Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Gregor Balthasar

Agentenbasierte Schwarmsteuerung in einer kompetitiven, dynamischen Umgebung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Gregor Balthasar

Agentenbasierte Schwarmsteuerung in einer kompetitiven, dynamischen Umgebung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Wolfgang Renz
Zweitgutachter : Prof. Dr. Michael Neitzke

Abgegeben am 28. August 2008

Gregor Balthasar

Thema der Bachelorarbeit

Agentenbasierte Schwarmsteuerung in einer kompetitiven, dynamischen Umgebung

Stichworte

Multiagentensystem, Schwarmsteuerung, Koordination, Planung, Agentenorientierte Softwareentwicklung (AOSE)

Kurzzusammenfassung

Diese Arbeit beschreibt die Entwicklung eines Prototypen, der am Agent Contest 2008 teilgenommen hat. In diesem Rahmen wird ein Multiagentensystem entworfen und entwickelt, dass es schafft, erfolgreich im vorgegebenen Szenario des Contests zu agieren. In diesem Szenario gilt es, mit einem koordinierten und kooperierenden Team von Agenten, Schwärme von Kühen in einer möglicherweise „feindlichen“ Umgebung in das eigene Gatter zu steuern. Die Realisierung des Multiagentensystems erfolgt mittels einer Multiagentenplattform.

Gregor Balthasar

Title of the paper

Agentbased Swarmcontrol in a competitive, dynamic Environment

Keywords

Multi-Agent System, Swarmcontrol, Coordination, Planning, Agentoriented Softwareengineering (AOSE)

Abstract

This paper describes the development of a prototype, which has participated in the Agent Contest 2008. In this process a multi-agent system is designed and implemented, which is able to operate successfully in the specified scenario of the contest. A coordinated and cooperative set of agents has to steer swarms of cows through a possibly hostile environment to cope with this scenario. The conceived design is implemented in a multi-agent platform.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einführung

Multiagentensysteme bzw. die Agentenorientierung sind ein noch junger Bereich der Informatik. Es gibt mittlerweile eine Vielzahl von Ansätzen der *Agentenorientierten Softwareentwicklung* (AOSE) und ebenso völlig verschiedene Plattformen, die zu deren Realisierung genutzt werden. So verwundert es kaum, dass jetzt auch in diesem Teilbereich vermehrt Wettkämpfe stattfinden, um möglicherweise die Effizienz der verschiedenen Ansätze/Plattformen bewerten zu können, und um den Wissensaustausch anzuregen.

In diesem Fall ist es der Agent Contest 2008[1], der dieser Bachelorarbeit zugrunde liegt. Ziel ist es, einen voll funktionsfähigen Prototypen zu entwickeln, der erfolgreich am Agent Contest 2008 teilnehmen kann und somit auch die durch den Titel dieser Arbeit aufgestellte Hauptproblematik löst, die „Agentenbasierte Schwarmsteuerung in einer dynamischen, kompetitiven Umgebung“. Das Szenario des Agent Contest 2008 unterliegt genau dieser Problematik (siehe auch Kapitel 2.3 „Agent Contest 2008“ ab Seite 15).

1.1 Problemstellung & Motivation

Der Bereich der Multiagentensysteme unterliegt aufgrund seines geringen Alters noch vielen nicht gänzlich geklärten Problemstellungen. Das oben genannte Entwicklungskonzept der AOSE ist eine dieser Problematiken. Dieses in letzter Zeit immer stärker aufkommende Paradigma der Softwareentwicklung hält eine Vielzahl semi-standardisierter Entwicklungsmethodiken vor, die allesamt einer ständigen Weiterentwicklung auf der Suche nach einem allgemeingültigen Standard unterstehen. So wird zur Entwicklung des Prototypen im Laufe dieser Arbeit eine dieser Methodiken benutzt, um eine Überprüfung und Bewertung des Entwicklungskonzepts vorzunehmen.

Auch gibt es mittlerweile viele verschiedene Multiagentensprachen/-plattformen. Hier soll im Verlauf der Arbeit eine Multiagentenplattform ausgewählt werden, um die Lösbarkeit der Aufgabenstellung mit dieser Plattform aufzuzeigen.

[1] http://cig.in.tu-clausthal.de/agentcontest2008/?content=scenario

Weiterhin bieten natürlich Agentenszenarien wie das des Agent Contest 2008 eine gute Möglichkeit des Tests für das Forschungsfeld der *verteilten Künstlichen Intelligenz*, zu dem Multiagentensysteme gehören. Dafür hält das Szenario des Contests eine dynamische, nichtdeterministische Umgebung vor, in der der zu entwickelnde Prototyp erfolgreich die Aufgabenstellung absolvieren soll.

1.2 Aufbau der Arbeit

Aufgebaut wird diese Bachelorarbeit chronologisch entlang der Vorgehensweise. Dementsprechend werden im folgenden Kapitel 2 ab Seite 11 zuerst die grundlegenden Technologien besprochen, mit einer anschließenden Klärung des Contest-Szenarios und den sich daraus ergebenen grundlegenden Begrifflichkeiten. Am Ende des Kapitels werden in einem Fazit die weiteren Herausforderungen vorgestellt. Darauf aufbauend kann in Kapitel 3 ab Seite 21 auf die Anforderungen eingegangen werden, die der Prototyp erfüllen soll, um erfolgreich am Contest teilnehmen zu können. Weiterhin wird in Kapitel 4 ab Seite 27 eine technische Analyse vorgenommen, um eventuell bereits vorhandene Technologien zur Lösung der in den Anforderungen herausgearbeiteten Problemstellungen zu identifizieren, zu bewerten und auszuwählen. Im darauf folgenden Kapitel 5 ab Seite 36 werden dann die finale Architektur und das Design vorgestellt, welche sich aus der inkrementellen Entwicklung von Prototypen ergeben haben. Ausgewählte Aspekte der Realisierung des Prototypen finden sich daraufhin in Kapitel 6 ab Seite 49. Folgend wird in Kapitel 7 ab Seite 55 zuerst aufgezeigt, auf welche Art und Weise der Prototyp während der Entwicklung Tests unterzogen wurde[2] und dann wird noch eine Bewertung der Funktionsweise des Prototypen im Contest selbst vorgenommen, sowohl bezüglich der Anforderungen als auch der Implementationen anderer teilnehmender Teams. Schließlich wird in Kapitel 8 ab Seite 65 sowohl ein zusammenfassendes Fazit geliefert, als auch ein Ausblick darauf, wie sich der Prototyp in Zukunft noch effektiv verbessern ließe.

[2] Tests in einem Multiagentensystem mit Agenten, die in einer dynamischen Umgebung agieren, unterliegen völlig anderen Anforderungen als solche in herkömmlichen Softwaresystemen.

2 Grundlagen

In diesem Kapitel werden die für das Verständnis und die Umsetzung dieses Themas notwendigen Grundlagen aufbereitet.

Dazu wird zuerst ein Blick auf die allgemeine Thematik von Multiagentensystemen und auf den Stand der Technik geworfen. Daraufhin wird der Agent Contest 2008, der dieser Arbeit zugrundeliegt, thematisiert und auf das Szenario des Contests eingegangen. Folgend werden die aus dem Szenario abgeleiteten Begrifflichkeiten für diese Arbeit festgelegt. Außerdem wird noch auf das Thema „Schwärme“ eingegangen und den damit zusammenhängenden Besonderheiten in dieser Arbeit.

Zuletzt werden in einem Fazit die Schlüsselaspekte, die die Grundlage dieser Arbeit bilden, zusammengefasst.

2.1 Multiagentensysteme

Multiagentensysteme (MAS) sind Systeme, die aus mehreren homogenen oder heterogenen, autonom agierenden und kommunizierenden, kleineren Systemen, bekannt als *Agenten*, bestehen [Wooldridge (2002)]. MAS fallen in den Bereich der Verteilten Künstlichen Intelligenz und beschäftigen sich damit, komplexe Probleme durch Koordination und Kooperation von Agenten zu lösen. Dabei spielt die Abstimmung des spezifischen Wissens, sowie der Ziele und Pläne einzelner Agenten eine große Rolle.

Für ein tieferen Einblick in die Thematik ist oben zitiertes Buch von Wooldridge zu empfehlen, im Literaturverzeichnis finden sich auch noch weitere, nicht zitierte aber dennoch empfehlenswerte Werke [Ferber (2001); Weiss (2000)].

Es bestehen verschiedene Konzepte zur Realisierung von Agenten, deren *weiche* Anforderungen [Wooldridge und Jennings (1995)] folgende sind:

- Autonomität: Agenten operieren ohne direktes Eingreifen durch Menschen.
- Soziale Fähigkeit: Agenten kommunizieren mit anderen Agenten über eine *Agent Communication Language* (ACL).

- Reaktivität: Agenten nehmen ihre Umgebung wahr und reagieren auf Veränderungen in dieser.
- Pro-Aktivität: Agenten reagieren nicht nur, sondern können ein zielgerichtetes Verhalten entwickeln, und können somit *die Initiative ergreifen*.

2.1.1 BDI-Agenten

Ein Ansatz, der sich immer stärkerer werdender Beliebtheit erfreut, ist die sogenannte BDI-Architektur (*belief, desire, intention*). Der grundlegende Gedanke dieses Modells ist philosophischer Natur, dahingehend, dass Menschen auch nur ein theoretisches Modell der Welt haben und anhand dessen ihr *Reasoning*[1] betreiben.

Beliefs repräsentieren dabei das „Wissen“ des Agenten über die ihn umgebende Welt. *Desires* oder auch *Goals* stehen für die Motivationen eines Agenten und sind meist hierarchisch angeordnet. *Intentions* spiegeln den deliberativen Status eines Agenten wieder, sprich sie zeigen an wofür sich ein Agent entschieden hat. Schließlich gibt es noch *Plans*, diese bestehen aus Abfolgen von Aktionen und ermöglichen dem Agenten bestimmte *Intentions* zu erfüllen.

2.1.2 Koordination in MAS

Koordination ist das zentrale Problem kooperativer Arbeit, das auftritt, sobald die Aktionen mehrerer Agenten in irgendeiner Form sich überschneiden oder Einfluss aufeinander nehmen können.

Zur Lösung dieses Problems bestehen verschiedene Ansätze [Wooldridge (2002)]:

- Koordination durch partielles globales Planen.
- Koordination durch verbundene *Intentions*.
- Koordination durch wechselseitige Modellierung.
- Koordination durch Normen und soziale Gesetze.

Eine genaue Beschreibung dieser Ansätze findet sich im zitierten Buch von Wooldridge.

[1] Reasoning = Entscheidungsfindung

2.2 MAS: State of the Art

Aktuelle Multiagentensysteme werden meist unter Nutzung von Multiagentenplattformen entwickelt. Diese Plattformen sind eine Form der Middleware und bieten grundlegende Dienste zur Realisierung eines Multiagentensystems. Die *Foundation for Intelligent Physical Agents* (FIPA)[2] hat dabei einen Standard für die generelle Architektur einer Multiagentenplattform formuliert [FIPA b], Abbildung 2.1 auf Seite 14 zeigt ein zugehöriges Architekturmodell. Hauptanforderungen dieser Spezifikationen an eine Plattform ist, dass sie Möglichkeiten zur Erstellung, Lokalisierung, Kommunikation und Zerstörung von Agenten bietet.

Es existieren verschiedene Multiagentenplattformen, die die FIPA Standards nutzen, z.B. JACK,[3] JADE,[4] Jadex.[5] Nähere Informationen zu diesen und noch weiteren Plattformen lassen sich auch auf der Homepage der FIPA finden. Auffällig ist, dass ein Großteil dieser Plattformen auf Java™aufsetzt.

Weiterhin bestehen verschiedene Methodiken zur Entwicklung von Multiagentensystemen, auf die in Kapitel 5 „Design“ näher eingegangen wird. Den „einen“ festen Standard gibt es allerdings nicht.

[2] zu finden unter: http://www.fipa.org
[3] zu finden unter: http://www.agent-software.com
[4] zu finden unter: http://jade.tilab.com
[5] zu finden unter: http://vsis-www.informatik.uni-hamburg.de/projects/jadex

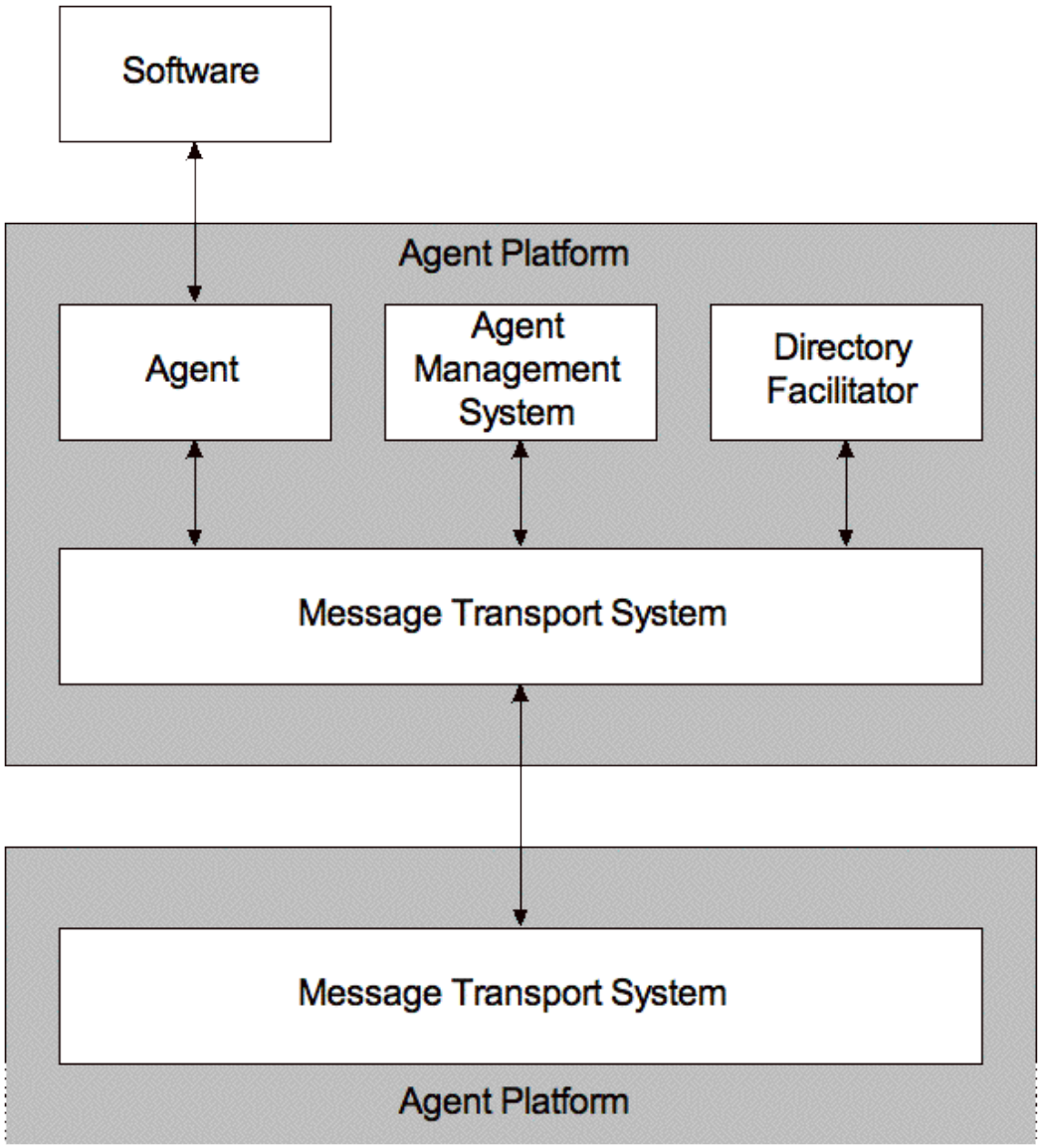

Abbildung 2.1: Agent Management Reference Model [FIPA b]

2.3 Agent Contest 2008

Der Agent Contest 2008 ist ein von einem Team rund um Prof. Dr. Jürgen Dix[6] ausgerichteter Wettbewerb. Ausgeschriebenes Ziel des Wettbewerbs ist es, die Forschung im Bereich der Multiagentensystementwicklung und -programmierung anzuregen, durch Identifizierung von Schlüsselproblemen und der Sammlung passender Benchmarks.

Der Wettbewerb wird das vierte Jahr in Folge durchgeführt, wobei er seit 2007 im Rahmen des *Programming Multiagent Systems Languages and tools* (ProMAS) Workshops[7] stattfindet, die zwei vorherigen Wettbewerbe fanden im Rahmen des *Computational Logic in Multi-Agent Systems* (CLIMA) Workshops[8] statt.

2.3.1 Szenario

Die genaue Beschreibung des Szenarios ist auf der Homepage des Agent Contest 2008 zu finden, an dieser Stelle wird eine Kurzbeschreibung geliefert, die die wesentlichen Aspekte des Szenarios herausarbeitet. Alle im Folgenden benutzten Begriffe werden im nächsten Kapitel „Abgeleitete Begrifflichkeiten“ dann noch ausführlicher erklärt.

Zur Kurzbeschreibung des Szenarios:

Jeweils sechs Agenten zweier Teams sollen auf einem gitterähnlichen Spielfeld vorher nicht bekannter, variabler Größe versuchen, die sich auf diesem Spielfeld befindlichen, zur Schwarmbildung fähigen Kühe in eine als Gatter markierte Zone des Spielfelds zu dirigieren. Dabei ist jeder Agent direkt mit dem Contestserver verbunden und hat nur die eigene Bewegung als Aktionsmöglichkeit. Somit können Agenten nur indirekt auf die Bewegung von Kühen/Schwärmen Einfluss zu nehmen, indem sie sich so neben eine Kuh positionieren, dass diese von ihnen wegzieht. Es gibt auf einem Spielfeld zwei der genannten Gatter, wobei jedem der zwei Teams ein Gatter zugewiesen wird. Beim Betreten eines Gatters durch eine Kuh, verschwindet diese und wird dem Team, dem das Gatter zugeordnet ist, als Punkt angerechnet. Weiterhin sind auf dem Spielfeld Hindernisse verteilt, die die Navigation und Steuerung der Schwärme zu erschweren. Das Team, das am Ende einer Simulation die meisten Punkte gesammelt hat, gewinnt diese Simulation und erhält drei Punkte im laufenden Wettbewerb, während das Team, das verloren hat, keine Punkte erhält. Bei einem Unentschieden erhalten beide Teams jeweils einen Punkt.

