Endbericht PG 467
AbSIntH - Agentenbasierte Simulation interaktiver Hochschulszenarien

Klaus Friedrichs Philip Geismann Patrick Lenger
Tycho Moencks Christian Mosblech Bastian Nau
Jan Salmen Marc Schlipsing Christian Schwede
Alessandro Selvaggio Nuhad Shaabani Tobias Wagner

29.07.2005

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Vorwort

Die Durchführung von Projektgruppen ist Bestandteil der Studiengänge Informatik und Angewandte Informatik der Universität Dortmund. Im Laufe von zwei Semestern besteht die Möglichkeit, in einem Team mit anderen Studenten eine Aufgabe durchzuführen, die aufgrund ihres Umfangs nicht von Einzelnen oder in überschaubarer Zeit zu bewältigen wäre. Das gibt sowohl Gelegenheit zur Übung in Teamarbeit und Koordination, als auch zur Bearbeitung einer komplexen Problemstellung über einen längeren Zeitraum.

Im Zeitraum vom Wintersemester 2004/2005 bis zum Ende des Sommersemesters 2005 fand an der Universität Dortmund im Fachbereich Informatik (Lehrstuhl 11) das Projekt *AbSIntH* (Agentenbasierte Simulation interaktiver Hochschulszenarien) statt. Der vorliegende Projektendbericht beschreibt die Zielsetzung, den Verlauf und das Ergebnis des Projektes.

Unser besonderer Dank gilt Prof. Dr.-Ing. Hans-Paul Schwefel, unter dessen Schirmherrschaft das Projekt stattgefunden hat, sowie unseren beiden Betreuern Christian Lasarczyk und Mike Preuß, die uns immer mit Rat und Tat zur Seite standen.

1.2 Aufbau des Endberichts

Die Gliederung und der Aufbau des Endberichts wird im Folgenden beschrieben.

Das Kapitel 1 liefert eine Einführung in die behandelte Thematik und Allgemeines über diese Projektgruppe. Im Abschnitt 1.3 werden die Ziele der Projektgruppe und eine Motivation für das Projekt erörtert. Weiter wird der zeitliche Aufwand dieses Projektes (unterteilt nach den einzelnen Phasen) dargestellt.

In Kapitel 2 folgt nun die Phase der Anforderungsdefinition, die ihrerseits in Agent/Umwelt, soziologische Anforderungen und Verhandlung unterteilt ist.

Das Kapitel 3 beschreibt das zeitliche Vorgehen bei der Entwicklung des Agenten, der Umwelt und der Verhandlung. Um dem Leser einen Einblick in die verwendete Arbeitsumgebung zu ermöglichen, werden auch die verwendeten Tools vorgestellt.

In Kapitel 4 wird die Phase des Entwurfes und der Implementierung näher beleuchtet. Dazu wird der Aufbau der wichtigsten Teilbereiche (Umwelt, Agent, Verhandlung, GUI) beschrieben sowie im Klassendiagramm Klassendiagramm. Bei zentralen Methoden werden die Berechnungsmethoden (z.B. die Entscheidungsfindung des Agenten aufgrund des verwendeten soziologischen Modells) erläutert und auf eventuelle Probleme während der Implementierung hingewiesen.

Das Kapitel 5 beschreibt aufgetretene Probleme während des Entwurfs und Änderungen während der Implementierungsphase.

Nachdem in den vorherigen Kapiteln die Modellierung und die hinter der Umsetzung der Simulation steckenden Ideen diskutiert wurden, werden im Kapitel 6 Szenarios zum Testen des Agenten und der Simulation vorgestellt. Beim ersten Szenario handelt es sich um das bekannte Spiel TicTacToe. Dieses Szenario ist zum Testen des Agenten und der bis dato umgesetzten Umwelt gedacht. Das zweite Szenario ist ein etwas komplexerer Test der Agenten. Als Umwelt wird das Spiel Robocode (siehe [ROBO]) verwendet, an das die Agenten Befehle zum Steuern eines Panzers schicken. Mit diesem Szenario war ein umfassender Test des Agenten (ohne Verhandlung) möglich. Das dritte Szenario modelliert eine Hochschule, die letztlich mit der hier entwickelten Software simuliert werden soll. Es werden sowohl Erwartungen an die Szenarios als auch Ergebnisse der Simulation beschrieben und erläutert. Weiter finden sich noch Screenshots, die das Ausgabeverhalten dokumentieren und es wird ein Bezug zur Realität hergestellt.

Das Kapitel 7 fasst die Ergebnisse dieses Projektes in einem Fazit zusammen. Das Fazit fasst die realisierten Funktionen und Ergebnisse zusammen und stellt mögliche Erweiterungen vor.

Im Anschluss an das letzte Kapitel folgen mehrere Anhänge, die weitergehende Informationen zum Projekt liefern. Der Anhang A dokumentiert die Seminar- und Tutoriumsphasen, die innerhalb der Projektgruppe abgehalten wurden. Es werden die Themen der einzelnen Beiträge kurz angeführt und ein Bezug zum Projekt hergestellt. Anhang B enthält das Handbuch, das die einzelnen Funktionen der GUI näher

beschreibt.

1.3 Motivation

Wie bereits im Vorwort angesprochen, ist das Ziel der Projektgruppe die Entwicklung einer Sozial-Simulation, um ein Hochschulszenario zu simulieren. Im Folgenden soll die Notwendigkeit einer Simulation näher betrachtet werden. Zu Beginn der Definition des Szenarios werden mögliche Einflussgrößen des Szenarios (z.B. Mitarbeiter, Geld, usw.) identifiziert und diese anschließend modelliert. Desweiteren werden Ausgabeparameter wie z.B. Studierendenzahlen, Anzahl der Abschlüsse, Anzahl der Vorlesungen, etc. festgelegt, die ein Indiz für eine gute Lehre sind bzw. deren Änderung die bestehende Lehre verbessern könnte. So kann man z.B. als Ziel definieren, dass möglichst viele Vorlesungen angeboten werden sollen, damit als Konsequenz zum einen die Vorlesungen nicht überfüllt sind und zum anderen für die Studenten ein breites Wissenspektrum abgedeckt wird. Eine andere Sichtweise wäre z.B. dass die Vergrößerung der Studentenzahlen auf eine bessere Lehre hinweist, weil einerseits die Chance auf mehr Studienabschlüsse steigt und andererseits das Studienangebot für die Studenten interessanter wird.
Anstatt der Lehre können auch andere Kriterien für eine Verbesserung herangezogen werden. Zum Beispiel könnte die Simulation als Ziel haben, die benötigten finanziellen Mittel zu minimieren und gleichzeitig eine feste Anzahl Vorlesungen anzubieten. Welches Ziel verfolgt werden soll, wird durch die Modellierung des Szenarios festgelegt. Die Simulation erlaubt es dann (in einem begrenzten Detailierungsgrad), die Werte dieser zu Beginn vermuteten Einflüsse zu verändern und so deren Wechselwirkungen zu analysieren.

Denkbar wäre anstatt einer Simulation auch eine Feldstudie, die z.B. innerhalb einer realen Universität durchgeführt wird. Vorteil dabei wäre, dass der Detailierungsgrad höher als bei einer Simulation ist. Dagegen sind die hohen Kosten und der hohe Zeitaufwand, bis man die Auswirkungen auf die Lehre sehen kann, bei dieser Methode Nachteile. Weiterhin muss man sich darüber im Klaren sein, dass die Studenten bei solch einer Studie an dieser Test-Universität zu unfreiwilligen Testobjekten werden, die eventuelle Nachteile (hervorgerufen durch schlechte Parametrisierungen der Einflussgrößen) erleiden müssen und dass eine Simulation flexibler ist (Änderungen können einfach modelliert werden, während in einer realen Universität erst eine gewisse Zeit vergeht, bis Änderungen eintreten).

Um die Auswirkungen von Parameteränderungen eines Hochschulszenarios zu untersuchen, hat es sich die Projektgruppe zur Aufgabe gemacht, ein Hochschulszenario zu modellieren und damit eine Simulation zu entwickeln.

1.3.1 Zustandekommen der PG

Im Regelfall entsteht ein Projektgruppenantrag auf Initiative eines Lehrstuhls. Dieser hat meist das Interesse, zur Erfüllung seiner Lehraufgabe Studenten in den eigenen Forschungskontext einzubeziehen. Beispielsweise hat der Lehrstuhl 1 der Universität Dortmund ein Interesse an der Verbesserung der fußballspielenden Roboter. Bei bestehendem Interesse an einem bestimmten Thema meldet der Lehrstuhl die Projektgruppe an und nach Prüfung wird die PG den Studenten vorgestellt, sodass sich diese für die PG bewerben können.

Diese Projektgruppe ist im Gegensatz zu den meisten PGs auf Initiative von Studenten entstanden. Auch wenn dies selten der Fall ist, dachten sich ein paar Studenten dieser Projektgruppe ein eigenes Thema aus. Sie interessierten sich für den Bereich der Simulation bzw. der künstlichen Intelligenz und informierten sich, welcher Lehrstuhl für diesen Themenbereich in Frage käme. Als ein Szenario für eine Simulation war zuerst ein Inselszenario geplant. Auf Nachfrage beim Lehrstuhl 11 wurden die Studenten an Christian Lasarczyk und Mike Preuss verwiesen, die einer Betreuung einer PG nicht abgeneigt waren. Im Folgenden stellten sie den beiden Mitarbeitern des Lehrstuhls ihre Ideen vor und man entwickelte gemeinsam ein Szenario, welches die Zusammenhänge an einer Hochschule simuliert. Da Christian Lasarczyk bei einem Sozionik-Projekt mitarbeitet, bot er die Unterstützung seitens der Fernuniversität Hagen an, die an dem Projekt beteiligt ist. Dieses Projekt beschäftigt sich ebenfalls mit der Modellierung von verteilten intelligenten Agenten, wobei soziologische Aspekte einfließen. Da auch die Ideen der Studenten in die Richtung einer Agentensimulation gingen, ließen sich beide Projekte gut in Verbindung bringen. Nachdem also Betreuer für diese Projektgruppe gefunden waren, verlief die Anmeldung der PG wie bei allen anderen PGs auch. Alle Studenten, die an der Idee dieser PG beteiligt waren, nahmen auch an ihr teil.

Bei dem Zustandekommen dieser PG stellte sich heraus, dass etwas spät begonnen wurde, Interessenten bzw. Betreuer für eine PG zu finden. Dies liegt darin begründet, dass die Studenten erst eine für einen Lehrstuhl interessante Idee entwickeln und dann noch einen Lehrstuhl mit Interesse an dieser Idee finden mussten. Hinzu kam, dass die Ausarbeitung des konkreten Themas mit den zukünftigen Betreuern auch etwas Zeit in Anspruch nahm, sodass für das Verfassen des eigentlichen PG-Antrags nur wenige Tage zur Verfügung standen. Diese Anmerkung ist besonders für diejenigen Leser von Bedeutung, die einem Lehrstuhl ein PG-Thema vorschlagen möchten. Von den Teilnehmern dieser PG wird darum empfohlen, so früh wie möglich zu den Lehrstühlen zu gehen und entsprechende Zeit für die Ausarbeitung mit den zukünftigen Betreuern und das Verfassen des PG-Antrags einzuplanen.

Dass die Studenten das Thema dieser Projektgruppe durch Eigeninitiative ins

Leben gerufen haben, hat abschließend betrachtet zwei Vorteile, die sich auch im Verlauf der PG bestätigt haben: zum einen konnten sie das Thema in die Richtung lenken, für die sie sich am meisten interessieren und zum anderen kannten sich schon ein Teil der Mitglieder (und auch deren einzelnen Stärken und Schwächen), sodass die Zusammenarbeit sehr gut verlief. Auch die weiteren Mitglieder, die nicht bei der Ideenfindung dabei waren, wurden schnell integriert.

1.4 Teamarbeit und Organisation

Ein pädagogisches Ziel der Projektgruppe ist es, den Beteiligten Erfahrung im Bereich der Teamarbeit zu vermitteln. Meistens bearbeiten Studenten ihre Übungszettel allein oder nur in Kleingruppen und Klausuren müssen sowieso alleine geschrieben werden. Zwar wurde im Softwarepraktikum auch im Team gearbeitet, aber die zwei dort bearbeiteten Projekte waren wesentlich kleiner im Umfang und die Anzahl der Beteiligten geringer. Die Projektgruppe soll also den Studenten ermöglichen, Erfahrungen im Arbeiten in einem größeren Team (in diesem Fall 12 Studenten) zu sammeln und sie somit auch in der Praxis auf den Beruf vorzubereiten. Jeder Einzelne soll spätestens in bzw. nach dem Projekt wissen, dass ein derartiger Umfang nur erfolgreich bearbeitet werden kann, wenn alle Teilnehmer ein Team bilden.

Einige Aufgaben, die innerhalb des Projektes zu erledigen waren, mussten von der Gruppe gemeinsam bearbeitet werden (etwa Entwurf, Ideen sammeln, Vorgehen abstimmen). Bei vielen anderen Aufgaben machte es hingegen wenig Sinn, sie mit allen Teilnehmern gleichzeitig zu bearbeiten (etwa Recherche, Dokumentation, Implementierung). Ein Grund hierfür ist z.B., dass kaum jemand über alle Einzelheiten des gesamten Projektes Bescheid wissen kann. Ein weiterer Grund sind die unterschiedlichen Erfahrungen bzw. Fähigkeiten der Projektmitglieder. Die einen verfügen z.B. schon über gute Kenntnisse in der GUI-Programmierung, während andere sich um die Kernalgorithmen kümmern. Demnach ist es effektiver, wenn Kleingruppen gebildet werden. Innerhalb der Projektgruppe können die Teilnehmer gut ausloten, wo ihre Stärken liegen und wie groß die einzelnen Teilgruppen sein sollten. Beispiele für Kleingruppen während der PG waren GUI, Agent, Umwelt und Szenario. Natürlich lernten alle Teilnehmer auch neue Dinge, da der Aufgabenbereich einer Teilgruppe einerseits nicht nur eine kleine Teilaufgabe beinhaltet und andererseits die Mitglieder der Teilgruppen auch wechselten.

Damit das Arbeiten in Kleingruppen effektiv vonstatten geht, ist eine gute Kommunikation zwischen ihnen unabdingbar. Ein Teil der Kommunikation läuft über zu Beginn des Projektes angefertigte (Klassen-)Diagramme. Desweiteren stellt man

während der 2 Semester zwischendurch fest, dass manche Schnittstellen noch nicht ausreichend abgesprochen sind und noch Diskussionsbedarf besteht. Doch aus diesen Erfahrungen lernen im Idealfall alle Teilnehmer, dass eine gute und vollständige Modellierung die Grundlage für ein erfolgreiches und effektiv durchgeführtes Projekt ist.

1.4.1 Aneignung von Wissen und Erfahrung

Auch wenn Teamarbeit ein wichtiger Aspekt für die Projektgruppe ist, soll auch weitergehendes Wissen selbständig erworben bzw. bestehendes vertieft werden. Nachfolgend werden ein paar Beispiele für die wichtigsten Themenbereiche genannt, die für dieses Projekt notwendig waren:

- (Multi-) Agentensysteme
- Modellierung eines soziologischen Agentenmodells
- Funktionsweise von Fuzzyregeln und deren Umsetzung mit FuzzyJ
- Umgang mit LaTeXund Javadoc für die Dokumentation

Um das benötigte Wissen effektiv zu erwerben und dieses der gesamten Projektgruppe zur Verfügung zu stellen, wurde zu Beginn der Projektgruppe eine Seminarphase durchgeführt. Auf diese Weise hat sich jeder PG-Teilnehmer ein vertieftes Wissen über ein bestimmtes Thema angeeignet und durch seinen Vortrag den anderen Teilnehmern einen Überblick verschafft (siehe Anhang A).

Wie bereits in Kap. 1.3.1 schon angesprochen, soll die Agentensimulation auf einem soziologischem Modell aufbauen. Dazu wurde der Projektgruppe seitens der Fernuniversität Hagen Unterstützung angeboten, da alle Studenten aus der Informatik stammen und noch keine Erfahrungen mit soziologischen Theorien hatten. Eine Herausforderung bestand darin, das soziologische Modell aus der Sichtweise eines Informatikers zu sehen und den Agenten entsprechend zu modellieren. Nähere Informationen über das soziologische Modell und zur Werterwartungstheorie finden sich in Kap. 2.2.

Ein anderer positiver Effekt der Projektgruppe ist die Vertiefung bereits bestehender Programmierkenntnisse in Java, da jeder Teilnehmer einen Teilbereich der Simulation implementiert hat. Somit sollte jeder aus diesem Projekt mit praktischen Programmiererfahrungen herausgehen.

1.4.2 Kommunikation

Um möglichst schnell und einfach über den Verlauf des Projektes kommunizieren zu können, bediente sich die Projektgruppe unterschiedlicher Möglichkeiten. So wurde zu Beginn festgelegt, dass jede Woche an 2 festen Terminen Besprechungen mit der ganzen Gruppe und beiden Betreuern stattfinden. Dabei sollte jeweils ein Protokoll erstellt werden, so dass auch zu späteren Zeitpunkten (z.B. für fehlende Mitglieder) der aktuelle Stand und aktuelle Beschlüsse nachvollziehbar sind.
Es wurde eine Homepage für die Projektgruppe eingerichtet, auf der die Protokolle elektronisch archiviert wurden. Darüberhinaus konnten hier Termine in einem Kalender verwaltet und Dateien für alle bereitgestellt (Dokumentation) werden. Außerdem wurden auf der Seite häufig Ideen gesammelt und Aufgaben verteilt. Zur direkten Kommunikation zwischen einzelnen Mitgliedern stand eine Kontaktliste zur Verfügung.
Schließlich wurde ein eMail-Verteiler eingerichtet, der benutzt werden konnte, um eMails an alle Teilnehmer der Projektgruppe zu senden.

1.4.3 Extreme Programming

Zu Beginn der Projektgruppe einigten sich die Mitglieder, dass speziell für die Phasen der Implementierung eine besondere Arbeitsweise gewählt werden sollte, nämlich das sogenannte Extreme Programming.
Extreme Programming ist ein Modell, das Auftraggebern und Kunden ermöglichen soll, schnell eine passende Software zu erstellen, wobei insbesondere alle Arten von Umwegen vermieden werden sollen [BA99] [LRW05] [URL01]. Solche Umwege könnten z.B. Fehler (und damit deren Beseitigung) sein oder aber das Implementieren nicht benötigter Teile.
Um dieses Ziel zu erreichen, sind an verschiedenen Stellen besondere Formen des Arbeitens und der Kommunikation nötig. So arbeiten z.B. beim Programmieren 2 oder 3 Entwickler gleichzeitig an einem Bildschirm, so dass bei Unklarheiten direkt Rücksprache stattfindet und Fehler kaum entstehen können. Zusätzlich findet regelmäßig ein Austausch einzelner Mitglieder zwischen verschiedenen solcher Gruppen statt. Dadurch wird gewährleistet, dass jeder Entwickler über möglichst viele Teile des Projektes Bescheid weiß und es kommt in den Programmiergruppen häufiger zu neuen und Ideen und besseren Lösungen.
Zusätzlich muss ständig über den Stand des Projektes gesprochen werden und mit den gesetzten Zielen verglichen werden (siehe auch Kap. 1.4.4). Dadurch können falsche Entwicklungen im Projekt schnell erkannt werden und entsprechende Maßnahmen getroffen werden. So könnten zum Beispiel die Aufgaben einiger Programmier-

Teams kurzzeitig geändert werden.

1.4.4 Zeitplan

Zu Beginn der beiden Semester diskutierte die Projektgruppe jeweils über das Aussehen eines für alle verbindlichen Zeitplans (siehe 1.4.4 bzw. 1.4.4). Darin wurden Meilensteine in Bezug auf die Projektarbeit festgelegt, so etwa Beginn und Ende von Anforderungsbeschreibung, Entwurf und Implementierung. Darüberhinaus sollte (als übergeordneter Zeitplan) am Ende des ersten Semesters ein einzelner Agent in einer Testumgebung agieren (siehe Kap. 6.1), am Ende des zweiten Semesters die Simulation einer Hochschule mit vielen Agenten lauffähig sein (siehe Kap. 6.2).
Durch das Festlegen des Zeitplans sollte sichergestellt werden, dass zwischen den Phasen der Parallelarbeit (siehe Kap. 1.4) immer wieder Teilziele erreicht sind - der Stand konnte danach von der ganzen Gruppe besprochen und beurteilt werden. Entweder wurde der Teilbereich daraufhin als abgeschlossen betrachtet oder nötige Nachbesserungen geplant. In jedem Fall wurde das weitere Vorgehen, insbesondere die Aufgabenverteilung, diskutiert.
Natürlich mussten im Laufe der Semster immer wieder kleine Anpassungen an den ursprünglichen Plänen vorgenommen werden. Gründe dafür waren z.B. Fehleinschätzungen, technische Schwierigkeiten oder unerwarteter Nachbesserungsbedarf.

Zeitplan im 1. Semester der PG (WS04/05)

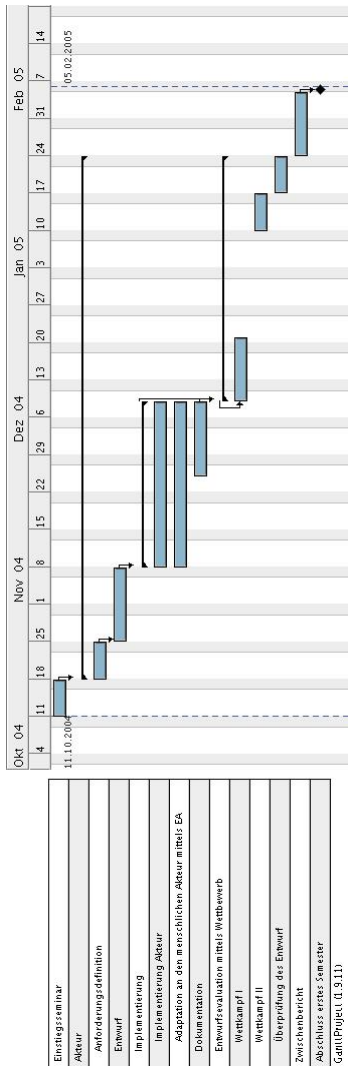

Abbildung 1.1: Zeitplan für das erste Semester

Zeitplan im 2. Semester der PG (SoSe05)

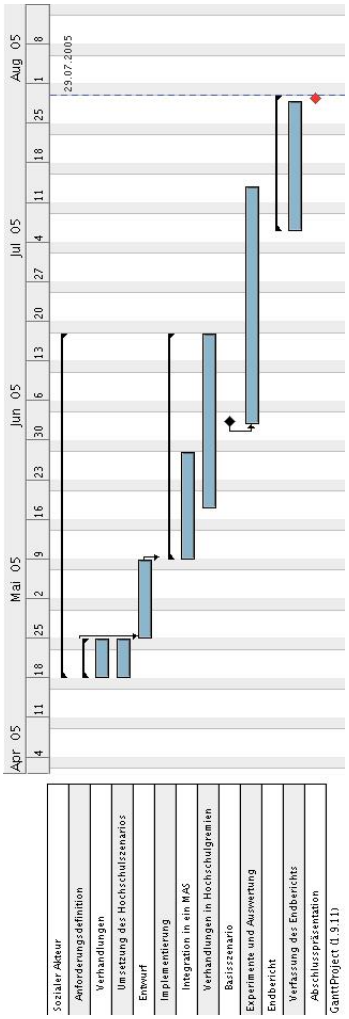

Abbildung 1.2: Zeitplan für das zweite Semester

Kapitel 2

Anforderungsbeschreibung

Die Entwicklung einer Software erfolgt typischerweise kundenbezogen, d.h. unter Berücksichtigung der Anforderungen der späteren Nutzer. Ein Anforderungsmodell hat den Zweck, die gewünschten Funktionen und Eigenschaften der Software möglichst präzise festzulegen. Diese Anforderungen werden durch den Kunden vorgegeben. Als "Kunde" der Projektgruppe sind zwei Parteien zu berücksichtigen. Auf der einen Seite war ein formaler Rahmen durch den Projektgruppenantrag gegeben, welcher ein Mindestziel (siehe Kap. 2.1) definiert. Weiterhin kamen Wünsche der mit der PG kooperierenden Soziologen hinzu.

Das Kapitel "Anforderungsbeschreibung" beschreibt die Festlegung der geforderten und gewünschten Funktionspunkte. So wird zunächst das Minimalziel aus der Projektgruppenbeschreibung zitiert und näher spezifiziert. Anschließend wird ein Einblick in die soziologischen Hintergründe dieses Projektes gegeben, der sowohl eine Beschreibung des genutzten Modells als auch die Zusammenarbeit mit der Fernuniversität Hagen beschreibt. Die Anforderungsmodellierung der Software wird über beide Semester getrennt betrachtet, um die Trennung aufzuzeigen, die auch bei der Implementierung zwischen dem Agent und einer Umwelt (1. Semester) sowie seiner Einbettung in ein Multiagentensystem (2. Semester) gemacht wurde.

2.1 Minimalziel

„Das Minimalziel ist die Umsetzung eines Simulationsszenarios mit sozialen Akteuren als Agenten, die in der Lage sind, mit anderen Agenten über die Verteilung von Ressourcen zu verhandeln. Das gemeinsame Ziel der Akteure muss dabei sein, eine Menge von Dienstleistungen zur Verfügung zu stellen. Damit das Szenario neben

dem Aspekt der Kooperation auch Konfliktsituationen enthält, sollte zumindest ein Teil der Akteure auch individuelle Interessen verfolgen, und die Situation sollte von den Akteuren subjektiv wahrgenommen werden.“ (PG-Antrag)

2.2 Soziologische Anforderungen

Dieses Kapitel beschäftigt sich mit der Auseinandersetzung und Entwicklung der soziologischen Theorien, die für die Projektgruppe (kurz: PG) verwendet wurden. Ein Verständnis dieser Theorien ist für die Anforderungsmodellierung erforderlich.

Abstimmung und Zusammenarbeit mit der Fernuniversität Hagen
Das Ziel der Projektgruppe, die Umsetzung eines Simulationsszenarios mit sozialen Akteuren als Agenten, sollte mit Hilfe des Fachbereichs „Kultur- und Sozialwissenschaften“ der Fernuniversität Hagen realisiert werden. Eine enge Zusammenarbeit mit dem Soziologen Dr. Thomas Kron sollte zur Konsistenzsicherung des sozialen Aspektes der Simulation beitragen.

2.2.1 Die soziologische Handlungstheorie

In der soziologischen Theorie bestehen Akteure aus verschiedenen Charaktertypen. Diese bestimmen das Verhalten des Akteurs, da sie bestimmte Ziele verfolgen oder sich an bestimmte Verhaltensregeln halten. So gibt es zum Beispiel das Modell eines „Homo Oeconomicus“, der durch eine wirtschaftliche Betrachtung und Auswertung der an ihn gestellten Aufgaben und Bedingungen, die für ihn beste Lösung berechnet. Auf der anderen Seite existiert zum Beispiel das Modell des „Homo Sociologicus“, dessen Entscheidung „von Normen (nach Dahrendorfs - juristisch inspirierter - Einteilung: Muss-, Soll- und Kann-Normen), von Erwartungen und von (belohnenden, bestrafenden) sozialen Sanktionen anderer (auch: des Rollenträgers selbst) geprägt werden“ [WISOC]. Die nach außen wirkende Handlung eines Akteurs wird letzlich durch die ihm zugeordneten Charaktertypen und deren Gewichtung determiniert. Auf der Basis weniger Charaktertypen ist es so möglich, eine Vielzahl von unterschiedlichen Akteuren zu erzeugen.

Um diese Theorie mathematisch zu erfassen wird die Entscheidungsfindung eines Charaktertypen weiter spezifiziert. Er trifft diese auf Basis einer Werterwartungstabelle, in der Aktionen möglichen Folgen gegenübergestellt werden. Jede Folge hat

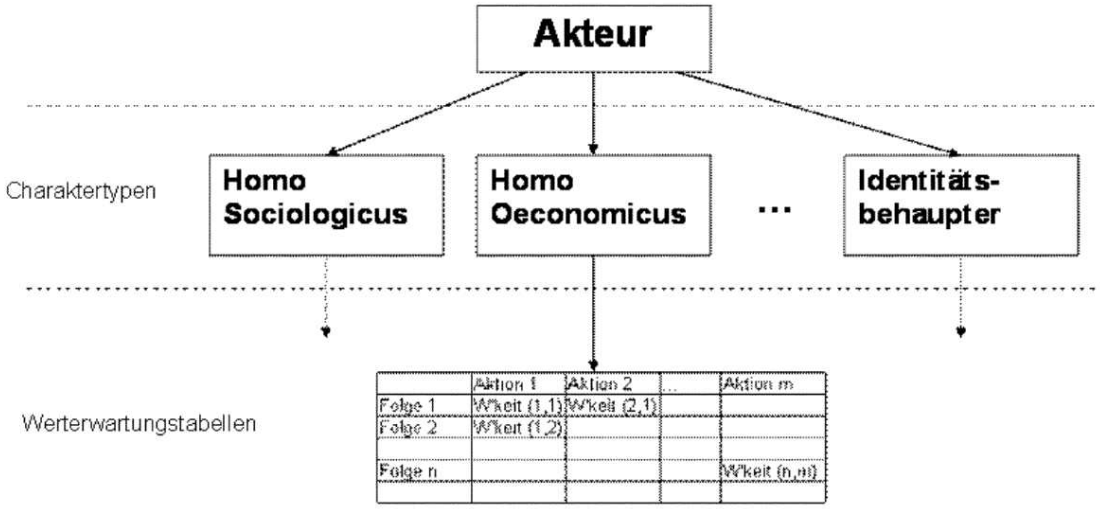

Abbildung 2.1: Schema der soziologischen Handlungstheorie

dabei einen Nutzen für den Akteur und jede Aktion ist mit individuellen Kosten verbunden. Diese Tabelle enthält Wahrscheinlichkeitswerte für alle möglichen Kombinationen aus Folgen und Aktionen, die sich aus Regeln ergeben, welche für jeden Charaktertyp individuell gestaltet werden. Es wird die Aktion mit höchster Wahrscheinlichkeit ausgeführt, die die höchste Werterwartung in der Werterwartungstabelle besitzt. Die Auswahl einer Handlung erfolgt also Fitnessproportional zu Ihrer Werterwartung, aber nicht deterministisch.
Abb. 2.1 gibt einen schematischen Überblick über die hier genutzte Variante der Theorie. Diese unterscheidet sich von der „Grundversion“ durch die Nutzung mehrerer Charaktertypen in einem Akteur. In der klassischen Werterwartungstheorie besteht jeder Akteur aus einem Charaktertypen.

2.2.2 Anforderungen aus der Soziologie an Agent und Umwelt

Die mathematische Modellierung und algorithmische Umsetzung der soziologischen Theorie der Werterwartungstabelle stellt eine wesentliche Anforderung dar. Agenten sollen ihre Entscheidungen auf Basis einer Werterwartungstabelle treffen, die den Aktionen mögliche Folgen gegenüberstellt. Die Regeln innerhalb der Werterwartungstabelle sollen auf vermuteten Umweltvoraussetzungen basieren (subjektive

Umweltbilder). Als Resultat ergibt sich eine Wahrscheinlichkeitseinstufung für die Kombination aus Handlung und erwarteter Folge. Ein einfaches Beispiel ist die Kombination „Einkaufen“(Aktion) - „Verarmen“(Folge) und die Regel „Wenn Geld hoch (Umweltvoraussetzung), dann Wahrscheinlichkeit niedrig“. Die Regel wird überprüft, wenn die Werterwartungstabelle aufgerufen wird und liefert für deren Eintrag an der Stelle „Einkaufen“ - “Verarmen“ eine niedrige Wahrscheinlichkeit, je nachdem wie viel Geld im Augenblick der Abfrage verfügbar ist.
Dieses Modell ist im ersten Semester komplett umgesetzt und mittels JAVA implementiert worden, so dass zum Abschluss dieses Semesters ein voll funktionsfähiges Akteursmodell auf der Grundlage autonomer Agenten realisiert ist. In einem auf Robocode basierenden Szenario (siehe Kap. 6.1) soll das Konzept auf seine Funktion hin getestet werden.

2.2.3 Anforderungen aus der Soziologie an das Multiagentensystem

Die Untersuchung von Verhandlungen sowie der dazu unumgänglichen Kommunikation der beteiligten Verhandlungspartner bilden ein weites Gebiet der soziologischen Forschung. Der Projektgruppe waren zunächst keine umsetzbaren Theorien bekannt, die den für die Simulation relevanten Teil beschreiben würden. Als Element der Einarbeitung in diese Thematik wurde daher ein Fragenkatalog verfasst, mit dessen Hilfe typische Merkmale fokussiert werden sollten. Dieser Fragenkatalog wurde via Email an die das Projekt beratenden Soziologen gesendet. Eine Übersicht der Antworten befindet sich im Anhang (siehe Kap. 7.2). Durch die Auswertung der Antworten ergaben sich die minimalen Anforderungen, die an Verhandlungen gestellt werden. Diese sind in Kap. 2.4 aufgeführt.

2.3 Anforderungsmodellierung von Agent und Umwelt

Um das komplexe Thema der Simulation von sozialen Systemen handhabbar zu machen, waren zunächst ausreichende Abstraktionsniveaus festzulegen. Mit der Aufteilung in die Arbeitsbereiche Agent und Umwelt (1. Semester) und Multiagentensystem (2. Semester) konnte eine erste Gliederung erreicht werden. Diesem Aufbau folgend, widmet sich dieses Kapitel der jeweiligen Entwicklung eines Anforderungsmodells, welches die umzusetzenden Eigenschaften und Funktionseinheiten des Systems determiniert.

Anwendungsfallbeschreibung aus Benutzersicht (siehe Abb. 2.2)

Benutzungsschnittstelle
Das System verfügt über eine Benutzungsschnittstelle, die die Kommunikation zwischen Anwender und Simulation erlaubt. Dabei kann der Anwender eine Simulation erstellen, indem er Agenten und die Umwelt (inkl. der notwendigen Parameter) erzeugt. Weiterhin kann eine gespeicherte Simulation geladen werden, um sie fortzusetzen. Die Eigenschaften der erzeugten Agenten bzw. der Umwelt können eingesehen und verändert werden. Um den Simulationsverlauf nachzuvollziehen, kann die History der Agenten eingesehen werden.

Simulation starten
Der Benutzer kann eine vollständig erstellte oder geladene Simulation starten. Falls keine Umwelt und kein Agent erzeugt wurden, wird der Benutzer aufgefordert diese zu erzeugen. Weiterhin können durch den Benutzer weitere Agenten hinzugefügt werden.

Agent erzeugen
Der Benutzer gibt durch eine Gewichtung der Charaktertypen das gewünschte Profil eines Agenten an. Weiter legt er die möglichen Handlungsalternativen als auch die Regeln zur Verbindung von Alternativen (Aktionen) mit den möglichen Folgen fest.

Umwelt erzeugen
Passend zum Szenario werden die Beziehungen der Agenten zueinander und die erforderlichen Parameter der Umwelt festgelegt.

Agent modifizieren
Der Benutzer kann die zuvor eingestellten bzw. erlernten (aktuellen) Parameter verändern.

Parameter verwalten (Agent)
Der Benutzer kann die zuvor eingestellten bzw. erlernten (aktuellen) Parameter einsehen.

Agent speichern
Es werden alle Parameter des Agenten und seine History in einer Datei gespeichert.

Agentenverlauf ausgeben
Der Benutzer kann die History (Entscheidungen der einzelnen Charaktertypen, Ge-

wichte, ausgeführte Handlungen) eines Agenten einsehen. Weiter kann der Kommunikationverlauf zwischen mehreren Agenten beobachtet werden.

Umwelt modifizieren
Der Benutzer kann die aktuellen Parameter der Umwelt ändern.

Parameter verwalten (Umwelt)
Der Benutzer kann die aktuellen Parameter der Umwelt einsehen.

Umwelt speichern
Die aktuellen Parameter der Umwelt werden in eine Datei gespeichert.

Simulation speichern
Die komplette Simulation (inkl. Umwelt und Agenten) wird persistent gespeichert, so dass sie aus diesem Zustand wieder fortgesetzt werden kann.

Simulation laden
Eine gespeicherte Simulation wird mit allen Agenten und der Umwelt geladen, sodass sie fortgesetzt werden kann.

Systeminterne Anwendungsfallbeschreibung (siehe Abb. 2.3)

Informationen aufnehmen
Ein Agent kann Informationen aus seiner Umwelt aufnehmen. Dies können Informationen bezüglich seiner eigenen Ressourcen (z.B. Geld) als auch sonstige Parameter (z.B. Anzahl der Studenten im aktuellen Semester) sein.

Entscheidungen treffen
Agenten können auf Basis der Umweltinformationen mit Hilfe ihrer Charaktertypen Entscheidungen autark treffen. Als Konsequenz einer Entscheidung werden optional Handlungen durch den Agenten ausgeübt. Hierbei können Informationen publiziert oder Umweltparameter (gemäß der gewählten Aktion) manipuliert werden.

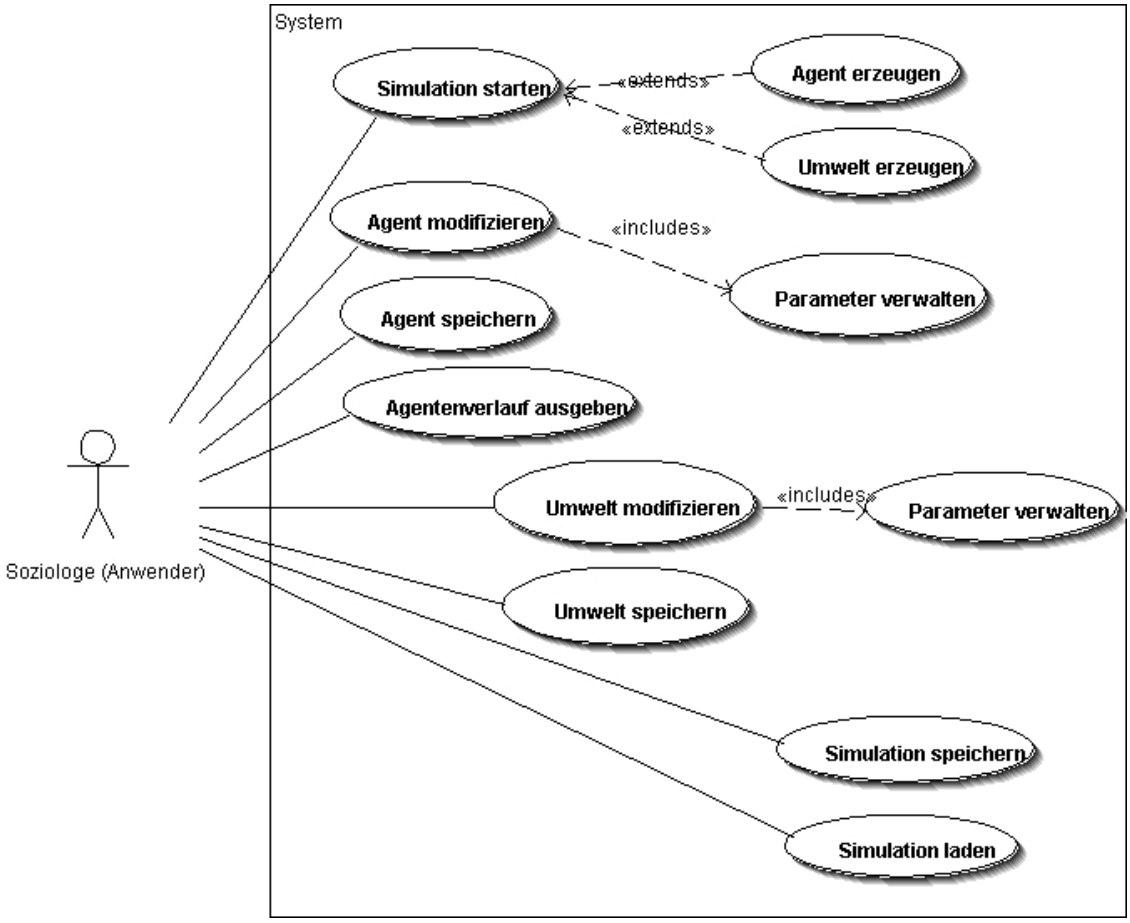

Abbildung 2.2: Use-Case Diagramm für den Benutzer

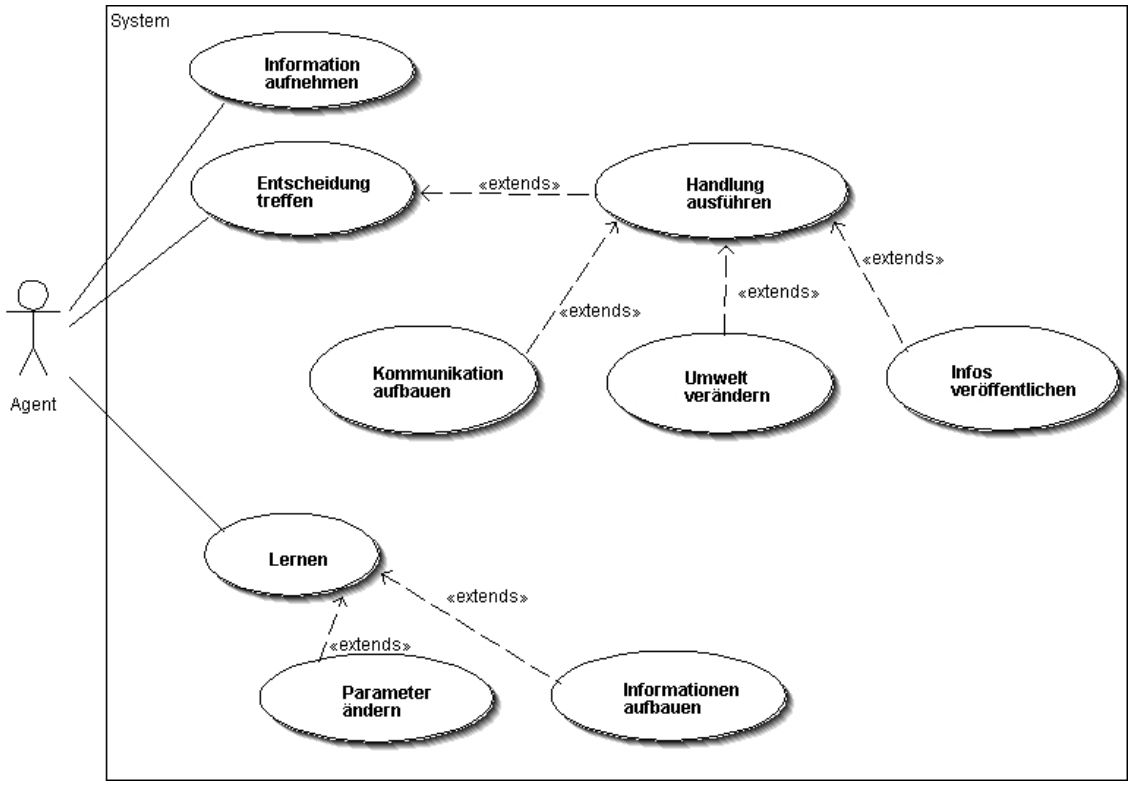

Abbildung 2.3: Use-Case Diagramm für den Agenten (systemintern)

2.3.1 Datenmodell von Agent und Umwelt

Die zu entwickelnde Simulations-Software soll primär dem Forschungsbereich der Soziologie als Werkzeug zur Evaluation von Verhaltenstheorien dienen. Der Benutzer soll hierzu virtuelle Agenten erstellen und definieren können. Da Agenten in einer Umwelt agieren, soll auch diese frei durch den Benutzer definierbar sein, um einen ausreichenden Freiheitsgrad des Systems gewährleisten zu können.

