alle Clients eine einheitliche und hohe Renderqualität ohne eigenen Rechenaufwand gewährt werden. Die hierbei entstehende Verzögerung wäre akzeptabel, da es bei der reinen Betrachtung nicht darauf ankommt, dass die Eingaben sofort umgesetzt werden. Dagegen spricht jedoch der hohe Bedarf an Bandbreite, der nicht überall gewährleistet werden kann. Hinzu kommt, dass entweder zusätzliche Kosten

[6] https://de.wikipedia.org/wiki/Charles_Joseph_Minard (09.08.2015)

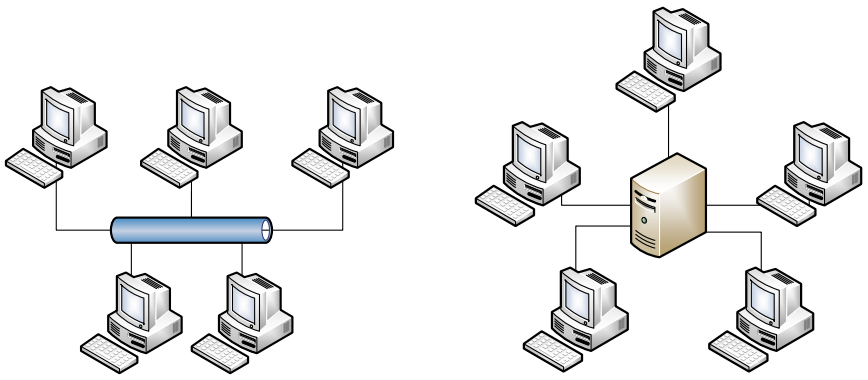
entstehen würden oder die Server-Hardware zusätzlich belastet werden würde. (Huang u. a., 2013; Claypool u. a., 2012; Chen u. a., 2011)

2.3.3 Streaming der Agenteninformationen

Eine weitere Möglichkeit ist es, die ermittelten Daten vom Server in einer möglichst kompakten Art an den Client zu übertragen, ähnlich wie es zum Beispiel bei Online-Spielen geschieht. Hierbei gibt es unterschiedliche Verfahren:

- Direkter Austausch unter den Spielteilnehmern
- Austausch über eine Client-Server-Struktur

Der direkte Austausch ist heute kaum noch üblich, besonders wenn es viele Teilnehmer gibt. Dies liegt vor allem daran, dass die Knoten leicht asynchron werden. In der Regel wird heute auf das Client-Server-Modell gesetzt. In diesem Modell wird die Synchronisierung über festgelegte Ticks zwischen dem Client und dem Server hergestellt. Während dieser Ticks simuliert der Server die Welt basierend auf den vorliegenden Daten und den Benutzereingaben, die ihm zugesendet werden. Nach Abschluss eines Ticks werden alle Clients über die Änderungen informiert. Zwischen den einzelnen Updates kümmert sich der Client darum, mittels Prognose und Interpolation, die Bewegungen möglichst fließend erscheinen zu lassen. (Fiedler, 2010; Valve, 2014)



(a) Direkte Kommunikation zwischen den Rechnern

(b) Kommunikation über einen Server

Abbildung 2.7: Übliche Arten der Kommunikation

Um die Kommunikation weiter zu optimieren wird häufig zusätzlich betrachtet, welche Entitäten für welchen Betrachter durch Sichtfeld oder Einflussbereich relevant sind.

Weiterhin werden Prioritäten für unterschiedliche Objekttypen gesetzt. So wird zum Beispiel zwischen einem Fahrzeug, dessen Status sich oft schnell ändert und häufig Updates senden muss, und dem Inventar eines Charakters, das sich seltener ändert, unterschieden. (Epic Games, 2012)

3 Analyse

In diesem Kapitel werden die Anforderungen, technischen Gegebenheiten und alternative Systeme analysiert.

3.1 Showcase: Der Krüger-Nationalpark

Als Showcase für diese Arbeit soll die Simulation eines Modells des Krüger-Nationalparks (KNP) dienen. Der KNP ist ein ca. 20.000 km² großes Wildschutzgebiet im Nordosten von Südafrika. Dieser Park beherbergt eine große Zahl von Säugetieren, Vögeln und Reptilien.

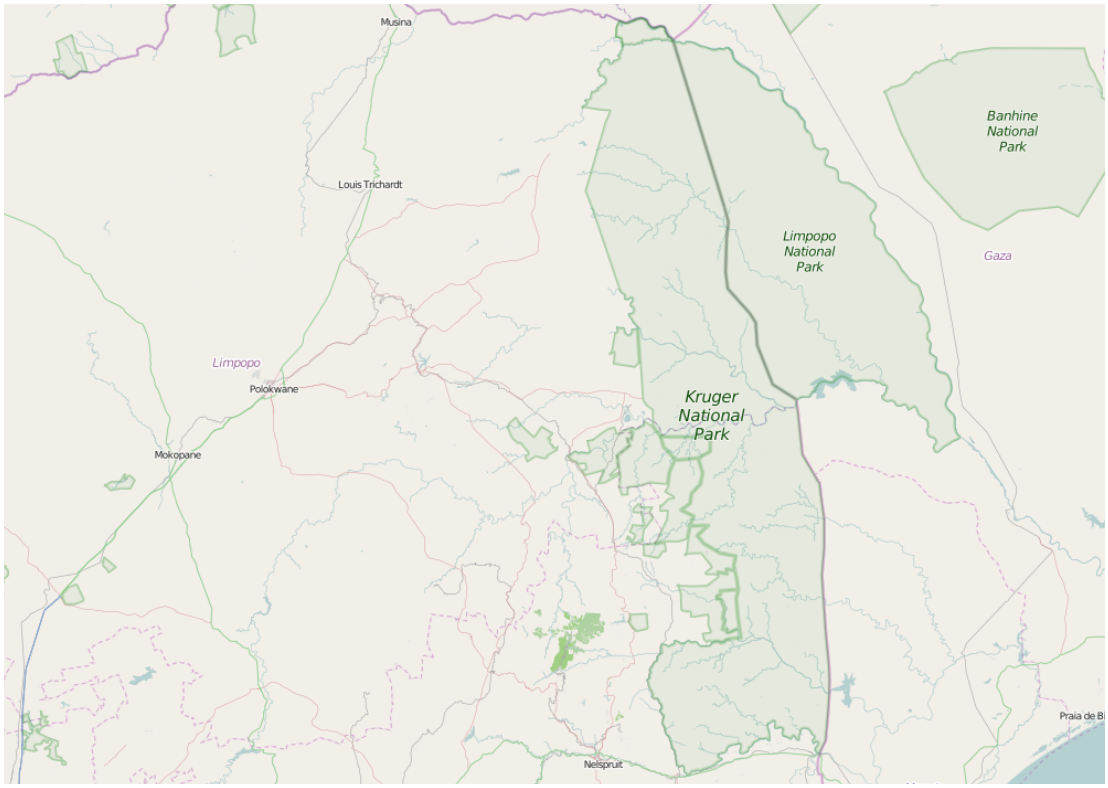


Abbildung 3.1: Der Krüger-Nationalpark im Nordosten von Südafrika (Quelle: Open Street Maps)

Das Modell wird in der ersten Stufe die Ausbreitung und die Überlebenswahrscheinlichkeit von Marula-Bäumen (Sclerocarya birrea) in diesem Gebiet simulieren. Dabei sollen insbesondere Verbreitung und Wachstum der Bäume simuliert werden. Der Marula-Baum wurde ausgewählt, da er am einfachsten zu simulieren ist und es bereits ausreichend Daten gibt. Als besondere Merkmale stechen seine Robustheit gegenüber Verletzungen der Rinde bis hin zu Feuer, sowie seine gesellschaftliche Relevanz als Heilpflanze hervor. Darüber hinaus ist er eine der wichtigsten Nahrungsquellen für Pflanzenfresser im KNP.

Im Verlauf der Modellentwicklung soll auch der detaillierte Einfluss durch Elefanten einbezogen werden, die auch als einzelne Agenten auftreten sollen. Diese helfen dem Baum zwar bei der Samenverbreitung, sind jedoch gleichzeitig auch Störer, da sie nicht selten die Rinde des Baumes verletzen, wodurch seine allgemeine Verletzbarkeit für einen gewissen Zeitraum zunimmt, oder ihn sogar umwerfen. (Helm, Wilson u. a., 2011; Helm, Witkowski u. a., 2009)

3.2 Anforderungsanalyse

Um ein Konzept erarbeiten zu können, müssen zunächst Anforderungen gesammelt und formuliert werden. Gemeinsam mit dem Showcase dienen diese als Basis für die weiteren Überlegungen zum Vorgehen.

1. Die Simulationsdaten sollen live dargestellt werden können.
2. Die Darstellung der Simulationsdaten soll pausiert oder zurückgespult werden können.
3. Um die Netzwerkauslastung zu minimieren, soll die Datenübertragung möglichst ressourcensparend ablaufen.
4. Das Szenario soll leicht austausch- oder erweiterbar sein.
5. Die einzelnen Simulationslayer sollen ein- und ausblendbar sein.
6. Flächendaten wie Temperatur oder Feuchtigkeit sollen als Höhenprofil und / oder Geländefärbung dargestellt werden können.
7. Es soll möglich sein, echte Kartendaten zu verwenden.
8. Agenten und Objekten sollen Gruppen zugewiesen werden können, die zuvor frei erstellt werden können.
9. Mittels einfacher Objekte wie z.B. Kugeln sollen beispielsweise Marken gesetzt werden können.

10. Die Visualisierung soll in die Websuite von MARS integriert werden.
11. Die Lösung für die Visualisierung soll kostenlos oder zumindest günstig sein.
12. Die Transparenzregeln für verteilte Systeme[1] sollen, soweit es möglich ist, eingehalten werden.

3.3 Herausforderungen

Das MARS-System zeichnet sich insbesondere durch Verteil- und Skalierbarkeit aus. Diese Eigenschaften werden unter anderem dazu genutzt, größere Mengen an Agenten gleichzeitig zu simulieren.

Wollte man alle simulierten Agenten gleichzeitig anzeigen lassen, würde dies mit hoher Wahrscheinlichkeit in einer ruckelnden und unbrauchbaren Darstellung enden. Außerdem würde dadurch ein immenses Datenaufkommen entstehen. Die Lösung zu diesem Problem orientiert sich an der Technik der Online-Spiele, da auch die Visualisierung nur Daten erhält, die für den betrachteten Ausschnitt relevant sind. (siehe 2.3.3)

Durch die Verteilung des System kann zudem keine zuverlässige Annahme über das Eintreffen der Daten gemacht werden. Hier muss sichergestellt werden, dass die eingetroffenen Daten zu dem aktuell dargestellten Tick gehören.

3.4 Darstellung

Da das exakte Aussehen für die Auswertung der Daten nicht relevant ist, genügt es, sich auf grundlegende Informationen zu beschränken, die dafür allerdings möglichst auf den ersten Blick erkennbar sein sollten. Uneinheitliche Agentenmodelle könnten die Auswertung der Simulation sogar erschweren. Aus diesem Grund wird allen Agenten einer Art das gleiche Modell zugewiesen und in einem angemessenen Rahmen, wie z.B. der Größe, modifiziert. (Deussen, 2003)

Für Agenten, die kein eigenes, sondern nur ein verallgemeinertes Modell haben, soll dieses jeweils für eine Agentenart individuell eingefärbt werden. Dadurch sollen auch diese Agenten unterscheidbar bleiben.

Ist der Betrachter an detaillierten Informationen, wie zum Beispiel den Abmessungen eines Baumes oder der Motivation eines Dorfbewohners interessiert, soll er diese durch Anklicken des entsprechenden Agenten in einem Infofenster angezeigt bekommen.

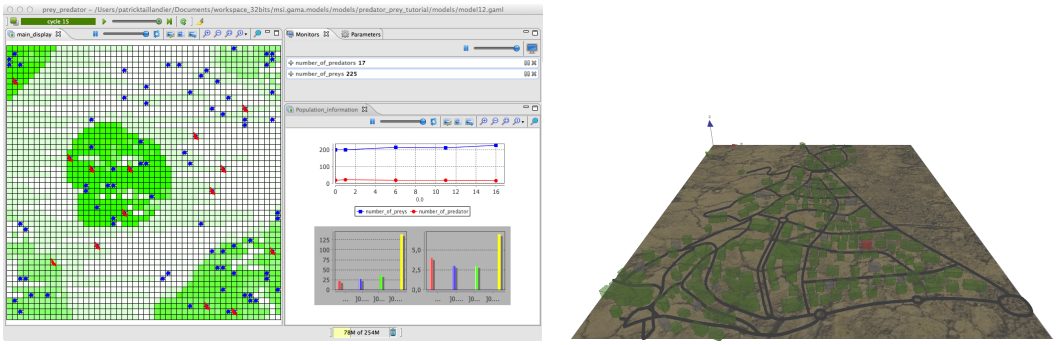
[1] https://de.wikipedia.org/wiki/Transparenz_(Computersystem) (31.08.2015)

3.5 Existierende Systeme

In diesem Abschnitt soll ein Überblick über nennenswerte Vertreter der Multiagentensysteme und deren Visualisierungstechniken gegeben werden.

