Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Sven Tennstedt

Genese sozialen Verhaltens in Multiagentensystemen durch genetische Verfahren

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sven Tennstedt

Genese sozialen Verhaltens in Multiagentensystemen durch genetische Verfahren

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

Betreuender Prüfer : Prof. Dr. rer. nat. Kai von Luck
Zweitgutachter : Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 28. August 2007

Sven Tennstedt

Thema der Bachelorarbeit

Genese sozialen Verhaltens in Multiagentensystemen durch genetische Verfahren

Stichworte

genetische Lernverfahren, genetische Programmierung, Optimierung, Fitness, Multiagentensysteme, soziales Verhalten

Kurzzusammenfassung

In dieser Arbeit wird ein Multiagentensystem mit einer simulierten Umgebung aufgebaut, in der Roboter Gegenstände, sogenannte Attraktionen, suchen und miteinander tauschen können. Diese Arbeit geht der Frage nach, ob genetische Programmierung ein geeignetes Mittel darstellt, ein Setting von Agentenverhalten zu erzeugen, das grundlegende Formen des sozialen Verhaltens zeigt. Zu diesem Zweck wurde ein genetischer Optimierer und eine simulierte Welt in Form eines Rasters entwickelt, in der sich Roboter und Attraktionen befinden können. Experimente mit diesem System sollten klären, ob sich eher Settings aus kommunikativen Agenten oder eher welche aus Einzelgängern durchsetzen. Die Entwicklung der Software und die Versuchsdurchführung sind in der Arbeit dokumentiert.

Sven Tennstedt

Title of the paper

Genesis of social behavior in multi-agent systems with evolutionary computing

Keywords

evolutionary computing, genetic programming, optimizing, fitness, multi-agent systems, social behavior

Abstract

In this thesis a multi-agent system with a simulated environment is developed. Robots and objects, so-called attractions, can be situated in this environment. These robots are able to search for attractions and to deal in it with each other. This thesis go further into the question, whether genetic programming represents a suitable instrument, to produce a setting of agentbehavior that shows a fundamental form of social behavior. For this purpose a genetic optimizer and a simulated raster-world are developed. Experiments with this system should find out, if either settings of communication-joyful agents or settings of selfish agents prevail. Software development and experiments are documented here.

meiner Freundin Karen
und Frau H.

Danksagung

Meiner Freundin Karen danke ich dafür, dass es sie gibt, und dafür, dass sie mich moralisch unglaublich unterstützt hat.

Ich möchte Kai von Luck herzlich für die unvergleichliche Betreuung danken. Er hat mich gefordert, aber auch zum Durchhalten ermutigt. Durch seine Bereitschaft mir in den Semesterferien zur Verfügung zu stehen, konnte es mir gelingen, diese Bachelorarbeit so zügig und damit pünktlich zur Bewerbungsfrist für den Masterstudiengang abzuschließen.

Ein dickes Dankeschön an meine tapferen Kritiker Wanja Menzel, Paul Litzbarski und Fabian Bohlmann.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Ihren Ursprung haben Multiagentensysteme in der verteilten künstlichen Intelligenz. Sie wurde in der Anfangszeit ausschließlich als Problemsolver verstanden und eingesetzt um Approximationen zu gegebenen formalen Problemen zu erstellen. Ein typisches Beispiel dafür ist das Traveling Salesman Problem.

Heutzutage werden Multiagentensysteme vor allem zum Simulieren von Modellen der Realität, wie z. B. Fischschwärme[1] verwendet, dieser Bereich läuft unter dem Schlagwort „Self organisation of social systems“. Außerdem werden Multiagentensysteme in Form mobiler Roboter eingesetzt, die Aufgaben kooperativ lösen sollen, wie z. B. in dem von der EU geförderten Projekt Swarm-Bots[2]. Die Swarm-Bots sind einfache mobile Roboter mit jeweils einem Arm als Werkzeug, die lernen sollten wie sie gemeinsam Aufgaben lösen können, welche für einen einzelnen Roboter nicht zu lösen wären. Zu diesem Zweck können sie sich auch physisch aneinander koppeln.

1.1. Problemstellung und Zielsetzung

Für diese Arbeit stand der ct-bot (Abbildung 1.1) Pate, der ein Projekt des ct-Magazins aus dem Heise Verlag[3] ist. Er ist ein einachsiger Roboter, der in seiner Grundausstattung schon diverse Sensoren besitzt und durch die offene Architektur des Microcontrollerboards über Erweiterungsmöglichkeiten verfügt. Z. B. war es angedacht ihn mit einer intelligenten Mini-Kamera auszustatten. Mit dieser Kamera wäre es ihm möglich über Farbsegmentierung farbige Objekte erkennen zu können, außerdem war geplant an ihm ein Zigbee Funkmodul anzuschließen, um mit anderen ct-bots Kontakt aufzunehmen. Letzteres wurde bereits umgesetzt und in einer anderen Bachelorarbeit[4] zur Übertragung von Lokalisationsdaten verwendet.

Der Gedanke war, dass es mit dieser Ausstattung möglich sein sollte ein System aufzubauen in dem eine Gruppe von ct-Bots gemeinsame Aufgabenstellungen bewältigen können.

[1] Hemelrijk und Kunz (2003)
[2] Swarm-Bots und Mondada u. a. (2002)
[3] Heise Zeitschriften Verlag GmbH & Co. KG (2006)
[4] Burka (2007)

Die grundlegende Idee war die des Teamwork und zur weiteren Entwicklung hat ganz entscheidend der Robocup 2006 in Bremen beigetragen. Beim Robocup treten so genannte Hardware Agenten, allgemein Roboter genannt, aber auch Software Agenten gegeneinander im Fussballspiel an.

Abstrakt betrachtet handelt es sich beim Robocup, aber auch bei der Gruppe von ct-bots, um Multiagentensysteme (siehe Kapitel 2.2).

Ein Multiagentensystem setzt sich aus einer Menge von Hardware oder Software Agenten, einer Umgebung in der sie agieren können und einer Kommunikationsebene zusammen.

Der Agentenbegriff beschreibt in der Informatik eine Einheit (Software oder Hardware), die auf gewisse Weise autark und selbstständig, getrieben von Bedürfnissen, agiert.

Zurück zum Grundgedanken der Gruppenarbeit für ct-bots. Da diese Roboter über einen Transportschacht verfügen, liegt es nahe sie etwas einsammeln zu lassen, das sie später miteinander tauschen könnten. So ließ sich eine niedrige Ebene des sozialen Verhaltens erzeugen. Das Zusammenspiel des Verhaltens der einzelnen Roboter im Vorwege zu planen erscheint nicht viel versprechend. Daher stellte sich die Frage, ob so ein Setting von Verhaltensmustern über genetische Verfahren erzeugt werden kann.

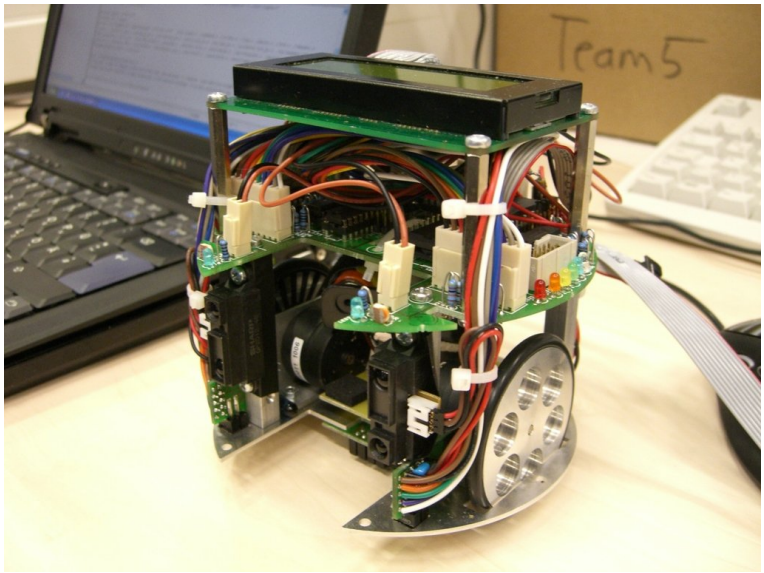


Abbildung 1.1.: ct-bot

Mit diesem Bild als Ausgangspunkt soll die Aufgabe allgemeiner formuliert werden und die ct-bots gegen eine Simulation ausgetauscht werden, um etwaige Hardwareprobleme bei dieser Themaik außer acht lassen zu können.

