Kommunikation in Multi-Agenten-Systemen

PS Einführung in die Bioinformatik (SS 99)

Markus Amersdorfer
markus.amersdorfer@sbg.ac.at

Michael Noisternig
michael.noisternig@sbg.ac.at

Version: 2. Juli 1999
Uhrzeit: 12:30 - 13:00
Gruppe: 5

LV-Leiter: Dipl.-Ing. Mag. Roland Schwaiger
rschwaig@cosy.sbg.ac.at

Zusammenfassung

Als erstes beschäftigen wir uns in diesem Artikel mit der Theorie von Agenten, Multi-Agenten-Systemen (MAS), Kommunikation in MAS und Künstlichen Neuronalen Netzen (ANN).
In Kapitel 1 werden die Grundzüge erläutert, die für einen Agenten charakteristisch sind. Weiters zu diesem Thema wird darauf eingegangen, welche Möglichkeiten ein MAS bietet und welche Voraussetzungen dafür nötig sind.
Hier schließt in Kapitel 2 das Thema der Kommunikation an, wobei neben wichtigen Grundbegriffen wiederum die Möglichkeiten und Probleme in Bezug auf MAS erläutert werden.
Kapitel 3 behandelt die wichtigsten Entwicklungen im Bereich der ANNs. Einleitend dazu wird auf den biologischen Hintergrund dieser Theorie - das Gehirn - eingegangen.

Kapitel 4 zeigt die Umsetzung der Theorie in die Praxis anhand eines Projektes.
Agenten müssen in einer Welt mit Hindernissen Futterquellen finden und werden dabei von einem weiteren Agenten gesteuert. Diese Richtungsanweisungen müssen allerdings erst mit Hilfe eines ANN erlernt werden.

Inhaltsverzeichnis

Kapitel 1

Agenten

1.1 Was ist ein Agent?

Zu allererst muß gesagt werden, daß es keine allgemein akzeptierte Definition eines Agenten gibt. Das Online-Wörterbuch "Merriam-Webster" [2] definiert einen Agenten folgendermaßen:

> "Ein Instrument, durch welches eine führende Intelligenz Resultate erreicht." bzw. "Jemand, der von einer anderen Person authorisiert ist, anstelle dieser zu handeln als: [...] c) ein (geschäftlicher) Repräsentat [...]"

Im "The Webster Dictionary" [3] wird der Begriff definiert als:

> "Ein Agent ist eine Komponente, welche für oder anstelle einer anderen autoritätsbesitzenden Komponente arbeitet."

Diese Definitionen sind allerdings schwer verständlich.
In [4] beschreiben M. Genesereth und S. Ketchpel einen Softwareagenten als ein Programm, das über eine ausdrucksvolle Agentenkommunikationssprache (ACL) mit anderen Programmen kommunzieren kann, um Informationen (Daten und Code) auszutauschen. (Näheres dazu siehe Abschnitt 2.2.)
Agenten haben eine nachrichtenbasierte Schnittstelle (siehe Abschnitt 2.1), die unabhängig ist von internen Datenstrukturen und Algorithmen (design autonomy). Das heißt, die programmiertechnische Implementierung spezieller Softwareagenten ist frei wählbar, da die Schnittstellen zur Außenwelt für die Kommunikation standardisiert sind.
Auf Nachrichten zeigen Agenten ein bestimmtes Verhalten. Jedoch besitzen auch Objekte in der Objektorientierten Programmierung (OOP) diese Eigenschaften.

Bemerkung: Bei Verwendung der Bezeichnung "Softwareagent" stellt sich folgende Frage: "Was ist ein Hardwareagent?" Dieser Begriff ist unseres Erachtens nicht definiert, jedoch kann man einen Hardwareagenten wohl am ehesten als Robotor, sprich Softwareagent mit Hardware, beschreiben.

1.2 Was ist der Unterschied zwischen einem Objekt in der OOP und einem Agenten?

Ein Unterschied ist, daß Agenten eine einheitliche und immer gleich verständliche Sprache haben. Dies ist jedoch nicht ausreichend, da es zum einen, wie später beschrieben wird, Nachrichten gibt, die gar nicht klar definiert sind und somit immer anders verstanden werden können, und zum anderen, weil auch Programme in einem System, in dem ein Kommunikationsprotokoll als Sprache verwendet wird, diese Eigenschaften erfüllen, z.B. Mail-Server mit SMTP als Kommunikationssprache.

M. P. Singh [5] liefert eine wesentlich bessere Definition eines Agenten:

> "Ein Agent ist ein Programm, das auf seine Umgebung reagieren und sowohl mit anderen Agenten zusammenarbeiten (interoperability) als auch ohne sie agieren kann. Agenten können selbstständig handeln und insofern Handlungen ablehnen oder annehmen (execution autonomy)."

Auf den Punkt bringt es [6]:

> "Im Gegensatz zu Objekten haben Agenten Ziele und handeln aktiv, um diese zu erreichen."

1.3 Was sind also die wichtigsten Eigenschaften eines Agenten?

- *Autonomie:* ein Agent kann beliebig aufgebaut sein (design autonomy) und er kann selbst wählen, welche Handlungen er ausführt (execution autonomy)
- *Reaktivität:* jeder Agent muß auf Veränderungen in seiner Umgebung entsprechend reagieren können; das bedeutet nicht unbedingt eine nach außen sofort erkennbare Reaktion, sondern möglicherweise auch einfach nur eine agenteninterne Veränderung von Zuständen oder Zielen
- *Interoperabilität:* Agenten müssen Informationen mit anderen Komponenten (Agenten, Programmen, Menschen) austauschen können

1.4 Klassifizierung von Agenten

Weitere Eigenschaften kann man auch zur Klassifizierung verwenden [7], [6], [8]:

- *Intelligenz*: ein Agent kann die beste Lösung in einer bestimmten Situation finden
- *Adaptivität, Flexibilität:* ein Agent verändert sein Verhalten aufgrund seiner früheren Erfahrungen
- *Zielorientierung:* ein Agent arbeitet aktiv, um ein Ziel zu erreichen
- *Charakter:* ein Agent besitzt eine "Persönlichkeit" und einen emotionalen Status

- *Mobilität:* ein Softwareagent kann sich zwischen verschiedenen Rechnern bewegen

Den Charakter eines intelligenten Agenten bestimmt vor allem, ob der individuelle mentale Status des Agenten oder die sozialen Aspekte der Kommunikation betont werden. Nach M. P. Singh [5] wird ein mentaler Status am einfachsten durch "beliefs" und "intentions" beschrieben, die die gesamte Handlungsweise des Agenten bestimmen.

Von besonderer Bedeutung ist das BDI-Modell *(belief-desire-intention)*. Wie in [9] ersichtlich ist, wird hierbei der interne Status eines Agenten als ein Konstrukt von Wissen und daraus resultierenden Wünschen und Absichten dargestellt. Ein Agent hat hierbei Ziele bzw. Wünsche und sucht einen Weg, diese mit Hilfe seines Wissens zu erreichen. Aufgrund neu erlernten Wissens kann er dann entweder den Weg oder das Ziel ändern.

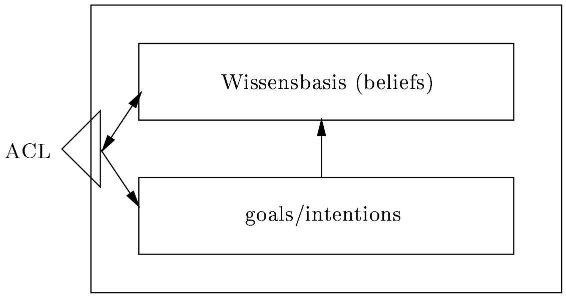
Ein Beispiel:

> Ich glaube, daß ich gut bezahlt werde, wenn ich hart arbeite.
> Ich wünsche mir einen Ferrari.
> Also arbeite ich hart, um mir einen Ferrari zu kaufen.
>
> Nach mehreren Jahren harter Arbeit habe ich noch immer keinen Ferrari!
> Also könnte ich
>
> - mir einen Lotterieschein kaufen (meine Absicht ändern)
> - erkennen, daß das nur ein Traum ist, und auf einen Mazda hinarbeiten
> - erkennen, daß harte Arbeit nicht genug Geld bringt und auf kriminelle Weise zu Geld kommen.

Soziales Verhalten ist das erklärte Ziel von Agentenentwicklern; ein Ziel, das noch lange nicht erreicht ist. Solche *sozialen Agenten* sollten zum Beispiel die Zusammenhänge von Handlungen in der Umgebung verstehen, alle Gesprächspartner gleich behandeln, den Kontext eines Gespräches verstehen und alle Kommunikationsarten beherrschen (siehe 2.2.2 und [5]).

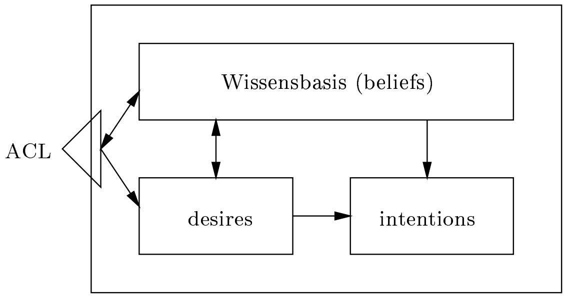
1.5 Einfache Agentenmodelle

Einfacher Agent



1. Agent kommuniziert in ACL
2. beliefs können durch assertives erweitert werden
3. goals werden durch die ACL festgelegt
4. ein erreichtes Ziel erweitert das Wissen

BDI-Agent



1. Agent kommuniziert in ACL
2. Informationen verändern beliefs und desires
3. aus beliefs und desires entstehen intentions
4. erreichte und nicht erreichte Ziele verändern beliefs

1.6 Anwendungsgebiete für Agenten

Im folgenden einige Projekte der Software Agent Group [10] am MIT, gegründet von Pettie Maes.

1. *Butterfly* (1998-Present)
   by Neil W. Van Dyke
   Butterfly ist ein Agent, der stichprobenartig tausende laufende IRC-Channels (Internet Relay Chat) dursucht, und für den Benutzer interessante vorschlägt.

2. *Amalthea* (1996-Present)
   by Alexandros G. Moukas, Pattie Maes
   Anstatt einen Agenten mit der übermächtigen Aufgabe zu betrauen, für den Benutzer relevante jedoch weitverstreute Informationen des Internets zu sammeln, basiert Amalthea auf einem Multi-Agenten-System, welches diese Aufgabe mittels mehrerer Agenten ausführt und dem Benutzer das Ergebnis präsentiert. Weiters paßt sich das System den sich verändernden Bedürfnissen des Benutzers an.

3. *Yenta* (1995-Present)
   by Lenny Foner
   Yenta ist ein agentenbasiertes System, welches verschiedene Benutzer mit gleichen Interessen oder Arbeitsaufgaben gruppiert und zu zwischenmenschlicher Kommunikation anregt, dabei jedoch nicht die Privatshpäre der Einzelnen Gruppenmitglieder vernachlässigt.

4. *Challenger* (1996)
   by Alexandros G. Moukas, Anthony Chavez, Pattie Maes
   Challenger ist ein Multi-Agenten-System, welches für die Durchführung von Aufgaben die Rechenzeit verschiedener Computer verwendet.