Agenten und Charaktertypen

Der Agent soll einen einzelnen Akteur repräsentieren, der aus der simulierten Umwelt Informationen aufnimmt und daraus Entscheidungen ableitet und handelt. Er ist aus unterschiedlichen Charaktertypen zusammengesetzt, welche durch die Soziologie motiviert sind. Ihre Handlungsempfehlungen stellen die Basis der Entscheidungsfindung für den Agenten dar. Als Grundlage für das Generieren der Handlungsempfehlungen soll hier die Werterwartungstheorie eingesetzt werden, die für den Benutzer über Werterwartungstabellen administrierbar ist.

Werterwartungstabelle und Regelbasen

Die Werterwartungstabelle (kurz: VET) soll die Datenhaltung für die in der Regelbasis berechneten Werte bilden, auf die der Charaktertyp dann zugreifen kann. Dabei muss die VET für jede Aktion eines Charaktertypen eine Wahrscheinlichkeit für das Eintreffen jeder seiner Handlungsfolgen speichern. Neben dem Hinzufügen und Löschen von Aktionen und Folgen für den Charaktertypen, muss die VET ebenfalls ein Interface zur Konfiguration der Regelbasen bereitstellen. Jeder der $a \cdot f$ ($a \equiv$ Anzahl der Aktionen, $f \equiv$ Anzahl der Folgen) Wahrscheinlichkeitswerte soll durch eine eigene Regelbasis berechnet werden. Die Berechnung der Tabelle wird mit der Eingabe der aktuellen Umweltinformationen vom Agenten angestoßen. Diese leitet die VET an jede in ihr enthaltene Regelbasis zur Auswertung der Fuzzy-Regeln weiter. Die hier benutzte Regelbasis muss in der Lage sein, bei Eingabe der aktuellen Variablenwerte auf Grundlage der zuvor angelegten Regeln und linguistischen Terme einen Wahrscheinlichkeitswert zurückzuliefern. Die Funktionalität wird mit Hilfe der FuzzyJ-Bibliothek (siehe Kap. 3.3) realisiert. Später soll die Berechnung von Handlungsabläufen durch die Umwelt auch mit Hilfe einer Regelbasis realisiert werden.

Ressourcen

Eine Ressource soll Informationen über den jeweiligen atomaren Baustein der Umwelt enthalten. Das kann zum Beispiel ein Kontostand, eine familiäre Beziehung oder ein Wahrheitswert sein. Es muss gewährleistet sein, dass Agenten auf diesen Ressourcen nur erlaubte Operationen durchführen. Das zentrale Verwaltungsrecht über die Ressourcen muss daher bei der Umwelt liegen, welche die Agenten dann mit den Ressourcen versorgt, auf die sie Lese- bzw. Schreibrechte besitzen. Die für die Agenten sichtbaren Ressourcen bilden das Umweltbild des jeweiligen Agenten. Dieses soll die Basis für die Entscheidungsfindung darstellen.

Die Umwelt

Die Umwelt soll den Steuermechanismus der Simulation beherbergen. Sie verwaltet alle Ressourcen, informiert Agenten über die Veränderungen und muss nach den Aktionen der Agenten passende Veränderungen in den Ressourcen berechnen. Dabei soll ein eventbasierter Mitteilungsmechanismus realisiert werden. Das bedeutet, dass nur bei Änderungen in den Ressourcen die Agenten, welche von diesen Änderungen betroffen sind, nach Ihrer Reaktion befragt werden. Diese Reaktionen können neue Änderungen bedingen, wobei diese nicht zwangsweise sofort, sondern nach einer gewissen Dauer bzw. zyklisch situiert sein können. Bevor die Änderungen berechnet werden, muss die Umwelt jeweils überprüfen, ob die Rechte bzw. Voraussetzungen zur Ausführung der Aktion bestehen. So kann ein Agent beispielsweise nur eine Vorlesung anbieten, wenn er über das entsprechende Personal verfügt. Außerdem muss sichergestellt sein, dass die Umwelt die einzige Stelle des Systems ist, an welcher Ressourcen, Agenten und Aktionen verändert, erzeugt und gelöscht werden können. Sie wird somit die direkte Schnittstelle zur GUI darstellen. Dieser zentrale Verwaltungsmechanismus sichert die Konsistenz des Systems.

Beschreibung des Problembereichsmodells

Das vorliegende Diagramm (siehe Abb. 2.4) beschreibt das Modell einer Simulation, in der ein komplexer Agent in einer Umwelt agiert.
Ein Agent besteht dabei aus einer Menge von Charaktertypen, die unabhängig voneinander Entscheidungen treffen. Agenten agieren in einer Umwelt, die Ressourcen und Informationen verwaltet. In Abhängigkeit von den Umweltinformationen treffen Agenten Entscheidungen, die zu einer Handlung führen. Die Folgen einer Handlung werden durch Änderung der die Umwelt beschreibenden Ressourcen berechnet.

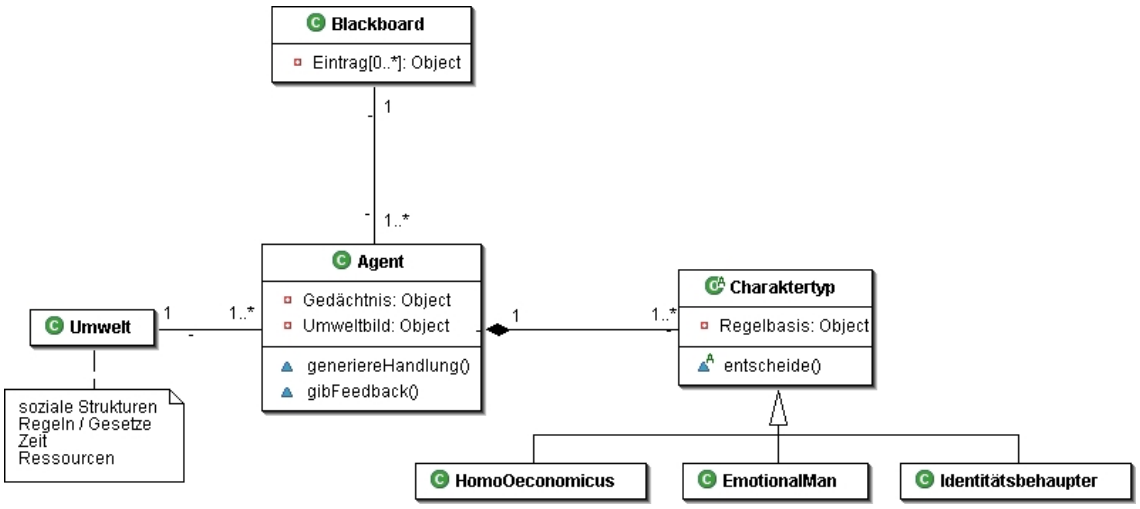

Abbildung 2.4: Problembereichsmodell Agent und Umwelt

2.4 Anforderungsmodellierung des Multiagentensystems

Nach der im ersten Semester erfolgten Realisierung von autonomen Agenten diente das zweite Semester der Projektgruppe dazu, die für Verhandlungen benötigte Infrastruktur bereitzustellen. Ziel war es, eine möglichst wirklichkeitsgetreue Abbildung einer Verhandlung zu realisieren. Ziel dieser Verhandlung soll nicht die Ermittlung eines rein rechnerisch optimalen Ergebnisses sein, sondern die Simulation von sozialen Interaktionen während eines Verhandlungsprozesses.

2.4.1 Beschreibung des Algorithmus des Multiagentensystems

Gegenstand einer Verhandlung sind Ressourcen. Die an der Verhandlung beteiligten Agenten werden über eine anstehende Verhandlung informiert. Es steht den Agenten jedoch frei, die Verhandlung aufzusuchen. Alle an der Verhandlung teilnehmenden Agenten (im folgenden kurz Agenten) erhalten von der Umwelt eine initiale Sicht der zu verhandelnden Ressourcen. In einer Vorrunde sollen alle Agenten einen Vorschlag abgeben, dieser wird allen anderen mitgeteilt. Hiermit soll ein initialer Gedankenaustausch - wie er im Vorfeld einer realen Sitzung meistens stattfindet - nachempfunden werden. Nun soll ein zufällig ausgewählter Agent einen Vorschlag äußern. Alle anderen Agenten stimmen über den gemachten Vorschlag ab. Wird eine vom Benutzer vorgegebene Quote (≡ Anzahl der Ja-Stimmen) erreicht, gilt die

Verhandlung als beendet. Der akzeptierte Vorschlag wird nun an die Umwelt zurückgegeben. Konnte keine Einigung erzielt werden, wird iterativ ein weiterer Agent nach einem Vorschlag befragt. Ein Abbruch erfolgt spätestens dann, wenn eine vorgegebene Anzahl an Verhandlungsrunden erreicht ist. Bei der Erstellung und Bewertung von Vorschlägen sollen die Agenten bzw. deren Charaktertypen auf ihre Werterwartungstabellen zurückgreifen (siehe Abb. 2.6).
Um zu vermeiden, dass die Agenten nur Ihren eigenen Nutzen optimieren, soll ein Steuerparameter eingeführt werden. Dieser ermöglicht es den Charaktertypen, Interessen der Allgemeinheit in die Berechnung mit einzubeziehen. Mit Hinblick auf das zu simulierende Uni-Szenario entspricht dieser Parameter einer globalen Lehre. Alle Agenten sollen dadurch ein definiertes Interesse daran zeigen, dass andere Lehrstühle ein Mindestmaß an Mitteln bekommen.
Dieses Modell ist aus der Kombination verschiedener Umsetzungsideen der PG-Teilnehmer entstanden. Es gab hauptsächlich zwei Einflussbereiche, die die Grundlage für dieses Konzept bilden: Zum einen war es den Teilnehmern wichtig, eine realitätsnahe Abbildung einer Verhandlung zu erzielen, während gleichzeitig möglichst umfassend das soziologische Modell der Werterwartungstheorie weiter genutzt werden sollte. Eine detaillierte Beschreibung des Entscheidungsprozesses, der diesem Algorithmus zu Grunde liegt, findet sich im Kapitel 3.2.

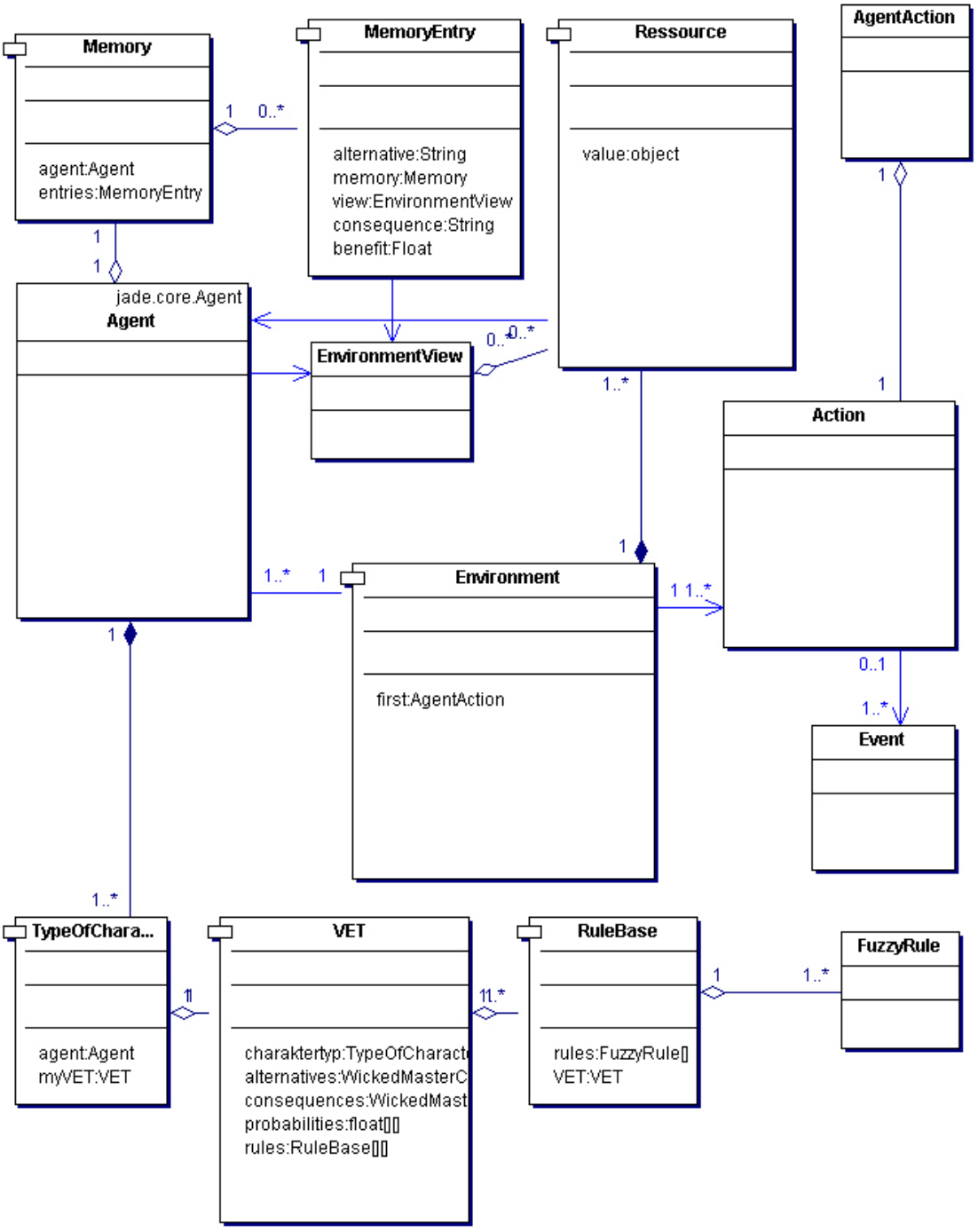

Abbildung 2.5: Problembereichsmodell des Multiagentensystems

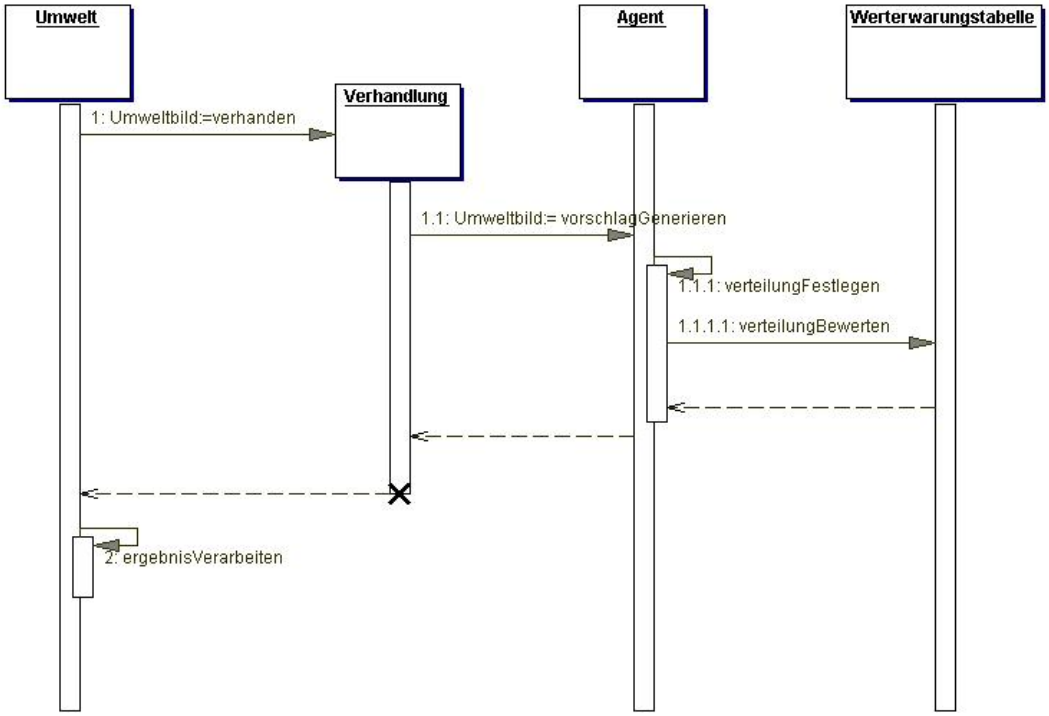

Abbildung 2.6: Sequenzdiagramm für die Verhandlung

Kapitel 3

Vorgehen und Konzeptentwicklung

Dieses Kapitel gibt einen Überblick über die Entwicklung der Ideen der Projektgruppe. Hierbei erfolgt eine Gliederung gemäß den aufeinander aufbauenden Themenbereichen Agent und Umwelt (im ersten Semester umgesetzt) sowie des im zweiten Semester der Projektgruppe realisierten Multiagentensystems.

3.1 Vorgehen bei der Realisierung von Agent und Umwelt

Die konzeptionelle Entwicklung des Projektes während des ersten Semesters verlief über einen Kernzeitraum von sechs Wochen. Um einen Einstieg in die Thematik zu bekommen wurden durch einen Gedankenaustausch die individuellen Vorstellungen der PG-Teilnehmer transparent gemacht. Hierbei äußerte jeder Teilnehmer die aus seiner Sicht wesentlichen Aspekte eines Agentensystems. In mehrfachen Iterationen -unterstützt durch die Vorträge (siehe Anhang A.1)- kristallisierten sich maßgebliche Punkte heraus. Die Gruppe einigte sich darauf, dass ein Agent eine vorgegebene Menge von Eigenschaften und Verhaltensmustern haben sollte. Durch Anregungen des das Projekt begleitenden Soziologen Dr. Thomas Krohn wurde dieser Ansatz weiter verfeinert. Um einen Agenten auf wesentliche Charakterspezifika reduzieren zu können, wurde ein Schichtenaufbau beschlossen. Hierbei besteht ein Agent aus einer Reihe von Charaktertypen, die in unterschiedlichen Gewichtungen das Verhalten des Agenten beeinflussen. Dieser Ansatz folgt stark den Ansichten der Soziologie und ermöglicht so eine Einbeziehung des entsprechenden Fachwissens. So zeigte sich die Entscheidungsfindung als wesentliches Element, d.h. es musste ein Konzept erstellt und umgesetzt werden, welches die menschliche Entscheidungsfindung bestmöglich

abbildet. Aus der Soziologie bekannt sind verschiedene Grundcharaktere (u.a. Homo Oeconomicus, Identitätsbehaupter, Emotional Man, Homo Politicus) deren Gewichtung zu individuellen Charakteren führt. Für die Realisierung dieser Gewichtung entschied sich die Projektgruppe für zwei Konzepte, die nebeneinander eingesetzt werden sollen: statische und dynamische Gewichtungsfunktionen. Dieser Ansatz ermöglicht es, Agenten in ihrem Grundverhalten zu determinieren und trotzdem noch Raum für dynamische Änderungen (z.B. über evolutionäre Strategien) zu behalten. Zur besseren Strukturierung der Konzepterstellung wurden zwei Teilgruppen gebildet. Ein Team legte dabei den Fokus der Arbeit auf den Entwurf des Agenten-Gerüstes, die restlichen Teilnehmer konzentrierten sich auf die Festlegung der Infrastruktur für das Agentensystem. Hierbei wurde auch im Hinblick auf das zweite Semester eine Kommunikations-Infrastruktur entwickelt, die auf Blackboard-Basis Informationsaustausch ermöglichen sollte. Durch ein derartiges Blackboard können Informationen von allen Agenten veröffentlicht und von berechtigten Agenten (den jeweiligen Empfängern) gelesen werden. Eine wichtige Forderung an die Umwelt war die Verwaltung von Ressourcen, im Besonderen Geld, Mitarbeiter, Räumlichkeiten, Aufgaben und Drittmittel. Weiterhin sollte die Umwelt die Fähigkeit haben, soziale Strukturen bzw. Hierarchien von Agenten abzubilden als auch Rahmenbedingungen definierbar zu machen.

Insbesondere über das einzusetzende Zeitmodell ergab sich eine rege Diskussion. Die Gruppe, die den Fokus auf die Umwelt gelegt hatte, ging von einer fortlaufenden (Echt-)Zeit aus, die systemintern gezählt wird und kontinuierlich fortschreitet, auch wenn keine Handlungen ausgeführt werden oder Kommunikationen stattfinden. Dieser Ansatz soll Aktionen ermöglichen, deren Auswirkungen erst in der Zukunft wirksam werden bzw. die Akteure in einen bestimmten Zustand versetzen, über den in der Umwelt Informationen zu jedem Zeitpunkt vorliegen müssen. Im Rahmen des Hochschulszenarios wäre dies z.B. die (komplexe, aber hier vereinfacht dargestellte) Handlung Diplomarbeit schreiben. Das Schreiben einer Diplomarbeit stellt eine unterbrechbare, also nicht atomare Tärigkeit dar. Ein Agent kann sich also für das Schreiben einer Diplomarbeit entscheiden, dieses unterbrechen (z.B. um Geld zu verdienen) und später wiederaufnehmen. Die Entscheidung für die aktuelle Handlung kann also zu jedem Zeitpunkt getroffen werden. Der Nachteil dieses Ansatzes ist hingegen, dass das System permanent die Zeit mitzählt, obwohl über lange Zeitspannen hinweg möglicherweise dazu keine Notwendigkeit bestünde. Dies ist besonders im Zusammenhang mit der Kommunikation der Agenten mit der Umwelt relevant: Zu dieser Zeit wurde auch die Möglichkeit nicht ausgeschlossen, die Umwelt in jedem Zeitschritt alle Agenten ansprechen zu lassen, um sie mit neuen Umweltbildern zu

versorgen. Bei diesem Ansatz besteht jeoch die Gefahr eines enormen Overheads aus verschickten Nachrichten, die keine zusätzlichen Informationsgehalt für die Agenten hätten. Ein von Seiten der Betreuer vorgeschlagenes Zeitmodell, welches sich für die Realisierung von zeitdiskreten Simulationen bewährt hat, kann diesen Nachteil verhindern: Die Systemzeit soll dabei nicht mehr als absoluter Indikator dienen, sondern vielmehr anhand von aufeinanderfolgenden Events definiert werden. Es ergibt sich eine Art Liste, in die die andauernden Handlungen und die dadurch ausgelösten Events relativ zueinander eingetragen werden. Die Umwelt, die diese Liste verwaltet, arbeitet sie Schritt für Schritt ab, wobei die Schritte nun nicht mehr atomaren Zeiteinheiten entsprechen, sondern abstrakter als Übergang zwischen Events verstanden werden. Diese Modellierung wirft weitere Fragen nach der Realisierung von andauernden Handlungen auf (vgl. Diplomarbeit schreiben), aber sie vermeidet den oben angesprochenen Overhead, da Nachrichten nur dann verschickt werden, wenn die Notwendigkeit dazu besteht.

Die Integration eines Gedächtnisses sowie die Abbildung von planbasiertem Handeln wurde für die erste Entwicklungsstufe aus Komplexitätsgründen zunächst abgelehnt.

3.2 Vorgehen bei der Realisierung des Multiagentensystems

Nach der im ersten Semester erfolgten Realisierung von autonomen Agenten diente das zweite Semester der Projektgruppe dazu, die für Verhandlungen benötigte Infrastruktur bereitzustellen.

Zu Beginn der Projektgruppe war kaum Vorwissen der Teilnehmer über die diesbezügliche wissenschaftliche Sichtweise in der Soziologie vorhanden. Der soziologische Hintergrund bezüglich der Kommunikation und möglichen Verhandlungsmustern wurde daher in zwei parallel zueinander laufenden Entwicklungssträngen erarbeitet. Zum einen wurden in kleinen Gruppen von den PG-Teilnehmern Vorschläge zum Aufbau und Ablauf von Verhandlungen erarbeitet, während zeitgleich ein Fragenkatalog erstellt wurde, der dem das Projekt begleitenden Soziologen Dr. Thomas Kron aus Hagen vorgelegt werden sollte. Aus den Untergruppen ergaben sich im Wesentlichen zwei unterschiedliche Modelle, die allerdings beide in einem Punkt starke Übereinstimmungen zeigten: Ein Verhandlung ist immer eine abgeschlossene Komponente in der zu entwickelnden Software. Die Verhandlung soll gestartet werden und ein verwertbares Ergebnis zurückliefern. Das erste Modell teilt eine Verhandlung in

kleine Einheiten, an denen maximal zwei Agenten teilnehmen sollten. Jeder Agent würde hierbei mit jedem anderen verhandeln. In jedem dieser Verhandlungsabläufe wird hierbei ein Teil der insgesamt zu verteilenden Objekte auf die beiden Teilnehmer aufgeteilt. So entsteht am Ende der Verhandlung eine Verteilung aller Objekte auf alle Agenten. Das zweite Modell sah die Verhandlung als eine Abfolge zwischen dem Einbringen eines Vorschlags und dem Annehmen oder dem Ablehnen dieses Vorschlages. So wird zufällig ein Agent aus der Gruppe in der Verhandlung ausgewählt, der einen Vorschlag generiert, in dem er die verhandelten Objekte auf alle Agenten verteilt. Anschließend wird dieser Vorschlag allen anderen Agenten übermittelt, die diesen annehmen oder ablehnen können. Sobald eine vom Benutzer vorgegebene Mindestanzahl von Zustimmungen oder eine maximale Anzahl von Runden erreicht worden ist, wird die Verhandlung beendet. Die Antworten zum Fragenkatalog aus Hagen erreichten die Projektgruppe erst zu einem Zeitpunkt, an dem die Vorüberlegungen und Planungen zum größten Teil bereits abgeschlossen waren. Erstaunlicherweise zeigten sich viele Parallelen zu den in der Projektgruppe entwickelten Konzepten und den beschriebenen Theorien von Dr. Kron. Durch diese Übereinstimmung wurde das in der Projektgruppe entwickelte Konzept hinsichtlich seiner Eignung, eine reale Verhandlung abbilden zu können, unterstützt. Die Teilnehmer integrierten in mehrfachen Diskussionen die Anregungen, die sich durch die Beantwortung des Fragenkataloges ergeben hatten. Letztlich wurde die Umsetzung des zweiten Vorschlags bezüglich der Verhandlungsrealisierung auf Grund der höheren Realitätsnähe auch gegen leichten Widerstand seitens eines Betreuers beschlossen.

3.3 Verwendete Tools

In diesem Kapitel werden die benutzten Programme und Werkzeuge kurz beschrieben, die zur Arbeit in der Projektgruppe genutzt werden. Zunächst wurde die benutzte Programmiersprache auf Java festgelegt, da diese allen Projektgruppenteilnehmern bekannt ist und es, durch seine explizite Plattformunabhängigkeit, eine einfache Portierung zwischen verschiedenen Betriebssystemen möglich macht.
Es wurde zusätzlich zu den Standardbibliotheken von Java 1.5.0 eine externe Klassenbibliothek verwendet: Das FuzzyJ Toolkit [FuzJ] bietet eine einfache Möglichkeit, Fuzzy-Logik in Java-Projekte zu integrieren. Hier werden Funktionen zum Anlegen und Verwalten von Fuzzy-Variablen und dem (De-)Fuzzyfizieren angeboten. Der Einsatz dieser Logik wurde vor Allem für eine einfache Eingabemöglichkeit der Regelbasis für die Berechnung der Werterwartungstabelle benötigt (siehe 2.2). So ist es in diesem Bereich möglich, Fuzzy-Variablen für jede Ressource anzulegen, die dann zum Zusammensetzen der Regeln innerhalb der Regelbasis genutzt werden. Der konkrete Einsatz von FuzzyJ wird im Kapitel 4.3.3 näher erläutert.
Die Implementierung wurde zu dem mit Hilfe der Entwicklungsumgebung Eclipse durchgeführt.

Kapitel 4

Entwurf

In diesem Kapitel wird auf die Entwurfs- und Implementierungsdetails eingegangen. So werden die wichtigen Einzelheiten der Umsetzung wie auch die notwendigen Abweichungen von der Anforderungsdefinition erläutert.

Im Folgenden wird die Klassenstruktur der Implementierung vorgestellt. Dazu wird das Programm in seine einzelnen logischen Einheiten (Umwelt, Verhandlung, Agent, GUI) zerlegt und beschrieben, wobei die grafische Oberfläche später erläutert wird (siehe Kapitel B im Anhang).

4.1 Umwelt

Zunächst werden die Implementierungsdetails der Umwelt vorgestellt. An die Beschreibung der zentralen Klassen schließt sich eine Einführung in die entscheidenden Abläufe an.

4.1.1 Klassenbeschreibung

Starter

Diese Klasse wird entweder von der grafischen Oberfläche (siehe Kapitel B) oder direkt von der Konsole aus aufgerufen. Aus speziell für die Simulation angelegten XML- Dateien werden hier die einzelnen Objekte der Simulation erzeugt und die Simulation über die Umwelt (cycleBehaviour()) gestartet.

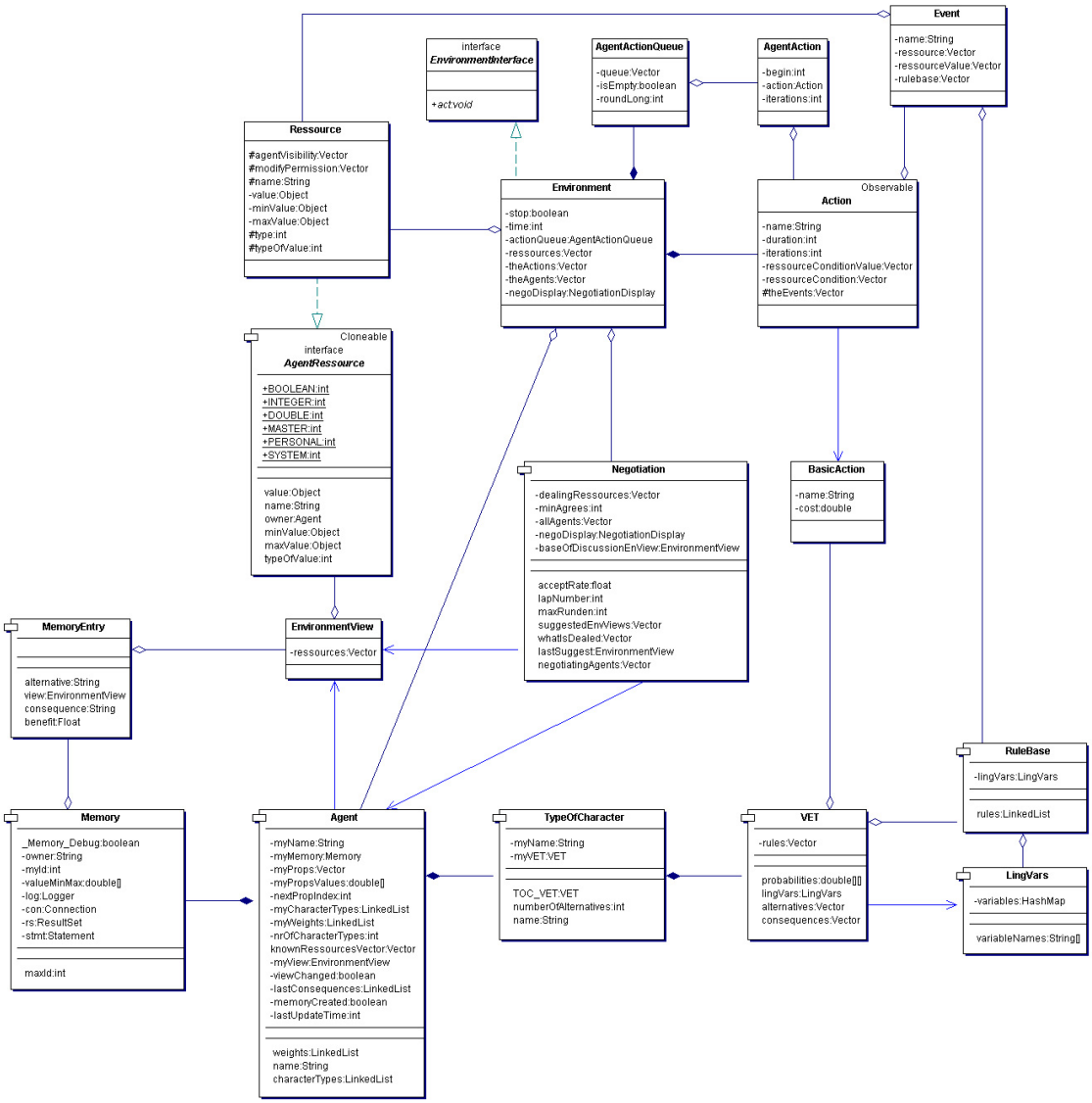

Abbildung 4.1: Das Klassendiagramm

Environment

Diese Klasse stellt die zentrale Verwaltung der Simulation dar: die Umwelt. Hier werden Vektoren von allen Agenten, Ressourcen, Verhandlungen, Aktionen und zugehörigen Events angelegt. Außerdem wird hier die Liste zum Aufrufen der konkreten Aktionen, die AgentActionQueue und die Zeit verwaltet. Das Zentrum der Umwelt ist die AgentActionQueue. In dieser werden alle Aktionen der Simulation chronologisch verwaltet. Aktionen der Agenten und auch Verhandlungen werden hier eingeordnet. Die einzelnen Verhandlungen werden hierzu in speziellen Aktionsklassen verwaltet, den NegotiationActions. Diese werden im Konstruktor der Umwelt zusammen mit einer einleitenden Aktion erzeugt, die vor dem Beginn der eigentlichen Verhandlung aufgerufen wird, um diese anzukündigen. Hierzu wird die boolsche Variable „VerhandlungFindetStatt_*Verhandlungsname*“ benutzt.
Die Umwelt wird über die Methode cycleBehaviour() in einen theoretisch ewigen Zyklus versetzt, indem sie die Aktionen aus der AgentActionQueue ausführt und neue hinzufügt, die von den Agenten stammen. Dieser Zyklus würde theoretisch nur beendet, sobald die AgentActionQueue keine Aktionen mehr aufzuweisen hat. Um die Simulation trotzdem anhalten zu können, wurde der Starter (siehe Kapitel 4.1.1), der die Umwelt aufruft als Thread modelliert, der von außen (z.B. über die GUI) gesteuert werden kann. Die Umwelt setzt außerdem das im Kapitel 2.3 beschriebene Zeitmodell um. Die Umwelt passt nach jeder Aktion die Zeit entsprechend der für diese Aktion benötigten Zeitspanne an. Überprüfung der Zugriffsrechte und Veränderung der einzelnen Ressourcen werden auch von der Umwelt angestoßen. Zu dem wird hier die Kommunikation mit den Agenten über die Umweltbilder (EnvironmentView) eingeleitet. Die Agenten erhalten nur dann ein Umweltbild, wenn sich eine Ressource verändert hat, auf die sie das Sichtrecht besitzen.

Ressource

Die Ressourcen beschreiben den aktuellen Zustand der Umwelt. Ressourcen haben grundsätzlich Namen und Werte mit entsprechenden Grenzen für den minimalen und maximalen Wert. Die Werte können vom Typ Boolean, Integer oder Double sein, wobei sich das System beliebig erweitern lässt und auch selbstdefinierte Objekte zulässt. Jede Ressource wird in eine von drei Kategorien eingeteilt: MASTER, PERSONAL oder SYSTEM.
Master-Ressourcen sind nur abstrakte Typen von Ressourcen. Sie dienen in den Aktionen und Regelbasen als Platzhalter für die Personal-Ressourcen, um nicht für jeden Agenten eine eigene Umwelt-Aktion und einen eigenen Charaktertypen mit den persönlichen Reglbasen anlegen zu müssen. So gibt es also zu jeder Master-

Ressource für jeden Agenten eine Personal-Ressource, die den eigentlichen Wert der Ressource enthält. (z.B. muss die Aktion „GeldAusgeben“ nicht für jeden Agenten neu erschaffen werden, da in ihr nur die Master-Ressource angegeben wird, aus der dann die korrespondierende Personal-Ressource bestimmt werden kann.)
Personal-Ressourcen haben einen Agenten als Besitzer (owner) und die übergeordnete Master-Ressource unter ihren Attributen.
System-Ressourcen schließlich sind Ressourcen, die keinen Besitzer haben und so allgemein der Umwelt zugeordnet werden. Zu jeder Ressource können Sicht- und Änderungsrechte angegeben werden. Diese Rechte werden in zwei Vektoren verwaltet, welche die zugehörigen Agenten enthalten. Um Ressourcen über ein Umweltbild (siehe Kapitel 4.1.1) einem Agenten zukommen zu lassen, gibt es noch eine abgeschwächte Ressourcenversion, das Interface AgentRessource. Hier können lediglich der Name und der aktuelle Wert abgefragt werden, um zu gewährleisten, dass nicht zu viele Informationen an den einzelnen Agenten preisgegeben werden.

Action

Diese Klasse stellt die Möglichkeit der Agenten dar, auf die Umwelt einzuwirken. Eine Aktion kann nur dann ausgeführt werden, wenn bestimmte Bedingungen erfüllt sind. Dies bedeutet, dass eine bestimmte Menge einer Ressource vor Ausführung der Aktion vorhanden sein muss und somit auch wirklich für die Aktion aufgewendet werden kann (so erfordert z.B. die Aktion “Studieren“ ein finanzielles Mindestpolster). Dazu muss der Agent außerdem die entsprechenden Rechte besitzen (siehe Kapitel 4.1.1). Diese Bedingungen werden in zwei Vektoren abgespeichert. In ressourceCondition werden die Ressourcen verwaltet, die vorhanden sein müssen und in ressourceConditionValue die Änderung der Werte dieser Ressourcen.
Aktionen haben neben einem Namen auch noch eine Dauer Duration und eine Wiederholungsanzahl Iterations. Die Dauer gibt den Zeitpunkt an, nachdem die Aktion aus der AgentActionQueue der Umwelt genommen wird, um ausgeführt zu werden. Beispiel: Ein Agent möchte eine Aktion mit Dauer 3 zum Zeitpunkt 6 ausführen. Die Umwelt nimmt diese Aktion dann zum Startzeitpunkt 9 von der AgentActionQueue und führt sie aus.
Die Iterations wiederum geben an, wie oft die Aktion hintereinander ausgeführt werden soll, wobei die Dauer die Abstände zwischen den Iterationen angibt. So können zum Beispiel jährlich wiederkehrende Verhandlungen erzeugt werden. Die Auswirkungen einer Aktion werden schließlich über einen Vektor von Events (siehe Kapitel 4.1.1) angegeben. Zusätzlich zu den normalen Aktionen gibt es noch eine weitere Aktionenmenge, die NegotiationActions. Hier werden die Verhandlungen des Szena-

rios abgespeichert und somit zusätzlich noch die minimale Anzahl an Stimmen, der Beginn und die maximale Rundenanzahl angegeben.

Event

Hier werden die Auswirkungen einer Aktion auf die Umwelt gespeichert und berechnet. Ein Event kann verschiedene Ressourcen verändern, die in dem Vektor ressource abgespeichert sind. Zu jeder dieser Ressourcen korrespondiert der entsprechende Eintrag in zwei weiteren Vektoren: rulebase und ressourceValue. Über die Methode computeConsequences werden mit Hilfe der Regelbasen (siehe Kapitel 4.3.3) die Veränderungen der jeweiligen Ressource berechnet.
Regel-Beispiel für Bankzinsen:

- "wenn VIEL GELD dann VIEL ZINSEN"
- „wenn WENIG GELD dann WENIG ZINSEN"

Die Ergebnisse dieser Berechnung werden dann in dem Vektor ressourceValue abgespeichert.

EnvironmentView

Diese Klasse stellt die Sicht des Agenten auf die Umwelt (d.h. ihre Ressourcen) dar. Sie enthält einen Vektor bestehend aus AgentRessources (siehe Kapitel 4.1.1), der die Ressourcen enthält, auf die der spezielle Agent Sichtrechte besitzt.
Die enthaltenen Werte müssen nicht mit denen der Umwelt übereinstimmen, wodurch das Prinzip des unvollständigen oder auch falschen Bildes der Welt eines Agenten umgesetzt wird.

4.1.2 Grundlegender Ablauf

Im Folgenden wird der Simulationsablauf aus Sicht der Umwelt dargestellt.
Zu Beginn sortiert die Umwelt die vom Benutzer vorgegebenen Verhandlungen als Aktionen (siehe Kapitel 4.1.1) in die AgentActionQueue ein. Nun wird das erste Umweltbildbild (siehe Kapitel 4.1.1) an alle Agenten der Reihe nach gesendet und diese damit zum Handeln aufgefordert. Handeln die Agenten, so fügt die Umwelt die gewünschten Aktionen ebenfalls in die AgentActionQueue ein. Nun beginnt ein Kreislauf, der nur dann beendet wird, wenn die Umwelt keine Aktionen mehr auszuführen hat.
Die erste Aktion wird aus der Queue genommen, überprüft und die Simulationszeit

wird auf die Anfangszeit der Aktion gesetzt. Handelt es sich um eine reguläre Aktion (siehe Kapitel 4.1.1), so werden zuerst die Bedingungen überprüft. Das heißt, sind genug Ressourcen vorhanden, um die Aktion auszuführen und darf der Agent diese Ressourcen überhaupt verändern? Ist dieses bejaht worden, so werden die einzelnen Events (siehe Kapitel 4.1.1) der Aktion berechnet und ausgeführt. Zum Schluss bekommen noch alle Agenten, für die sich etwas an den Ressourcen geändert hat, ein neues Umweltbild zugesendet und können erneut handeln.
Ist die Aktion allerdings eine Verhandlung, so wird erst die entsprechende Ressource „VerhandlungFindetStatt_*Verhandlungsname*“ auf *„TRUE“* gesetzt und die Agenten bekommen Zeit, sich für die Verhandlung anzumelden. Nach drei Runden startet dann die Verhandlung (siehe Kapitel 4.2) mit allen Agenten, die sich angemeldet haben und auch teilnehmen dürfen. Nach der Verhandlung werden die Ressourcen angepasst und der Kreislauf beginnt erneut.