[6] Professor an der Universität Clausthal, Lehrstuhl für „Computational Intelligence“
[7] zu finden unter: http://www.cs.uu.nl/ProMAS/
[8] zu finden unter: http://research.nii.ac.jp/climaVII/

2.4 Abgeleitete Begrifflichkeiten

An dieser Stelle werden kurz die aus dem Szenario abgeleiteten Begrifflichkeiten erklärt, die in dieser Arbeit immer wieder auftauchen, um Missverständnisse bzw. fortlaufend nötige Erklärungen zu vermeiden.

2.4.1 Agent

Wird im weiteren Verlauf der Arbeit der Begriff „Agent“ genutzt, ist damit einer der sich direkt auf dem Spielfeld befindenden Agenten gemeint, und nicht die klassische Definition eines Agenten in einem Multiagentensystem. Ob es sich um einen der eigenen oder der gegnerischen Agenten handelt oder um einen möglicherweise zusätzlich zu den sechs „agierenden“ Agenten im System befindlichen Agenten, wird dann explizit erwähnt.

Ein Agent hat als Aktionsmöglichkeiten pro Runde der Simulation nur die Bewegung in die acht Richtungen, unter „Spielfeld“ genauer erklärt, sowie die Aktion „skip“, die zur Folge hat, dass der Agent für diese Runde keine Aktion ausführt. Sendet der Agent keine Aktion innerhalb des serverspezifischen Timeouts an den Server, wird der Zug des Agenten ebenfalls als „skip“ gewertet.

2.4.2 Kuh

Eine Kuh ist ein serverseitig gesteuertes, bewegliches Objekt, das sich nach einem vorgegebenen Algorithmus auf dem Spielfeld bewegt. Hier eine Kurzbeschreibung des Algorithmus:[9]

- Kühe fühlen sich von anderen Kühen angezogen, das heißt sie bewegen sich aufeinander zu. (Schwarmbildung)
- Kühe fürchten sich vor Hindernissen, das heißt sie bewegen sich von solchen nach Möglichkeit weg.
- Kühe fürchten sich vor Agenten, der Faktor ist höher als bei Hindernissen, allerdings spielt sowohl bei Hindernissen als auch Agenten die Entfernung keine Rolle, die „Furcht“ ist immer gleich stark ausgeprägt.

[9] eine genaue Beschreibung des Algorithmus findet sich unter: http://cig.in.tu-clausthal.de/agentcontest2008/scenario-r4.pdf

Sowohl Anziehungskraft als auch Furcht gelten nur für Objekte, die sich im Sichtradius einer Kuh befinden. Dieser Sichtradius wird forthin Vision genannt und weiter unten im Kapitel genauer erklärt.

Weiterhin bekommt jede Kuh eine serverseitige geheime ID zugewiesen und am Anfang jeder Runde wird folgender Vergleich ausgeführt:

$$(a \bmod m) = (b \bmod m)$$

mit $a = geheimeID$, $b = momentaneRunde$ und $m = 3$.
Ist der Vergleich wahr, darf die betreffende Kuh in der momentanen Runde ziehen, andernfalls nicht. Das führt dazu, dass jede Kuh nur jede dritte Runde ziehen kann. Zusätzlich zu der internen ID haben Kühe noch eine externe ID, die zur Identifizierung der Kühe durch die Agenten dient und in keinem Zusammenhang mit der internen ID steht.

Die Anzahl der Kühe ist für jede Simulation festgelegt, das heißt es tauchen während einer Simulation keine weiteren Kühe auf.

Schließlich gilt den Kühen das Hauptinteresse, da mit diesen in Verbindung mit den Gatterfeldern die Punkte erzielt werden, die über Sieg oder Niederlage bei einer Simulation entscheiden. Das richtige Ausnutzen des Algorithmus der Kühe ist also nötig, um diese zu „steuern“.

2.4.3 Hindernis

Hindernisse sind serverseitig gesetzte, sich auf dem Spielfeld befindliche, unbewegliche Objekte. Alle anderen beweglichen Objekte können weder auf Zellen, die mit Hindernissen belegt sind, ziehen, noch in irgendeiner Form mit diesen interagieren. Es ist nicht festgelegt, ob diese Hindernisse zu Simulationsbeginn zufällig auf dem Spielfeld verteilt werden oder im Vorfeld Spielfelder mit bestimmten Formationen von Hindernissen erstellt werden. Eventuell ist also mit regelrechten Labyrinthen zu rechnen.

Hindernisse haben keine spezielle ID, sind aber durch ihre Position eindeutig zu bestimmen.

2.4.4 Gatter

Zu Beginn jeder Simulation werden serverseitig zwei rechteckige, aus mehreren Zellen bestehende Flächen auf dem Spielfeld definiert, das heißt jede zu dieser Fläche zugehörige Zelle erhält die Eigenschaft „Gatter“. Jede dieser Flächen wird jeweils einem der zwei an

der Simulation teilnehmenden Teams zugewiesen. Zu Beginn einer Simulation erhält jedes Team die Position des eigenen Gatters, nicht aber die des gegnerischen.[10]

Betritt eine Kuh eine Gatterzelle, verschwindet diese und es wird dem Team ein Punkt zugewiesen, dem diese Gatterzelle zugeordnet ist.

Wird fortan also von einem Gatter gesprochen, ist damit das gesamte Feld von Gatterzellen gemeint.

2.4.5 Spielfeld

Das Spielfeld besteht aus einem mit Koordinaten versehenem Gitter, dessen Größe von Simulation zu Simulation unterschiedlich sein kann. [11] Auf einer Zelle des Gitters kann sich zur gleichen Zeit immer nur eines der folgenden Objekte befinden:

- Agent (eigener oder gegnerischer)
- Kuh
- Hindernis

Zusätzlich zu einem dieser Objekte kann eine Zelle allerdings noch die Eigenschaft „Gatter“ innehaben.

Bewegungszüge der Agenten und auch Kühe sind auf dem Spielfeld in acht Richtungen möglich, im weiteren betitelt mit „nord“, „nordost“, „ost“, „südost“, „süd“, „südwest“, „west“ und „nordwest“. Ein Zug geht immer über eine Zelle pro Runde in die gewünschte Richtung, kann aber nur erfolgreich sein, wenn diese Zelle nicht belegt ist. Versuchen zwei bewegliche Objekte gleichzeitig dieselbe Zelle zu belegen, wird serverseitig ausgewürfelt, welches davon Erfolg hat. Das erfolglose Objekt bewegt sich dann in dieser Runde nicht.

2.4.6 Vision

Als Vision wird fortan der Sichtbereich der Agenten und der Kühe bezeichnet.

Der Sichtbereich der Agenten ist definiert als ein Quadrat bestehend aus 289 Zellen mit dem Agenten als Mittelpunkt.[12] Das heißt ein Agent bekommt am Anfang jeder neuen Runde einer Simulation Informationen über den Zustand aller in seiner Vision befindlichen Zellen. Diese Informationen beinhalten die Positionen und IDs von Kühen, die Positionen eigener

[10] Die Position des gegnerischen Gatters kann allerdings erkundet werden, beschrieben unter „Vision“.
[11] Die tatsächliche Größe des Gitters wird den Agenten per XML-Paket am Anfang der Simulation mitgeteilt.
[12] entspricht einem Radius von 8 Zellen.

und gegnerischer Agenten, die Positionen von Hindernissen sowie die Information, ob eine Zelle die Eigenschaft „Gatter“ innehat, wobei hier in eigene und gegnerische Gatter unterteilt wird.

Der Sichtbereich der Kühe ist definiert als ein Quadrat bestehend aus 81 Zellen mit der Kuh als Mittelpunkt.[13] Objekte, die Einfluss auf den Bewegungsalgorithmus einer Kuh haben, spielen nur eine Rolle für diesen, wenn sie sich in der Vision der Kuh befinden.

2.5 Schwärme & koordinierte Schwarmsteuerung

In der *verteilten Künstlichen Intelligenz* werden Schwärme und Schwarmintelligenz zur Lösung spezifischer Probleme betrachtet, z.B. zum Rerouting von Netzwerk Traffic, wie von Bonabeau und Theraulaz in einem Artikel beschrieben [Bonabeau und Theraulaz (2000)].

An dieser Stelle muss eine Abgrenzung erfolgen. Ziel dieser Arbeit ist es nicht ein Multiagentensystem zur Simulation eines oder mehrerer Schwärme zu erstellen, sondern sich auf dem Spielfeld befindliche Schwärme mit Agenten einzig über deren Positionierung hinsichtlich eines Schwarms gezielt in bestimmte Richtungen zu steuern.

Dazu ist es nötig, dass die Agenten kooperativ und koordiniert vorgehen und im Zuge dessen Planung betreiben, um die Zielsetzung zu erfüllen.

[13] entspricht einem Radius von 4 Zellen.

2.6 Fazit

Zusammenfassend werden hier die Herausforderungen aufgezählt, die sich bei der Entwicklung des prototypischen Multiagentensystems stellen:

- Koordination, Kooperation und Planung zur Realisierung der Schwarmsteuerung
- Umgang mit einer dynamischen, nicht-deterministischen Umgebung
- verteilte Künstliche Intelligenz in einem Multiagentensystem
- Turnierbedingungen:
  - gegnerische Agenten befinden sich ebenfalls auf dem Spielfeld
  - Simulationsläufe über mehrere Tage hinweg (Stabilität des Systems)
- Kommunikation mit dem Contestserver über das Internet

Somit können nun im folgenden Kapitel 3 die spezifischen Anforderungen an das Multiagentensystem aufgestellt werden.

3 Anforderungen

Die Anforderungen für den Prototypen ergeben sich in diesem Fall direkt aus den Teilnahmebedingungen und der Szenariobeschreibung des Contests. Hier erfolgt dann noch eine Aufteilung in fachliche und technische Anforderungen.

3.1 Fachliche Anforderungen

Die fachlichen Anforderungen sehen anders aus, als die herkömmlicher Softwarelösungen, da es sich um ein Multiagentensystem handelt, in dem die beteiligten Agenten autonom agieren. Folglich wird es keine Use-Cases geben, da ein Benutzer keinen direkten Einfluss auf das System hat. Somit wird hier als fachliche Anforderung gewertet, was das System leisten muss, um dem Szenario gerecht erfolgreich arbeiten zu können.

3.1.1 Navigation (Pathfinding) der Agenten auf dem Spielfeld

Die Spielfelder sind nicht von vorneherein bekannt und können sich von Simulation zu Simulation dauernd verändern und auch die Anzahl der Runden pro Simulation ist begrenzt. Somit ist es nötig, dass die Agenten eine Möglichkeit zur effizienten Navigation erhalten, um auch über längere Strecken hinweg zielgerichtet navigieren zu können, was die Anzahl der benötigten Runden pro zurückgelegtem Weg minimiert.

Klassifizierung: must

3.1.2 Erkundung des Spielfelds

Da nicht davon auszugehen ist, dass zu Beginn einer Simulation die Agenten so auf dem Spielfeld verteilt sind, dass alle Kühe in ihrer Vision sind, und das Geschehen auf dem Spielfeld dynamisch vonstatten geht, sprich die Kühe sich bewegen und auch die Agenten des

Gegners eine Rolle spielen, müssen die Agenten das Spielfeld immer wieder erkunden. Weiterhin sollen die Hindernisse in die Wissensbasis aufgenommen werden, um die Navigation der Agenten zu erleichtern.

Klassifizierung: must

3.1.3 Steuerung von einzelnen Kühen bzw. Schwärmen

Dies ist der essentiellste Punkt, die Agenten müssen die Möglichkeit haben, einzelne Kühe und auch Schwärme über das Spielfeld hinweg an Hindernissen vorbei in das eigene Gatter zu steuern. Dazu kann auch eine Koordination der Agenten nötig sein, um die Steuerung großer Schwärme durch mehrere Agenten zu ermöglichen.

Klassifizierung: must

3.1.4 Stören des Gegners

Bei jeder Simulation befinden sich zusätzlich zu den eigenen sechs Agenten noch weitere sechs Agenten eines gegnerischen Teams auf dem Spielfeld. Die eigenen Agenten sollten gegebenenfalls Strategien parat haben, um die Steuerung von Kühen und Schwärmen durch die gegnerischen Agenten zu stören bzw. völlig zu unterbinden. Dies könnte einhergehen mit einer „Tit-for-tat“-Strategie, so dass die eigenen Agenten erst dann anfangen destruktiv gegen den Gegner vorzugehen, wenn dem ähnliche Aktionen durch den Gegner vorhergingen.

Klassifizierung: nice to have

3.1.5 Spieltheoretischer Ansatz

Bekanntermaßen besteht eine Simulation aus einer vom Contestserver vorgegebenen, den Agenten bekannten, Anzahl von Runden. Daraus folgt, dass, nähert sich die Simulation ihrem Ende, gewisse Entscheidungen der Agenten anders getroffen werden sollten, als z.B. zur Mitte des Spiels. Ein Beispiel dafür wäre folgende Situation: Angenommen es sind zwei Schwärme im Sichtbereich der Agenten, der eine bestehend aus fünf Kühen mit 15 Feldern Entfernung zum Gatter, der andere bestehend aus zehn Kühen mit 25 Feldern Entfernung zum Gatter. Jetzt sollte dementsprechend, ob es überhaupt noch möglich ist den grösseren Schwarm aufgrund der höheren Entfernung unter Hinzunahme der noch verbleibenden Runden ins eigene Gatter zu steuern, entschieden werden, welcher Schwarm zuerst gesteuert

wird und nicht wie eventuell sonst anhand einer anderen Bewertungsfunktion, z.B. der Größe des Schwarms.

Klassifizierung: nice to have

3.2 Technische Anforderungen

Da nun die fachlichen Anforderungen aufgestellt sind, ist es jetzt das Ziel eine geeignete Technologie zu suchen, mit der sich diese umsetzen lassen. Die Teilnahmebedingungen des Contests implizieren einen Teil der technischen Anforderungen.[1] Zusätzlich werden weitere technische bzw. nicht-funktionale Anforderungen aus dem Szenario sowie dem vorgesehenen Ablauf des Contests abgeleitet.

3.2.1 Multiagentensprache bzw. -plattform

Laut Teilnahmebedingung soll die Realisierung mittels einer Multiagentensprache oder -plattform umgesetzt werden. Die aufgestellten fachlichen Anforderungen gehen prinzipiell über nichts hinaus, was vorhandene Multiagentensprache bzw. -plattformen mitbringen, womit der Hauptaspekt für die Wahl der Sprache/Plattform auf möglichen bereits gemachten Erfahrungen mit einem speziellen System liegt, bzw. auf einer kurzen Einarbeitungszeit. Freie Verfügbarkeit ist ein weiterer Punkt. Mögliche Kandidaten wären z.B. JADE,[2] Jadex[3] oder JIAC-IV/TNG.[4] Weiterführend noch einige Aspekte, die das System der Wahl auf jeden Fall mitbringen muss.

Kommunikation mit dem Contestserver

Da der Contest auf einer Client/Server-Architektur basiert, muss jeder der sechs sich auf dem Spielfeld befindlichen Agenten direkt mit dem Server kommunizieren, sprich laut festgelegtem Protokoll eine TCP-Verbindung zum Server aufbauen und halten. Ferner muss jeder Agent die eintreffenden XML-Pakete verarbeiten und die eingetroffenen Daten gegebenenfalls in die eigene Beliefbase einarbeiten, als auch protokollgerechte XML-Pakete erzeugen und versenden.

Klassifizierung: must

[1] zu finden unter: http://cig.in.tu-clausthal.de/agentcontest2008
[2] zu finden unter: http://jade.tilab.com
[3] zu finden unter: http://vsis-www.informatik.uni-hamburg.de/projects/jadex
[4] zu finden unter: http://www.jiac.de

Kommunikation zwischen den Agenten

Jeder Agent bekommt pro Runde ein Datenpaket vom Server, in dem seine aktuelle Vision beinhaltet ist.[5] Um diese Informationen, z.B. die über die Positionen von Hindernissen oder Kühen, auch den anderen Agenten zuteil kommen zu lassen, müssen die Agenten untereinander kommunizieren.

Ebenfalls denkbar ist z.B. ein als Koordinator eingesetzter Agent, für den auch die Kommunikation zwischen Agenten essentiell wäre.

Klassifizierung: must

Graphische Aufbereitung des aktuellen Simulationsgeschehens

Debugging in Multiagentensystemen ist eine der schwierigeren Aufgaben und somit ist es für den Entwickler absolut unerlässlich so früh wie möglich eine Visualisierung der Simulation umzusetzen, um alle weiteren Implementationen und deren Auswirkungen direkt sehen zu können.[6]

Klassifizierung: must

Reaktionszeit des Systems

Laut Protokollbeschreibung muss das System auf die REQUEST-ACTION Nachrichten des Servers innerhalb eines in dieser Nachrichten spezifizierten Zeitlimits antworten, ansonsten wird automatisch die SKIP Aktion für den jeweiligen Agenten ausgeführt. Dieses Zeitlimit liegt laut Szenariobeschreibung bei mindestens 2 und maximal 10 Sekunden.

Stabilität

Der Contest wird laut Beschreibung möglicherweise über mehrere Tage hinweg laufen, was die Anforderung an die Sprache/Plattform stellt, möglichst stabil zu laufen. Dazu gehört auch die Möglichkeit des automatischen Neustarts eventuell abgestürzter Agenten.