3.5.1 GAMA

GAMA (GIS & Agent-based Modelling Architecture)[2] wird von einer französisch-vietnamesischen Forschungsgruppe entwickelt. Diese Gruppe hat sich zum Ziel gesetzt, ein System zu schaffen, das es auch Nicht-Programmierern erlaubt, komplexe Modelle zu erschaffen. Das System basiert auf der Programmiersprache Java und wird unter der Verwendung der Eclipse-Plattform als komplette Entwicklungsumgebung für Multi-Agenten-Simulationen bereitgestellt. GAMA bietet eine einfach zu bedienende, aber mächtige Definitionssprache namens *GAML* an, die sich deutlich an XML orientiert. Das Framework ermöglicht es Entwicklern, GIS-Daten in einfacher Weise zu importieren und in der Simulation zu nutzen. Außerdem gibt es die Möglichkeit, das System einfach mit Plug-Ins zu erweitern. Eine Verteilung des Systems auf mehrere Computer ist nicht vorgesehen. Simulationen in GAMA sind mit 75.000 Agenten schon sehr stark ausgelastet. Auch die Visualisierung von Ergebnissen wird mit GAML konfiguriert, wobei diverse Optionen zur Auswahl stehen. Dabei ist es möglich, eine beliebige Anzahl Displays mit zwei- oder dreidimensionalen Darstellungen zu definieren. Ebenso lässt sich ein Aktualisierungsintervall und ein Zeitversatz festlegen. (Amouroux u. a., 2009; Hüning, 2013)



(a) 2D-Visualisierung mit Diagrammen

(b) 3D-Visualisierung mit GIS-Daten

Abbildung 3.2: 2D- und 3D-Visualisierungen in GAMA (Quelle: http://gama-platform.org/)

[2] http://gama-platform.org/ (02.09.2015)

3.5.2 JADE

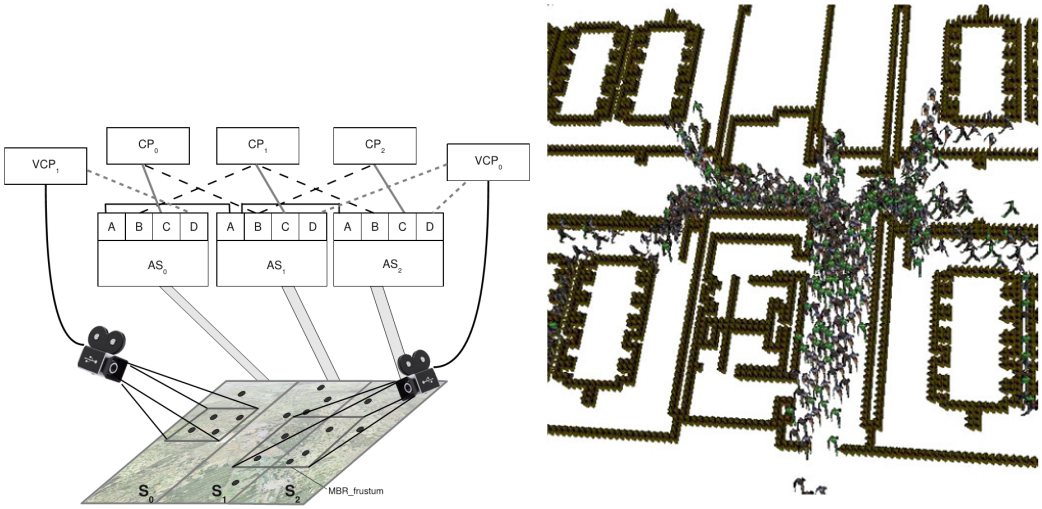
Das JADE-Framework[3] wurde von der Telecom Italia entwickelt und implementiert die Agenten-Architekturvorgaben der FIPA (Foundation for Intelligent Physical Agents), deren Nutzung sehr weit verbreitet ist. JADE gehört mittlerweile zu den meistgenutzten MAS-Tools und bietet diverse Erweiterungen. Zu den größten Kritikpunkten zählt der hohe Zeitaufwand der Kommunikation. Die Verteilung in JADE wird mit so genannten Containern umgesetzt. Es gibt einen *main container*, in dem unter anderem die anderen Container verwaltet werden. Agenten werden in ihren eigenen Containern als einzelne Threads gestartet, erhalten eine global einzigartige Adresse und werden in das Register des *main containers* eingetragen. Das Verzeichnissystem für die Agenten verwendet LDAP (*Lightweight Directory Access Protocol*). Für die Kommunikation der Agenten untereinander wird, wenn sie im gleichen Container liegen, das Java-Eventsystem verwendet oder anderenfalls RMI (*Remote Method Invocation*). Sowohl die übermäßige Verwendung des LDAP-Service mit überflüssigen Anfragen als auch die verhältnismäßig langsame Kommunikation zwischen den Agenten erzeugen bei diesem System Flaschenhälse. (Mengistu u. a., 2008) JADE selbst verfügt über keine Visualisierung. Es gibt jedoch Modifikationen wie *JGOMAS*[4], welches auf Simulationen im Spielebereich ausgelegt wurde. (Barella u. a., 2007)

3.5.3 Vigueras

Vigueras, J. M. Orduña u. a. (2013) und Lozano u. a. (2009) beschreiben ein System, das MARS sehr ähnlich ist. Die Verteilung beschränkt sich in diesem System auf Geländeabschnitte und verzichtet auf Layer. Pro Gebietsabschnitt gibt es einen *Action Server* (AC), der unter anderem die Aktionen der Agenten (CP von *Client Process*) koordiniert, Kollisionserkennung ermöglicht und die Visualisierungsclients (VCP) mit Daten versorgt.

In dem System wird dem Action Server ein Zeitlimit von 250 ms für die Ausführung von Aktionen, wie z.B. der Beantwortung von Anfragen gesetzt. Dies ist laut Henderson und Bhatti (2003) die maximale, vertretbare Verzögerung für eine interaktive Darstellung. Ausstehende Aktionen werden im Folgetick bearbeitet. Eine Visualisierung kann kontinuierlich mit Daten versorgt werden. Riskant ist hierbei, dass sich irgendwann Aktionen aus vergangenen Ticks aufstauen und das System ins Stocken gerät. Als Lösung für diesen Flaschenhals wird das Hinzufügen von Hardware vorgeschlagen.

[3] http://jade.tilab.com/ (02.09.2015)
[4] http://gti-ia.upv.es/sma/tools/jgomas/index.php (02.09.2015)



(a) Architektur (b) Evakuierungsszenario

Abbildung 3.3: Konzept von Vigueras (Quelle: Vigueras, J. Orduña u. a. (2011))

3.5.4 High Level Architecture

Die *High Level Architecture* (HLA) wurde 1995 vom US Department of Defense entwickelt, um ein Framework zu schaffen, mit dem die Interaktion zwischen Simulationen vereinfacht wird. Ein weiteres Ziel war bei der Konzeption die Wiederverwendbarkeit von Simulationsteilen. Im Jahr 2000 wurde HLA als IEEE-Standard angenommen. Die generelle Idee hinter HLA ist es, aus mehreren kleinen Simulationen eine große Simulation zu bauen. Im Zentrum stand zum Zeitpunkt der Entwicklung damit nicht die Verteilung einer einzelnen Simulation.

Die HLA ist mittlerweile sehr weit verbreitet. Simulationen, die diese Architektur nutzen, kommen aus den unterschiedlichsten Bereichen. Dazu zählen sowohl Lehre, Training, Analyse als auch Unterhaltung.

Ein System, das mit dieser Architektur arbeitet, wird im ganzen als *Federation* bezeichnet. Diese besteht aus einzelnen *Federates*, die zum Beispiel Simulationen, bemannte Simulatoren oder ein Datensammler sein können. Die HLA legt nicht fest, welche Daten dargestellt werden oder wie dies geschieht. Wichtig ist jedoch, dass alle *Federates* bestimmte Eigenschaften implementieren, die der Kommunikation untereinander dienen.

Die Koordination der Kommunikation läuft über die *Runtime Infrastructure* (RTI). Alle Interaktionen zwischen den *Federates* laufen durch die RTI, wodurch diese zur zentralen Komponente wird. Obwohl sie verteilt angelegt werden kann, wird die RTI dadurch auch gleichzeitig ein potentieller Flaschenhals.

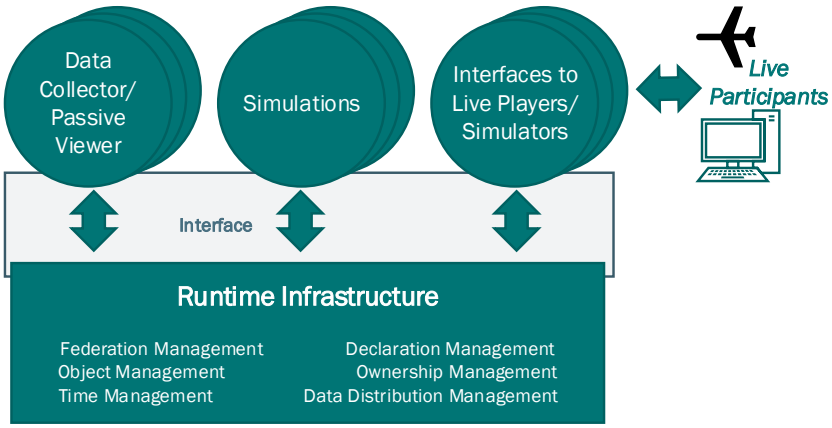


Abbildung 3.4: Aufbau einer HLA-Federation (nach Dahmann, Fujimoto und Weatherly (1997))

Mit dem *HLA Federation Object Model* (FOM) wird definiert, welche Objekte, Attribute und Interaktionen innerhalb der *Federation* zur Verfügung stehen. Im *HLA Simulation Object Model* (SOM) wird für die jeweilige Simulation festgelegt, welche Informationen für andere Teilnehmer der *Federation* verfügbar gemacht werden. Der Inhalt der FOM und SOM sind nicht definiert. Der Aufbau der Objekte ist jedoch durch das *HLA Object Model Template* festgelegt. (Dahmann, Fujimoto und Weatherly, 1997)

Yu und Guan (2011) bemängeln an der HLA vor allem die Verschwendung von Rechenleistung, die ineffiziente Arbeit mit großen Datenmengen und große Hürden bei der Lastenverteilung.

3.5.5 Das MARS-System

Das MARS-System ist eine Eigenentwicklung der HAW. Es zeichnet sich vor allem durch seine Skalier- und Verteilbarkeit aus. Ebenso wurde viel Wert auf die Wiederverwendbarkeit und eine einfache Bedienung für nicht-fachkundige Benutzer gelegt. (Hüning u. a., 2014) Die Simulation wird zentral über ein Webinterface gesteuert, in das der Nutzer seine Daten und Agenten- sowie Modelldefinitionen hochlädt und den Simulationslauf startet. Damit ist das System in die Kategorie MSaaS einzuordnen.

Die Knoten von MARS werden zentral durch die Komponente LIFE koordiniert, an die auch die Komponenten Surveyor, die zur Überwachung dient, und VIS, das die Schnittstelle für die Visualisierung anbietet, angebunden sind.

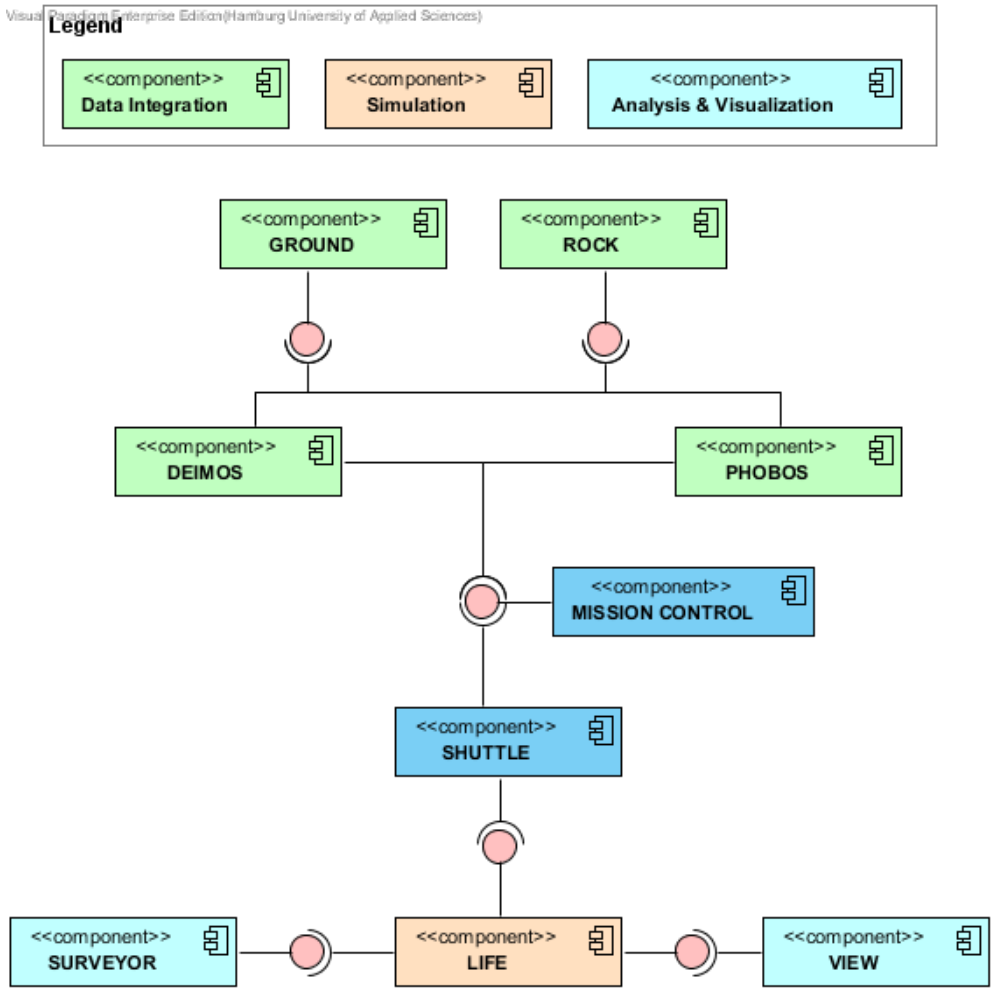


Abbildung 3.5: Die Architektur des kompletten MARS-Systems (Grafik von Christian Hüning)

Mittels LIFE werden Layer erstellt, die Agentengruppen zusammenfassen. Diese Layer können verteilt ausgeführt werden.