4.2 Verhandlung

Da unsere Agenten über Ressourcen verhandeln sollen, ist es erforderlich eine Kommunikationsinfrastruktur hierfür bereitzustellen. Diese Aufgabe erledigt die Klasse Negotiation. Dementsprechend besteht eine Verhandlung aus folgenden Hauptelementen:

- Verhandelnden Agenten, die bestimmte Ressourcen verteilen sollen.
- Ressourcen, über die zu verhandeln ist. Diese entsprechen den persönlichen Ressourcen.
- System-Ressourcen (also vom Typ NegotiationRessources'), wobei eine System-Ressource die verhandelnden Agenten darüber informiert, wie viel von einer Ressource zur Verfügung steht.
- der Anzahl der Runden, die die maximale Dauer einer Verhandlung festlegt. Dies ist notwendig, weil die Teilnehmer einer Verhandlung nicht zwangsläufig zu einer Einigung kommen.

Sobald die Umwelt eine Verhandlung erzeugt, lässt sie diese durch den Aufruf der Methode startNegotiation() laufen. startNegotiation() teilt der Umwelt das Scheitern der Verhandlung durch Zurückgeben des Wertes „null“ mit. Wenn die Verhandlung erfolgreich abgeschlossen ist, dann liefert startNegotiation() an die Umwelt ein Objekt vom Typ EnvironmentView, das die Verteilung der System-Ressourcen an die

persönlichen Ressourcen, also das Ergebnis der Verhandlung, kapselt. Der Verlauf einer Verhandlung sieht folgendermaßen aus:
Es wird zuerst von jedem der an der Verhandlung beteiligten Agenten ein initialer Vorschlag angefordert, wobei ein Vorschlag durch ein Objekt vom Typ EnvironmentView repräsentiert wird. Ein Agent liefert einen initialen Vorschlag durch den Aufruf seiner Methode makeFirstProb(), die diesen basierend auf dem aktuellen Umwelt-Zustand generiert. Falls ein Agent die initialen Vorschlägen in seinen Entscheidungen im Verlauf der Verhandlung berücksichtigen will, steht ihm die Methode getSuggestedEnvViews() zur Verfügung. Dieses Vorgehen simuliert den einer realen Verhandlung meist vorausgehenden "Smalltalk", mit dem sich die Teilnehmer ein Bild über die Meinungslage verschaffen. Nach der Sammlung aller initialen Vorschläge beginnt ein iterativer Prozess, der solange wiederholt wird, bis eine Einigung erzielt werden konnte oder die Verhandlung auf Grund der erreichten maximalen Anzahl an Runden abgebrochen wird. In jeder Iteration wird gleichverteilt zufällig ein Agent ausgewählt, einen Vorschlag für die Verteilung der zu verhandelnden Ressourcen zu unterbreiten. Dies erfolgt über die Methode makeProp() des Agenten. Hierbei wird sichergestellt, dass innerhalb einer Verhandlungsrunde ein Agent nicht mehrfach zur Vorschlagsabgabe aufgefordert wird. Der angesprochene Agent berechnet einen Vorschlag und liefert diesen als Umweltbild (EnvironmentView) zurück. Alle anderen an der Verhandlung beteiligten Agenten werden nun aufgefordert, über diesen Vorschlag abzustimmen. Hierbei kann von jedem Agenten der Vorschlag angenommen oder abgelehnt werden. Die Funktion der Bewertung von Vorschlägen wird im Agenten durch die Methode getOpinion() realisiert und basiert ebenso wie die Vorschlagsgenerierung auf der Werterwartungstabelle der Charaktertypen des Agenten.
Da Agenten auf Grund von Sichtbarkeitsbeschränkungen selber kein Umweltbild erzeugen können, werden Vorschläge als Vektor von Ressourcen zurückgegeben. Die Verhandlungs-Klasse erzeugt dann aus diesem Vektor das korrespondierende Umweltbild.

4.3 Agent

Der Agent verantwortet und realisiert den Entscheidungsprozess. Er erhält über updateView() die für die Entscheidungsfindung nötigen Informationen, veranlasst dann die Entscheidungsfindung, und gibt schließlich in reactOnViewChanged() die Aktion, für die er sich entschieden hat, an die Umwelt zurück.
Die zentrale Rolle bei der Entscheidungsfindung spielt das Umweltbild des Agenten, welches als Vektor von Ressourcen in myView abgespeichert wird. Der Agent aktualisiert die darin enthaltenen Informationen mit jedem neuen Umweltbild.

Beim Aktualisieren des gespeicherten Umweltbildes wird in der Methode compareView() das alte Umweltbild mit dem neuen abgeglichen. An dieser Stelle würde das Gedächtnis ansetzen (siehe Kapitel 5.2).
Schließlich ruft der Agent seine decide()-Methode auf. Hierbei wird das vervollständigte Umweltbild durch den Aufruf der Methode evaluateAlternatives() nacheinander an die einzelnen Charaktertypen übergeben. Diese bewerten unabhängig alle zur Verfügung stehenden Aktionen. Auf Basis dieser Bewertungen der Charaktertypen kann der Agent entsprechend deren Gewichtungen eine kumulierte Werterwartung für jede zur Verfügung stehende Aktion berechnen (dabei wird die Methode cumulate() verwendet). Die vielversprechenste Handlung (mit höchster Werterwartung) hat die höchste Wahrscheinlichkeit, ausgewählt zu werden (nichtdeterministische, fitnessproportionale Aktionen-Auswahl).

Neben der Realisierung des zuvor erörterten Werterwartungsmodells (siehe Kapitel 2.2.1) im Agenten ist das Ziel ebenfalls, ein möglichst intuitives bzw. realitätsnahes Verhalten im Agenten zu implementieren. Weitgehend kann dieses Verhalten vom Benutzer über Regeln erzeugt werden. Jedoch gibt es einige feste Verhaltensmuster, wie zum Beispiel das Verhandlungsverhalten. Dazu wurden folgende Annahmen über den realen Agenten gemacht:

- Aus dem Pool seiner Vorschläge zu einer Verhandlung bringt ein Agent zunächst die für ihn besten Vorschläge vor
- Ein Agent stimmt einem Vorschlag seines Gegenüber genau dann zu, wenn er selbst keinen für ihn besseren Vorschlag mehr vorzubringen hat
- Ein Agent handelt stets nach der Beschaffenheit seines Charakters bzw. seiner Charaktertypen

4.3.1 Agent in Verhandlung

Befindet sich der Agent in einer Verhandlung, so wird er von der Klasse Negotiation angestoßen (siehe Kapitel 4.2). Dies geschieht mit der Methode makeFirstProp(). Die Vehandlung übergibt dabei ihr eigenes Objekt an den Agenten, damit er alle nötigen Informationen zur Erstellung eines Vorschlags zur Verfügung hat. Daraufhin berechnet der Agent Vorschläge, die er im weiteren Verlauf der Verhandlung vorbringen kann. Alle Vorschläge haben die Form eines neuen Umweltbildes und basieren auf dem Umweltbild, welches zu Beginn der Verhandlung aktuell war (baseOfDiscussionEnView).

Die neuen Umweltbilder (Vorschläge) werden durch Änderung der einzelnen Ressourcen der teilnemenden Agenten generiert. Dabei betrachtet der Agent nur die Ressourcen, welche in der Verhandlung als „verhandelbar“ gekennzeichnet sind (z.B. Geld). Zunächst wird der gesamte verhandelbare Betrag einer Ressource auf die Teilnehmer gleichverteilt. Der Agent ändert nun den ihm zughörigen Anteil der Ressource (z.B. Geld_Agent1) um einen gewissen Prozentsatz, beispielsweise zwischen -40% und +40% vom Gleichanteil ($= \frac{\text{Groesse der Ressource}}{\text{Anzahl der Agenten}}$) und verteilt den übrigen Betrag auf die anderen Agenten. Dabei verringert sich die Schrittweite mit steigender Anzahl der Vorschläge, die der Agent generieren soll. Hierfür wird zunächst die maximale Rundenzahl der Verhandlung gewählt. Da es auch möglich ist, dass mehrere Ressourcen innerhalb einer Verhandlung verteilt werden sollen, findet die Verteilung der Ressourcen für einen Vorschlag in mehreren Rekursionsstufen statt. (Methode rekProp()). Siehe hierzu Abbildung 4.2.

Anschließend werden die so veränderten Verhandlungs-Ressourcen mit dem Rest des zum Verhandlungsstart aktuellen Umweltbildes zu einem neuen Umweltbild zusammengesetzt, sodass der Agent zu Beginn genügend Vorschläge für die Verhandlung zur Verfügung hat. Diese Vorschläge werden nun noch nach der Reihenfolge sortiert, in der der Agent sie abgeben soll. Für den Agenten ist es sinnvoll, seine für ihn besten Vorschläge zuerst zu machen, in der Hoffnung, sie könnten angenommen werden. Je mehr Runden in der Verhandlung gelaufen sind, um so nachgiebiger wird er. Dieses Verhalten ist nicht nur aus Sicht des Agenten sinnvoll, sondern ist auch realitätsnah. Die Vorschläge werden bezüglich der Werterwartung der VETs seiner Charaktertypen sortiert. Für jeden Vorschlag wird hierfür der Wert der „besten“ Aktion berechnet. Da Vorschläge aus den Umweltbilden bestehen, die nach der Verhandlung vorzufinden wären, kann hierfür die Methode evaluateAlternatives() benutzt werden. Details zur Bewertung im Uni-Szenario finden sich im Kapitel 6.2.

Im Folgenden beginnt die Verhandlung und wird den Agenten später nach einem Vorschlag fragen oder aber seine Meinung (in Form von Zustimmung oder Ablehnung) zu einem Vorschlag eines anderen Agenten wissen wollen. Um den nächsten Vorschlag des Agenten zu bekommen, ruft die Verhandlung die Methode makeProp() auf und bekommt lediglich das erste Element der sortierten Umweltbilder geliefert. Anschließend löscht der Agent den Vorschlag aus seinem Vektor und liefert beim nächsten Aufruf seine zweitbeste Variante.

Wird der Agent zur Abstimmung über einen fremenden Vorschlag befragt (Methode getOpinion()), so sagt er entweder „ja“ oder „nein“. Als erstes muss der Agent selbstverständlich den Vorschlag, wie zuvor seine eigenen, mittels seiner Werterwartungstabelle bewerten. Stehen ihm noch „bessere“ Vorschläge in seinem Vektor zur Verfügung lehnt er ab. Ist dies nicht der Fall lautet seine Anwort an die Verhandlung

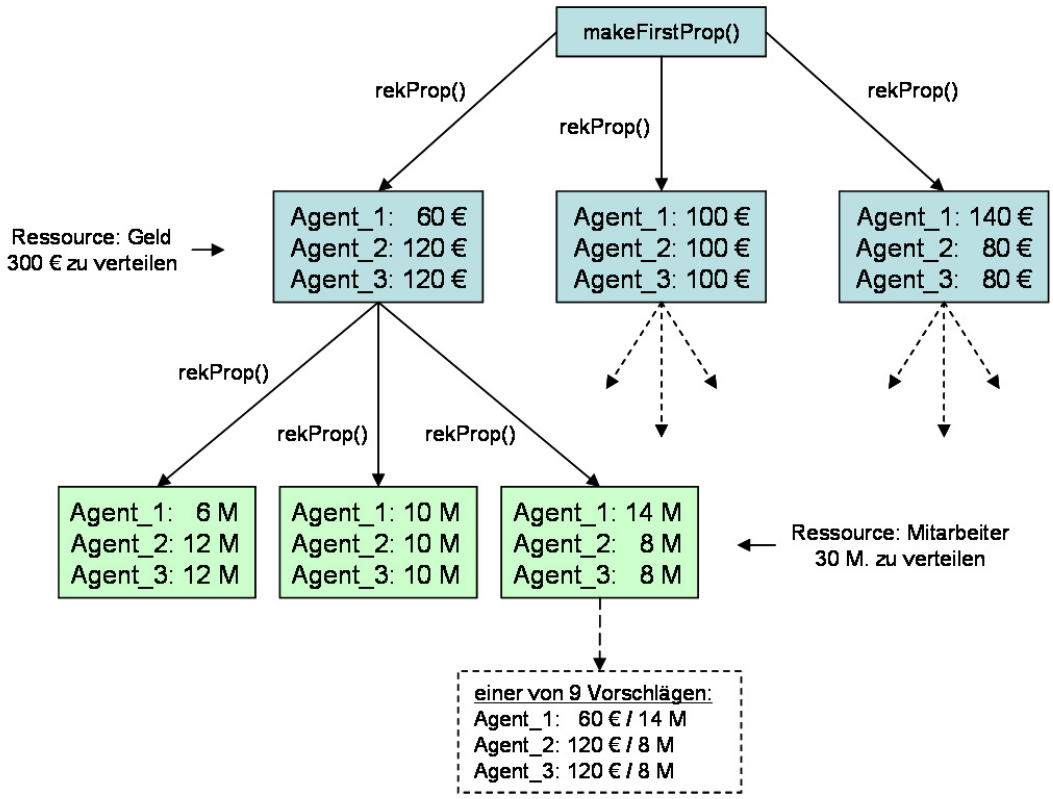

Abbildung 4.2: Generierung der Vorschläge eines Agenten zu Beginn der Verhandlung (3 Agenten verhandeln um 2 Ressourcen)

„Ja“. Diese Vorgehensweise ist recht inuitiv, da ein Agent hoffen kann, später noch einen besseren Vorschlag machen zu können.

4.3.2 Charaktertyp und Werterwartungstabelle

In erster Linie ist die Werterwartungstabelle (*VET - Value Expectation Table*) eine Datentabelle, welche die Grundlage für die Entscheidungsfindung des Agenten darstellt. In ihr werden nach dem soziologischen Konzept der Werterwartungstheorie Zutreffenswahrscheinlichkeiten für jede Kombination von Handlungsalternativen (Action) und Handlungsfolgen (Consequence) abgelegt. Dabei sagt jeder dieser Werte aus, wie groß die Wahrscheinlichkeit des Eintreffens einer Handlungsfolge aufgrund der Ausführung einer Alternative in den Augen des Charaktertypen ist. Der zugehörige Wert ist jedoch nicht fest, wie im soziologischen Werterwartungsmodell (siehe Kapitel 2.2.1). Er wird mit Hilfe einer Fuzzy-Regelbasis aufgrund der bestehenden Umweltsituation dynamisch berechnet (siehe Kapitel 4.3.3).
Ebenfalls hier gespeichert sind die Bewertungen der Alternativen und Folgen, die für die Entscheidungsfindung des Agenten von großer Bedeutung sind und die Aufgabe haben, unterschiedliche Zielsetzungen der Charaktertypen zum Ausdruck zu bringen. Alternativen haben jeweils einen Kostenwert, der ausdrückt, wie aufwendig die Aktion für den jeweiligen Charktertypen ist. Der Nutzenwert stellt die subjektiven Bewertung einer Folge durch den CT dar.
Der Charaktertyp initiiert vor jeder Benutzung der VET durch die Methode evaluateAlternatives() die erneute Berechnung der einzelnen Tabellen-Einträge. Dafür wird eine Schleife über alle Alternativen und alle Folgen verwendet und jeweils die Methode calculate() mit dem aktuellen Umweltbild aufgerufen. Nachdem die VET aktualisiert wurde, berechnet der Charaktertyp auf Basis der Wahrscheinlichkeiten sowie seinen Kosten und Nutzen die Wertwerterwartungen für alle zur Verfügung stehenden Aktionen. Dabei werden für jede Aktion die Nutzenwerte der Folgen mit den zugehörigen Wahrscheinlichkeiten multipliziert, aufsummiert und anschließend um die Kosten der Aktion verringert. Der Strukturelle Aufbau der VET ist in Abbildung 4.3 detailiert dargestellt. Es wird ebenfall deutlich, dass sich hinter jedem Wert eine Fuzzy-Regelbasis versteckt.
Um die Wahrscheinlichkeits-Tabelle zu erstellen, liefert der Charaktertyp alle BasicActions und Consequences, die für den Agenten zur Verfügung stehen und alle linguistischen Variablen (Klasse LingVars) für die Fuzzy-Regelbasis. Die LingVars sind eine Repräsentation aller Umwelt-Ressourcen durch linguistische Terme. Dieses Objekt wird hier lediglich hinterlegt, um später in jeder RuleBase verwendet zu werden.

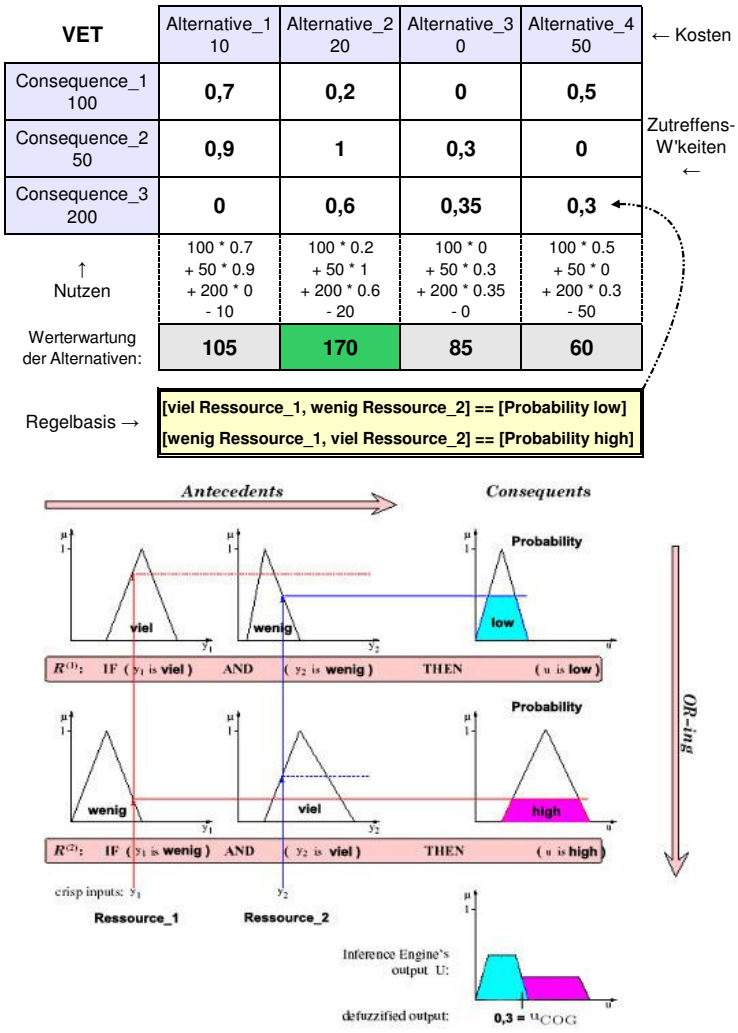

Abbildung 4.3: Schaubild zur Werterwartungstabelle

Daraufhin wird für jeden Tabelleneintrag des zweidimensionalen Vektors eine leere Regelbasis angelegt. Die Tabelle lässt sich im Folgenden um Alternativen und Folgen beliebig erweitern bzw. reduzieren. Die neu angelegten Regelbasen lassen sich mit Hilfe der Methoden addRule() und removeRule() konfigurieren. Die einzelnen Regeln werden in der GUI zusammengesetzt und hier lediglich als ein Objekt der Klasse FuzzyRule weitergegeben. Ist die VET soweit vollständig, kann die Simulation beginnen.
Zur Berechnung der VET ruft der besitzende Charaktertyp die calculate()-Methode auf. Die Aufgabe der Werterwartungstabelle besteht nun darin, jeden Eintrag zu aktualisieren. Dazu wird grundsätzlich einmal jede Regelbasis mit dem weitergeleiteten Umweltbild aufgerufen. Diese liefern dann die neuen $(a * f)$ Eintretenswahrscheinlichkeiten.
Nach dem Durchlauf durch die gesamte Tabelle, liegen die Ergebnisse der Fuzzy-Regeln in Form von Wahrscheinlichkeitswerten vor und somit hat der zugehörige Charaktertyp nun die Möglichkeit, die Werterwartungstabelle auszuwerten und seine subjektiven Bewertungen für alle Aktionen an den Agenten (Kapitel 4.3) zurückzuliefern.

4.3.3 Die Regelbasis

Eine Designentscheidung des Entwurfs ist die Verwendung einer Regelbasis für die Bestimmung der Einträge der Werterwartungstabelle (VET). Dabei soll jeder Eintrag der Tabelle durch eine Regelmenge beschrieben werden, deren Auswertung eine konkrete Wahrscheinlichkeit liefert. Die Regeln sollen abhängig vom aktuellen Umweltbild (also von dem aktuellen Zustand der Umwelt) sein.
Um die Funktionalität der VET von der Funktionalität der Regelbasis zu trennen, existiert die Klasse RuleBase. Sie besitzt alle benötigten Funktionen, eine Regelmenge zu erstellen und diese aufgrund eines Umweltbildes auszuwerten. Dabei wird ein rationaler Wert (double) zurückgeliefert, der im Fall der Werterwartungstabelle einen Wertebereich zwischen 0 und 1 hat. Da die Regelbasis auch für die Berechnung der Ressourcenänderung in der Umwelt benutzt werden soll, können auch Werte außerhalb des Wertebereichs zwischen 0 und 1 zurückgeliefert werden. Eine Möglichkeit ist z.B., mit Hilfe der Regelbasis die Veränderung der Ressource *Geld* eines Lehrstuhls in Abhängigkeit von den ihm zur Verfügung gestellten finanziellen Mitteln zu beschreiben. Dann soll die Regelbasis den Wert liefern, um den sich die Ressource *Geld* verändert. Diese Änderung bewegt sich meist außerhalb des Intervalls $[0, 1]$.
Zur Trennung zwischen der Auswertung für die VET und der Umwelt werden zwei unterschiedliche Methoden zur Berechnung verwendet, da sich beide Fälle in Details

unterscheiden und somit unterschiedlich gehandhabt werden müssen. Im Folgenden werden die grundlegenden Konzepte der RuleBase erläutert:

Verwaltung der linguistischen Variablen

Eine Regelbasis verwendet üblicherweise linguistische Variablen, wobei jede linguistische Variable eine Eingabe repräsentiert. Ein Beispiel für eine linguistische Variable ist die Temperatur. Jede linguistische Variable besteht aus linguistischen Termen, die umgangssprachliche Namen für mögliche Werte der linguistischen Variable haben. Um bei dem Beispiel zu bleiben, sind die Adjektive *kalt* oder *heiß* mögliche Terme der Variable *Temperatur*.
Da in unterschiedlichen Regeln oft die gleichen Variablen verwendet werden, war das Ziel, alle linguistischen Variablen in einer Datenstruktur zu kapseln. Dabei wird zum einen weniger Speicher benötigt und zum anderen die Laufzeit verkürzt, da die Variablen nicht innerhalb einer RuleBase doppelt erzeugt werden müssen. Diese Datenstruktur ist die Klasse LingVars, die sich in dem Package tools befindet. Sie besteht aus einer Menge von Objekten der Klasse FuzzyVariable, die von dem Toolkit *FuzzyJ* importiert wird. Damit der Zugriff auf eine einzelne Variable über ihren Namen (also z.B. Temperatur) schnell möglich ist, sind die Variablen intern als HashMap abgspeichert. Eine Besonderheit ist die Variable MemoryDegree. Diese Variable ist in jedem LingVars-Objekt enthalten und modelliert einen Wert zwischen 0 und 1, der die Häufigkeit der betrachteten Aktion im Verhältnis zur Anzahl der zu dem aktuellen Umweltbild passenden Gedächtniseinträge angibt. Auf diese Weise kann in den Regeln formuliert werden, wie oft die aktuelle Handlungsalternative in der aktuellen Situation schon durchgeführt wurde. Zur konkreten Berechnung des MemoryDegree wird auf das Gedächtnis des Agenten zurückgegriffen. Damit diese Variable in allen Szenarien zur Verfügung steht, wurde sie fest in die RuleBase integriert und muss nicht durch externe Konfigurationsdateien definiert werden. Die per Default vorgegebene Definition der linguistischen Terme lässt sich jedoch löschen und benutzerdefinierte Terme hinzufügen. Die Defaultdefinition dient nur einer anfangs sinnvoll definierten linguistischen Variable. Die Existenz dieser speziellen Variablen ist besonders für den Charaktertyp *Identitätsbehaupter* interessant, da dieser geneigt ist, eine schon öfters durchgeführte Handlungsalternative ein weiteres Mal durchzuführen. Somit kann man z.B. folgende Voraussetzung einer Regel beschreiben: *Wenn ich diese Handlungsalternative schon oft durchgeführt habe...*
Neben der Speicherung der Variablen enthält die Klasse eine Reihe von Methoden, die die Erstellung und den Zugriff auf die Variablen und Terme erleichtern. Dabei leiten die meisten Methoden die Parameter an die entsprechende Methode von Fuz-

zyJ weiter, um z.B. eine Variable zu erzeugen oder zu löschen, und dienen nur der Kapselung aller Variablen in einem Objekt.

Verwaltung der Regeln

Innerhalb der RuleBase werden die Regeln in einer linearen Liste (rules) abgespeichert, da einerseits jede Regelbasis beliebig viele Regeln haben kann und andererseits zur Auswertung der Regeln jede Regel nacheinander ausgewertet wird (die Reihenfolge spielt keine Rolle). Daher kann man auf einen Direktzugriff verzichten und eine Liste verwenden. Jedes Element der Liste ist ein Objekt der Klasse FuzzyRule, die von FuzzyJ importiert wird. Zur Verwaltung der Regeln stehen die Methoden getRules(), setRules(), addRule() und removeRule() zur Verfügung. Diese Methoden sind selbsterklärend und zum Verständnis des Konzepts nicht notwendig. Daher werden sie nicht näher erläutert.

Auswertung der Regeln

Interessanter sind die Methoden zur Auswertung der Regeln. Wie oben bereits angedeutet, soll die RuleBase sowohl für die Berechnung der Werterwartungstabelle als auch für die Bestimmung der Ressourcenveränderung verwendet werden. Dabei unterscheiden sich die Berechnungen in mehreren Details, weshalb zwei getrennte Methoden (calculateValue()) zur Auswertung implementiert werden, die sich durch unterschiedliche Parameter unterscheiden. Der prinzipielle Ablauf der Auswertung ist in beiden Methoden gleich. Die Methode für die Werterwartungstabelle hat das aktuelle Umweltbild und den MemoryDegree als Parameter, wohingegen die Methode für die Umwelt nur das aktuelle Umweltbild als Parameter hat. Die Auswertung aller Regeln einer Regelbasis läuft in folgenden Schritten ab:

1. Extrahierung aller Ressourcen aus der EnvironmentView (Umweltbild).

2. Fuzzifizierung jeder Ressource mit einem Singleton an der Stelle des aktuellen Ressourcenwertes. Es wird dabei ein FuzzyValue (Klasse von FuzzyJ) erzeugt, der dann zu der HashMap fuzzifiedInputs hinzugefügt wird. Somit kann man eine einmal fuzzifizierte Eingabe in allen Regeln verwenden.

3. Alle Regeln werden nacheinander mit den benötigten Eingaben versorgt. Dabei werden innerhalb einer Regel die Eingaben in der Reihenfolge der Ressourcen in der Vorbedingung hinzugefügt. Für den schnellen Zugriff wird hierbei die HashMap fuzzifiedInputs verwendet.

4. Die erste Regel wird mittels execute() ausgeführt und das Ergebnis in dem FuzzyValue result gespeichert. Zur Auswertung jeder Regel wird dabei der *MamdaniMinMaxMinRuleExecutor* von FuzzyJ benutzt, der in Anlehnung an den Mamdani-Controller den Mamdani-Min-Inferenzoperator und die MaxMin-Komposition verwendet. Bei diesem Verfahren wird jeder Konklusions-Term auf die Höhe der am wenigsten erfüllten Prämisse abgeschnitten. Somit feuert jeder Output maximal mit der Stärke der schwächsten Regel. Diese Eigenschaft entspricht der intuitiven Auswertung einer Regel und wird darum für die RuleBase verwendet.
   Da FuzzyJ nur eine Regel auf einmal auswerten kann, müssen die Ergebnisse der anderen Regeln mit dem Ergebnis der Veroderungen der vorherigen Ergebnisse verodert werden. Also werden die nachfolgenden Regeln nacheinander ausgeführt und mit dem FuzzyValue result verodert.

5. Zum Schluss muss noch der resultierende FuzzyValue defuzzifiziert werden. Hier unterscheiden sich die beiden Methoden zur Auswertung der Regeln (siehe Abbildung 4.4).
   Im Falle der Methode für die VET wird die Momentenmethode (Center-of-Gravity) benutzt, die den Schwerpunkt des FuzzyValues berechnet und den zugehörigen x-Wert zurückliefert. Diese Defuzzifizierungsmethode hat den Vorteil, dass Bereiche mit hoher Zugehörigkeit stärker gewichtet werden und im Falle mehrerer Maxima ein Mittel berechnet wird. Das Berechnen eines Mittels ist im Falle der VET erwünscht, da alle Regeln in Abhängigkeit ihrer Feuerstärke Einfluss auf die berechnete Wahrscheinlichkeit ausüben sollen.
   Im Falle der Methode für die Umwelt wird die Maximummethode verwendet, wobei bei mehreren Maxima das mit dem kleinsten x-Wert bestimmt wird. Da bei den Regeln der Umwelt auch rein boolesche Regeln verwendet werden können (z.B. um zu beschreiben, ob etwas erlaubt ist oder nicht), macht es keinen Sinn eine 0,5 auszugeben, wenn eine Regel bei 0 und eine bei 1 feuert (für boolesche Terme werden Singletons bei 0 und 1 verwendet).

FuzzyJ

Die Fuzzy-Regeln können für jeden Tabelleneintrag einzeln in einer Regelbasis vom Benutzer definiert werden. Dabei sollen die linguistischen Terme aller zur Verfügung stehenden Ressourcen benutzt werden können. Zur Realisierung der Regelbasis wird die Bibliothek von FuzzyJ benutzt, da diese bereits alle nötigen Objekte und Operationen unterstützt. Dabei reduziert sich der hier benötigte Teil auf die Erzeugung

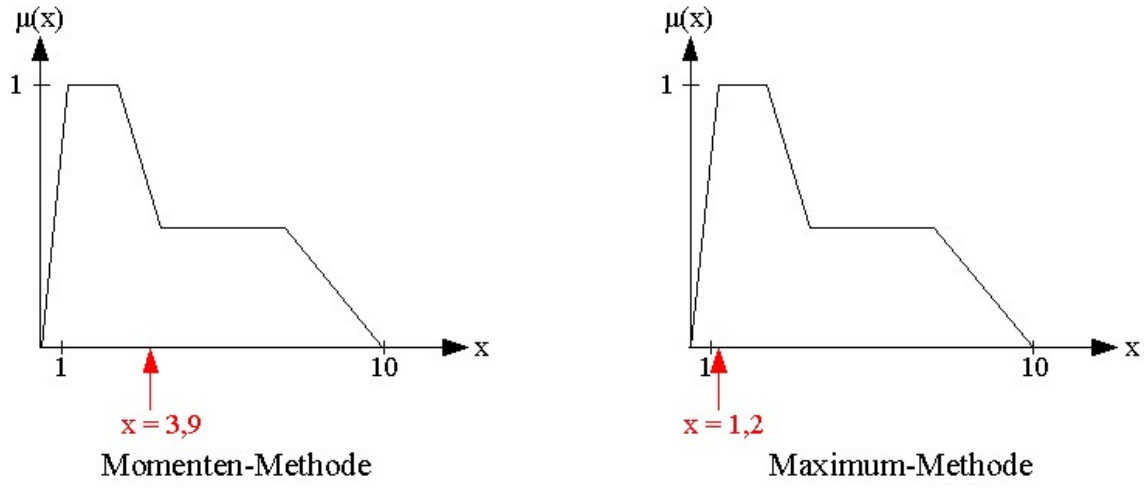

Abbildung 4.4: Vergleich der beiden Defuzzifizierungsmethoden

einer Fuzzy Variablen mit ihren Termen, das Zusammensetzen der Regeln aus konjugierten Vorbedingungen und einer Konklusion (Wahrscheinlichkeitsterm) und die Auswertung der Regeln auf den Ressourcenwerten mit anschließender Defuzzifizierung. Des Weiteren unterstützt FuzzyJ den Testprozess und das Abstimmen der Regeln durch eine graphische Ausgabe von Fuzzy-Werten.

4.4 GUI

Zum Starten der Simulation benötigt das Programm zum einen das Szenario in Form von XML-Dateien und zum anderen den Aufruf des Starters (4.1.1). Beides kann man per Hand erzeugen/starten, falls man ein z.B. ein Batch-Sytem benutzen möchte oder über keine graphische WIMP-Oberfläche verfügt. Die weitaus komfortablere Lösung ist es allerdings, das Graphical User Interface zu verwenden, um einerseits das Szenario zu erstellen und die XML-Dateien anzulegen und andererseits die Simulation zu starten, zu pausieren, zu überwachen und die verschiedenen graphischen Ausgaben zu nutzen (für weiterführende Informationen siehe: Anhang B).

Kapitel 5

Entwurfskorrektur/Probleme

Im Folgenden wird auf Abweichungen vom ursprünglichen Entwurf eingegangen und es werden Probleme, die während der Implementierung auftraten erläutert. Außerdem werden zwei Programmteile angesprochen, die leider nicht zum Einsatz kamen, weil sie aus zeitlichen Gründen nicht umgesetzt wurden („Lernverhalten“), oder der Einsatz vorher nicht bis zum Schluss durchdacht wurde („Gedächtnis“) und das Programm in Teilen anders umgesetzt werden musste als geplant.

5.1 Struktur der Ressourcen

Eine entscheidende Entwurfsumstellung fand im zweiten Semester in Bezug auf die Ressourcen statt. Im ersten Semester wurde eine Typisierung von Ressourcen nicht vorgenommen. Das bedeutet, dass es nicht die Möglichkeit gab die Ressourcen der einzelnen Agenten zusammenzufassen (z.B. Geld_a1 und Geld_a2 zu Geld). Daraus resultierte, dass man eine unnötig große Menge an Aktionen anlegen musste um einzelne Ressourcen zu verändern (z.B. GeldAusgeben_a1, GeldAusgeben_a2 und nicht GeldAusgeben). Außerdem konnte man nicht eine Menge von verschiedenen Charaktertypen (siehe Kapitel 4.3.2) erzeugen und auf alle Agenten anwenden, da für die verschiedenen Agenten unterschiedliche Ressourcen und Aktionen zu berücksichtigen waren. Der erste Ansatz bestand darin, in der Umwelt für jeden Agenten eine Liste anzulegen, die angibt, wie die allgemeinen Ressourcen aus den Charaktertypen in die persönlichen Ressourcen zu übersetzten sind. Der Vorteil dieser Lösung war, dass man nun die Ressourcen beliebig setzten konnte, also nicht nur die eigenen Ressourcen in den Regeln vorkommen konnten, sondern auch die eines speziellen Konkurrenten. Die Lösung wurde anfänglich umgesetzt, dann aber gegen einen weniger aufwändigen und praktikableren Ansatz eingetauscht, welcher letzlich auch endgültig beibehalten

wurde. Hierbei werden die Ressourcen in 3 Typen unterteilt (siehe Kapitel 4.1.1). Bei dieser Lösung bleibt erstmal der Nachteil, dass man nur die eigenen (persönlichen) Ressourcen verallgemeinern kann, was sich aber bei unseren Szenarien als durchaus sinnvoll herausstellte. Außerdem ist dieses System ebenfalls leicht erweiterbar, wenn auch nur über den Quelltext und nicht vom Benutzer selbst.

5.2 Das Gedächtnis

Das Prinzip des Gedächtnisses eines Agenten bestand schon seit der ersten Entwurfsphase. Allerdings haben sich während der Implemetierung des Agenten einige Schwächen bezüglich der Einbindung in den Entscheidungsprozess gezeigt. Im Folgenden wird dieses und das Funktionsprinzip des Gedächtinis näher erläutert:
Agenten bekommen von der Umwelt ein Umweltbild übergeben. Dieses besteht aus den für den Agenten sichtbaren Ressourcen, sowie den entsprechenden Werten. Das Umweltbild ist die Basis für die Entscheidungsfindung der Charaktertypen und damit für die Handlung, für die sich der Agent entscheidet. Sobald der Agent eine Entscheidung getroffen hat, speichert er diese, sowie das zugehörige Umweltbild in seinem Gedächtnis. Des Weiteren wird eine Zeitmarke mit abgespeichert. Der Agent hat so die Fähigkeit, auf seine Erinnerungen zurückzugreifen und diese bei folgenden Entscheidungen zu berücksichtigen. Hierzu stehen ihm Methoden zur Verfügung, die es z.B. auch ermöglichen, auf Basis des aktuellen Umweltbildes eine nach Relevanz geordnete Ausgabe der am besten passenden Erinnerungen zu erhalten (auf Basis einer Hammington-Distanzberechnung). Mit Hilfe des Gedächtnisses wäre der Agent auch in der Lage, tatsächliche Auswirkungen seiner Handlung mit den vermuteten abzugleichen und so neu zu bewerten und dadurch zu lernen.
Hierfür benutzt die RuleBase den Parameter memoryDegree. Wird das Gedächtnis des Agenten mit dem aktuellen Umweltbild aufgerufen, liefert es die Einträge zurück, die mehr oder weniger mit der aktuellen Situation übereinstimmen. Der memoryDegree gibt an, wie oft die gerade betrachtete Alternative im Verhältnis zu den restlichen gewählt wurde. Es ist vorgesehen, dass diese Information später so oder in abgewandelter Form in der Regelbasis verwendet werden kann.
Bei der Initialisierung eines Agenten wird eine Tabelle für sein Gedächtnis angelegt. Hierbei werden die Spalten gemäß seines Umweltbildes erzeugt. Bevor er nun eine Handlung an die Umwelt übergibt, wird diese mitsamt dem zugrunde liegenden Umweltbild und einer Zeitmarke in die Tabelle geschrieben.

Das Gedächtnis und seine Methoden bieten also dem Agenten die Möglichkeit, herauszufinden, welche Aktionen in der Vergangenheit bei ähnlicher Umweltbeschaffen-

heit (verglichen mit dem aktuellen Umweltbild) ausgeführt wurden und mit welchem Grad die damalige Situation der aktuellen ähnelt. Da der Agent aber in der VET neben einer Aktion auch immer die möglichen Handlungsfolgen betrachtet, wäre es wichtig die eingetretene Folge zu der vergangenen Aktion zu kennen. Diese kann weder das Gedächtnis liefern, da sie zum Zeitpunkt der Speicherung nicht bekannt sein kann, noch von Agenten ermittelt werden. Dies hängt mit dem Aufbau des soziologischen Systems in Form der Aktion-Folge-Paare zusammen:
Die Handlungsfolgen der VET sind rein subjektive Erwartungen, die der Agent an eine Aktion stellt und stimmen nicht unbedingt mit den wirklichen Folgen (Ressourcenänderungen), welche durch eine Handlung in der Umwelt herforgerufen werden, überein. In sehr vereinfachten Szenarien ist es eventuell denkbar, dass der Agent aus der für ihn sichtbaren Änderung der Umwelt (Umweltbild) eine direkte Folge seiner zuvor gewählen Aktion schließen kann. Dies ist hier allerdings auf Grund von vielen möglichen handelnden Agenten, einer großen Menge von Ressourcen sowie einer nicht-trivialen Umwelt-Regeln ausgeschlossen.
Somit würde sich die Nutzung des Gedächtnisses darauf beschränken, zu erfahren, welche Aktion in ähnlicher Situation häufig gewählt wurde. Die könnte nützlich sein, um den speziellen Charaktertypen *Identitätsbehaupter* zu realisieren.

Kapitel 6

Anwendung

6.1 Test mit Robocode

Im ersten Test des ABSINTH-Systems stand der einzelne Agent im Vordergrund. Unabhängig von der für das zweite Semester geplanten Interaktion mit der Umwelt sollte erst einmal untersucht werden, inwieweit der Agent durch die Eingabe von Fuzzy-Regeln in den VETs der Charaktertypen zu sinnvollen Handlungen veranlasst werden kann. Im Vorfeld wurde zwar die fehlerfreie Funktion des Codes sichergestellt, aber es fehlte der Nachweis, dass alle Teile des Agenten (Charaktertypen, Konsequenzen und deren Gewichtung) so zusammenspielen, dass mit dem implementierten Modell ein weiteres sinnvolles Arbeiten möglich war. Zu diesem Zweck musste der Agent in ein Szenario eingebettet werden, das einerseits nicht zu kompliziert sein durfte, schliesslich sollten die Handlungen aus den Regeln grob nachvollziehbar sein. Andererseits durfte es auch nicht so einfach bleiben, wie während des Debuggings, da ja gerade das Zusammenspiel der Agentenkomponenten untersucht werden sollte. Obwohl die Funktionen der Umwelt schon programmiert wurden, wurde bei diesem Test zunächst ausser acht gelassen. Also musste ein System verwendet werden, das eine Art Umwelt bereits beinhaltet: Die Wahl fiel auf das Java-Framework Robocode.

6.1.1 Integration in Robocode

Robocode ist eine Javaanwendung von IBM, die Programmieranfängern auf spielerische Weise Java-Praxis vermitteln soll. Dazu wird ein Framework zur Verfügung gestellt, in dem Roboter (Panzer) auf einem Spielfeld gegeneinander antreten und sich mit selbstprogrammierten Strategien bekämpfen. Die Roboter sind mit drehbaren Geschütz- und Radartürmen ausgestattet; ihr Ziel ist es, zu überleben , sich

also möglichst selten treffen zu lassen und die anderen Roboter möglichst stark zu beschädigen. Sie haben jeweils eine bestimmte Energie die sinkt, wenn sie getroffen werden und steigt, wenn sie selbst einen anderen Panzer treffen. Ein Kampf geht über eine einstellbare Anzahl von Runden und kann mit einer beliebigen Anzahl von Robotern durchgeführt werden. Programmiert werden muss nur das jeweilige Verhalten eines Roboters, da das System die gesamte Steuerung des Kampfes übernimmt, also Berechnung der Roboter-/ Geschosspositionen sowie die grafische Darstellung. Diese Eigenschaften prädestinieren das Robocode-Framework als Testumgebung für einen einzelnen Agenten des ABSINTH-Systems: Jeder Agent steuert einen Roboter, der Spielablauf des Gesamtsystems wird von Robocode kontrolliert. Durch die grafische Ausgabe ist zudem eine einfache Bewertung der Erfolgschancen einer bestimmten Strategie möglich.