Klassifizierung: must

[5] Weiterhin beinhaltet dieses Paket den aktuellen Spielstand und die momentane Runde.
[6] Gegebenenfalls als räumliches Modell.

3.3 Fazit

Die Anforderungen sind nun erhoben und klassifiziert worden. Der Übersichtlichkeit halber werden alle fachlichen und alle technischen Anforderungen noch einmal in Tabelle 3.1 dargestellt.

fachliche Anforderung	**Klassifizierung**
Navigation (Pathfinding) der Agenten auf dem Spielfeld	must
Erkundung des Spielfelds	must
Steuerung von einzelnen Kühen bzw. Schwärmen	must
Stören des Gegners	nice to have
Spieltheoretischer Ansatz	nice to have
technische Anforderung	**Klassifizierung**
Multiagentensprache bzw. -plattform	must
Kommunikation mit dem Contestserver	must
Kommunikation zwischen den Agenten	must
Graphische Aufbereitung des aktuellen Simulationsgeschehens	must
Reaktionszeit des Systems	must
Stabilität	must

Tabelle 3.1: fachliche & technische Anforderungen

Anhand dieser Anforderungen kann nun im folgenden Kapitel 4 „Technische Analyse“ weitergearbeitet werden.

4 Technische Analyse

In diesem Kapitel werden verschiedene Technologien und mögliche Lösungsansätze für die in den Anforderungen als Hauptaspekte herausgearbeiteten Punkte diskutiert, bewertet[1] und es wird eine Auswahl getroffen, die mitbestimmend für die spätere Realisierung des Prototypen ist.

Dazu wird zuerst anhand der technischen Anforderungen die Wahl der Multiagentensprache/-plattform vorgenommen, da diese starken Einfluss auf die Bewertung und Wahl der weiteren Technologien hat.[2]

Weiterhin werden dann folgende aus den fachlichen Anforderungen herausgearbeiteten Hauptaspekte analysiert:

- Navigation (Pathfinding) der Agenten auf dem Spielfeld (Anforderung beschrieben unter Kapitel 3.1.1)
- Erkundung des Spielfelds (Anforderung beschrieben unter Kapitel 3.1.2)
- Steuerung von einzelnen Kühen bzw. Schwärmen (Anforderung beschrieben unter Kapitel 3.1.3)

Zusätzlich wird auch noch der dynamische Aspekt des Szenarios analysiert.

Am Ende des Kapitels werden alle im Prototypen zum Einsatz kommenden Technologielösungen noch einmal zusammenfassend aufgeführt.

4.1 Wahl der Multiagentensprache/-plattform

Die Auswahl von Multiagentensprachen und -plattformen ist mittlerweile nicht mehr völlig überschaubar, sowohl problemspezifische Lösungen, kommerzielle Produkte und OpenSource-Projekte sind erhältlich. Da diese Arbeit aber nicht den Anspruch hat, eine wirklich tiefgehende Analyse zwischen verschiedenen Lösungen für Multiagentensysteme

[1] Bewertung erfolgt anhand der Effizienz der Lösung für die jeweilige Problemstellung.
[2] bezüglich Umsetzbarkeit in bestimmten Programmiersprachen, bzw. bereits vorhandener Implementationen

vorzunehmen, wird im Folgenden eine Machbarkeitsanalyse anhand der Multiagentenplattform Jadex vorgenommen, worauf dann in einem Fazit noch Schlüsselelemente mit anderen Plattformen verglichen werden.

4.1.1 Machbarkeitsanalyse Jadex

Zuerst erfolgt an dieser Stelle eine Kurzbeschreibung der Jadex-Plattform, woraufhin die in Tabelle 3.1 auf Seite 26 festgehaltenen technischen Anforderungen hinsichtlich der Realisierungsmöglichkeit mit Jadex ebenfalls in einer Tabelle abgearbeitet werden. Einzelne Punkte, die noch einer kurzen Erklärung bedürfen, werden zuletzt besprochen.

Jadex ist eine Erweiterung der JADE Plattform. Einen guten Überblick über die JADE Plattform und die Jadex Erweiterung gibt Sudeikat in seiner Diplomarbeit [Sudeikat (2004)]. So beschreibt er die JADE Plattform als ein in Java™ implementiertes Framework, das die Implementation von Multiagentensystemen durch eine FIPA konforme Middleware, eine Klassenbibliothek und weitere Werkzeuge ermöglicht und unterstützt. Weiterhin erklärt er, dass Jadex die JADE Plattform erweitert, und zwar indem die BDI-Architektur implementiert wurde. Jadex bietet umfangreiche Werkzeuge für Entwickler zur Definition und Implementation von BDI-Agenten und geht somit über den Schritt einer reinen Middleware hinaus. BDI-Agenten bestehen dabei in Jadex aus zwei Hauptbestandteilen. Hierbei umfasst der erste Teil eine XML-Datei, das *Agent Definition File* (ADF), welches die dem Agenten bekannten initialen Ziele, Pläne und angebotene Services beschreibt und der zweite Teil die in Java™ vorgenommene Implementation der im ADF beschriebenen Pläne. Weiterhin integriert Jadex eine FIPA-konforme[3] *Agent Communication Language* (ACL), die es den Agenten erlaubt untereinander zu kommunizieren.

Weiterhin bindet Jadex ein Modularisierungskonzept über sogenannte *Capabilities* ein [Braubach u. a. (2005); Busetta u. a. (2000)]. Eine solche Capability ist unter Jadex wie ein Agent aufgebaut, es bestehen ein ADF und zugehörige Pläne. Die Capability wird dann im eigentlichen Agenten aufgerufen, woraufhin ihre Goals, Beliefs und Pläne für diesen verfügbar werden.

Eine genaue Beschreibung der Jadex-Architektur findet man in oben genannter Diplomarbeit [Sudeikat (2004)] bzw. auf der Homepage des Jadex-Projekts.[4]

Tabelle 4.1 auf Seite 29 bespricht die technischen Anforderungen hinsichtlich der Realisierung durch Jadex.

In den folgenden Unterkapiteln werden die Anforderungen besprochen, die noch einer näheren Erläuterung bedürfen.

[3] Foundation for Intelligent Physical Agents, Spezifikationen zu finden unter: http://www.fipa.org
[4] zu finden unter: http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

technische Anforderung	Realisierung durch Jadex
Kommunikation mit dem Contestserver	möglich
Kommunikation zwischen den Agenten	wird unterstützt
Graphische Aufbereitung des aktuellen Simulationsgeschehens	möglich
Reaktionszeit des Systems	möglich
Stabilität	wird unterstützt

Tabelle 4.1: technische Anforderungen: Analyse

Kommunikation mit dem Contestserver

Da Jadex in Java™ implementiert ist und Java™ auch die Grundlage aller Pläne in Jadex darstellt, ist es ohne weiteres möglich, eine TCP-Verbindung mit dem Server herzustellen und zu halten. Begünstigt wird hier Jadex zusätzlich dadurch, dass der Wettbewerbausrichter auch eine in Java™ implementierte Klasse zum Verbindungsaufbau mit dem Server bereitgestellt hat, die überdies auch XML-Pakete verarbeiten und erzeugen kann, wodurch eine nahezu vollständige Fokussierung auf das Hauptthema der Arbeit ermöglicht wird.

Graphische Aufbereitung des aktuellen Simulationsgeschehens

Wie schon unter Kommunikation mit dem Contestserver beschrieben, kann hier auf Java™ zurückgegriffen werden. Unter Java™ gibt es z.B. Swing oder AWT zur Implementation einer Visualisierung.

Reaktionszeit des Systems

Dem Autor liegen leider keine Ausführungen hinsichtlich der Reaktionszeit von Jadex vor, allerdings wurde mit Jadex bereits im Zusammenhang mit einer Echtzeitumgebung[5] gearbeitet [Renz (2007)], so dass diese Anforderung als realisierbar gewertet wird.

Stabilität

Jadex bietet verschiedene Möglichkeiten die Stabilität und auch das unbeaufsichtigte Laufen zu gewährleisten. Ein Beispiel dafür ist ein Observer-Agent, der alle gewünschten beim

[5] In diesem Fall war es die Entwicklung von Gamebots in Jadex für Unreal Tournament 2003™.

Directory Facilitator (DF)[6] angemeldeten Agenten überwacht und bei Bedarf neustarten kann.

4.1.2 Fazit

Jadex kann in allen Punkten den Anforderungen als Multiagentenplattform gerecht werden. Ein großer Vorteil von Jadex zum Beispiel zu JADE[7] sind die vorhandenen Werkzeuge zur Implementation von Agenten bzw. BDI-Agenten, auf die ein Entwickler zurückgreifen kann. Es bestehen auch weitere Vorteile zu anderen Multiagentenplattformen, die von verschiedenen Teams im Laufe des Contests benutzt worden sind. Ein Beispiel ist JASON,[8] welches einen Java™-basierten Interpreter der Multiagentensprache AgentSpeak(L) [Rao (1996)] und eine Plattform zur Kommunikation der Agenten aufweist, aber aufgrund der Struktur des Interpreters Schwächen in der Reaktionsgeschwindigkeit hat. Dies hat sich im Laufe des Contests auch bewahrheitet. Als letztes wird hier das oben genannte und in dieser Form einzigartige Modularisierungskonzept von Jadex aufgeführt, welches besonders hinsichtlich der Wiederverwendbarkeit als Vorteil gegenüber allen anderen Plattformen gewertet wird.

Somit wird zur Entwicklung des Prototypen die Jadex-Plattform genutzt, deren Besonderheiten sich natürlich auch in Architektur und Design auswirken.

[6] Der Directory Facilitator ist der von Jadex mitgebrachte Verzeichnisdienst, bei dem sich Agenten für verschiedene Dienste anmelden können und fortan über den DF auffindbar sind.

[7] zu finden unter: http://jade.tilab.com

[8] zu finden unter: http://jason.sourceforge.net

4.2 Analyse gängiger Wegfindungsalgorithmen

Informierte Suchalgorithmen bzw. der A* Algorithmus im speziellen sind ein allgemein anerkanntes Verfahren zur Lösung von Wegfindungsproblemen und finden einen breiten Einsatz z.B. in Computerspielen. Trotzdem wird an dieser Stelle der A* Algorithmus selbst, sowie seine Einsatzmöglichkeiten in Bezug auf das Szenario analysiert, um zu verdeutlichen, warum gerade in diesem Szenario der Einsatz des A* Algorithmus besonders effizient ist.

4.2.1 A* Algorithmus

Der A* Algorithmus gehört zu der Klasse der informierten Suchalgorithmen. Russel und Norvig beschreiben die grundlegende Funktionsweise des Algorithmus wie folgt [Russel und Norvig (2003)]:

Der Algorithmus bewertet Knoten durch Kombination von $g(n)$, den tatsächlichen Kosten um den Knoten n zu erreichen, und $h(n)$, den geschätzten Kosten um vom Knoten n zum Ziel zu gelangen:

$$f(n) = g(n) + h(n).$$

Daraus ergibt sich weiterhin:

$$f(n) = \text{geschätzte Kosten der günstigsten Lösung über } n.$$

Soll also versucht werden die günstigste Lösung zu finden, wird zuerst der Knoten mit dem niedrigsten Wert $f(n)$ gesucht. Solange dabei die heuristische Funktion $h(n)$ zulässig (sie überschätzt die Kosten nie) bzw. monoton ist,[9] gilt für den A* Algorithmus:

- vollständig: Falls eine Lösung existiert, wird sie gefunden.
- optimal: Es wird immer die optimale Lösung gefunden. Existieren mehrere optimale Lösungen, wird eine davon gefunden (abhängig von Implementierungsdetails).
- optimal effizient: A* expandiert eine minimale Anzahl an Knoten, es gibt also keinen anderen Algorithmus, der die Lösung unter Verwendung der gleichen Heuristik schneller findet.

[9] je nachdem ob mit einer Closed-List gearbeitet wird oder nicht. Ohne Closed-List reicht die Eigenschaft *zulässig*.

Einsatzmöglichkeit zur Wegfindung der Agenten

Da die Sichtweite der Agenten acht Zellen in jede Richtung beträgt, ist es effektiv von Simulationsbeginn an möglich, die Wege der Agenten mit dem A* Algorithmus zu optimieren. Bedenkt man weiter, dass ein BDI-Agent die Möglichkeit hat Informationen in seiner *Belief-base* zu hinterlegen, wächst die Effizienz des A* Algorithmus mit den Informationen über das Spielfeld zusammen an.

Nachteil: Aufgrund der anfänglich unvollständigen Informationen über das Spielfeld, sowie der Dynamik des Szenarios, ist es unerlässlich in jeder Runde aufs neue mit dem A* Algorithmus einen vollständigen Weg zu suchen.

Einsatzmöglichkeit zur Wegfindung für die Schwarmsteuerung

Eine weitere Einsatzmöglichkeit des A* Algorithmus ist die der Wegfindung für die Schwarmsteuerung. So sollen grössere Schwärme unter Umständen nicht nah an Hindernisketten vorbei gesteuert werden, um zu vermeiden, dass einzelne Kühe in Ecken oder ähnlichem hängenbleiben. Da der A* Algorithmus ja mit Kosten arbeitet, wäre an dieser Stelle ein Pre-Processing der in den A* Algorithmus eingegebenen Repräsentation des Spielfelds möglich, bei dem Bereiche um Hindernisse herum z.B. mit höheren Kosten ausstaffiert werden.

4.2.2 Fazit

Eine effiziente Methode der Wegfindung ist auf beliebig komplex mit Hindernissen versehenen Spielfeldern unerlässlich, um die Zahl der pro Strecke benötigten Runden so niedrig wie möglich zu halten. Da der A* Algorithmus im Zusammenspiel mit einer zulässigen bzw. monotonen Heuristik *optimal effizient* ist, wird er dieser Anforderung gerecht.

Weiterhin kann der A* Algorithmus als Grundbaustein für die Schwarmsteuerung genutzt werden, womit er für zwei Anforderungen eine wichtige Rolle spielt.

Der A* Algorithmus ist somit die Technologie der Wahl zur Realisierung der Wegfindung der Agenten, sowohl im Bereich der direkten Wegfindung der Agenten, als auch im Bereich Wegplanung für die Schwarmsteuerung, und wird somit auch in die Realisierung des Prototypen einbezogen.

4.3 Analyse der Möglichkeiten zur Anpassung an die dynamische Umgebung

Die drei herausgearbeiteten Anforderungen bezüglich der Erkundung des Spielfelds, der Steuerung von Schwärmen und ggf. des Störens des Gegners sind konkurrierende Aktivitäten. Daher ist es notwendig einen Mechanismus zu finden, durch den alle drei Anforderungen ermöglicht werden, unter dem Aspekt, dass die Umgebung dynamisch ist und diese drei Anforderungen in unterschiedlicher Quantität benötigt werden.

Die zwei offensichtlichsten Methoden dafür werden im Folgenden kurz bezüglich des Aufgabenfelds besprochen und im Anschluss bewertet.

4.3.1 Homogenes Agententeam mit Rollenabhängigkeit

Eine Möglichkeit alle Aktivitäten zu vereinbaren wäre es, ein homogenes Agententeam zu entwickeln, bei dem jeder Agent je nach Notwendigkeit entweder die Rolle eines Erkunders, die Rolle eines Schwarmtreibers oder die Rolle eines „Gegnerstörers“ annehmen kann. Ob diese Entscheidung vom Agenten selbst oder von z.B. einem Koordinator Agenten getroffen würde, ist dabei noch nicht relevant. Beides wäre möglich.

4.3.2 Heterogenes Agententeam

Eine andere Möglichkeit wäre es, ein heterogenes Agententeam zu entwickeln, bei dem verschiedene Aufgaben auf die einzelnen Agenten verteilt werden. Dabei wären unterschiedliche Konstellationen denkbar, z.B. vier Schwarmtreiber Agenten, ein Erkunder Agent und ein „Gegnerstörer“ Agent. Diese Agenten wären dann permanent jeweils nur zur Abarbeitung ihrer spezifischen Aufgabe fähig.

4.3.3 Fazit

Durch den Aufbau eines heterogenen Agententeams geht sehr viel Flexibilität verloren, besonders unter Berücksichtigung der dynamischen Umgebung. Ein homogenes Team mit Rollenabhängigkeit der einzelnen Agenten erhält die Flexibilität ohne dabei anderen Nachteilen zu unterliegen. Dementsprechend fällt die Entscheidung eindeutig zugunsten eines homogenen Agententeams mit Rollenabhängigkeit aus und wird sich auch so im Design wiederfinden.

4.4 Analyse der Planungsarten zur Steuerung von Schwärmen

Um Schwärme von Kühen über das Spielfeld zum eigenen Gatter steuern zu können, ist es notwendig die Vorgehensweise dafür zu planen. Dies betrifft sowohl die Auswahl eines Schwarms und die Koordination der Agenten, als auch den Weg, den ein gesteuerter Schwarm nehmen soll.

Für die Art und Weise in der die Planung implementiert werden soll, bestehen drei Ansätze [Wooldridge (2002)], die an dieser Stelle alle kurz unter Berücksichtigung der Problemstellung betrachtet werden. Im Anschluss wird dann eine Bewertung und Auswahl vorgenommen.

4.4.1 Zentralisiertes Planen für verteilte Pläne

Um diese Form des Planens zu realisieren, müsste zusätzlich zu den sechs *agierenden* Agenten ein weiterer Planungsagent eingeführt werden, bei dem alle Informationen der sechs Agenten zusammenfließen. Daraufhin kann dieser Agent Pläne entwickeln und diese den anderen Agenten zur Verfügung stellen.