5. *Webdoggie* (1994-1995)
   by Yezdi Lashkari, Pattie Maes
   Webdoggie ist ein System, welches mittels zusammenarbeitendem Filtern für den Benutzer möglicherweise interessante Internetseiten findet.

1.7 Multi-Agenten-Systeme (MAS)

Eine Möglichkeit der Lösung von immer größer werdenden und mehr Rechenzeit in Anspruch nehmenden Probleme ist deren Aufteilung auf mehrere, zusammenarbeitende Programme. Sind diese Programme einheitlich aufgebaut, so kann man eine Aufgabenverteilung durch das klassische Multiprozessorensystem anwenden.
Sind diese jedoch individuell verschieden und können durch ein eigenständiges Agieren in einer Welt, die entweder einen realen Weltausschnitt darstellt oder die künstlich mittels Objekten beschrieben ist, beschrieben werden, so ist dies das Forschungsgebiet der "Verteilten Künstlichen Intelligenz" (Distributed Artificial Intelligence = DAI). [11] [12]
Dieses wird in zwei Bereiche unterteilt:

- "Distributed Problem Solving" (DPS) und
- "Multi-Agenten-Systeme" (MAS).

DPS beschäftigt sich damit, wie eine Aufgabe auf mehrere Systeme aufgeteilt werden kann, die kooperieren, indem sie Wissen über das Problem und den Fortschritt ihrer Lösungen untereinander austauschen. Jedliche Handlungen zwischen den Systemen sind dabei fest in das DPS-System integriert.

In einem MAS werden Aufgaben an mehrere autonome Agenten gestellt, die selbst für die Kommunikation und die gemeinsame Lösung des Problems sorgen müssen. Ein MAS kann also als ein "lose gekoppeltes Netzwerk von Problemlösern, den Agenten" angesehen werden, "die alleine ein Problem nicht lösen können".

Die wesentlichen Vorteile eines MAS sind ein schnelleres Lösen von Problemen bei Ausnützung der parallelen Problemlösung der Agenten, verringerte Kommunikation aufgrund der Übertragung von nur wesentlichen Informationen direkt an andere Agenten, größere Flexibilität (verschiedene Agenten können verschiedene Eigenschaften haben) und größere Zuverlässigkeit, wenn man Agenten erlaubt, daß sie anderen Agenten, welche ihre Aufgaben nicht lösen, diese wieder entziehen.

1.7.1 Organisation in MAS

Um ein Problem erfolgreich lösen zu können, müssen folgende Punkte in einem MAS sichergestellt werden:

- Jedes bestehende Teilproblem muß von mindestens einem Agenten gelöst werden können,
- Agenten müssen derart kommunizieren, daß die Lösung ihrer Teilprobleme vorangetrieben wird und diese in eine gemeinsame Lösung eingebaut werden kann,

- Die oben genannten Punkte müssen mit den vorhandenen Resourcen erreichbar sein.

Um ein funktionierendes MAS aufzubauen, müssen folgende Punkte untersucht werden:

- Koordination
- Kooperation
- Planung
- Kommunikation

Kommunikation wird im nächsten Kapitel ausführlich behandelt.

1.7.2 Koordination

Koordination betrifft die Aufteilung von raren Resourcen und die Mitteilung von Zwischenergebnissen.

Resourcen werden im einfachsten Fall einfach von Agenten, die das selbe Ziel haben, gemeinsam benutzt *(mutual adjustment)*. Hat ein Agent die Kontrolle über diese Resourcenteilung, so spricht man von *direct supervision*. Dieser Supervisor-Agent hat meistens bestimmte Regeln für die Verteilung der Resourcen. Mit der Methode der direct supervision kann man bei kleinen MAS gut eine hierarchische Struktur aufbauen. In größeren MAS ist der Kommunikationsaufwand jedoch zu hoch.

Koordination ist ein wichtiger Punkt, da jeder Agent nur eine kleine Perspektive des Problems hat. Damit nicht jeder Agent irgendein Problem löst, ist es wichtig, daß sie eine Vorstellung der beliefs und intentions anderer Agenten haben (*soziale Agenten*).

1.7.3 Kooperation

Ziele für Kooperation sind:

- die Problemlösung durch Parallelität zu beschleunigen,
- die Anzahl der gleichzeitig lösbaren Probleme erhöhen durch Resourcenteilung,
- mehrfache Lösung von Problemen eliminieren und
- störende Interaktionen zwischen Agenten zu vermeiden.

Effektive Kooperation beruht auf einem balancierten Einsatz von Kommunikation und Resourcen. Der Grad der Kooperation reicht von total bis antagonistisch. Bei totaler Kooperation bekommen Agenten oft unabhängige Probleme und man bezahlt insofern mit hohen Kommunikationskosten. Im Gegensatz dazu könnten antagonistische Agenten überhaupt nicht zur Lösung eines Problems kommen, weil es nur durch totale Kooperation erreicht werden kann (*cooperative distributed problem solving*).

1.7.4 Planung

Unter Planung versteht man die kontrollierte Abfolge der einzelnen Aktivitäten der Agenten in einem MAS. Es gibt dafür verschiedene Ansätze:

Bei *zentralisierter Planung* wird ein Agent für die Erstellung eines Multiagentenplans gewählt, der eine Übersicht über alle anderen Agenten hat.

Verteilte Planung wird vor allem angewendet, wenn eine derartige Übersicht nicht gewährleistet werden kann. Dabei wird entweder ein Problem in immer kleinere Teilprobleme zerlegt und weiterverteilt *(task-driven planning)*, oder, wenn jeder Agent bereits seine zu lösenden Probleme hat, versucht, diese miteinander zu vereinbaren *(plan coordination)*.

Grundsätzlich braucht Multiagentenplanung eine Plansynchronisation. Diese kann entweder von einem einzelnen Agenten (*coordinator* oder *mediator*) durchgeführt werden oder sie kann durch Verhandlungen im MAS erreicht werden.

Kapitel 2

Kommunikation

2.1 Kommunikations-Mechanismen

Unter Kommunikations-Mechanismen in Zusammenhang mit Agenten und MAS wollen wir Methoden verstehen, um Nachrichten zwischen Agenten auszutauschen. Als die zwei wichtigsten Mechanismen werden in [13] angeführt:

- *Message-Passing (Bringschuld):* Charakteristisch für diesen Mechanismus ist, daß die Agenten direkt miteinander kommunizieren. Weiters ist der benachrichtigende Agent A selbst dafür verantwortlich, daß seine Nachricht zu einem bestimmten Agenten B gelangt. Das heißt, er muß in einer Verbindung zu seinem Kommunikationspartner die Nachricht zu diesem "bringen".

- *Blackboarding (Holschuld):* Dieser Mechanismus ist vergleichbar mit der Funktionsweise von Newsgroups. Agent A schickt seine Nachricht zu einer Zentrale, über die sämtliche Nachrichten laufen. Sein Kommunikationspartner Agent B ist nun selbst dafür veranwortlich, eigenständig die Nachricht bei der Zentrale abzuholen.

Diese Zentrale kann entweder implementiert werden als herkömmliche Datenbank, oder aber auch als ein eigener Agent C, der sich zur Verfügung stellt um die Kommunikation zwischen anderen zu ermöglichen. In diesem Fall allerdings ergibt sich zwischen den Agenten A und C wiederum eine auf Message-Passing basierende Kommunikation.

Diese beiden Mechanismen können auch kombiniert werden. Zum Beispiel bestehen komplexere Agenten zum Teil aus mehreren Untersystemen ("Subagenten"), welche untereinander Information lokal mittels "Blackboarding" transferieren. Die Agenten selbst jedoch kommunizieren mit Hilfe von "Message-Passing". Auch verschiedene MAS machen sich diese Architektur zunutze, indem sie diese zum Beispiel zur Lösung von Problemen betreffend Überwachen und Interpretation von Daten einsetzen.

2.2 Ansätze für Kommunikation

Um Multi-Agenten-Systeme implementieren zu können, benötigt man als grundlegendes Werkzeug der Kommunikation eine Sprache. In [4] werden zwei übliche Ansätze für Sprachen genannt:

- Skriptsprachen
- deklarative Sprachen.

Skriptsprachen (wie TCL, Apple Events, Telescript) tauschen ganze Prozeduren aus und sind somit einfach und leistungsfähig. Deren Nachteil allerdings liegt darin, daß sie nur unidirektional und schwierig zu koordinieren sind, wenn mehrere Skripts von mehreren Agenten gleichzeitig ankommen.
Deklarative Sprachen verstehen Kommunikation als Austausch von deklarativen Aussagen wie zum Beispiel Definitionen oder Annahmen. Das heißt, sie können ohne Interpreter ausgeführt werden. Wie T. Finin in [14] beschreibt, sind sie einerseits ausdrucksfähig genug, um vielfältige Informationen weiterzugeben und andererseits gleichzeitig kompakt.
Die Entwicklung von nachfolgenden deklarativen Sprachen wird vom ARPA Knowledge Sharing Effort [4] vorangetrieben. Sie besitzen einen erweiterungsfähigen, englischsprachigen Wortschatz, um die Lesbarkeit für Menschen zu erhöhen.

2.2.1 KIF

KIF (Knowledge Interchange Format) ist eine Sprache mit Prädikatenlogik erster Ordnung und einigen Erweiterungen, um ihre Ausdruckskraft zu erhöhen. Laut M. Genesereth und S. Ketchpel [4] soll KIF jedoch in erster Linie einfache Daten logisch darstellen können und besitzt dazu logische Operatoren wie Negationen, Disjunktionen, Regeln usw. Weiters kann KIF Wissen von anderen Wissensbasen beziehen.
Weil KIF jedoch kontextunabhängig ist, müßte jede Nachricht implizite Informationen über den Empfänger, den Sender, die Zeit einer Nachricht usw. enthalten. Die Effizienz einer Kommunikation kann erhöht werden, indem sprachliche Mittel verwendet werden, bei denen der Kontext eine Rolle spielt.

2.2.2 KQML

KQML (Knowledge Query and Manipulation Language) ist eine auf KIF basierende ACL, die in dieser Hinsicht erweitert wurde. Das erste Wort einer Nachricht bestimmt den Typ der Kommunikation.

Obwohl KQML im Prinzip nur Feststellungen und Anweisungen unterscheidet, so könnte man diese Typen grundsätzlich in sieben Kategorien einteilen und eine ACL dahingehend erweitern [5] (siehe 1.4):

- *Feststellungen (assertives)*, mit der Aufgabe zu informieren: "Die Tür ist geschlossen."
- *Anweisungen (directives)*, welche um etwas ersuchen: "Schließ die Tür!" - bzw. anfragen *(queries)*: "Können Pelikane fliegen?"
- *Verpflichtungen (commissives)*, die etwas versprechen: "Ich werde die Türe schließen."
- *Erlaubnisse (permissives)*, die Erlaubnis erteilen: "Du kannst die Türe schließen."

- *Verbote (prohibities)*, die eine Aktion untersagen: “Du darfst die Türe nicht schließen.”
- *Deklarationen (declaratives)*, welche Ereignisse in sich verursachen: “Ich nenne diese Tür die Golden Gate”
- *Äußerungen (expressives)*, um Emotionen und Einschätzungen auszudrücken: “Ich wünschte, diese Tür wäre geschlossen.”