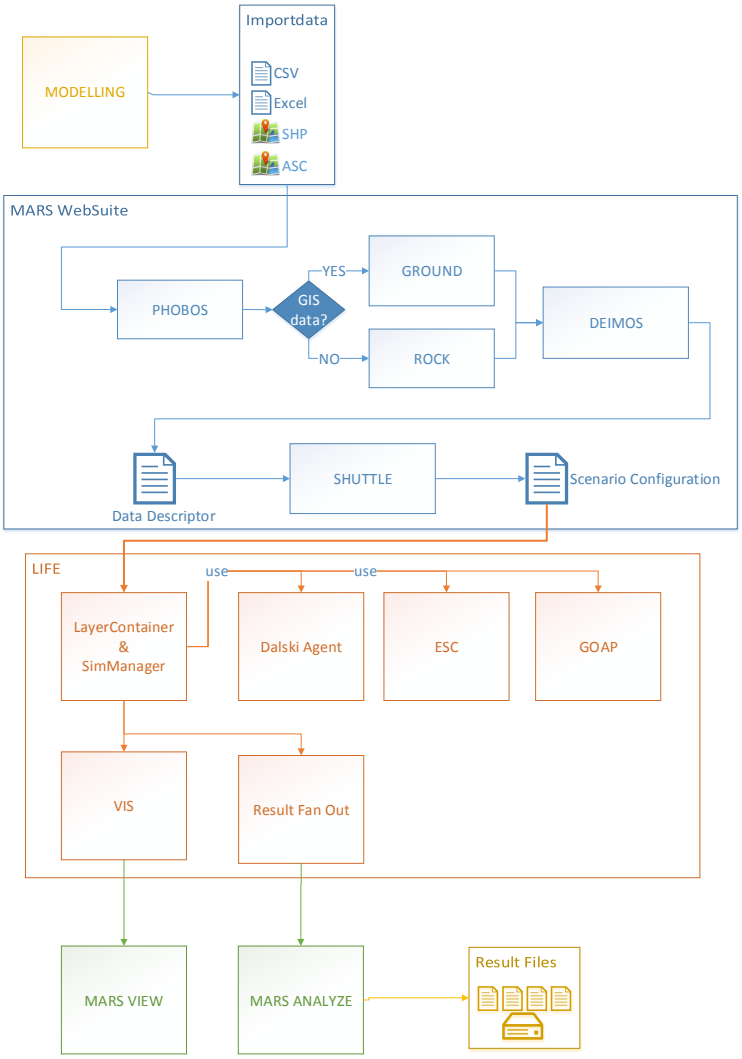


Abbildung 3.6: Der Workflow in MARS. (Grafik von Christian Hüning)

4 Konzeption und Umsetzung

In diesem Kapitel soll ein Konzept erarbeitet werden, dessen Umsetzung erklärt und die Verwendung beispielhaft dargestellt wird.

Obwohl das Protokoll nicht von einer 3D-Engine abhängig sein soll, wird in diesem Abschnitt die Unity 3D-Engine als Referenz verwendet. Die Verwendung dieser Engine während des Entwicklungsprozesses hatte unvermeidlich auch Einfluss auf die Umsetzung. So wurden einige Abschnitte für die Zusammenarbeit mit Unity optimiert, sollten jedoch auch mit anderen Engines kompatibel sein.

4.1 Grundlegendes Konzept

Aus dem MARS-System sind für die Visualisierung insbesondere die Komponenten LIFE, VIS und VIEW sowie GROUND relevant. Außerdem legt das System eine Snapshot-Datenbank über den Verlauf der Simulation an. Die Komponente LIFE ist unter anderem für die Organisation der Simulationsknoten zuständig und stellt den Agenten-Entwicklern die nötigen Schnittstellen zur Verfügung. VIS fungiert als Schnittstelle zwischen der Simulation selbst und dem Client VIEW, der letztendlich zur Betrachtung verwendet wird.

VIS entscheidet nach einer Anfrage von VIEW darüber, von wo die Simulationsdaten abgerufen werden müssen. Wird der aktuelle Stand der Simulation betrachtet, leitet er die Anfrage an die LIFE-Simulationsknoten weiter. Die Knoten senden ihre Daten danach direkt an den anfragenden Client. Informationen über die Knoten, wie zum Beispiel die Verteilung der Layer, erhält VIS vom Simulation Manager.

Fragt der Client Daten aus der Vergangenheit an, lädt VIS die entsprechenden Daten aus der Datenbank und gibt diese zurück.

Da der Client fortlaufend über Änderungen in dem für ihn relevanten Areal informiert werden muss, speichert die zuständige Komponente, welches Areal der Client zur Zeit betrachtet. Finden Änderungen in diesem Bereich statt, wird der Client darüber informiert.

4.2 Annahmen

Aus der Anforderungsanalyse (siehe 3.2) kann eine große Bandbreite an Bedürfnissen abgeleitet werden. Um einen Entwurf anfertigen zu können, werden einige Annahmen getroffen, die diese Bedürfnisse einschränken oder konkretisieren.

4.2.1 GIS-Daten und Geländetexturen

Ausgehend vom aktuellen Showcase und den meisten geplanten Simulationen wird angenommen, dass für die Bodentexturen in der Regel Kartenmaterial vorliegt. Möchte man das Gelände anhand der Höhe automatisch färben, müsste ein Shader[1] erstellt und dieser konfiguriert werden. Ebenso ist das Aufbringen von Texturen aufwändig, da hierzu sogenannte *Splatmaps*[2] übertragen und die entsprechenden Texturen auf die Areale projiziert werden müssen. Als Quelle für die GIS-Daten wird ein Server angenommen, der *Web Map Service* (WMS) zur Verfügung stellt. (Siehe 3.2, Punkte 6 und 7)

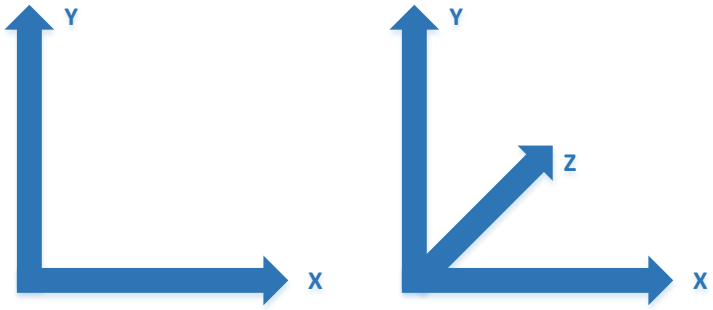
4.2.2 Koordinatensystem

Die verwendeten Koordinatensysteme orientieren sich an dem System von Unity 3D. Bei der Terraingenerierung und -aufteilung wird in der Vogelperspektive ein x-y-Achsensystem verwendet. Ortsangaben werden in einem x-y-z-Achsensystem umgesetzt, wobei z in die Tiefe des Raumes und y in die Höhe geht. Bei allen Koordinatenangaben wird davon ausgegangen, dass sie sich auf das Koordinatensystem der Visualisierung beziehen.Die Koordinaten (0,0,0) sind dabei die untere linke Ecke des dargestellten Raumes. Alle Entitäten haben somit positive Koordinaten.

4.2.3 Terrain

Die Terraininformationen, wie beispielsweise die Einheiten der Höhenangaben, können sehr unterschiedlich formatiert sein. Es wird davon ausgegangen, dass der Server die Daten zuvor einheitlich aufbereitet hat. Ebenso übernimmt er ggf. die weitere Vorbereitungen des Terrains.

[1] https://de.wikipedia.org/wiki/Shader (24.07.2015)
[2] https://en.wikipedia.org/wiki/Texture_splatting (24.07.2015)



(a) Achsenaufteilung im 2D-System (b) Achsenaufteilung im 3D-System

Abbildung 4.1: Koordinatensysteme im Visualisierungsprotokoll

4.2.4 Definitionen und Modelle

Für eine einfache Handhabung wird davon ausgegangen, dass der Simulationsmodellierer bereits zu Beginn alle benötigten 3D-Modelle und die zugehörigen Definitionen festlegt. Ebenso ist es zunächst nicht vorgesehen, nachträglich neue Gruppen einzufügen.

4.2.5 3D-Modelle

Die Modelle für die Unity 3D-Engine müssen zunächst in ein entsprechendes Format (*Prefab*) umgewandelt werden. Eine entsprechende Umwandlung wird für die Beispielimplementierung vorausgesetzt.

4.2.6 Interaktivität

Bei der Entwicklung wurde keine Einflussnahme auf die Simulation zur Laufzeit eingeplant.

4.3 Nachrichtenaustausch

Da die Engine keine Threads erlaubt, kann im Hintergrund keine Kommunikation stattfinden. Es bleibt somit nur die Möglichkeit, die Nachrichten im Hauptthread, der bei jedem Bildaufbau aktiv wird, zu senden und abzurufen. Von den Quellen können jedoch jederzeit neue Daten eintreffen. Diese müssen daher bis zur nächsten Bildberechnung zwischengespeichert werden.

Da es gegen die Transparenzregeln verstoßen würde, wenn die Visualisierungskomponente die einzelnen Knoten kennt, kommt ein direkter Kontakt zwischen diesen beiden Komponenten

nicht in Frage. Es bleibt daher nur noch die Möglichkeit, VIS mit einem Nachrichtenspeicher zu ergänzen oder einen Message-Broker einzusetzen.

Die Erweiterung von VIS hätte zusätzlichen Entwicklungs- und Wartungsaufwand zur Folge. Ein Message-Broker wie RabbitMQ ist bereits ausgereift und verfügt zudem über die Möglichkeit, verteilt ausgeführt zu werden, falls dies gewünscht wird.

Im aktuellen Entwurf horcht VIS auf einer Queue auf Anfragen oder Befehle von Clients. Bei der ersten Anfrage eines Clients wird eine eigene Queue mit einer eindeutigen ID für diesen Client erstellt. Alle Informationen, egal ob von VIS oder von den Simulationsknoten, erhält er später über diese Queue. So wird die gewünschte Transparenz hergestellt.

4.4 Nachrichtenformat

Um die Nachrichten zu übertragen ist eine Serialisierung notwendig. Hierfür gibt es zahlreiche Formate, jedes mit gewissen Vor- und Nachteilen. Als Kandidaten werden JSON, XML, ProtoBuf und MessagePack einem Vergleich unterzogen. Diese Vorauswahl berücksichtigt unter anderem die Unterstützung möglichst vieler Programmiersprachen.

Bei den meisten Datenformaten gibt es Bedingungen oder Einschränkungen, die auch Auswirkungen auf den Entwurf und die Implementierung des Protokolls haben. Häufig sind dies Einschränkungen bei den verwendbaren Datentypen oder die Notwendigkeit, alle zu serialisierenden Attribute zu kennzeichnen. Aus diesem Grund ist eine Vorauswahl und entsprechende Anpassung notwendig.

Hauptkriterien für die endgültige Wahl des Datenformats sind Verarbeitungsgeschwindigkeit und Freiheiten bei der Architektur. Die Verarbeitungsgeschwindigkeit ist kritisch, da ein langsames Entpacken auf Clientseite zu Rucklern führen kann. Außerdem soll der Simulation selbst so viel Bandbreite wie möglich zur Verfügung stehen. (Siehe auch 3.2, Punkt 3) Freiheiten bei der Architektur werden in vielen Fällen durch zwangsweise öffentliche Variablen oder parameterlose Konstruktoren eingeschränkt.

4.4.1 JSON

JSON (JavaScript Object Notation) ist ein menschenlesbares Datenformat, dessen Aufbau sich an JavaScript orientiert. Gültige JSON-Dokumente können deshalb auch direkt in JavaScript interpretiert werden. Für den Test wurde die Implementierung *Json.NET* von *Newtonsoft* verwendet, da diese sehr verbreitet ist. Um Daten serialisieren zu können, müssen alle Variablen, die serialisiert werden sollen, gekennzeichnet werden.

4.4.2 XML

XML (Extensible Markup Language) ist eine menschenlesbare Auszeichnungssprache, die an HTML erinnert. Im Vergleich zu JSON haben XML-Dokumente, bedingt durch ihre Formatierung, einen größeren Overhead. Mit dem XML-Serialisierer von C# können Dictionaries nicht serialisiert werden. Nutzt man einen Umweg über den *DataContractSerializer* von .NET, müssen die zu serialisierenden Variablen gekennzeichnet werden. Visualisierungen wären zudem darauf festgelegt, dass sie den *DataContractSerializer* unterstützen.

4.4.3 MessagePack

MessagePack ist ein binäres Datenformat das sich ein Vorbild an der Formatierung von *JSON* nimmt. Die Werte werden jedoch komprimiert, sodass sie nicht mehr menschenlesbar sind. Zur Verwendung ist eine externe Bibliothek notwendig. Klassen, die serialisiert werden sollen, müssen einen parameterlosen, öffentlichen Konstruktor haben und dürfen nicht abstrakt sein. Zudem müssen alle zu serialisierenden Variablen öffentlich sein. Diese Bedingung kann mit einem benutzerdefinierten Serialisierer umgangen werden.