4.4.2 Verteiltes Planen

Soll die Problematik durch verteiltes Planen gelöst werden, müsste das System nicht wie beim *zentralisierten Planen* um einen, sondern um eine Gruppe von Agenten erweitert werden. Diesen müssten ebenfalls von den *agierenden* Agenten Informationen zur Verfügung gestellt werden, anhand derer sie dann kooperativ einen zentralisierten Plan entwerfen. Dabei würde die Gruppe der Planeragenten typischerweise aus *Spezialisten* bestehen, die jeweils bestimmte Aspekte der Planung übernehmen. Ausgeführt würde der entwickelte Plan dann von den sechs *agierenden* Agenten.

4.4.3 Verteiltes Planen für verteilte Pläne

Bei dieser Form des Planens müsste das System um keine weiteren Agenten erweitert werden. Die Agenten müssten kooperieren um individuelle Pläne aufzustellen und ihre Aktivitäten dynamisch koordinieren. Bei eventuell auftretenden Koordinationskonflikten müssten Verhandlungen geführt werden.

4.4.4 Fazit

Verteiltes Planen für verteilte Pläne ist sicherlich die eleganteste Form in einem Multiagentensystem zu planen. Allerdings bringt diese mit sich, dass den Agenten sehr gute Koordinations- und Verhandlungsmechanismen implementiert werden müssten. Auch ein damit verbundenes hohes Kommunikationsaufkommen im System wäre die Folge. Aufgrund der Komplexität eines solchen Kooperationsmodells wird von dieser Planungsart Abstand genommen.

Verteiltes Planen und *Zentralisiertes Planen für verteilte Pläne* ähneln sich in der Hinsicht, dass jeweils zentralisierte Pläne entwickelt werden. Für die gegebene Problemstellung der Planung scheint jedoch eine Gruppe von Planeragenten überdimensioniert, da augenscheinlich erst weitaus komplexere Planungsaufgaben diese Form der Planung rechtfertigen.

Damit ist die Planungsart der Wahl das *Zentralisierte Planen verteilter Pläne* und wird somit einen wesentlichen Teil der Architektur des Prototypen mitbestimmen.

4.5 Fazit

Die technische Analyse wurde vorgenommen und hat folgendes ergeben:

- Auswahl von Jadex als Multiagentenplattform
- Nutzung des A* Algorithmus zur Wegfindung und Planung der Schwarmsteuerung
- Einführung eines rollenbasierten, homogenen Agententeams um dem dynamischen Charakter des Szenarios gerecht zu werden
- Planungsform: zentralisiertes Planen für verteilte Pläne

Somit kann nun im folgenden Kapitel mit der Architektur und dem Design fortgefahren werden.

5 Design

Das Multiagentensystem unterliegt der inkrementellen Entwicklung von Prototyp Agenten, die sich aus Anforderungen und technischer Analyse ergeben haben. Vor Verfügbarkeit der Testserver-Umgebung wurde mit einer modifizierten Version des in Jadex enthaltenen *Hunter-Prey Szenarios*[1] als Grundlage gearbeitet wurde, um Anhand erster Prototypen Schwarmsteuerungsstrategien entwickeln und bewerten zu können. Die in diesem Kapitel vorgestellte Architektur und das Design der Agententypen spiegeln allerdings den Stand des MAS-Prototypen wieder, der am Contest teilgenommen hat, alle Schritte des Prototypings hier zu besprechen würde den Rahmen dieser Arbeit sprengen.

Im Folgenden wird zuerst die gewählte Architektur für das MAS erläutert. Daraufhin wird dann auf die herausgearbeiteten Agententypen eingegangen und deren Feindesign bis direkt vor die Implementationsebene vorgestellt. Zuletzt wird die Funktionsweise des Gesamtsystems erläutert, um letzte eventuell offene Fragen zu klären.

Da die *Unified Modeling Language*™(UML®) [OMG] zwar für die Objektorientierte Softwareentwicklung ein sehr gutes Mittel zur Modellierung darstellt, aber hinsichtlich MAS-Spezifika nicht geeignet ist Multiagentensysteme hinreichend zu modellieren, muss auf andere Methodiken zurückgegriffen werden. Mögliche Methodiken sind z.B. Tropos [Troposproject], Prometheus [RMIT] oder AUML [FIPA a].

Die Wahl fällt auf Tropos, da Tropos gute Möglichkeiten vorgibt, die Architektur des Systems zu modellieren, ebenso wie das Design. Außerdem kam Tropos schon für das *Preliminary Design* zum Einsatz, das zur Teilnahme am Contest eingereicht werden musste. Weiterhin werden UML®Sequenzdiagramme genutzt, um den Ablauf der Interaktionen darzustellen.

Anmerkung: Die Notationen in Tropos, sowie auch die spätere Implementation werden englischsprachig aufgeschrieben, da es so einfacher ist die Konsistenz zwischen Programmiersprachen-Konstrukten und Bezeichnern zu wahren.

[1] In Jadex implementiertes MAS, in dem „Jäger“ Agenten auf einem Gitter-Spielfeld „Beute“ Agenten verfolgen und „fressen“ sollen.

5.1 Architektur des MAS

In diesem Kapitel wird nicht die Gesamtsystem-Architektur inklusive der Jadex-Plattform vorgestellt, da diese nicht Thema der Arbeit ist, sondern die Architektur des MAS-Prototypen.

Die in der technischen Analyse getroffene Entscheidung auf ein *zentralisiertes Planen verteilter Pläne* zu setzen, hat zur Folge, dass das System aus sechs homogenen, im Folgenden als *Teammate Agenten* bezeichneten Agenten und einem Koordinator-Agenten, im Folgenden als *Herding Officer* bezeichnet, besteht. Dadurch, dass es ein homogenes Agententeam ist und somit alle Teammate Agenten dieselben Beziehungen, Abhängigkeiten, Ziele, Pläne, etc. haben, wird in allen folgenden Tropos-Modellen darauf verzichtet diese alle darzustellen. Stattdessen wird nur ein einzelner Teammate Agent stellvertretend für alle modelliert.

Abbildung 5.1 auf Seite 38 veranschaulicht die gewählte Architektur. Dort werden sowohl die identifizierten Agententypen als auch deren Abhängigkeiten untereinander verdeutlicht, wie auch die Kommunikation mit dem Contest-Server. So hängt der Herding Officer Agent von der Kommunikation der Teammate Agenten mit ihm ab, bezüglich deren Wahrnehmung und des Spielstatus, um diese Informationen vorhalten und die zentralisierte Planung durchführen zu können. Auf der anderen Seite hängen die Teammate Agenten vom Herding Officer Agenten ab, um die Ergebnisse der zentralisierten Planung abrufen (mit denen auch das rollenbasierte Verhalten einhergeht, s. 5.2.1 „Teammate Agent“ auf Seite 39) sowie Hilfestellung bei der Auswahl eines zu explorierenden Bereichs des Spielfelds erhalten zu können. Die Teammate Agenten kommunizieren direkt mit dem Contest-Server. Sie hängen damit davon ab, dass dieser ihnen ihre lokalen Visions liefert, und können Bewegungsaktionen an diesen versenden.

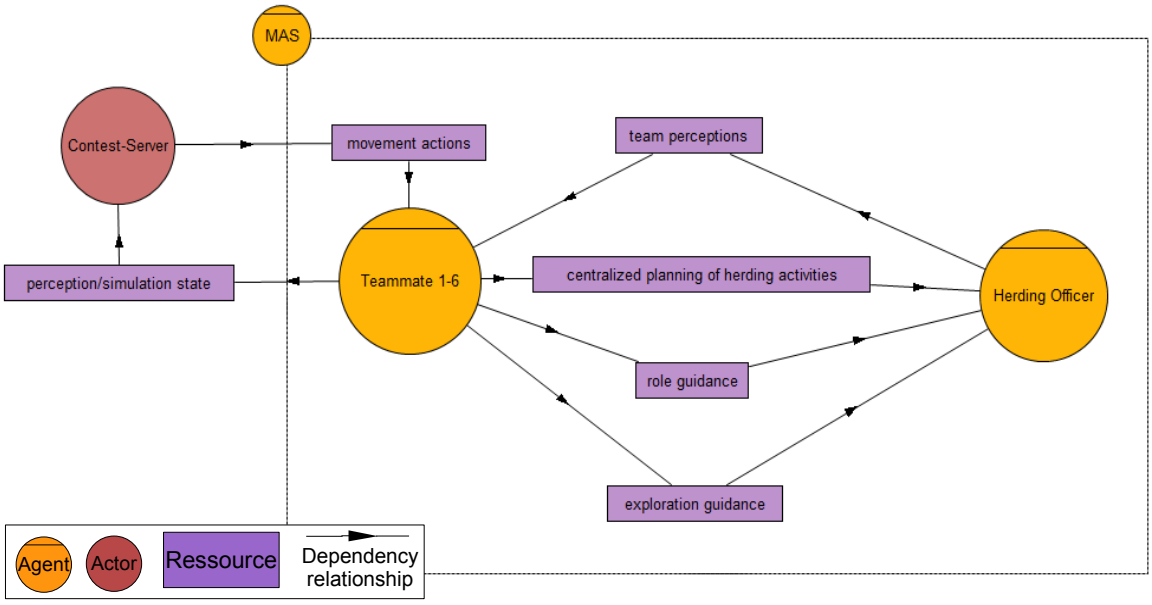

Abbildung 5.1: Architektur des Prototypen. Tropos Modell für die Abhängigkeiten zwischen Teammate Agenten, dem Herding Officer Agenten, sowie dem Contest-Server.

5.2 Design der Agententypen

Dieses Kapitel dient der genaueren Beschreibung der beiden in der Architektur vorgestellten Agententypen. Dazu wird jeweils ein Tropos Design-Diagramm zur Veranschaulichung des detaillierten inneren Aufbaus der Agenten verwendet, sowie eine textuelle Beschreibung bzw. ein grober Pseudocode der wichtigsten Pläne/Capabilities, da sowohl die Tropos Methodiken, als auch andere Methodiken hinsichtlich der Beschreibung von Capabilities und Plänen vom Autor als nicht anschaulich genug bewertet werden.

5.2.1 Teammate Agent

Die Teammate Agenten sind die sechs direkt mit dem Contest-Server in Verbindung stehenden, agierenden Agenten. Wie schon erwähnt sind diese homogen.

Abbildung 5.2 auf Seite 40 beschreibt den inneren Aufbau eines Teammate Agenten. Ganz oben in der Hierarchie steht das „Cow Herding“ Goal, dieses wählt den „Cow Herding“ Plan zur Erfüllung aus. Leider bietet Tropos nicht die Möglichkeit der Modellierung von Sub-Goals die aus Plänen heraus instanziert werden. Eigentlich ist nämlich das „Explore Area“ Goal ein vom „Cow Herding“ Plan instanziertes Sub-Goal, das genau dann instanziert wird, wenn laut Planung des Herding Officer Agenten kein Schwarm getrieben werden muss. Im Diagramm wird dieses Goal als *oder* Möglichkeit zu dem vom „Cow Herding“ Goal gewählten Plan modelliert, was dem immerhin nahe kommt. Das „Explore Area“ Goal greift wiederum zur Realisierung auf den „Explore Area“ Plan zurück. Die beiden Goals „Inform Herding Officer about Initial Values“ und „Inform Herding Officer about Perception“ werden von der „Client-Server Kommunikation“ Capability aus instanziert, sobald diese Informationen in Form von Nachrichten vom Contest-Server in der Beliefbase der Agenten gespeichert wurden (Auch hierfür besteht leider keine Modellierungsmöglichkeit). Diese beiden Goals wählen dann jeweils ihren Plan zur Realisierung.

Im Folgenden werden sowohl die Capability zur Client-Server Kommunikationen, als auch die beiden Pläne „Cow Herding“ und „Explore Area“ genauer betrachtet. Die Funktionsweise dieser beiden überschaubaren Pläne im Teammate Agenten wird dabei mittels selbsterklärendem Pseudocode beschrieben. Die den Goals „Inform Herding Officer about Initial Values“ und „Inform Herding Officer about Perception“ zugeordneten Pläne werden ausgelassen, da diese als trivial bewertet werden.

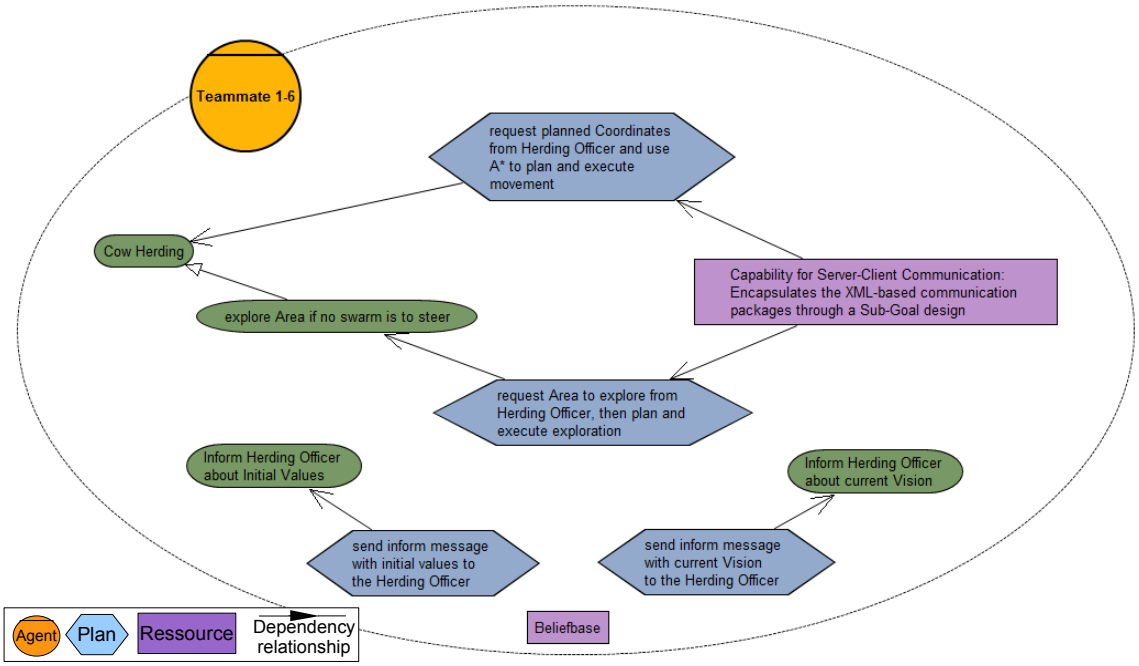

Abbildung 5.2: Design des Teammate Agenten. Tropos Diagramm zur Beschreibung der Goals und der von diesen auswählbaren Pläne. Beliefbase und Capability werden hier als Ressource modelliert.

Capability zur Kapselung der Agent-Server Kommunikation

Diese Capability kapselt die gesamte Client-Server Kommunikation. Dazu instanziert sie eine vom Contest-Ausrichter bereitgestellte, leicht modifizierte Java-Klasse als Adapter und speichert diese Instanz in der Beliefbase des Agenten. Diese Java-Klasse bekommt einen *External Access* zum Agenten, und kann über diesen wie agenteneigene Pläne die Beliefbase modifizieren und Sub-Goals instanzieren.

Diese Adapterklasse baut eine TCP Verbindung zum Contest-Server auf und hält sie. Ankommende XML-Pakete werden mittels DOM geparsed und die Daten in der Beliefbase des Agenten verfügbar gemacht. Nach dem Speichern der Daten in der Beliefbase wird dann je nach Art der Daten (Initial Werte oder Vision-Daten) das jeweilige oben genannte Inform Sub-Goal instanziert.

Weiterhin bietet die Capability ein „Move“ Goal, das der Agent zusammen mit einem Richtungsparameter nutzen kann, um dem Contest-Server die geplante Richtung für die aktuelle Runde zu übermitteln. Dazu wird im zugehörigen Plan ein protokollkonformes XML-Paket zusammengesetzt und dem Contest-Server mittels des Adapters über die bestehende TCP

Verbindung gesendet. Zusätzlich wird der aktuell geplante Weg an den Herding Officer versendet.

Die Adapterklasse und die Nutzung von DOM werden hier nicht weiter erläutert, da diese Klasse zur Verfügung stand und auf der Homepage des Contests heruntergeladen werden kann.

Plan: Cow Herding

```
1  wiederhole:
2  {
3          blockierendes Warten auf Rundenwechsel/ersten Rundenbeginn;
4          beziehe alle benötigten, aktualisierten Werte aus der Beliefbase;
5          blockierendes Warten auf Planungsvollzugsbestätigung des Herding
               Officer Agenten;
6          frage Herding Officer nach Koordinaten zur Schwarmsteuerung;
7          blockierendes Warten auf Antwort;
8          wenn valides Koordinatenpaket empfangen wurde:
9          {
10                 plane den Weg zur Koordinate mittels A*;
11                 instanziere Sub-Goal "'Move"' mit errechneter Richtung;
12         }
13         sonst:
14         {
15                 instanziere Sub-Goal "'Explore Area"';
16         }
17 }
```

Abbildung 5.3: Pseudocode Plan: Cow Herding

Plan: Explore Area

```
1  beziehe alle benötigten, aktualisierten Werte aus der Beliefbase;
2  wenn noch kein Explorationsplan in der Beliefbase vorliegt:
3  {
4          frage Herding Officer nach zu explorierender Area;
5          blockierendes Warten auf Antwort des Herding Officers;
6  }
7  berechne nächste Zielkoordinate zur Exploration der Area;
8  wenn berechnete Zielkoordinate valide:
9  {
10         plane den Weg zur Koordinate mittels A*;
11         speichere Explorationsplan in der Beliefbase;
12         instanziere Sub-Goal "'Move"' mit errechneter Richtung;
13 }
14 sonst:
15 {
16         lösche Explorationsplan aus Beliefbase;
17 }
```

Abbildung 5.4: Pseudocode Plan: Explore Area

5.2.2 Herding Officer Agent

Der Herding Officer Agent ist für die zentralisierte Planung zuständig. Aus diesem Grund werden ihm alle von den Teammate Agenten gesammelten Daten kommuniziert. Diese Daten werden jede Runde aufs Neue ausgewertet und ein Plan wird erstellt. Außerdem hält er in seiner Beliefbase eine instanzierte Java™Swing Klasse vor, die ständig alle von ihr überwachten Änderungen in der Beliefbase visuell umsetzt, um dem Entwickler die Möglichkeit zu geben, die Geschehnisse auf dem Spielfeld zu beobachten.