Ziel ist es ein Setting in einem Multiagentensystem zu erzeugen, in dem soziales Verhalten aus den Handlungen der einzelnen Agenten als emergente Eigenschaft hervortritt. Es soll ermittelt werden, in wie weit genetische Verfahren dazu geeignet sind ein Setting zu erzeugen, dass aus Verhaltensmustern besteht, die eine primitive soziale Struktur hervorbringen (siehe *Genese sozialen Verhaltens*). Dazu ist es notwendig, eine Softwareumgebung mit einem Simulator und der Basis für genetische Programmierung zusammen zu stellen oder zu entwickeln, in der die Versuchsläufe stattfinden können.

Genese sozialen Verhaltens Als soziales Verhalten soll angesehen werden, wenn die Handlungen der einzelnen Agenten sich gegenseitig beeinflussen und zu einem gewissen Teil von der Kommunikation untereinander abhängen, so dass sich dadurch das Gefüge einer primitiven sozialen Struktur entwickelt. Es wird hier als das Verhalten von Individuen definiert, die zwar ihre eigenen Ziele verfolgen, aber dazu mit anderen Individuen in Kontakt treten (müssen), also kommunizieren, und mit ihnen verhandeln. Die Bezeichnung soziales Verhalten orientiert sich zwar am Sinn von Max Webers[5] „soziales Handeln“, dennoch ist es als eine Metapher zu verstehen, die hervorheben soll, dass emergente[6] Eigenschaften und Phänomene im System beobachtet werden sollen. Mit anderen Worten, es soll eine Genese, eine Entstehung, dieser emergenten Eigenschaften und Phänomene stattfinden.

Auf die Simulation von sozialen Prozessen der Gesellschaft wird hier nicht eingegangen.

1.2. Kurzfassung der Arbeit

In den ***Grundlagen*** (S. 15) wird erklärt was es mit dem Agentenbegriff (S. 15) auf sich hat, auf welche Weise dieser Term in der Informatik angewendet wird und dass dieser eine andere Sicht als üblich auf Software ist. Von den Agenten geht es zu den Multiagentensystemen (S. 20), die für verschiedene Aufgaben eingesetzt werden und ganz unterschiedliche Charakteristiken aufweisen können.

Mit diesen Grundlagen wird in der ***Analyse*** (S. 30) im einzelnen die Beschaffenheit der Aufgabe Untersucht und heraus gearbeitet was für die konkrete Umsetzung des Ziels benötigt wird. Hier wird sich zeigen, dass eine direkte Umsetzung des ct-bot Szenarios nicht notwendigt ist, um die formulierte Aufgabe zu Untersuchen. Das Szenario läßt sich einfacher gestalten.

[5] Weber (2005)

[6] Emergenz ist das „Erscheinen“ von Phänomenen auf der Makroebene eines Systems, die erst durch das Zusammenwirken der Subsysteme (die Systemelemente auf der Mikroebene) zustande kommen. (Wikipedia (2007))

Diese Gedanken werden in ***Entwurf und Realisierung*** (S. 45) zur Entwicklung der geeigneten Software verwendet. Anschließend werden mit dieser Software Versuche (S. 59) durchgeführt und ausgewertet.

Zum Abschluss werden die Ergebnisse in der ***Zusammenfassung*** (S. 68) besprochen und bewertet und ein Ausblick für weitere Möglichkeiten gegeben.

Zeitgleich entstand die Arbeit von Christian Maschmann[7], die thematisch mit dieser Arbeit verwandt ist. Es fand ein reger Austausch an Wissen und Erfahrungen statt, vor allem zu den Grundlagen. Daher können in einigen Bereichen Überschneidungen mit seiner Arbeit auftreten.

[7] Maschmann (2007)

2. Grundlagen

Dieses Kapitel wird in die Themen *Agenten* und *Multiagentensysteme*, sowie *genetische Lernverfahren* einführen. Es werden alle relevanten Kenntnisse vermittelt, die zum Aufbau der angestrebten Aufgabe notwendig sind. Im Einzelnen werden die grundlegenden Eigenschaften von *Agenten* beleuchtet und näher auf den Unterschied zwischen Software und Hardware Agenten eingegangen. Anschließend werden verschiedenen Ansätze zum Aufbau des Verhaltens von Agenten dargestellt.

Die für die Agenten betrachteten Aspekte finden sich bei den *Multiagentensystemen* wieder, in denen sich eine beliebig große Menge von Agenten begegnen und je nach Art des Multiagentensystems auf verschiedenen Weise zusammenwirken können. Wichtig bei Multiagentensystemen ist die Ebene der Kommunikation, auf die am Ende des Abschnittes genauer eingegangen werden wird.

Ein anderer Bereich sind die *genetischen Lernverfahren*. Ausgehend von der Biologie wird die Grundidee *genetischer Lernverfahren* beschrieben. Im nächsten Schritt wird auf die *genetische Programmierung* eingegangen. Dabei werden unter anderem die Methoden der *Selektion*, sowie *genetische Operatoren*, wie das Kreuzen und die Mutation von Erbgut erläutert.

2.1. Agenten

In unserer Gesellschaft werden Agenten mit Personen assoziiert, die im Auftrag einer anderen Person oder der Gesellschaft handelt. Dabei verfolgt dieser Agent bestimmte Ziele, die durch seinen Auftrag bestimmt werden. Sehr bekannt sind Agenten für jedwede Art von Künstlern, die die Interessen des Künstlers vertreten und Verträge abschließen oder neue Engagements heran ziehen.

Ein weiteres Beispiel wäre ein Makler, der ein Objekt im Auftrag eines Verkäufers verkauft. In diesem Fall hätte dieser Agent einen konkreten Auftrag, den er bestmöglich erfüllen soll, nicht wie im vorherigen Beispiel, in dem der eigentliche Auftrag mehr Spielräume zur Interpretation der Qualität der Ausführung betreffend läßt.

Andererseits kann auch der Mitarbeiter eines Unternehmens als ein Agent bezeichnen werden. Er besetzt eine bestimmte Position im Unternehmen, durch die er eine gewisse Menge an Aufgaben zugeordnet bekommt, die er im Rahmen dieser Position selbstständig und eigenverantwortlich ausführt.

Agenten der Informatik werden für den herkömmlichen Verbraucher im Internet sichtbar z.b. bei ebay der Bietagent, der automatisiert für den Bieter bis zu einem bestimmten Betrag bei einer Auktion mitsteigert.

Diese Beispiele vermitteln schon eine gewisse Vorstellung von dem Agentenbegriff, zeigen aber auch, wie vielseitig er angewandt werden kann. Es wäre daher hilfreicher diesen Begriff näher zu klassifizieren und die speziellen Eigenschaften die einen Agenten kennzeichnen heraus zu stellen.

Im Softwarebereich beschreibt der Agentenbegriff eine ganz andere Sicht auf Software als es Objekte und Module tun. Um den Begriff einzugrenzen orientiert sich diese Arbeit an Definitionen aus Ferbers *Multiagentensysteme*[1] und Wooldridges *MultiAgent Systems*[2].