4.4.4 Protocol Buffers

Protocol Buffers basiert auf einer Entwicklung von Google. Es handelt sich dabei um ein binäres und damit nicht menschenlesbares Datenformat. Die zu serialisierenden Variablen müssen mit einem Attribut markiert werden, können allerdings auch, genau wie der parameterlose Konstruktor, privat sein. Außerdem verfügen Protocol Buffers über eine eingebaute *Interface Definition Language*, was die Verwendung von unterschiedlichen Programmiersprachen bei den beteiligten Komponenten erleichtert, sollte dies gewünscht werden.

4.4.5 Wahl des Formates

Wie in den Performancetests (siehe 6.2) deutlich wird, sind nur *MessagePack* und *Protocol Buffers* auch bei größeren Datenmengen noch effizient. In den meisten Fällen ist *Protocol Buffers* am schnellsten oder gleich auf mit *MessagePack*. Dies gilt insbesondere im Bereich der Deserialisierung. *Protocol Buffers* bietet bei der Architektur mehr Freiheiten wie beispielsweise private Konstruktoren und Variablen. Aus diesem Grund wird *Protocol Buffers* für die Beispielimplementierung verwendet.

4.5 Agenten- und Objektdarstellung

Ein Ziel des Protokolls ist leichte Erweiterbarkeit und Flexibilität. (Siehe auch 3.2, Punkt 4) Im Idealfall wäre es möglich, dem Client beliebige Informationen zu einem Agenten oder Objekt zu übergeben und dieser generiert daraus ein 3D-Modell. Eine solche Möglichkeit würde jedoch einen deutlichen Aufwand bedeuten. So müsste zum Beispiel detailliert definiert werden, wo sich bei dem Modell Gelenke und Animationspunkte befinden und wie sich diese verhalten. Trotzdem muss eine Möglichkeit geschaffen werden, die es erlaubt, mit einfachen Mitteln unterschiedliche Agenten und Objekte mit unterschiedlichen Modellen darzustellen.

Um dies zu ermöglichen, wird ein Dictionary mit grundlegenden Definitionen für 3D-Modelle angelegt. Um die Palette der verfügbaren Modelle zu erweitern, kann der Simulations-Modellierer weitere Einträge hinzufügen. Diese werden zunächst über einen Oberbegriff, wie zum Beispiel “Vierbeiner” oder “Baum” eingegrenzt. Mit beliebigen Attributen, beispielsweise Spezies und Geschlecht, können den Agenten noch spezifischere Modelle zugewiesen werden.

Wenn der Client Informationen über einen Agenten erhält, werden dessen Attribute mit den existierenden Modellzuordnungen verglichen und das am besten passende Modell gewählt. Für den Fall, dass sich keine Übereinstimmung findet, sollte der Modellierer auch einen Eintrag ohne jegliche Attribute hinzugefügt haben. Dieser wird genutzt, falls kein anderer Eintrag passt. Sollte ein solcher Eintrag nicht vorhanden sein, wird das einfachste Agentenmodell verwendet.

Darüber hinaus soll es dem Modellierer möglich sein, Gruppen zu definieren. (Siehe auch 3.2, Punkt 8) Diese erhalten einen Namen, der als ID fungiert, und eine Farbe, mit der die Zugehörigkeit dargestellt wird. Jeder Agent kann Mitglied mehrerer Gruppen sein. Diese Zugehörigkeiten werden in einer entsprechenden Liste des jeweiligen Agenten verwaltet.

In der aktuellen Umsetzung wird zwischen Agenten und passiven Objekten unterschieden. Sie sind in bewegliche und unbewegliche Agenten bzw. Objekte aufgeteilt. Dies soll einer leichteren Verwaltung in der Visualisierung dienen. Sie ist bis jetzt nicht zwangsweise notwendig, könnte allerdings zum Beispiel zur Priorisierung genutzt werden. Sich bewegende Objekte müssen unter Umständen mit höherer Priorität aktualisiert werden als feststehende.

Zu Beginn der Entwicklung gab es einen weiteren Ansatz. Dieser basierte darauf, für jeden Agententyp eine eigene Klasse anzulegen. Dieser Ansatz wurde jedoch aufgrund mangelnder Flexibilität und Wartbarkeit verworfen. Insbesondere, da sowohl Eingriffe in den Code der Visualisierung als auch in die Protokollbibliothek notwendig gewesen wären, die Laien nicht zumutbar sind.

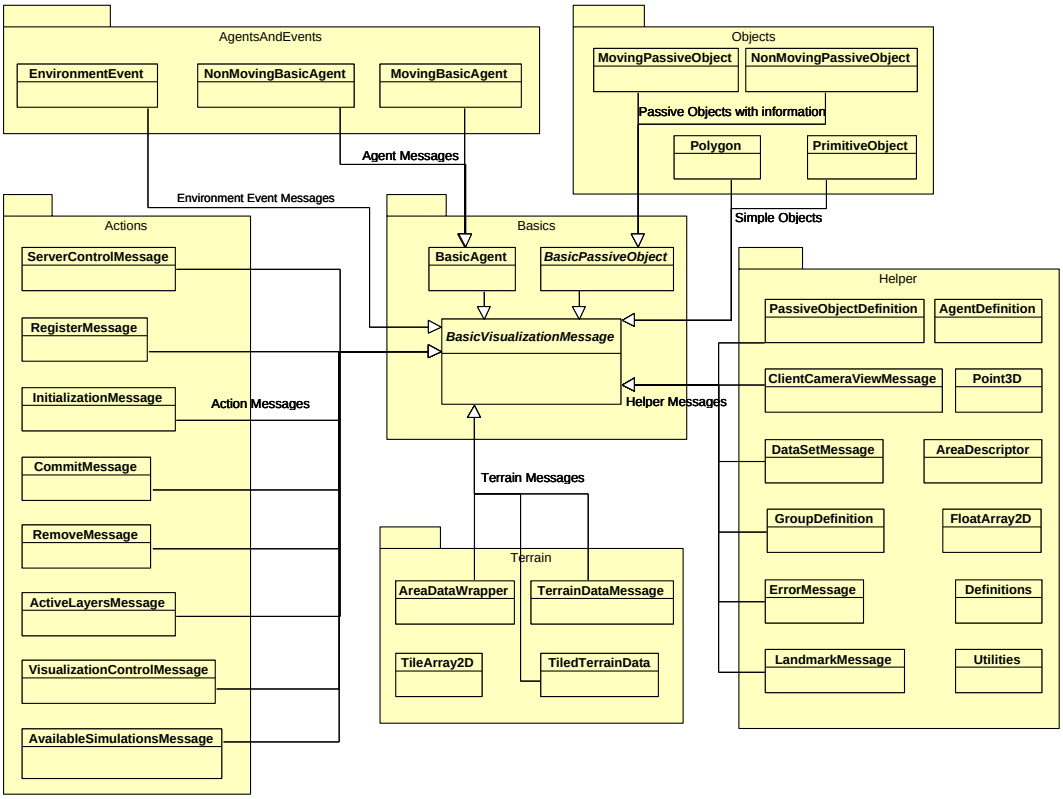


Abbildung 4.2: Klassenstruktur des Protokolls

4.6 Beispielhafte Nutzung

Zu Beginn erkundigt VIEW sich bei VIS über verfügbare Simulationen und wählt eine von diesen aus. Er erhält daraufhin Informationen über den Aufbau der Simulation. Sollten GIS-Daten benötigt werden, ruft der Client diese von GROUND oder, falls angegeben, einem anderen GIS-Dienst ab.

Mit diesen Informationen stellt VIEW die erste Ansicht zusammen. Außerdem wird VIEW bei VIS mit dem angefragten Ausschnitt und einer eindeutigen ID registriert, welche später u.a. für das Versenden von Updates genutzt wird.

Um eine unnötige Belastung des Netzwerkes zu vermeiden, werden dabei lediglich Informationen über die Agenten gesendet, die sich im Sichtbereich der Kamera befinden. Außerdem hat dies den Vorteil, dass VIEW keine überflüssigen Daten speichern muss, die womöglich zu einem Performance-Einbruch führen könnten.

Ändern sich Daten eines Agenten wie z.B. Laufrichtung, Geschwindigkeit oder Größe, werden alle Betrachter, die in dem betroffenen Bereich aktiv sind, darüber informiert. Gleiches gilt, wenn ein neuer Agent den Bereich betritt oder dort neu entsteht.

Benötigt ein Client keine Updates mehr, weil er beendet oder die Darstellung der Simulation angehalten wurde, sendet er eine Nachricht an den VIS, der daraufhin die beteiligten Knoten darüber informiert, dass keine weiteren Updates benötigt werden.

4.6.1 Inititalisierung

Da in MARS viele Simulationen parallel laufen können, muss VIS wissen, welche dieser Simulationen der Client darstellen möchte.

Der Client kann zunächst bei VIS anfragen, welche Simulationen zum Abruf zur Verfügung stehen. Daraufhin erhält er eine Liste der Simulationen.

Zum Start der Visualisierung benötigt der Client diverse Informationen. Diese werden in einer *InitializationMessage* übermittelt. Mit dieser Nachricht erhält der Client Informationen über:

- Agentendefinitionen
- Gruppendefinitionen
- Layer, die dargestellt werden können
- Verfügbare Heightmaps und Kartendaten
- Bei Bedarf ein alternativer GIS-Server
- Transaktionale oder direkte Darstellung
- Startzeitpunkt der Simulation und Taktung
- Orientierungspunkte

4.6.2 Terraingenerierung

Ausgehend von unserem Showcase verfügen wir über ein sehr großes Areal, das dargestellt werden soll. (Siehe auch 3.2, Punkt 7) Die meisten Engines haben mit einer solchen Dimension in mehreren Punkten Probleme. Unter anderem können die Heightmap und die Texturen, die die Karte später enthalten kann, nur eine begrenzte Größe haben.

Die Lösung besteht in der Unterteilung des Areals in Kacheln. Auf diese Weise könnten auch sehr große Areale gut gehandhabt werden und unter Umständen sogar ein Performancegewinn

erzielt werden, da Kacheln außerhalb des Sichtbereichs ausgeblendet werden können. Ebenso können Veränderungen im Terrain einfacher durchgeführt werden.

Umgesetzt wird diese Technik, indem der Client zunächst eine Nachricht vom Typ *TerrainDataMessage* erhält, der die grundlegenden Informationen zum darzustellenden Terrain enthält. Zu diesen Daten gehören unter anderem:

- Zahl der Kacheln entlang der jeweiligen Achsen
- Größe einer Kachel
- Auflösung der Heightmap
- Höchster Punkt der Karte[3]
- Koordinaten und Koordinatensystem, falls diese verwendet werden
- Position in der Höhenachse, um beispielsweise die Darstellung von Arealen mit mehreren Ebenen zu ermöglichen

Im Anschluss erhält der Client die Daten für die einzelnen Kacheln in *TiledTerrainData*-Nachrichten. Diese beinhalten unter anderem die Höheninformationen in Form eines zweidimensionalen Arrays und die Ebene, auf der die jeweilige Kachel liegen soll. Die *TiledTerrainData*-Nachrichten werden in einem *AreaDataWrapper* gesammelt versendet. Diese Entscheidung basiert auf den Messungen aus 6.2.

Die Protokollbibliothek liefert einige Methoden für die serverseitige Partitionierung des Geländes mit, um Kompatibilität zu gewährleisten. Es hat sich gezeigt, dass dieser Vorgang recht rechenintensiv ist und daher möglichst selten durchgeführt werden sollte.

Wurden Koordinaten angegeben, kann die Engine für die einzelnen Kacheln passende Kartendaten von einem GIS-Server herunterladen und als Textur verwenden. Bei der Konzeption wurde davon ausgegangen, dass es sich bei der Quelle um einen WMS-Server handelt und die entsprechenden Informationen über verfügbare Kartenlayer entsprechend angegeben werden.

4.6.3 Darstellung eines Kartenausschnittes

Die Menge der Daten, die an die Clients verteilt werden, soll möglichst gering sein, um sowohl Datenverkehr als auch den clientseitigen Verwaltungsaufwand zu minimieren. Der einfachste Ansatz dafür besteht darin, nur die Daten zu senden, die für den Client auch relevant sind.

[3] In Unity werden die Höhenwerte mit einem Wert zwischen 0 und 1 angegeben. Dieser wird mit dem höchsten Wert multipliziert und ergibt den entsprechenden Höhenwert für das Feld.

Dieses Verfahren orientiert sich an der Technik des *Frustum Culling*, die bei den meisten Engines bereits im Einsatz ist. Dabei werden nur die Objekte, die auch in Sichtweite der Kamera sind, gerendert.

Geraten Agenten oder Objekte aus dem Sichtbereich, werden sie gelöscht und geben so Speicherplatz frei. Der Client sendet dazu den Sichtbereich seiner Kamera, sobald sich deren Position geändert hat. Der Server sendet daraufhin eine Liste aller Agenten und Objekte zurück, die sich im darzustellenden Bereich befinden. Abhängig vom betrachteten Zeitpunkt bezeichnet Server hier entweder VIS oder die LIFE-Simulationsknoten.

Der Sichtbereich wird ermittelt, indem ausgehend von der Kameraposition eine Pyramide aufgebaut wird. Die Koordinaten, an denen die Strahlen auf den Boden treffen, stellen die Eckpunkte für die Grenzen des Sichtbereiches dar. Alle Agenten und Objekte, die sich innerhalb des Sichtbereiches befinden, müssen an den Client übertragen werden. Dazu zählen auch Objekte, die nur teilweise sichtbar sind. In Abb. 4.4 ist dies beispielhaft gezeigt. Grüne und gelbe Objekte würden übertragen werden, graue hingegen nicht.