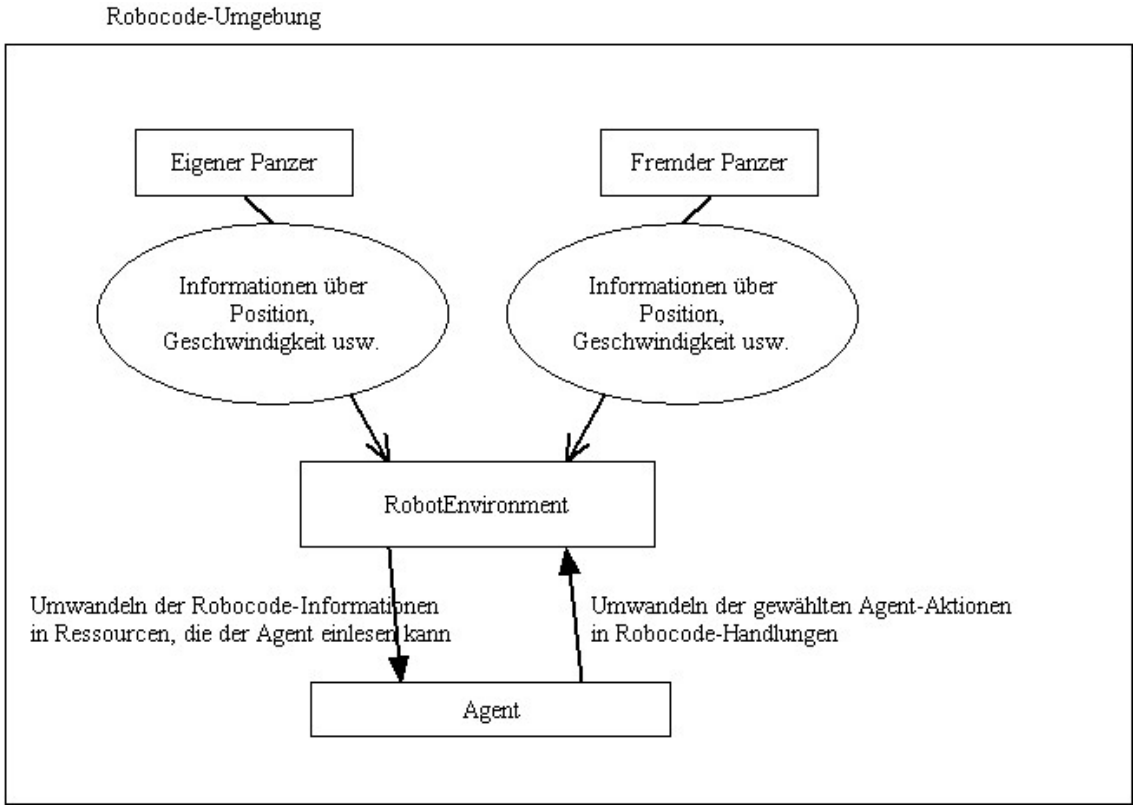

Abbildung 6.1: Die Funktion der Klasse RobotEnvironment

Technisch gesehen erfolgte die Integration des Agenten in Robocode durch Implementierung der Klasse RobotEnvironment, die von der Robot-Klasse von Robocode

abgeleitet wurde. Die Robot-Klasse ist die einzige Schnittstelle zwischen der Laufzeitumgebung von Robocode und selbstgeschriebenem Roboter-Verhaltenscode, d.h. Robocode kommuniziert nur über die Schnittstelle dieser Klasse mit externem Code. Im ABSINTH-System existiert aber eine Umwelt, in die die Agenten eingebettet sind und die ihnen die notwendigen Ressourcen zur Verfügung stellt und ihre Handlungen ausführt. Da die Interaktion zwischen Umwelt und Agent bereits fertig implementiert war, lag es nahe, diese Strukturen nicht zu verändern und stattdessen eine neue Adapterklasse zu erzeugen. Diese hat die Aufgabe, die von Robocode zur Verfügung gestellten Informationen über den eigenen und die gegnerischen Panzer so aufzubereiten, dass die Agenten sie unmittelbar für ihre Entscheidungsfindung nutzen können. Die Daten des eigenen Panzers wie Energie, Geschützturmrichtung oder Position werden in entsprechend benannte Ressourcen umgewandelt. Um das Testszenario nicht zu komplex werden zu lassen, wird lediglich ein gegnerischer Panzer in den Ressourcen berücksichtigt. Auf diese Weise werden alle wichtigen Information dem Agenten in Form eines Umweltbildes übergeben. Die Handlung, für die er sich dann entscheidet, wird ebenfalls von RobotEnvironment umgesetzt: Durch ein einfaches Parsing des zurückgegebenen Handlungsnamens wird die entsprechende, von Robocode erwartete Methode mit den jeweiligen Parametern aufgerufen. Beispielsweise wird für die Handlung „mediumAhead“ die Robocode-Methode `ahead(100)` ausgeführt. Dass bei solchen notwendigerweise festen Zuordnungen ein wenig Handlungsspielraum verloren geht, liegt auf der Hand. Somit kann ein Agent mit einer Handlung nur in drei Abstufungen geradeaus/rückwärts fahren oder seinen Geschützturm drehen. Im Vergleich zu den nativen Robotern, die alle Informationen direkt verwenden können, erscheint der Roboter-Agent also etwas eingeschränkt. Dennoch wurden gegen einige Testroboter auch durchaus positive Resultate erzielt.

6.1.2 Ergebnisse des Contest

Zum testen des Robocode-Szenarios wurde die Projektgruppe in 4 Gruppen aufgeteilt. Ziel jeder Gruppe war es, einen möglichst erfolgreichen Agenten zu entwerfen. Jede Gruppe musste also entscheiden aus wie vielen Charaktertypen ihr Agent bestehen sollte, für jeden dieser Typen die Regeln schreiben und für jede Ressource die linguistischen Variablen definieren. Eine Regel in diesem Szenario sah dann z.B. folgendermaßen aus:

```
<entry key="heavyFire X hit0">
    [Low enemyDistance, Center1enemyToMyGunBearing]
       == [Probability High]</entry>
```

Diese Regel bedeutet: Die Aktion stark schießen führt mit einer hohen Wahrscheinlichkeit zur (positiv bewerteten) Folge “hit0“, wenn die Distanz gering ist und die Kanone auf den Gegner gerichtet ist. Die verschiedenen Gruppen entwickelten vor allem durch Herumprobieren immer bessere Agenten.
Zum Abschluss wurde ein Turnier gespielt, in dem jeder Agent gegen jeden anderen

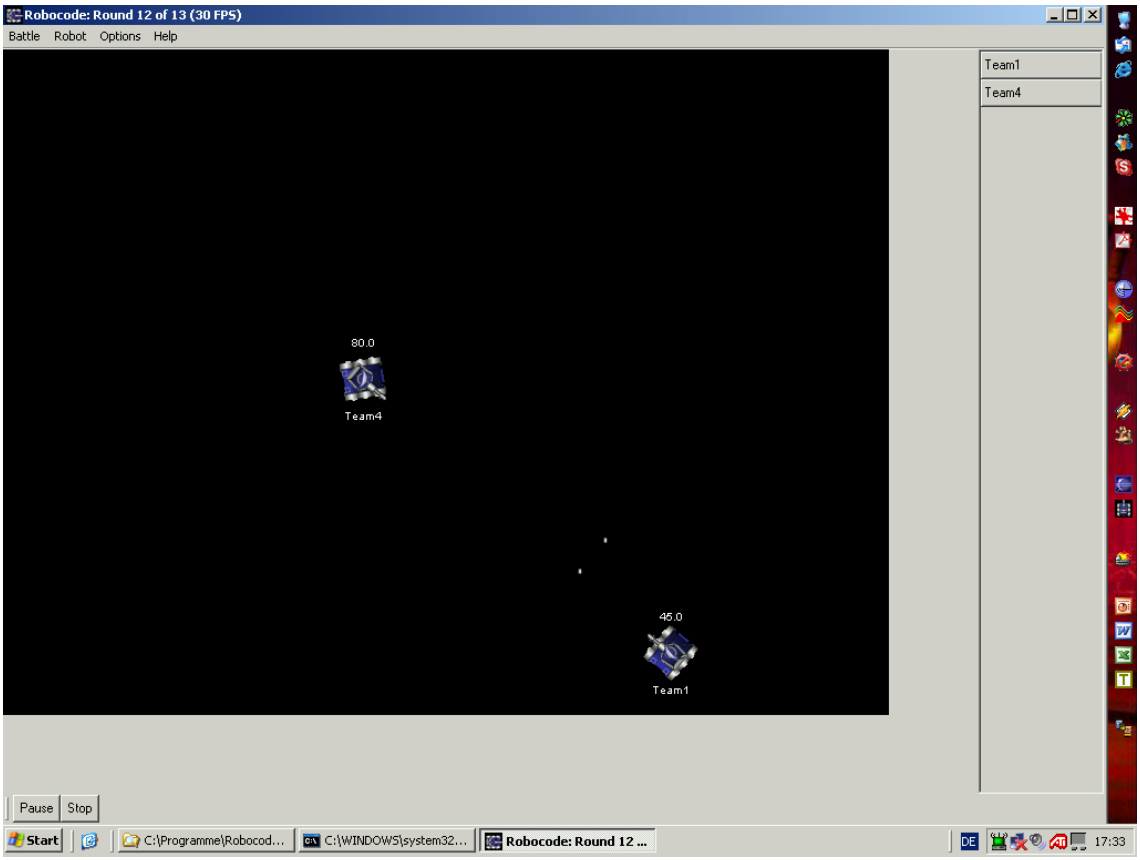

Abbildung 6.2: Ein Screenshot vom Robocodeturnier

Agenten 30 mal in einem Zweikampf antrat. Die Strategie des Siegeragenten war relativ einfach aber auch sehr effektiv. Er wechselte einfach schießen und ausweichen immer miteinander ab, so dass er die gegnerischen Panzer oft traf, aber nur schwer selbst getroffen werden konnte. Diesem Panzer gelang es auch, gegen die nativen Robocode-Panzer teilweise recht ansehnliche Ergegnisse zu erzielen, obwohl ihm nur diskretisierte Handlungen zur Verfügung standen. Da auch die anderen Agenten nicht schlecht waren, hatten wir durch dieses Szenario also gezeigt, dass man mit unserem

System “intelligent“ handelnde Agenten entwickeln kann. Allerdings zeigte sich auch ein erstes Problem: Wenn man einem Charaktertypen zu viele Regeln gab wurde es schnell unübersichtlich und schwer nachvollziehbar wieso der Agent zu bestimmten Entscheidungen kam. Deshalb hatte auch von allen entwickelten Charaktertypen keiner mehr als 50 Regeln.

6.2 Szenarien

6.2.1 Producer/Consumer Szenario

Das Producer-Consumer-Szenario diente zunächst als Test für die korrekte Funktionsweise der Umwelt des Absinth-Systems. Darüberhinaus sollte überprüft werden, inwieweit die Agenten sinnvoll handeln und wie sich ihre Handlungen auf die Ressourcen auswirken, d.h. ob sich die Charaktertypen so verhalten würden, wie sie modelliert wurden. Der große Bereich „Verhandlung“ wurde hierbei ausser acht gelassen, nur die Aktionen und Entscheidungen der Agenten sollte Dynamik in die Simulation bringen. Benötigt wurde also ein Szenario, in dem Agenten mit einer Menge vorgegebener Handlungen sinnvolle und nachvollziehbare Entscheidungen trafen. Andererseits durften die Regeln, anhand derer die Umwelt die Ressourcen aktualisierte, nicht zu kompliziert sein. Die Wahl fiel daher auf eine Adaption des typischen Producer-Consumer-Szenarios; eine überschaubare Anzahl von Agenten, Ressourcen und Regeln und die Tatsache, dass dieses Szenario ohne Verhandlung auskommt, machten es zum nächsten logischen Schritt auf dem Weg zum abschließenden Hochschulszenario.

Aufbau des Szenarios

Das Producer-Consumer-Problem ist eine typische Problemstellung in der Informatik, wenn es darum geht, die Synchronisation zwischen zwei oder mehreren Prozessen zu untersuchen bzw. zu simulieren. Dabei übernimmt ein sogenannter Producer die Aufgabe, Daten bereitzustellen, die ein oder mehrere Consumer benötigen. Üblicherweise wird dabei ein Puffer verwendet, in den der Producer schreibt und aus dem der Consumer liest. Dieses Grundgerüst wurde in Absinth nachgebildet, wobei die verwendete Terminologie mehr an die eigentliche Wortbedeutung von Produzieren und Konsumieren angelehnt ist: Futter herstellen und Futter aufessen.

Charaktertypen der Agenten Den Agenten wurden nun die beiden Rollen entsprechend zugewiesen, so gibt es in der hier beschriebenen Konfiguration zehn Ver-

braucher und einen Produzenten. Ihr Verhalten wird über die Charaktertypen gesteuert; der Produzent wird zu 100 Prozent durch den Charaktertypen „Producer“ repräsentiert, den Verbrauchern stehen dahingegen zwei verschiedene Verhaltensschablonen zur Verfügung, festgelegt durch die Charaktertypen „C-Gier“ und „C-Sozial“.

Ressourcen Für das Szenario kommt man mit zwei Arten von Ressourcen aus: Die Systemressource „Futter“ steht abstrakt für den Speicher, in den der Produzent das produzierte Futter hineinlegt und aus dem die Verbraucher das Futter herausnehmen. Zusätzlich verfügt jeder Agent über eine persönliche Ressource „Napf“; hier wird festgehalten, wieviel Futter sich jeder Verbraucher gerade genommen, aber noch nicht verzehrt hat. Der „Napf“ jedes Agenten kann höchstens 20 Futtereinheiten enthalten. Die Agenten in der Simulation „leben“ nicht ewig. Sobald ihre Ressource „Napf“ auf 0 sinkt, sterben sie - dies wurde im System dadurch realisiert, dass sie keine Handlungen mehr ausführen.

Aktionen Der Produzent hat die Wahl zwischen den beiden Aktionen „Produzieren“ und „Schlafen“. „Produzieren“ erhöht den allgemeinen Futterbestand um zehn Einheiten, solange der Futterspeicher nicht voll ist (der Höchstwert für Futter liegt bei 100 Einheiten). Entscheidet sich der Produzent für „Schlafen“, so hat das für den Produzenten selbst keine direkten Folgen. Bei beiden Aktionen wird jedoch die Napfressource der Verbraucher jedesmal um eine Einheit verringert. Auf diese Weise kann der stetige Verbrauch der Consumer modelliert werden. Den Verbrauchern steht ebenfalls die Aktion „Schlafen“ mit den gleichen Auswirkungen zur Verfügung. Darüberhinaus können sie aber auch „Fressen“ - dann werden ihrem Napf-Bestand drei Einheiten hinzugefügt und vom globalen Futtervorrat abgezogen.

Folgen Um das Verhalten des Produzenten zu steuern, werden für diesen Agenten die Folgen „Futter-voll“ und „Futter-leer“ benutzt. Ihn kümmert nur, ob durch seine Handlungen der Futtervorrat insgesamt steigt oder fällt - seine Nutzenwerte (Futter-voll=3 und Futter-leer=1) sind so eingestellt, dass er hautpsächlich seiner Produktionspflicht nachkommt, also meistens produziert und eher selten schläft; er sorgt dafür, dass die Ressource Futter ständig in einem bestimmten Bereich bleibt. Für die Consumer-Agenten ist nur wichtig, wie sich ihr Futter und das der anderen ändern wird. Sie unterscheiden die Folgen „Ich-satt“ und „Alle-satt“. Letzteres ist nötig, um ein rudimentäres Sozialverhalten abbilden zu können. So ist dem Charaktertyp „C-Gier“ wichtig, dass er selbst satt ist (Nutzen 5), weniger dass die anderen satt sind (Nutzen 1). Der sozialere Charaktertyp „C-Sozial“ dagegen schätzt das allgemeine Wohlergehen der anderen höher ein: Die Folge „Alle-satt“ hat für ihn den

Nutzen 4, sich selbst satt zu machen den Nutzen 3.

Modellierung der Charaktertypen

- Produzent

  Die Regeln des Produzenten sind so ausgelegt, dass er mit großer Wahrscheinlichkeit Futter herstellt, sobald der allgemeine Futtervorrat gering ist. Dabei kann er nicht über das Maximum von 100 Einheiten hinaus produzieren; ab einem Betrag von etwa 10 Einheiten wird er höchstwahrscheinlich immer Futter produzieren. Ist der Futterbestand bereits auf einem hohen Niveau, kann er sich schon eher für die Aktion Schlafen entscheiden, je mehr Futter also vorhanden ist, desto größer ist die Schlaf-Wahrscheinlichkeit des Produzenten.

  Beispielregeln für die Handlung „Produzieren“ und die Folge „Futter-voll“:

  - WENN Futter=wenig UND Futter=nichtVoll DANN Probability=hoch
  - WENN Futter=mittel UND Futter=nichtVoll DANN Probability=mittel
  - WENN Futter=viel UND Futter=nichtVoll DANN Probability=gering

- Charaktertyp C-Gier

  Dem Verbraucher-Charaktertypen „C-Gier“ geht es in erster Linie um seinen eigenen Futtervorrat. Wenn sein Napf nicht bereits voll ist, frisst er solange, bis kein Futter mehr da ist. Insbesondere berücksichtigt er bei seinen Entscheidungen weder den allgemeinen Futterbestand noch versucht er Annahmen über den Napf-Inhalt der anderen Verbraucher zu machen. Auch wird er meist nur schlafen, wenn sein Napf voll ist und ihm somit keine bessere Aktion zur Verfügung steht.

  Beispielregel für die Handlung „Fressen“ und die Folge „Ich-satt“:

  - WENN Futter=nichtLeer UND Napf=nichtVoll DANN Probability=hoch

- Charaktertyp C-Sozial

  Der soziale Charaktertyp beginnt erst dann zu fressen, wenn sein Napf relativ leer ist. Sein soziales Verhalten gegenüber den anderen zeigt sich darin, dass er weniger frisst, wenn das Futter insgesamt knapp ist und er selbst noch über einen Vorrat verfügt. Anders ausgedrückt: er frisst, wenn auch den anderen noch genug Futter übrigbleibt, und hält sich zurück, wenn er selbst nicht „dringend“ fressen muss. Dennoch will auch dieser Charaktertyp überleben und

muss sich auch mit Futter versorgen; daher ist die Nutzen-Gewichtung der Folge „Ich-satt“ mit einem Wert von 3 nur geringfügig niedriger als die der Folge „Alle-satt“ mit einem Wert von 4.

Beispielregeln für die Handlung „Fressen“ und die Folge „Alle-satt“:

- WENN Futter=wenig UND Futter=nichtLeer UND Napf=nichtVoll DANN Probability=hoch
- WENN Futter=mittel UND Futter=nichtLeer UND Napf=nichtVoll DANN Probability=mittel
- WENN Futter=viel UND Futter=nichtLeer UND Napf=nichtVoll DANN Probability=gering

Ergebnisse

Bei der Erstellung des Szenarios stand zunächst die Frage im Vordergrund, ob die Agenten bzw. Charaktertypen ihre Handlungen entsprechend ihrer Modellierung durchführen würden. Von einem gierigen Agenten erwartet man beispielsweise, dass er solange frisst, wie es Futter gibt, von einem sozialen Agenten, dass er eher pausiert, wenn ihm noch genug Futter zur Verfügung steht. Dieses grundlegende Verhalten konnte in den Testläufen auch beobachtet werden. Zusätzlich wurde jedoch noch eine Reihe von Fragestellungen entworfen, unter deren Gesichtspunkten die Simulationsergebnisse zu bewerten waren.

Bei allen Simulationsdurchläufen wurden zehn Verbraucher-Agenten mit verschiedenen charakterlichen Zusammensetzungen und ein gleichbleibender Produzenten-Agent verwendet. Der Produzent stellt bei seiner Handlung „Produzieren“ immer 10 Futtereinheiten her, so dass theoretisch alle Verbraucher satt werden könnten. Zu Beginn hat jeder von ihnen 10 Einheiten in seinem Napf, der globale Futtervorrat beträgt 50 Einheiten. Die Ergebnisse, wie sie in den Diagrammen visualisiert sind, beruhen auf der Mittelung von jeweils zehn Durchläufen pro Konfiguration über einen Zeitraum von 100 Zeiteinheiten. Aufgrund des Kurvenverlaufs und der Dimensionierung der übrigen Parameter wie Anfangsbestände von Napf und Futter lässt sich feststellen, dass die Anzahl der Zeitschritte ausreichend ist.

Rein-charakterliche Agenten Zunächst wurden die Merkmale von Szenarien mit rein-charakterlichen Agenten untersucht. Handlungen von Agenten, die nur durch einen einzigen Charaktertypen zu Hundert Prozent repräsentiert werden, lassen sich naturgemäß noch am ehesten vorhersagen. Daher macht es Sinn, erst einmal das

Verhalten der einzelnen Charaktertypen zu analysieren, bevor man sich verschiedenen Zusammensetzungen und Gewichtungen zuwendet.

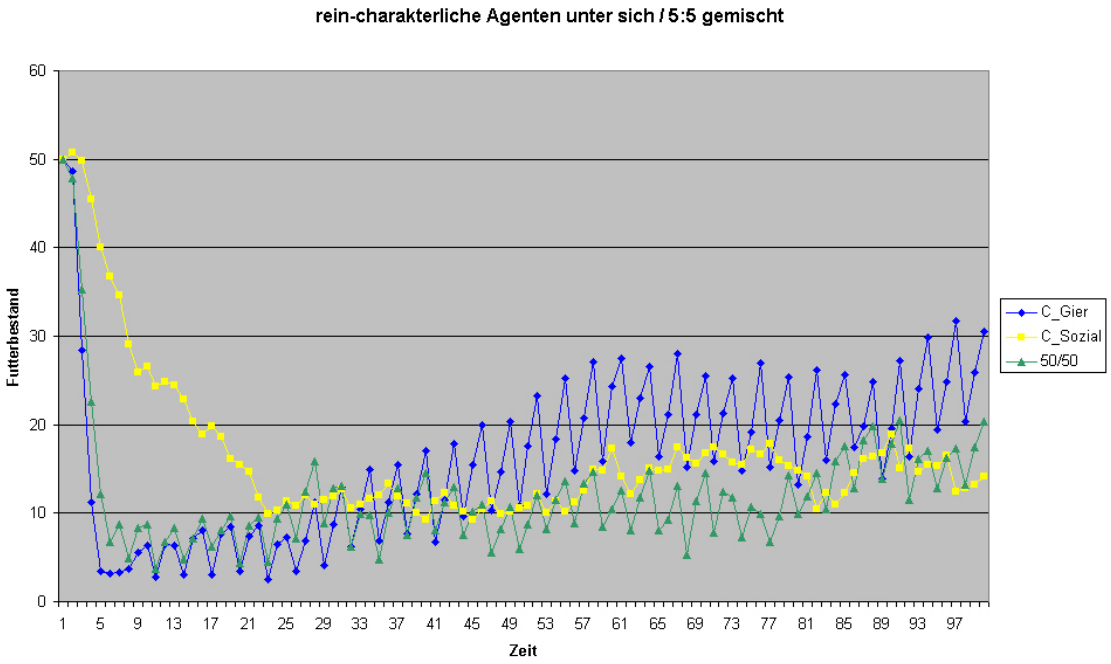

Abbildung 6.3: Globaler Futterbestand in drei Agententenkonfigurationen

Das Diagramm 6.3 zeigt den Verlauf der Ressource „Futter“ für drei verschiedene Konfigurationen:

- 10 Agenten mit reinem Gier-Charakter
- 10 Agenten mit reinem Sozial-Charakter
- 5 Agentent mit reinem Gier-Charakter, 5 Agenten mit reinem Sozial-Charakter

Die Kurven der beiden Szenarien mit rein gierigen und sozialen Charakteren weisen besonders zu Beginn einen starken Unterschied auf: Während der Futterbestand bei den gierigen Agenten rapide fällt (anfangs ist ihr Napf nur halbvoll, daher gibt es keinen Grund, nicht zu fressen), ist der Abstieg bei den sozialen Agenten nicht so stark, da sie insgesamt mit geringerer Prioriät fressen. In beiden Fällen erreicht der Futterbestand aber ein Minimum, steigt dann wieder etwas an und bleibt schließlich

einigermaßen konstant (im Mittel). Ab einem gewissen Zeitpunkt, wenn nämlich der Anfangsfutterbestand abgetragen ist und die Versorgung nur noch durch den Produzenten erfolgt, pendelt sich der Futterbestand auf einem Niveau ein, wo in etwa ähnlich viel produziert wie konsumiert wird.

Dieses Niveau ist bei den gierigen Agenten zum Ende hin höher als bei den sozialen. Diese auf den ersten Blick ungewöhnliche Diskrepanz (man würde vielleicht eher vermuten, dass die Gierigen insgesamt mehr Futter verbrauchen) ist damit zu erklären, dass bei den gierigen Agenten direkt zu Beginn durchschnittlich ein oder zwei Agenten sterben. Dadurch steigt das Futter für die überlebenden Agenten indirekt an, da sie das Futter unter sich aufteilen können, das die anderen sonst verbraucht hätten. Demgegenüber ist bei rein sozialen Agenten keine zeitliche Abhängigkeit ihres Todeszeitpunkts festzustellen, d.h. sie sterben ungefähr gleichverteilt über den gesamten Zeitraum. Daher leben sie im Durchschnitt gegen Ende des Zeitabschnitts länger und verbrauchen mehr Futter, so dass der Gesamt-Futterbestand dann eher niedrig ist.

Betrachtet man die Kurve des Szenarios, in dem sich fünf gierige und fünf soziale Agenten um das Futter streiten, dann erkennt man hierin einen Mittelweg des Futterbestands. Durch die gierigen Agenten fällt der Futterbestand am Anfang wiederum stark ab, doch sobald ein gewisser Zeitpunkt überschritten ist, bleibt der Verlauf stabil. In diesem Szenario sind in den Testläufen nur Individuen der sozialen Agenten verhungert. Ist also ein solcher Agent erst einmal gestorben, tritt eine stabile Situation ein, in der sich die gierigen Agenten ihren Napf maximal füllen können, da sich die sozialen Agenten zu ihren Gunsten zurückhalten. Daher bleibt insgesamt weniger Futter übrig als bei den rein Gierigen und sogar als bei den rein Sozialen.

An Diagramm 6.4 lässt sich der stabile Zustand der beiden rein-charakterlichen Szenarien besonders gut beobachten. Nach dem Tod einiger Agenten im Szenario mit ausschließlich gierigen Agenten pendelt sich der Napf-Vorrat jedes Agenten etwa bei 14 Einheiten ein. Bei den sozialen Agenten beginnt der stabile Zustand nach etwa 30 Zeitschritten, danach hat jeder Agent immer etwa 8 Einheiten in seinem Napf. Diese Unterschiede stehen nicht im Widerspruch zu Abbildung 6.3, da von den gierigen Charakteren durchschnittlich mehr gestorben sind und jedem von ihnen theoretisch mehr Futter zur Verfügung steht als einem Agenten im sozialen Szenario.

Im Szenario mit gemischten Agenten bildet sich ein ähnlicher Verlauf heraus. In der Grafik 6.5 zeigt sich, dass hier die gierigen Agenten nach der anfänglichen Sammelphase konstant etwa 18 Einheiten (also mehr als im rein-charakterlichen Szenario) in ihrem Napf haben. Dahingegen beträgt der Napfvorrat der sozialen Agenten zwischen 8 und 10 Einheiten (also weniger als im rein-charakterlichen Szenario) - aus diesen beiden Beobachtungen läßt sich die schon oben angedeutete Erkenntnis fol-

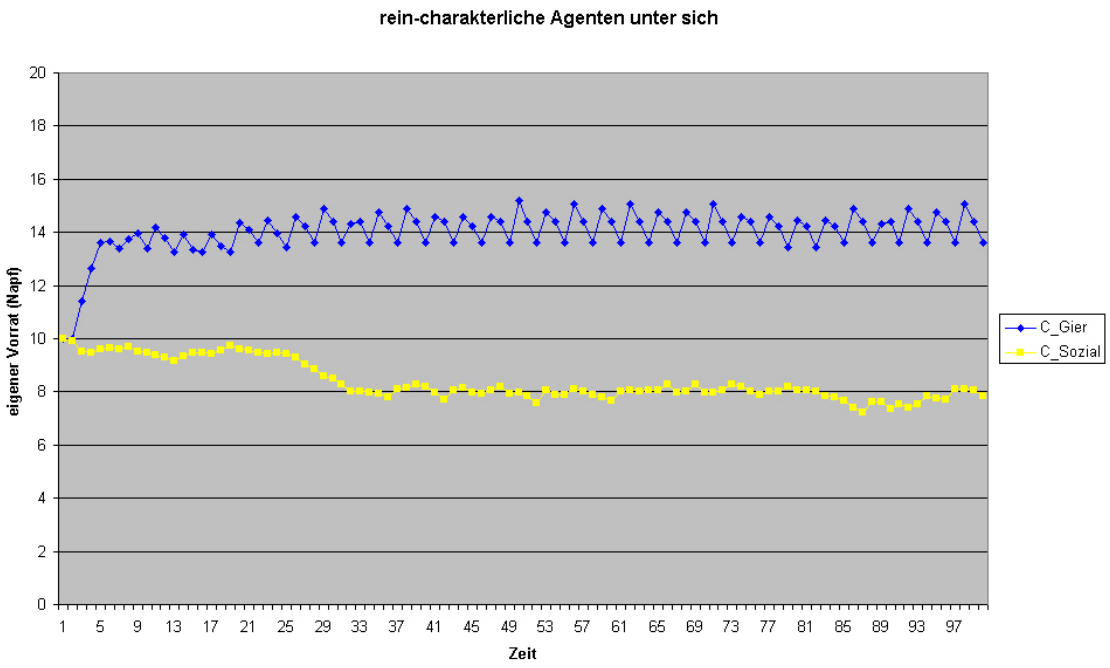

Abbildung 6.4: Eigener Vorrat der rein-charakterlichen Agenten

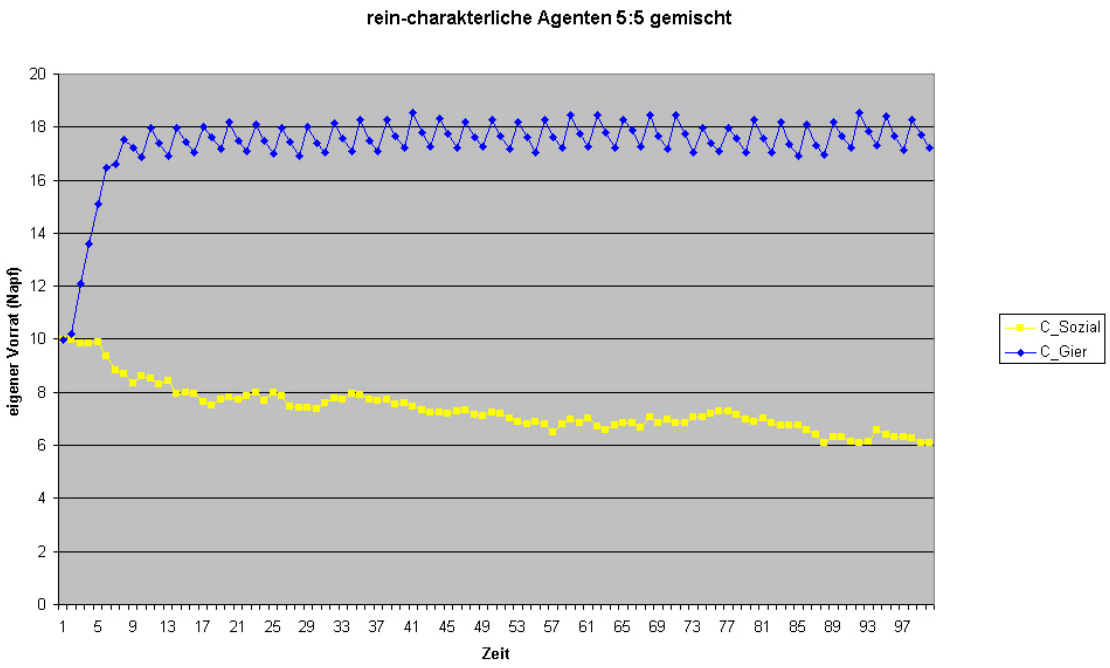

Abbildung 6.5: Eigener Vorrat der gemischten, rein-charakterlichen Agenten

gern, dass im gemischt-charakterlichen Agenten Szenario die sozialen Agenten von den gierigen ausgenutzt werden.

Gemischt-charakterliche Agenten Ein wesentliches Merkmal des gewählten Agentenmodells ist die Möglichkeit, ihnen verschiedene freie Gewichtungen von Charaktertypen zuzuordnen. Dies führt zu der Frage, wie sich verschiedene Gewichtungen von gierigem und sozialem Charakter auf die Gesamtsituation auswirken, d.h. insbesondere wie sich Futter- /Napfbestand und Sterberate entwickelt.

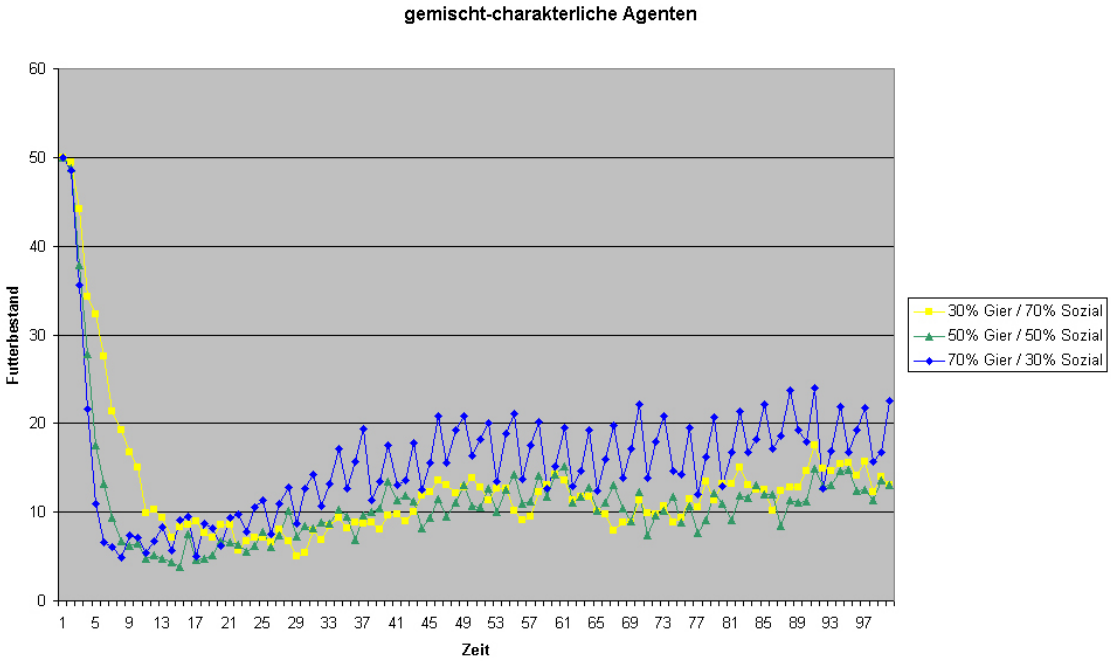

Abbildung 6.6: Globaler Futterbestand für drei verschiedene gemischt-charakterliche Agentenkonfigurationen

Im Diagramm 6.6 ist der Verlauf der Futterressource für jeweils drei verschiedenen Charaktergewichtungen dargestellt:

- 10 Agenten mit 30 Prozent Gier-Charakter und 70 Prozent Sozial-Charakter
- 10 Agenten mit 50 Prozent Gier-Charakter und 50 Prozent Sozial-Charakter

- 10 Agenten mit 70 Prozent Gier-Charakter und 30 Prozent Sozial-Charakter

Die beobachteten Verläufe entsprechen im großen und ganzen den Erwartungen. Die Kurve der Agenten mit dem höchsten Gier-Anteil entspricht fast der Kurve der reinen Gier-Agenten (vgl. Diagramm 6.3), die der Agenten mit dem höchsten Sozial-Anteil fast der Kurve der reinen Sozial-Agenten. Die 50:50 - Verteilung bildet schließlich eine Zwischenabstufung der beiden extremeren Verteilungen.

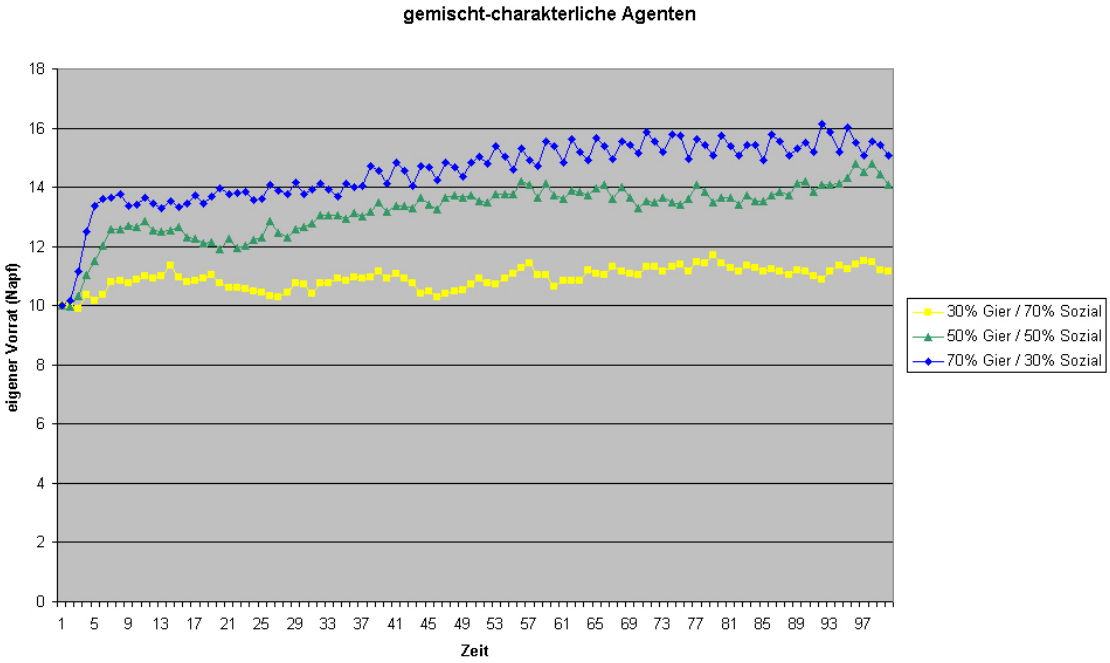

Abbildung 6.7: Eigener Vorrat der gemischt-charakterlichen Agenten

Der Verlauf der eigenen Napfvorräte der Agenten in Abbildung 6.7 schließlich bietet ebenfalls kaum Überraschungen: Je höher der Anteil des gierigen Charaktertypen in einem Agenten ist, desto mehr Futter sammelt er auch an. Aus den beiden letzten Diagrammen kann man ablesen, dass sich die Gewichtung von Charaktertypen durchaus sinnvoll nutzen läßt und auch nachvollziehbare Ergebnisse liefert.

Untersucht man die Sterberaten der Agenten in den einzelnen Szenarien und benutzt sie als Bewertungskriterium, so kann man einige Aussagen über die Güte der Konfigurationen treffen. In Diagramm 6.8 sind die Sterberaten aller getesteten Szenarien dargestellt. Die drei blauen Balken auf der linken Seite zeigen die mittlere

Anzahl von gestorbenen rein-charakterlichen Agenten. Gemäß der oben beschriebenen Modellierung ihrer Verhaltensweisen macht es Sinn, dass das Verhalten der gierigen Charaktere eher dazu führt, dass Individuen verhungern als es bei sozialen Agenten der Fall ist.

Interessant ist jedoch der Vergleich mit den drei rechten Balken, die die Sterberaten in den gemischt-charakterlichen Szenarien wiedergeben. Hier fällt auf, dass die Individuen der Konfiguration mit 30 Prozent gierigen und 70 Prozent sozialen Charaktertypen eine höhere Überlebenswahrscheinlichkeit haben als sogar die rein sozialen. Bewertet man das Szenario also auf Grundlage der Sterberate, so zeigt sich, dass man das Gesamtverhalten von sozialen Charakteren verbessern kann, wenn man ihnen einen kleineren „gierigen“ Charakterzug erlaubt. Sogar im Verhältnis 50:50 bleibt die Sterberate der gemischten Charaktertypen noch unter der der rein sozialen. Erst mit zunehmendem Anteil gieriger Verhaltensweisen wird sich die Sterberate ihrem Maximum nähern, das durch rein gierige Charaktertypen festgelegt wird.

Zur Erklärung dieser Beobachtung muss man sich vor Augen führen, dass sich in einem Szenario mit rein sozialen Charakteren nicht automatisch für alle die optimale Verteilung herausbildet. Die Zurückhaltung *aller* Agenten sorgt in ungünstigen Fällen dafür, dass einige verhungern, weil sie sich nicht rechtzeitig genug Vorräte gesammelt haben. Diese Entscheidung haben sie aber nur getroffen, weil auch die anderen nicht genug Futter hatten (auf deren „Sattheit“ sie ja auch achten). Gesteht man den Agenten nun durch einen gierigen Charakteranteil zu, öfter mal einfach zu fressen, ohne auf die anderen zu achten, so hat das für alle positive Folgen: Insgesamt erhöht sich der durchschnittliche Futtervorrat eines Agenten durch teilweise gieriges Verhalten, d.h. Sterben durch zögerliches Abwarten ist nicht mehr so wahrscheinlich. Je mehr man andererseits den gierigen Charakteranteil erhöht, desto mehr treten auch die Nachteile dieses Charaktertyps hervor: Ab einem bestimmten Punkt überwiegen im Durchschnitt dann die zu egoistischen Aktionen, was dann auch wieder zum Verhungern einiger Individuen führt.