Abbildung 5.5 auf Seite 43 beschreibt den inneren Aufbau des Herding Officer Agenten. Die fünf durch Message-Events getriggerten Pläne sind als trivial zu betrachten. In diesen Plänen werden entweder die genannten Daten in der Beliefbase eingelagert (Initiale Werte, Vision der Teammate Agenten und geplante Wege der Teammate Agenten) oder die von den Teammate Agenten angeforderten Daten (Koordinaten zur Schwarmsteuerung und zu explorierende Area) an diesen zurückgesendet. Ist der entsprechende Teammate Agent nicht zur Schwarmsteuerung eingeteilt, wird ein Objekt mit dem Wert „NULL“ versendet. Diese Pläne werden nicht näher erläutert. Der Hauptaspekt im Design des Herding Officer Agenten ist das „Centralized Planning“ Goal. Dieses wird durch eine im ADF verankerte Condition

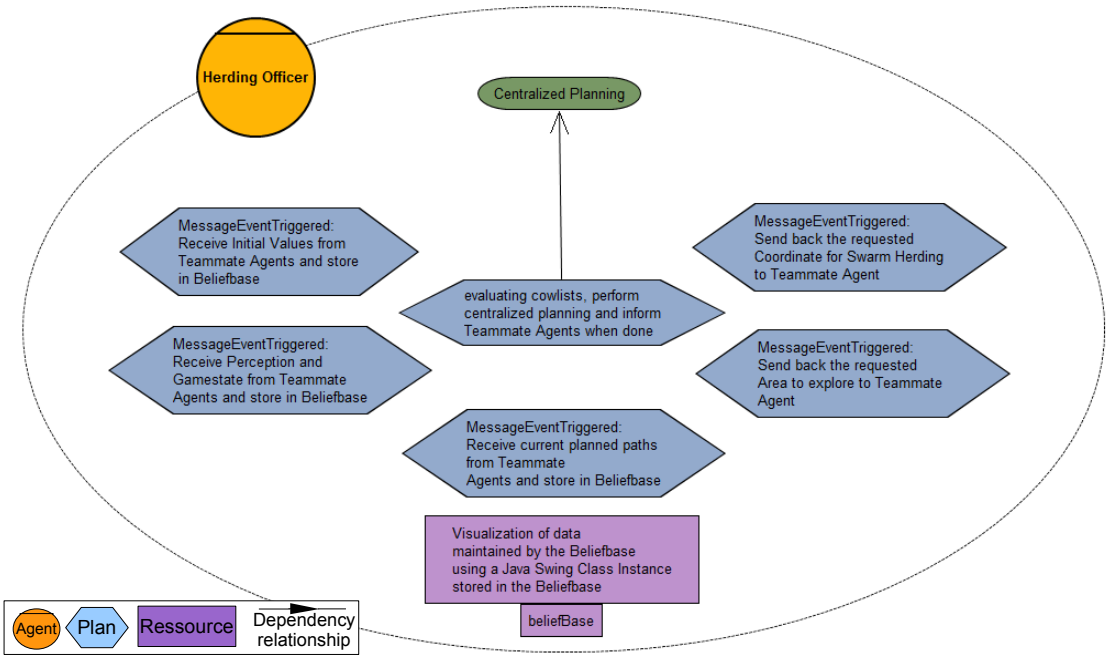

Abbildung 5.5: Feindesign des Herding Officer Agenten. Tropos Diagramm zur Beschreibung der Goals und der von diesen auswählbaren Pläne. Beliefbase und Visualization werden hier als Ressource modelliert.

getriggert, die dann erfüllt ist, wenn die Informationen über die aktuelle Vision aller Teammate Agenten durch den zugehörigen Plan in der Beliefbase gespeichert wurden. Diesem Goal zugeordnet ist ein Plan, in dem die Auswertung und Zusammenführung der Visions der einzelnen Teammate Agenten und daraufhin die zentralisierte Planung vorgenommen wird. Zuletzt wird von dem Plan eine Vollzugsmeldung an die Teammate Agenten verschickt. Dieser Plan wird im Folgenden aufgrund seiner Komplexität noch näher besprochen und auch noch weiterführend in Kapitel 6 „Ausgewählte Aspekte der Realisierung“ erläutert. Weiterhin hält der Herding Officer Agent eine Instanz einer Java™Swing Klasse in der Beliefbase vor, die zur Visualisierung des Spielgeschehens über einen *External Access* auf die Beliefbase zugreift und alle Änderungen direkt visuell anzeigt. Beliefbase und Visualisierung sind hier als Ressource modelliert.

Plan: Evaluate Cowlists and Plan Swarm Steering

Zur besseren Veranschaulichung, wird hier der Ablauf des Plans gelistet erläutert, da der Plan sehr umfangreich ist und die Verwendung von Pseudocode oder anderer Diagramme diesen nur schlecht veranschaulichen würde.

Ablauf des Plans:

1. Beziehe alle benötigten, aktualisierten Werte aus der Beliefbase.
2. Führe die aktuellen Vision-Daten der einzelnen Teammate Agenten bezüglich tatsächlicher und vermuteter Kuh-Positionen unter Konsistenzwahrung zusammen.
3. Bilde Anhand der Entfernung zwischen den Kühen Schwärme aus diesen.
4. Errechne den Koordinatenmittelpunkt der Schwärme durch Mittelung der Koordinaten der zugehörigen Kühe.
5. Vergleiche die Schwärme der letzten und der aktuellen Runde miteinander.
6. Kopiere für alle Treffer, wenn vorhanden, die Zusatzdaten (eingeteilte Teammates, nichtBewegtSeit) der alten Schwärme auf die neuen.
7. Kopiere alle Schwärme, für die Teammates eingeteilt sind, in die zuTreiben-Liste.
8. Ziehe die bereits eingeteilten Teammates von der Liste verfügbarer Teammates ab.
9. Sortiere die verbleibenden Schwärme aufsteigend nach Entfernung ihres Mittelpunkts zum Gatter.
10. Gehe die Schwarmliste durch und kopiere solange Schwärme in die zuTreiben-Liste, wie genug Teammates zum Treiben verfügbar sind (unter Entfernung eingeteilter Teammates von der Verfügbarkeitsliste).
11. Berechne für alle Schwärme der zuTreiben-Liste mittels Preprocessing des Spielfelds und A* Algorithmus den möglichst optimalen Weg zum Gatter.
12. Bestimme anhand des ersten Knotens der Weg-Ermittlung mittels eines „geometrischen“ Algorithmus die Koordinaten zur Schwarmsteuerung.
13. Weise die Koordinaten unter Berücksichtigung möglichst kurzer Wege für die Agenten den eingeteilten Teammates zu und kopiere diese Daten in die zugehörigen Schwärme der zuTreiben-Liste.
14. Speichere die zuTreiben-Liste in der Beliefbase.
15. Versende eine Planungsvollzugsmeldung an die Teammate Agenten.

Es stechen vor allem das Zusammenführen alter und neuer „Kuh-Listen“ unter Konsistenzwahrung, das Preprocessing des Spielfelds und die geometrische Anordnung der Agenten zur Schwarmsteuerung heraus, diese Punkte erfahren wie erwähnt später eine genauere Betrachtung.

5.3 Funktionsweise des Gesamtsystems

Architektur und Feindesign sind aufgestellt, doch soll an dieser Stelle noch einmal ein besserer Gesamtüberblick über das System gegeben werden. Dazu werden im Folgenden anhand des Sequenzdiagrammes in Abbildung 5.6 auf Seite 47 die Interaktionen der einzelnen Akteure und die Funktionsweise des Gesamtsystems kurz erklärt.

Exemplarischer Ablauf einer Simulation (inkl. Simulationsstart, einer Runde und Simulationsende):

1. Die Teammate Agenten versuchen direkt nach ihrer Instanzierung eine Verbindung zum Contest-Server aufzunehmen. Dies ist durch eine Capability realisiert, die in jedem dieser Agenten implementiert ist und die die gesamte Client-Server Kommunikation in den Teammate Agenten handhabt. Dabei kapselt sie die Kommunikation, der Agent selbst nutzt dann nur die von der Capability vorgenommen Änderungen in der Beliefbase und z.B. das von der Capability angebotene "Move"Goal plus Richtung, die Capability erzeugt darauf ein XML-Paket im Rahmen des Protokolls und versendet dieses. (s. Abbildung 5.6: A)

2. Mit dem Simulationsstart wird vom Contest-Server eine Sim-Start-Message an die Teammate Agenten verschickt. Diese enthält initiale Werte wie z.B. die maximale Rundenzahl, die Grid-Größe, etc. (s. Abbildung 5.6: B)

3. Die angesprochene Capability speichert diese Werte in der Beliefbase des Teammate Agenten und versendet über die von Jadex angebotene FIPA konforme ACL eine *inform* Nachricht mit den Werten an den Herding Officer, der diese Werte ebenfalls in seiner Beliefbase speichert. (s. Abbildung 5.6: C)

4. Der Contest-Server sendet nun am Anfang jeder Runde Request-Action-Messages an die Teammate Agenten, die die Vision des jeweiligen Agenten und Informationen zum Spielstatus enthält, sowie die Deadline für den 2-10 sekündigen Timeout, nach dem keine Antwort mehr vom Server akzeptiert wird. (s. Abbildung 5.6: D)

5. Die empfangenen Daten werden ebenfalls von der Capability in der Beliefbase gespeichert und eine entsprechende *inform* Nachricht an den Herding Officer Agenten verschickt. Das Goal „Cow Herding“ wird aktiv. (s. Abbildung 5.6: E)

6. Nachdem von allen sechs Teammate Agenten die „sendPerceptionInform“ Nachrichten beim Herding Officer angekommen sind, werden die Daten miteinander abgeglichen, in der Beliefbase gespeichert, und ein Plan wird für die aktuelle Runde erstellt, der ebenfalls in der Beliefbase abgespeichert wird. Sobald der Plan erstellt wurde, wird eine *inform* Nachricht an die Teammate Agenten versandt, der diese über den zur Verfügung stehenden Plan informiert. (s. Abbildung 5.6: F)

7. Im Rahmen des zur Erfüllung des „Cow Herding“ Goals ausgeführten Plans wird eine *request* Nachricht vom Teammate Agenten an den Herding Officer Agenten verschickt. Wenn der anfragende Agent bei der Planung zum Treiben eines Schwarms eingeteilt worden ist, erhält er ein gültiges Koordinantenobjekt als Antwort. Ist dies nicht der Fall, erhält er ein Koordinatenobjekt mit dem Wert „NULL“ zurück. (s. Abbildung 5.6: G)

8. Wenn eine gültige Koordinate empfangen wird, plant der Teammate Agent den Weg dorthin mittels A* und übersendet die berechnete Richtung mittels oben beschriebener Instanzierung des „Move“ Goals an den Contest-Server. Wird keine gültige Koordinate empfangen, wird das Sub-Goal „Explore Area“ instanziert, was in diesem Beispiel der Fall ist. (s. Abbildung 5.6: H)

9. Der vom Sub-Goal „Explore Area“ ausgewählte Plan überprüft zuerst in der Beliefbase des Teammate Agenten, ob dieser bereits in vorherigen Runden die Exploration eines Gebietes vorgenommen und noch nicht vollständig abgeschlossen hat. Ist dies der Fall, wird an der Stelle weitergemacht, an der vorher aufgehört wurde und per A* die nächste Richtung bestimmt, die dann per „Move“ Sub-Goal an den Contest-Server gesendet wird. Damit ist die Runde für den spezifischen Agenten vorbei. Liegt noch keine angefangene Exploration in der Beliefbase vor, erfragt der Teammate Agent per *request* Nachricht einen zu erkundenden Bereich vom Herding Officer Agenten. Dieser übermittelt dem anfragenden Agenten darauf einen längere Zeit nicht zur Exploration vergebenen Bereich des Spielfelds. Mögliche Bereiche sind „nordwest“, „nordost“, „südwest“ oder „südost“. (s. Abbildung 5.6: I)

10. Sobald der Teammate Agent den Bereich empfangen hat, wird noch im selben Plan die Exploration geplant und in der Beliefbase gespeichert. Das weitere Vorgehen ist dasselbe, als hätte er schon einen Plan zur Exploration in der Beliefbase vorgefunden, wie oben beschrieben. (s. Abbildung 5.6: J)

11. Dieser Ablauf wiederholt sich ab Punkt 4 so lange, bis die vorher festgelegte Anzahl an Runden erreicht ist und der Contest-Server eine Sim-End-Message an die Teammate Agenten verschickt. Diese bringen darauf ihre Beliefbase zurück in den Anfangszustand, versenden eine *inform* Nachricht an den Herding Officer Agenten, der daraufhin ebenfalls seine Beliefbase in den Anfangszustand bringt, und warten auf die nächste Sim-Start-Message. (s. Abbildung 5.6: K)

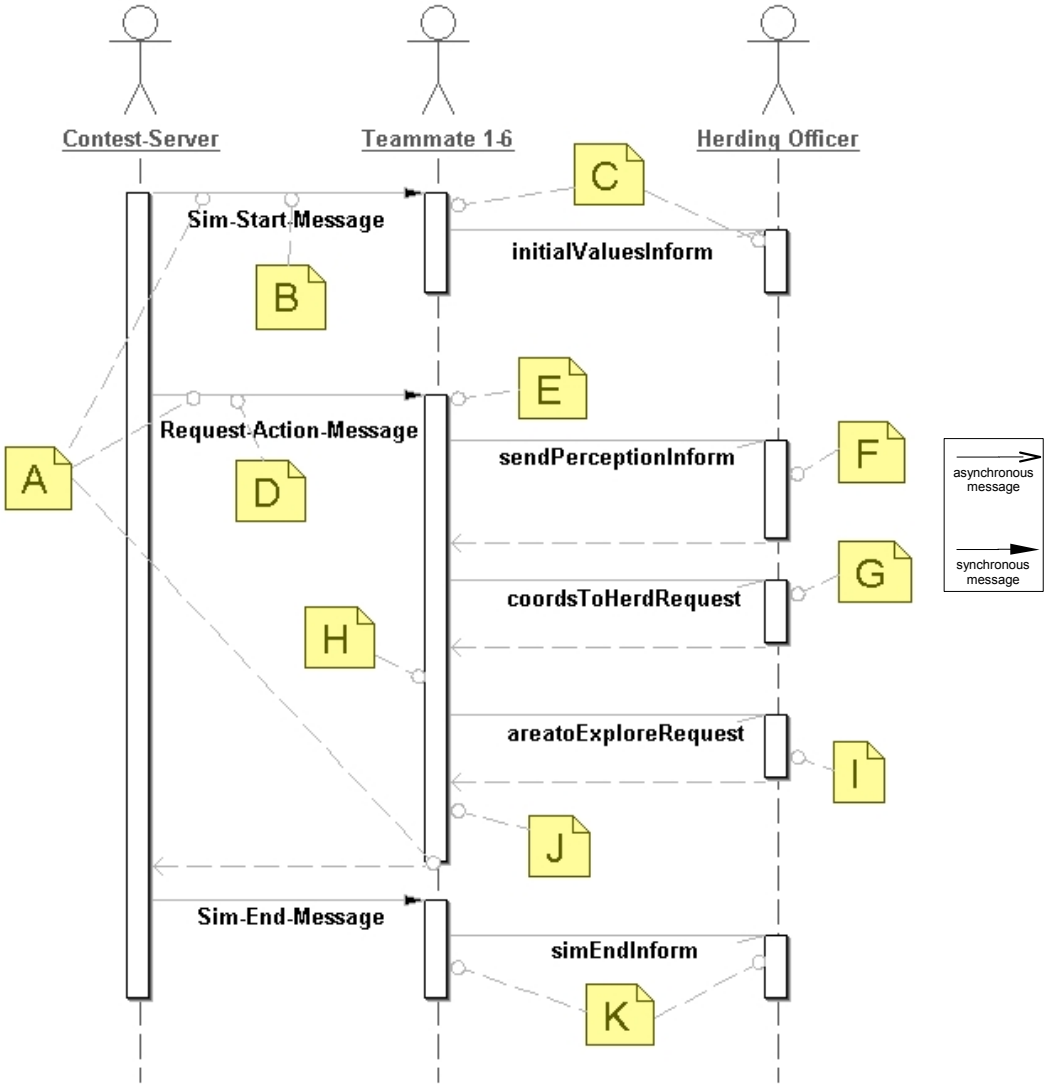

Abbildung 5.6: Architektur des Prototypen. Sequenzdiagramm zur Verdeutlichung der Interaktionen zwischen den Akteuren und des Ablaufs einer Simulation.

5.4 Fazit

Die Architektur des Multiagentensystems sowie das Feindesign der identifizierten Agententypen sind erstellt. Auch die Funktionsweise des Systems sollte jetzt verständlich geworden sein. Anhand dieser detaillierten Vorgaben wird der Prototyp umgesetzt. Die als *nice-to-have* identifizierten Anforderungen bezüglich des „Störens des Gegners“ und des „Spieltheoretischen Ansatzes“ haben es nicht ins endgültige Design geschafft, da der enge Zeitrahmen deren Umsetzung nicht ermöglicht hätte.