Schwierig wird dieser Prozess, wenn es sich um eine Simulation handelt, die nicht nur Agenten und Objekte auf dem Boden kennt, sondern auch in der Luft, wie z.B. Vögel oder Flugzeuge.
Die Nachricht (*ClientCameraViewMessage*) über den Sichtbereich enthält:

- die Kameraposition
- die Eckkoordinaten des Pyramidenfußes
- den Bildwinkel der Kamera auf der Y-Achse
- die maximale Sichtweite der Kamera

Das aktuelle Konzept sieht nicht vor, dass permanent Daten während der Kamerabewegung nachgeladen werden, sondern gewartet wird, bis die Kamera zum Stillstand gekommen ist.

Wählt der Benutzer Layer zur Darstellung an oder ab, wird VIS mit einer *ActiveLayersMessage* darüber informiert. Die Information wird an die entsprechenden Simulationsknoten weitergeleitet, sodass diese vorerst keine Daten mehr senden, bis sie in der Visualisierung wieder aktiviert werden.

4.6.4 Empfang und Verarbeitung von Updates

Da der Server die Kamerapositionen aller Clients kennt, kann er bei Updates feststellen, welche Agenten und Objekte dieser erhalten muss. Dies können sowohl Datenupdates als auch

komplett neue Instanzen sein, die im aktuellen Sichtbereich des Clients auftauchen. Das kann beispielsweise geschehen, wenn sie in der Simulation in diesem Bereich erzeugt wurden oder sich dorthin bewegt haben.

Agenten und Objekte, die aus der Simulation gelöscht wurden, werden mit einer *RemoveMessage* entfernt. Im Bedarfsfall kann hier auch eine Nachricht mitgegeben werden.

Je nachdem, ob die Darstellung transaktional ist oder nicht, wartet der Client mit dem Update, bis er von VIS eine *CommitMessage* mit dem entsprechenden Tick bekommen hat. Zuvor muss VIS von allen Simulationsknoten eine entsprechende *CommitMessage* für den betreffenden Client erhalten haben.

Basierend auf den Messungen aus 6.2 erhalten die Clients die Updates in einer *DataSetMessage* verpackt. Diese enthält:

- bewegliche und nicht-bewegliche Agenten
- bewegliche und nicht-bewegliche passive Objekte
- Umweltevents, wie beispielsweise Feuer
- Informationen über Agenten und Objekte, die seit dem letzten Tick entfernt wurden

4.6.5 Abmeldung

Wird der Client beendet, so meldet er sich mit einer *ServerControlMessage* mit dem entsprechenden Befehl bei VIS ab. VIS leitet die Abmeldung an die beteiligten Knoten weiter, sodass diese keine weiteren Updates an den Client verschicken.

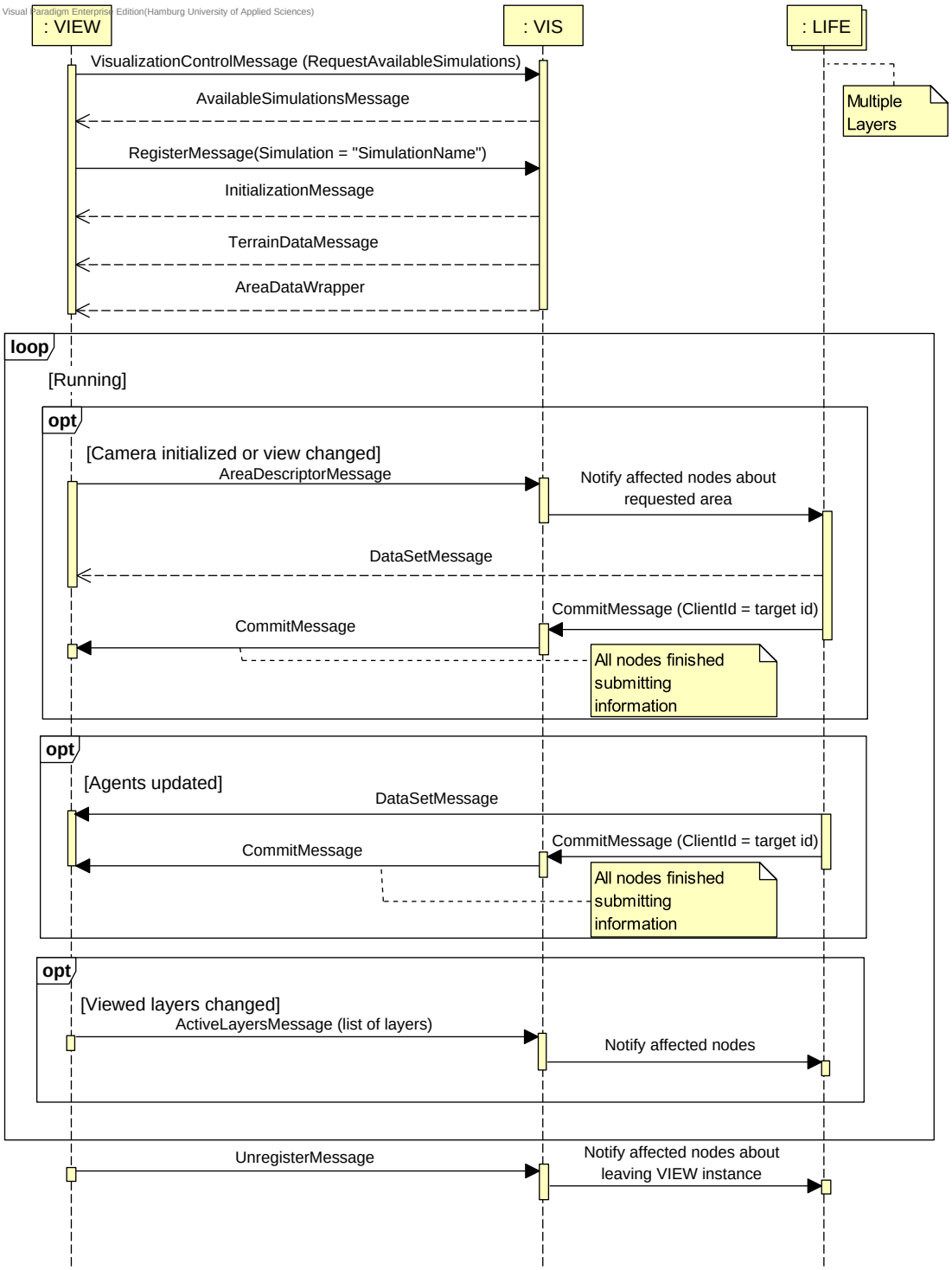
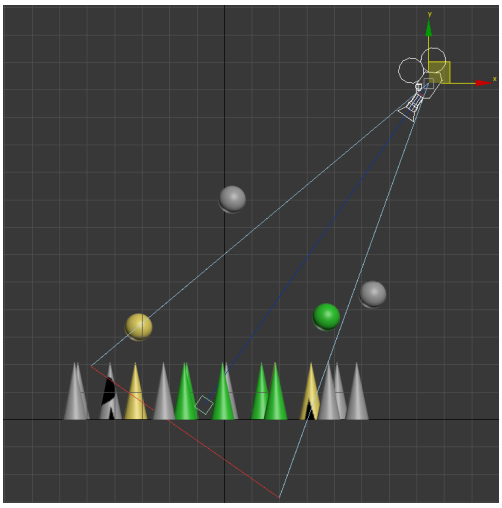
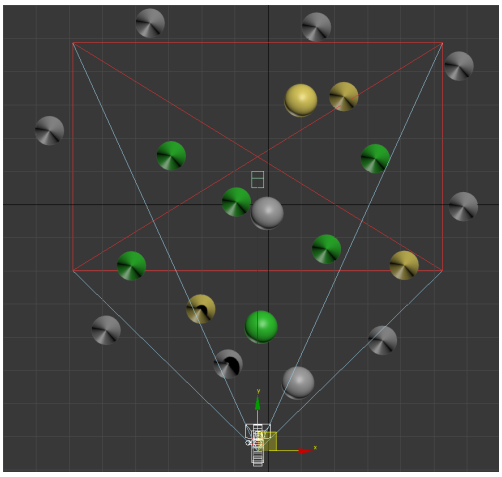


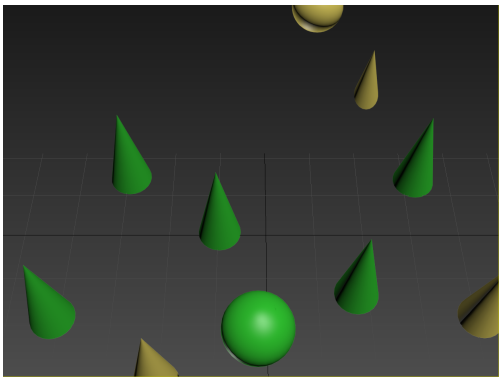
Abbildung 4.3: Beispielhafte Kommunikation der Visualisierung



(a) Seitliche Ansicht



(b) Ansicht von oben



(c) Kameraperspektive

Abbildung 4.4: Beispiel für Frustum Culling. Komplett im Sichtkegel befindliche Objekte sind grün. Teilweise im Blickfeld befindliche Objekte in gelb, nicht sichtbare grau.

5 Visualisierung

Da zur Überprüfung und Ideensammlung auch eine Beispielimplementierung notwendig wurde, soll auch deren Umsetzung beschrieben werden.

5.1 Wahl der Engine

Wie schon in Kapitel 4 erwähnt, wurde die Unity-3D-Engine (kurz: Unity) für die Implementierung verwendet. Dementsprechend sind die in diesem Kapitel beschriebenen Probleme und deren Lösungen nicht immer allgemeingültig.

Aus der Vielzahl der verfügbaren Engines wurde Unity ausgewählt. Ein Hauptkriterium war die genutzte Programmiersprache. Unity unterstützt die Programmiersprachen C# bzw. Mono von Haus aus. Damit benutzt die Visualisierungskomponente die gleiche Sprache wie der Rest des Projektes.

Bei vielen Engines scheint es zudem ein Problem zu sein, zur Laufzeit das Gelände zu modifizieren. Lösungen hierfür waren entweder nicht zu finden oder waren wesentlich komplexer als in Unity.

Im Vergleich zu vielen anderen Engines verfügt Unity zudem über eine ausführliche Dokumentation und eine aktive Community. Dies sollte die Implementierung, Wartung und Zusammenarbeit vereinfachen.

Zum Zeitpunkt der Entscheidungsfindung erschien Unity damit als am besten geeignet, um die Visualisierung umzusetzen.

Da Unity in der neuesten Version Projekte auch in HTML5 kompilieren kann, sollte die Implementierung auch in die MARS WebSuite eingebunden werden. Es stellte sich heraus, dass die notwendigen DLL-Dateien, die für das Protokoll und den Zugriff auf die Queue nötig sind, in HTML5-Projekten nicht verwendbar sind. Eine Umsetzung mit Unity in HTML5 ist daher nicht ohne weiteres möglich.

5.2 Abruf und Darstellung von GIS-Daten

Da es zu den Anforderungen gehörte, GIS-Daten auf dem Gelände darstellen zu können, muss die Engine fähig sein, eine Verbindung mit einem entsprechenden Dienst aufbauen zu können. Nach der Untersuchung diverser Dienste bzw. Protokolle wurde deutlich, dass die Verwendung eines GIS-Servers mit dem *Web Map Service* (WMS) sinnvoll ist. Bei dieser Schnittstelle kann ein Kartenausschnitt anhand diverser Koordinatensysteme definiert und in beliebiger Größe als Grafik abgerufen werden. Dies wird unter Umständen durch Serverkonfigurationen, zwecks Auslastungsbegrenzung, beschränkt. Häufig sind die Karten in mehrere Layer unterteilt, die beliebig kombiniert werden können.

Der WMS wird über eine zusammengesetzte URL angesprochen. Diese enthält unter anderem die darzustellenden Layer, die Koordinaten und das Koordinatensystem. Der GIS-Server erstellt daraufhin das Bild für den angeforderten Bereich. An dieser Stelle wird ein Problem deutlich, das den Einsatz von WMS einschränkt: Der Betrieb von Servern, die WMS unterstützen, ist recht ressourcenintensiv, da sie für jede Anfrage neue Bilder rendern müssen. Aus diesem Grund sind die meisten Server, die diesen Dienst bereitstellen, kostenpflichtig oder stellen nur sehr spezialisierte Daten zur Verfügung. Vorteilhaft ist daher die Pflege eines eigenen Servers, der alle Daten zur Verfügung stellt, die benötigt werden.

Das erstellte Bild wird als Textur auf dem Terrain hinzugefügt. Die maximale Größe der einzelnen Texturen werden sowohl durch die Hardware als auch durch die 3D-Engine beschränkt.

Die meisten heute verfügbaren Dienste wie *Google Maps*, *Bing Maps* oder *OpenStreetMaps* nutzen aktuellere Protokolle wie *Tile Map Service*, bei denen bereits in verschiedenen Zoomstufen gerenderte Bilder in einer Datenbank liegen und abgerufen werden. Dieser Dienst ist jedoch ungeeignet, da sich keine Koordinaten für einen Kartenausschnitt (*Bounding Box*) angeben lassen.