6.2.2 Das Hochschulszenario

Nachdem die korrekte Funktionsweise von Umwelt und Charaktertyp-Handlungen sichergestellt war, wurde ein abschließendes Hochschul-Szenario erdacht. Hier konnten alle Möglichkeiten des Absinth-Systems integriert werden, von Aktionen der Agenten über Reaktionen der Umwelt bis hin zur Verhandlung zwischen Agenten. Bei der Modellierung der Hochschulumgebung wurde von der Wirklichkeit stark abstrahiert. So hat beispielsweise die Verhandlung kein konkretes Universitätsgremium zum Vorbild, sondern kapselt vielmehr vorstellbare Entscheidungen, die von Lehrstühlen zu

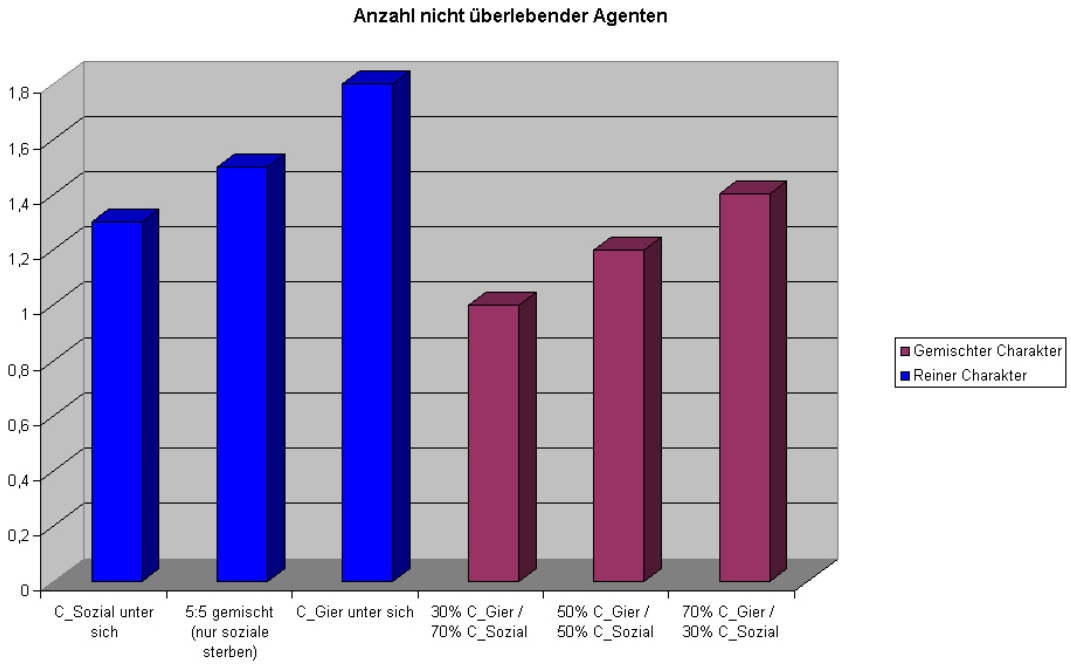

Abbildung 6.8: Sterberaten in verschiedenen Szenarien

Beginn eines Semesters getroffen werden könnten. Ähnliche Überlegungen spielten bei der Wahl geeigneter Umweltabhängigkeiten eine Rolle. Indem nur rudimentäre Regeln benutzt wurden, blieb der Simulationsablauf bis zu einem gewissen Grade planbar. Insgesamt wurde versucht, die interessantesten Beziehungen zwischen typischen Universitätsfaktoren wie Qualität der Lehre, Anzahl der Studenten oder Anzahl der Lehrstuhlmitarbeiter herauszufiltern und innerhalb eines Szenarios sinnvoll in Beziehung zu setzen.

Ein besonderer Fokus lag dabei auf dem Einfluß von Studiengebühren. Es sollte untersucht werden, welche Auswirkungen die Höhe der Studiengebühren auf das Gesamtsystem der Universität haben würden. Der Anspruch dabei war, die Charaktere der Studenten einigermaßen realistisch zu gestalten, um die Entscheidungen „echter“ Studenten so genau wie möglich abzubilden. Darüberhinaus sollten bei der Beurteilung der entstehenden Situationen auch die Reaktionen der Lehrstuhlagenten miteinbezogen werden.

Aufbau des Szenarios/Ablauf der Simulation

Der Aufbau des Szenarios gliedert sich in die logischen Einheiten „Lehrstühle“ und „Studenten“. Die Lehrstühle finanzieren sich durch zwei Einnahmequellen:

- *Drittmittel* (Gelder, die die Lehrstühle von externen Unternehmen für ihre Forschung bekommen)
- *Geld vom Land NRW*

Dabei erhält jeder Lehrstuhl seine Forschungsgelder unmittelbar und allein für sich. Das Geld vom Land NRW ist dagegen ein Betrag, der jedes Semester vom Land NRW zur Verfügung gestellt wird und unter allen Lehrstühlen aufgeteilt werden muss.

Der zeitliche Ablauf der Simulation besteht aus zwei Phasen. In der ersten Phase wird eine Semesteranfangsverhandlung abgehalten, in der die Lehrstühle das Geld von NRW unter sich aufteilen. Anschließend folgt eine Phase, in der sie ihr Geld investieren, d.h. verschiedene Aktionen gemäß ihrer Möglichkeiten ausführen. Beide Phasen zusammen bilden ein Semester.

Im folgenden sollen die einzelnen Bestandteile des Szenarios aufgeführt werden.

Agenten Die Agenten im Uniszenario lassen sich in zwei verschiedene Gruppen einteilen. Zunächst gibt es Agenten, die Lehrstühle repräsentieren, in diesem Szenario LS1, LS2 und LS11 genannt. Jeder dieser Agenten vertritt einen gesamten Lehrstuhl und verfolgt somit Ziele, die auch den gesamten Lehrstuhl betreffen. Auf der

anderen Seite gibt es die Studenten-Agenten: Ein solcher Agent trifft Entscheidungen aufgrund von rein persönlichen Interessen. In dieser Form des Szenarios wird die gesamte Studentenschaft durch zehn verschiedenartige, beispielhafte Agenten verkörpert. Etwas außerhalb dieser beiden Gruppen steht der Agent „Dekan“. Er hat nur programminterne Aufgaben und nimmt weder an Verhandlungen teil, noch führt er für die Ergebnisfindung wichtige Handlungen aus.

Ressourcen Jeder Lehrstuhlagent verfügt über folgende Ressourcen:

- *Geld:* Die Ressource „Geld“ gibt an, über welches Budget der Lehrstuhl verfügt.
- *FreieZeit:* Diese Ressource gibt die Zeit in Semesterwochenstunden an, die einem Lehrstuhl noch zur Verfügung stehen. Diese Stunden werden benötigt, um Handlungen wie „Lehren“ oder „Forschen“ auszuführen.
- *Mitarbeiter:* Jeder Lehrstuhl beschäftigt eine bestimmte Anzahl von Mitarbeitern. Je mehr Mitarbeiter ein Lehrstuhl hat, desto mehr Zeit kann er in Forschung oder Lehre investieren.
- *GeldertragForschen:* Der Geldbetrag, den ein Lehrstuhl für die Aktion „Forschen“ bekommt, kann für jeden Lehrstuhl unterschiedlich sein. Denkbar wären z.B. unterschiedlich gute Kontakte zu Unternehmen, mit denen sich ein Lehrstuhl besser positionieren kann als andere.
- *GeldertragLehren:* Mit Hilfe dieser Ressource kann ein Lehrstuhlagent abschätzen, wie viel Geld ihm eine „Lehren“-Aktion durchschnittlich bringen könnte. Da der tatsächliche Betrag als Ergebnis der Semesteranfangsverhandlung ermittelt wird, ist dieser Wert nur eine Annahme, die es dem Agenten erlaubt, sich besser zwischen Lehren und Forschen entscheiden zu können.

Für die Studentenagenten gibt es keine Entsprechungen. Sie besitzen keine Ressourcen im eigentlichen Sinne, sondern machen ihre Entscheidungen von globalen Systemressourcen abhängig.

Die wichtigsten Systemressourcen sind:

- *AnzahlStudenten:* In dieser Ressource wird die Anzahl der Studenten gespeichert, die sich aktuell durch ihre Handlung zum Studieren entschlossen haben, d.h. immatrikuliert sind.
- *Unikasse:* Die Gebühren, die Studenten pro Semester einzahlen (so sie denn tatsächlich studieren möchten) werden zu dieser Ressource jeweils aufaddiert.

Sie gibt also in jedem Semester an, wieviel Studiengebühren eingenommen wurden.

- *LehreGesamt*: Eine eher abstrakte Ressource, die angibt, wie viel Lehre die Lehrstühle zusammen im aktuellen Semester anbieten. Der Wert berechnet sich also aus den einzelnen Semesterwochenstunden, die, ausgelöst durch die Aktion „Lehren“, zum Lehren verbraucht werden.

- *GeldVonNRW:* Der Geldbetrag, den das Land NRW der Universität pro Semester zur Verfügung stellt. Dieser Betrag wird zu Semesterbeginn gleich dem Betrag der Ressource Unikasse gesetzt - dies bedeutet, dass die gesamten eingenommenen Studiengebühren allen Lehrstühlen zur Verteilung übertragen wird.

- *QualitätDerLehre:* Der Ruf der Universität spiegelt sich in der Ressource „QualitätDerLehre“ wieder. Dieser Wert verringert sich mit jedem Studenten, der sich entscheidet, zu studieren: Hat eine Universität zu viele Studenten, so wird sie möglicherweise für einige potentielle Studenten unattraktiv. Die Qualität der Lehre steigt jedoch andererseits an, wenn insgesamt viel Lehre angeboten wird, ist also abhängig von „LehreGesamt“.

- *KostenDerLehre:* Diese Ressource ist der zentrale Stellparameter des Hochschulszenarios. Sie gibt an, wieviel Studiengebühren ein Student bezahlen muss, um studieren zu können.

Aktionen Die Lehrstuhlagenten können in der zweiten Simulationsphase (nach der Verhandlung) zwischen verschiedenen Aktionen wählen. Eine Aktion wie Forschen oder Lehren verbraucht dabei eine Semesterwochenstunde. Die Agenten fahren solange mit ihren Handlungen fort, wie sie noch freie Zeit bzw. Geld haben. Kann keiner der Lehrstuhlagenten mehr eine Aktion ausführen, wird vom Dekan eine neue Verhandlung gestartet und das nächste Semester beginnt.

- *Forschen:* Entscheidet sich der Lehrstuhl zu forschen, dann wird ihm sein entsprechender Geldbetrag aus „GeldertragForschen“ gutgeschrieben. Die Ressource „GeldertragForschen“ wird dabei jedesmal um 0.1 Einheiten erniedrigt; auf diese Weise wird verhindert, dass der Agent „ewig“ profitabel forschen kann. Diese Annahme ist sinnvoll, da in der Realität die Forschungskapazitäten (Kontakte) auch irgendwann erschöpft sind und sich die Einnahmen nicht beliebig steigern lassen.

- *Lehren:* Bietet ein Lehrstuhl Lehrveranstaltungen an, so kostet ihn das zusätzlich zu einer SWS auch eine Geldeinheit, da erforderliche Lehrmittel bezahlt werden müssen.

- *Mitarbeiter einstellen:* Das Einstellen eines Mitarbeiters bringt dem Lehrstuhl eine zusätzliche SWS an freier Zeit ein, dafür kostet ein Mitarbeiter 100 Geldeinheiten.

- *Mitarbeiter entlassen:* Hat ein Lehrstuhl zuviel Zeit und zuwenig Geld zur Verfügung, kann er einen Mitarbeiter entlassen - dies bringt ihm 80 Geldeinheiten ein, also weniger als er für den Mitarbeiter zunächst bezahlt hat.

- *Verhandeln:* Mit der Aktion „Verhandeln“ entschließt sich der Lehrstuhlagent schließlich, an der Semesteranfangsverhandlung teilzunehmen. Diese wird erst gestartet, wenn kein Lehrstuhlagent mehr eine andere Handlung ausführen möchte.

In der zweiten Simulationsphase haben auch die Studentenagenten die Möglichkeit, Handlungen auszuführen. Ihnen bieten sich zwei Alternativen an, die jeweils für das gesamte Semester Gültigkeit haben:

- *Studieren:* Ein potentieller Student wird zum immatrikulierten Studenten, wenn er sich für die Aktion „Studieren“ entschließt. Er bezahlt die in „KostenDerLehre“ eingestellten Studiengebühren und ist damit ein Student der simulierten Universität für das aktuelle Semester.

- *NichtStudieren:* Der Studentenagent gehört nach Ausführung dieser Aktion nicht (mehr) zur Gruppe der Studenten im aktuellen Semester.

Charaktertypen der Agenten Die Lehrstuhl-Agenten können sich aus drei verschiedenen Charaktertypen zusammensetzen. Der „Homo Oeconomicus“ handelt nach rein ökonomischen Gesichtspunkten, d.h. er wählt die Aktion aus, die ihm voraussichtlich das meiste Geld bei gleichzeitig möglichst geringen Kosten einbringt. Das Hauptziel des Charaktertyps „Studentenfreund“ ist es hingegen, möglichst viel Lehre anzubieten und damit Studenten an die Universität zu locken. Der dritte Charaktertyp „Wissenschaftler“ schließlich interessiert sich hauptsächlich für die Forschung und versucht, seine Lehrverpflichtung gerade zu erfüllen, ansonsten aber die freie Zeit in Forschungen zu investieren.

Die Studenten verfügen über eine ähnlich große charakterliche Vielfalt. Ihnen stehen mit den Charaktertypen „BillichWillich“, „Bonze“, „Mr.Hanf“ und „Qualitaetsgeil“ verschiedene Verhaltensmuster zur Auswahl, die die meisten denkbaren

Studentenverhalten (etwas überspitzt) abbilden. Beispielsweise studiert der Charaktertyp „BillichWillich“ nur, wenn die Studiengebühren gering sind und achtet dabei nicht auf die Qualität der Lehre. Dem Charaktertypen „Bonze“ ist dagegen wichtig, dass die Kosten der Lehre gerade hoch sind - er will sich von anderen, vielleicht ärmeren Studenten absetzen. „Mr.Hanf“ kümmert sich weder besonders um die Qualität noch um die Kosten der Lehre - er ist bei beiden mit durchschnittlichen Werten zufrieden. Der Charaktertyp „Qualitaetsgeil“ berücksichtigt bei seiner Entscheidung, zu studieren ausschließlich die Qualität der Lehre - die Studiengebühren sind ihm mehr oder weniger egal.

Bei der Wahl der Charaktertypen ist zu beachten, dass die vorgestellten Persönlichkeitsmerkmale sicherlich überzogen dargestellt sind. Ein gesamtheitlicher Charakter ergibt sich jedoch immer aus einer Kombination verschieden gewichteter Charaktertypen, die ähnlich wohl auch bei einem realen Studenten bzw. Lehrstuhl/Professor auftreten könnten.

Ergebnisse

Beim Testen des Uniszenarios konzentrierten wir uns auf die Fragestellung wie sich Studiengebühren auf die Qualität bzw. den Ruf der Uni und auf die Anzahl der Studenten auswirken. Wir verglichen daher Studiengebühren in Höhe von 10 mit Studiengebühren in Höhe von 50. Für jeden dieser Werte ließen wir das Szenario 12 mal bis zum 10. Semester durchlaufen.
Vor jedem Testlauf machten wir folgende Annahmen:
LS1 ist ein reiner “Homo Oeconomicus“, LS2 zu 60 Prozent “Homo Oeconomicus“ und zu 40 Prozent “Wissenschaftler“ und LS11 zu 60 Prozent “Homo Oeconomicus“ und zu 40 Prozent “Studentenfreund“.
Die Zusammensetzung der 10 Studenten wurde für jeden Testpaarlauf zufällig neu generiert, wobei im Schnitt der Bonze deutlich weniger vorkommen sollte als die anderen Charaktertypen und außerdem wurde ausgeschlossen, dass ein Agent aus Bonze und aus BillichWillich bestehen konnte .
Jeder Lehrstuhl erhielt am Anfang eines jeden Testlaufs ein Startkapital von 100.
LS1 hatte zu Anfang immer 20 , LS2 12 und LS11 17 Mitarbeiter.
Die Ressource “GeldertragForschen“ hatte für den LS1 den Startwert 3, für den LS 2 den Startwert 4 und für den LS11 den Startwert 3,5.
Die Startwerte der Lehrstuhlressourcen waren zwar mehr oder weniger willkürlich gesetzt, aber wir legten Wert darauf für jeden Lehrstuhl andere Werte zu wählen, um zu verhindern, dass sie zu oft die gleichen Entscheidungen treffen würden. Unsere Untersuchung bezog sich dann auf die Ressourcen “AnzahlStudenten“, “QualitätDerLehre“

und auf die Mitarbeiterzahl der einzelnen Lehrstühle nach 10 Semestern. Anhand dieser Kriterien sollte dann hinterher bestimmt werden, welche Studiengebührhöhe in diesem Szenario die bessere ist. Die Mitarbeiterzahl schauten wir uns genauer an, da durch sie der Wohlstand der Lehrstühle abgelesen werden kann, denn unsere Regeln bewirkten, dass die Lehrstühle immer ihr überschüssiges Geld in neue Mitarbeiter investierten.
Das Gesamtergebnis sieht man anhand der Grafik 6.9.

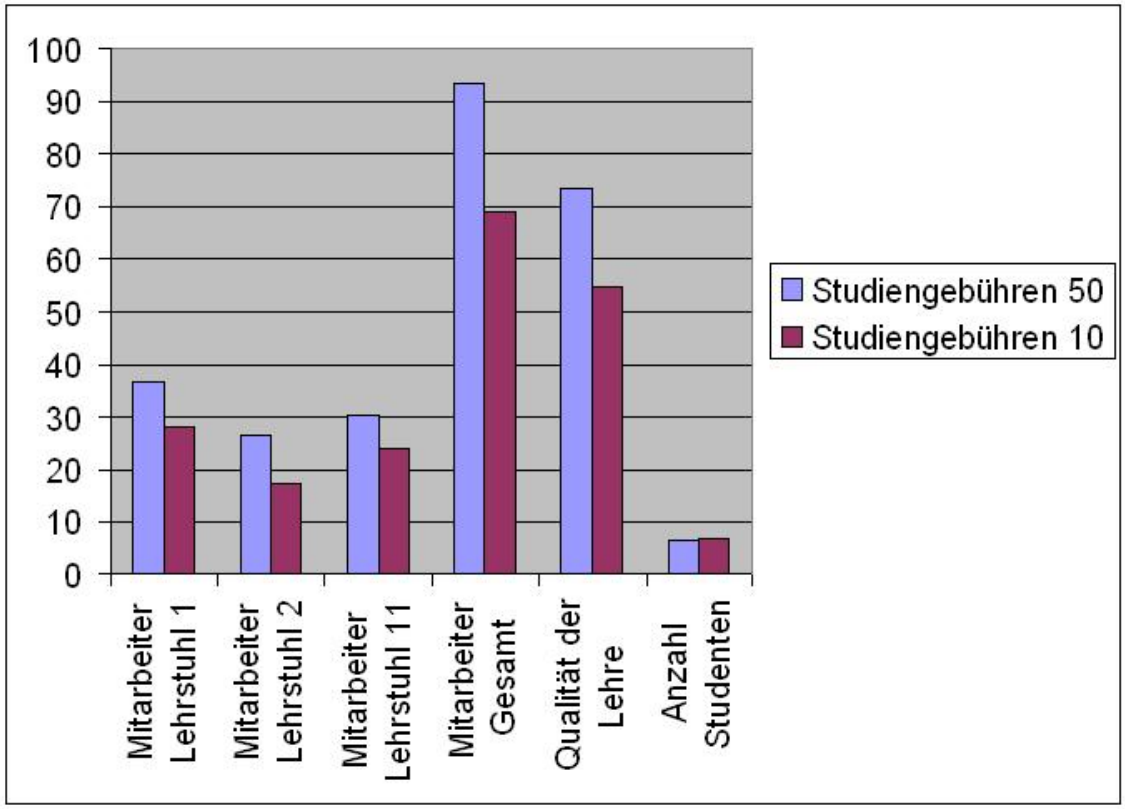

Abbildung 6.9: Ergebnisse des Uniszenarios

Wie hier zu sehen ist, ist in unserem Szenario die Qualität der Lehre bei Studiengebühren von 50 deutlich höher als bei Gebühren von nur 10. Dies liegt vor allem daran, dass die Lehrstühle bei höheren Gebühren mehr Geld zur Verfügung haben und daher dann wie auf der Grafik zu erkennen ist auch mehr Mitarbeiter haben, die dann natürlich auch mehr Lehre anbieten können. Verstärkt wird dieser Effekt noch dadurch, dass bei Studiengebühren von 50 auch weniger Studenten studieren. Das

bei Studiengebühren von 10 die Anzahl der Studenten etwas höher ist, ist damit zu erklären, dass sie ungefähr zu gleichen Teilen auf Qualität und Kosten achten, wobei der Unterschied der Kosten dabei allerdings weitaus drastischer ist als der Unterschied der Qualität. Etwas abgemildert wird dies noch durch das Vorhandensein des Charaktertypen Bonze, der sich allerdings nur relativ wenig auswirkt.
Durch dieses Szenario konnten wir auch schließlich die Verhandlung ausgiebig testen. In der Grafik 6.10 sieht man, wie die Lehrstühle zu Beginn der Verhandlung noch sehr egoistische Vorschläge machen. In Runde 1 fordert der LS2 für sich selbst den Großteil der aufzuteilenden 50 Geldeinheiten und will den anderen beiden Lehrstühlen nur relativ wenig abgeben. Diese lehnen diesen Vorschlag aber natürlich ab und deshalb fordert er in Runde 2 schon deutlich weniger für sich. Im weiteren Verlauf nähern sich die Lehrstühle somit immer weiter an, bis sie sich dann in diesem Beispiel in der 4. Runde auf eine Verteilung einigen.

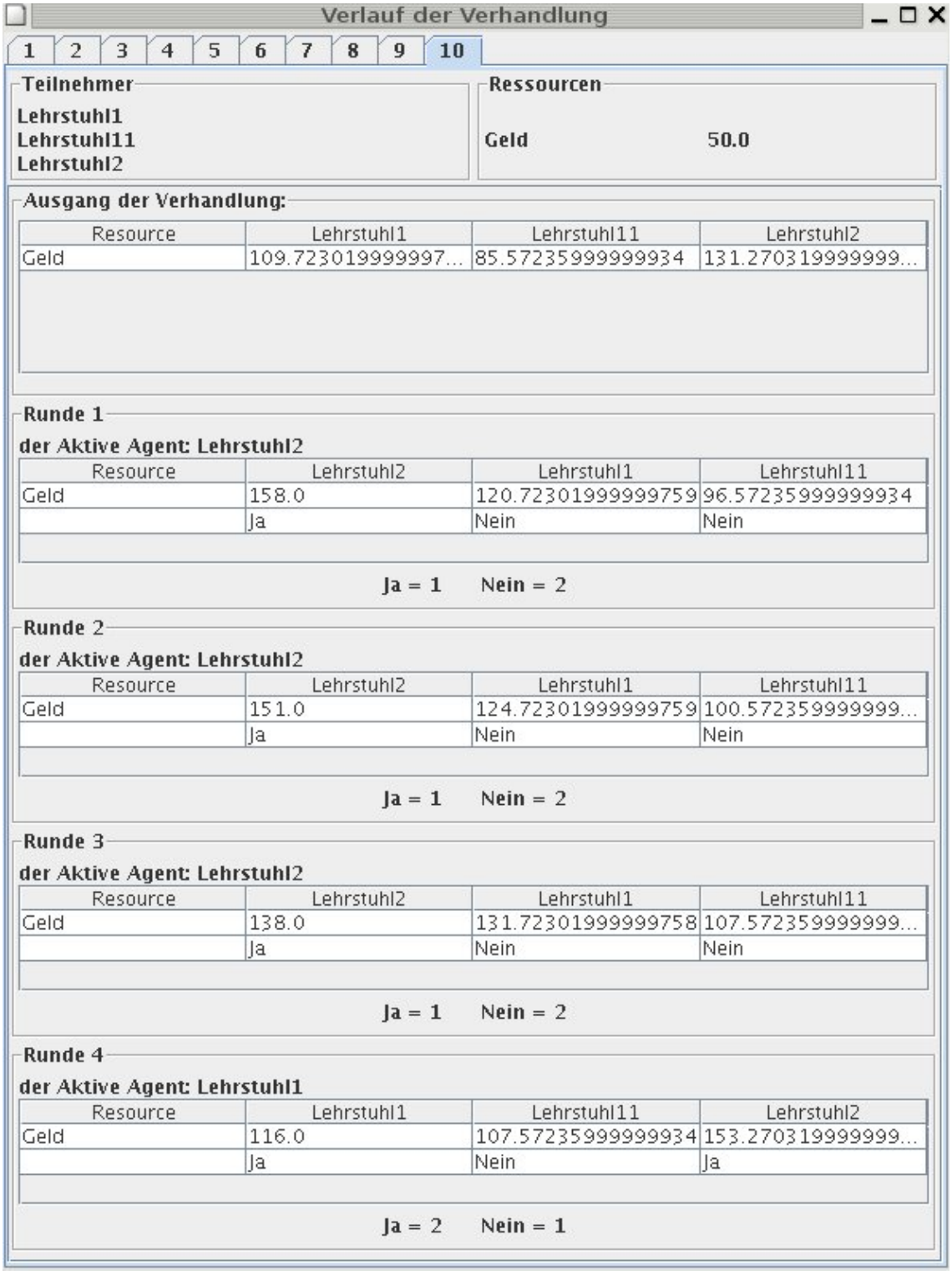

Abbildung 6.10: Verlauf der Verhandlung

Kapitel 7

Fazit

7.1 Fazit

Dieser Abschnitt soll, nachdem die Projektgruppe (im Folgenden PG) abgeschlossen wurde, eine kurze, rückblickende Zusammenfassung der Projektgruppe sowie einen zukunftsgerichteten Ausblick geben. Dabei liegt der Schwerpunkt auf der Entwicklung der Vorstellungen bzgl. der Realisierung des Projektes.
Zu Beginn stand die Idee ein Inselszenario in Spielform zu entwickeln, in welchem die Akteure untereinander Ressourcen verhandeln könnten. Die Grundidee war es Agenten zu erschaffen, welche selbstständig durch Aufgabenteilung und Austauschen von Ressourcen (Verhandeln) das Überleben im diesem Szenario sichern. Dabei sollten die Agenten unterschiedliche Fähigkeiten zur Ressourcengewinnung haben, so dass ein Tausch von Ressourcen unumgänglich wäre. Diese Idee wurde mit dem PG-Antrag auf das Universitätsszenario übertragen, wobei die Grundanforderungen an die Agenten, selbstständig agieren und verhandeln zu können, übernommen werden sollten. Zudem verfolgte man nicht weiter die Idee ein Spiel zu programmieren, sondern legte das Augenmerk auf den Simulationscharakter des Programmes, wobei die Verteilung beliebiger Ressourcen möglich sein sollte. Die Entscheidungsfindung des einzelnen Agenten sollte durch eine Regelbasis realisiert werden.
Im Laufe der Anforderungsdefinition wurde entschieden, bei der Erstellung der Regelbasen auf das Expertenwissen von Soziologen zurückzugreifen (siehe Kapitel 7.2), und die soziologische Werterwartungstheorie bei der Auswertung der Regeln zu benutzen (siehe Kapitel 2.2.1). Dabei setzt sich der Agent aus verschieden gewichteten Charaktertypen zusammen, welche individuelle Regelbasen besitzen. Weil dieses Modell mit festen Gewichtungen und Regeln als sehr statisch angenommen wurde, sollte zusätzliche Dynamik durch die Einführung eines Gedächtnisses, eine dynamische

Erweiterung der Regelbasen oder eine Veränderung der Charaktertypgewichtungen erreicht werden. Im Gespräch mit den Experten aus der Soziologie stellte sich jedoch heraus, dass zu viel Dynamik die Nachvollziehbarkeit der Simulation mindert und die Auswertung erschwert. Daher wurden die Ideen zunächst zurückgestellt mit der Option, diese nachträglich zu ergänzen. Das Gedächtnis hat hierbei eine Sonderrolle, da es bereits fertig implementiert ist, jedoch aufgrund von Zeitmangel nicht in das System eingebunden werden konnte.
Die Implementierung der Agenten nach dem obigen Modell wurde gegen Ende des 1. Semesters in der Robocode Umgebung von IBM getestet (siehe Kapitel 6.1). Zur Motivationssteigerung wurde die Testphase mit einem Wettbewerb abgeschlossen, bei welchem die Agenten, mit den in Kleingruppen entworfenen Regelbasen, gegeneinander antraten. Es wurde festgestellt, dass die Erstellung der Regelbasen mit Hilfe der GUI zum derzeitigen Stand unkomfortabel war, was zu einer Erneuerung der GUI-Struktur im zweiten Semester führte. Die Ergebnisse des Wettbewerbes übertrafen unsere Erwartungen, da die Agenten nicht nur das gewünschte Verhalten zeigten, sondern auch trotz „fehlender“ Dynamik komplexe Strategien erfolgreich umsetzten. Im weiteren stellte sich in dieser Phase heraus, dass die als sehr statisch angenommene Realisierung der Agenten, aufgrund der dynamischen Umwelt, vielseitige Handlungsmöglichkeiten zuläßt und das Modell, welches vorrangig auf soziale Interaktion ausgelegt ist, leicht portierbar ist.
Im zweiten Semester wurde die externe Testumwelt durch eine eigene, flexibel erstellbare Umwelt ersetzt. Der Benutzer kann beliebige Szenarien und Agentenkonfigurationen zusammenstellen und diese simulieren. Die zweite große Anforderung im zweiten Semester war es, dieses Konzept um die Möglichkeit, Verhandlungen in die Simulation einzubetten, zu erweitern.
Da die Entwicklung von Ansätzen in Kleingruppen im ersten Semester so erfolgreich war, wurden die Ideen für die Realisierung der Verhandlung wieder in solchen Gruppen ausgearbeitet. Bei der Fusion der Ideen kristallisierten sich zwei Ansätze heraus. Zum Einen die Reduzierung der Verhandlung auf Zweiergruppen, in welchen je ein Agent mit allen anderen Agenten, unter Benutzung einer speziellen Werterwartungstabelle (im Folgenden VET), eine Verhandlung führte. Aus diesen einzelnen Verhandlungsergebnissen wurde abschließend das globale Verhandlungsergebnis ermittelt. Zum Anderen ein Ansatz, welcher eine reale Verhandlung imitieren sollte. Hierbei generiert jeder Agent Vorschläge, welche er unter Benutzung seiner VET auswertet und sortiert. Diese werden nun im Laufe der Verhandlung absteigend vorgeschlagen. Danach wird abgestimmt, ob dieser Vorschlag akzeptiert wird, wobei erneut die VET zur Auswertung benutzt wird.
Da das vorrangige Ziel die Erzeugung einer, den Vorstellungen der Projektgruppe

nach, realitätsnahen Simulation war, wurde der zweite Vorschlag ausgewählt. Die Auswertung der VET allein berücksichtigt jedoch nur rein subjektive Interessen, daher wurde zusätzlich eine globale Komponente bei der Bewertung eines Vorschlages eingefügt. In einem speziellen Testszenario ergab sich, dass dieser Ansatz, je nach Zusammenstellung der Charaktertypen zu unterschiedlichen Verteilungspräferenzen führt. Diese Ergebnisse wurden im Uniszenario bestätigt.
Eine weitere wichtige Diskussion innerhalb des zweiten Semesters stellte das Festlegen der Arbeitshypothese der PG dar. Hierbei verschob sich der Schwerpunkt der Simulation weg von der Ausführung einzelner Aktionen, wie es in der Spielform vorgesehen war, hin zu einer globaleren Ursache- und Wirkungsanalyse. Dabei sollten die Änderungen von Charaktertypzusammenstellungen bzw. Ressourcenallokationen bei gegebenen Regelbasen auch Änderungen im Simulationsverhalten hervorrufen. Um dies zu testen sollten mehrere Szenarios entwickelt werden. Als erster, einfacherer Ansatz wurde ein Consumer/Producer Szenario erstellt, um die Auswirkung verschiedener Charaktertypgewichtungen zu untersuchen. Als Charaktertypen wurden ein Producer, ein gieriger und ein sozialer Consumer erstellt, wobei der Schwerpunkt auf der Untersuchung der verschiedenen Consumer gelegt wurde (siehe Kapitel 6.2.1). Es stellte sich heraus, dass verschiedene Verteilungen der Consumer-Charaktertypen auch zu unterschiedlichen Futterbeständen und Sterbensraten innerhalb der Population führten. Aufbauend auf diesem Ergebnis, konnte nun der Schwerpunkt auf die Optimierung des Szenarios gelegt werden, um herauszufinden, welche Zusammenstellung der Charaktertypen, die Sterbensrate minimiert. Dieses Szenario stellte außerdem einen erfolgreichen Funktionstest der dynamischen Verwaltung der Ressourcen innerhalb der Umwelt dar.
Die vorangegangen Szenarios zeigten, dass die einzelnen Module der Simulation voll funktionsfähig sind. Das Verhalten war dabei jeweils der Arbeitshypothese entsprechend. Als Ziel der Projektgruppe wurde abschließend ein umfangreiches Uniszenario erstellt, um die Auswirkung von Studiengebühren auf die Lehrqualität und die Studentenzahlen zu untersuchen (siehe Kapitel 6.2.2). Dieses Szenario stellt eine Verbindung aller vorangegangen Szenarios dar, da sowohl die Entwicklung der Ressourcen, bei unterschiedlicher Zusammenstellung der beteiligten Agenten, als auch die Verteilung von nötigen Mitteln durch Verhandlungen untersucht werden. Bei der Simulation kam es zu zahlreichen Ergebnissen. Zum einen zeigte sich ein umgekehrt proportionaler Zusammenhang zwischen den Kosten der Lehre und den Studierendenzahlen, wobei die Stärke des Rückganges jedoch von vielen weiteren Parametern, wie den Einkünften der Lehrstühle durch Forschung, der maximal möglichen Anzahl von Mitarbeiten und der zufälligen Zusammenstellung der Studentenschaft abhingen. Durch die Festlegung der Regelbasen und Ressourcenstartwerte, waren zahlreiche Va-

riationen der Entwicklung des Hochschulszenarios möglich. Eine empirische Studie zur Festlegung der Regelbasen und der Größenordnungen der Ressourcenwerte, sowie viele weitere Validierungsmaßnahmen sind für ein verwertbares Ergebnis unumgänglich.
Die Anforderungen, die im PG-Antrag definiert wurden, konnten somit letztendlich in ein lauffähiges System mit dem gewünschten Verhalten umgesetzt werden. Die Vorgehensweise war (wenn auch nicht immer formal und strikt im Voraus definiert) systematisch und gruppenorientiert. Dies zeigt sich nicht zuletzt darin, dass Entscheidungen durchweg auf eine solide Diskussionsbasis gestellt wurden und selten „aus dem Bauch heraus“ gefällt wurden. Es bestand eine gute Mischung zwischen den Zielen einer Lehrveranstaltung und den Bedingungen, wie man sie bei der realen Softwareentwicklung vorfindet. So haben alle PG-Teilnehmer ausnahmslos in ihren Rollen neue Kenntnisse und Erfahrungen erworben, die von der Projektplanerstellung, dem Gestalten und Abhalten von Meetings, über das Halten von Vorträgen, bis hin zu so technischen Kenntnissen wie der Benutzung von Entwicklungsumgebungen, Versionierungssystemen und weiteren Tools reichen und sicherlich über das Ende der PG hinweg von Nutzen sein werden. Viele dieser Kenntnisse konnten schließlich in das fertige Produkt einfließen, so dass keineswegs von einem wissenschaftlichen „herumprobieren“ auf einem zu hohen Abstraktionsniveau gesprochen werden kann. Hierfür steht die lauffähige Simulationsoftware AbSintH als Beweis. Doch auch die Software, welche letztendlich nicht zum Einsatz kommen konnte, wurde zu Beginn der PG ausführlich in Form von Tutorien erläutert, so dass Ansätze dieser Software für das Projekt weiterentwickelt werden konnten oder zumindest bei Aufgaben in der Zukunft auf diese zurückgegriffen werden kann. Beispiele hierfür sind die Multiagentenumgebung JADE (`http://jade.tilab.com`).
Nicht zuletzt war die Zusammenarbeit aller PG-Teilnehmer sehr gut. Es kam zwar zu Differenzen, die jedoch nicht langfristig waren und auf Diskussionsbasis überwunden werden konnten. Die Argumente und Diskussionen waren durchweg sachlich geprägt. Auch außerhalb der PG bestand zwischen vielen PG-Teilnehmern ein freundschaftliches Verhältnis, welches sich in zahlreichen Aktionen außerhalb der PG-Arbeit widerspiegelte.
Zusammenfassend lässt sich die Arbeit der PG als erfolgreich bezeichnen, obwohl einige Vorstellungen bezüglich des Umfangs der Simulation eingeschränkt werden mußten. Allen nachfolgenden PGs wünscht das gesamte AbSintH-Team viel Spaß und Erfolg bei der Projektgruppenarbeit!

7.2 Zusammenarbeit mit den Soziologen

Einen besonderen Aspekt der PG stellte die fachübergreifende Zusammenarbeit mit dem Fachbereich Soziologie der Fernuni Hagen dar. Zu Beginn der Entwicklung gab Dr. Thomas Kron eine Einführung in das soziologische Modell (siehe Kapitel 2.2.1), welches bei der Implementierung der Entscheidungsfindung berücksichtigt wurde. Die Verwendung von linguistischen Fuzzyregeln sollte es den Soziologen ermöglichen, ohne große Erfahrungen mit regelbasierten Systemen, die Agenten und deren Charaktertypen nahe am soziologischen Modell zu entwickeln. Als am Ende des ersten Semesters die Implementierung des Agenten abgeschlossen war, wurden Dr. Thomas Kron und seine Kollegen eingeladen. Die Einstellung des Agenten mit Hilfe der Regelbasen wurde mit Hilfe von Robocode (siehe Kapitel 6.1.1) vorgeführt. Im Nachhinein stellte sich heraus, dass dies ein Fehler war, da kämpfende Panzer keinen direkten Zusammenhang zu Sozialsimulationen haben und die Soziologen somit in Ihren Erwartungen enttäuscht wurden. Der Kontakt beschränkte sich daher im zweiten Semester auf das Beantworten einer E-Mail mit den gesammelten Fragen, die vor der Erstellung des Uniszenarios und der Konzeptionierung der Verhandlung innerhalb der Projektgruppe aufkamen. In dieser Mail stellte sich heraus, dass die Soziologie keiner eigenen Definition von Verhandlung folgt, sondern diese im Sinne der Spieltheorie als kooperatives Spiel mit konkurrierenden Interessen versteht. Als Schlüsselaktionen der Verhandlung bezeichnet Dr. Thomas Kron „Vorschlag machen“, „über Vorschlag verhandeln (Annahme / Ablehnung)“ und den abschließenden „Mehrheitsentscheid“. Die Frage nach Verhaltensregeln für die einzelnen soziologischen Charaktertypen beantwortet Dr. Kron allgemein: „...für einen Professor: ... Achterhaltung, mehr Ressourcen, Bildung für Alle, Elitenförderung, Einhaltung der Gruppennorm“, „für einen Studenten: ... möglichst gute Ausstattung für möglichst gute Lehre im Rahmen möglichst guter Forschung“ und „für einen Wissenschaftl. Mitarbeiter: ...Möglichkeiten der Weiterbeschäftigung und Qualifikation... dann zählt das Renomée des Profs und wie der so seine Sklaven behandelt. Dann die allgemeine Forschungs- und Betreuungsatmosphäre“. Ausgehend von diesen recht allgemeinen Informationen wurden die Verhandlung und das Uniszenario von der Projektgruppe selbst entworfen. Ein Benutzung Absinths durch den Fachbereich Soziologie in Hagen wird wahrscheinlich nicht vorgesehen.
Ein Grund für das Scheitern der fachübergreifenden Zusammenarbeit könnte das Fehlen von klaren für die Implementierung nutzbaren Definitionen innerhalb der Soziologie sein. Am Beispiel der VET sieht man, dass ein Konzept, welches klar beschrieben ist, auch erfolgreich in der Simulation eingesetzt werden kann. Leider fehlten für das Design der Charaktertypen und der Verhandlung solche Vorgaben.

Ein weiterer Grund für das mangelnde Interesse liegt anscheinend auch in der Scheu der Soziologen, selbst Computer zu benutzen, um ein Szenario zu erstellen. Aus bestehender Zusammenarbeit mit den Soziologen in anderen Aktivitäten hat sich herausgestellt, dass die Soziologen fertig aufbereitete Ergebnisse, welche direkt in ihre Arbeiten übernommen werden können, bevorzugen. Zum Anderen hat die Projektgruppe Fehler gemacht, da die Vorstellung der Architektur der Agenten innerhalb des RoboCode-Szenarios nicht ausgereicht hat, um die Soziologen von der Simulation zu überzeugen. Ein Szenario mit direktem sozialen Bezug hätte hier vorgezogen werden sollen.

7.3 Ausblick

Mit Blick in die Zukunft bleibt zu hoffen, dass die Arbeit und das Ergebnis der PG noch weitere positive Anstöße geben kann, da Aspekte, wie die soziologisch motivierte Architektur des Agenten und der Auswertung mit Hilfe der Werterwartungstheorie, neue Ansätze in der Sozialsimulation darstellen. Durch die Arbeit der PG ist belegt, dass die obigen Ansätze realisierbar sind und somit Forschungspotential bieten. Durch die offene Architektur von AbSintH sind eine Vielzahl von unterschiedlichen Simulationsszenarien möglich. Die bisherigen Szenarien zeigen dies. Eine Veröffentlichung der Ergebnisse aus dem Producer/Consumer oder dem Uniszenario in anerkannten Magazinen, wie JASSS (`http://jasss.soc.surrey.ac.uk/JASSS.html`) liegt nah.
Viele der Vorstellungen, die aufgrund des Zeitmangels nicht innerhalb des Zeitraums der PG realisiert wurden, könnten im Nachhinein umgesetzt werden. Dies umfasst zum einen eine Weiterentwicklung der bestehenden Szenarien, als auch eine Erweiterung der Programmfunktionalität. Ein Beispiel ist das Gedächtnis, welches den Zugriff auf vergangene Entscheidungen und Umweltbilder ermöglicht und somit eine Berücksichtigung von vergangenen Ereignissen zuläßt. Die Klasse ist vorhanden und bietet Schnittstellen, welche die Anbindung an die Regelbasis ermöglicht. Ebenso können die Ansätze zur dynamischen Anpassung von Gewichten und Regelbasen, welche im Laufe des ersten Semesters zurückgestellt wurden, ohne großen Aufwand in die Simulation eingefügt werden, was letztendlich eine Verbesserung des Produktes zu einer realitätsnäheren Simulation ermöglichen kann.