Die Tropos-Methodik hat wie erwartet gute Modellierungsmöglichkeiten für die Architektur geboten, schon im Feindesign ist die Methodik allerdings an ihre Grenzen geraten. Es gibt keine Modellierungsmöglichkeit für von Plänen instanzierte Sub-Goals, Beliefs und Beliefbase lassen sich nur notdürftig modellieren und die vorgeschlagene Modellierungsart für Capabilities scheint ungeeignet für die Art Capabilities, die Jadex bietet, und wird deshalb nicht genutzt. Bei anderen betrachteten Methodiken sehen genannte Punkte allerdings ähnlich aus. Hier scheint noch Forschungsbedarf zu bestehen, um eine völlig plattformunabhängige Methodik zu entwickeln.

6 Ausgewählte Aspekte der Realisierung

Die im vorigen Kapitel vorgestellte Architektur und das Design reflektieren die tatsächliche Realisierung des im Contest genutzten Prototypen. Aus diesem Grund wird hier darauf verzichtet, die Realisierung des Gesamtsystems vorzustellen. Stattdessen werden nur die Punkte betrachtet, die im vorigen Kapitel aufgrund zu hoher Komplexität nur angerissen worden sind und deren Realisierung nicht trivial ist. Weiterhin stellt die Realisierung dieser Punkte den Hauptaspekt bzw. die essentiellsten Faktoren des Hauptaspekts dieser Arbeit dar, die Schwarmsteuerung.

Diese Punkte erfahren eine nähere Betrachtung:

1. Zusammenführung der von den einzelnen Teammate Agenten zur Verfügung gestellten „Kuh-Listen“
2. Preprocessing des Spielfelds zur Nutzung des A* Algorithmus für die Schwarmsteuerung
3. Geometrische Anordnung der Teammates zur Schwarmsteuerung

Die Punkte stammen alle aus dem im Herding Officer implementierten Plan „Evaluate Cowlists and Plan Swarm Steering“ und werden in den folgenden drei Unterkapiteln jeweils einzeln aufgegriffen. Zur Verdeutlichung des 1. Punktes wird ein Pseudocodefragment herangezogen, für die Punkte 2 und 3 wird darauf verzichtet. Bei diesen Punkten steht die Idee im Vordergrund und die Funktionsweise wird anhand von Grafiken und textueller Beschreibung erklärt. Die Implementation selbst ist trivial und kann gegebenenfalls im Sourcecode des Projektes angeschaut werden.

Der Pseudocode oder auch Code des A* Algorithmus wird hier nicht weiter veranschaulicht, dieser ist gut dokumentiert, und in jeglichen Programmiersprachen sind Implementationen vorhanden [Russel und Norvig (2003); Patel].

6.1 Zusammenführung der von den einzelnen Teammate Agenten zur Verfügung gestellten „Kuh-Listen“

Der Herding Officer Agent erhält wie beschrieben nach Rundenbeginn immer von jedem der Teammate Agenten dessen aktuelle Vision. Diese ist in einer Java™-Klasse gekapselt, die Listen der verschiedenen Objekte in der Vision des Agenten enthält. Diese Objekte sind wiederum in spezifischen Java™-Klassen gespeichert, die (bei Kühen) den eindeutigen Bezeichner, die Koordinaten des Objekts auf dem Spielfeld und je nach Objekt Zusatzinformationen (z.B. Zeitpunkt der letzten Sichtung) enthalten. Zusätzlich zur Liste der aktuell gesehenen Kühe bietet der Teammate Agent aber noch eine weitere Liste an. Dieses ist die Liste der Kühe, die in der Vorrunde gesehen wurden, aber in der aktuellen Runde durch die eigene Bewegung des Teammate Agenten nicht mehr sichtbar sind. Im Folgenden wird diese Liste „vermutete Kuhpositionen“ Liste genannt.

Jetzt sollen aus diesen insgesamt zwölf Listen zwei neue konsistente Listen erzeugt werden, zum einen die Liste aller aktuell gesehener Kühe, zum anderen die Liste aller vermuteter Kuhpositionen.

Die Listen der aktuell gesehenen Kühe können dabei problemlos zusammengeführt werden. Bei den Listen der vermuteten Kuhpositionen muss allerdings zuerst geschaut werden, ob eine *vermuteten* Kuh nicht aktuell gesehen wird, dann fällt diese Weg. Ferner soll nur die aktuellste vermutete Position einer Kuh in die Gesamtliste aufgenommen werden, dazu wird jeweils die Zusatzinformation „Zeitpunkt der letzten Sichtung“ verglichen. Zuletzt werden alle vermuteten Kuhpositionen, die sich in Bereich der aktuellen Vision der Teammate Agenten befinden entfernt.

Abbildung 6.1 auf Seite 51 zeigt mittels Pseudocode, wie dieser Algorithmus im Prototypen implementiert ist.

```
beziehe alle benötigten, aktualisierten Werte aus der Beliefbase;
für alle Listen aktuell gesehener Kühe (agK){
        für alle in der Liste agK enthaltenen Kuhobjekte{
                wenn nicht bereits in der Gesamtliste agK vorhanden{
                        füge Kuhobjekt der Gesamtliste agK zu;
                        }
                }
        }
für alle Listen vermuteter Kuhpositionen (vK){
        für alle in der Liste vK enthaltenen Kuhobjekte{
                wenn nicht bereits in der Gesamtliste agK vorhanden{
                        wenn nicht bereits in der Gesamtliste vK
                           vorhanden{
                                füge Kuhobjekt der Gesamtliste vK zu;
                                }
                        sonst{
                                wenn Zeitpunkt letzter Sichtung aktueller
                                   {
                                        ersetze vorhandenes Kuhobjekt mit
                                           neuem;
                                        }
                                }
                        }
                }
        }
für alle Teammate Agenten Objekte{
        bilde ein der Vision entsprechendes Rechteck um die Koordinate
           des Objekts;
        für alle in der Gesamtliste vK enthaltenen Kuhobjekte{
                wenn Koordinate des Kuhobjekts in Rechteck enthalten{
                        lösche Kuhobjekt aus der Gesamtliste vK;
                        }
                }
        }
```

Abbildung 6.1: Pseudocodefragment, das die Implementation der Zusammenführung der von den einzelnen Teammate Agenten zur Verfügung gestellten „Kuh-Listen“ veranschaulicht.

6.2 Preprocessing des Spielfelds zur Nutzung des A* Algorithmus für die Schwarmsteuerung

Schwärme sollen nicht dicht an Hindernissen vorbeigeführt werden, damit einzelne Kühe sich nicht eventuell abspalten. Ebenso sollen Engpässe nach Möglichkeit vermieden bzw. wenn nicht anders möglich mittig passiert werden. Gegnerische Gatter sollen ebenfalls vermieden werden, genauso wie gegnerische Agenten. Zusätzlich sollen Schwärme nicht dicht an anderen Schwärmen vorbeigeführt werden, da der serverseitig implementierte Kuh-Algorithmus bei zu großen Herden eine Art *Auskristallisierung* bewirkt, so dass es nahezu unmöglich wird diese zu steuern.

Gelöst wird diese Problematik in der Implementation des Prototypen durch ein Preprocessing des Spielfelds vor der Nutzung des A* Algorithmus zur Wegplanung. Da der A* Algorithmus mit Kosten arbeitet, können also Gebiete mit höheren Kosten vermieden werden. Also werden beim Preprocessing des Spielfelds die benachbarten Zellen aller oben genannter Objekttypen mit höheren Kosten versehen (gegnerische Gatter selbst werden als *solid*, also unpassierbar, markiert), so dass um diese Objekte herum eine Pufferzone aus höheren Kosten entsteht, die der A* Algorithmus dann soweit möglich umgeht (s. Abbildung 6.2). Dieses Verfahren löst alle oben beschriebenen Teilprobleme gleichzeitig.

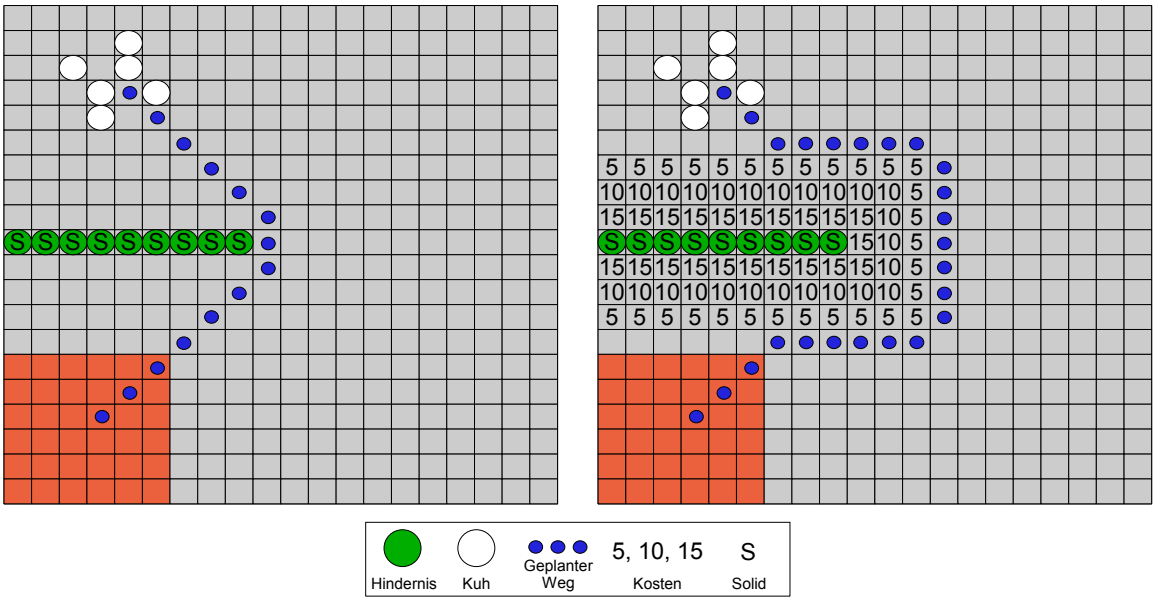

Abbildung 6.2: Preprocessing des Spielfelds. Links: generierter Pfad ohne Preprocessing. Rechts: generierter Pfad nach Preprocessing

6.3 Geometrische Anordnung der Teammate Agenten zur Schwarmsteuerung

Eine der zentralen Fragen zur Realisierung der Schwarmsteuerung ist die der Positionierung der einem Schwarm zugeordneten Teammate Agenten, um den Schwarm möglichst zielgerichtet entlang des vorher ermittelten Weges zu steuern.

Der dafür entwickelte und implementierte Algorithmus wird nachfolgend anhand Abbildung 6.3 auf Seite 54 erläutert. Der erste Schritt ist es den aus Schwarmmittelpunkt und erstem Knoten des A* Pfades gebildeten Vektor umzukehren. Daraufhin wird eine zu diesem Vektor senkrechte durch den Schwarmmittelpunkt verlaufende Hilfsgerade erzeugt und solange entlang des Richtungsvektors verschoben, bis alle Koordinaten der dem Schwarm zugehörigen Kühe auf einer Seite der Geraden liegen (s. Abbildung 6.3: a). Der Schnittpunkt des Vektors (unter Hinzuziehen des Schwarmmittelpunkts ebenfalls eine Gerade) mit der verschobenen Hilfsgeraden ergibt die zentrale Treibeposition. Im nächsten Schritt wird aus dem Richtungsvektor und dem Schwarmmittelpunkt die nächste Hilfsgerade gebildet. Diese wird entlang eines neu ermittelten, zu der Hilfsgerade senkrechten Richtungsvektors wieder solange verschoben, bis alle Kühe auf einer Seite dieser Geraden sind (s. Abbildung 6.3: b). Aus der zentralen Treibeposition und dem aktuellen Richtungsvektor wird wieder eine Gerade gebildet. Deren Schnittpunkt mit der aktuellen Hilfsgeraden ergibt die linke Treibeposition. Der Vorgang wird mit der umgekehrten Variante des zuletzt genutzten Richtungsvektors wiederholt und die rechte Treibeposition bestimmt (s. 6.3: c). Damit sind für alle Schwärme, die aus dreizehn oder weniger Kühen bestehen, die Treibepositionen festgelegt (s. Abbildung 6.3: d). Besteht ein Schwarm aus mehr als dreizehn Kühen, werden die linke und die rechte Treibeposition um ein bis zwei Zellen (ab sechzehn Kühen zwei Zellen) in Richtung des aus Schwarmmittelpunkt und erstem Knoten des A* Pfades gebildeten Vektors verschoben (s. Abbildung 6.3: e + f).

Bei einer Schwarmgröße von eins wird dann nachfolgend nur die zentrale Treibeposition genutzt, bei Schwarmgrößen kleiner elf die linke und die rechte Treibeposition und bei allen Schwärmen mit einer Anzahl von elf oder mehr Kühen werden alle drei Treibepositionen besetzt.

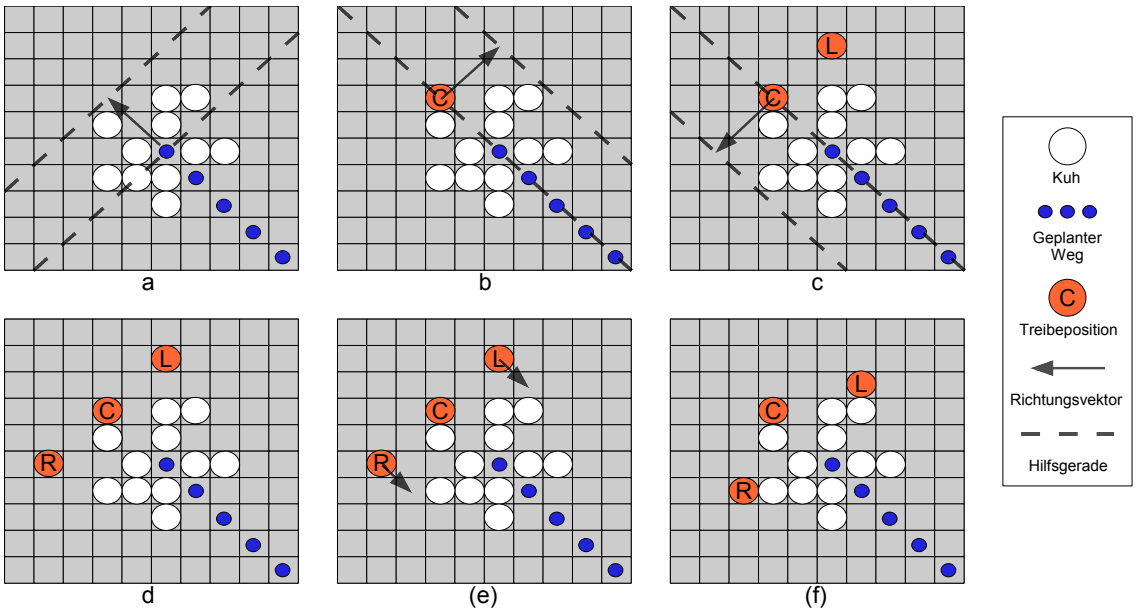

Abbildung 6.3: Ablauf der geometrischen Treibeposition-Bestimmung. Die Punkte e und f werden erst bei Schwärmen > 13 Kühen genutzt.

7 Test & Bewertung

Das Testen und Verifizieren von Multiagentensystemen ist ein zur Zeit stark diskutiertes Thema, gibt man in einer Suchmaschine die zwei Stichwörter *Multiagentensystem* und *test* ein, wird man überflutet mit einer Vielzahl von Papieren über Test-Frameworks für Multiagentensysteme. Diese sind jedoch in der Mehrzahl speziell für bestimmte Plattformen bzw. Problemstellungen entwickelt. Auch Ansätze für flexible Frameworks zur Verifikation sind zu finden [Dennis u. a. (2008)].

Gemein haben diese Frameworks jedoch alle eines: Sie brauchen eine große Einarbeitungszeit und auch Anpassung an das eigene Multiagentensystem. Daher wurde unter Berücksichtigung des Zeitfaktors auf den Einsatz eines solchen Frameworks verzichtet.

Trotzdem sind Tests natürlich unverzichtbar und wurden fortwährend in die Entwicklung des Prototypen eingebunden. Auf welche Art und Weise das während der Entwicklung vorgenommen wurde, wird unter 7.1 „Tests während der Entwicklung“ beschrieben.

Weiterhin ist natürlich interessant, wie die Funktionalität des Prototypen hinsichtlich der aufgestellten Anforderungen schließlich zu bewerten ist. Dafür bietet sich hier natürlich der Contest an, bzw. wie sich der Prototyp im Contest geschlagen hat. Diese Betrachtung findet unter 7.2 „Bewertung des Prototypen während des Contests“ statt.

Ebenfalls betrachtet wird der Prototyp bezüglich des Aufbaus hinsichtlich aktueller Multiagentensystemstandards und der Nutzung von BDI-Features unter 7.3.

Schließlich wird unter 7.4 ein Fazit präsentiert, bei dem auch das Ergebnis des Wettbewerbs, bzw. das Abschneiden des eigenen Prototypen preisgegeben wird.

7.1 Tests während der Entwicklung

Wie im Vorfeld erwähnt, sind Tests auch in einem Multiagentensystem unerlässlich, wobei die Art und Weise etwas anders ausfällt als bei *normalen* Softwaresystemen, da Multiagentensysteme im allgemeinen dynamisch und nicht-deterministisch sind.

Da Pläne in Jadex mit Java™ realisiert werden, lassen sich natürlich einzelne Fragmente des Codes auch White- und Black-Box-Tests unterziehen, wobei hier jedoch auf White-Box-Tests verzichtet wird[1] und Black-Box-Tests auch nur z.B. beim A* Algorithmus genutzt werden.

Der eigentlich wirklich interessante Teil der Tests in diesem Prototypen ist allerdings anderer Art. Hier kommt wieder die in Kapitel 3.2.1 „Graphische Aufbereitung des aktuellen Simulationsgeschehens“ formulierte Anforderung ins Spiel. Denn die Visualisierung der Simulation ist für Debugging und Testen des Gesamtsystems das Mittel der Wahl. Führt man sich vor Augen, dass es für jegliche Softwaresysteme nur die Möglichkeit des Beweises der partiellen Korrektheit gibt, wird dies für das hier vorliegende Multiagentensystem noch schwieriger. Da es nämlich nicht möglich ist Test-Fälle für den Prototypen zu erzeugen, abgesehen von verschiedenen Spielfeldern,[2] bleibt als einzige Alternative übrig solche Test-Fälle zu beschreiben und im Rahmen einer Simulation auf das Eintreten dieser zu warten.