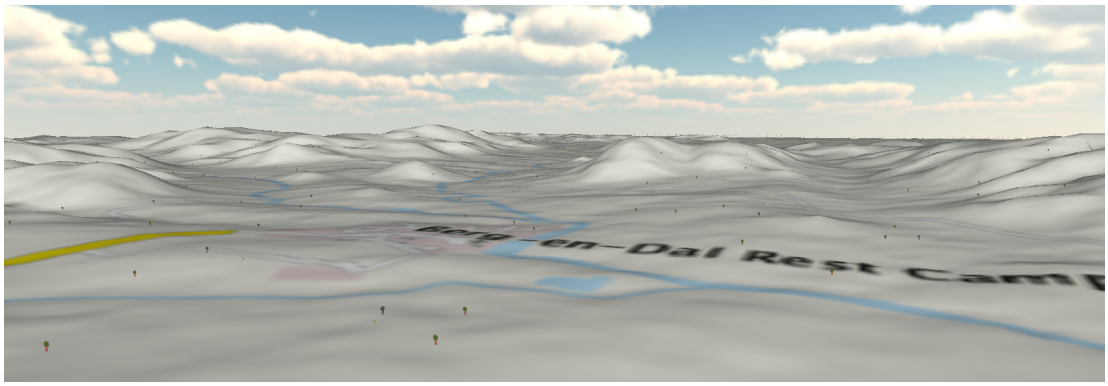


Abbildung 5.1: Ansicht des *Berg-en-Dal Rest Camp* umgeben von zufallsgenerierten Agenten

5.3 Terrain

Um auch bei großem Terrain noch eine akzeptable Performance zu haben, kann dieses nicht aus einem Stück bestehen. Ebenso gibt es eine maximale Größe für die aufgelegten Texturen, die derzeit bei 4096x4069 Pixeln liegt. Bei älteren Grafikkarten liegt das Maximum unter Umständen niedriger. Würde man ein solches Bild auf ein größeres Terrain legen, würde die Textur unansehnlich und z.B. Namen nicht mehr leserlich dargestellt werden.

Wie bereits in 4.6.2 beschrieben, wird das Terrain zur Lösung dieses Problems in Kacheln unterteilt.

Die Ladezeiten der Texturen sind derzeit noch recht hoch, da sie jedesmal vom GIS-Server geladen werden. Ausgehend davon, dass ein Terrain bei jedem Durchlauf gleich aufgeteilt wird, könnte man hier durch Caching der Texturen Zeit sparen.

5.4 Dynamisches Laden von Objekten

In Unity können Objekte aus einem entsprechenden Ordner bei Bedarf nachgeladen werden. Dies kommt bei der Zuordnung von 3D-Modellen zu Agenten und Objekten zum Einsatz.

Der Modellierer muss bei der Definition eines Agenten oder eines Objektes, die in der Visualisierung vorkommen sollen, den Pfad und den Dateinamen angeben. Mit dieser Information kann die Engine das entsprechende Modell laden und der zuvor mit Kategorie und Attributen festgelegten Definition zuordnen. (Siehe auch 4.5)

Das 3D-Modell muss vor der Verwendung einmalig mittels Unity in ein sogenanntes *Prefab* umgewandelt werden. Dies kann auch mit der kostenfreien Version geschehen.

5.5 Agenten und Objekte

Da die Performance der Engine mit zunehmender Objektzahl sinkt, ist dies ein wichtiger Ansatzpunkt zur Verbesserung der Leistung. Sobald ein Objekt den aktuellen Kameraausschnitt verlässt, wird es deshalb aus der Visualisierung gelöscht.

Häufig werden die Modelle in solchen Fällen wiederverwendet. Dabei werden sie in einem Pool von inaktiven Modellen abgelegt, der die Ressourcen schont und eine Neuinstanziierung überflüssig macht. Die Beispielvisualisierung verfügt derzeit nicht über einen solchen Pool.

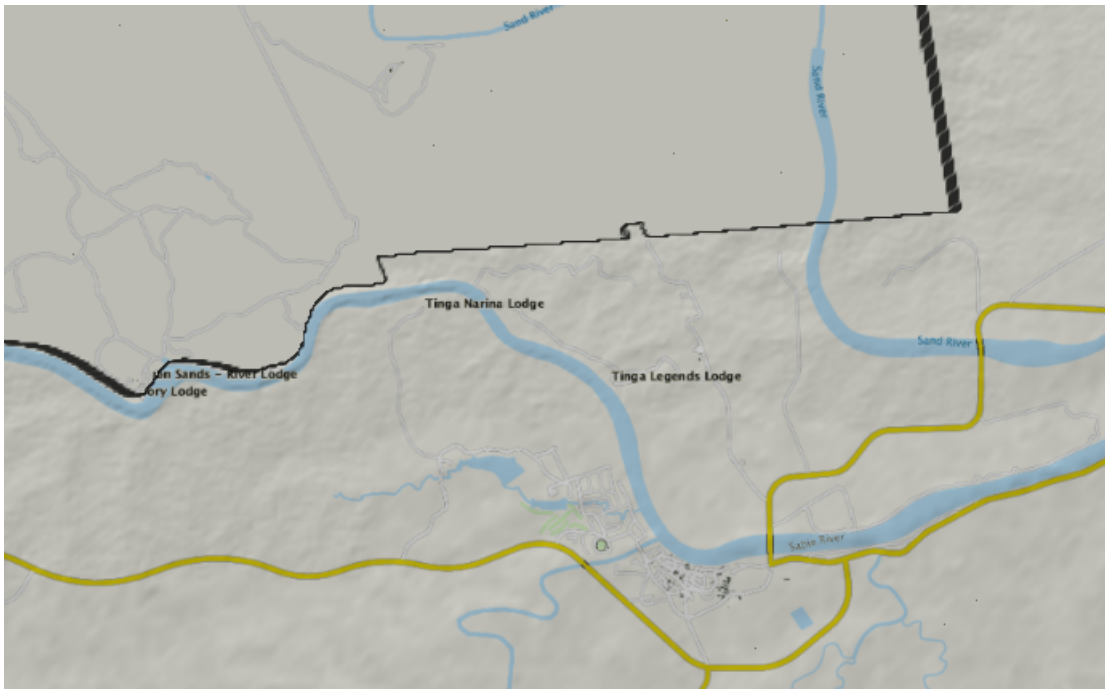


Abbildung 5.2: *Tinga Legends Loge* und Umgebung aus der Vogelperspektive

Sowohl in Abb. 5.1 als auch in Abb. 5.2 befinden sich zufallsgenerierte Agenten. Während sie im ersten Bild noch zu erkennen sind, sind sie auf dem zweiten Bild nicht mehr zu sehen, da sie zu weit entfernt sind. Bei solch großen Entfernungen könnte es hilfreich sein, Agenten so zu gruppieren, dass sie erkennbar bleiben. Dabei muss jedoch ersichtlich sein, dass es sich um eine Gruppe handelt. Denkbar wäre hier ein stark vergrößertes Agentensymbol oder ein speziell gekennzeichnetes Symbol, das sich vom ursprünglichen Symbol für den Agenten ableitet. In der Kartografie wird ein vergleichbares Verfahren unter dem Begriff *Generalisierung*[1] angewandt. Berner u. a. (1998) und Campo, Orosco und Teyseyre (1997) stellen dafür Algorithmen vor, die darauf basieren, die einzelnen Bestandteile hierarchisch anzuordnen. Damit soll ein dynamisches Zusammenfassen der Objekte zu einer übergeordneten Gruppe ermöglicht werden.

[1] https://de.wikipedia.org/wiki/Generalisierung_(Kartografie) (14.08.2015)

5.6 Bedienung

Zu Beginn der Entwicklung war die Bedienung an das Bedienungsschema von Spielen angelehnt. Die Kamera wird mit der Tastatur bewegt, der Blickwinkel mit der Maus verändert und bei Bedarf gezoomt.

Diese Bedienung bietet viele Freiheiten, jedoch hat sich bei Versuchen mit weniger fachkundigen Anwendern gezeigt, dass dies leicht dazu führen kann, dass sich der Nutzer "verirrt".

Das Problem wurde gelöst, indem zusätzlich zu einer frei beweglichen Kamera eine feste Ansicht aus der Vogelperspektive eingebaut wurde. Diese kann, ähnlich wie in einer Kartensoftware, wie beispielsweise Google Maps, mit Drag & Drop bedient werden. Besteht weiterhin der Bedarf einer Verbesserung der frei beweglichen Kamera, könnte diese in den Achsen beschränkt werden und die Bewegung über die Tastatur ebenfalls durch Drag & Drop ersetzen.

Bei großen Gebieten wie dem KNP wird das Navigieren ohne Anhaltspunkte relativ mühsam, da die Darstellung derzeit im Maßstab 1:1 geschieht.[2] Als hilfreich könnten sich, neben der bereits vorhandenen Koordinatenangabe, beispielsweise eine Liste von "Points of Interest", wie zum Beispiel Tränken, Camps oder große Agentenansammlungen, anbieten. Diese könnten vom Simulationsentwickler zu Beginn definiert werden oder abhängig vom aktuellen Geschehen in der Simulation entstehen.

Die GIS-Daten können insbesondere bei großen Geländeausschnitten, bei denen weit herausgezoomt werden muss, nur bedingt weiterhelfen. Durch die Technik der Engine wird ab einer gewissen Entfernung die Qualität der einzelnen Texturen reduziert, sodass sich beispielsweise Namen nicht mehr lesen lassen. Gröbere Orientierungspunkte wie Flüsse oder Seen sind noch zu erkennen und können bei der Orientierung helfen.

[2] In Unity entspricht eine Längeneinheit einem Meter.

6 Diskussion

In diesem Abschnitt werden folgende Punkte der zuvor vorgestellten Lösung genauer untersucht:

- Die Umsetzung des Prototypen, bestehend aus Visualisierung und Protokoll
- Die Vollständigkeit der Umsetzung
- Die Performance des gewählten Serialisierers

6.1 Umsetzung des Prototypen

Die Bachelorarbeit und die damit verbundene Softwareimplementierung ist Teil eines großen Projektes, das sich selbst noch in Entwicklung befindet und Veränderungen unterworfen ist. Daher beruhen einige Entwicklungsschritte auf Annahmen (siehe 4.2) oder konnten nur teilweise fertiggestellt werden. Hier soll zusammengefasst werden, welche Unterschiede es zwischen Entwurf und Implementierung gibt.

6.1.1 Visualisierung

Da MARS selbst zur Entwicklungszeit noch nicht lauffähig war, musste ein Datengenerator geschrieben werden. Dieser ist verhältnismäßig einfach aufgebaut. So sind zum Beispiel Funktionen wie das Vor- und Zurückspulen nicht implementiert. Ebenso beherrscht das System noch nicht das Generieren von Polygonen oder das Einfärben nach Gruppen.

Andere, recht komplexe Aufgaben sind jedoch umgesetzt. Dazu zählen zum Beispiel das Abrufen von GIS-Daten als Textur oder das Laden von Objekten zur Laufzeit und deren Auswahl nach Attributen (siehe 4.5). Die nicht implementierten Funktionen sind laut Recherchen jedoch in der Engine umsetzbar.

Bisher wurde auch auf die Implementierung einer vollwertigen GUI verzichtet. Es ist bisher möglich, Koordinaten aufzurufen und zwischen Vogelperspektive und einer frei beweglichen Kamera zu wechseln. Momentan macht die Visualisierung keinen Gebrauch von Animationen.

Es ist fraglich, wie wichtig und nützlich deren Verwendung wäre oder ob sie sich sogar eher negativ auswirken würde.

Verglichen mit dem von Shneiderman (1996) entwickelten "Visual Information Seeking Mantra" (siehe 2.2) fehlen der aktuellen Version der Visualisierungskomponente einige Features.

Die in 5.5 erwähnte Generalisierung der Agenten, die als Teil des Mantras betrachtet werden kann, ist beispielsweise nicht umgesetzt. In wie weit die Attribute, die dem Agenten mitgegeben werden können, für eine Umsetzung reichen, müsste erprobt werden. Unter Umständen ist hier eine Erweiterung im Protokoll nötig.

6.1.2 Protokoll

Die Implementierung des Protokolls ist, gemäß des beschriebenen Konzeptes, erfolgt. Die Funktionsfähigkeit wurde anhand der zuvor genannten Visualisierung beschrieben. Daraus folgt, dass beispielsweise das Erstellen von Polygonen nicht in der Praxis erprobt werden konnte.

6.2 Performance

Um ein Gefühl für den Zeitbedarf und die Effizienz der Serialisierer bei der entworfenen Datenstruktur zu bekommen, wird eine Performancemessung durchgeführt. Dabei werden mehrere Messungen mit steigender Agentenzahl durchlaufen. Gleichzeitig wird auch untersucht, ob es effizienter ist, die Agenten in einer einzelnen Nachricht zu verpacken oder einzeln zu serialisieren.