2.3 Architektur von MAS

Sobald man eine ACL und die Möglichkeit, Agenten zu bauen, hat, braucht man nur noch eine funktionierende Organisation zwischen ihnen. Zwei verschieden Ansätze wurden dafür erforscht [4]:

- direkte Kommunikation und
- assistierte Koordination.

Für direkte Kommunikation gibt es die Möglichkeit einer Handlung per Vertrag *(contract net approach)* oder über ein Weitergeben von Spezifikationen *(specification sharing approach)*.
Im “contract net approach” verschicken Agenten, die keine Arbeit haben, Ansuchen um Dienste. Die Empfänger können diese dann auswerten und Angebote zurückschicken. Die ursprünglichen Agenten wiederum können dann entscheiden, welchen Job sie annehmen.
Im “specification sharing approach” schicken arbeitslose Agenten einfach Informationen über ihre Fähigkeiten und Bedürfnisse an andere Agenten, die diese dann auswerten können.
Direkte Kommunikation ist in großen Systemen, wie zum Beispiel dem Internet, jedoch viel zu kostenintensiv. Und auch die Agenten selbst sind sehr komplex, da sie alle in dieser Form mit anderen Agenten kommunizieren können müssen.

Bei assistierter Koordination kommunizieren Agenten nur mit lokalen Systemprogrammen, sogenannten *facilitators*, und diese dann mit anderen “facilitators”, welche wiederum mit “ihren eigenen” Agenten kommunizieren können. Dies zeigt auch schon, daß Agenten in einem solchen System (genannt *federated system*) ihre Autorität an ihre “facilitators” übergeben, welche somit für die Erfüllung ihrer Bedürfnisse verantwortlich sind.

2.4 OSI-Referenz-Modell

W. Bergt definiert in seinem Online-Lexikon [15] den Begriff *OSI-Referenz-Modell* folgendermaßen:

> Ein von der ISO entwickeltes Kommunikationsprotokoll, das allgemeine Festlegungen/Regeln für die Kommunikation in Netzwerken enthält. Das OSI-Referenz-Modell (auch OSI-Schichtenmodell) ist in sieben Schichten (sog. Layer) organisiert, die man in drei sog. Anwendungsschichten und in vier sog. Transportschichten unterteilen kann. Die Anwendungsschichten enthalten Regeln für den

Aufbau und den Abbau einer Sitzung, für die Datendarstellung und für anwendungsnahe Grundfunktionen (z. B. Daten- und Dateiübertragung). Die Transportschichten enthalten Regeln für den physikalischen Aufbau einer Sitzung, für die Synchronisation einer Verbindung, für die Wegbestimmung (Routing) und für den logischen Verbindungsaufbau.

Die sieben Schichten werden folgendermaßen beschrieben:

7. Anwendungsschicht Umfaßt sowohl die Netzwerk-Services (Datei-, Druck-, Nachrichten-, Anwendungs- und Datenbank-Service) als auch die dazu benötigten Regeln.

6. Darstellungsschicht Konvertierung von Daten in ein beiderseitig akzeptiertes Format (z.B. ASCII $\rightarrow$ ANS.1)

5. Sitzungsschicht (auch Kommunikationssteuerungsschicht) Erleichtert eine Wiederaufnahme einer abgebrochenen Kommunikation an bestimmten Punkten *(token)*

4. Transportschicht Verbindet die Netzwerkstrukturen mit den Strukturen der höheren Schichten. Sie unterteilt die Nachrichten der höheren Schichten in Segmente und liefert die Segmente wieder zuverlässig an die Sitzungsschicht. Gebräuchliche Transportprotokolle sind TCP, SPX, NWLink und NetBEUI.

3. Netzwerkschicht Organisation des Austausches von Daten zwischen bestimmten Orten des Netzwerks durch Vermittlung, Netzwerkschicht-Adressierung und Leitweg-Algorithmen. (einen Kommunikationsport festlegen)

2. Datensicherungsschicht Organisieren der Bits der Bitübertragungsschicht in logische Gruppen von Informationen, die Rahmen genannt werden, Fehlerfeststellung, Steuern des Datenflusses

1. Bitübertragungsschicht (auch Physikalische Schicht) Koordination der Regeln für die Übertragung der Bits. Die Schicht legt folgendes fest:

- Physische Netzwerkstrukturen,
- Mechanische und elektrische Spezifikationen für die Benutzung des Übertragungsmediums,
- Bitbertragungscodierung und Timing-Regeln.

Nun soll versucht werden, Kommunikationsmodelle bzw. hier die *natürliche Kommunikation* zwischen Menschen auf das OSI-Schichtenmodell abzubilden.

Ablauf auf der Senderseite:

7. Anwendungsschicht Der Kommunikationspartner wird identifiziert.

6. Darstellungsschicht Es wird ein beiderseitig akzeptiertes Format der Kommunikation gesucht. Zum Beispiel wird Englische Sprache festgelegt.

5. Sitzungsschicht (auch Kommunikationssteuerungsschicht) Einbringen von Pausen (token), um die Reaktion des Gesprächspartners abzuwarten

4. Transportschicht Transport der Information durch mündliche Mitteilung

3. Netzwerkschicht entfällt - es kann kein Port bestimmt werden, da der Schall sich in jede Richtung verbreitet

2. Datensicherungsschicht Fehlerfeststellung ist auf Senderseite nicht erforderlich, da der Wortschatz viele Nicht-Code-Wörter enthält

1. Bitübertragungsschicht (auch Physikalische Schicht) Übertragung der Informationen über das Medium Luft mit technischen Daten wie Schallgeschwindigkeit

Ablauf auf der Empfängerseite:

1. Bitübertragungsschicht Aufnahme der akustischen Signale durch das Gehör

2. Transportschicht Weiterleitung der akustischen Reize an das Gehirn

3. Datensicherungsschicht Überprüfen auf Sinnhaftigkeit (Trennung von Nicht-Code-Wörtern vom Wortschatz) und Fehlerkorrektur aufgrund sprachlicher Redundanz

4. Netzwerkschicht entfällt

5. Sitzungsschicht Pausen können für Reaktionen auf erkannte Fehler verwendet werden, um die Nachricht zu wiederholen

6. Darstellungsschicht entfällt - die Informationen können in einem beliebigen Format gespeichert werden

2.5 Ein alternatives System nach [1]

Ablauf auf der Senderseite:

1. Intention Der Sprecher will, daß der Hörer eine Information P glaubt

2. Erstellung der Information Der Sprecher wählt bestimmte Worte für die Bedeutung der Information

3. Synthese Der Sprecher spricht die Worte (Abbildung der Information auf die mündlichen Signale)

Ablauf auf der Empfängerseite:

1. Aufnahme der Information Der Hörer nimmt die Information wahr

2. Analyse Der Hörer versteht die Nachricht

3. Eindeutigkeit Der Hörer folgert, daß ihm der Sprecher eine bestimmte Information P' übermitteln wollte, wobei P' = P sein sollte

4. Aufnahme Der Hörer entscheidet, ob er sich die Information merkt oder nicht

Diese Darstellung beleuchtet die Fehler, die bei einer Kommunikation entstehen können.

Kapitel 3

Artificial Neural Networks (ANN)

3.1 Motivation und biologische Grundlagen

R. Rojas [16] beschreibt *Künstliche Neuronale Netze* als neuartiges Berechnungsmodell. Der wesentliche Unterschied zwischen neuronalen Netzen und herkömmlichen algorithmischen Modellen ist, daß ANN eine Struktur und ein Lernverfahren vorgegeben bekommen. Es wird jedoch nicht für jedes Problem ein spezielles Programm geschrieben, sondern das Netz selber muß in einem Lernprozeß die richtige Konfiguration finden.
Die Grundidee bei ANN ist die Nachbildung der Informationsflüsse im Gehirn. Die Funktionsweise dieser kann folgendermaßen erklärt werden [17]:

3.1.1 Funktionsweise von Neuronalen Netzen im Gehirn

Die verschiedenen Abschnitte des Gehirns bestehen aus Neuronen (Nervenzellen), die miteinander mittels Synapsen verbunden sind. Ein Neuron besteht aus einem Zellkern, Dendriten auf der Eingangsseite und einem axonalen Strang auf der Ausgangsseite. Im Durchschnitt ist jedes Neuron sowohl bei den Dendriten als auch über das Axon mit ca. 8000 weiteren Neuronen verbunden. Um die Informationsverarbeitung in einer angemessenen Zeit zu ermöglichen - allein der menschliche Kortex hat schon 10^{10} bis 10^{11} Neurone - ist das kortikale Vernetzungsprinzip nicht *terminal ramification* - alle Neurone sind linear hintereinandergeschaltet - sondern *distributed ramification* - verteilte Vernetzung. Dies bedeutet, daß eben jedes Neuron mit mehreren anderen gleichzeitig verbunden ist.
Die Funktionsweise von Neuronen basiert auf dem Weiterleiten bzw. Verarbeiten von elektrischen Impulsen. Impulse werden von einer Nervenzelle nur weitergeleitet, wenn sie dementsprechend aktiviert bzw. gereizt wird. Um das zu erreichen, muß ein Grenzwert überschritten werden, indem sich Impulse von vorgeschalteten verschiedenen Neuronen beim Dendritenbaum aufsummieren - das sogenannte Aktionspotential entsteht. Sobald der Grenzwert am Zellkern überschritten wird, schickt die Nervenzelle selbst einen elektrischen Impuls weiter, der wiederum mittels der Synapsen am Axon an etwa 8000 Neurone weitergeleitet wird. Je mehr Nervenzellen also gleichzeitig ein Signal senden, desto schneller wird der Grenzwert überschritten und desto schneller erfolgt eine Gesamtverarbeitung der Information.
Weiters soll noch erwähnt werden, daß nicht die Stärke eines Aktionspotentials entscheidend ist für die Information, sondern daß die Frequenz der Impulse die Nachricht verschlüsselt.

3.2 Übersicht über ANN

Im folgenden ist eine Zusammenfassung eines Artikels von D. Olmsted [18] bzw. der Literatur von R. Rojas [16] zu lesen.

3.2.1 Neurale Logik

N. Rashevsky veröffentlichte 1938 seine Idee, daß das Gehirn mittels binären logischen Operation organisiert sei, da die Aktionspotentiale als binärer Wert "1" interpretiert werden können. Er präsentierte auch eine Möglichkeit, das XOR (ausschließendes oder) der binären Logik zu implementieren.
1943 führten W. McCulloch und W. Pitts die sogenannte *threshold logic* ein, die auf der Überlegung basiert, daß Aktionspotentiale "Alles-oder-nichts"-Aktionen sind. Daher kann bei einem Grenzwert 2 das logische UND realisiert werden, bei dem Grenzwert 1 das ODER. Diese Operationen werden graphisch dargestellt mittels logischer Schaltungen mit zwei Inputs und einem Output.

3.2.2 Zufällig verschaltete Netzwerke

Diese eben beschriebenen Ansätze konnten jedoch erst mit der fortschreitenden technischen Entwicklung in der Computerbranche getestet und realisiert werden. Ausschlaggebend dafür war die von D. Hebb 1949 aufgestellte Regel bezüglich der Lernfähigkeit von Zellen/Vernetzungen:

> When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

Das heißt, daß durch einen Wachstums- oder Stoffwechselprozeß die nachgeschaltete Zelle B leichter und rascher erregt werden kann.