Anhang A

Seminar- und Tutoriumsphase

Die Seminar- und Tutoriumsphase hatte zwei elementare Ziele. Zum einen wurden grundlegende Informationen zu den behandelten Themengebieten und der benötigten Software vermittelt, zum anderen wurde das gegenseitige Kennen lernen unterstützt. Deshalb wurde die Seminarphase in zwei Tagen innerhalb einer Woche zu Beginn der Projektgruppe abgehalten, um sowohl eine schnelle Einarbeitung in den wissenschaftlichen Kontext zu bekommen, als auch durch Gespräche nach den Vorträgen und in den Pausen die gegenseitige Scheu abzulegen. Der zweite Punkt wurde auch erreicht, indem die Tutorien in Gruppen von je zwei Leuten gehalten wurden, was zusätzlich den Vorteil hatte, dass im Nachhinein jeweils zwei Ansprechpartner pro Thema vorhanden waren.

A.1 Seminare 1. Semester

In der ersten Seminarphase wurden die folgenden Vorträge gehalten:

- **Christian Mosblech - Hochschulsysteme**
  Da in der Simulation ein Hochschulszenario dargestellt werden soll, stellte der Vortrag die Organisation und Hierarchie der Universität Dortmund dar und verglich sie mit alternativen Ansätzen wie z.B. in den USA.

- **Tycho Möncks - Überblick Multiagentensysteme**
  Der Agent besitzt die zentrale Rolle der Simulation, daher beschrieb der Vortrag die grundsätzlichen Methoden und Modelle zum Design von Softwareagenten.

- **Alessandro Selvaggio - Verhandlungen in Multiagentensystemen**
  Im zweiten Teil der Projektgruppe ist das Ziel, mehrere Agenten in Verhand-

lungen Entscheidungen herbeiführen zu lassen. Der Vortrag diente dem Kennenlernen der gängigsten Konzepte zur Organisation von Kommunikation unter Softwareagenten. Das Blackboardkonzept, welches ebenfalls vorgestellt wurde, soll im 2. Semester der Projektgruppe umgesetzt werden.

- **Tobias Wagner - Lernen in Multiagentensystemen**
  Dieser Vortrag beschäftigte sich mit Realisierungen des Lernens in Multiagentensystemen. Sollte zum Abschluss der Projektgruppe noch Zeit vorhanden sein, so ist vorgesehen die komplexen sozialen Agenten auch Lernen zu lassen. Das Konzept des Case-Based Learning ist dabei eine mögliche Alternative.

- **Bastian Nau - Artificial Societies**
  Der Vortrag stellte bereits existierende Sozialsimulationen vor, was bei der Einordnung der Simulation helfen könnte.

- **Patrick Lenger - Algorithmen zur Ressourcenverteilung**
  Da es sich bei unserem Projekt um ein Problem der Ressourcenverteilung hadelt, stellte der Vortrag klassische Algorithmen zur Ressourcenverteilung mit ihren Vor- und Nachteilen vor.

- **Marc Schlipsing - Fuzzy Systems**
  Fuzzy Regeln spielen bei der Simulation eine zentrale Rolle. Sie werden sowohl zur Berechnung der Wahrscheinlichkeiten für die Werterwartungstabelle der Charaktertypen, als auch für die Berechnung von Änderungen in der Umwelt benötigt. Der Vortrag stellte die Konzepte der Fuzzy Logik vor und beschrieb grundsätzliche Methoden.

- **Nuhad Shaabani - Evolutionäre Algorithmen zur Optimierung**
  Evolutionäre Algorithmen dienen der Optimierung und können zum Abschluss der Arbeit der Projektgruppe genutzt werden, um die Sozialsimulation in Richtung einer effizienten Ressourcenverteilung zu optimieren. Der Vortrag stellte die bekanntesten Verfahren und Ausprägungen von evolutionären Algorithmen vor.

- **Christian Schwede - Learning Classifier Systems**
  Dieser Vortrag beschäftigte sich mit einer weiteren Realisierung des Lernens von Softwareagenten. Dabei wird eine binäre Codierung genutzt.

- **Jan Salmen - Expertensysteme und regelbasiertes Lernen** Bei der Erstellung der Simulation wird die Hilfe von Soziologen hinzugezogen. Da diese im Idealfall die Regelbasis erstellen, werden Ansätze eines Expertensystemes

realisiert. Der Vortrag stellte die Konzeption und Vor- und Nachteile von bestehenden logikbasierten Expertensystemen vor.

- **Philip Geismann - Spieltheorie**
  In der Wirtschaft stellt die Spieltheorie die zentrale Modellierung von Verhandlungsprozessen dar. Um die Augen nicht vor anderen Disziplinen zu verschließen, wurden die Ansätze zur Modellierung von Verhandlungen zwischen zwei Spielern der klassischen Spieltheorie vorgestellt.

- **Klaus Friedrichs - Evolutionäre Spieltheorie**
  Aufbauend auf dem vorherigen Vortrag wurde zum Abschluss eine Verbindung der Ansätze aus der Wirtschaft mit evolutionären Methoden präsentiert.

A.2 Seminare 2. Semester

In der Seminarphase des zweiten Semesters wurden nur kurze Seminare gehalten, die hauptsächlich dazu dienten, ähnliche Projekte kennen zu lernen. Dabei sollte in jedem Vortrag besonders darauf eingegangen werden, welche Teile des betrachteten Projekts Gemeinsamkeiten mit unserem Projekt aufweisen. Die Artikel, zu denen die Vorträge gehalten wurden, stammen größtenteils aus „JASSS - The Journal Of Artificial Societies and Social Simulation“ [JASSS].
In diesem Abschnitt befinden sich die kompletten Ausarbeitungen zu dieser Seminarphase:

A.2.1 Innovation Networks - A Simulation Approach [InNet], Klaus Friedrichs

In diesem Paper von Nigel Gilbert, Andreas Pyka und Petra Ahrweiler wird beschrieben wie mittels eines Multiagentensystems untersucht wird, wie Firmen Partnerschaften und Netzwerke bilden um effizienter Innovationen zu entwickeln. Agenten sind hierbei Firmen, deren Ziel es ist möglichst viel Geld zu verdienen. In jeder Runde macht jeder Agent ausgehend von seiner Wissensbasis einen Innovationsvorschlag. Dieser wird dann von einem Orakel nach festen Regeln, die die Agenten allerdings nicht kennen, bewertet und in Abhängigkeit von der Güte erhält jeder Agent Geld. Jeder Innovationsvorschlag kostet allerdings auch Geld und falls ein Agent kein Geld mehr hat, so ist er bankrott und scheidet aus. Ein Agent wird also versuchen zu lernen wie er eine erfolgreiche Innovation hervorbringt. Dies kann er entweder alleine

versuchen oder er kann sich einen Partner suchen, um mit diesem sein Wissen zu tauschen. Dabei wählen die Agenten Partner aus die eine ähnliche Forschungsrichtung haben und zusätzlich werden solche bevorzugt mit denen schon einmal eine Zusammenarbeit stattgefunden hat. Anschließend bringen die Partner eine gemeinsame Innovation hervor und teilen das Geld für diese untereinander auf. Partnerschaften werden immer nur für eine Runde gebildet, im Kontrast dazu können aber auch längerfristige Netzwerke gebildet werden. Wenn ein Agent mit einem anderen ein zweites mal eine Partnerschaft bildet, so lädt er diesen in sein Netzwerk ein. Falls dieser noch in keinem Netzwerk ist wird er annehmen und dem Netzwerk beitreten. Mit der Zeit entstehen so immer größere Netzwerke. Ein Netzwerk besteht so lange, bis es eine bestimmte Anzahl an Runden keinen Erfolg gehabt hat und wird dann aufgelöst. Dieses Szenario wurde dann mit zwei verschiednen Firmenstrukturen getestet. Die eine Struktur sollte den Kommunikationssektor und die andere den Biotechniksektor darstellen. Als Ausgabe wurde angezeigt wie viele Agenten es in jeder Runde gibt, wie viele mit einem Partner zusammenarbeiten und wie viele in einem Netzwerk sind. Hier zeigt sich aber, dass die Ergebnisse bei gleichen Vorraussetzungen sehr unterschiedlich ausfallen können, da vieles nicht deterministisch entschieden wird. Daher muss man an dieser Stelle auch an dem Paper Kritik äußern, denn anscheinend wurden nur wenige Testläufe durchgeführt, so dass die Ergebnisse wenig aussagekräftig sind.

A.2.2 The GeoGraph 3D Computational Laboratory [GEO], Philip Geismann

Das von Catherine Dibble und Philip G. Feldman an der Universität von Maryland entwickelte „GeoGraph 3D“ stellt eine grafische Erweiterung des „RePast“ - Frameworks dar. RePast (Recursive Porous Agent Simulation Toolkit) bietet ein Toolkit für die Modellierung von sozialen Interaktionen über Multi-Agenten-Systeme. Es stellt dem Anwender eine Vielzahl von Agenten mit vorgefertigten Verhaltensweisen zur Verfügung, die auch zur Laufzeit geändert werden können. Die eigentliche Simulation wird hierbei über verschiedene Scheduler gesteuert, die sowohl sequentielle als auch parallele Ereignisfolgen verarbeiten können. RePast existiert in mehreren Programmiersprachen und lässt sich von http://repast.sourceforge.net/ herunterladen; Beispielprojekte wie das bekannte „Game of life“ veranschaulichen die Funktionsweise. Im RePast-Framework gibt es die Möglichkeit, die Ergebnisse und Zwischenschritte mit rudimentären Mitteln zu visualisieren, was jedoch bei komplexeren Simulationen nicht immer ausreicht. Hier hilft GeoGraph, indem es verschiedene grafische Modellierungsmöglichkeiten anbietet, wobei die Parameter mit der Simulati-

on zusammenhängen und deren Auswirkungen unmittelbar anzeigen. Dabei verfolgt GeoGraph den speziellen Ansatz, die Simulationen bzw. Modellierungen in einem räumlichen Netzwerk anzusiedeln. Dieses Netzwerk kann von abstrakten, skalenfreien Umgebungen über Innenräume (von Gebäuden) bis hin zu Stadtvierteln oder virtuellen Städten reichen. In diesen dreidimensionalen Landschaften bewegen sich dann modellspezifische Agenten allein oder in Gruppen in bestimmte Richtungen, entlang von Netzwerkkanten oder Links. Für diese intuitive Art der Interpretation des Landschafts-Begriffs sollen hier drei Beispiele dienen: In einem Beispielszenario werden die Auswirkungen einer Epidemie innerhalb eines mehrere Städte umfassenden Gebietes untersucht. Einwohner werden durch Agenten modelliert, deren Verhalten davon abhängt, ob sie zu den gesunden, infizierten, kranken, toten oder immunen Individuen gehören. Anhand der vorgegebenen, möglicherweise sehr detaillierten und damit realistischen Landschaftsdaten (inklusive Verbindungen für die Agenten wie Busse oder Bahnlinien) kann nun untersucht werden, wie schnell oder raumgreifend sich eine virtuelle Epidemie ausbreiten würde. Dem zweiten Beispielszenario liegt ein Gitternetz zugrunde, das Straßenzüge und Blocks modelliert. Darin bewegen sich wiederum Agenten, die entweder zu den Zivilisten oder den Peacekeepern gehören. Mit einem individuellen Verhalten ausgestattet, entscheiden sich die Zivilisten eventuell, bestimmte Häuser zu plündern. Aufgabe der Peacekeeper ist es daher, sie davon abzuhalten, indem sie mit ihrer Präsenz für Abschreckung sorgen und auch Inhaftierungen vornehmen können. Die beiden oben genannten Beispiele fassen die Agenten als Individuen auf, die zwar durchaus gemeinsam ein Ziel haben mögen, jedoch nicht untereinander kommunizieren, um dieses Ziel zu erreichen. Im dritten Beispiel steht mehr die soziale Interaktion im Vordergrund (und wird daher vornehmlich visualisiert): Es werden verschiedene hierarchische Gruppen in einer Landschaft angesiedelt, deren Mitglieder jeweils einer festen Befehlskette entsprechend handeln. An der Spitze stehen die sog. Warlords, eine Schicht darunter folgen die Lieutenants, die wiederum den einfachen Soldaten Befehle erteilen. In jeder Schicht hat jedes Individuum nur seiner Schicht entsprechende Ziele, so ist der Warlord darauf bedacht, strategische Entscheidungen zu treffen und im Ansehen bei den anderen Warlords zu steigen (modelliert durch einen großen Nahrungsvorrat), Lieutenants kümmern sich um das Ausspähen von unbewachten Knoten, an denen möglichst viel Nahrung lagert, und den Soldaten schließlich obliegt das Einsammeln der Nahrungseinheiten. GeoGraph übernimmt hier die Aufgabe, diese Befehlsstrukturen grafisch zu veranschaulichen und die konkurrierenden Gruppen gegeneinander abzugrenzen. Verglichen mit dem AbSIntH-Projekt verfügt das RePast-Framework mit dem GeoGraph-Frontend über eine visuell deutlich attraktivere Erscheinungsform. Um sich diese zunutze zu machen, müsste das eigene Projekt die minimal notwendigen Schnittstellen zwischen

RePast und GeoGraph unterstützen. Das umzusetzen, dürfte kein leichtes Unterfangen werden, da es sich bei RePast um ein relativ umfangreiches Framework handelt. Da sich aber möglicherweise Eigenschaften von RePast für die im nächsten Semester anfallenden Aufgaben als nützlich erweisen könnten (Kommunikation zwischen Agenten, Gruppen von Agenten), ist eine weitere Beschäftigung mit RePast zum jetzigen Zeitpunkt durchaus als sinnvoll zu erachten.

A.2.3 Critical Incident Management [CIM], Patrick Lenger

Das Paper mit dem Titel *Critical Incident Management* von Scott Moss beschäftigt sich ebenso wie diese Projektgruppe mit einer Sozialsimulation. In dieser Simulation geht es um die Abbildung des Verhaltens von Managern, die für ein Wasserversorgungsunternehmen tätig sind und aufgrund von gemeldeten kritischen Zwischenfällen (wie z.B. niedriger Wasserdruck, verfärbtes Wasser, Ausfall der Stromversorgung) auf eine mögliche Ursache schließen müssen und entsprechende Gegenmaßnahmen (z.B. Reparaturen) anordnen müssen. Die Simulation beeinhaltet verschiedene Aufgabenbereiche: *Operations-Site* (liefert Messdaten), *Network-Controller* (Überwacht die Operations-Site), *Operations Control Centre* (sammelt die Messdaten und die von den Haushalten gemeldeten Vorfälle), *Customer Service Centre* (nimmt tagsüber Meldungen von Haushalten entgegen), *Control Systems* (Manager entscheiden, welche Gegenmaßnahmen bei Vorfällen zu treffen sind), *Work Planing and Scheduling* (plant die druchzuführenden Reparaturen) und *Repair Gang* (führt die Reparaturen durch). Dabei enthält der Aufgabenbereich *Control Systems* die Agenten um die es bei dieser Simulation eigentlich geht.

Die Zwischenfälle werden zum Teil von den angeschlossenen Haushalten gemeldet und zum Teil durch Messgeräte erfasst. Durch die Simulation werden zufällig Zwischenfälle ausgelöst, wobei diese evtl. weitere Zwischenfälle mit einer durch Experten festgelegten Wahrscheinlichkeit auslösen können (z.B. folgt auf einen Rohrbruch meist ein Druckabfall). Zu Beginn der Simulation kennen die Manager die Zusammenhänge zwischen Ursache (z.B. Rohrbruch) und Wirkung (z.B. Druckabfall) nicht und müssen im Laufe der Simulation diese Zusammenhänge erlernen, um die jeweilige Ursache in möglichst kurzer Zeit zu beheben.

In diesem Szenario haben die Agenten (also die Manager) bewusst nur solche Aufgaben, die von einem Individuum durchgeführt werden können. Der Grund liegt zum einen in der für Gruppenarbeit notwendigen Kooperation, die die Komplexität der Simulation erhöhen würde, und zum anderen an der Tatsache dass die Simulation einmal mit Informationsaustausch zwischen den Managern und einmal ohne durchgeführt wird. Auf diese Weise wird untersucht, inwiefern sich der Informati-

onsaustausch auf die Effizienz oder auf die Entwicklung eigener Erfahrung auswirkt. Der Austausch von Informationen wird derart realisiert, dass der Manager der seine Arbeitszeit beendet hat seine gewonnenen Erfahrungen dem anfangenden Manager mitteilt. Es findet kein Multicast zwischen Agenten statt.

Die Durchführung der Simulationen jeweils ohne und mit Austausch von Informationen lieferten experimentelle Hinweise, dass die Agenten durch den Informationsaustausch nicht andere Erkenntnisse erlernen, sondern weniger unnütze Erkenntnisse in ihrer Datenbank abspeichern (also die Effizienz erhöht wird).

Nachfolgend sollen die Gemeinsamkeiten und Unterschiede zwischen der Simulation von Scott Moss und der Projektgruppe kurz dargelegt werden.

Wie in der 1. Phase der Projektgruppe sind die Aufgaben eines Agenten in der Simulation von Scott Moss nur vom ihm zu bewältigen. Es ist also keine Teambildung zwischen Agenten möglich. Allerdings können im Gegensatz zu dem Ansatz der Projektgruppe innerhalb der 1. Phase die Agenten in diesem Szenario Informationen austauschen (auch wenn dies nur zwischen zwei aufeinanderfolgenden Agenten der Fall ist). Allerdings sind für die 2. Phase Kommunikation und Teambildung vorgesehen.

Der Aufbau eines Agenten ist bei beiden Ansätzen vollig unterschiedlich. Die Agenten in dem Wasserunternehmen kennen zu Beginn keine Zusammenhänge zwischen Ursache und Wirkung und lernen diese durch Vergleich der vor einer Aktion eingetretenen Wirkungen mit denen nach Durchführung der Aktion (also ob die Wirkung durch die Aktion verschwindet oder nicht). Demgegenüber besitzen die in der Projektgruppe modellierten Agenten feste Charaktertypen, die aufgrund einer Regelbasis eine Werterwartung berechnen und die Aktion mit der höchsten Werterwartung ausführen. Das Lernen der Agenten wird dabei nur über die Gewichtung der Charaktertypen realisiert und über die Möglichkeit Umweltsituationen in einem Gedächtnis abzulegen. Dabei hat das Lernen in dem von der Projektgruppe entwickelten Agenten eine untergeordnete Rolle (im Gegensatz zu dem Ansatz von Scott Moss). Aus der Existenz einer Regelbasis (bzw. mehrerer Regelbasen bei mehreren Charaktertypen) folgt, dass ein derartiger Agent zu Beginn mit Expertenwissen in Form von Regeln konfiguriert werden muss, was mit einem nicht unerheblichen Aufwand verbunden ist. Dies ist bei den Agenten bei der Simulation von Scott Moss nicht vonnöten.

Insgesamt lässt sich zusammenfassen, dass mit den beiden hier betrachteten Simulationen unterschiedliche Ziele verfolgt werden. Der Ansatz von Scott Moss konzentriert sich auf das Erlernen von Kombinationen aus Ursache und Wirkung, wobei zu Beginn keine Kombinationen bekannt sind. Der Ansatz der Projektgruppe zielt auf die Umsetzung von soziologischen Aspekten bei der Modellierung eines Agen-

ten ab. Zu den soziologischen Aspekten zählen die zu einem Agenten gehörenden Charaktertypen und ihre jeweilige Werterwartungstabelle.

A.2.4 Reciprocity, Normative Reputation, and the Development of Mutual Obligation in Gift-Giving Societies [Reci], Tycho Möncks

Der vorliegende Text beschäftigt sich mit sozialen Beziehungen in Gesellschaften, in denen eine wechselseitige Annahme über Ansehen vorherrscht. Hierbei soll auch untersucht werden, welchen Einfluss die Kommunikation von Ansehen haben kann. Die Ausarbeitung basiert auf der Veröffentlichung von Stephen Younger (2004): Reciprocity, Normative Reputation, and the Development of Mutual Obligation in Gift-Giving Societies (veröffentlicht im JASS-Journal). Die Bearbeitung erfolgte unter Berücksichtigung der für die PG 467 (Uni Dortmund) wesentlichen bzw. verwertbaren Aspekte.

Wechselseitige soziale Beziehungen spielen in fast allen Gesellschaften eine bedeutende Rolle. Derartige Interaktionen lassen sich in drei Gruppen einteilen. In frühen menschlichen Zivilisationen wurde z.B. das Ergebnis einer gemeinsamen Jagd unter dem gesamten Stamm aufgeteilt, ohne dass eine spezifische (wirtschaftliche) Gegenleistung erwartet wurde. Dieses Beschenken hatte vielmehr sozialen Charakter, erwartet wurde eine Verbesserung des sozialen Stati des Schenkenden. Diese Form der Interaktion wird im weiteren als positive Gegenseitigkeit bezeichnet. Handel ist eine Form der sozialen Interaktion, die einen Austausch von materiellen oder immateriellen Subjekten zum Gegenseitigen Nutzen vorsieht, wobei auch dem Handel eine soziale Funktion zukommen kann. Primär ist jedoch der ökonomische Nutzen. Handel ist eine Form der neutralen Gegenseitigkeit. Eine dritte Form ist die negative Gegenseitigkeit. Ihr Spektrum reicht vom Vorenthalten eines erwarteten Geschenks bis hin zum aktiven Stehlen. Eine Einheit der Gesellschaft kann sowohl ein einzelner Akteur als auch eine Gruppe sein. Die jeweils von einer Einheit der Gesellschaft ausgeübte Form der Interaktion beeinflusst seine Reputation (Ansehen) als auch seine wirtschaftliche Stärke. Die Entscheidung bezüglich der Ausführung einer Handlung setzt sich also aus einer Erwartung bezüglich ökonomischer und sozialer Konsequenzen zusammen. Die Bedeutung dieser beiden Belohnungsstrukturen sollen im folgenden in einer diskreten Agenten-Simulation untersucht werden.

Eingesetzt wurde die Agenten-Simulationssoftware MICROS. Die Agenten haben hierbei in einer vordefinierten und begrenzten Umwelt eine beschränkte, festgelegte Menge an Handlungsoptionen. Aktionen können nur sequentiell ausgeführt werden. Jeder Agent hat Bedürfnisse bezüglich Schlaf, Ernährung, sozialen Kontakten und

Arbeit. Nachwachsende Nahrungsvorräte befindet sich in der Umwelt, außerdem Ruhestellen an denen Agenten schlafen können. In dem Modell existieren zwei Arten von Agenten: Diebe (negative Gegenseitigkeit wird ausgeübt) und Händler (neutrales Gegenseitiges Verhalten). Trifft nun ein Agent auf einen anderen, muss er entscheiden ob er mit diesem handeln möchte oder nicht. Bei einem Handel gibt der Agent seinem Handelspartner Ware und erhofft sich eine Gegenleistung. Bleibt diese aus (weil er mit einem Dieb gehandelt hat), erleidet er einen Verlust. Wir jedoch eine Gegenleistung gewährt, haben beide Seiten einen zusätzlichen Nutzen erwirtschaftet. Darüber hinaus rauben Diebe andere Agenten aus, wenn diese schwächer sind als sie selbst. Jeder neutrale Agent hat Annahmen über alle anderen Agenten (sog. Interaktionsmatrix). Jede erfolgreiche Interaktion erhöht die Anerkennung des betroffenen Agenten in dieser Matrix, jeder Diebstahl reduziert diese. Ist die Reputation auf Null gesunken, wird mit dem entsprechenden Agenten kein Handel mehr betrieben (je nach Modell führt dies auch zu einer aktiven Flucht, um Diebstahl zu vermeiden). Ein Agentenleben dauert maximal 4000 Runden, wenn der Agent nicht vorher verhungert oder an Übermüdung stirbt. In Versuchen wurden folgende grundsätzliche Varianten untersucht:

1. Agenten können nicht untereinander kommunizieren.
2. Agenten kommunizieren untereinander die Reputation bezüglich anderer Agenten. Diebe verbreiten hierbei jedoch keine falschen Informationen, sie kommunizieren nicht. Tauschen zwei Agenten ihre Interaktionsmatrix aus, berechen die Agenten den Mittelwert beider Matrizen und nehmen diese Werte als neue Matrix.

Flucht vor bekannten negativen Agenten führte zu einer Verminderung der Stärke der negativen Agenten um den Faktor drei, die Handelsrate stieg deutlich an. Eine leichte Verringerung der Stärke der neutralen Agenten war zu beobachten (wegen beständiger Flucht ist weniger Zeit zum sammeln verfügbar). Kommunikation von Reputation stärkt die neutralen Agenten wesentlich. Insgesamt ist auffällig, dass negative Agenten in fast allen Szenarios eine deutlich höhere Fitness haben. Hierbei muss jedoch berücksichtigt werden, dass in der Simulation keine direkte Bestrafung für negative Interaktion (Diebstahl) vorgesehen ist. Lediglich die Reputation eines Agenten nimmt durch einen Diebstahl ab.

Anwendbarkeit der Ergebnisse auf die Projektgruppe

Die Untersuchung liefert keine wirklich überraschenden Ergebnisse, bestätigt aber die intuitive Meinung dass Kommunikation von Erfahrungen dem Wohl des Einzel-

nen zu Gute kommt. Schlechte Erfahrungen müssen so nicht von jedem Agenten aufs Neue gemacht werden. Gleiches gilt für die Meidung von Sozialkontakten mit Agenten, die eine negative Reputation aufweisen. Auch hier vermutet man intuitiv, dass die Meidung derartiger Kontakte einen Vorteil bringt. Dies wird durch die Simulation bestätigt. Kommunikation und eine neutrale Gegenseitigkeit führen zu einer Stärkung der Agentengesellschaft insgesamt und sind somit auch für den Einzelnen von Vorteil. Jedoch können innerhalb einer neutralen Gesellschaft einzelne negative Agenten für sich einen deutlichen Vorteil erarbeiten. Für eine realistische Simulation eines Hochschulszenarios sollte die Kommunikation von Reputation auf Grund ihres Einflusspotentiales berücksichtigt werden.

A.2.5 Simulating Norms, Social Inequality, and Functional Change in Artificial Societies [NJSAH], Christian Mosblech

Der Artikel „Simulating Norms, Social Inequality, an Functional Change in Artificial Societies“ von Nicole J. Saam und Andreas Harrer beschreibt die Nutzung einer normenbasierten Artificial Society im Vergleich zwischen einer soziologischen und einer durch die Informatik geprägten Sichtweise. Dabei werden zunächst die Unterschiede zwischen beiden Sichtweisen herausgestellt, anschließend mit Hilfe des Conte-Castelfranchi-Models ein Beispiel für ein soziologische Problem gegeben, welches implementiert wird und mit verschiedenen Ansätzen erneut simuliert wird. Abschließend werden die Ergebnisse ausgewertet und ein Ausblick auf zukünftige Ideen gegeben. Hierbei legen die Autoren viel Wert auf die „finder-keeper“-Norm (etwa: „Wer es findet darf es behalten“) und ein Verweis auf die Unzulänglichkeiten von soziologischen Norm-Modelle in System mit nicht gleich verteilter Macht gegeben.
Zwischen den Normen im soziologischen Sinn und der Nutzung in der Informatik wird wie folgt unterschieden: Die soziologischen Normen sind sich entwickelnde, definierbare Regeln, die das soziale Verhalten von Akteuren beeinflussen ([NJSAH] Kap. 2.1). Auf der anderen Seite sieht die Informatik Normen als ein Regelwerk, welches Agenten zu bestimmten Handlungen zwingt ([NJSAH] Kap 2.2). Hier ist ein klarer Vorteil der Soziologie gegenüber der Informatik erkennbar: Agenten in einem normbasierten System agieren nicht nutzlos, dumm oder selbstzerstörerisch ([NJSAH], Kap 2.6), wenn sie sich an globale Regeln halten müssen. Die Normen sind also sowohl Makro- als auch Micro-Objekte, da sie in jedem Agenten und in der gesamten Umwelt verwendet werden können ([NJSAH] Kap 2.5). Es ist also sinnvoll, Agenten zu kreieren, die unabhängig voneinander, Normen befolgen können. Sie haben dabei vier unterschiedliche Herangehensweisen an die Regeln: Zum einen können Normen

erstellen, empfangen, verteidigen oder nur beobachten. Die globalen Regeln dienen dabei entweder dazu, die Agenten untereinander zu koordinieren oder ihr interaggressives Verhalten zu kontrollieren.
Im Beispiel von Conte und Castelfranchi ist diese Theorie implementiert worden. Jeder Agent hat die Möglichkeit, sich zu bestimmten Kosten für oder gegen das befolgen von Regeln zu entscheiden. Unterschiedliche Agentenentscheidungen erzeugen einen Ruf, der jedem anderen Agenten zur Verfügung steht. Es wird ein finder-keeper-Szenario simuliert, in dem 50 Agenten auf einem 10x10 Felder großen Spielbrett zufällig verteilt werden. Zusätzlich werden 25 Essenseinheiten mit einem Wert von 20 ausgegeben. Die Agenten haben nun die Möglichkeiten, zu essen, sich zu bewegen oder einen anderen Agenten anzugreifen. Das Angriffsverhalten wird in drei Kategorien eingeteilt: blinde Aggressivität („Angreifen, wenn kein Futter mehr zu geringeren Kosten erhältich ist“), strategisches Vorgehen („Angreifen, wenn der Gegner schächer ist und wenn kein Essen zu kleineren Kosten verfügbar ist“) und das normative Verhalten („Nur angreifen, wenn das Attribut Besitzer vom Essen noch nicht gesetzt ist“).
Zunächst wurden Spiele mit Agenten gespielt, die alle das Gleiche Verhalten hatten. Es zeigte sich, dass die Agenten, die sich an die Normen hielten, das beste Ergebnis lieferten. Eine Mischung von Agenten aus zwei Verschiedenen Kulturen zeigt, dass das normative Verhalten in der Konkurennz das schlechteste Ergebnis lieferte.
Anschließend wurde einge kleine Veränderungen vorgenommen: Die Einführung der Fortpflanzung und des „privaten Besitztums“, ähnlich wie es in der Marxschen Theorie Verwendung findet. Mit diesen Methoden sollte die Ungleichheit der Agenten beschrieben werden (Besitz = Macht).
Das nächste Experiment wurde an ein biblisches Zitat angelehnt: Wenn eine neugenerierte Essenseinheit auf ein ein Feld fiel, auf dem bereits ein Agent ist, hat es einen höheren Wert als das normale Essen. Der Wert ist abhängig von der Stärke des Agenten. Hier zeigt sich, dass der normative Ansatz die höchste Ergebnis liefert, was im Umkehrschluss aber heißt, dass es zu einer immer stärkeren Ausprägung zwischen arm und reich kommt.
Abschließend zeigt sich laut den Autoren, dass in der Informatik bislang zwei Dinge der soziologischen Simulation durch Artificial Societies unberücksichtigt geblieben sind: Eine Machtverteilung wurde bislang nicht generiert, sondern durch Regeln aufgeprägt, und es fehlt die Möglichkeit, Normen in ihrer Gesamtbeeinflussung der Gesellschaft, ihrer Entsteheung, ihrer Entwicklung und ihrer Festlegung zu betrachten.

Bedeutung für die Projektgruppe

Das Ziel der Projektgruppe wird natürlich auch dadurch erreicht, dass den einzelnen Akteuren im MAS Regeln aufgeprägt werden, die sie zu Handlungen zwingen. Dies entspricht der undynamischen Sichtweise, der von den Autoren beschriebenen Definition von Normen in der Informatik. Die einzige Möglichkeit, sich hier weiter an die Sichtweise der Soziologie anzunähern, wäre eine dynamische Erweiterung der Regeln zu ermöglichen. So kann eine Betrachtung stattfinden, ob und wo sich populationsweite Regeln bilden und durchsetzen, also eine grundlegende Erkenntnis in jedem Agenten stattfindet. Der für die Umsetzung vorgeschlagene Ansatz, einzelne künstliche Intelligenzen als Agenten einzusetzen ist bereits durch das soziologisch basierte Modell der Agenten realisiert. Um nun eine weitere Annäherung der PG-Agenten reale Bedingungen zu schaffen, müsste jedem Agenten die Möglichkeit gegeben werden, ein Befolgen der Regeln zu bewerten und dies auch als eine Handlungsalternative zu nutzen. Da aber die Simulation eines Hochschulszenarios Ziel der Projektgruppe ist, sollte diese Möglichkeit nicht gegeben werden. Es ist im vorliegenden Fall nicht nötig, das Verhalten von Rebellen gegen ein bestehendes System zu betrachten.

A.2.6 DIAGAL: A Tool for Analyzing and Modelling Commitment-Based Dialogues between Agents [LAB04], Bastian Nau

Der Artikel beschreibt die Entwicklung des Kommunikations-Simulations-Programms DIAGAL, dass automatische Konversationen in Büros modelliert und analysiert. Die Aktionen der Agenten sind dabei: Versprechen machen, Fakten darlegen, nach Informationen fragen, u.ä. Über diese Aktionen können sie ihre Einstellung ändern und ihr gegenwärtiges uns zukünftiges Verhalten anpassen. Eine wichtige Fragestellung bei diesem Artikel ist, wie die die Teilnehmer einer Konversation zu einer Vereinbarung gelangen. Diese Fragestellung deckt sich mit dem Problem der PG, wie man eine Mehrheit bei einer Verhandlung findet.
Die Verhandlungsbeiträge in DIAGAL bestehen aus Verpflichtungen („Commitments“) und habe folgende Notation: $C(x, y, \alpha, t, s_x, s_y)$, was soviel bedeutet wie: x ist y zu α verpflichtet zur Zeit t unter den Sanktionen s_x und s_y. Dabei ist s_x die Sanktion wenn x seine Verpflichtung nicht einhält und s_y die Sanktion wenn y die Verpflichtung nicht annimmt.
Beispiel: $c_1 = C(Al, Bob, sing(Al, midnight), now, 10, 20)$ bedutet: Al verpflichtet sich gegenüber Bob, um Mitternacht zu singen. Macht er das nicht, muss er eine Strafe von 10 zahlen. Entzieht sich Bob dem Gesang, muss dieser eine Strafe von 20

zahlen.
Es gibt auch noch weitere, kompliziertere Notationsmöglichkeiten.
Die eigentliche Simulation besteht aus „Dialog-Spielen“. Diese enthalten Eingangsbedingungen, Erfolgsbedingungen für Initiator und Partner (die Ziele der Teilnehmer des Spiels), Fehlerbedingungen (unter denen der Dialog als gescheitert gilt) und Dialogregeln. Spiele können nacheinander, zufällig oder während ein Spiel läuft gestartet werden.
Die Autoren haben vier Basistypen von Spielen erstellt: 1. Request-Game: Der Initiator (I) erfragt seinen Partner (P) und dieser kann zusagen oder ablehnen. 2. Offer-Game: I macht ein Angebot, dass je nach Bedingung von P angenommen oder abgelehnt wird. 3. Inform-Game: P kann einer Aussage von I zustimmen oder nicht. 4. Ask-Game: Dabei stellt I eine Frage, auf die P dann eine korrekte Antwort geben kann.
Das Tool DIAGAL führt diese Spiele zusammen und verwaltet die Simulation. In einer Agenda werden alle Spielteilnahmen und Aussagen jedes Agenten festgehalten. Spiele können im Action Board mitverfolgt werden.
Als Beispiel haben die Autoren eine Simulation erstellt, die ein Sommerfestival planen soll. Die Agenten sind dabei *AgArtist* (Künstler, die eine Einladung zum Festival annehmen oder ablehnen können), *AgPlanner* (verantwortlich für das Einladen der Künstler, verwaltet das Budget und den Terminplan), *AgSecretary* (empfängt Aufträge von *AgPlanner*), *AgHotels* (verwaltet Hotelzimmer, die für die Künstler reserviert werden können) und *AgTravelAgency* (sucht Flugtickets für die Künstler).
Der grobe Ablauf sieht folgendermaßen aus: Der *AgPlanner* erstellt unter der Bedingung des Budgets eine Einladung für einen Künstler und fragt diesen, ob der die Einladung annimmt und ob er einen Flug und Hotelzimmer braucht. Die Wünsche des Künstlers werden an einen Agenten vom Typ *AgSecratary* weitergegeben, der mit *AgHotels* und *AgTravelAgency* Reservierungen vornimmt. Ist das erledigt, gehen die Ergebnisse der Reservierungen an *AgPlanner*, der sie als Bestätigung an den betreffenden *AgArtist* weitergibt.
Wichtig bei dieser Simulation ist, dass die Dialoge immer nur zwischen zwei Agenten ablaufen. Scheitert ein Dialog, hat das auch Auswirkungen auf Dialoge anderer Agenten.

A.2.7 Design Versus Cognition [DvsC], Jan Salmen

Im angegebenen Artikel wird das Zusammenwirken zwischen den Fähigkeiten einzelner Agenten und der Struktur eines Multiagentensystem untersucht. Anhand eines

Beispielszenarios können verschiedene Konfigurationen getestet und verglichen werde.

Motivation

Beim Entwurf eines Multiagentensystems sind verschieden Entscheidungen zu treffen. So kann die „Intelligenz“ jedes einzelnen Agenten durch unterschiedliche Methoden realisiert werden. Hier sollen daraus die folgenden, typischen untersucht werden:

- Entscheidungen auf Grund fester Regeln (→ Expertensystem)
- Fallbasiertes Lernen, also case-based-reasioning
- Randomisiertes Entscheiden, wobei die Wahrscheinlichkeit jeder Entscheidung durch das Vertrauen in ihre Richtigkeit festgelegt wird

Außerdem werden in den folgenden Experimenten auch Menschen im Szenario getestet. Sie bilden einen Vergleichspunkt und stellen die wohl ausgereifteste Intelligenz im Testfeld dar.

Außer der Entscheidung beim Design der einzelnen Agenten muss in einem Multiagentensystem auch die gesamte Struktur festgelegt werden. Hier bieten sich die Möglichkeiten

- Team - die Agenten treffen eine gemeinsame Entscheidung durch eine Abstimmung
- Hierarchie - Entscheidungen eines Agenten werden an die nächst höhere Stufe weitergegeben, bis irgendwo eine endgültige getroffen wird
- Gemischte Struktur: Agenten-Teams innerhalb einer Hierarchie

Untersucht werden soll nun, welche Kombinationen von Agenten-Typen und Struktur besonders gut sind, bzw. ob es eine in jedem Zusammenhang beste Wahl gibt. Dafür werden alle möglichen Kombinationen der oben genannten Optionen getestet.

Das Testszenario

Getestet werden die Multiagentensysteme in einem klassischen Testszenario, dem Radar-Problem. Dabei muss ein „Objekt“, dass durch 9 Eigenschaften beschrieben wird, die jeweils 3 Ausprägungen annehmen können, als freundlich, feindlich oder neutral klassifiziert werden.

Zunächst werden die verschiedenen Systeme mit 30 Fällen trainiert. Danach werden weitere 30 Fälle zu Evaluierungs-Zwecken benutzt.

Ergebnisse und Fazit

Es zeigt sich, dass die einzelnen Typen von Agenten in unterschiedlichen Strukturen ihre jeweils beste Leistungsfähigkeit entwickeln. Umgekehrt gibt es für jede Struktur einen bestimmten Agententyp, der sich als am besten geeignet herausstellt. Menschen erzielen die besten Ergebnisse in einer nicht-hierarchischen Teamstruktur.

Zusammenhang zu Absinth

Durch den bei Absinth gewählten, soziologischen Ansatz, stand von Anfang an fest, dass die Agenten im Sinne der Werterwartungsmaximierung entscheiden sollten. Somit wurden an dieser Stelle keine alternativen Ansätze betrachtet.

Die Struktur, innerhalb derer die Agenten Entscheidungen treffen, wird je nach gewähltem Szenario ebenfalls feststehen, so z.B. in Form einer Hochschul-Struktur.

Trotzdem kann man die beschriebenen Ergebnisse im Hinterkopf behalten, so, wenn zum Beispiel in einem simulierten Szenario au0ergewöhnliche, überraschende Ergebnisse auftreten.

A.2.8 Group Reputation Supports Beneficent Norms [Group], Marc Schlipsing

Einleitung

Der Artikel befasst sich mit einer künstlichen Gesellschaft, in der einzelne Agenten Entscheidungen aufgrund von Informationen über andere Agenten treffen. Dabei werden Agenten in Gruppen unterteilt, in der Eigenschaften für alle Agenten der Gruppe gelten bzw. dies angenommen wird. Hales greift hier ein spezielles Szenario nahrungssuchender Agenten auf, wobei er ein Modell anderer Autoren vorstellt und mit seiner leicht abgeänderten Version vergleicht. Dabei stellt sich heraus, dass lediglich quantitative Unterscheide im Ergebnis auftreten. Dort angekommen erweitert er das Modell um seinen Ansatz der Gruppen-Reputation.

Das Basis-Szenario

Zugrunde liegt ein Spielfeld der Größe 10×10 Felder. Auf einem Feld kann sich entweder ein Agent, eine Nahrungseinheit, beides oder nichts befinden. Anfangs werden 50 Agenten und 25 Nahrungseinheiten zufällig auf dem Spielfeld verteilt. Ein Agent kann seine 4 direkten Nachbarfelder sehen (Nahrung, Agent) und auf den 8 weiteren Feldern Nahrung riechen. Er beginnt bei einer Energie von 40 Punkten und verliert mit jeder Bewegung einen Punkt und gewinnt bei jeder Nahrungsaufnahme

20 Punkte. Die Aufnahme der Nahrung erfolgt in den Schritten Aufsammeln, Vorbereiten, Essen. Jeder Schritt benötigt einen Zyklus. Wird eine Nahrung gegessen, so entsteht eine neue Einheit zufällig auf dem Spielfeld, sodass die Anzahl der Nahrungseinheiten konstant bleibt. Neben dem Nichtstun hat der Agent auch noch die Möglichkeit einem Agenten eines Nachbarfeldes noch nicht gegessenen Nahrung abzunehmen (Raub). Dies kostet beide Agenten 4 Energiepunkte. Zuletzt gibt es noch einen Marker (Flag) für jede Nahrungseinheit, die angiebt wem sie gehört. Wird eine neue Einheit erzeugt, ist der nächststehende Agent der Besitzer. Im weiteren stellt Hales 3 Grundstrategien vor: Die Bewegung zur nächsten Nahrungseiheit ist intuitiv. Ist keine Einheit in Sicht oder zu riechen, so wird zufällig eine Bewegungsrichtung gewählt.