So sind nach Ferber Agenten physikalische oder virtuelle Entitäten, die folgendes auszeichnet:

Dass sie

- die Fähigkeit besitzen in einer Umwelt zu agieren,
- mit anderen Agenten kommunizieren können,
- getrieben von einer Vielzahl von Neigungen sind,
- eigene Resourcen besitzen,
- ihre Umgebung wahrnehmen können,
- von dieser Umgebung jeweils nur eine partielle Repräsentation besitzen,
- Fähigkeiten besitzen und Services anbieten können,
- sich selbst reproduzieren können,
- ihr Verhalten in Richtung der Erfüllung ihrer inneren Ziele tendiert, indem sie ihre Ressourcen, Fähigkeiten nutzen, und abhängig von ihren Erkenntnissen, ihrer Repräsentation und der Kommunikation ist.

[1] Ferber (2001)
[2] Wooldridge (2002)

Abbildung 2.1.: Agent in seiner Umgebung. Agent nimmt seine Umwelt mit seinen Sensoren wahr und verändert seine Umwelt über seine Aktionen. (Wooldridge (2002) S. 16)

Für die Zielsetzung dieser Arbeit kann der Punkt der eigenen Reproduktion ausgenommen werden und das Anbieten von Services für andere Agenten wird keine direkte Bedeutung bekommen.

Aus den Punkten der Aufzählung geht jedoch hervor, dass Agenten Neigungen und Ziele besitzen, sich in einer Umwelt befinden, die sie partiell wahrnehmen können. Um ihre Ziele zu verfolgen, haben sie die Möglichkeit Aktionen auszuführen, mit denen sie ihre Umwelt beeinflussen können und sie können mit anderen Agenten kommunizieren. Damit wird die einleitend gegebene eher allgemeine Beschreibung mit konkreten Aspekten besetzt. In der Informatik kann man den Begriff noch weiter eingrenzen.

Agenten können grob in folgende Kategorie eingeteilt werden:

Hardwareagenten (mobile Agenten) sind Agenten, die in der realen Welt agieren, also z.B. ein Roboter, Flugzeug oder Auto. Diese Agenten besitzen einen physischen Körper (embodied Agents) und können über Sensoren mit ihrer Umwelt in Kontakt treten.

Eine prominente Veranstaltung, bei der mobile Agenten zum Einsatz kommen, ist der bereits erwähnte weltweite Robocup, bei dem mobile Agenten im Team gegeneinander Fussball spielen (Abbildung 2.4 S. 22).

Softwareagenten agieren hingegen in einer künstlichen Umgebung und besitzen keinen physischen Körper. Sie sind, wie der Name schon sagt, reine Software. Sie können über Softwareschnittstellen mit ihrer Umwelt und anderen Agenten in Kontakt treten.

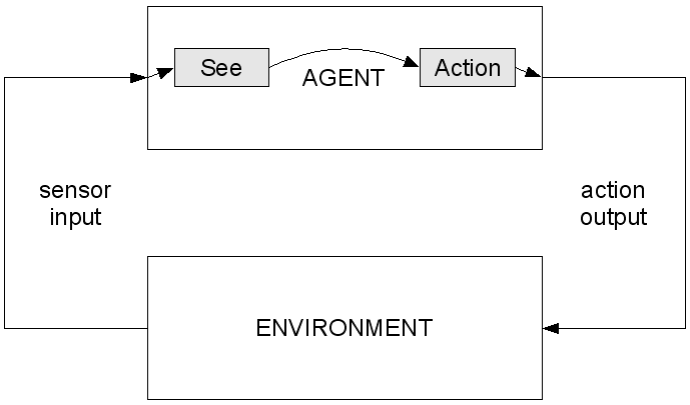


Abbildung 2.2.: Reaktiver Agent. Der Zustand seiner Wahrnehmung („See“) ist direkt mit einer Handlung („Action“) gekoppelt. (Wooldridge (2002) S. 34)

Der Begriff Umwelt ist in diesem Zusammenhang sehr interessant, da von reiner Software gesprochen wird. Wenn zum besseren Verständnis noch einmal das Beispiel des Bietagenten herangezogen wird, so ist dort die Umwelt des Agenten das Auktionshaus eBay. So wird auch schnell deutlich, was partielle Repräsentation der Umwelt bedeutet. Der Agent sieht nur eine Auswahl von Auktionen, genauer gesagt nur eine Auktion. Für die Erfüllung seines Ziels reicht das aus. In Abbildung 2.1 (S. 17) ist schematisch dargestellt, wie ein Agent mit seiner Umwelt interagiert. Ein Agent nimmt seine Umwelt über Sensoren wahr, wertet diese aus und wird darauf hin Aktionen ausführen, die auf die Umwelt wirken.

Aber warum kennt der Agent nicht die gesamte Umwelt? Einmal ist es unmöglich, dass dem Agenten jederzeit alle Daten seiner Umwelt zur Verfügung stehen. In diesem Zusammenhang sei hier auch das Naive Physics Manifesto von Pat Hayes[3] erwähnt, in dem er alltägliche Phänomene formal erklärt. Demnach ist eine vollständige Beschreibung des aktuellen Zustands der Umwelt nicht notwendig, um erfolgreich Handeln zu können. Wenn ein Mensch eine Tasse Tee trinkt, braucht er sich weder über die physikalischen Gesetze Gedanken machen, noch benötigt er Kenntnis über die Position aller Atome, um die Tasse bewegen und zum Mund führen zu können. Ein Mensch interagiert mit seiner Umwelt über mehrere Abstraktionsebenen hinweg. Der Grad und Art der Abstraktion kann von Aufgabe zu Aufgabe unterschiedlich sein, je nachdem wie viele Informationen notwendig sind, um die Aufgabe erfolgreich ausführen zu können.

Wenn Agenten sich in einer Umwelt befinden die sie über Sensoren wahrnehmen können, werden sie auch *situierte Agenten* genannt, im Gegensatz zu Agenten, die nur über Kommunikation andere Agenten bemerken.

Es gibt verschiedene Arten von Agenten, die unterschiedlichen Ansätze und Sichtweisen

[3] Hayes (1978)

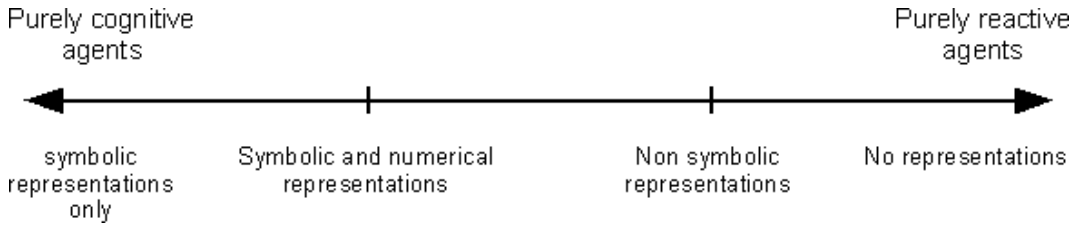


Abbildung 2.3.: Die Unterscheidung zwischen kognitiv und reaktiv führt zu dieser zweckmäßigen Achse, um die kognitive Kapazität eines Agenten zu ermessen, damit er komplexe Aufgaben bewältigen und seine Aktionen planen kann. (Ferber (2001) S. 20)

aufweisen. Im Anschluss wird sich zeigen, dass eine starre Trennung in der Praxis eher ungünstig ist, dennoch werden diese Ansätze zum besseren Verständnis einzeln aufgeführt.