1954 realisierten Farley und Clark das erste zufällig vernetzte "Reverberatory Network" basierend auf der Hebb-Regel. Es bestand aus Neurone repräsentierenden Knoten, die zufällig miteinander mittels unidirektionalen, gewichteten Verknüpfungen verbunden waren. Sobald die sich aufsummierenden Inputs an den Knoten einen bestimmten Grenzwert überschritten, wurde als Output der Wert 1 geliefert. Eine Mustererkennung wurde dann als erfolgreich eingestuft, wenn das angelegte Muster in bestimmten Abschnitten des Netzwerks andauernd mehr Aktivität hervorrief als in anderen.
Notwendig war nun nur noch, entscheiden zu können, ob der Summationswert erhöht oder vermindert werden sollte. Dies wurde erreicht mittels einer Bewertung der Verbindungen zwischen den Knoten und deren ständige Beobachtung.

Der nächste Schritt war ein Netzwerk, bei dem das Output von Zellen (zufällig verschaltet) als Input anderer Zellen fungierte. Mittels mehrerer Modifikationen konnte erreicht werden, daß bei einem Stop der Inputs nur bestimmte Teile des Netzwerks wellenartigen Nachhall zeigten.

Hiermit war eine Lernfähigkeit des Netzwerks bezüglich öfter vorkommender Muster erreicht.

F. Rosenblatt initiierte den nächsten Schritt 1958, indem er ein Netzwerk schuf - *Perceptron* genannt -, bei dem der Gedächtniseffekt mittels der Verbindungen zwischen Output und Input in der Vernetzung realisiert wurde, wobei sich diese bei ständiger Benutzung veränderten bzw. umstrukturierten. D.h., daß Muster, die öfter vorkamen, "gelernt" wurden. Dieser Typ von Netzwerken wird "selbstlernend" *(auto-learning network)* genannt.
Die Funktionsweise: Mehrere Input-Linien liefen bei einem Summationsknoten zusammen. Dieser war mit einer Grenzwert-Einheit verbunden, die den Wert 1 weitergab, sobald die Summe der Input-Linien den Grenzwert überschritt. Mehrere positive binäre Werte und weniger negative Werte wurden als Input verwendet. Der von der Grenzwert-Einheit produzierte Wert 1 wurde dann mittels einem nachfolgenden Multiplikations-Faktor zwischen 0 und 1 - dem Gewicht - modifiziert. Die so erhaltenen Werte einzelner Subschaltkreise wie eben beschrieben wurden zu einer Vergleichseinheit geliefert, die den höchsten Wert herausfilterte und diesen zu dem entsprechenden Gewicht wieder zurücklieferte, um dieses zu erhöhen. Auf diese Art wurde die Übertragungseffizienz bestimmter öfters verwendeter Einheiten erhöht, was der oben erwähnten Hebb-Regel entspricht.
Netzwerk die wie hier eine Vergleichseinheit verwenden werden "Konkurrenznetzwerke" *competitive networks* genannt, da die einzelnen Subschaltkreise konkurrieren, daß deren Signal gewählt wird.
Die Verwendung von gänzlich zufälligen Verbindungen ist nicht effizient, da einige dieser nie benutzt würden. Zusätzlich führt die Verwendung jener zu einer kombinatorischen Explosion, da die Anzahl und Komplexität der Muster ansteigt. Daraus folgt, daß die biologische Evolution mit großer Wahrscheinlichkeit auf einen anderen Weg setzt als eben die Verwendung von gänzlich zufälligen Verknüpfungen.

3.2.3 Neuronale Musterklassifizierung

ADALINE ist eine Abkürzung für ADAptive LINear Element. Adaline wurde von B. Widrow und M. Hoff entwickelt, um binäre Muster dahingehend zu erkennen, daß eine Vorhersage betreffend dem nächsten Bit in einer Folge von Bits möglich sei.
Das erste mal ist das Gewicht - der Multiplikationsfaktor - vor dem Summationsknoten, und es existiert keine Grenzwerteinheit. Adalines basieren auf einer ein Ziel suchenden Lernprozedur, wobei der Output-Wert eines Subschaltkreises als Ziel für ein jedes Muster definiert ist. Der Subschaltkreis muß lernen, einen richtigen Multiplikationsfaktor zu finden, um für ein Set von Input-Mustern den entsprechenden Zielwert zu erreichen.

Ein nächster Schritt wurde wiederum von 1962 F. Rosenblatt initiiert, indem er das eben beschriebene Prinzip mit den vorher erwähnten verschmolz. Das daraus resultierende Netzwerk änderte als Lernvorgang den Multiplikationsfaktor einer fixen Formel entsprechend, sodaß bei einer Fehlklassifizierung des Musters die Gewichte um einen fixen Betrag proportional zum Gesamtfehler erhöht wurden.
Der Prozeß des Findens der richtigen Multiplikationsfaktoren, welche die beste Mustertrennung erlauben, ist sehr komplex und benötigt die Verwendung verschiedener Optimierungsmethoden.

Trotz verschiedener und laufender Verbesserungen konnten gewisse Probleme bei der Musterklassifizierung jedoch nicht gelöst werden.

3.2.4 Assoziierungs-Netzwerke

1972 ging man dann in eine etwas andere Richtung. Die Grundidee war immer noch, daß das Gedächtnis im Gehirn auf vernetzten und verzweigten Pfaden basiert. Nun sollte versucht werden, die eben erwähnten Musterklassifizierungs-Probleme bei Seite zu lassen und sich dem *wie* der Funktionsweise von Gedächtnis zuzuwenden. Ein Grundgedanke von J. Anderson:

> If a group of neurons projects to another we shall show that strengthening or weakening the synaptic connection between the two groups according to a simple multiplicative function of activity in pre- and postsynaptic cells automatically generates an interactive memory ...

Das heißt, daß durch Veränderungen - realisierbar mittels einer einfachen Multiplikationsfunktion - in den synaptischen Verbindungen zwischen Neuronen(-Gruppen) automatisch ein interaktives Gedächtnis entsteht.

Als Ergebnis der unabhängigen Arbeiten von J. Anderson und T. Kohonen entstand jeweils ein ähnliches Netzwerk, die jedoch beide im Grunde Arrays von analogen ADALINE-Schaltungen waren.
Praktischer Nutzen eines solchen Netzwerks war beispielsweise, ein bestimmtes Input-Muster mit einem gewünschten Output-Muster zu assoziieren.

3.2.5 Mehrschichtige Netzwerke

1975 erweiterte K. Fukushima - basierend auf der Fähigkeit des Gehirns der Selbstorganisation - die ursprüngliche Funktionsweise des Perceptrons - siehe Seite 17.
Wie das originale Perceptron ist das *Cognitron* fähig, Muster ohne Mechanismen zu erlernen, um eine Musterentsprechung als erfolgreich oder nicht anzuzeigen. Im Gegensatz zum Perceptron kann das Cognitron besser mit dem "pattern subset problem" umgehen, bei dem ein Muster zur Gänze in einem anderen enthalten ist.
Um für bestimmte Berechnungen die Anzahl der Input-Linien klein zu halten, unterteilte Fukushima das Cognitron in mehrere Sektionen, die wiederum mittels mehreren Schichten miteinander verbunden wurden. Spätere Netzwerke (wie das Hybrid-Netzwerk) übernahmen dieses Schichtenprinzip, obwohl sie ansonsten sehr unterschiedlich strukturierte Netzwerke sind.

3.2.6 Das Hopfield-Assoziierungs-Netzwerk

1982 stellte J. Hopfield ein verändertes "Reverberatory Network" vor - siehe Seite 16.
Das Hopfield-Netzwerk unterscheidet sich zu dem ursprünglichen hauptsächlich durch bidirektionale Verbindungen zwischen den Summationsknoten und betonte individuelle Zellen (Knoten) anstatt Zellverbände. Der Grenzwert der Summationsknoten liegt bei 0, und wie schon bekannt wird ein Output-Wert 1 nur dann geliefert, wenn dieser Grenzwert

überschritten wird. Die Multiplikationsfaktoren können zwischen -1 und 1 liegen, was sowohl die Implementierung von exzitatorischen (addierenden) als auch von inhibitorischen (subtrahierenden) Operationen ermöglicht. Das Lernvermögen wird ermöglicht durch die Erhöhung der Gewichte (Mulitplikationsfaktoren) der aktiven Knoten - das sind jene, die den Output-Wert 1 produzieren - sowie der gleichzeitigen Verminderung der Gewichte aller anderer Knoten. Dies wird durch Normalisierung erreicht, d. h. die Summe aller Werte bleibt auf einem gleichen Niveau.
Um nun eine Assoziierung zwischen einem Input- und einem Output-Muster zu erlernen, muß das Netzwerk durch eine Trainingsphase, wobei das Input-Muster und zugleich das gewünschte Output-Muster gleichzeitig dem Netzwerk präsentiert werden. Der Vorgang der Gewichts-Anpassung wird so lange wiederholt, bis sich die Werte (fast) nicht mehr verändern.
Ein Nachteil dieses Netzwerks ist jedoch, daß in der Praxis die Anzahl der verschiedenen Assoziierungen, die erlernt werden können, nur etwa 15 % der Anzahl von Summationsknoten beträgt. Weiters wird aufgrund der Struktur das Netzwerk rasch sehr kompliziert, und es ist limitiert auf binären Input bzw. die Normalisierung der Gewichte.

3.2.7 Das Hybrid-Netzwerk

Dieses ebenfalls 1982 entwickelte Netzwerk basiert auf 2 Schichten (*Layers*), wobei jede Schicht verschiedene Strategien verfolgt. Die erste (Input-)Schicht konvergierender Subschaltkreise ist verantwortlich für Generalisierung, während die zweite (Output-)Schicht für Spezifizierung verantwortlich ist. Dies ist eine Grundstruktur, die für spätere Hybrid-Netzwerke fundamental ist.
Eine der vielen das Netzwerk aufbauende Sektionen besteht aus 3 Muster-Inputs und 2 Musterklassifizierungen (Outputs). Die Klassifizierung der Muster erfolgt mittels Unterscheidung in der Aktivierung der Output-Linien - Klasse A aktviert den ersten Output, Klasse B den zweiten.
Bemerkenswert ist, daß auch nach der Grenzwerteinheit Gewichte (Multiplikationsfaktoren) angebracht sind. Durch geschickte Methoden werden die Gewichte vor und nach der Grenzwerteinheit beim Lernprozeß so manipuliert, daß eine richtige Mustererkennung möglich wird.
Der Nachteil dieses Netzwerks liegt sowohl in den Limitierungen bezüglich den Multiplikationsfaktoren und Summationsknoten als auch bei der Notwendigkeit, daß die Input-Muster normalisiert werden müssen. Die Stärke des Netzwerks allerdings ist, daß nicht-lineare Muster klassifiziert werden können, da es die "Fundamental Classification Strategy" verwendet - 2 Schichten mit unterschiedlichen Strategien.