1. Blind - attackiert immer
2. Strategisch - attackiert nur, wenn der Gegner schwächer ist (weniger Energie)
3. Normativ - attackiert nur, wenn dadurch die Besitz-Norm nicht verletzt wird (Nahrung gehört dem Opfer nicht)

Die Simulation läuft grundsätzlich 100 mal, wobei jeder Lauf aus 2000 Zyklen besteht. Anschließend werden für jeweils jede Gruppe folgende Endergebnisse zur Bewertung herangezogen:

- Gesamtengergie
- Standardabweichung der einzelnen Energiewerte
- Aggressivität - Anzahl der Attacken auf andere Agenten

Reputation und Kommunikation

Das Basis-Szenario wird nun noch durch die Reputation erweitert. Jeder normativ handelnde Agent verwaltet ein Datum für jeden anderen Agenten, das aussagt, ob dieser schon einmal gegen die Besitznorm verstoßen hat. Tut dies ein Agent, wird also das Opfer ein Flag setzen, das den Angreifer für immer als Cheater markiert. Cheatern gegenüber verhält sich ein normativer Agent wie ein strategischer, also respektiert er ebenfalls nicht die Besitznorm.
Um diese Informationen schneller nutzen zu können werden die normativen Agenten darüberhinaus noch die Fähigkeit bekommen die gewonnen Informationen auszutauschen. Befinden sich 2 solche Agenten auf benachbarten Felden, teilen sie sich mit, welchen Agenten bereits als Cheater identifiziert wurden.

Auswertungen der Simulationen

Um einen grundsätzlichen Eindruck der Leistung der einzelnen Gruppen zu bekommen, lässt Hales nun einige Simulationen mit verschiedenen Agentenmischungen laufen und wertet diese dann stets gruppenweise aus. Zuallererst bestehen die 50 Agenten jeweils nur aus einer der 3 Grundtypen. Anschließend lässt er immer 25 Agenten einer Sorte gegen 25 einer anderen antreten. Vorerst jedoch ohne die Reputations-Flags und Kommunikation.
Die Ergebnisse, die sich dabei ergeben sind nicht weiter überraschend und auch an dieser Stelle nicht erwähnenswert. In den 25:25-Simulationen hat sich der strategische Ansatz klar durchgesetzt und der Normative als schwächster herausgestellt. Der Nachteil der norm-respektierenden Agenten ist das Verhalten gegenüber Cheatern. Um dieses zu verbessern werden nun noch die Gruppen NormRep (mit Reputationsflags) und NormRepCom (zusätzlich mit Kommunikation) auf die Strategen losgelassen. Dabei wird festgestellt, dass vor allem die NormRepComs zu besseren Ergebnissen kommen und im Bezug auf die Gesamtenegie nur noch knapp unterlegen sind. In allen Versuchsreihen zeichen sich die Gruppen der Normbefolger deutlich durch eine geringere Standardabweichung der einzelnen Agenten aus. Dieses Maß für eine soziale Spaltung der Gruppe ist bei den anderen Agenten meist 3mal so groß. Um noch bessere Ergebnisse mit den Normbefolgern zu bekommen, wird im Folgenden versucht, die anfängliche Unwissenheit zu beheben. Die NormRepComs müssten Cheater noch eher adäquat gegenübertreten können. Momentan ist für das Erkennen eines Cheaters immer nötig, dass ein normativer Agent praktisch wehrlos überfallen wird.

Gruppen-Reputation

Zu Beginn jeder Simulation werden die 50 Agenten (25 Strategen, 25 NormRepcoms) zufällig in 10 Gruppen unterteilt, wobei nun das Reputationsflag für eine gesamte Gruppe gilt. Verstößt eine Gruppenmitglied gegen die Besitznorm, wird die gesamte Gruppe vom Opfer als Cheater markiert. Dieser Ansatz stellt sich schnell als nicht besonders effektiv heraus. Dadurch, dass nun auch Normbefolger als Cheater markiert werden können, somit von anderen ihrer Art als solche behandelt werden und ebenfalls als Cheater bezeichnet werden, ergibt sich durch die Kommunikation der Flags eine Kettenreaktion. Der vorher beobachtete Gruppenzusammenhalt ist hier nicht menr zu beobachten.
Jedoch eine Verbesserung der Ergebnisse zeigt sich bei der Idee, die Gruppen a priori so aufzuteilen, dass nur Agenten mit gleicher Strategie in der selben Gruppe sind. Die Informationen werden nun wesentlich schneller erkannt und kommuniziert, so-

dass hier auch die Gesamtenergie der NormRepComs die der Strategen übersteigt! Leider fällt die Gruppeneinteilung sozusagen vom Himmel, da nicht klar wird woher das Wissen über solch eine artenspezifische Gruppierung kommen soll. Eine leichte Verunreinigung der Norm-Gruppe mit einem Cheater reicht schon aus, um das Resultat zu kippen.

Bewertung

Das Modell, mit der die Reputationen verwaltet werden ist recht statisch. Ist eine Gruppe einmal als Cheater erkannt, bleibt dies für immer so. Leider benutzt ein Agent die Information, die andere Agenten über ihn haben nicht. Wüsste er, wie ihn seine Nachbarn einschätzen, könnte er sicher besser und effizienter reagieren. Dieses Problem erkennt Hales, aber behandelt es nicht weiter.
Die Simulationen zeigen jedoch klar, dass eine wohltätig eingestellte soziale Gruppe nicht nur isoliert Erfolg haben kann. Wichtig bei einer Mischung der Charaktere ist es allerdings für jeden Agenten seinen Gegenüber richtig einschätzen zu können. Das Konzept der Stereotypen, wie es Hales nennt, nämlich die Bildung von Vorurteilen über eine Gruppe hat zwei Seiten. Bei richtiger Einschätzung bringt es große Vorteile für die sozial (wohltätig) handelnden Agenten. Findet jedoch eine Misklassifikation statt, so kann dies die ganze Gesellschaft kippen (Keiner traut keinem) und alle handeln nur noch zum eigenen Vorteil.
Für eine Optimierung ist dieser Ansatz sicher nicht geeignet. Allerdings scheint es sinnvoll, solches Gruppenansehen in eine Gesellschaftssimulation einzubeziehen. Dazu wäre aber ein etwas dynamischeres Modell nötig, was Hales als Reputation Management bezeichnet.

A.2.9 VIR-POX: An Agent-Based Analysis of Smallpox Preparedness and Response Policy [VIRPOX], Christian Schwede

Diese Arbeit von Benjamin M. Eidelson und Ian Lustick ist vor dem Hintergrund einer bioterroristischen Attacke durch Pockenerreger entstanden. Diskussionen zu diesem Thema wurden durch die Geschehnisse der nahen Vergangenheit wieder erneut entfacht und da es glücklicherweise wenig „Feldstudien“ zum diesem Thema gibt, bietet sich hier eine Simulation des Infizierungsablaufs als auch der Wirksamkeit entsprechender Gegenmaßnahmen an. Im Speziellen ging es besonders um die Simulation von verschiedenen Impfstrategien, um deren Wirkungen festzustellen. Betrachtet wurden hierbei die drei Hauptstrategien, die auch in den gegenwärtigen

Diskussionen thematisiert worden sind:

- Massenimpfung vor einer Attacke
- Massenimpfung nach einer Attacke
- Ringimpfung nach einer Attacke

Hierbei muss erwähnt werden, dass das Impfen aller Menschen vor einer Attacke sicherlich die sicherste Möglichkeit darstellt, aber es sich hier um einen relativ gefährlichen Impfstoff mit hohen Kosten handelt, weshalb heutzutage meist die Ringimpfung (Impfen aller Bekannten eines Infizierten) bevorzugt wird. In der Arbeit wurden entgegen den läufigen Simulationen auf diesem Gebiet („top-down“) ein „bottom-up“-Ansatz gewählt. Dass bedeutet es werden keine mathematischen Verbreitungsmodelle implementiert und deren Gesetzmäßigkeit von oben herab auf die einzelnen Individuen übertragen, sondern mit Hilfe einer Agentenbasierten Simulation, die einzelnen Individuen dargestellt um von ihrem Zusammenwirken auf die Auswirkungen „im Großen“ zu schließen. Die Agenten wurden nach dem ABIR (Agent- Based Identity Repertoire) Model konstruiert. Dieses Modell basiert auf der gesellschaftswissenschaftlichen „Constructivist identity theory“. Die Theorie besagt im Groben, dass die Gruppenbildung einer Gesellschaft über die Selbstbilder der einzelnen Mitglieder geschieht (gleiche Selbstbilder in einer Gruppe), diese Selbstbilder oder Identitäten werden aus einem Repertoire aus begrenzt vielen möglichen gewählt. Die Repertoires können sich nun entsprechend Konformitätsprinzipien/ Grundsätzen oder der Beobachtung von erfolgreichen Identitäten in der der Umwelt ändern. Das Vir-POX Modell besteht aus einem 42 * 42 Felder großen Bereich, diese Felder stellen verschiedene Agenten in verschieden Farben entsprechend ihrer Identität da. Die Simulation startet bei t = 0 und endet nach 5 Monaten bei t = 150, nach 8 Tagen (hier kann sich die zufällige Startverteilung stabilisieren) wird ein zufälliger Agent infiziert. Infizierte Agenten können den Virus an alle 8 umgebenden Agenten mit einer gewissen Wahrscheinlichkeit übertragen (abhängig von gemeinsamen Interessen). Der Erfolg einer Strategie wird anhand von zwei Messgrößen bestimmt:

- Anzahl infizierte Agenten / Anzahl nicht geimpfter Agenten (nach einem Durchlauf)
- Prozentsatz der Durchläufe nach x Durchläufen, in denen der Virus erfolgreich bekämpft wurde

Außerdem wird für jede zufällige Starteinstellung ein so genannter „seed“ abgespeichert, mit welchem man eben diese Situation wiederherstellen kann um so die Auswir-

kungen verschiedener Einflussgrößen testen zu können, ohne dabei von der Willkür verschiedener Startpopulation beeinfluss zu werden.

Die 19600 Durchläufe der Simulation bestätigten teilweise die in den Diskussionen eingebrachten Argumenten für eine Ringimpfung, zeigten aber gleichzeitig auf, dass der Erfolg einer solchen Impfung stark an bestimmte Vorbedingungen geknüpft ist. So ist es wichtig, dass man eine gewisse Grundresistenz von größer 40% gegen einen Virus in der Gesellschaft benötigt (natürlich vorhanden oder durch Impfungen vor der Attacke) und dass man die Infizierten schnell genug identifizieren kann, um ihre Bekannten zu impfen. Allgemein kann gesagt werden, dass sich einen Schutzstrategie nicht auf nur eine Maßnahme versteifen sollte, denn es gab bei keinem der gemachten Experimente eine Ausrottungsquote von 100% will heißen, eine ungünstige Ausgangsituation kann zu sehr schlechten Ergebnissen führen.

A.2.10 An Individual-Based Modeling Approach to Simulating Recreation Use in Wilderness Settings [Wild], Alessandro Selvaggio

In dem Paper „An Individual-Based Modeling Approach to Simulating Recreation Use in Wilderness Settings“ von Randy Gimblett, Terry Daniel und Michael J. Meitner wird ein agentenbasiertes System beschrieben mit dessen Hilfe man das Verhalten von Wanderern, Mountenbikefahrern und Jeepfahrern in gefährdeten bzw. stark besuchten Naturschutzgebieten simulieren kann.

Viele Naturfreunde wollen die unberührte Natur genießen. Allerdings werden diese Ausflüge oftmals durch die hohe Anzahl anderer Besucher beeinträchtigt, da so die Natur sehr stark beschädigt wird. Daher ist es wichtig der Natur genügend Zeit zu geben, um sich erholen zu können. Um dies zu ermöglichen ist es für die Planung wichtig das Verhalten der einzelnen Personen simulieren zu können. Die wichtigsten Daten sind in diesem Falle die Strecken und die Standorte an denen sich die einzelnen Gruppen treffen Daher. wurde in einer Studie der Simulator RBSim implementiert, mit dem man die einzelnen Personen mit den verschieden Charaktertypen mit Hilfe von Agenten simulieren kann. Die Agenten können sich jeweils selbständig bewegen und können sich Gruppen und die jeweiligen Strecken und Orte, an denen sie diese Gruppen treffen, merken. Als spezieller Ort den man simulieren wollte hat man sich den Broken Arrow Canion in der nähe von Sedona, Arizona ausgesucht, da dieser Ort, aufgrund der hohen Besucherzahl, beispielhaft für alle anderen gefährdeten Naturschutzgebiete simuliert werden soll. Um mit der Simulaiton beginnen zu können werden allerdings die verschiedenen Charaktertypen benötigt. Daher wurden insgesamt 1041 Besucher befragt, um eine ausreichende große Menge an Daten zu

erhalten. Bei dieser Auswertung stellte sich heraus, dass wir im Allgemeinen drei große Gruppen haben. Die Wanderer, die Mountenbiker und die Jeepfahrer mit jeweils eigenen Charakteren. Schließlich werden die einzelnen Aktuere in den RBSim geladen und mit diesen die Simulation gestartet. Die Zeit in der Simulation wird diskret dargestellt, wobei in jeder Zeiteinheit zufällig verschiedene Agenten erzeugt werden. Ebenfalls in jeder Zeiteinheit, werden die Agenten nacheinander vollständig ausgeführt. Das heißt, die Agenten scannen beim betreten einer neuen Zelle ihrer Simulationsumgebung das Gebiet und suchen nach ihrem Weg. Dabei berechnen sie ihre Geschwindigkeit in Abhängigkeit, ob sie auf- oder abwärts gehen. Natürlich wird hierbei auch gespeichert wann sie welche anderen Agenten auf ihren Weg getroffen haben. Weiterhin können die Agenten mit den anderen Agenten kommunizieren und Verhandeln (zum Beispiel wenn sie einen anderen Weg wählen wollen, weil sie einen überfüllten Standort umgehen wollen), sie haben Intentionen und Ziele (z.B. wo sie rasten wollen oder welche Sehenswürdigkeiten sie auf jeden Fall besichtigen wollen) und sie können adaptiv lernen. Die Umwelt wird durch das GIS visualisiert, wobei die Daten mit Hilfe von GPS Systemen ermittelt wurden. Mit dem GIS können die Abstände, die Zeit in der ein Agent unterwegs ist und sonstige wichtige Daten bestimmt werden.

Das System kann somit eingesetzt werden um sogenannte „what if“ Szenarios zu simulieren.

A.2.11 Introducing Emotions into the Computational Study of Social Norms [Emo], Tobias Wagner

Der Artikel „Introducing Emotions into the Computational Study of Social Norms: A First Evaluation“ von Alexander Staller und Paolo Petta beschreibt eine Sozialsimulation, die Emotionen als wichtigen Faktor der Entscheidungsfindung berücksichtigt. Dies wird motiviert durch den doppelten Zusammenhang zwischen Emotionen und Normen, welche bisher zusammen mit der Rationalistätsforderung den wichtigsten Faktor der Entscheidungsfindung darstellten. Die Autoren begründen dies damit, dass Emotionen wie Scham dafür sorgen, dass Normen eingehalten werden und Normen durch Emotionen motivierte Aggressionen unterbinden. Dabei gewinnen die Emotionen an Wichtigkeit, je stärker bestimmte Bedürfnisse verletzt werden.
Als Simulationsgrundlage dient ein Szenario, welches von Conte und Castelfranchi eingeführt wurde. Hierbei suchen Agenten in einem 10×10 Feld nach zufällig verteiltem Futter und haben die Wahl Agenten anzugreifen, um jenen Ihr Futter zu stehlen. Conte und Castelfranchi betrachten in dem Szenario nur die Fälle mit aggressiven Agenten und ohne Aggressionen. Hier setzen die Autoren an, indem sie die

Aggression ansteigen lassen, wenn längere Zeit nicht gegessen wurde und die Stärke mit der die Normen eingehalten werden verändern. Sie kommen für die Grenzfälle zu den selben Ergebnissen wie schon Conte und Castelfranchi, dass die durchschnittliche Stärke der Agenten am Höchsten ist, wenn die Normen Aggressionen verhindern und erhalten fallende Werte für die durchschnittliche Stärke für sinkende Werte der Normeinhaltung.
Der Artikel ist in soweit interessant, als das er Emotionen und Normen als wichtige Komponenten bei der Entscheidungsfindung und Interaktion von Agenten berücksichtigt und implementiert. Diese Elemente werden in unserer Simulation in Form der Charaktertypen „Homo Sociologicus“ und „Emotional Man“ ebenfalls berücksichtigt. Der Aufbau des Agenten in der von den Autoren benutzten $TABASCO_{JAM}$ Architektur ähnelt unserer Architektur, da auch hier ein Controller verwendet wird, welcher die Umweltinformationen an die Agenten weitergibt und die Aktionen der Agenten auf die Umwelt anwendet. Diese Aufgabe wird in unserer Simulation von dem Environment Package übernommen.
Über diese Parallelitäten hinweg ist der Artikel eher als uninteressant zu werten, da er nur bereits vorhandene Ergebnisse bestätigt und nicht selbst neue Ergebnisse hervorbringt. Das benutzte Szenario ist dabei sehr einfach und Emotionen und Normen beziehen sich jeweils nur auf den speziellen Aspekt der Futterbeschaffung. Da unsere Simulation eine weit höhere Komplexität besitzt und viele weitere Faktoren berücksichtigt kann nur eine geringe Teilmenge der Erkenntnisse aus der Interaktionen von Normen und Emotionen berücksichtigt werden, wobei dies voraussichtlich durch die Soziologen bei der Erstellung der Regelbasen geschehen wird.

A.3 Tutorien

Die Tutorien wurden in den ersten Wochen der Projektgruppe jeweils vor den Sitzungen durchgeführt. Sie dienten der Einarbeitung in die benutzte Software. Es wurden die folgenden Tutorien gehalten:

- **CVS / Subversion**
  Da bei der parallelen Arbeit von zwölf Leuten ein Versionisierungsprogramm notwendig ist, stellte das Tutorium die bestehenden Alternativen und ihre Anwendung vor.

- **LaTeX**
  Für die Erstellung der Dokumentationen soll LaTeX benutzt werden. Das Tutorium gab deshalb einen Überblick über mögliche Editoren für Windows und Unix und führte die grundlegenden Befehle vor.

- **Java 5.0**
  Zur Implementierung der Simulation wurde sich auf Java 5.0 geeinigt. Da Grundkenntnisse über Java bereits vorhanden waren, zielte der Vortrag besonders darauf ab, die Änderungen von Java 5.0 zu den Vorgängerversionen hervorzuheben.

- **Eclipse**
  Eclipse wird von der Projektgruppe als Entwicklungsumgebung eingesetzt. Im Tutorium wurden die Grundfunktionen der Software, sowie die Möglichkeit der Einbindung von CVS-, UML- und anderen Plug-Ins erläutert.

- **FuzzyJ**
  FuzzyJ ist eine Java-Klassenbibliothek, welche eine komfortable Einbindung von Methoden der Fuzzy Logik in Java erlaubt. Das Tutorium diente der Einarbeitung in grundlegende Befehle und Features der Bibliothek. Es wird in der aktuellen Simulation zur Berechnung der einzelnen Werterwartungen eingesetzt.

- **JADE**
  JADE ist ebenfalls ein Java-Package. Es stellt gewisse Grundtypen von Softwareagenten und deren Verhalten zur Verfügung. Ebenfalls sind dort Möglichkeiten für die Interaktion zwischen mehreren Agenten vorgesehen, was JADE interessant für den Einsatz im zweiten Teil der Projektgruppenarbeit macht. Das Tutorium stellte an Beispielen die Benutzung von JADE vor.

Anhang B

Handbuch

B.1 Einleitung

In diesem Kapitel wird die Bedienung von AbSintH beschrieben. Das Prorgamm lässt sich auf zwei Arten bedienen: Über die grafische Benutzeroberfläche (GUI) oder über die Direkteingabe der Daten in XML-Dateien. Der erste Teil des Handbuchs beschreibt die Funktionen der GUI, im zweiten Teil wird der Aufbau der XML-Dateien und die Vorgehensweise zum Starten der Simulation erläutert.

B.2 GUI

B.2.1 Allgemeines

Die Baumstruktur

Abbildung B.1: Der Baum, in dem sich alle Einträge befinden

Im linken Teil des Programmfensters befindet sich die Baumstruktur, in der alle Inhalte angezeigt werden, die für die Simulation benötigt werden (s. Abb. B.1). Dazu gehört als Oberkategorie das Szenario und die Kategorien Agenten, Charaktertypen, Ressourcen, Folgen, Aktionen, Ereignisse und Verhandlungen. Befindet sich in einer Kategorie noch kein Eintrag, ist diese als Blatt dargestellt. Sobald mindestens ein Eintrag vorhanden ist, erscheint sie als Ordner und man kann sie durch einen Doppelklick erweitern, um die Einträge anzuzeigen. Um die Details der Einträge zu sehen und zu ändern, klickt man den betreffenden Eintrag an bzw. wechselt mit den Pfeiltasten der Tastatur dorthin. Die Details der Kategorien werden in den folgenden Kapiteln behandelt.

Die Knopfleiste

Abbildung B.2: Die Knopfleiste zum steuern der grundlegenden Programmfunktionen

Die Knopfleiste befindet sich am oberen Rand des Programms und beinhaltet die Schaltflächen Neu, Laden, Entfernen und Start (s. Abb. B.2).

Neu Mit dieser Schaltfläche werden neue Einträge in den Baum eingefügt. Dazu klickt man auf die Kategorie oder einen Eintrag der Kategorie, zu der man einen neuen Eintrag hinzufügen möchte, und danach auf Neu. Es öffnet sich ein Fenster, in dem man den Namen des neuen Eintrags eingeben kann. Ist bereits ein Eintrag mit diesem Namen vorhanden, wird man zur Änderung aufgefordert.

Laden Um einen bestehenden Eintrag in eine Kategorie einzufügen, klickt man auf Laden und wählt aus dem Dateifenster die entsprechende Datei. Unter Dateityp kann man dabei die Kategorie des zu ladenden Eintrags auswählen.

Entfernen Möchte man einen Eintrag aus dem Programm entfernen, klickt man diesen an und danach auf Entfernen. Der Eintrag wird aus der Baumstruktur entfernt, aber nicht gelöscht. Das bedeutet, dass man ihn später über die Schaltfläche Laden wieder hinzufügen kann. Kategorien können nicht gelöscht werden.

Start Diese Schaltfläche startet die Simulation. Dabei wechselt die Ansicht zur Auswertung (s. Kap. B.3) und der Startknopf wird durch die Schaltflächen Stop und Pause ersetzt, mit der man die Simulation beenden oder anhalten kann.

B.2.2 Szenario

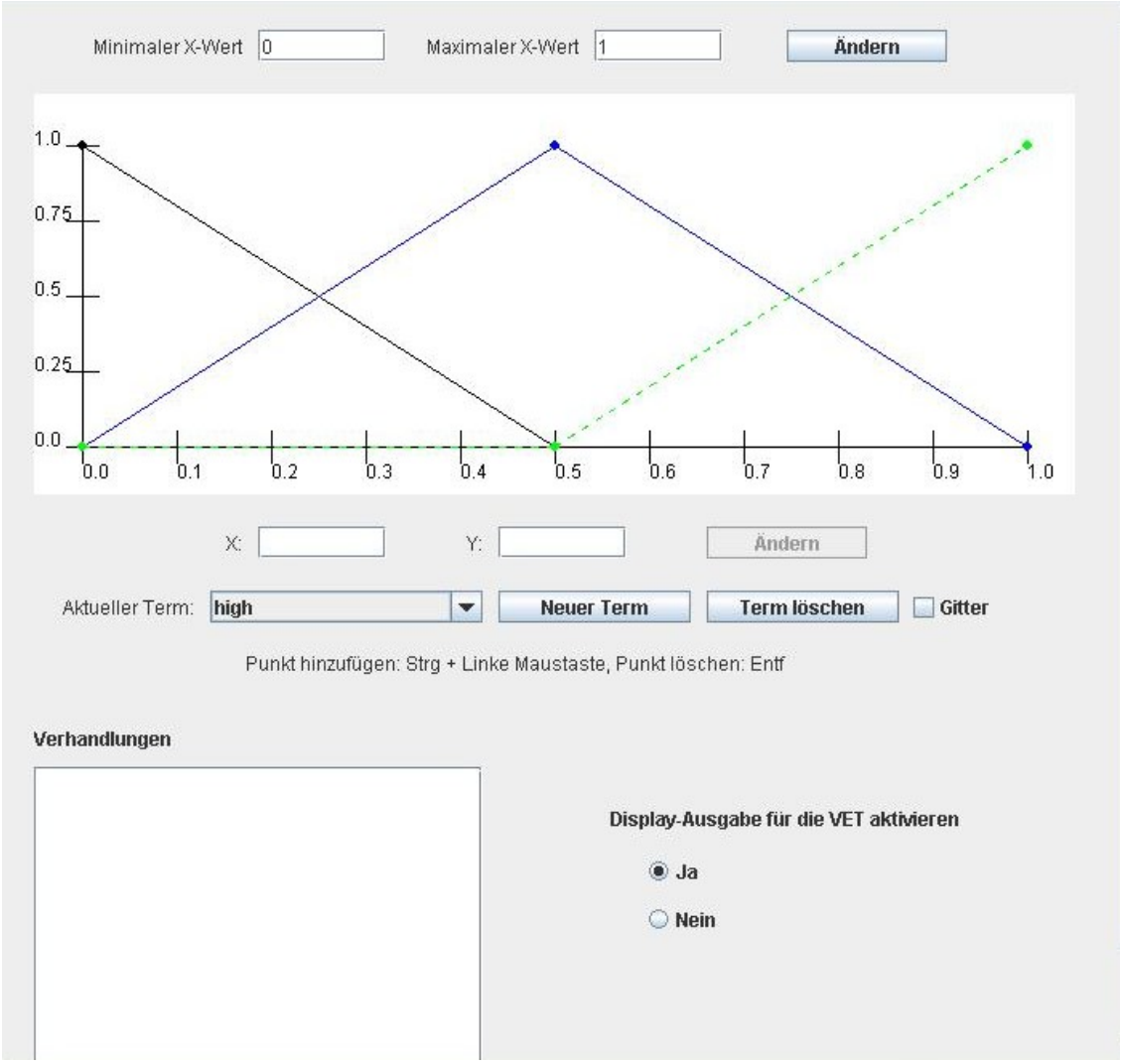

Abbildung B.3: Grundeinstellungen für das Szenario

Das Szenario stellt die Hauptkategorie des Programms dar. Für jede neue Si-

mulation muss deshalb ein neues Szenario angelegt werden. Um ein neues Szenario anzulegen, klickt man in der Menüleiste auf Datei → Neu. Im sich öffnenden Speicherfenster kann man nun den Namen und Speicherort des Szenarios festlegen.
Möchte man zwischen verschiedenen Szenarios wechseln, klickt man in der Menüleiste auf Datei → Öffnen und wählt ein anderes Szenario aus, um dieses in AbSintH zu bearbeiten. Mit einem Szenario werden automatisch alle zugehörigen Einträge der Kategorien geladen. Die Einträge lassen sich, wie in Kapitel B.2.1 beschrieben, verwalten.
Im Szenario können verschiedene Grundeinstellungen vorgenommen werden (s. Abb. B.3):

Globale linguistische Terme

Im Szenario werden die globalen linguistischen Terme festgelgt, die für die Wahrscheinlichkeit der Folgen in der Regelbasis der Charaktertypen verwendet werden (s. Kap. B.2.4). Dazu gibt man als Erstes den minimalen und maximalen Wert für die X-Achse ein und bestätigt mit Ändern. Als nächstes erstellt man Terme mit der Schaltfläche Neuer Term. Diese können z.B. „viel“, „mittel“ oder „wenig“ heißen. Zu jedem Term muss nun ein Graph erstellt werden, der die Zugehörigkeitsfunktion des Terms definiert. Dazu wählt man den Term aus, dessen Zugehörigkeitsfunktion man bearbeiten möchte und klickt bei gedrückter Strg-Taste in den Zeichenbereich. Dadurch wird ein neuer Punkt eingefügt, der sich dann an die Stelle ziehen lässt, die man gerne hätte. Alternativ kann man die Werte für einen Punkt auch in den Feldern unterhalb des Zeichenbereichs eingeben, nachdem der betreffende Punkt mit der Maus selektiert wurde. Es lassen sich pro Graph beliebig viele Punkte angeben. Punkte können mit der Entf-Taste gelöscht werden.

Verhandlungen

In das Textfeld Verhandlungen lassen sich per Drag&Drop die Verhandlungen hineinziehen, die in diesem Szenario durchgeführt werden sollen.

Display-Ausgabe

Mit der Wahl zwischen Ja oder Nein legt man fest, ob man während der Ausführung der Simulation eine Display-Ausgabe der Werterwartungstabelle mit Zwischenergebnissen haben möchte oder nicht. Achtung: Die Ausgabe erscheint für jede Entscheidung und verlangsamt dadurch die Simulation erheblich.

B.2.3 Agenten

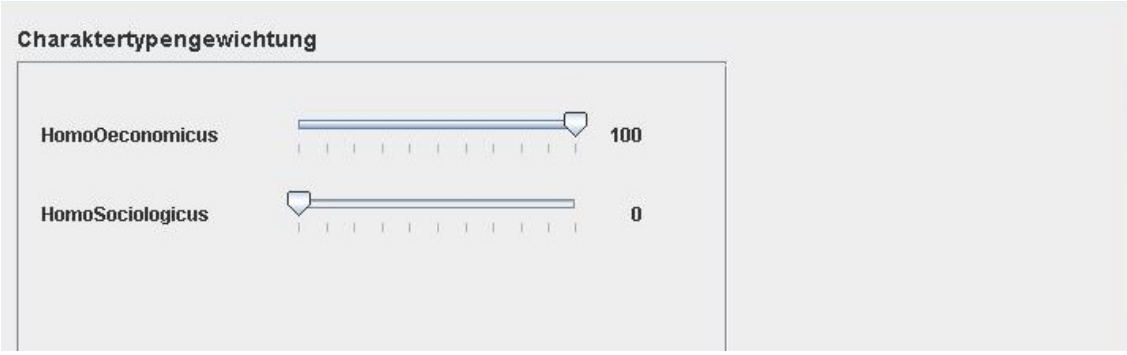

Abbildung B.4: Einstellung der Charaktertypengewichtung für Agenten

Im Agentenbereich kann man die Gewichtung der Charaktertypen für jeden Agenten angeben (s. Abb. B.4). Über die Schieberegler wird der Anteil angegeben, den ein Charaktertyp im Agenten einnimmt. Die Summe der Gewichte muss nicht zwangsweise 100 ergeben, da sie für die Simulation normiert werden (s. Kap. B.2.4). Diese Einstellungen lassen sich erst vornehmen, wenn Charaktertypen vorhanden sind (s. Kap. 4.3).

B.2.4 Charaktertypen

Für jeden Charaktertypen existiert eine Werterwartungstabelle, die aus Regeln besteht, aus denen die Wahrscheinlichkeit für eine Folge bei einer gegebenen Aktion berechnet wird (s. Kap. 4.3.2, Abb. B.5). Um die Regeln für die Berechnung eines Eintrags der Werterwartungstabelle anzugeben, klickt man auf die entsprechende Schaltfläche im Schnittpunkt der entsprechenden Folge und Aktion. Sind noch keine Regeln für einen Eintrag vorhanden, ist die Schaltfläche leer. Bei vorhandenen Regeln, erscheint ein Kreuz auf der Schaltfläche.

Regelbasis der Charaktertypen

Eine Regel wird in mehreren Schritten erstellt (s. Abb. B.6). Im ersten Schritt müssen die Ressourcen ausgewählt werden. Dazu zieht man diese aus der Baumstruktur in das Feld Ressourcen der Regelbasis. Es öffnet sich ein Fenster, in dem man den linguistischen Term der Ressource festlegen kann (wird für jede Ressource definiert, s. Kap. B.2.5). Die Einträge in der Ressourcen-Liste können mit der Entf-Taste wieder gelöscht werden. Hat man alle benötigten Ressourcen eingegeben, wählt man im

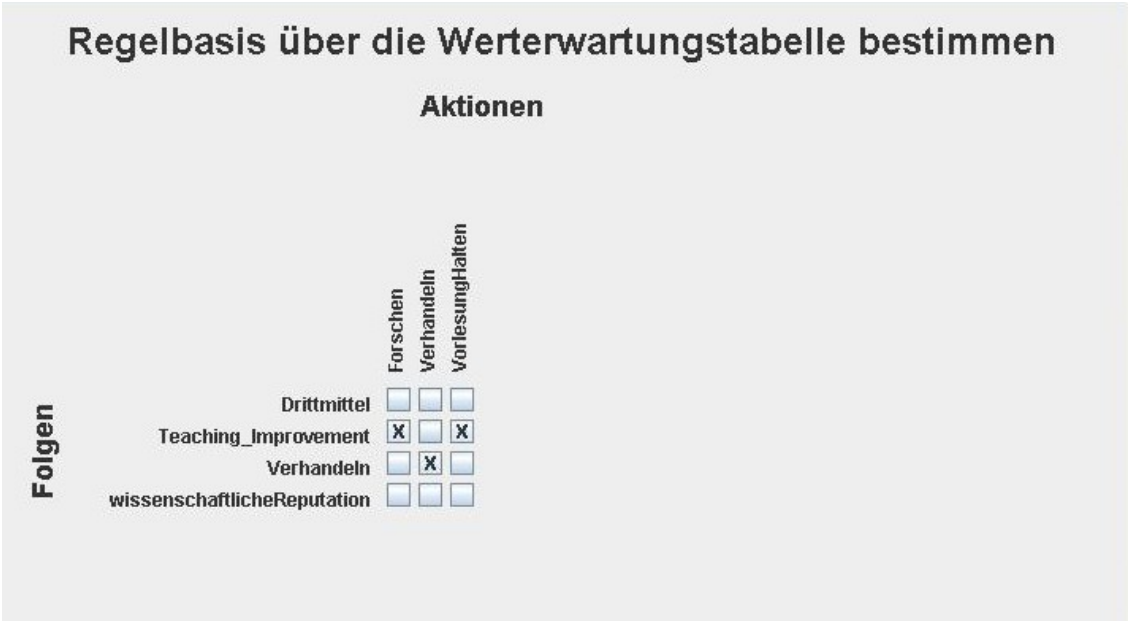

Abbildung B.5: Werterwartungstabelle für Charaktertypen

rechten Teil unter Wahrscheinlichkeitswert über den Schieberegler den Wahrscheinlichkeitswert (wird unter Szenario festgelegt, s. Kap. B.2.2) für das Eintreten einer Folge zu einer bestimmten Aktion aus (s. Kap. 4.3.3). Jetzt kann man die Regel durch klicken der Schaltfläche Hinzufügen zu den bestehenden Regeln hinzufügen und damit abspeichern. Zu jedem Eintrag der Werterwartungstabelle lassen sich beliebig viele Regeln erstellen.
Zum nachträglichen Bearbeiten einer Regel, wählt man diese im Bereich Bestehende Regeln aus und klickt dann auf Bearbeiten. Möchte man die linguistische Variable einer Ressource ändern, muss man diese aus der Ressourcen-Liste entfernen und danach wieder hinzufügen.
Um eine bestehende Regel zu löschen, wählt man diese aus und drückt die Entf-Taste.

B.2.5 Ressourcen

Im Fenster Ressourcen werden die Einstellungen für linguistische Terme, Startwerte, Typ, Änderungsrechte und Sichtbarkeiten festgelegt (s. Abb. B.7).

Abbildung B.6: Anlegen der Regeln für einen Charaktertypen

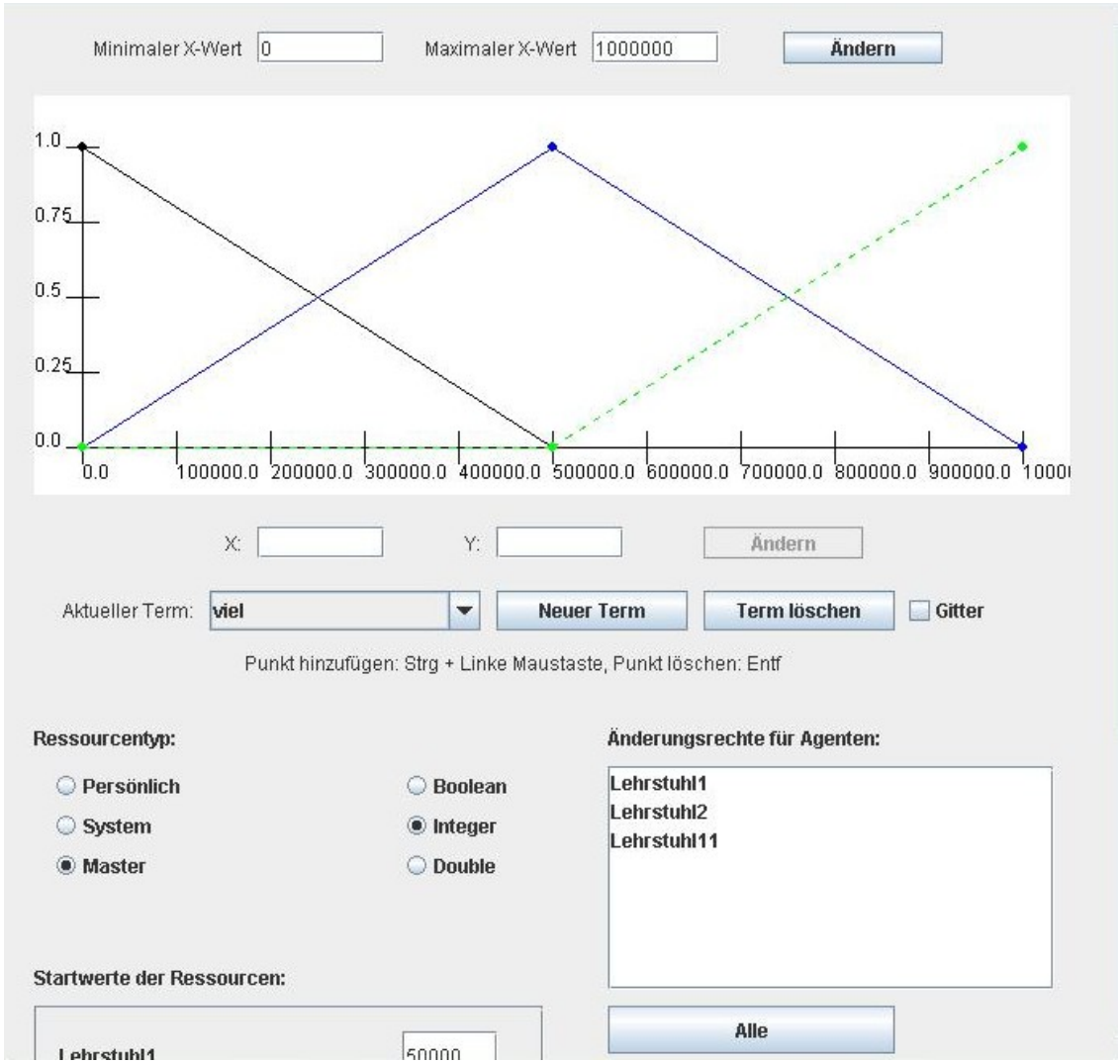

Abbildung B.7: Anlegen der Ressourcen des Szenarios

Linguistische Terme

Hier werden die Wahrscheinlichkeitsterme über einen FuzzyJ-Graph für jede Ressource festgelegt. Die Erstellung erfolgt analog zu der Erstellung der Terme im Szenario (s. Kap. B.2.2).

Ressourcentyp

Der Typ, um den es sich bei der Ressource handelt. Eine Beschreibung der Typen befindet sich im Kapitel 4.1.1.

Wertebereich

Der Wertebereich einer Ressource kann vom Typ Boolean, Integer oder Double sein.

Änderungsrechte

Hier wird festgelegt, welche Agenten das Recht haben, diese Ressource zu verändern. Die Agenten werden per Drag&Drop in das Textfeld gezogen. Über die Schaltfläche Alle können alle Agenten gleichzeitig hinzugefügt werden. Mit der Entf-Taste können Agenten wieder entfernt werden.

Startwert

Der Startwert einer Ressource ist der Wert, den die Ressource zu Beginn der Simulation hat und wird abhängig von seinem Typ angegeben. Bei einer persönlichen Ressource muss neben dem Startwert auch der Besitzer angegeben werden. Eine System-Ressource hat keinen Besitzer, sondern nur einen Startwert. Bei einer Master-Ressource muss zu jedem Agenten der Startwert angegeben werden.

Sichtbarkeit

Unter Sichtbarkeit für Agenten wird festgelegt, welche Agenten die Ressource bzw. deren Wert sehen können. Diese Einstellung ist auch abhängig vom Typ der Ressource. Bei persönlichen und System-Ressourcen kann man Agenten per Drag&Drop in das Textfeld ziehen. Über die Schaltfläche Alle können alle Agenten gleichzeitig hinzugefügt werden. Mit der Entf-Taste können Agenten wieder entfernt werden. Bei Master-Ressourcen kann man wählen, ob nur der Besitzer die Ressourcen sehen darf oder alle Agenten.

B.2.6 Folgen

Abbildung B.8: Eingabe der Nutzenwerte für Charaktertypen

In diesem Fenster kann man die Nutzenwerte für die einzelnen Charaktertypen für jede Folge angeben (s. Kap. 4.3, Abb. B.8).

B.2.7 Aktionen

Hier werden die Einstellungen für die Aktionen vorgenommen, die in der Simulation ausgeführt werden können (s. Abb. B.9).

Dauer und Wiederholungen

In diesen beiden Feldern wird angegeben, wie lange eine Aktion dauert und wie oft sie wiederholt wird. Dabei bedeuten 0 Wiederholungen, dass die Aktion ein Mal ausgeführt, aber nicht wiederholt wird.

Ereignisse

Ereignisse können mit der Maus in Ausgewählte Ereignisse gezogen und mit der Entf-Taste wieder gelöscht werden (s. auch Kap. B.2.8).

Ressourceneigenschaften

Hier wird festgelegt, welchen Wert eine Ressource als Voraussetzung zur Ausführung einer Aktion benötigt und um welchen Wert sich die Ressource nach der Ausführung ändert.