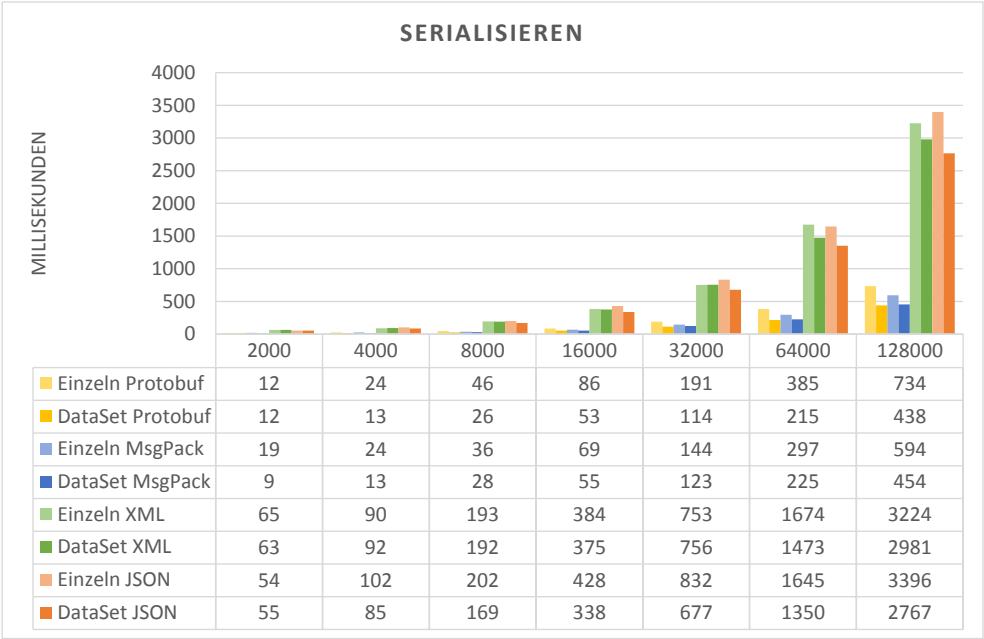
Die Messdaten aus 6.1 und 6.2 zeigen, dass nur *MessagePack* und *Protocol Buffers* auch bei einer großen Anzahl von Agenten effizient arbeiten.

Die Messungen verdeutlichen außerdem, dass keine eindeutige Aussage darüber getroffen werden kann, ob die Daten gesammelt oder einzeln verschickt werden sollten. Die Entscheidung hängt im wesentlichen davon ab, ob die Daten parallel verarbeitet werden können oder nicht. Bei der sequenziellen Verarbeitung steigt der Zeitbedarf bei einer großen Agentenzahl sehr stark an. In diesem Fall ist es sinnvoll, die Daten zu sammeln und als Gesamtes zu verschicken. Kann der Empfänger die Daten jedoch parallel deserialisieren, ist ein deutlicher Performancegewinn möglich. In der Beispielimplementierung wird davon ausgegangen, dass die Daten gesammelt versendet werden.

Nach der Serialisierung ist nicht mehr direkt erkennbar, um welchen Nachrichtentyp es sich handelt. Dies muss jedoch für eine korrekte Deserialisierung bekannt sein. Als Behelf wird in einer Variablen in der Basisklasse (*BasicVisualizationMessage*) der Nachrichtentyp festgehalten.

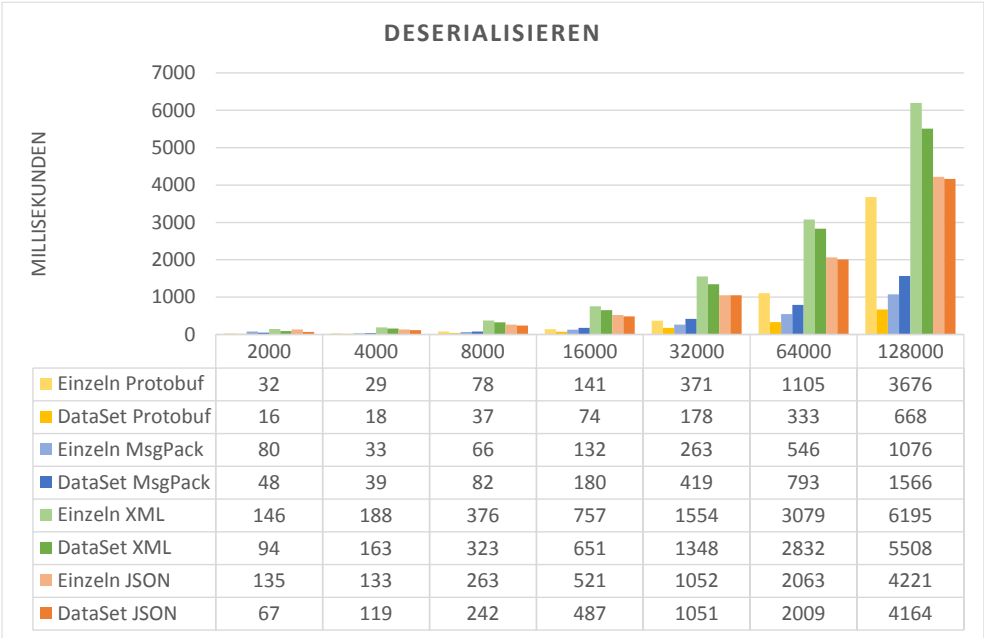
Da alle Nachrichten von dieser Klasse erben, können sie auch als diese deserialisiert werden. Im Anschluss kann der Typ der Nachricht ausgelesen werden, sodass diese mit dem korrekten Typ ein weiteres Mal deserialisiert werden kann und alle Variablen lesbar sind.

Einige Queues unterstützen ein weiteres Feld in dem der Typ der Nachricht als String angegeben werden kann. Wenn diese Möglichkeit zur Verfügung steht, lässt sich ein zweifaches deserialisieren vermeiden.



	2000	4000	8000	16000	32000	64000	128000
Einzeln Protobuf	12	24	46	86	191	385	734
DataSet Protobuf	12	13	26	53	114	215	438
Einzeln MsgPack	19	24	36	69	144	297	594
DataSet MsgPack	9	13	28	55	123	225	454
Einzeln XML	65	90	193	384	753	1674	3224
DataSet XML	63	92	192	375	756	1473	2981
Einzeln JSON	54	102	202	428	832	1645	3396
DataSet JSON	55	85	169	338	677	1350	2767

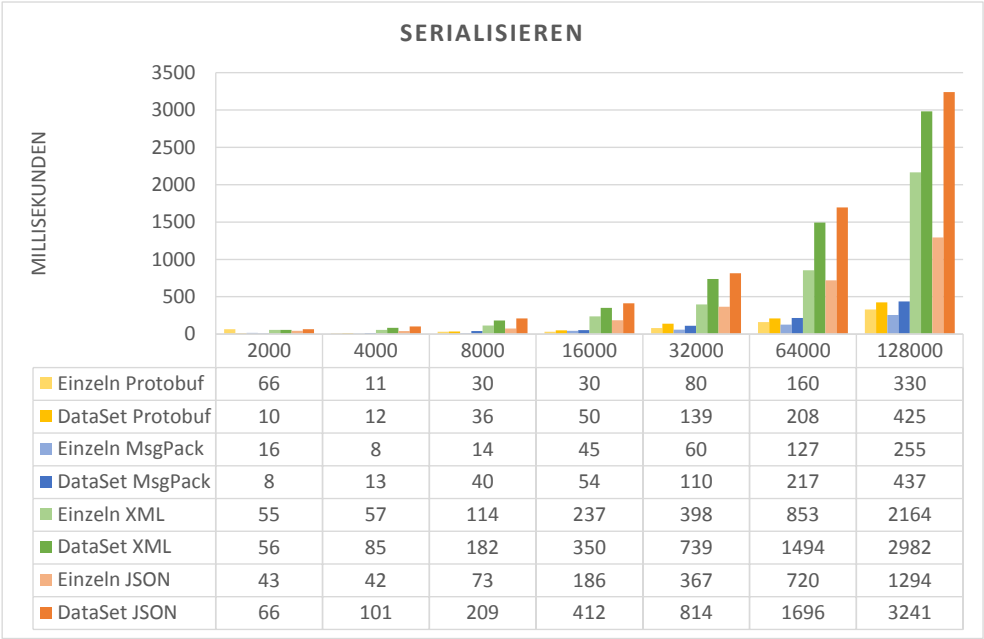
(a) Serialisierung



	2000	4000	8000	16000	32000	64000	128000
Einzeln Protobuf	32	29	78	141	371	1105	3676
DataSet Protobuf	16	18	37	74	178	333	668
Einzeln MsgPack	80	33	66	132	263	546	1076
DataSet MsgPack	48	39	82	180	419	793	1566
Einzeln XML	146	188	376	757	1554	3079	6195
DataSet XML	94	163	323	651	1348	2832	5508
Einzeln JSON	135	133	263	521	1052	2063	4221
DataSet JSON	67	119	242	487	1051	2009	4164

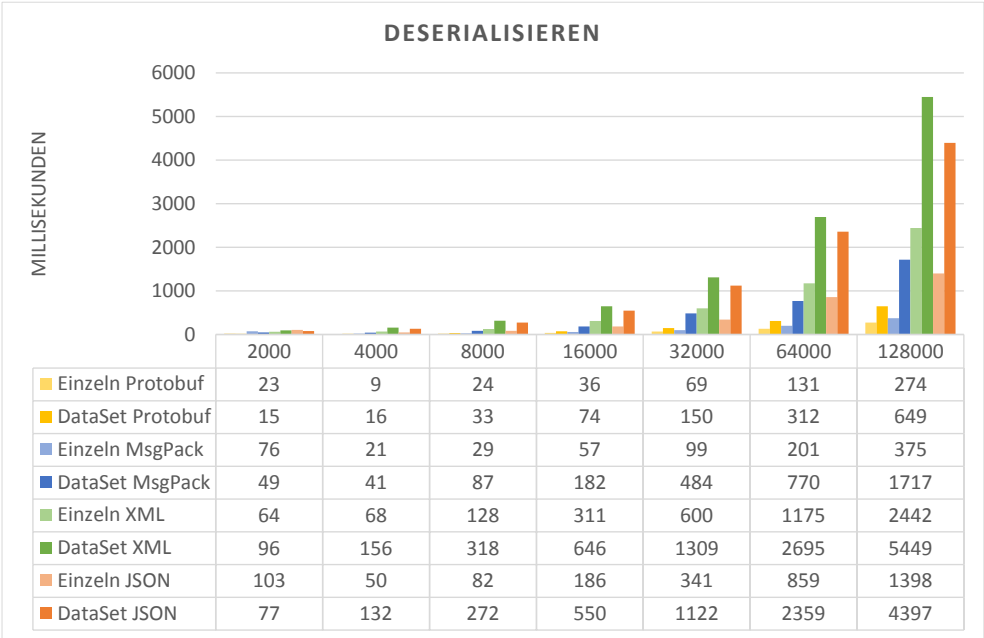
(b) Deserialisierung

Abbildung 6.1: Benchmark der Serialisierer mit sequenzieller Verarbeitung



	2000	4000	8000	16000	32000	64000	128000
Einzeln Protobuf	66	11	30	30	80	160	330
DataSet Protobuf	10	12	36	50	139	208	425
Einzeln MsgPack	16	8	14	45	60	127	255
DataSet MsgPack	8	13	40	54	110	217	437
Einzeln XML	55	57	114	237	398	853	2164
DataSet XML	56	85	182	350	739	1494	2982
Einzeln JSON	43	42	73	186	367	720	1294
DataSet JSON	66	101	209	412	814	1696	3241

(a) Serialisierung



	2000	4000	8000	16000	32000	64000	128000
Einzeln Protobuf	23	9	24	36	69	131	274
DataSet Protobuf	15	16	33	74	150	312	649
Einzeln MsgPack	76	21	29	57	99	201	375
DataSet MsgPack	49	41	87	182	484	770	1717
Einzeln XML	64	68	128	311	600	1175	2442
DataSet XML	96	156	318	646	1309	2695	5449
Einzeln JSON	103	50	82	186	341	859	1398
DataSet JSON	77	132	272	550	1122	2359	4397

(b) Deserialisierung

Abbildung 6.2: Benchmark der Serialisierer mit paralleler Verarbeitung

7 Zusammenfassung und Ausblick

Im abschließenden Kapitel wird ein zusammenfassender Rückblick auf die Arbeit gegeben. Außerdem werden Ansatzpunkte für mögliche Verbesserungen genannt.

7.1 Fazit

Ziel dieser Arbeit ist es, ein Kommunikationsprotokoll zu entwerfen, mit dem eine, in einem verteilten System laufende, Multiagentensimulation visualisiert werden kann.

Nach Einführung in die Grundlagen der Multiagentensimulationen wurden Anforderungen und aktuell existierende Systeme in Hinsicht auf Flexibilität, Geschwindigkeit und Nutzerfreundlichkeit untersucht. Das daraus entstandene Konzept deckt die Bedürfnisse bestmöglich ab.

Das entwickelte Protokoll ermöglicht es, Simulationen mit beliebigem Terrain und beliebigen Agentendefinitionen und daraus resultierenden Agentenmodellen zu visualisieren. Ebenso wird es ermöglicht, die Simulation zu einem beliebigen Zeitpunkt zu betrachten oder bereits abgeschlossene Simulationen anzuzeigen.

Die im Zuge der Implementierung entstandene Visualisierungskomponente wurde in Verbindung mit einem Datengenerator getestet. Dabei lieferte sie Erkenntnisse für die Umsetzung einer vollständig integrierten Visualisierung. Dazu zählen sowohl technische Anforderungen, wie zum Beispiel eine Terrainaufteilung, als auch die Bedienung durch den Nutzer.

7.2 Technische Verbesserungen und Erweiterungen

Während der Entwicklung haben sich einige Ansatzpunkte zur Erweiterung im Gesamtsystem ergeben, die hier kurz beschrieben werden sollen.

7.2.1 Die VIS-Komponente

Zum Zeitpunkt der Konzeption des Protokolls existierte VIS nur als grobes Konzept. Sie soll die Schnittstelle zwischen Visualisierung und Simulation bereitstellen. Gleichzeitig soll sie den

Abruf von Daten aus der Vergangenheit ermöglichen. Dazu wird auf die Simulationsdatenbank zugegriffen und die Daten werden so zusammengesetzt, dass sie für die Visualisierung verwendbar sind. Der Client kann über Nachrichten beliebig in der Simulation vor- und zurückspringen.