Deliberative Agenten (Reasoning) In der symbolischen KI[4] besitzen Agenten über eine *symbolische* Repräsentation der Umwelt in Form von logischen Theoremen. Aus ihren Sensordaten erschließen sie sich über logische Schlüsse weitere Informationen über den Zustand ihrer Umwelt. Ihre Ziele verfolgen sie gegebenermaßen, indem sie die internen Theoreme beweisen und damit Handlungspläne erzeugen. Dieser Ansatz wird auch deduktiv (Schlussfolgernd) denkende Agenten[5] genannt. Eine Erweiterung dieses Ansatzes nennt Wooldridge die praktisch denkenden Agenten[6].

Allgemein besitzen deliberative Agenten eine mehr oder weniger komplexe interne Repräsentation ihrer Umwelt, verfügen über einen aktuellen Stand ihres Zustand, bzw. erschließen sich weitere Informationen und haben so etwas wie Erinnerung. Sie verfolgen ihre Ziele indem sie Handlungspläne erzeugen und diese immer wieder dem aktuellen Wissensstand des Zustandes ihrer Umwelt anpassen.

Daraus ergeben sich recht komplexe und umfangreiche Agenten, die aufwändige Algorithmen benötigen und einen hohen Ressourcenverbrauch bezüglich der Rechenleistung und des Speicherverbrauch haben.

Reaktive Agenten Wie es schematisch in Abbildung 2.2 (S. 18) dargestellt ist, reagieren reaktive Agenten aufgrund von äußerlichen Reizen und haben keine logische Einheit, die pro aktiv ihre Handlungen plant. Zudem besitzen sie im Grunde keine interne Repräsentation ihrer Umwelt, höchstens eine Art numerische Darstellung der Umwelt. Ihr Wissen über spezielle Eigenschaften des Zustands ihrer Umwelt wird in Form von Parametern ablegen. Die interne „Verdrahtung“ ihrer Reaktionen stellt das Modell ihrer Umwelt dar. Dadurch entstehen softwaretechnisch sehr leichtgewichtige Agenten.

[4] die symbolische künstliche Intelligenz bezieht ihren Namen daraus, dass. . .
[5] Wooldridge (2002) S. 47ff „Deductive Reasoning Agents“
[6] Wooldridge (2002) S. 65ff „Practial Reasoning Agents“

Sehr populär ist hier die Subsumptionarchitektur von Brooks[7], bei der die Reaktionen des Agenten in einer Prioritätenliste geordnet sind. Jede Reaktion wird über eine bestimmte Konstellation von Sensorwerten ausgelöst, Reaktionen mit höherer Priorität verdrängen dabei solche mit niedrigerer Priorität. Sogenannte Überlebensfunktionen wie z. B. die Kollisionsvermeidung bekommen hohe Prioritäten. Diese Verdrängen dann alle anderen Aufgaben, die das eigentliche Ziel des Agenten verfolgen, wie z. B. das Fahren zum nächsten Zielobjekt.

Wie anfangs schon erwähnt ist eine scharfe Trennung dieser Agententypen unpraktikabel. Wie aus der Abbildung 2.3 (S. 19) zu sehen, ist es praxisnäher je nach Einsatz des Agenten eine Mischung aus kognitiven Fähigkeiten und reaktiven Eigenschaften zu betreiben und danach zu ermessen, in welcher Art eine interne Repräsentation der Umwelt notwendig ist.

2.2. Multiagentensysteme

Ein Multiagentensystem (MAS) besteht aus einer Menge von Agenten. Einige der in Abschnitt 2.1 vorgestellten Eigenschaften von Agenten machen erst dann Sinn, wenn mehr als ein Agent in einem System existiert. Daher sind Eigenschaften wie die Kommunikation oder das Anbieten von Services ausschlaggebend für ein Multiagentensystem. Ursprünglich kommen Multiagentensysteme aus der *Verteilten künstlichen Intelligenz* und wurden als Problem Solver[8] eingesetzt. Heutzutage spannt sich der Begriff weiter auf und steht für ein breites Spektrum von Aufgaben und Einsatzgebieten. Allgemein besitzt ein MAS folgende Eigenschaften[9]:

- Es hat eine Umgebung (environment), E, diese Umgebung ist ein Raum, der in der Regel ein Volumen besitzt.
- Es gibt eine Menge von Objekten, O. Diese Objekte sind situiert, d. h. zu einem gegebenen Zeitpunkt ist jedes Objekt mit einer bestimmten Position in E assoziiert. Diese Objekte sind passiv, sie können von den Agenten wahrgenommen, erschaffen, zerstört und modifiziert werden.
- Es besitzt eine Gruppe von Agenten, A, spezielle Objekte ($A \subseteq O$), die die aktiven Entitäten des Systems repräsentieren.
- Es beinhaltet eine Menge von Relationen, R, die die Objekte (Agenten eingeschlossen) miteinander verbinden.

[7] Wooldridge (2002) S. 90
[8] Ferber (2001) S. 24ff
[9] Ferber (2001) S. 11

- Es stellt eine Menge von Operationen zur Verfügung, Op, die es den Agenten aus A ermöglicht Objekte aus O wahrzunehmen, zu produzieren, konsumieren, transformieren und manipulieren.
- Es verfügt über Operatoren mit der Aufgabe die Anwendung dieser Operationen (Op) abzubilden und die Reaktion der Welt auf diesen Modifikationsversuch.

Das bedeutet, ein MAS zeichnet sich dadurch aus, dass eine Menge von Agenten in einer Umgebung eingebettet sind und in Relationen zu einander stehen. Letzteres ist das, was ein MAS maßgeblich von einer Umwelt mit nur einem Agenten unterscheidet. Die Agenten sind zu Aktionen fähig, die die Umgebung verändern. Konkret können die Agenten Objekte in der Umgebung wahrnehmen, erschaffen und manipulieren, aber insbesondere können sie sich gegenseitig erkennen und auf einer vom MAS abhängigen Ebene miteinander kommunizieren.

Die Ebene der Kommunikation ist entscheident für die Art des Multiagentensystems. Sie können aher in folgende Kategorien eingeteilt werden:

Rein kommunizierendes MAS Sollte die Menge der Agenten A gleich der Menge der Objekte O sein, so handelt es sich um ein rein kommunizierendes MAS. In diesem Fall stellen die Relationen zwischen den Agenten ein Netzwerk dar und dementsprechend sind die Agenten ausschließlich auf Kommunikation ausgerichtet. Da keine Umwelt außerhalb dieses Agentennetzes existiert, können die Agenten sie nicht wahrnehmen. Als Umgebung dienen für sie die anderen Agenten im System, von denen sie nur einen Teil kennen.

Rein situiertes MAS Es verfügt über situierte Agenten, die sich in einer Umgebung befinden, die allgemein einen metrischen Raum darstellt. Rein situierte Agenten verfügen laut Ferber[10] zusätzlich über die Eigenschaft von einer Überlebens- bzw. Zufriedenheitsfunktion getrieben zu werden und ihr Verhalten auf Erfüllung dieser Funktion zu richten. Zudem besitzen sie Fähigkeiten, die sich ggf. reproduzieren lassen. Die Agenten können über ihre Fähigkeiten der Wahrnehmung und des Handelns den Zustand des Systems verändern, haben jedoch selbst keine oder nur eine vage Vorstellung von ihrer Umgebung. In einem rein situierten MAS kommunizieren die Agenten nicht direkt miteinander, sondern können ausschließlich über ihre Aktoren und Sensoren miteinander in Kontakt treten. Ein einfaches Beispiel dafür wäre ein MAS mit *reaktiven Agenten*.

Zwischen *rein kommunizierenden* und *rein situierten* Multiagentensystemen gibt es keine klare Trennlinie. Einige legen mehr Wert auf die Kommunikation, andere betonen die Situiertheit, beide besitzen aber zur gleichen Zeit Eigenschaften des anderen Systems.