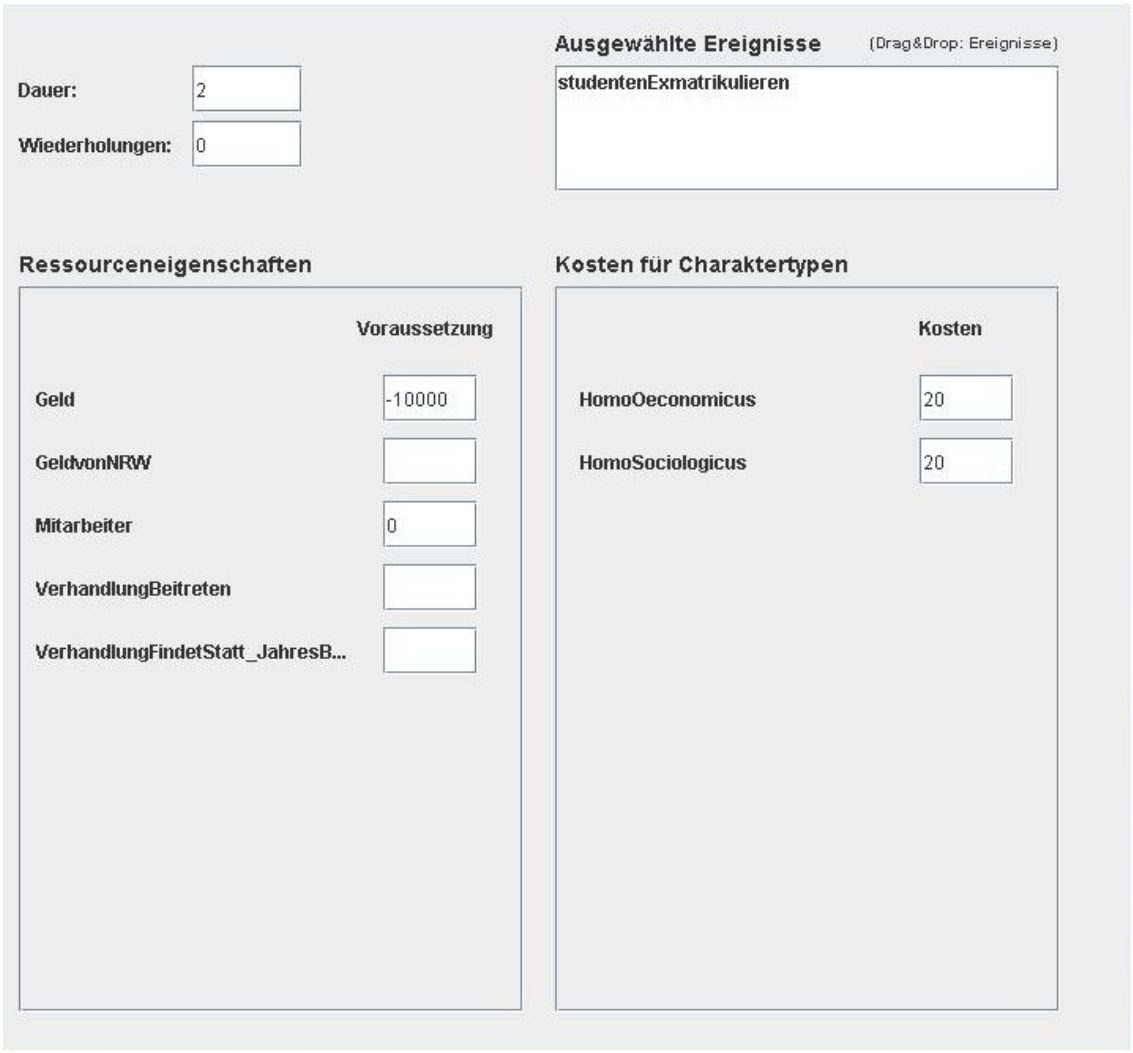

Abbildung B.9: Einstellungen für Aktionen

Kosten

Für jeden Charaktertypen entstehen bei Ausführung einer Aktion Kosten, die in diesem Feld festgelegt werden.

B.2.8 Ereignisse

Abbildung B.10: Werterwartungstabelle für Ereignisse

Ereignisse regeln die Veränderung von Ressourcen bei Ausführung einer Aktion. Dazu können zu jedem Ereignis für jede Ressource Regeln angegeben werden (s. Abb. B.10). Als erstes zieht man dazu, wie schon bei der Regelerstellung für Charaktertypen (s. Kap. B.2.4), die benötigten Ressourcen in die Ressourcenliste. Im rechten Teil unter Ergebnis als linguistische Variable erstellt man nun eine Veränderungsregel für dieses Ereignis. Dazu hält man Strg gedrückt und klickt mit der linken Maustaste in den Zeichenbereich, um einen neuen Punkt hinzuzufügen. Die Punkte lassen sich mit der Maus verschieben und der Entf-Taste löschen. Zum Hinzufügen der Regel klickt man auf die Schaltfläche Hinzufügen, zum Bearbeiten auf Bearbeiten. Bestehende Regeln können mit Entf gelöscht werden (s. Abb. B.11).

B.2.9 Verhandlung

Für eine Verhandlung benötigt man die verhandelnden Agenten und Ressourcen, über die verhandelt wird (s. Abb. B.12). Agenten werden mit der Maus in die Liste Verhandelnde Agenten gezogen, Ressourcen in die Liste Zu verhandelnde Ressourcen. Für jede Ressource muss dabei noch angegeben werden, um welchen Wert verhandelt wird und von welcher anderen Ressource (Quell-Ressource) dieser Wert abgezogen werden soll. Wird als Quell-Ressource Keine ausgewählt, wird der zu verhandelnde Wert von keiner vorhandenen Ressource abgezogen, sondern kommt von außerhalb

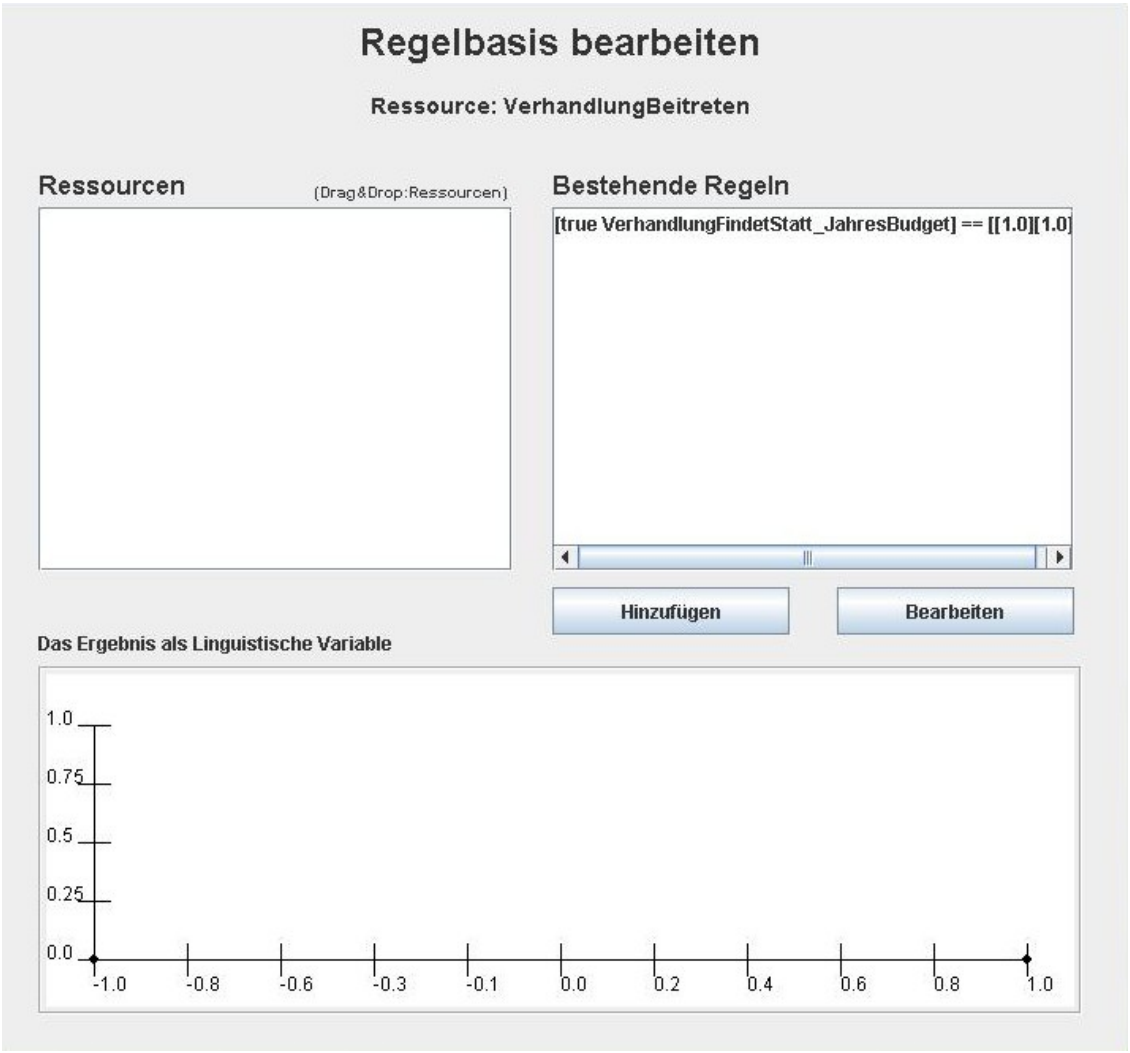

Abbildung B.11: Regelerstellung für Ereignisse

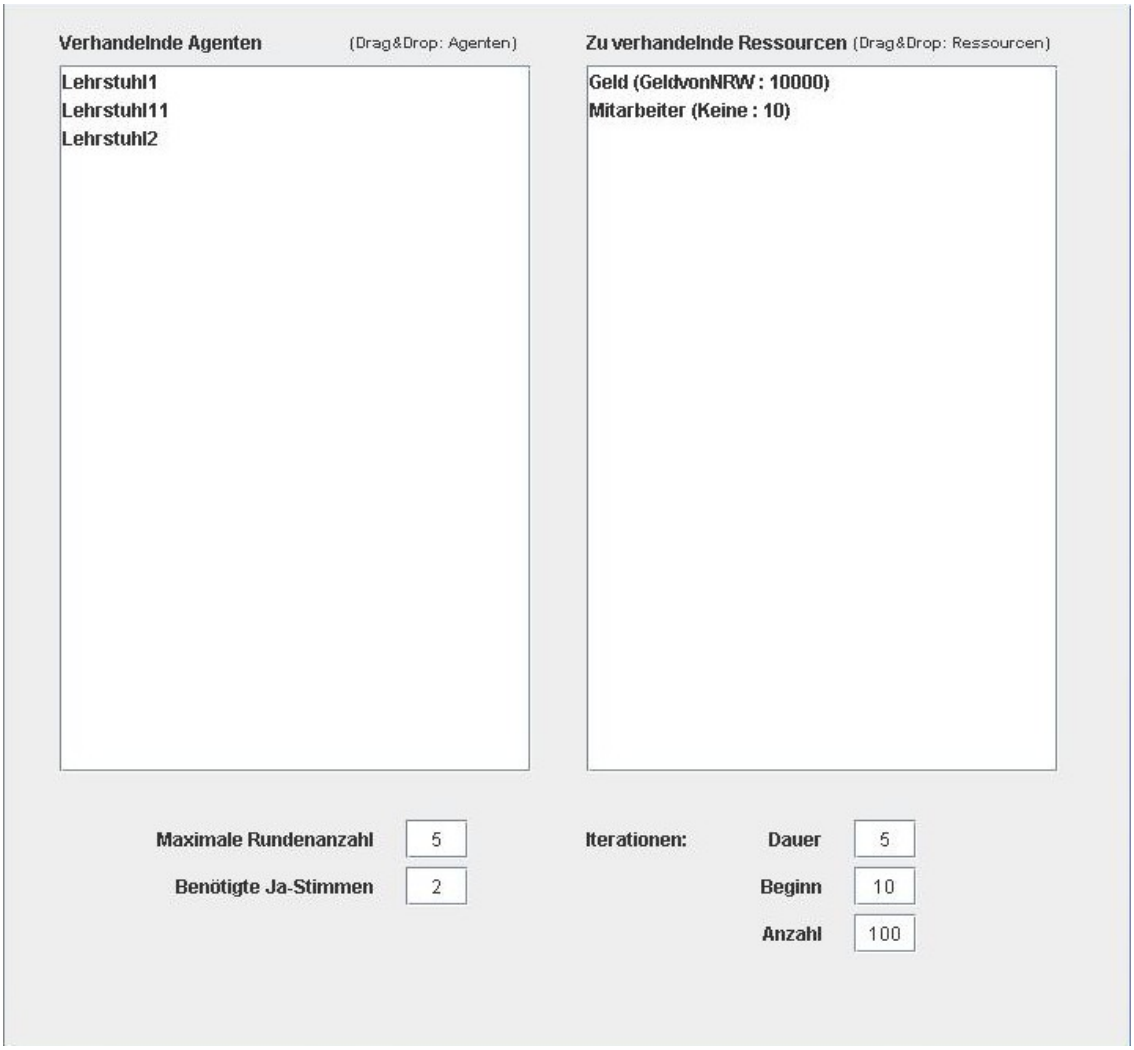

Abbildung B.12: Anlegen der Verhandlungen

neu in die Simulation hinzu.
Im unteren Bereich wird die maximale Rundenzahl angegeben, nach der eine Verhandlung als gescheitert abgebrochen wird.
Eine Verhandlung ist erfolgreich, wenn die eingestellten „Ja“-Stimmen erreicht werden.

Iterationen

Hier wird festgelegt, wann die Verhandlung innerhalb des Simulationslaufs ausgeführt wird (Beginn), wieviel Zeit zwischen zwei Verhandlungen vergeht (Dauer) und wie oft sie ausgeführt werden kann (Anzahl).

B.3 Auswertung

B.3.1 Simulation starten

Nachdem man alle Daten vollständig in das Programm eingegeben hat, startet man die Simulation mit dem Startknopf in der Knopfleiste (s. Abschnitt B.2.1). Anstelle des Startknopfes erscheinen jetzt die Schaltflächen Pause und Stop, mit denen man die Simulation unterbrechen oder anhalten kann (s. Abb. B.13). Zusätzlich öffnet sich

Abbildung B.13: Knopfleiste nach Starten der Simulation

nach dem Start ein Fenster, in dem man die Art der Auswertung auswählen kann (s. Abb. B.14). Dabei wird unterschieden zwischen Auswertung für Ressourcen, Aktionen, Verhandlungen und VETs. Man kann sich auch alle Auswertungen gleichzeitig anzeigen lassen. Je nach Auswahl öffnen sich nach dem Anhalten der Simulation über den Stop-Knopf ein oder mehrere verschiedene Fenster, die in den folgenden Abschnitten näher erklärt werden.

B.3.2 Auswertung für Ressourcen

Im Auswertungsfenster für Ressourcen (s. Abb. B.15) findet man zu jeder Ressource einen Graphen, der den Wert der Ressource über die Zeit anzeigt. Auf der rechten Seite befinden sich Schaltflächen, mit denen man die anzuzeigende Ressource

Abbildung B.14: Auswahlfenster für Auswertungsmethode

auswählen kann. Mit den Schaltflächen Vergrößern und Verkleinern am unteren Rand lässt sich in den Graphen hinein- und herauszoomen.

B.3.3 Auswertung für Aktionen

Das Fenster Auswertung der Aktionen (s. Abb. B.16) zeigt die Häufigkeit der Aktionen für jeden Agenten und die Verteilung über der Zeit. Der obere Teil des Fensters enthält einen Zeitstrahl, über dem die einzelnen Aktionen als Farbmarkierung für jeden Agenten und für das System aufgetragen sind. Daran kann man erkennen, welcher Agent zu welchem Zeitpunkt welche Aktion durchgeführt hat. Die Legende der Farbmarkierungen befindet sich oben rechts. Im unteren Bereich des Fensters befindet sich eine Aufstellung über die Verteilung der Aktionen. Das linke Kreisdiagramm gibt die Anzahl der Aktionen für jeden Agenten einzeln wieder, wobei man den gewünschten Agenten über das Dropdown-Menü auswählen kann. Das rechte Diagramm stellt die Gesamtverteilung über alle Agenten dar.

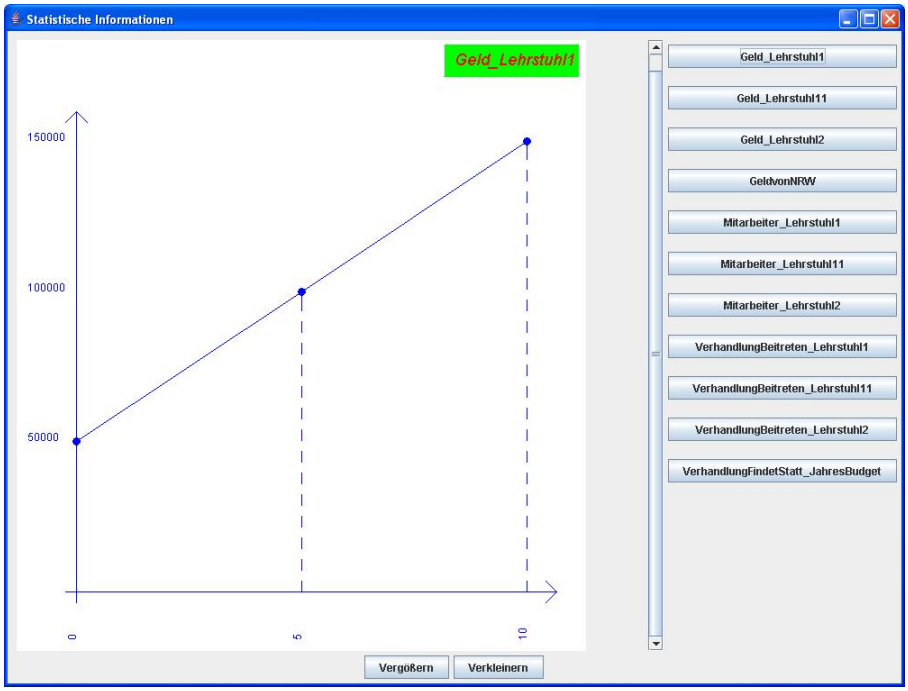

Abbildung B.15: Auswertung der Ressourcen

B.3.4 Auswertung für Verhandlungen

Im oberen Bereich des Fenster befinden sich links die Teilnehmer der Verhandlung und rechts die Ressourcen mit ihrem Wert über die verhandelt wird. Direkt darunter steht die Ausgangssituation, also alle Agenten mit ihren Ressourcen und deren Wert vor der Verhandlung. Darunter folgen alle Runden der Verhandlung. Der Aktive Agent ist dabei derjenige, der den Vorschlag für eine Verteilung in dieser Runde gemacht hat. Der Vorschlag befindet sich direkt unter ihm und darunter folgt das Abstimmungsergebnis. Wird ein Vorschlag angenommen, bricht die Verhandlung und somit auch die Auswertung für diese Verhandlung ab. Bei weiteren Verhandlungen wird die Auswertung für diese fortgesetzt (s. Abb. B.17.)

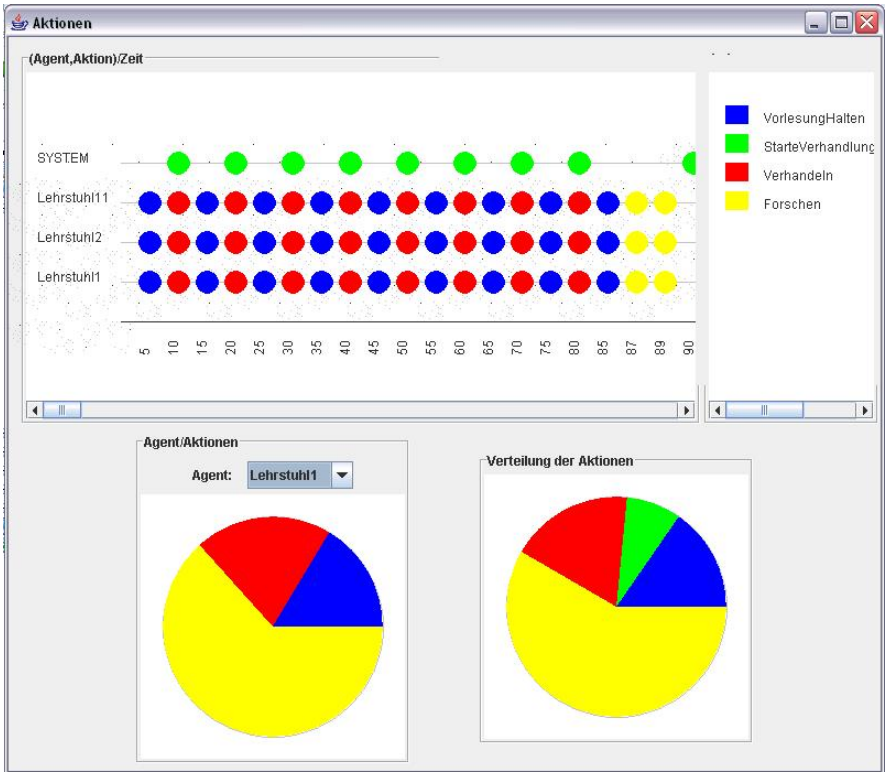

Abbildung B.16: Auswertung der Aktionen

B.3.5 Auswertung für VETs

In dieser Auswertung wird nach jeder Runde die Werterwartungstabelle eines Agenten für seine jeweiligen Charaktertypen angzeigt (s. Abb. B.18). Links befinden sich die Folgen der Simulation und oberhalb der Tabelle die Aktionen. Gelb markiert ist die Gesamtwerterwartung eines Charaktertypen an eine Aktion.

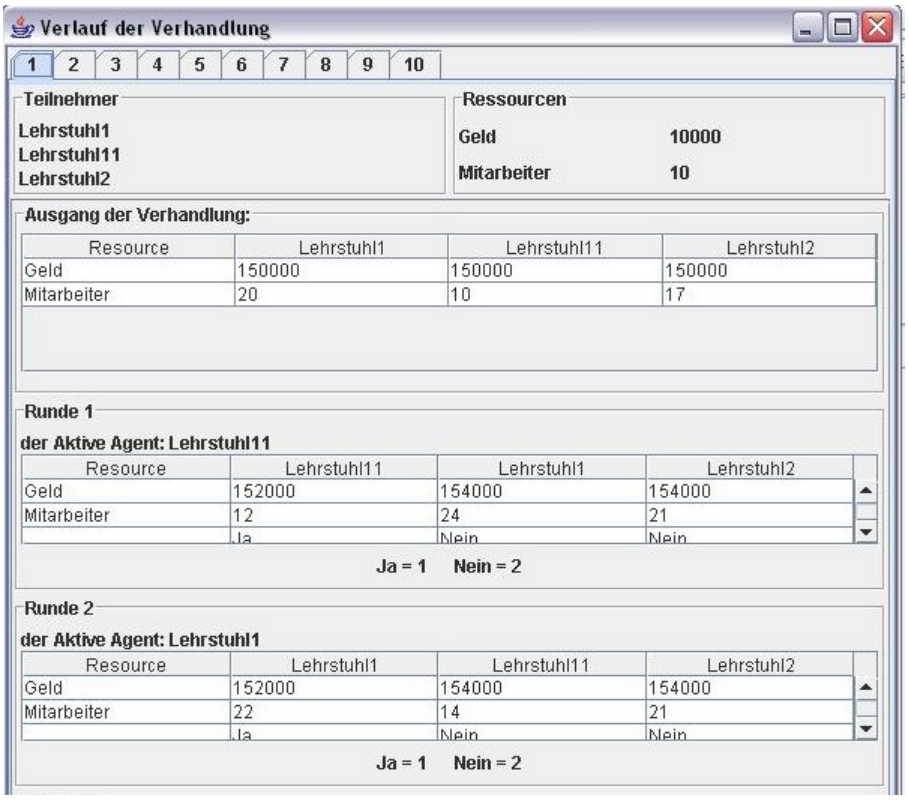

Abbildung B.17: Auswertung der Verhandlungen

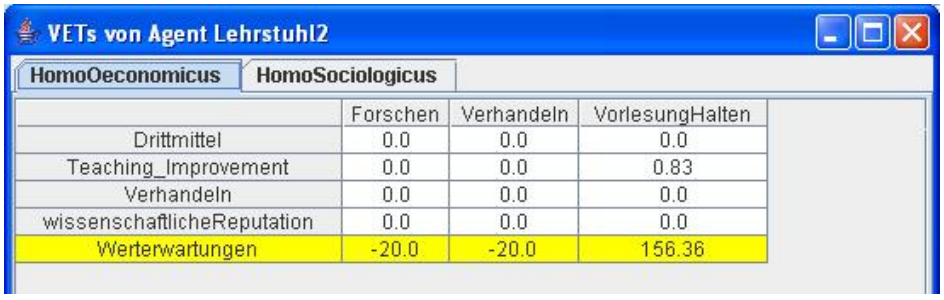

Abbildung B.18: Auswertung der VETs

B.4 Die XML-Dateien

B.4.1 Allgemeines

Alle bisher beschriebenen Daten können auch über XML-Dateien eingegeben werden. Dazu kann man für alle Einträge die links in der Baumstruktur zu finden sind eigene XML-Dateien erzeugen, die eine entsprechenden Endung besitzen. Will man z.B. eine Agenten Datei erzeugen mit einem beliebigen Agentennamen „X“, so muss man als erstes eine XML-Datei „X.AGENT“ erzeugen. Auf diese Weise kann man zunächst alle benötigten XML-Dateien erzeugen, um sie letztendlich mit Inhalt zu füllen. In den nächsten Kapiteln sollen folgende mögliche Endungen vorgestellt werden: .layout, .Action, .Agent, .Consequence, .CType, .Event, .Negotiation und .Resource. Bevor die einzelnen Dateitypen eingeführt werden soll jedoch der Aufbau einer solchen Datei erläutert werden. Die Dateien beginnen immer mit einem Rahmen, welcher wie Folgt aussehen sollte:
Bsp. Rahmen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>AbSIntH</comment>
...
..
.
<properties>
```

Dieser Rahmen sieht in allen Dateien gleich aus und kann auch aus diesem Beispiel kopiert werden, um eine neue Datei zu erzeugen.
Hat man diesen Rahmen in die Property-Datei kopiert, so kann man beginnen die eigentlichen Keys und Einträge einzufügen. Ein Ausdruck, welcher aus einem Key und einem Eintrag besteht, beginnt immer mit einer spitzen Klammer und endet auch immer mit einer spitzen Klammer. Im ersten Block steht der Name des Keys, zwischen den beiden Blöcken steht der Wert des Eintrages. Dabei kann der Eintrag entweder eine Zahl oder ein Array darstellen (bei den Array-Einträgen muss auf das Freizeichen nach dem Komma geachtet werden!).
Im Folgenden soll ein Beispiel die Syntax für einen Ausdruck veranschaulichen.
Bsp. mit einer Zahl als Eintrag:

```
<entry key="Beispielname">42</entry>
```

Bsp. mit einem Array als Eintrag:

```
<entry key="Beispielname">[Wert1, Wert2, ...]</entry>
```

Nachdem die Struktur der Property-Dateien erläutert wurde, sollen die möglichen Keys dargestellt werden. Dabei benötigen die einzelnen Dateien spezielle Keys mit speziellen Einträgen, die auf den nächsten Seiten erläutert werden. Weiterhin ist es wichtig, dass die Ausdrücke so aussehen, wie es in den Beispielen erläutert wird, da es sonst zu Problemem beim Auslesen der Dateien kommen kann.

B.4.2 Layout

Speicherdatei für das Layout-Fenster (siehe auch Kapitel B.2.2).

Key	**Eintrag**	**Erläuterung**
Path	./save/	Pfad des Save-Ordners
ConsequenceX	Name	Name der Folge, X von 0 bis ∞ aufeinanderfolgend
ResourceX	Name	Name der Ressource, X von 0 bis ∞ aufeinanderfolgend
AgentX	Name	Name des Agenten, X von 0 bis ∞ aufeinanderfolgend
EventX	Name	Name des Ereignisses, X von 0 bis ∞ aufeinanderfolgend
CharactertypeX	Name	Name des Charaktertypen, X von 0 bis ∞ aufeinanderfolgend
ActionX	Name	Name der Aktion, X von 0 bis ∞ aufeinanderfolgend
VetDisplay	ON|OFF	“ON“: Diplay-Ausgabe fuer die VET aktiviert, “OFF“: Ausgabe deaktiviert
Min_Value	x	x $\in$ N; minimaler X-Achsen Wert
Max_Value	x	x $\in$ N; maximaler X-Achsen Wert
NegotiationX	Name	Name der Verhandlung
Negotiations	[Name1, Name2,...]	Namen der Verhandlungen in richtiger Reihenfolge
NegotiationTime	[x1, x2,...]	Startpunkte der Verhandlungen, aufsteigend sortiert
terms	[Name1, Name2,...]	Namen der linguistischen Terme
Name_X	[x1, x2,...]	X-Werte der in Terms definierten Termnamen. x sind die x-Koordinaten der Funktion
Name_Y	[y1, y2,...]	Y-Werte der in Terms definierten Termnamen. y sind die y-Koordinaten der Funktion

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>AbSIntH</comment>
<entry key="NegotiationTime">[1]</entry>
<entry key="Term2_Y">[0.0, 0.51]</entry>
<entry key="Term2_X">[0.0, 1.0]</entry>
```

```
<entry key="Negotiations">[Verhandlung1]</entry>
<entry key="Consequence1">Folge2</entry>
<entry key="Consequence0">Folge1</entry>
<entry key="Agent1">Agent2</entry>
<entry key="Agent0">Agent1</entry>
<entry key="Charactertype1">Charakter2</entry>
<entry key="Charactertype0">Charakter1</entry>
<entry key="Resource1">Ressource2</entry>
<entry key="Resource0">Ressource1</entry>
<entry key="terms">[Term2, Term1]</entry>
<entry key="Action1">Aktion2</entry>
<entry key="Action0">Aktion1</entry>
<entry key="Negotiation1">Verhandlung2</entry>
<entry key="Negotiation0">Verhandlung1</entry>
<entry key="Term1_Y">[0.74, 0.0]</entry>
<entry key="Term1_X">[0.0, 1.0]</entry>
<entry key="Max_Value">1</entry>
<entry key="Path">./save/</entry>
<entry key="Min_Value">0</entry>
<entry key="Event1">Ereignis2</entry>
<entry key="Event0">Ereignis1</entry>
<entry key="VetDisplay">OFF</entry>
</properties>
```

B.4.3 Action

Speicherdatei für das Aktionen-Fenster (siehe auch Kapitel B.2.7).

Key	Eintrag	Erläuterung
ResourceX - Condition	x	Bedingung der Resource für das Eintreten des Ereignisses
Duration	x	Dauer einer Aktion
Charaktertypname - Benefit	x	Nutzen des Charaktertypen
Cycles	x	Anzahl der Wiederholungen
EventX	Name	Name des eintretenden Events, wenn alle Voraussetzungen erfüllt sind. X von 0 bis ∞ aufeinanderfolgend
Events	[Name1, Name2,...]	Namen der Events

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
```

```
<comment>AbSIntH</comment>
<entry key="Duration">21</entry>
<entry key="Actionname">Aktion1</entry>
<entry key="Cycles">12</entry>
<entry key="Ressource3 - Condition">97</entry>
<entry key="Ressource1 - Condition">5</entry>
<entry key="Charakter2 - Benefit">5</entry>
<entry key="Event1">Ereignis1</entry>
<entry key="Event0"></entry>
<entry key="Events">[, Ereignis1]</entry>
<entry key="Ressource2 - Condition">567</entry>
<entry key="Charakter1 - Benefit">6</entry>
</properties>
```

B.4.4 Agent

Speicherdatei für das Agent-Fenster (siehe auch Kapitel B.2.3).

Key	**Eintrag**	**Erläuterung**
Agentname	Name	Name des Agenten
CharaktertypnameX	x	Name Name und Gewichtung des Charaktertypen, $0 \leq x \leq 100$

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>AbSIntH</comment>
<entry key="Charakter1">81</entry>
<entry key="Agentname">Agent2</entry>
<entry key="Charakter2">0</entry>
</properties>
```

B.4.5 CType

Speicherdatei für das Charaktertypen-Fenster (siehe auch Kapitel B.2.4).

Key	Eintrag	Erläuterung
Charactertypename	Name	Name des Charaktertypen
ConsequenceX	Name	Name der Folge, $0 \leq x \leq \infty$ aufeinanderfolgend
Folge X Aktion	[ling.Var1 Ressource] == [Probability ling.Var2]	ling.Var1 ist eine in der Ressourcen-Datei definierte lingusitische Variable, ling.Var2 eine der in der Szenario-Datei definierten linguistischen Variablen für die Eintrittswahrscheinlichkeit

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>AbSIntH</comment>
<entry key="Charactertypename">Charakter1</entry>
<entry key="Aktion1 X Folge10">[Term2 Ressource1, term1 Ressource2]
      == [Probability Term1]</entry>
<entry key="Aktion2 X Folge20">[Term3 Ressource1, term1 Ressource3]
      == [Probability Term1]</entry>
</properties>
```

B.4.6 Event

Speicherdatei für das Ereignisse-Fenster (siehe auch Kapitel B.2.8).

Key	Eintrag	Erläuterung
Eventname	Name	Name des Events
RessourcennameX	Name	Name der Ressource für das Event
ResourceX	Ressourcenname	Name der Ressource, $0 \leq x \leq \infty$ aufeinanderfolgend
Ressourcenname	[ling.Var.1 Ress1, ling.Var.2 Ress2,...] == [[x1, x2, x3,...][y1, y2, y3,...]]	linguistische Variablen der Ressourcen mit jeweiligen Ressourcennamen. x als X-Werte der Koordinatenpunkte, von 0 bis maximaler X-Wertebereich. y als Y-Werte der Koordinatenpunkte, von 0 bis maximaler Y-Wertebereich

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>AbSIntH</comment>
<entry key="Eventname">Ereignis1</entry>
<entry key="Ressource20">[term1 Ressource2, term1 Ressource3]
    == [[-100.0, 100.0][0.0, 0.0]]</entry>
<entry key="Resource0">Ressource2</entry>
```

```
</properties>
```

B.4.7 Negotiation

Speicherdatei für das Verhandlung-Fenster (siehe auch Kapitel B.2.9).

Key	Eintrag	Erläuterung
MaxRounds	x	maximale Anzahl der Verhandlungsrunden
Agents	[Agentenname1, Agentenname2,...]	Name verhandelnden Agenten
Negotiationname	Name	Name Verhandlung
IterationNumber	x	Anzahl der Wiederholungen
IterationDuration	x	Dauer einer Wiederholung
IterationBegin	x	Startpunkt einer Wiederholung
YesVotes	x	Anzahl der Stimmen, um eine Verhandlung erfolgreich abzuschließen
ResourcesValue	[x1, x2,...]	Startwerte der zu verteilenden Ressourcen
NegotiatedResourceX	[RessourcennameX_Agentname1, RessourcennameX_Agentname2,...]	Verknüpfung der zu verhandelnden Ressourcen mit den jeweiligen Agenten
Resources	[Ressourcenname1, Ressourcenname2,...]	Name der zu verhandelnden Ressourcen
GiverResources	[Name1, Name2,...]	Name der Ressourcen, die auf den zu verhandelnden Ressourcen verteilt werden

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>AbSIntH</comment>
<entry key="MaxRounds">2</entry>
<entry key="Agents">[Agent1, Agent2]</entry>
<entry key="Negotiationname">Verhandlung1</entry>
<entry key="IterationNumber">2</entry>
<entry key="IterationDuration">234</entry>
<entry key="Resources">[Ressource3]</entry>
<entry key="IterationBegin">3</entry>
<entry key="YesVotes">342</entry>
<entry key="ResourcesValue">[324]</entry>
<entry key="NegotiatedResource0">[Ressource3_Agent1, Ressource3_Agent2]</entry>
<entry key="GiverResources">[none]</entry>
</properties>
```

B.4.8 Resource

Speicherdatei für das Ressourcen-Fenster (siehe auch Kapitel B.2.5).

Key	Eintrag	Erläuterung
ResourceType	0|1|2	0 = Masterressource, 1 = Persönliche-Ressource, 2 = System-Ressource
ObjectType	0|1|2	Wertebereiche der Ressource. 0 = Boolean, 1 = Integer, 2 = Double
Changing	[Agentenname1, Agentenname2, ...]	Name der Agenten, die die Ressource verändern dürfen
Viewing	[Agentenname1, Agentenname2, ...] | Owner	Name der Agenten, die die Ressource sehen können|Bei Masterressourcen kann man auch nur den Besitzer eintragen
Owner	Name	Name des Besitzers. Muss nur eingetragen werden, wenn unter ResourceType der Wert 1 eingegeben wurde.
Value	x	Startwert der Ressource mit nur einem Besitzer. Muss nur eingetragen werden, wenn unter ResourceType der Wert 1 eingegeben wurde.
Resourcename	Name	Name der Ressource
terms	[Name1, Name2, ...]	Namen der linguistischen Terme
Agent_Name	x	Startwert der Ressource für den Agenten „Name“. Der Eintrag wird nur bei ResourceType != 0 benötigt
Min_Value	x	$x \in \mathbb{N}$; minimaler X-Achsen Wert
Max_Value	x	$x \in \mathbb{N}$; maximaler X-Achsen Wert
NegotiationX	Name	Name der Verhandlung, X von 0 bis ∞ aufeinanderfolgend
Negotiations	[Name1, Name2,...]	Namen der Verhandlungen, nach Startpunkte aufsteigend sortiert
NegotiationTime	[x1, x2,...]	Startpunkte der Verhandlungen, aufsteigend sortiert
terms	[Name1, Name2,...]	Namen der linguistischen Terme
Name_X	[x1, x2,...]	X-Werte der in Terms definierten Termnamen. x sind die x-Koordinaten der Funktion
Name_Y	[y1, y2,...]	Y-Werte der in Terms definierten Termnamen. y sind die y-Koordinaten der Funktion

Beispiel für Masterressourcen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>AbSIntH</comment>
<entry key="term1_Y">[0.0, 0.7]</entry>
<entry key="term1_X">[0.0, 100.0]</entry>
<entry key="Agent_Agent2">11</entry>
<entry key="terms">[term2, term1]</entry>
<entry key="Agent_Agent1">213</entry>
<entry key="term2_Y">[0.62, 0.0]</entry>
<entry key="term2_X">[0.0, 100.0]</entry>
<entry key="Max_Value">100</entry>
<entry key="Changing">[Agent1, Agent2]</entry>
<entry key="ResourceType">0</entry>
```

```
<entry key="Resourcename">Ressource3</entry>
<entry key="Min_Value">0</entry>
<entry key="ObjectType">2</entry>
<entry key="Viewing">Owner</entry>
</properties>
```

Beispiel für Persönliche-Ressourcen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>AbSIntH</comment>
<entry key="ResourceType">1</entry>
<entry key="Term2_Y">[0.0, 0.69]</entry>
<entry key="Term2_X">[0.0, 100.0]</entry>
<entry key="Changing">[Agent1, Agent2]</entry>
<entry key="Viewing">[Agent1, Agent2]</entry>
<entry key="Owner">Agent1</entry>
<entry key="terms">[Term1, Term2, Term3]</entry>
<entry key="Agent_Agent2"></entry>
<entry key="Agent_Agent1"></entry>
<entry key="Term1_Y">[0.8, 0.0]</entry>
<entry key="Term1_X">[0.0, 100.0]</entry>
<entry key="Resourcename">Ressource1</entry>
<entry key="Term3_Y">[0.29, 0.0]</entry>
<entry key="Term3_X">[0.0, 100.0]</entry>
<entry key="Max_Value">100</entry>
<entry key="Min_Value">0</entry>
<entry key="Value">213</entry>
<entry key="ObjectType">0</entry>
</properties>
```

Beispiel für Systemressourcen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>AbSIntH</comment>
<entry key="term1_Y">[0.0, 1.0]</entry>
<entry key="term1_X">[0.0, 100.0]</entry>
```

```
<entry key="Agent_Agent2"></entry>
<entry key="terms">[term1, term2, term3]</entry>
<entry key="Agent_Agent1"></entry>
<entry key="term2_Y">[0.44, 0.0]</entry>
<entry key="term2_X">[0.0, 100.0]</entry>
<entry key="Value">231</entry>
<entry key="Max_Value">100</entry>
<entry key="Changing">[Agent1, Agent2]</entry>
<entry key="ResourceType">2</entry>
<entry key="term3_Y">[0.92, 0.0]</entry>
<entry key="Resourcename">Ressource2</entry>
<entry key="term3_X">[0.0, 100.0]</entry>
<entry key="Min_Value">0</entry>
<entry key="ObjectType">1</entry>
<entry key="Viewing">[Agent1, Agent2]</entry>
</properties>
```

Literaturverzeichnis

[BA99] K.Beck, D.Andres: Extreme Programming Explaine, Addison-Wesley, 1999

[LRW05] Martin Lippert, Stefan Roock, Henning Wolf: Software entwickeln mit eXtreme Programming, DPunkt, 2005

[URL01] http://www.extremeprogramming.org/start.htm

[ROBO] http://www.alphaworks.ibm.com/tech/robocode

[WISOC] Wikipedia - die freie Enzyklopädie, Artikel „Homo Sociologicus“, Stand 30.04.2005, URL: http://de.wikipedia.org/wiki/Homo_sociologicus

[FuzJ] FuzzyJ Toolkit, http://ai.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit.html

[JASSS] JASSS - The Journal Of Artificial Societies and Social Simulation, http://jasss.soc.surrey.ac.uk

[CIM] Critical Incident Management: An Empirically Derived Computational Model, http://www.soc.surrey.ac.uk/JASSS/1/4/1.html

[GEO] The GeoGraph 3D Computational Laboratory: Network and Terrain Landscapes for RePast, http://jasss.soc.surrey.ac.uk/7/1/7.html

[InNet] Innovation Networks - A Simulation Approach, http://jasss.soc.surrey.ac.uk/4/3/8.html

[Reci] Reciprocity, Normative Reputation, and the Development of Mutual Obligation in Gift-Giving Societies, http://jasss.soc.surrey.ac.uk/7/1/5.html

[Group] Group Reputation Supports Beneficent Norms, http://jasss.soc.surrey.ac.uk/5/4/4.html

[VIRPOX] VIR-POX: An Agent-Based Analysis of Smallpox Preparedness and Response Policy, http://jasss.soc.surrey.ac.uk/7/3/6.html

[Wild] An Individual-Based Modeling Approach to Simulating Recreation Use in Wilderness Settings, http://www.wilderness.net/library/documents/Gimblett_4-11.pdf

[DvsC] Design Versus Cognition: The interaction of agent cognition and organizational design on organizational performance, http://jasss.soc.surrey.ac.uk/1/3/4.html

[Emo] Introducing Emotions into the Computational Study of Social Norms: A First Evaluation, http://jasss.soc.surrey.ac.uk/4/1/2.html

[NJSAH] N.Saam, A. Harrer: Simulating Norms, Social Inequality, an Functional Change in Artificial Societies, 1999, Journal of Artifivial Societies and Social Simulation vol.2, no.1, http://jasss.soc.surrey.ac.uk/2/1/2.html

[LAB04] M. A. Labrie et. al., DIAGAL: A Tool for Analyzing and Modelling Commitment-Based Dialogues between Agents IN: Advances in Artificial Intelligence, Heidelberg 2002.