7.2.2 Interaktivität

Die Interaktionsmöglichkeiten sind derzeit noch darauf beschränkt, die Kameraposition zu ändern, Layer ein und auszublenden und die einzelnen Agenten zu markieren, um Informationen zu erhalten. Eine Veränderung der Simulation zur Laufzeit ist momentan nicht vorgesehen. Insbesondere eine Beeinflussung der Simulation aus der Visualisierung heraus könnte jedoch sehr interessant sein und birgt Potential zur Weiterentwicklung.

7.2.3 Optimierung der Kommunikation

Es ist davon auszugehen, dass mit einer wachsenden Anzahl von Agenten auch die Belastung des Netzwerkes steigt. Da die Kommunikation der Agenten und Layer untereinander eine höhere Priorität hat als die Visualisierung, gibt es hier Möglichkeiten zur Verbesserung. Dazu könnte man zum Beispiel beobachten, ob sich die Sichtfelder von Clients überlappen. Ist dies der Fall, könnten die verantwortlichen Server Updates mit einer Nachricht in mehrere Queues einreihen, statt die sie mehrfach zu senden. Außerdem könnte man das allgemeine Protokoll von MARS mit einer Paketpriorisierung versehen und so wichtigen Daten Vorrang einräumen.

7.2.4 Aufteilung des Terrains

Bei besonders großen Gebieten kann es trotz Unterteilung in Kacheln zu Problemen kommen. Grund dafür ist, dass irgendwann auch hier eine Zahl an Kacheln erreicht ist, die die Visualisierung zu stark fordert, selbst wenn sie gerade nicht angezeigt werden. Ebenso wächst mit einer zunehmenden Größe des dargestellten Geländes auch die Unübersichtlichkeit. Als Lösung wäre eine Unterteilung des Gebietes in Regionen denkbar. Ob diese während der Kamerabewegung dynamisch nachgeladen werden können oder von Hand zwischen ihnen umgeschaltet werden muss, müsste dabei noch getestet werden.

Literatur

Amouroux, Edouard u. a. (2009). “GAMA: An Environment for Implementing and Running Spatially Explicit Multi-agent Simulations”. In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*. Bd. 5044 LNAI, S. 359–371.

Barella, A. u. a. (2007). “Multi-agent systems applied to virtual environments: a case study”. In: *Proc. 2007 ACM Symp. Virtual Real. Softw. Technol. - VRST ’07*. Bd. 1. 212, S. 237.

Berner, S. u. a. (1998). “A visualization concept for hierarchical object models”. In: *Proc. 13th IEEE Int. Conf. Autom. Softw. Eng. (Cat. No.98EX239)*, S. 225–228.

Campo, M., R. Orosco und a. Teyseyre (1997). “Interactive abstraction control in visualization systems”. In: *Proc. 17th Int. Conf. Chil. Comput. Sci. Soc.* 7000, S. 29–37.

Chen, Kuan-Ta u. a. (2011). “Measuring the latency of cloud gaming systems”. In: *Proc. 19th ACM Int. Conf. Multimed. - MM ’11*, S. 1269.

Claypool, Mark u. a. (2012). “Thin to win? Network performance analysis of the OnLive thin client game system”. In: *Annu. Work. Netw. Syst. Support Games*, S. 1.

Dahmann, Judith, Richard Fujimoto und Richard Weatherly (1997). “The Department of Defense High Level Architecture”. In: *Proc. 1997 Winter Simul. Conf.* S. 142–149.

Deussen, Oliver (2003). “Nicht-Fotorealismus : Neue Darstellungsformen in der Computergraphik”. In: *Inform. 2003, Fachwissenschaftlicher Inform. Novemb. 2003, Bad Schussenried* November 2003, S. 1–4.

Epic Games (2012). *Unreal Networking Architecture*. URL: `http://udn.epicgames.com/Three/NetworkingOverview.html` (besucht am 21. 08. 2014).

Feddersen, Christian (2010). *Repositionierung von Marken: ein agentenbasiertes Simulationsmodell zur Prognose der Wirkungen von Repositionierungsstrategien*. Gabler Research.

Fiedler, Glenn (2010). *What every programmer needs to know about game networking*. URL: `http://gafferongames.com/2010/01/24/what-every-programmer-needs-to-know-about-game-networking-3/` (besucht am 20. 08. 2014).

Gaikai Inc. (2015). *Gaikai*. URL: `https://www.gaikai.com/` (besucht am 07. 04. 2015).

Görz, Günter, Josef Schneeberger und Ute Schmid (2013). *Handbuch der Künstlichen Intelligenz*. Hrsg. von Günther Görz, Josef Schneeberger und Ute Schmid. 5. Aufl. DE GRUYTER.

Helm, Chantal Vinisia, Gwyneth Wilson u. a. (2011). “Investigating the vulnerability of an African savanna tree (Sclerocarya birrea ssp. caffra) to fire and herbivory”. In: *Austral Ecol.* 36.8, S. 964–973.

Helm, Chantal Vinisia, E. T F Witkowski u. a. (2009). “Mortality and utilisation of Sclerocarya birrea subsp. Caffra between 2001 and 2008 in the Kruger National Park, South Africa”. In: *South African J. Bot.* 75.3, S. 475–484.

Henderson, Tristan und Saleem Bhatti (2003). “Networked games - a QoS-sensitive application for QoS-insensitive users?” In: *Proc. ACM SIGCOMM Work. Revisiting IP QoS What have we Learn. why do we care? - RIPQoS ’03* August, S. 141–147.

Hernández, Josefa Z., Sascha Ossowski und Ana Garcia-Serrano (2002). “Multiagent architectures for intelligent traffic management systems”. In: *Transp. Res. Part C Emerg. Technol.* 10.5-6, S. 473–506.

Huang, Chun-Ying u. a. (2013). “GamingAnywhere”. In: *Proc. 4th ACM Multimed. Syst. Conf. - MMSys ’13*, S. 36–47.

Hüning, Christian (2013). “Konzeption und Entwurf einer Architektur zum Einsatz von Multi-Agenten-Simulationen in der ökologischen Systemmodellierung”. Bachelorarbeit. Hochschule für Angewandte Wissenschaften Hamburg.

Hüning, Christian u. a. (2014). “MARS – A next - gen multi - agent simulation framework”. In: 2008, S. 1–14.

Huttner, Sebastian u. a. (2009). “Strategies for Mitigating Thermal Heat Stress in Central European Cities : the Project Klimes”. In: *seventh Int. Conf. Urban Clim.* Bd. 49. July, S. 5.

Kagaya, Seiichi u. a. (2005). “An Application of Multi-agent Simulation to Traffic Behavior for Evacuation in Earthquake Disaster”. In: *J. East. Asia Soc. Transp. Stud.* 6, S. 4224–4236.

Keim, Daniel A. (2002). “Datenvisualisierung und data mining”. In: *Datenbank-Spektrum*, S. 1–22.

Klügl, Franziska (2006). “Multiagentensimulation”. In: *Informatik-Spektrum* 29.6, S. 412–415.

Lämmel, Gregor (2011). “Escaping the Tsunami : Evacuation Strategies for Large Urban Areas. Concepts and Implementation of a Multi-Agent Based Approach”. In: S. 1–149.

Lehmann, Dirk J. u. a. (2010). “Visualisierung und Analyse multidimensionaler Datensätze”. In: *Informatik-Spektrum* 33.6, S. 589–600.

Lozano, M. u. a. (2009). “A new system architecture for crowd simulation”. In: *J. Netw. Comput. Appl.* 32.2, S. 474–482.

Mengistu, Dawit u. a. (2008). “Scalability in distributed multi-agent based simulations: The JADE case”. In: *Proc. 2008 2nd Int. Conf. Futur. Gener. Commun. Networking, FGCN 2008* 5, S. 93–99.

OnLive Games (2014). URL: http://onlive.com/ (besucht am 18. 08. 2014).

Paradox Interactive (2015). *Workers not getting to work has no consequences!* URL: http://forum.paradoxplaza.com/forum/index.php?threads/workers-not-getting-to-work-has-no-consequences.842526/page-9%5C#post-19022719 (besucht am 01. 06. 2015).

Russell, Stuart und Peter Norvig (2004). *Künstliche Intelligenz: Ein moderner Ansatz*. 2. Aufl. Pearson Studium.

Schild, Klaus und Stefan Bussmann (2007). "Self-organization in manufacturing operations". In: *Commun. ACM* 50.12, S. 74–79.

Shneiderman, B. (1996). "The eyes have it: a task by data type taxonomy for information visualizations". In: *Proc. 1996 IEEE Symp. Vis. Lang.* S. 336–343.

Sony Computer Entertainment (2015). *PlayStation Now*. URL: http://www.playstation.com/en-us/explore/psnow/ (besucht am 07. 04. 2015).

Sundermeyer, Kurt und Stefan Bussmann (2001). "Einführung der Agententechnologie in einem produzierenden Unternehmen — ein Erfahrungsbericht". In: *Wirtschaftsinformatik* 43.2, S. 135–142.

Valve (2014). *Source Multiplayer Networking*. URL: https://developer.valvesoftware.com/wiki/Source%5C_Multiplayer%5C_Networking (besucht am 18. 08. 2014).

Vigueras, Guillermo, Juan M. Orduña u. a. (2013). "A scalable multiagent system architecture for interactive applications". In: *Sci. Comput. Program.* 78.6, S. 715–724.

Vigueras, Guillermo, Juan Orduña u. a. (2011). "A distributed visualization system for crowd simulations". In: *Integr. Comput. Aided. Eng.* 18.4, S. 349–363.

Westra, Joost u. a. (2009). "Adaptive Serious Games Using Agent Organizations". In: *SIKS Diss. Ser.* S. 206–220.

Yu, Yunxia und Dong Guan (2011). "Research on distributed simulation of HLA based on grid". In: *Proc. - 2011 Int. Conf. Comput. Inf. Sci. ICCIS 2011*, S. 856–859.

Glossar

Engine Komponente eines Programms die meist auf einen Aufgabenbereich spezialisiert ist. So gibt es beispielsweise Engines zur Physikberechnung oder für grafische Darstellung. Eine Sonderform bilden Spieleengines, die verschiedene Werkzeuge zur Spieleentwicklung und diverse Engines (Physik, Grafik, Sound etc.) bündelt. 7, 26, 31, 33, 37–39, 41, 42

GIS Geoinformationssystem. 18, 25, 30–32, 38, 39, 41, 42

GROUND MARS-Komponente die Geodaten zur Verfügung stellt. 24, 30

Heightmap Eine Datenstruktur (meistens ein Array), die Informationen über die Höhe der Karte an einem bestimmten Punkt enthält. Zwischen den Punkten interpolieren die meisten Engines üblicherweise automatisch. 31, 32

KI Künstliche Intelligenz ist der Versuch, Computer und Roboter so zu programmieren, dass sie wahrnehmen, schlussfolgern und entsprechend handeln können. In einigen Fällen kommt das Bestreben hinzu, die KI menschlich wirken zu lassen. (Görz, Schneeberger und Schmid, 2013). 7

KNP Krüger-Nationalpark, ein im Nordosten von Südafrika gelegenes Wildschutzgebiet. Mit ca 20.000 km² einer der größten Nationalparks Afrikas. 15, 16, 41

Layer Im GIS-Kontext wird damit eine Informationsebene wie Geländehöhe oder Niederschlag bezeichnet. Bei MARS bezeichnet ein Layer hingegen die Gruppierung zusammenhängender Informationen oder Agenten. So werden zum Beispiel Elefanten parallel zu den Marula-Bäumen in getrennten Layern simuliert. 16, 19, 22, 24, 31–33, 38, 48

LDAP Das Lightweight Directory Access Protocol ist ein Protokoll zur Abfrage oder Änderung von Daten in einem Verzeichnisdienst. 19

LIFE MARS-Komponente die unter anderem den Entwicklern der Agenten Schnittstellen zur Verfügung stellt und die Simulationsknoten verwaltet. 21, 24, 33

MARS Multi-Agent Research & Simulation. vi, 1, 2, 17, 19, 21, 22, 24, 31, 37, 42, 48

MAS Multiagentensysteme (MAS) bezeichnen ein Softwaresystem aus mehreren Agenten, die gemeinsam ein Problem lösen. Die beteiligten Agenten können dabei gleichartig oder auch unterschiedlich spezialisiert sein. 4–6, 19

MSaaS Modeling & Simulation as a Service. Simulationen sollen, ähnlich wie bei Software as a Service (SaaS), für den Nutzer skalierbar und ohne nennenswertes Hintergrundwissen oder Wartungsaufwand verfügbar sein.. 21

RabbitMQ Message Broker Software, die zur Kommunikation innerhalb des MARS-Projektes eingesetzt wird. 27

RMI Remote Method Invocation bezeichnet den Aufruf von Methoden eines Java-Objektes das in einer anderen Java-VM ausgeführt wird. 19

VIEW Die Client-Komponente, die letztendlich für die Darstellung zuständig ist. 1, 24, 30

VIS MARS-Komponente die als Client-Server-Schnittstelle für die Visualisierung dient. 1, 21, 24, 30, 33

WMS Der *Web Map Service* ist ein Dienst zur Bereitstellung von Kartendaten. Dabei können beliebige Ausschnitte und Abmessungen angegeben werden. Abhängig von den vorliegenden Daten ist es zudem möglich, die Daten verschiedener Karten zu kombinieren. 25, 32, 38

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 07. September 2015 Alexander Schnoor