[10] Ferber (2001) S. 12ff

Abbildung 2.4.: Robocup „4 Legged League“

Über das Maß der Interaktionsmöglichkeiten und der Kooperation zwischen den Agenten können Multiagentensysteme weiterhin in drei Kategorien eingeteilt werden:

Koexistenz Die Agenten sind in einer Umgebung und agieren autonom ohne Kenntnis der anderen Agenten und ohne Kommunikation zwischen ihnen. Als Beispiel sind hier Schwarmsysteme zu nennen. Ein prominente Aufgabe für ein Schwarmsystem ist die Approximation des *Traveling Salesman Problem*.

Kooperation Die Agenten kennen sich gegenseitig und kommunizieren miteinander. Sie verfolgen gemeinsame Ziele und arbeiten zusammen.

Konkurrenz Die Agenten verfolgen zum Teil unterschiedliche und sogar konkurrierende Ziele. Pläne eines Agenten können von gegnerischen Agenten gestört werden. Beispielhaft seien hier ein weiteres Mal die Fußball spielenden Roboter des Robocups erwähnt, die in Mannschaften gegeneinander antreten (Abbildung 2.4).

Das Beispiel des Robocups zeigt, dass es auch hier Mischformen gibt. Die Mannschaft untereinander ist kooperativ, doch steht sie in Konkurrenz mit der gegnerischen Mannschaft, die das Ziel verfolgt selbst zu gewinnen. Die gegnerische Mannschaft wird keinerlei Informationen über die Positionen der eigenen Spieler verraten, ebenso wenig wird sie beabsichtigt den Ball abgeben.

Koexistiernde Multiagentensysteme werden hier außer Acht gelassen, da sie eher in den Bereich der Schwarmsysteme und zu den Problemsolvern gehören und den Rahmen der Arbeit sprengen würden.

Kooperierende und konkurrierende MAS bestimmen jedoch im Besonderen die Qualität der Kommunikation und die Form der Bereitschaft, einen anderen Agenten zu unterstützen.

In kooperierenden Umgebungen kann ein Agent davon ausgehen, dass ein anderer Agent ihm bereitwillig hilfreiche Informationen übergibt, während in konkurrierenden Umgebungen eher das Gegenteil der Fall ist. Ganz gleich um welche Art Umgebung es sich handelt, ist die Kommunikation in diesen Systemen ein wichtiger Punkt. Agentenplattformen tauschen in der Regel Nachrichten einer hohen Abstraktionsebene aus, anders als es bei Verteilten Systemen sonst üblich ist, die z. B. mit CORBA oder allgemein über RMI[11] Mechanismen arbeiten.

So tauschen Agenten Nachrichten aus, die sich von einfachen Methodenaufrufen unterscheiden. Es handelt sich eher um eine intentionierte Kommunikation, die keine direkte Wirkung bei dem angesprochenen Agenten haben muß. Sie hat Einfluss auf die internen Zustände der Agenten, was wiederum eine veränderte Handlungsweise zur Folge haben kann.

Standardisierungen haben in diesem Bereich besonders die Open Agent Architekture[12] (OAA) und FIPA[13] geschaffen. Während die OAA eine fertige Implementation einer Agentenplattform darstellt, ist FIPA eine reine Spezifikation. Beide Ansätze sind ähnlich. Beide stellen die Kommunikation in den Vordergrund, so können beide Plattformen als eine Art Kommunikations-Middleware um Multiagentensysteme aufzubauen, verstanden werden.

Die FIPA spezifiziert u. a. die ACL (Agent Communication Language), die als Basis für die Agentenkommunikation dient. Sie orientiert sich an der Sprechakt-Theorie[14], die von den Philosophen Austin und Searle begründet wurde. Wie oben schon beschrieben, haben Nachrichten in einem Agentensystem nicht mehr die Bedeutung einfacher Methodenaufrufe, sondern folgen einer Intention. Die Anlehnung der ACL an die Sprechakt-Theorie macht dies deutlich. Sie spezifiziert wie Agenten Intentionen deutlich machen können und wie sie darauf antworten können.

2.3. Genetische Lernverfahren

Genetische Lernverfahren orientieren sich an dem Prinzip der Evolultion wie sie durch Charles Darwin[15] erforscht wurde. Im Zentrum steht das Genom, das so genannte Erbmate-

[11] Remote Method Invocation

[12] Open Agent Architecture (2007)

[13] FIPA (2007)

[14] Die Grundlagen der Theorie hat Austin (1962) geschaffen, sein Schüler John Searle hat diese Theorie später in seinem Buch „Speech Acts“ ausgebaut. Sprechakte verändern danach die Welt im physikalischen Sinne, da diese immer einer Intention folgen und bei dem Empfänger eine Reaktion auslösen. Genauer ist es zum Beispiel in Jung (2001) (S. 23ff) zu lesen.

[15] Darwin (1859)

rial. Als Bauplan in jeder Zelle eines Lebewesens zu finden, bestimmt es, wie das Lebewesen aussieht und welche Fähigkeiten es besitzt. Durch *Mutation*, d. h. zufällige Veränderungen des Genoms, *natürliche Selektion* und das *Kreuzen* von Erbmaterial passen sich die Lebewesen einer Art an sich wandelnde Umweltbedingungen über Generationen an. Dies geschieht mit dem Ziel ihre Überlebenschancen zu verbessern. Darwin nannte dieses Prinzip „survival of the fittest“ („das Überleben des bestangepassten“). Innerhalb einer Art variiert das Erbmaterial durch spontane Mutation und durch die zweigeschlechtliche Fortpflanzung, also der oben genannten Kreuzung. Daraus entstehen innerhalb einer Art Individuen mit geringfügig physischen Veränderungen und ggf. neuen Fähigkeiten. Wenn eine oder mehrere dieser Veränderungen eine verbesserte Anpassung an die Umweltbedingungen darstellt, so wird dieses Individuum eher überleben und sich damit eher fortpflanzen können. Im Gegensatz dazu bedeutet eine Veränderung, die keine Verbesserung darstellt oder sogar eine Verschlechterung, dass dieses Individuum eine weniger gute Überlebenschance besitzt und sich mit geringerer Wahrscheinlichkeit erfolgreich fortpflanzen wird. Dieser Prozess wird als natürliche Selektion bezeichnet und verläuft in der Natur kontinuierlich.

Übertragen auf Computer stellen genetische Lernverfahren eine heuristische Suche in einem *Lösungsraum* dar. Die oben beschriebenen Vorgänge werden als Metaphern übernommen. Das Genom ist hierbei z. B. ein Programm oder ein Lösungsvektor, das im Laufe des künstlichen Evolutionsprozesses über Selektion, Mutation und Kreuzung immer weiter optimiert wird, bis es eine adäquate Lösung der Aufgabe darstellt.

Besonders geeignet sind diese Verfahren in Bereichen, in denen nicht bekannt ist, wie eine adäquate Lösung aussehen könnte und keine Algorithmen hergeleitet werden können.

Verwendung von genetischen Lernverfahren sind z. B. in den Arbeiten von Timo Storjohann[16] und Sven Lund[17] zu finden. Die folgende Beschreibung der Verfahren stützt sich auf diese beiden Arbeiten, sowie auf Jochen Heinsohns *Wissensverarbeitung*[18] und John Kozas Buch *Genetic Programming*[19].

Von der Natur wurden dafür die wesentlichen Begriffe dieses Prozesses übernommen. Das Genmaterial eines Individuums wird als **Genotyp** bezeichnet, der wie oben schon erwähnt, den Bauplan darstellt. Das entstandene Lebewesen stellt dagegen den **Phänotypen** dar. Aus einem Genotyp können somit verschiedene Phänotypen hervorgehen.