3.2.8 Weitere Multi-schichtige Netzwerke

Der nächste Schritt bezüglich mehrschichtiger Netzwerke war 1986 das sogenannte *Back Propagation network*, basierend auf der Idee, daß Fehler in der Mustererkennung auf das Netzwerk auf bestimmte Art und Weise aufgeteilt werden.
Die Gewichte können einen kleinen positiven oder negativen Wert haben. Nach dem Summationsknoten gehen die Werte in eine Funktion, deren Aufgabe es ist, den Output auf 1 oder geringer zu halten. Fehler sind die Differenz zwischen dem gewünschten und dem eigentlichen Output. Dieses "Back Propagation network" verwendet mehrere Schichten.

Der Nachteil dieses Netzwerks ist, daß es sehr langsam lernt und oft tausende von Durchläufen benötigt um ein Muster-Set zu erlernen.

3.2.9 Adaptive Resonance (ART) Networks

1987 wurde zuerst ein Netzwerk entwickelt, welches konvergente Subschaltkreise mit zufällig verbundenen Input-Linien verwendete. Die Input-Verknüpfungen, welche einigen präsentierten Mustern entsprachen, wurden verstärkt. Allerdings stieß man bald an mathematische Probleme bezüglich des Wachstums der Verknüpfungen bei einer steigenden Anzahl von Mustern.
Die 2. Generation von G. Carpenter und S. Grossberg verwendete binäre Input-Muster - das ART 1 (1987a) - bzw. analoge Input-Muster - das ART 2 (1987b). Diese beiden Netzwerke waren die ersten, welche die *convergent subcircuit dynamic allocation strategy* verwendeten. Diese Startegie weist einem präsentierten Muster einen neuen konvergenten Subschaltkreis zu, wenn kein anderer schon bestehender eine hinreichende Entsprechung liefert.
Das Problem liegt darin, entscheiden zu können, wann ein Subschaltkreis nicht einem Muster entspricht. Der einfachste Ansatz dafür liegt darin, davon auszugehen, daß je größer der Output ist, desto besser ist die Enstsprechung. Um das zu implementieren genügt die Verwendung einer Vergleichseinheit.
Um jedoch ein weiterhin bestehendes Problem zu lösen - mit obiger Technik ist nicht gewährleistet, daß nicht der eventuell numerisch größere Wert eines Mismatchs, sondern der numerisch kleinere Wert des Matchs verwendet wird - bedient man sich des Ansatzes des "positive feedback" des ART. Mit der Voraussetzung einer exakten Taktung der Signale ist es mittels einer komplizierten Struktur möglich, auch dieses Problem zu lösen.
Der Prozeß des Lernens findet statt, wenn kein konvergenter Subschaltkreis einem Input-Muster entspricht. Mehrere Durchläufe finden statt, wobei schließlich bei dem am besten geeigneten Subschaltkreis die Multiplikationsfaktoren entsprechend geändert werden. Ein extra Schaltkreis im ART 2 ist dafür verantwortlich, mittels einer "positive feedback"-Schleife nach einer bestimmten Zeitspanne den Stop der Druchläufe zu signalisieren. Weiters wird wieder Normalisierung der Signale verwendet, um die Werte in einem bestimmten Rahmen zu halten.

Kapitel 4

Projekt

Nun soll versucht werden, die erarbeitete Theorie mittels eines Projektes in die Praxis umzusetzen.

Die Idee unseres Projektes ist folgende: Mit Hilfe des Simulationsprogramms für MAS *Swarm 1.4.1* [19] wird eine Welt erstellt, in der sich

- Hindernisse
- Futter
- ein Master
- Slaves (Bugs)

befinden. Der Master kennt die Position sowohl aller Slaves als auch aller Futterquellen. Seine Aufgabe ist es nun, jeden der als Agenten implementierten Slaves zu einer dieser Futterstellen zu lenken. Dies erfolgt mit Hilfe von Richtungsanweisungen an die Slaves wie "gehe links".
Nun wissen die Slaves jedoch anfangs nicht, was die Hinweise des Masters bedeuten und wie sie mit diesen umgehen sollen. Sie müssen erst lernen, sich nach *links* zu bewegen, wenn der Master den Befehl "gehe links" gibt. Die Implementierung der Slaves erfolgt unter anderem mittels eines ANN. "Lernen" bedeutet für sie daher, mit den Signalen des Masters als Input für das ANN dieses so umzustrukturieren, daß die Bewegungen des Slaves (der Output) letztendlich zum gewünschten Ziel führen.

4.1 Lernverhalten

Für das Lernen der Slaves kommen zwei mögliche Ansätze in Frage:

- Supervised Learning
- Reinforcement-Learning

Supervised Learning ist anwendbar, wenn der gewünschte Output des ANN bereits bekannt ist. Das ANN wird dann aufgrund der Differenz zu dem tatsächlichen Output umstrukturiert. Bei unserem Projekt wäre das umsetzbar, indem der Slave dem Master *nachfährt*, wobei die Bewegungen des Masters den gewünschten Output darstellen.

Bei *Reinforcement-Learning* erhält das ANN sogenannte "Reinforcement-Signale". Diese Signale stellen eine Bewertung des Outputs dar, aufgrund derer dann das ANN umstrukturiert wird.
L. P. Kaelbling und M. L. Littman [20] definieren diesen Begriff folgendermaßen:

> "Reinforcement-Learning" behandelt das Problem eines Agenten, ein Verhalten erlernen zu müssen. Dies geschieht mittels "Trial-and-Error" Interaktionen mit einer dynamischen Umwelt.

Bei unserem Projekt haben wir uns für Reinforcement-Learning entschieden. Das bedeutet, daß die Slaves vom Master "ferngesteuert" werden. Dabei ist allerdings der Output für das ANN nicht direkt bekannt. Ein Slave bekommt hier nur eine "Strafe" sobald er etwas falsch gemacht hat bzw. eine "Belohnung" für richtiges Handeln.
Diese Bewertungen kann er auf zwei Arten bekommen - entweder durch den Master oder durch ein Hindernis. Näheres dazu siehe Abschnitt 4.3.1.

4.2 Kommunikation

Die Kommunikation zwischen Master und Slave kann entweder *unidirektional* oder *bidirektional* sein.
In jedem Fall sendet der Master Signale an die Slaves. Bei bidirektionaler Kommunikation senden diese nach erfolgter Auswertung ihre neue Position an den Master, während bei unidirektionaler Kommunikation der Master den Überblick über alle Slaves und deren Position hat.

Da im Rahmen unseres Projekts Reinforcement-Learning zum Einsatz kommt, und die Slaves damit selbständig lernen (siehe Abschnitt 4.3.1), ist eine Kommunikation von Slave zu Master nicht erforderlich. Deshalb haben wir uns für unidirektionale Kommunikation entschieden.

4.3 Die Komponenten

4.3.1 Slave

Nun soll näher auf den *Slave als Agenten* und dessen theoretischen Aufbau bzw. dessen Eigenschaften eingegangen werden.

Der Slave besitzt all jene wichtigen Eigenschaften, die in Abschnitt 1.3 beschrieben werden:

Autonomie: Der Slave wird - aufgrund der Möglichkeit der "design autonomy" und der an ihn gestellten Anforderungen - wie bereits erwähnt unter anderem mittels eines ANN realisiert.
Die "execution autonomy" besteht in dem eigenverantwortlichen Verändern der Gewichte des ANN bezüglich seiner richtigen oder falschen Reaktionen.

Reaktivität: Dies ist offensichtlich hier eine der wichtigsten Eigenschaften des Slave, ist es doch eine seiner Aufgaben, auf die Anweisungen des Masters zu hören und diese

bestmöglich auszuführen.
Wie schon erwähnt bezieht sich "Reaktivität" jedoch nicht nur auf nach außen sofort erkennbare Reaktionen wie zum Beispiel ein Richtungswechsel. Vielmehr ist hier genauso zu bedenken, daß die unten noch näher beschriebene Lernfähigkeit des Slave als Reaktivität verstanden werden kann, da diese eine agenteninterne Veränderung von Zuständen des ihn aufbauenden ANN bedeutet.

Interoperabilität: Da der Slave wissen muß, in welche Richtung er sich bewegen soll, ist es natürlich nötig, auch wenn nur unidirektional, mit dem Master kommunizieren zu können. In Swarm besteht hier die Möglichkeit, die Kommunikation mittels relativ einfacher programmiertechnischer Anweisungen zu gestalten, sodaß die Einbindung einer standardisierten ausgereiften Agent-Communication-Language entfällt.

Anhand dieser und der folgenden Aufstellung (vergleiche Abschnitt 1.4) wird klar, daß alle theoretisch wichtigen Eigenschaften im praktischen Einsatz auch unerläßlich sind:

Intelligenz: Im Rahmen dieses Projektes ist eine "Intelligenz" des Slave unentbehrlich. Diese drückt sich hauptsächlich dadurch aus, daß der Agent lernfähig ist, was zum nächsten Punkt "Adaptivität" überleitet.

Adaptivität, Flexibilität: "Ein Agent verändert sein Verhalten aufgrund seiner früheren Erfahrungen."
Da der Slave anfangs die Bedeutung der Anweisungen des Masters nicht versteht und infolgedessen fehlinterpretiert, wird sich der Agent als Reaktion auf eine Anweisung des Masters mit großer Wahrscheinlichkeit in eine falsche Richtung bewegen. Erst nach und nach kann das, ihn mittels Output steuernde, ANN so angepaßt werden, daß der Slave die gewünschte Aktion ausführt, das heißt sich in die gewünschte Richtung bewegt.

Zielorientierung: Das Ziel des Slave ist es, sein Futter zu finden. Um dies zu bewerkstelligen, muß er wie eben beschrieben das ihn aufbauende ANN umstrukturieren. Das bedeutet er muß mittels der in Kapitel 3 beschriebenen Funtkionsweisen eine Veränderung der Gewichte in der Struktur des ANN erreichen.

Hier stellt sich nun die Frage, *wie* diese beschriebenen Veränderungen ausgelöst und berechnet werden. Woher weiß der Agent, ob die von ihm durchgeführte Aktion richtig war? Woher, daß er eben in die falsche Richtung gefahren ist?
Als Lösungen bieten sich hier die zwei in Abschnitt 4.1 genannten Ansätze des Lernens an, wobei wir uns, wie schon erwähnt, für Reinforcement-Learning entschieden haben.

Reinforcement-Learning (RL)

Es gibt zwei mögliche Reinforcement-Signale:

- positiv
- negativ

Versucht ein Slave als Reaktion auf die Richtungsanweisungen des Masters sich aufgrund seines Outputs in ein Hindernis zu bewegen, so bekommt er ein *negatives* Reinforcement-Signal, bewegt er sich auf eine Futterquelle, bekommt er ein *positives* Reinforcement-Signal. Bei einem negativen Reinforcement-Signal werden die den Output auslösenden Gewichte verringert und die anderen erhöht - er wird bestraft. Bei einem positiven Signal werden nur die auslösenden Gewichte erhöht - er wird belohnt. Näheres dazu siehe Abschnitt 4.4.

Ergänzungen

Als zusätzliche Erweiterung wurde eine Lebensdauer der Slaves implementiert. Dies bedeutet, daß sie als Startwert eine bestimmte Anzahl von Schritten ermöglicht bekommen. Wenn ein Slave nicht rechtzeitig lernt, mit den Anweisungen des Masters umzugehen, wird er "verhungern". Kommt er jedoch rechtzeitig zu einer Futterquelle, so bekommt er Energie für weitere 100 Schritte.