Das zwingt dem Entwickler folgendes Vorgehensmodell auf:

1. Revisioniertes Arbeiten an vornehmlich einem Feature zur Zeit.
2. Eine Liste mit Test-Fällen erzeugen, die das Feature abdecken.
3. Sobald eine Veränderung vorgenommen wurde, direkt im Hintergrund oder auf einem anderen System den Prototypen laufen lassen.
4. Den Prototypen solange laufen lassen, bis die Liste mit den Test-Fällen anhand von Beobachtung der GUI bzw. diverser von Jadex mitgebrachter Werkzeuge[3] komplett abgehakt werden konnte.
5. Beim Auftreten eines Fehlers in einem Test-Fall den Fehler im Code suchen, eine Veränderung vornehmen und wieder zu Punkt 3 springen.

Das ganze ist sehr zeitintensiv und darum ist die Anforderung an die Visualisierung die, möglichst viel der aktuellen Planung der Agenten und deren innerer Zustände preis zu geben. Dementsprechend werden in der GUI zusätzlich zu allen sich in der aktuellen Vision der Agenten befindlichen Objekten auch die als nächstes von den Agenten geplanten Wege

[1] Die Zahl der Code-Zeilen pro Code-Fragment sind überschaubar genug.
[2] Unterschiedliche Größe, verschiedene Anordnung der Hindernisse, verschiedene Hindernisdichten, etc.
[3] Haupttool ist hier der *Introspector*, mittels dessen man die aktuellen Werte in den Beliefs einzelner Agenten betrachten kann.

dargestellt, sowie die vom *Herding Officer* vorgenommene Wegplanung für die Schwarmsteuerung, wie in Abbildung 7.1 zu sehen ist.

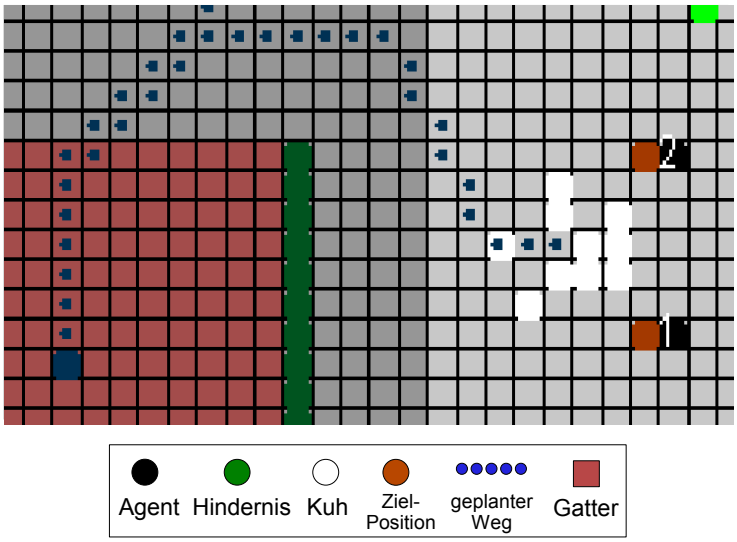

Abbildung 7.1: Ausschnitt der Prototyp-GUI.

Nicht zu vernachlässigen ist natürlich, dass man für diese Art von Test eine Testumgebung braucht. Diese wurde vom Wettbewerbsausrichter (leider zu einem recht späten Zeitpunkt) in Form eines lokal installierbaren Testservers geliefert. Dieser Testserver entsprach dem, der im Contest genutzt wurde, die im Contest genutzten Spielfelder waren allerdings nicht enthalten. Bevor dieser Testserver zur Verfügung stand, wurde ein selbst entwickelter Serverprototyp genutzt, der anhand der ersten (noch sehr unvollständigen) Revision der Szenariobeschreibung umgesetzt wurde und hauptsächlich zur generellen Verdeutlichung des Szenarios und der Ideenfindung zur Planung der Schwarmsteuerung diente. Für die tatsächlichen Tests wurde dann der bereitgestellte Testserver genutzt.

Nachdem die Art der Tests theoretisch abgehandelt wurde, soll jetzt anhand eines Beispiels aus der Entwicklung das ganze noch einmal verdeutlicht werden:

1. Folgendes Feature soll implementiert werden:
    - Schwärme sollen möglichst nicht zu nah aneinander gesteuert werden, um zu verhindern, dass sich ein großer eventuell nicht mehr steuerbarer Schwarm bildet.

2. Zugehörige Testfälle:
    - Zwei momentan gesteuerte Schwärme befinden sich in einer Position, bei der die optimal geplanten Wege kollidieren würden.
    - Zwei momentan gesteuerte Schwärme befinden sich in einer Position, bei der die optimal geplanten Wege kollidieren würden, allerdings gibt es keinen anderen möglichen Weg.
    - Mehr als zwei momentan gesteuerte Schwärme befinden sich in einer Position, bei der die optimal geplanten Wege kollidieren würden.
    - Mehr als zwei momentan gesteuerte Schwärme befinden sich in einer Position, bei der die optimal geplanten Wege kollidieren würden, allerdings gibt es keinen anderen möglichen Weg.
3. Prototyp wird gestartet...
4. ...und solange beobachtet, bis alle Testfälle eingetreten sind. Ist ein Fehler dabei aufgetreten, muss dieser behoben und wieder bei Punkt 3 angefangen werden.

Diese Methode des Testens ist nicht optimal, zum einen weil möglicherweise wichtige Testfälle nicht bedacht wurden, zum anderen, da gewisse Testfälle nie erreichbar sind. Ein Beispiel hierfür ist der Gegner in einer Simulation. Zu diesem Zweck wurde zwar zum Testen eine zweite Instanz des Prototypen gestartet, um überhaupt einen agierenden Gegner zu haben, allerdings deckt dies natürlich nicht die möglicherweise völlig anderen Strategien und Verhaltensweisen eines tatsächlichen Gegners ab.

Eine große Erleichterung für das Testen wäre eine Testserverumgebung gewesen, mit der sich gewisse Simulationszustände speichern bzw. reproduzieren ließen. Dies war aber nicht möglich, da zum einen der Source-Code des Testservers nicht offen lag und zum anderen der Zeitaufwand zu hoch gewesen wäre.

Abschließend ist zusammenzufassen, dass intensives Testen eine erfolgreiche Implementation des Prototypen zwar überhaupt erst ermöglicht hat, jedoch der endgültige Test und die Bewertung erst im Contest selbst vorgenommen werden kann, beschrieben im folgenden Kapitel 7.2 „Bewertung des Prototypen während des Contests“.

7.2 Bewertung des Prototypen während des Contests

Der Contest ist der zentrale Punkt, an dem die Bewertung des Prototypen hinsichtlich der Anforderungen und im Vergleich zu den anderen teilnehmenden Teams erfolgt. Erst hier sind alle Punkte gegeben, die eine Bewertung bezüglich des Gesamtszenarios ermöglichen:

- unbekannte Spielfelder
- unbekannte Gegenspieler
- vor einer Simulation unbekannte Rundenzahl

Zuerst wird in 7.2.1 „Ablauf des Contests“ ein kurzer Überblick den Ablauf gegeben, um diesen zu verdeutlichen. Darauf wird die Bewertung in zwei Kapitel getrennt betrachtet, namentlich 7.2.2 „Bewertung bezüglich der Anforderungen“ und 7.2.3 „Bewertung bezüglich der Implementationen anderer Teams“, da beide Thematiken unterschiedliche Aspekte abdecken und jeweils für sich beleuchtet werden sollten.

7.2.1 Ablauf des Contests

Am Contest haben insgesamt sieben Teams teilgenommen, wobei im Laufe der vier Tage jedes Team jeweils drei Simulationen gegen jedes andere Team bestreiten musste. Diesen drei Simulationen lag jeweils ein anderes, vor dem Contest unbekanntes Spielfeld zugrunde, wobei sich die Spielfelder dann von Begegnung zu Begegnung wiederholten. Jede einzelne Simulation wurde mit drei Punkten für einen Sieg, einem Punkt für ein Unentschieden oder keinem Punkt für eine Niederlage gewertet. Das Team, das am Ende aller Simulationen die meisten Punkte erzielt hatte, war auch der Gesamtsieger, wobei bei einem Gleichstand auch der direkte Vergleich bzw. die Anzahl eingefangener Kühe herangezogen worden ist.

7.2.2 Bewertung bezüglich der Anforderungen

Zuerst wird hier die Erfüllung der technischen Anforderungen betrachtet. Gegliedert werden diese wie schon in Kapitel 3.2 „Technische Anforderungen“, die erste Darstellung erfolgt in Tabelle 7.1 auf Seite 60, wobei alles, was mit *tadellos* bewertet wird, keiner genaueren Inspektion unterzogen wird.

Kleinere Probleme mit der Stabilität erfolgten auf einem der im Contest genutzten Spielfelder. In unregelmäßigen Abständen stürzte einer der sechs Agenten ab und musste vom Observer-Agenten neu gestartet werden. Der Fehler konnte während des laufenden Contests nicht gefunden werden, allerdings erfolgte der Neustart des jeweiligen Agenten immer so schnell, dass es keine Auswirkungen auf die Arbeit des Gesamtsystems hatte.

technische Anforderung	Bewertung
Kommunikation mit dem Contestserver	tadellos
Kommunikation zwischen den Agenten	tadellos
Graphische Aufbereitung des aktuellen Simulationsgeschehens	tadellos
Reaktionszeit des Systems	tadellos
Stabilität	kleinere Probleme

Tabelle 7.1: technische Anforderungen: Bewertung

Als nächstes werden in Tabelle 7.2 die fachlichen Anforderungen betrachtet, wobei hier allerdings jeder Punkt nachfolgend noch genauer beschrieben wird.

fachliche Anforderung	Bewertung
Navigation (Pathfinding) der Agenten auf dem Spielfeld	tadellos
Erkundung des Spielfelds	zu wenig forciert
Steuerung von einzelnen Kühen bzw. Schwärmen	kleinere Probleme
Stören des Gegners	nicht implementiert
Spieltheoretischer Ansatz	nicht implementiert

Tabelle 7.2: fachliche Anforderungen: Bewertung

Die Navigation der Agenten auf dem Spielfeld mit Hilfe des A* Algorithmus funktionierte problemfrei, Auffälligkeiten gab es keine. Keine Karte machte hier Probleme.

Zur Erkundung des Spielfelds sind einige Punkte offen. Die Erkundung des Spielfelds an sich funktionierte, die gesammelten Daten wurden dem *Herding Officer* problemlos kommuniziert. Allerdings hätte die Erkundung besonders in der Anfangsphase der Simulationen stärker fokussiert werden müssen. Eine schnellstmögliche komplette Erkundung des Spielfelds hätte eines der zwei im folgenden Punkt behandelten Probleme bei der Steuerung von Schwärmen vermieden.

Die Schwarmsteuerung an sich funktionierte so wie gewünscht, allerdings traten wie oben beschrieben Probleme in der Anfangsphase von Simulationen auf. Dadurch, dass das Spielfeld noch nicht gänzlich erkundet war, konnten mit dem A* Algorithmus keine optimale Wege geplant werden, sodass Herden teilweise durch die daraus bedingten Fehlannahmen zuerst in falsche Richtungen getrieben wurden. Eine von Anfang an forcierte Erkundung des gesamten Spielfelds in Kombination mit besonders hohen Kosten für noch nicht erkundete Zellen im Zusammenspiel mit dem A* Algorithmus könnten eine Lösung dieser Problematik darstellen. Weiterhin wurde die Auswahl des nächsten zu treibenden Schwarms über einen *greedy* Algorithmus unter Hinzunahme der Euklidischen Distanz getroffen. Dies hatte zur

Folge, dass auf Spielfeldern mit sehr großen, aus zusammenhängenden Hindernissen gebildeten Strukturen solche Schwärme als erste ausgewählt wurden, die zwar laut Euklidischer Distanz sehr nah am eigenen Gatter lagen, der eigentliche durch die Hindernisse bestimmte Weg zum Gatter aber sehr viel größer war als der von anderen Schwärmen. Dies hätte schon im Vorfeld herausgefunden werden können, hätte man ähnliche Spielfelder zum Testen erzeugt.[4] Eine Lösung dieser Problematik wäre es, auch für die Auswahl des nächsten Schwarms anstelle der schnell berechneten Euklidischen Distanz, die Distanz unter Zuhilfenahme des A* Algorithmus zu bestimmen.

Der Aspekt des *Störens des Gegners* wurde nicht mehr implementiert und erfährt eine genauere Betrachtung unter 7.2.3 „Bewertung bezüglich der Implementationen anderer Teams“.

Die letzte Anforderung, der *Spieltheoretische Ansatz*, die sowohl aus Zeitgründen als auch aus dem Grund der zu geringen Relevanz[5] nicht mehr implementiert wurde, hat auch tatsächlich, soweit man das aus den Beobachtungen schließen kann, keine große Rolle gespielt. Um diese Rolle zu vergrößern, hätte der ganze Strategieansatz stärker der Spieltheorie unterliegen müssen.

7.2.3 Bewertung bezüglich der Implementationen anderer Teams

Die Gegner in den Simulationen spielten eine nicht zu vernachlässigende Rolle, interessant bei deren Betrachtung ist zum einen, wie die eigenen Agenten mit diesen umgegangen sind und zum anderen welche Strategien von diesen benutzt worden sind. Außerdem werden noch eventuelle Auffälligkeiten der von anderen Teams benutzten Multiagentenplattformen betrachtet.

Umgang der eigenen Agenten mit den gegnerischen Agenten

Da keine aktive Störung des Gegners als Strategie implementiert wurde, ist es interessant wie mit den gegnerischen Agenten passiv umgegangen wurde. Hierzu wurden diese und die Zellen um sie herum wie in Kapitel 6.2 „Preprocessing des Spielfelds zur Nutzung des A* Algorithmus für die Schwarmsteuerung“ beschrieben im Treibealgorithmus als nicht passierbar bzw. mit erhöhten Kosten für den A* Algorithmus berücksichtigt, so dass versucht wurde Schwärme um gegnerische Agenten herum zu treiben. Dies sollte als *nicht destruktive* Strategie vermeiden, dass Pattsituationen entstanden, wie z.B. durch reines Blocken.

[4] Die Erzeugung von Spielfeldern ist sehr aufwendig, weshalb nur wenige Spielfelder zu Testzwecken erzeugt werden konnten.

[5] Die Gesamtstrategie des Treibens war generell *greedy* aufgebaut, was diese Anforderung weniger relevant machte.

Für Schwärme die aus mehr als einer Kuh bestanden hat diese Strategie funktioniert, da in solchen Fällen mindestens zwei eigene Agenten versucht haben einen Schwarm zu treiben, alle Gegner nur mit reinen Blockstrategien gearbeitet haben und dadurch der Großteil eines Schwarms dem Gegner „entführt“ werden konnte, ohne dass eine Pattsituation auftrat. In eins zu eins Situationen, bei denen ein eigener Agent, eine Kuh und ein gegnerischer Agent involviert waren, ging die Strategie nicht auf, hier konnte das Treiben der Kuh durch den Gegner nur verzögert, nicht aber verhindert werden. Um dies zu verbessern, wäre es z.B. möglich die genutzte Strategie nur für Schwärme mit mehr als einer Kuh zu benutzen und bei einzelnen Kühe auf direktes Blocken zu wechseln.[6]

Gegnerische Strategien im direkten Vergleich mit den eigenen

Die meisten anderen Teams nutzten nur jeweils einen ihrer Agenten zum Treiben eines Schwarms. Dies funktionierte weniger gut als die eigene Strategie, da die Schwärme dadurch aufsplitterten und nicht zusammengehalten werden konnten. Ein wirklich destruktives Blocken des Gegners wurde nur von einem der gegnerischen Teams implementiert und auch nur auf einer Karte eingesetzt. Dort haben deren Agenten einen aus Hindernissen geformten Flaschenhals zwischen den Kühen und dem gegnerischen Gatter identifiziert und darauf zwei eigene Agenten abgestellt, die dort effektiv jegliche Schwarmsteuerung durch den Gegner verhindert haben.

Auffälligkeiten der spezifischen im Contest genutzten Multiagentenplattformen

An dieser Stelle fiel nur die Jason-Plattform[7] bei den Beobachtungen auf, deren Reaktionsgeschwindigkeit im Gegensatz zu allen anderen genutzten Plattformen deutlich langsamer ausfiel. Ebenso bestanden bei dem Team, das die Jason-Plattform genutzt hat, immer wieder Verbindungsprobleme einzelner Agenten, wobei diese Probleme nicht eindeutig auf die Plattform zurückgeführt werden können.[8]

[6] Allerdings entsteht dabei die genannte Pattsituation, es geschieht keine Bewegung seitens der Kuh, erst wenn entweder der Gegner oder der eigene Agent aufgeben, kann diese aufgelöst werden.

[7] zu finden unter: http://jason.sourceforge.net

[8] Diese Probleme könnten z.B. auch an der Netzwerkinfrastruktur gelegen haben.

7.3 Bewertung des Prototypen bezüglich der Nutzung von MAS-Spezifika und BDI-Features

Die Entwicklung moderner Multiagentensysteme versucht immer stärker auf völlige Dezentralisierung zu setzen. Es wird häufig versucht zu zeigen, dass dezentralisierte Systeme bei gleicher Aufgabenstellung äquivalenten aber zentralisiert modellierten Systemen überlegen sind. Diesem Ansatz wird der hier entwickelte Prototyp nicht gerecht, er setzt wie mehrfach erläutert auf ein *zentralisiertes Planen verteilter Pläne*. Auch wenn eine solche Art der Planung nicht unüblich ist, ist es nicht die eleganteste und entspricht nicht dem neuesten Stand der Technik. Allerdings ist die zentralisierte Form des Planens in diesem Szenario äußerst effizient gewesen und deshalb wird dieser Punkt auch nicht als schlecht bewertet.