Im wesentlichen gibt es zwei verschiedene Strategien der künstlichen Evolution:

- genetische Algorithmen
- genetische Programmierung

[16] Storjohann (2005) S. 21ff
[17] Lund (2006) S. 21ff
[18] Heinsohn u. a. (2007) Kap. 4
[19] Koza (1992)

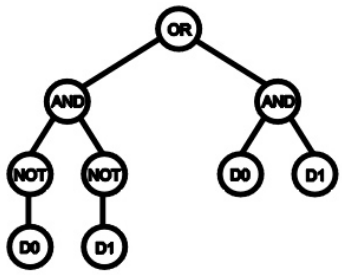


Abbildung 2.5.: Genom (Quelle: Storjohann (2005))

Diese Arbeit konzentriert sich auf die genetische Programmierung, da diese für die gegebene Aufgabe das geeignetere Verfahren darstellt. Entwickelt wurde die genetische Programmierung von John Koza und wird in seinem Werk *Genetic Programming: On the Programming of Computers by Means of Natural Selection* umfassend beschrieben. Entstanden ist sie aus den genetische Algorithmen, die 1975 von John Holland entwickelt wurden. Doch die genetischen Algorithmen stellen heutzutage eher einen Spezialfall dar, mit unnötigen Beschränkungen. Zum Beispiel verwenden die genetischen Algorithmen einen Lösungsvektor fester Länge. Dieser Vektor stellt den Genotypen dar und kann aus reellen Zahlen oder einem Binärmuster bestehen. Jedoch engt die feste Länge die Lösung auf eine vorab definierte Dimension ein, Lösungen anderer Dimensionen können damit nicht gefunden werden. Im folgenden liegt die Konzentration auf der genetischen Programmierung, wobei die grundsätzlichen Eigenschaften auch für die evolutionären Algorithmen gelten. Ausführlich werden die evolutionären Verfahren in *Wissensverarbeitung* erläutert oder in den beiden oben genannten Abschlussarbeiten.

2.3.1. Genetische Programmierung

Genotyp In der genetischen Programmierung stellen die Genotypen ein ausführbares Programm dar, bzw. eine programmartige Struktur, die von einem Interpreter ausgeführt werden kann, im Gegensatz zu den genetischen Algorithmen, deren Genotypen Parameter einer Funktion darstellen. Repräsentationsarten von Genotypen in der genetischen Programmierung sind (Heinsohn u. a. (2007)):

- symbolische Ausdrücke
- Zustandsautomaten
- Befehlssequenzen

In Abbildung 2.5 (S. 25) sieht man die Repräsentation als symbolischen Ausdruck, in Form einer Baumstruktur. In *Genetic Programming*[20] werden Lisp Ausdrücke verwendet um Genotypen darzustellen, aber genauso ist es möglich Bäume aus Objekten in einer objektorientierten Sprache zu bilden.

Phänotyp Der Phänotyp beschreibt das Verhalten des Genotypen, also die Ausprägung. Problematisch ist, dass ähnliche Phänotypen nicht unbedingt auf ähnliche Genotypen zurückzuführen sind. Im Gegenteil ein und der selbe Phänotyp kann auf verschieden strukturierte und verschieden große Genotypen zurück zu führen sein.

Ablauf des Algorithmus

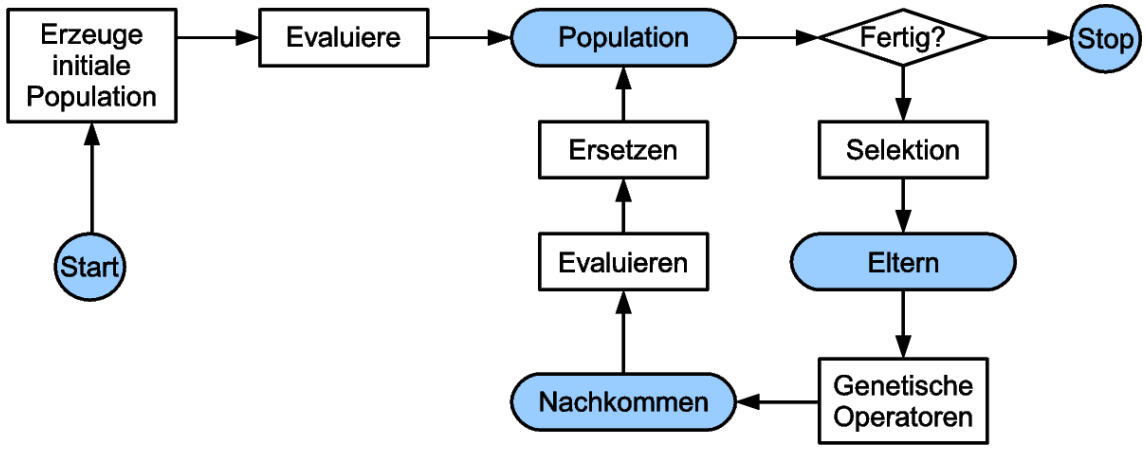


Abbildung 2.6.: Algemeiner Ablauf evolutionärer Algorithmen (Quelle: Heinsohn u. a. (2007))

Im wesentlichen arbeitet ein evolutionärer Algorithmus so wie es die Abbildung 2.6 zeigt. Ausgehend von dieser Abbildung werden die grundlegenden Begriffe des Verfahrens erklärt.

Population Die Population stellt eine Menge an Lösungsvorschlägen dar, bestehend aus unterschiedlichen Genotypen. Zu Beginn wird eine erste Population aus rein zufällig erzeugten Genotypen erstellt. Diese Population ist die erste Generation.

Diese Population wird dann evaluiert, d. h. es wird die Qualität der einzelnen Lösungen ermittelt, was bedeutet, dass alle Genotypen werden auf ihre Fitness hin untersucht. Auf das

[20] Koza (1992)

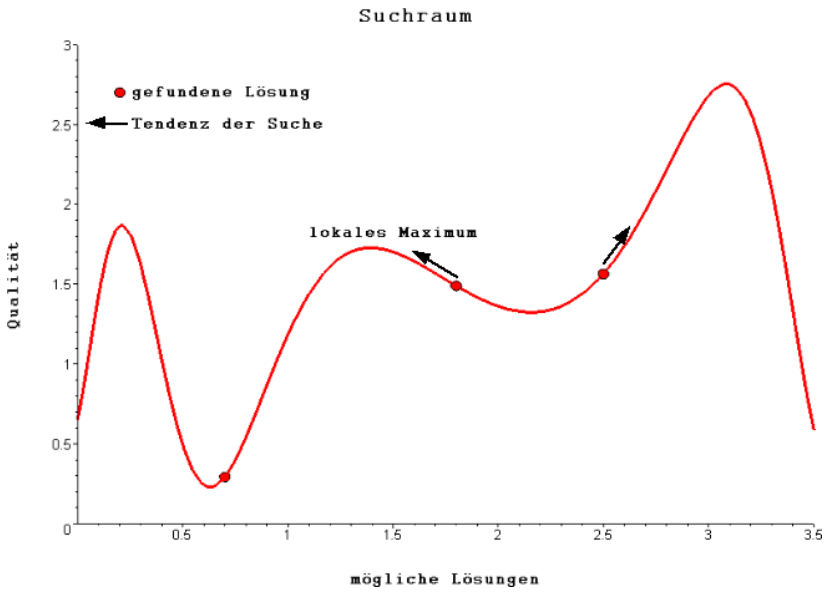


Abbildung 2.7.: Gefundene Lösungen im Suchraum. Das Weiterverfolgen einiger Lösungen kann in ein lokales Maximum führen.

Thema Fitness wird später näher eingegangen. Wird das vorher definierte Qualitätsziel erreicht, so wird das Verfahren beendet und eine Lösung ist gefunden. Andernfalls wird eine Nachfolgegeneration zu der aktuellen Population erzeugt.