Eine zusätzliche Idee war es, den Slaves - ähnlich den "Heatbugs" im gleichnamigen Beispielprogramm von Swarm - eine weitere Variable mitzugeben, die den aktuellen Wert der Glücklichkeit eines Slaves beinhaltet. Diese wäre zu Beginn mit einem durchschnittlichen Startwert vorbelegt.
Bei den ersten Schritten wird der Slave mit großer Wahrscheinlichkeit noch falsch auf die Anweisungen des Masters reagieren. Ist dies der Fall, wird der Slave von diesem bestraft, indem der Glücklichkeitswert herabgesetzt wird. Neben dem Hauptziel, sein Futter zu finden, ist es ein weiterer für den Slave wichtiger Punkt, "glücklich zu sein". Er wird sein fehlerhaftes Reagieren als falsch erkennen und eine entsprechende Umgewichtung im ANN vornehmen, um beim nächsten Versuch besser zu handeln.
Nach und nach wird der Slave lernen, auf die Anweisungen richtig zu reagieren. Ist dies der Fall, wird er vom Master belohnt - und seine Glücklichkeit steigt. Er weiß also, daß seine Handlungen richtig sind und kann dann noch mehr daraufhinzielen, diese perfekt auszuführen.
Anstelle der Glücklichkeitsvariable haben wir uns allerdings dafür entschieden, eine Bewertung der Reaktionen der Slaves und eine damit einhergehende Umstrukturierung der ANNs direkt vorzunehmen, wie oben beschrieben.

Der eben erläuterte Prozeß stellt einen extremen Ansatz im Umgang mit RL dar, da der Slave hier nach jedem Schritt belohnt oder bestraft würde.
Das andere Extrem wäre der Ansatz, daß ein Slave erst am Ende eines Durchlaufs vom Master belohnt oder bestraft würde. Dies wäre jedoch nicht sinnvoll, da der Slave mit dieser Methode zu wenig Rückmeldung vom Master bekommt. Er weiß nicht, sobald er beim Finden der Futterquelle belohnt wird, welche der vielleicht hunderten Aktionen, die er ausgeführt hat, richtig war.

Bezüglich dieser RL-Ansätze sind wir im Rahmen unseres Projektes einen Mittelweg gegangen, indem die Slaves, wie oben beschrieben, immer wieder ein Reinforcement-Signal bekommen, sobald sie sich in ein Hindernis bewegen bzw. eine Futterquelle finden.
In der Praxis hat sich auch herausgestellt, daß es zu einem langsameren Lernprozeß führt, wenn man die Slaves belohnt sobald sie einem Hindernis "ausweichen". Aus diesem Grund entschieden wir uns dazu, solche positiven Reinforcement-Signale wegzulassen.

4.3.2 Master

Den Slaves steht der *Master als Agent* gegenüber.
Dessen innere Struktur ist wesentlich unterschiedlich zu der eines Slaves. An den Master sind beispielsweise keine Anforderungen bezüglich Lernfähigkeit gestellt. Er kann nicht seine Anweisungen anders formulieren, um dem Slave ein schnelleres Lernen zu ermöglichen, da dies eher den umgekehrten Effekt hätte.

Autonomie: Viel ausgeprägter als beim Slave ist jedoch die "execution autonomy". Ist der Slave nur der ausführende Agent ohne auffallender Eigenverantwortung, so ist es der Master, der entscheidet, in welche Richtung er den Slave schickt bzw. welche Handlungen ausgeführt werden sollen.

Reaktivität: Der Master muß auch auf die Aktionen des Slave reagieren können. Er tut dies hauptsächlich dadurch, diesem eine weitere Anweisung zu geben, sobald ein Slave mit seiner Handlung fertig ist.

Interoperabilität: Dies bedeutet die Kommunikation zwischen den beiden Agenten. Hier besteht kein wesentlicher Unterschied zwischen Master und Slave.

Intelligenz: Auch beim Master bezieht sich "Intelligenz" auf das Finden der besten Lösung. Anders als beim Slave jedoch ist hier ein Lerneffekt, wie schon erwähnt, nicht vorhanden. Vielmehr hat der Master zu entscheiden, welcher der möglichen Wege der beste und kürzeste ist, um einen Slave zu seinem Ziel zu führen.

4.3.3 Futterquelle

In unserem Projekt besteht die Grafikausgabe in Swarm unter anderem aus einem Array, welches Werte für das Anzeigen von Objekten enthält. Eine Futterquelle ist dabei ein solches Objekt. Ist an einer Position, in die sich ein Slave bewegt, der Wert für Futterquelle gesetzt, so hat er das Futter gefunden.
Um einen längeren Lernprozess für die Slaves zu ermöglichen, wächst sofort eine Futterquelle nach, sobald ein Slave eine bestehende "gefressen" hat. Dies hat den Vorteil, daß die Slaves - sollten sie nur "zufällig" auf ein Futter treffen, obwohl sie die Anweisungen des Masters noch nicht zu verstehen gelernt haben - gleich die nächste Aufgabe "finde wieder Futter!" bekommen.
Weiters ist es möglich, die Dichte der Futterquellen einzustellen und damit direkten Einfluß auf die Überlebenschancen der Slaves zu nehmen.

4.3.4 Hindernis

Ein Hindernis ist genau wie eine Futterquelle implementiert, mit dem Unterschied, daß sich ein Slave nicht auf dieses bewegen kann.
Auch die Dichte der Hindernisse ist beliebig einstellbar.

4.4 ANN

Nun wollen wir noch den Aufbau und die Funktionsweise des ANNs eines unserer Slaves beschreiben.

Die Richtungsanweisungen des Masters dienen als Input für das “Feed-Forward-ANN”, welcher aus einer eindeutigen 4-Bit-Kodierung für die entsprechenden Richtungen besteht.

Ein einfaches Beispiel einer Kodierung:

1 0 0 0 bedeutet “up”,

0 1 0 0 bedeutet “right”,

0 0 1 0 bedeutet “down”,

0 0 0 1 bedeutet “left”.

Zu Beginn des Programms wird jedes ANN mit zufälligen Werten im Bereich [-1, 1] belegt. Somit werden alle möglichen Verbindungen der Input-Knoten zu den Output-Knoten mittels Gewichten bewertet. Der Wert für beispielsweise den Output-Knoten “up” ergibt sich, indem die einzelnen Inputwerte mit den Gewichten der Verbindungen, die zu diesem Knoten führen, multipliziert und dann aufsummiert werden. Der Gesamt-Output wird berechnet nach dem Prinzip *winner takes all*, was bedeutet, daß nur der größte Outputwert als tatsächlicher Output verwendet wird.

Abbildung 4.1 zeigt eine solche Belegung, wobei die verstärkten Striche den zu einem Input entsprechenden stärksten Output symbolisieren. Schickt der Master diesen Slave nun nach oben und wird als Input das Muster **1 0 0 0** angelegt, bewegt sich der Slave dementsprechend richtig nach oben, denn der Output entspricht der Form **1 0 0 0**, was ja wiederum “up” bedeutet. Der Input “down” (**0 0 1 0**) allerdings führt zu dem falschen Output “left” (**0 0 0 1**), genauso wie die Richtungsanweisungen “right” und “left” vom Slave falsch verstanden werden.

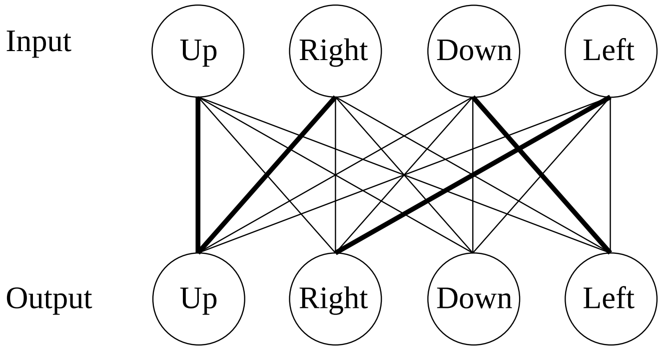


Abbildung 4.1: Zufällige Belegung des ANN eines Slaves

Auf seinem Weg zum Futter wird der Slave nun anfangs relativ oft versuchen, sich auf ein Feld zu bewegen, wo sich allerdings ein Hindernis befindet. Durch das implizite Wissen des

Slaves, daß ihn der Master immer richtig lenkt erkennt er, daß er einen Fehler gemacht hat und somit die Verbindung des aktuellen Inputs zu dem entsprechenden Output nicht stimmen kann. Als Folge wird eine Umstrukturierung des ANNs vorgenommen, wobei die zum falschen Output führende Verbindung um einen vom Benutzer einstellbaren Faktor δ herabgesetzt und die anderen drei - ebenfalls um den Faktor δ - verstärkt werden. Trifft der Slave allerdings auf eine Futterquelle, so wird sein letzter Schritt als richtig bewertet und das Gewicht der entsprechenden Verbindung um den Faktor Δ erhöht.
Nach mehrmaligem Wiederholen dieses Vorgangs ergibt sich schließlich die in Abbildung 4.2 gezeigte Struktur des ANNs, welche bei jedem der vier möglichen Inputs den entsprechenden richtigen Output liefert.

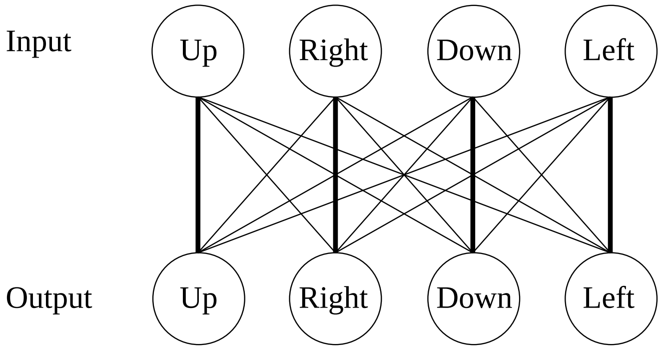


Abbildung 4.2: Gewünschte Struktur des ANN eines Slaves

Ein konkretes *Beispiel*: Der Master schickt einen Slave mit dem in Abbildung 4.1 gezeigten ANN nach rechts. Die Belegung des ANNs führt zu dem falschen Output "up", sodaß der Slave versucht, sich nach oben zu bewegen. Wenn sich an dieser Stelle jedoch ein Hindernis befindet, wird die Verbindung "right - up" abgeschwächt, und die drei anderen Möglichkeiten, unter denen sich mit Sicherheit die richtige Verbindung befindet, werden verstärkt. Somit ist beim nächsten Input "right" die Chance für "right - right", "right - down" und "right - left" größer, ein entsprechendes Output-Signal zu erzeugen.

4.5 Einstellbare Parameter

Folgende Parameter sind am Start der Simulation einstellbar:

- Weltgröße (*worldXSize* und *worldYSize*)
- Dichte der Hindernisse (*seedProbObst*)
- Dichte der Futterquellen (*seedProbFood*)
- Dichte der Slaves (*bugDensity*)
- Startenergie (*bugLifeTime*)
- Stärke der Reinforcement-Signale (*delta* (δ) und *DELTA* (Δ))

Nach dem Start kann man mit einem Rechtsklick auf einen Slave dessen Eigenschaften ablesen und ändern, wobei letzteres nur für die Position (*xPos* und *yPos*) und die restliche Energie (*lifetime*) interessant ist.