Weiterhin werden BDI-Features wie z.B. Deliberation von Goals[9] nicht genutzt. Auch werden im Herding Officer Agenten zu viele unterschiedliche Aspekte durch Abarbeitung eines einzigen Plans realisiert, eine stärkere Modularisierung unter Nutzung aller BDI Features wäre wünschenswert.

Der erste der angesprochenen Punkte unterliegt allerdings auch einer sehr früh getroffenen Entscheidung in der technischen Analyse und steht in direktem Zusammenhang mit dem engen Zeitrahmen dem die Entwicklung des Prototypen unterlag. Zwischen Verfügbarkeit der Testumgebung und Beginn des Contests lag der geringe Zeitraum von etwa einem Monat. Die weiteren Punkte hängen ebenfalls mit dem Zeitrahmen zusammen, denn durch die nötige, frühe Erstellung von Prototyp Agenten und deren inkrementeller Entwicklung ist es im Verlauf der Arbeit nicht mehr möglich gewesen, bestimmte Aspekte des Systems umzugestalten ohne dabei die Möglichkeit zu verlieren am Contest teilzunehmen.

[9] Auswahl zwischen konkurrierenden Goals

7.4 Fazit

Das Endergebnis des Contests spricht für sich, das eigene Agententeam konnte den zweiten von sieben Plätzen erringen, mehr Kühe wurden nur vom Siegerteam ins Gatter gesteuert. Dieses spielte mit einer ganz ähnlichen Strategie wie das eigene Team. Im direkten Vergleich konnte beobachtet werden, dass dieses Team einzig die vorher in der Bewertung identifizierten Fehler vermieden hat, es wurde sowohl eine frühe Exploration des Spielfelds vorgenommen, als auch ein direktes Blocken bei einzelnen Kühen. Allerdings wurden anscheinend auch noch entscheidende Modifikationen während des Contests vorgenommen, da deren Team an den letzten zwei Contesttagen sehr viel effizienter agierte als an den ersten beiden. Beim eigenen Prototypen hingegen wurden nur minimale Änderungen an einigen Parametern[10] vorgenommen.

Die getroffenen Designentscheidungen und die gewählte Strategie sind im allgemeinen sehr gut aufgegangen; Punkte, die nicht völlig so funktionierten wie geplant, konnten identifiziert werden und somit ist eine sehr gute Grundlage geschaffen, um an weiteren Wettbewerben mit ähnlichen Szenarien noch erfolgreicher teilzunehmen.

Auch die zur Implementation des Multiagentensystems genutzte Jadex-Plattform ist allen Anforderungen gerecht geworden. Sie bietet zahlreiche Werkzeuge zur Implementation der Agenten und die Möglichkeit der Modularisierung fiel sehr positiv auf. Auch in Hinsicht auf Reaktionsgeschwindigkeit und Stabilität gab es bis auf gelegentliche Agenten-Abstürze auf einer der Spielkarten keine Kritikpunkte.

Gänzlich kritikfrei kann und soll dieses Fazit allerdings nicht bleiben. So war zwar die Entscheidung auf ein *zentrales Planen verteilter Pläne* zu setzen im Hinblick auf die kurze zur Verfügung stehende Zeit und ihre Effizienz die richtige, aber hinsichtlich der Entwicklung moderner MAS nicht die eleganteste. Wünschenswert wäre es für kommende Wettbewerbe noch vermehrt auf die stärkere Nutzung der BDI Aspekte, wie z.B. die Deliberation von Goals, zu setzen genauso wie auf ein *verteiltes Planen verteilter Pläne* um die Autonomität der einzelnen Agenten noch stärker in den Vordergrund zu stellen. Genaueres zu diesen Punkten ist in Kapitel 8.3 „Ausblick“ nachzulesen.

[10] In diesem Fall die Kosten für den A* Algorithmus.

8 Zusammenfassung & Ausblick

Das Ergebnis dieser Arbeit ist in mehrfacher Hinsicht sehr aufschlussreich. So ist der zentrale Aspekt, die agentenbasierte Schwarmsteuerung in einer dynamischen, kompetitiven Umgebung, mit der erarbeiteten Architektur erfolgreich realisiert worden. Interessant ist, dass die Zeit, die für ein Projekt zur Verfügung steht, besonders bei Multiagentensystemen eine große Rolle für die Gesamtarchitektur zu spielen scheint. Ein enger Zeitrahmen zwingt dem Entwickler nahezu die inkrementelle Entwicklung eines Prototypen als Vorgehensweise auf, durch die sich dann gewisse sehr frühe Designentscheidungen, auch wenn sich diese als nicht optimal erweisen, nicht mehr rückgängig machen lassen. Gerade für die Architektur- und Designphase sollte also ausreichend Zeit zur Verfügung stehen, um ein Multiagentensystem auf dem aktuellsten Stand der Technik entwickeln zu können.

Weiterhin ist während der Architektur- und Designphase festgestellt worden, dass sich die zur Verfügung stehenden semi-standardisierten Methodiken zur *Agentenorientierten Softwareentwicklung* nur bedingt nutzen lassen. So ließen sich sowohl die Koordination der Agenten zur Schwarmsteuerung, als auch die dynamischen Rollen nicht ausreichend in der genutzten Tropos-Methodik abbilden. Auch das von Jadex bereitgestellte Modularisierungskonzept ließ sich in dieser Methodik nicht hinreichend darstellen. Hier scheint noch weitere Standardisierungsarbeit nötig zu sein, um mit einer einheitlichen Methodik die Features moderner Multiagentenplattformen abdecken zu können.

Als sehr positiver Punkt bei dieser Arbeit ist der vielfältige Einsatz des A* Algorithmus zu bewerten. Sowohl zur Wegfindung der Teammate Agenten, als auch zur Planung der Wege zur Schwarmsteuerung unter der Nutzung von Preprocessing des Spielfelds hat diese Methode große Effizienz bewiesen. Ebenfalls positiv ist die geometrische Anordnung der Teammate Agenten zur Schwarmsteuerung ausgefallen. Dieses Konzept hat sich im Verlauf des Contests erfolgreich bewiesen. Auch das oben erwähnte Modularisierungskonzept konnte seine Stärke zeigen, eine Capability zur Client-Server Kommunikation wurde erstellt und kann fortan für weitere Agenten genutzt werden.

Negativ bzw. nicht optimal zu bewerten ist die geringe Nutzung der BDI-Aspekte, sowie die sehr zentralisierte Architektur des Multiagentensystems. Beides ist im Wettlauf mit der Zeit vernachlässigt worden und sollte für zukünftige Designs stärker berücksichtigt werden (s. auch 8.3 „Ausblick“).

Insgesamt aber kann gesagt werden, dass die Zielstellung erreicht worden ist; ein lauffähiger Prototyp ist entwickelt worden, der sich mit dem Erreichen des 2. Platzes im Contest auch beweisen konnte. Auch wurde das Konzept der *Agentenorientierten Softwareentwicklung* auf Architektur und Design des Prototypen angewandt und Schwächen darin wurden aufgedeckt. Zuletzt wurde auch gezeigt, dass die Lösung der Aufgabenstellung durch *verteilte Künstliche Intelligenz* im Allgemeinen und durch ein Multiagentensystem, in diesem Fall unter Nutzung der Jadex-Plattform, im Speziellen möglich ist.

Nachfolgend wird der Contest noch einer kurzen Bewertung unterzogen, sowie die noch offenen Punkte diskutiert und ein Ausblick gegeben.

8.1 Bewertung des Contests

Der Contest selbst wird einer kurzen Betrachtung unterzogen, zum einen hinsichtlich der Ziele, die die Ausrichter mit diesem verfolgen, zum anderen hinsichtlich einiger Probleme, die sich im Laufe des Contests ergeben haben.

Hinsichtlich der Zielstellung des Contests, bei der Identifizierung von Schlüsselproblemen zu helfen und eine brauchbare Testumgebung für Multiagentenplattform- und sprachen zu liefern, kann gesagt werden, dass beides erreicht wird. Die Komplexität und Dynamik des Szenarios machen es unabdingbar sich stark mit zentralen Aspekten wie Koordination, Kooperation und Planung auseinanderzusetzen und so konnten im Verlauf dieser Arbeit auch Schlüsselprobleme identifiziert werden, die in den vorherigen Kapiteln besprochen worden sind. Auch lassen sich mithilfe des Contests Reaktionsgeschwindigkeit und Stabilität der eingesetzten Multiagentenplattform bzw. -sprache ausreichend testen. Hier konnten z.B. Probleme bei der Jason-Plattform[1] beobachtet werden. Ebenfalls positiv ist die durch einen Chat vorhandene Möglichkeit des Austauschs zwischen den Teams im laufenden Contest aufgefallen.

Die anfangs angesprochenen Probleme beziehen sich hauptsächlich auf die frühe Phase des Contests. Hier war hinderlich, dass die finalen Beschreibungen des Szenarios und des Client-Server Protokolls, sowie eine Testserverumgebung erst etwa einen Monat vor Contest-Beginn zur Verfügung standen, so dass große Teile der Entwicklung erst zu diesem Zeitpunkt stattfinden konnten. Für zukünftige Contests wäre ein früherer Release vorteilhaft, um eine stärkere Konzentration auf Architektur und Design des Multiagentensystems zu ermöglichen.

Ein Problem hat sich allerdings durch den gesamten Contest gezogen. Dieses ist der serverseitig implementierte Kuh-Algorithmus, der bei Schwärmen bestehend aus mehr als etwa 18

[1] zu finden unter: http://jason.sourceforge.net

Kühen zu einer Art *Auskristallisierung* des Schwarms geführt hat und somit jegliche Steuerungsbemühungen seitens der Agenten verhinderte. Eine Überarbeitung dieses Algorithmus für zukünftige Contests mit ähnlichem Szenario wäre wünschenswert.

Insgesamt überwiegen aber die positiven Seiten des Contests, es konnten viele Erfahrungen hinsichtlich der Entwicklung moderner Multiagentensysteme gesammelt und Probleme in Architektur und Design identifiziert werden.

8.2 Offene Punkte

Die zwei in den Anforderungen als *nice-to-have* klassifizierten Punkte, das *Stören des Gegners* sowie der *Spieltheoretische Ansatz* wurden aus Zeitgründen nicht implementiert. Auch wenn der *Spieltheoretische Ansatz* in den relativ kurzen Simulationsläufen scheinbar keine zentrale Rolle spielt, wäre doch das Einbeziehen der Spieltheorie ein interessanter Aspekt. Eine Eingliederung in das bestehende System wäre auch durchaus denkbar und möglich und wird im folgenden Kapitel noch kurz thematisiert.

8.3 Ausblick

Mit der Erstellung des Prototypen im Rahmen dieser Arbeit wurde eine sehr gute Grundlage für weitere Wettbewerbe mit ähnlichen Szenarien geschaffen. Schlüsselprobleme sind identifiziert worden und Verbesserungsmöglichkeiten thematisiert, sowie eine Capability zur Client-Server Kommunikation erstellt, die für weitere Contests, die auf ähnlicher Kommunikation beruhen,[2] genutzt werden kann.

An dieser Stelle soll nun ein Ausblick gegeben werden, wie das bestehende System modifiziert werden könnte, um sowohl bei weiteren Wettbewerben in ähnlichen Szenarien noch erfolgreicher teilnehmen zu können, als auch den Anforderungen an moderne Multiagentensysteme besser gerecht zu werden.

Eine höhere Autonomität der einzelnen Agenten könnte z.B. erreicht werden durch:

- Umstellung des Systems auf *verteiltes Planen verteilter Pläne*
- Goal-Deliberation

Dabei könnte der Herding Officer Agent etwa immer noch eine Beraterfunktion innehaben, die z.B. bei Koordinationsfragen (z.B. welcher Agent welchen Schwarm treiben soll) Hilfestellung leistet. Die Planung der Schwarmsteuerung würden dann die Teammate Agenten selbst innehaben, die dann die im Rahmen dieser Arbeit entwickelten Algorithmen zur Schwarmsteuerung (Preprocessing des Spielfelds plus Nutzung des A* Algorithmus zur Pfadgenerierung, geometrische Anordnung der Treibepositionen) nutzen könnten. Mit Hilfe von Goal-Deliberation ließen sich dann auch die im vorherigen Kapitel erwähnten noch offenen Punkte einfach in die Teammate Agenten integrieren.

Ebenfalls könnten z.B. die Pläne zur Wegfindung in Capabilities modularisiert werden, um so auch in anderen Multiagentensystemen eingesetzt werden zu können.

Als letzter Punkt soll an dieser Stelle noch auf eine interessante Alternative zum A* Algorithmus hingewiesen werden. Dies ist der D* Algorithmus (dynamischer A*), der in der Lage ist dynamische Informationsänderungen effizient zu verarbeiten, und somit ein ständiges Neuplanen bei Änderungen auf dem Spielfeld hinfällig macht, wobei er dennoch wie der A* Algorithmus optimal und optimal effizient ist [Stentz (1994)]. Wenngleich die Rechenzeit und der Speicherbedarf des Systems unter ständiger Neuplanung mit dem A* Algorithmus bei den im Contest genutzten Kartengrößen keinerlei Probleme aufgeworfen haben, wäre der Einsatz des D* Algorithmus hinsichtlich Skalierbarkeit des Systems und *verteilten Planens verteilter Pläne* durchaus interessant.

[2] Der Agent Contest 2008 setzte auf ein sehr ähnliches Kommunikationsprotokoll, wie das des Contests von 2007, dadurch ist anzunehmen, dass auch die Protokolle weiterer Contests in diesem Rahmen ähnlich ausfallen werden.

Literaturverzeichnis

[Bonabeau und Theraulaz 2000] BONABEAU, E. ; THERAULAZ, G.: Swarm smarts. In: *Scientific American* (2000), S. 72–79

[Braubach u. a. 2005] BRAUBACH, Lars ; POKAHR, Alexander ; LAMERSDORF, Winfried: Extending the Capability Concept for Flexible BDI Agent Modularization. In: *Proc. of PROMAS-2005*, 2005

[Busetta u. a. 2000] BUSETTA, Paolo ; HOWDEN, Nicholas ; RÖNNQUIST, Ralph ; HODGSON, Andrew: Structuring BDI Agents in Functional Clusters. In: *ATAL '99*, Springer, 2000, S. 277–289. – ISBN 3-540-67200-1

[Dennis u. a. 2008] DENNIS, Louise A. ; FARWER, Berndt ; BORDINI, Rafael H. ; FISHER, Michael: A Flexible Framework for Verifying Agent Programs. In: *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent systems (AAMAS2008), Estoril, Portugal*, URL http://www.ifaamas.org/Proceedings/aamas08/proceedings/pdf/paper/AAMAS08_0491.pdf, 2008

[Ferber 2001] FERBER, Jacques: *Multiagentensysteme*. Addison-Wesley, 2001. – ISBN 3-8273-1679-0

[FIPA a] FIPA A, Modeling-TC: *Agent UML (AUML)*. – URL http://www.auml.org/

[FIPA b] FIPA B, TC-B: *Agent Management Specification*. – URL http://www.fipa.org/specs/fipa00023/SC00023K.pdf

[OMG] OMG, Object Management G.: *Unified Modeling Language (UML)*. – URL http://www.omg.org/spec/UML/

[Patel] PATEL, Amit: *Game Programming Page: Notes on A**. – URL http://theory.stanford.edu/~amitp/GameProgramming/

[Rao 1996] RAO, Anand S.: AgentSpeak(L): BDI agents speak out in a logical computable language, Springer-Verlag, 1996, S. 42–55. – URL http://www.cs.mu.oz.au/682/agentspeak.pdf

[Renz 2007] RENZ, Wolfgang: *Materialien zur Lehrveranstaltung AOSE SoSe 2007*. 2007. – URL http://mmlab.haw-hamburg.de/AOSE/

[RMIT] RMIT, University: *The Prometheus Methodology*. – URL http://www.cs.rmit.edu.au/agents/SAC2/methodology.html

[Russel und Norvig 2003] RUSSEL, Stuart J. ; NORVIG, Peter: *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. – ISBN 0-13-080302-2

[Schneider und Werner 2001] SCHNEIDER, Uwe ; WERNER, Dieter: *Taschenbuch der Informatik*. Fachbuchverlag Leipzig, 2001. – ISBN 3-446-21753-3

[Stentz 1994] STENTZ, Anthony: Optimal and Efficient Path Planning for Partially-Known Environments. In: *ICRA International Conference on Robotics and Automation*, URL http://www.frc.ri.cmu.edu/~axs/doc/icra94.pdf, 1994

[Sudeikat 2004] SUDEIKAT, Jan: *Betrachtung und Auswahl der Methoden zur Entwicklung von Agentensystemen*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2004. – URL http://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/189/sudeikat_da.pdf

[Troposproject] TROPOSPROJECT: *Tropos: Requirements-Driven Development for Agent Software*. – URL http://www.troposproject.org/

[Weiss 2000] WEISS, Gerhard: *Multiagent systems : a modern approach to distributed artificial intelligence*. Cambridge, Mass. [u.a.] : MIT Press, 2000

[Wooldridge und Jennings 1995] WOOLDRIDGE, M. ; JENNINGS, N. R.: Intelligent Agents: Theory and Practice. In: *Knowledge Engineering Review 10(2)*, URL http://www.csc.liv.ac.uk/~mjw/pubs/ker95.pdf, 1995

[Wooldridge 2002] WOOLDRIDGE, Michael: *An introduction to multiagent systems*. Wiley, 2002

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28. August 2008	
Ort, Datum	Unterschrift