Fitness Die Fitness ist ein Maß für die Qualität einer Lösung. Ermittelt wird die Fitness mittels einer Fitnessfunktion. Als Beispiel stellt man sich einen Roboter vor, der auf einer schwarzen Linie entlang fahren soll. Als ein Qualitätsmaß der Lösung könnte die Zeit genommen werden, in der der Roboter unter sich die schwarze Linie registrierte und die zurückgelegte Strecke. Die Fitnessfunktion ist das wichtigste Element auf der Suche nach einer guten Lösung. Eine Fitnessfunktion mit unzureichender Qualitätsbeschreibt kann zu ungewollten Effekten führen. Würde man in dem oben genannten Beispiel die Strecke aus der Fitnessfunktion heraus nehmen, könnte auch als gute Lösung gelten, dass der Roboter auf der Linie stehen bleibt oder sich im Kreis dreht.

Selektion Mit Hilfe von verschiedenen Selektionsverfahren werden möglichst erfolgreiche Genotypen in die nächste Generation übernommen und aus ihnen mittels genetischen Operatoren Nachkommen gebildet. Zu diesen Verfahren zählt die *Elite Strategie* oder auch *Rankingmethode*, sowie das *Roulette-* und *Turnierverfahren*.

Bei der *Elite Strategie* werden alle Genotypen anhand ihrer Fitness nach sortiert, und die n fittesten Genome werden in die nächste Generation übernommen.

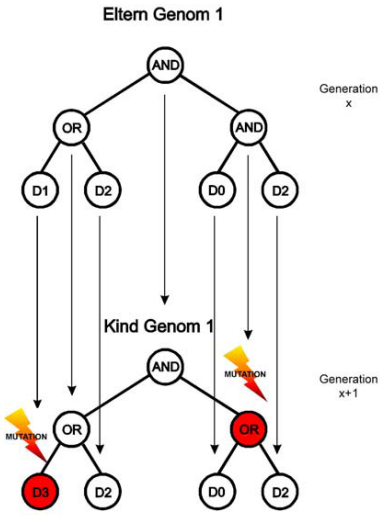


Abbildung 2.8.: Mutation von Genomen (Quelle Storjohann (2005))

Beim Rouletteverfahren wird für jedes Genom anhand ihrer Fitness und der kumulativen Fitness aller Genome eine relative Fitness ermittelt, die gleichbedeutend der Selektionswahrscheinlichkeit ist. Anschließend werden n Genome mittels eines Zufallsgenerators auswählt. Man kann es sich vorstellen wie ein Rouletterad, auf dem die Genome soviel Platz proportional zu ihrer Selektionswahrscheinlichkeit einnehmen. Die Kugel fällt nach dem Zufallsprinzip in einen dieser Bereiche. Sie trifft dabei mit höherer Wahrscheinlichkeit die Genotypen, die mehr Platz auf dem Rouletterad beanspruchen.

Anders beim Turnierverfahren, das unabhängig von der Fitness m Genotypen zufällig auswählt, vergleichbar mit einer Schöpfkelle, und daraus das fitteste in die nächste Generation übernimmt. Dieser Vorgang wird n mal wiederholt. Die „Schöpfkellengröße“ bewegt sich in der Regel zwischen 3 und 5 Individuen. Es kann zwar sein, dass dieses Verfahren nicht die fittesten Genotypen findet, jedoch ist durch die „Schöpfkelle“ ausgeschlossen, dass die schlechtesten selektiert werden.

Das erste Verfahren stellt sicher, dass sich die Fitness von Generation zu Generation nie verschlechtern kann, birgt jedoch die Gefahr, dass die Suche in einem lokalen Maximum hängen bleibt (Abbildung 2.7). Die anderen beiden Verfahren hingegen sorgen für eine breit gestreute Suche im Lösungsraum. Denn sie ermöglichen es auch weniger fitten Genotypen am Selektionsprozess teilzunehmen, was unter Umständen zu einem viel versprechenderen Anstieg im Lösungraum führen kann, als mit den Zeit fittesten Genotypen.

Über Parameter kann der jeweilige *Selektionsdruck* dieser Verfahren beeinflusst werden, mit anderen Worten die Chance für Genotypen mit guter Fitness in die nächste Generation übernommen zu werden bzw. wie schnell die mit weniger guter Fitness aus der Populati-

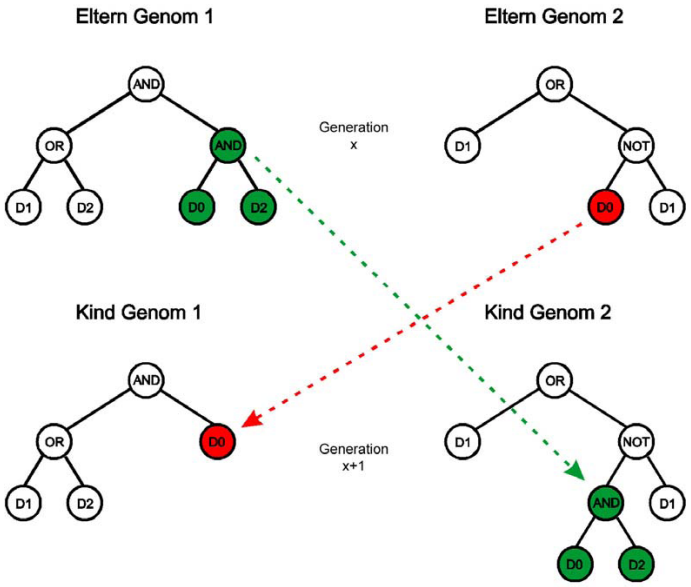


Abbildung 2.9.: Kreuzen von Genomen (Quelle: Storjohann (2005))

on verschwinden. Ein hoher Druck führt dazu, dass gefundene Anstiege im Lösungsraum schnell erklommen werden, kann aber dazu führen, dass der Algorithmus eher in einem lokalen Maximum hängen bleibt.

Genetische Operatoren dienen dazu das genetische Material zu verändern, d. h. im näheren Umkreis der schon gefundenen guten Lösungen zu suchen. Dabei werden aus den durch die Selektionsverfahren gefundenen Eltern mit einstelligen (Mutation: Abbildung 2.8 S. 28) oder zweistelligen (Kreuzen: Abbildung 2.9 S. 29) Operatoren Nachkommen gebildet.

Die Fitness dieser Nachkommen wird ebenfalls evaluiert und dann ersetzen sie die alte Generation. Ausgehend davon, **dass die Fitness eines Individuums absolut ermittelbar ist**, wird die Fitness der vorhandenen Individuen nicht erneut evaluiert.

Nach diesem Prozess wird wieder überprüft, ob die gewünschte oder sogar die bekannte maximale Qualität erreicht wurde. Wenn dies nicht der Fall ist, beginnt der Prozess von neuem mit der Selektion und eine neue Generation Genotypen wird geschaffen und evaluiert.

3. Analyse

In der Analyse werden die Gedanken aus der Zielsetzung konkretisiert und weiter ausgebaut. Es wird hier der Weg bereitet, um eine Anwendung zu entwerfen und zu realisieren, die den Anforderungen der Zielsetzung entspricht. Dabei bauen die Gedanken auf die vorher geschaffenen Grundlagen auf.

Die folgenden Überlegungen sind in informaler Sprache gehalten.

Als erstes werden grundlegende Gedanken zum Eingrenzen der Problemstellung dargelegt und die Komplexität der Aufgabe erschlossen (Abschnitt 3.1). Anschließend wird genauer auf den Versuchsaufbau (Abschnitt 3.2) eingegangen und die Eigenschaften des Simulators (Abschnitt 3.3) ermittelt, da dieser die virtuelle Maschine der Agenten bereitstellt und maßgeblich den Versuchsaufbau bestimmt. Zum Abschluß wird auf den Aufbau der Genotypen (Abschnitt 3.4) eingegangen. Diese repräsentieren ein Set von Agentenverhalten (*Setting*).