Neben den bereits genannten Parametern gibt es noch einen wichtigen Parameter an der Kommandozeile: *--varyseed* oder kurz *-s*. Damit kann man die Erzeugung eines immer neuen Zufallsstartwertes erzwingen, da dieser sonst zu Testzwecken immer den selben Wert aufweist.

4.6 Ergebnisse

4.6.1 Auswirkung der Hindernisdichte

Die Dichte der Hindernisse hat keinen großen Einfluß auf die Überlebensrate der Slaves. Dies ist so zu erklären: Bei wenigen Hindernissen ist das Futter meist direkt zu erreichen, doch bei falscher Interpretation der Befehle dauert es wesentlich länger, bis ein Slave auf ein Hindernis trifft und die entsprechenden negativen Reinforcement-Signale bekommt. Gibt es viele Hindernisse, so ist zwar das Futter wesentlich schwieriger zu erreichen, dafür ist jedoch die Lernrate viel höher.

4.6.2 Auswirkung der Deltas (δ und Δ)

Die Größe der Deltas hat im allgemeinen keine Bedeutung, da es nicht wesentlich ist, um wieviel die Gewichte des ANNs erhöht bzw. erniedrigt werden, sondern nur, DASS sie es werden. Umgekehrt kommt jedoch den Anfangsgewichten des ANNs eine Bedeutung hinzu, wenn diese im Vergleich zu den Deltas groß sind und die Anfangsenergie der Slaves gering ist. Denn dann haben die Slaves mit den glücklicheren Anfangsgewichten (deren Output also schon eher dem gewünschten entspricht) die höheren Überlebenschancen. Auf die statistische Überlebensrate hat dies jedoch keine Auswirkung.

4.6.3 Auswirkung verschiedener Inputmuster des ANNs

Aufgrund der in Abschnitt 4.4 beschriebenen Funktionsweise der ANNs unserer Slaves, ist es auch möglich, die 4-Bit-Kodierung der Richtungsanweisungen des Masters zu verändern, ohne daß dies einen Einfluß auf die Lernfähigkeit eines Slaves hat. Die folgenden Abbildungen zeigen jeweils die Struktur eines ANNs nach einem erfolgreichen Lernvorgang mit unterschiedlichen Kodierungen des Inputs.

Bei dem ANN in Abbildung 4.3 wurden jeweils die beiden Richtungsanweisungen "up" und "down" bzw. "right" und "left" vertauscht. Ein Slave mit diesem ANN muß nun lernen, den Input **1 0 0 0** mit dem Output **0 0 1 0** zu assoziieren usw.

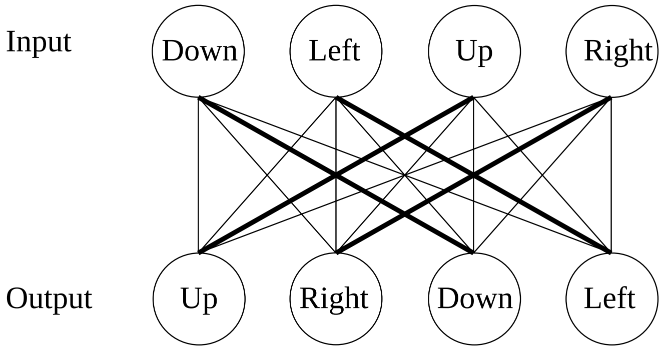


Abbildung 4.3: Gewünschte Struktur mit veränderter Input-Kodierung (1)

Einen Schritt weiter sind wir bei dem ANN in Abbildung 4.4 gegangen:

1 0 1 0 bedeutet "up",

0 1 0 0 bedeutet "right",

0 0 1 1 bedeutet "down",

1 0 0 1 bedeutet "left".

Wie man hier sehen kann, ist die Kodierung überlappend gestaltet (d.h. eine Eins an einer bestimmten Stelle kommt bei mehreren Input-Kodierungen vor), doch solange sie eindeutig ist und keine Obermenge von einer anderen ist, versteht das ANN auch diese Inputwerte korrekt.

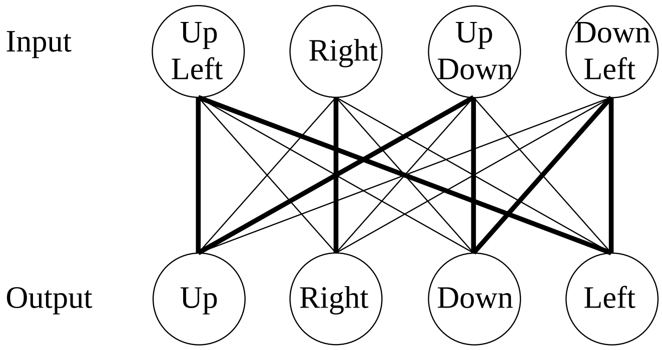


Abbildung 4.4: Gewünschte Struktur mit veränderter Input-Kodierung (2)

Die Abbildung 4.5 auf Seite 30 zeigt die Überlebensrate der Slaves bei unterschiedlicher Anfangsenergie (lifetime) und unterschiedlicher Input-Kodierung. Da wir die Überlebensrate mit dem Lernverhalten assoziieren können, sehen wir, daß ein Verändern der Kodierung keinen Einfluß auf die Lernfähigkeit der Slaves hat.

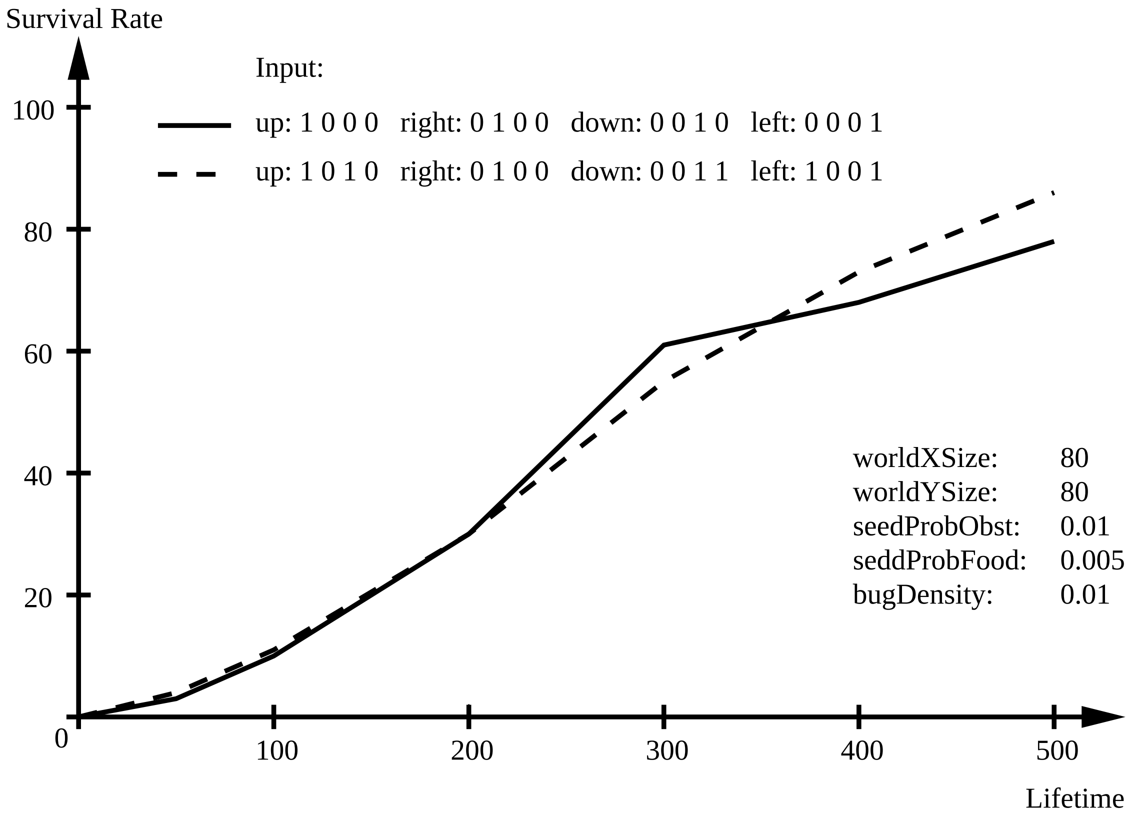


Abbildung 4.5: Durchschnittswerte mehrerer Programmdurchläufe

4.7 Screenshot

Auf Seite 32 ist ein Screenshot des Programms zu sehen. Nach dem Programmstart erscheinen zunächst drei Kontrollfenster mit folgenden Titeln:

- ProcCtrl
- ModelSwarm
- ObserverSwarm

Das Fenster “ProcCtrl” befindet sich links oben und dient zum Steuern des Programmablaufes. Im “ModelSwarm”-Fenster können die in 4.5 angegebenen Parameter vor dem Start geändert werden. “ObserverSwarm” bietet die Möglichkeit, die Welt jeweils erst nach “displayFrequency” Schritten neu zu zeichnen, da dies relativ langsam ist.

Mit dem Start des Programms erscheint die Welt im Fenster “Gr. 5 - Kommunikation in MAS”. Der Master sitzt in der Mitte der Welt, dargestellt durch einen hellblauen Punkt. Die Hindernisse sind rot, die Futterquellen grün dargestellt. Die weißen Punkte symbolisieren die Slaves. Bei diesem Programmdurchlauf sind offensichtlich einige gestorben, was an den Kreuzen zu erkennen ist. Die anderen noch lebenden haben gelernt, die Anweisungen des Masters zu verstehen.

Mit einem Rechtsklick auf einen Slave wird das Fenster “Bug” geöffnet, das detaillierte Informationen über ihn zeigt, die auch geändert werden können.

Die Terminal-Ausgabe des Programms (im Fenster “/bin/bash”) listet jeden Slave auf, zusammen mit dessen Position, Ziel und dem Input- und Outputmuster des jeweiligen ANNs.

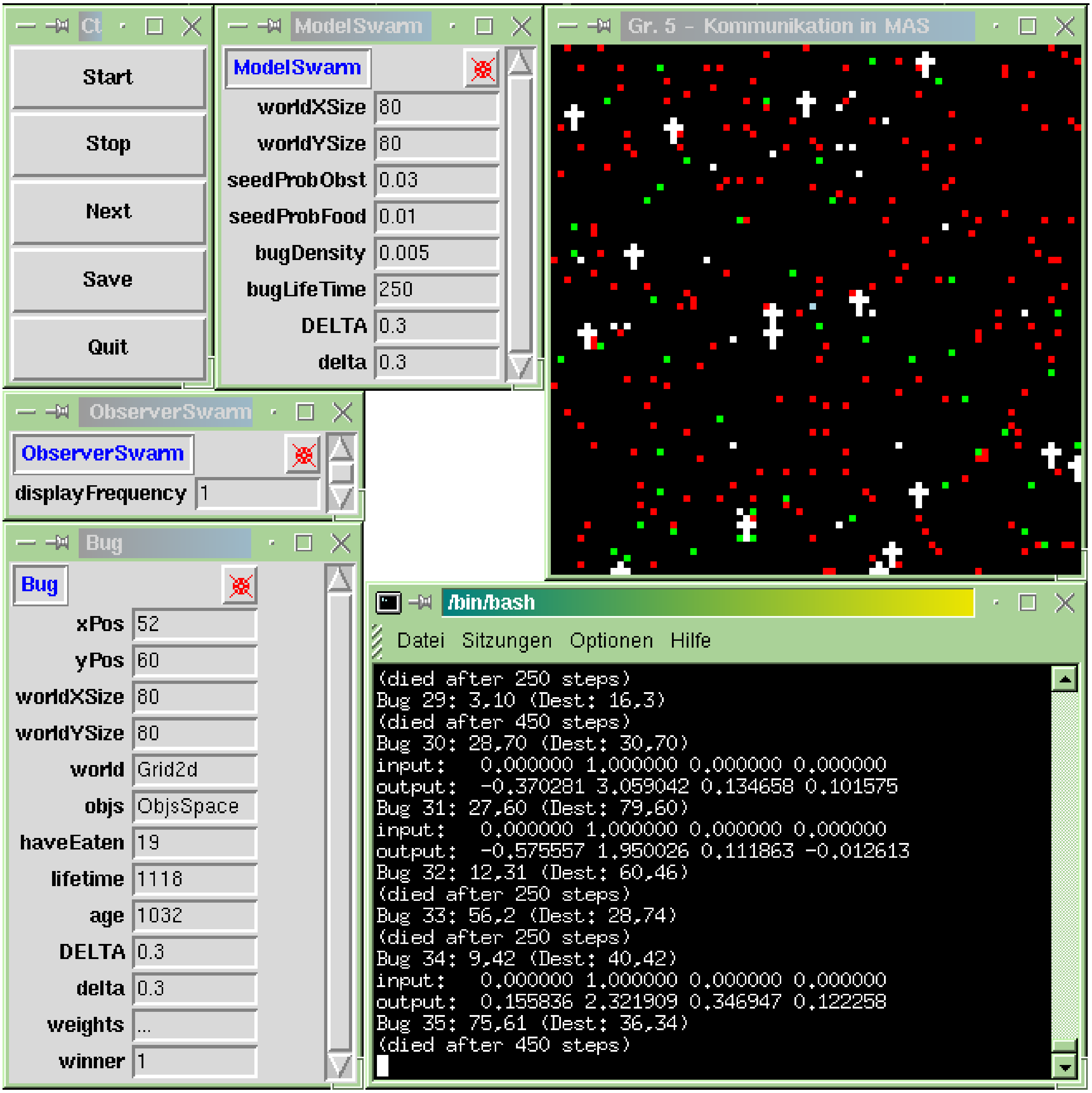


Abbildung 4.6: Screenshot

Anhang A

Protokolle

A.1 9. 3. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger
Teilnehmende Personen: R. Schwaiger, M. Noisternig
Protokollführer: M. Noisternig

Inhalt:

Grundsätzlicher Aufbau des zu erstellenden Dokuments (in LaTeX):

- Titelseite
- Abstract
- Einleitung zu den Fragen "Was ist ein Agent?" und "Was ist Kommunikation in diesem Zusammenhang?"
- Literaturverzeichnis (BIB-File)

Festlegung der Abgabe des Dokuments bis spätestens Montag 13:00 per E-Mail.

A.2 16. 3. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger
Teilnehmende Personen: R. Schwaiger, M. Amersdorfer, M. Noisternig
Protokollführer: M. Amersdorfer, M. Noisternig

Inhalt:

Besprechung des ersten Dokument-Entwurfs.

Zu erledigende Aufgaben:

- Unterschied zwischen Objekt (im Sinne von OO-Programmierung) und Agent
- Anwendungen für Agenten
- Klassifizierung für Agenten
- Muß ein Agent kommunizieren

A.3 23. 3. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger
Teilnehmende Personen: R. Schwaiger, M. Amersdorfer, M. Noisternig
Protokollführer: M. Amersdorfer, M. Noisternig

Inhalt:

Besprechung des bisherigen Dokumententwurfs.

Zu erledigende Aufgaben:

- Korrigieren des Dokuments
- Online-Referenzen für Webster
- Was sind Hardware-Agenten?
- Was sind Multi-Agenten-Systeme?
- Schematische Darstellung eines Agenten, mit internem Status, Zustaende, Schnittstellen, etc.
  Bzw: wie kann man einen Agenten graphisch darstellen?
- Kapitel Kommunikation
  - Welche Ansaetze gibt es?
  - Was ist KQML/KIF, kleiner Überblick und Unterschiede
  - Einsatz von ANN, GA, Fuzzy Logic in Kommunikation
  - Was ist das ISO-OSI Schichtenmodell und wie können Kommunikationsmodelle (natuerliche Kommunikation) darauf abgebildet werden?
    Bzw: Beispiel für natürliche Kommunikation (zwischen Menschen) in OSI-Modell zuordnen versuchen (Netzwerkperspektive)
  - Message-Passing (Bringschuld) - Blackboarding (Holdschuld)

A.4 13. 4. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger
Teilnehmende Personen: R. Schwaiger, M. Amersdorfer, M. Noisternig
Protokollführer: M. Amersdorfer, M. Noisternig

Inhalt:

Besprechung des bisherigen Dokumententwurfs.

Zu erledigende Aufgaben:

- Kapitel MAS erweitern (u.a. Unterschied zwischen MAS - DPS)
- Referenzen zu Kapitel 2.1 - Kommunikations-Mechanismen
- Begriff "Hardware-Agenten" im Auge behalten
- Schematische Darstellung eines Agenten:
  wie kann man interne Zustände graph. darstellen?
  Welche Symbole lassen sich finden?
- ISO-OSI-Referenzmodell: Vergleich Literatur "Artificial Intelligence - A Modern Approach"
- Zusammenfassung des Artikels "Einführung in ANN"
- Wo macht es Sinn, ANN, GA, Fuzzy Logic bei Kommunikation einzusetzen?

A.5 20. 4. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger
Teilnehmende Personen: R. Schwaiger, M. Amersdorfer, M. Noisternig
Protokollführer: M. Amersdorfer, M. Noisternig

Inhalt:

Besprechung des bisherigen Dokumententwurfs.

Erläuterungen zu ANNs.

A.6 4. 5. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger

Teilnehmende Personen: R. Schwaiger, M. Amersdorfer, M. Noisternig
Protokollführer: M. Amersdorfer, M. Noisternig

Inhalt:

Besprechung des Projekt-Vorschlages und Festlegung auf:

- Projekt bezüglich einem Agenten "Slave", der mittels eines ANN lernt, Richtungsanweisungen eines Agenten "Master" befolgen zu können, um so in einem Labyrinth das gewünschte Ziel "Futter" zu finden.
- Möglichkeiten der Steuerung:
  - Nachfahren - Slave bleibt immer in der Umgebung des Masters
  - Ferngesteuert - Slave reagiert auf Anweisungen eines sich nicht im Labyrinth befindenden Masters

  Einigung auf Methode "Ferngesteuert"
- Verwendung der bestehenden Simulationssoftware *Swarm* [19]

Zu erledigende Aufgaben:

- Erläuterungen bezüglich des Projekts im Dokument
- Näheres Überdenken der einzelnen Komponenten

A.7 18. 5. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger
Teilnehmende Personen: R. Schwaiger, M. Amersdorfer, M. Noisternig
Protokollführer: M. Amersdorfer, M. Noisternig

Inhalt:

Besprechung von und Einführung in Swarm
Zu erledigende Aufgaben:

- Weitere Ausführungen der Theorie
- Praktischer Einsatz von Swarm

A.8 1. 6. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger
Teilnehmende Personen: R. Schwaiger, M. Amersdorfer, M. Noisternig
Protokollführer: M. Amersdorfer, M. Noisternig

Inhalt:

Besprechung des bisherigen Dokumententwurfs.

Zu erledigende Aufgaben:

- Reinforcement-Learning
- Praktischer Einsatz von Swarm

A.9 15. 6. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger
Teilnehmende Personen: R. Schwaiger, M. Amersdorfer, M. Noisternig
Protokollführer: M. Amersdorfer, M. Noisternig

Inhalt:

Besprechung des bisherigen Dokumententwurfs.
Nähere Erläuterungen zu ANNs.

Zu erledigende Aufgaben:

- Praktischer Einsatz von Swarm

A.10 29. 6. 1999, 12:30 - 13:00

Ort: Universität Salzburg, Institut für Computerwissenschaften und Systemanalyse, Büro von R. Schwaiger
Teilnehmende Personen: R. Schwaiger, M. Amersdorfer, M. Noisternig
Protokollführer: M. Amersdorfer, M. Noisternig

Inhalt:

Nähere Erläuterungen zu ANNs und deren praktischen Einsatz im Projekt.
Vorbereitungen für die Präsentation.

Zu erledigende Aufgaben:

- Praktischer Einsatz von Swarm
- Präsentation vorbereiten

Literaturverzeichnis

[1] S. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach.* Alan Apt, prentice-hall ed., 1995.

[2] Merriam-Webster, *http://www.m-w.com/dictionary.htm.*

[3] T. W. Dictionary, *http://elza.lpi.ru/.*

[4] M. R. Genesereth and S. P. Ketchpel, *Software Agents.* Standford University.

[5] M. P. Singh, *Agent Communication Languages: Rethinking the Principles.* North Carolina State University, 1998.

[6] M. Belgrave, *What are agents?* http://www.ee.mcgill.ca/∼belmarc/UAA.what_are_agents.html.

[7] S. Franklin and A. Graesser, *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents.* University of Memphis, Insititute for Intelligent Systems, http://www.msci.memphis.edu/∼franklin/AgentProg.html.

[8] P. Csurgay, *Intelligent Agents for Telecom Applications.* ITEM/NTNU, http://www.item.ntnu.co/∼csurgay/eunice/index.htm.

[9] B. of Rural Sciences, *http://www.brs.gov.au/greenhse/kyoto/hood2.html.* 1998.

[10] Software-Agents-Group, *http://agents.www.media.mit.edu/groups/agents/.*

[11] C. Ranze and J. Mueller, *Ueber den Einsatz von Agenten in Umweltanwendungen.* http://www.informatik.uni-bremen.de/∼kcr/paper/UI97/version5.html, 1998.

[12] G. M. P. O'Hare and N. R. Jennings, *Foundations of Distributed Artificial Intelligence.* wiley-interscience publication ed.

[13] B. Moulin and B. Chaib-draa, *An Overview of Distributed Artificial Intelligence.* Wiley Interscience, 1996.

[14] T. Finin, *http://www.csee.umbc.edu/kse/.* University of Maryland Baltimore County.

[15] W. Bergt, *http://members.aol.com/WBLexikon/lexstart.htm.* 1999.

[16] R. Rojas, *Theorie der neuronalen Netze - Eine systematische Einfuehrung.* Springer-Verlag, 1993.

[17] W. Klimesch, *Vorlesungsunterlagen: Biologische Grundlagen der Psychologie I.* University of Salzburg, 1999.

[18] D. D. Olmsted, *History and Principles of Neural Networks.* http://www.neurocomputing.org/history.htm, 1998.

[19] R. Burkhart *et al.*, *The Swarm Simulation System.* http://www.santafe.edu/projects/swarm/.

[20] L. P. Kaelbling and M. L. Littman, *Reinforcement Learning: A Survey.* http://www.cs.washington.edu/research/jair/volume4/kaelbling96a-html/rl-survey.html.