Die Begriffe Genotyp und Setting sind äquivalent, werden allerdings in verschiedenen Kontexten benutzt. Der Begriff des Genotyps stellt die Perspektive der genetischen Programmierung dar und Setting die des Versuchsaufbaus.

3.1. Grundlegendes

Das Tauschkonzept aus der Zielsetzung sei hier noch einmal dargestellt.

Die Aufgabe soll mit Hilfe eines Simulators umgesetzt werden, um Probleme auszuschließen, die keinen Bezug zur eigentlichen Aufgabe haben und sich auf die Hardware zurückführen lassen.

Als Prototyp dient der ct-bot[1], ausgestattet mit Kamera- und Funkmodul. Er ist ein einachsiger Roboter mit einem zylinderförmigen Chassis. Ausgestattet ist er mit diversen Sensoren, wie z. B. Distanz- und Liniensensoren. Über Radencoder kann er seine Radbewegungen kontrollieren und über einen Maussensor die Länge der gefahrene Strecke erfassen. Zudem

[1] Heise Zeitschriften Verlag GmbH & Co. KG (2006)

besitzt er einen Transportschacht, in dem er ein Objekt der Größe einer Fotodose problemlos mitführen kann. Mittels einer Lichtschranke am Transportschacht kann er erkennen, ob er gerade ein Objekt mit sich führt.

Über das Kameramodul ist der Roboter fähig, farbige Objekte zu erkennen. Das Funkmodul ermöglicht es ihm, Kontakt mit einer Zentrale oder mit anderen Robotern aufzunehmen.

Die zur Verfügung stehenden Leistungsresourcen der Rechnereinheit des ct-bots ermöglichen nur den Aufbau eines reaktiven Systems. Das vom Heise Verlag zur Verfügung gestellte Framework orientiert sich an der Subsumption Architektur von Rodney Brooks[2].

Es sollte möglich sein, aus mehreren dieser Roboter ein System aufzubauen, in dem diese Roboter Objekte verschiedener Farben suchen und einsammeln können. Diese Objekte werden nachfolgend als *Attraktion* bezeichnet. Da die Roboter jeweils nur eine Attraktion transportieren können, brauchen sie eine Basis, in der sie diese dann sammeln können. Die Basis könnte ebenfalls farbig sein, damit der Roboter sie mit der Kamera aufspüren kann.

Die Roboter haben jeweils eine stärkere Neigung, Attraktionen einer bestimmten Farbe zu sammeln, daher sammeln die Roboter Attraktionen einer bestimmten Farbe bevorzugt ein als Attraktionen der jeweils anderen Farben.

Zusätzlich sollen sie über das Funkmodul miteinander in Kontakt treten können und auf diese Weise Tauschgeschäfte mit Attraktionen untereinander aushandeln.

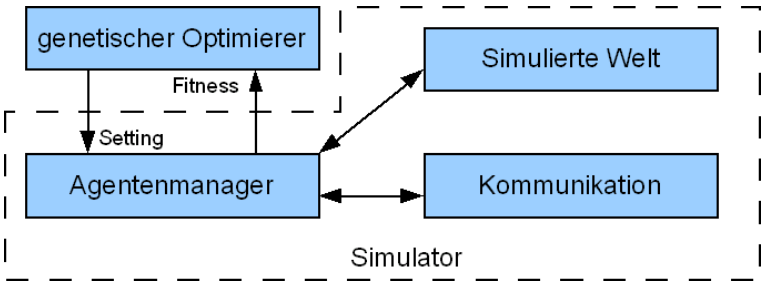


Abbildung 3.1.: Vier Module der Anwendung

Aus diesen Überlegungen ergeben sich vier Hauptbestandteile, aus denen die spätere Anwendung bestehen wird. Im Vordergrund steht hier der Simulator, der die Bereiche Agentenmanagment, Kommunikation und simulierte Welt umfasst. Der genetische Optimierer erstellt Populationen und übergibt dem Simulator *Settings*, deren Fitness der Simulator ermittelt und diese dem genetischen Optimierer zurück liefert. Dieses Konzept wird in Abbildung 3.1 mit den voraussichtlichen Relationen dargestellt.

Diese grundlegende Aufteilung soll als Orientierung für die folgende Analyse dienen.

[2] Wooldridge (2002) S. 90

3.2. Das Szenario

Ein *Szenario* besteht aus einem *Setting* und einer Welt, die sich aus einer Umwelt, einer bestimmten Anzahl positionierten Robotern und Attraktionen zusammensetzt. Wenn das Szenario als Zustand betrachtet wird, wird es als Situation bezeichnet. Ein Szenario besitzt eine bestimmte Qualität. Wie diese ermittelt wird soll hier theoretisch betrachtet werden. Als erstes wird beschrieben, wie eine Situation in eine Folgesituation überführt wird. Anschließend wird gezeigt wie hiermit die Qualität ermittelt wird.

Ein *Zustand* einer Welt zu einem bestimmten Zeitpunkt wird als Situation bezeichnet. Eine *Situation* wird durch die Positionen der Agenten, die Anzahl der verfügbaren und gesammelten Attraktionen, sowie die zum Tausch angeboten Attraktionen beschrieben.

Den Agenten steht eine Menge von Aktionen zur Verfügung. Diese Menge besteht aus Bewegungen und Sprechakten. Sprechakte haben keine physikalische Veränderung des Agenten zur Folge. Das Ausführen einer dieser Aktionen wird als Handlung bezeichnet. Eine Handlung besteht aus einem Handelnden und einer auszuführenden Aktion ($Handlung(handelnder \in Handelnde, aktion \in Aktionen)$).

Eine *Situation* S_i wird in eine *Folgesituation* S_j durch die Handlungen aller existierenden Agenten überführt ($\sum_{agent=1}^{n} h_{agent} \in Handlungen$) (Abbildung 3.2). Dieser Übergang entspricht dem Ausführen eines Simulationsschritts. Aus Sicht der Agenten ist eine Zeiteinheit verstrichen. Alle Aktionen benötigen eine Zeiteinheit zur Ausführung, woraus sich eine zeitdiskrete Simulationsumgebung ergibt.

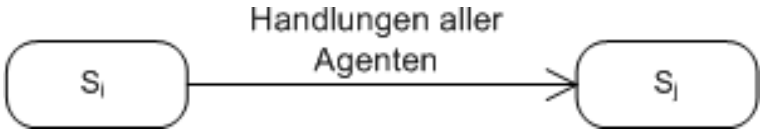


Abbildung 3.2.: Übergang von einer Situation in die Folgesituation

In der realen Welt gibt es mehrere Möglichkeiten für den Ausgang einer Handlung:

1. Die ausgeführte Aktion verändert den Zustand des ausführenden Agenten erwartungsgemäß.
2. Die ausgeführte Aktion verändert den Zustand des Agenten nicht erwartungsgemäß, d. h. durch eine Ungenauigkeit in dem Bewegungsapparat befindet sich der Agent an einer anderen Position als erwartet.
3. Der Agent ist gegen ein Hindernis gestoßen, so dass die Aktion nicht ausgeführt werden kann.

Der zweite Punkt wird in der Simulation nicht berücksichtigt, weil angenommen wird, dass diese Eigenschaft das Ergebnis der untersuchten Aufgabe prinzipiell nicht verändern würde. Aus diesem Grund werden physikalische Eigenschaften, wie z. B. das Rauschen von Sensoren oder Abweichungen beim Fahren einer Strecke, außer acht gelassen.

Wie oben beschrieben wird ein *Szenario* als Zustand betrachtet Situation genannt. Diese wird hier als Anfangssituation (S_1) bezeichnet.

Um die Qualität des Szenarios zu ermitteln, wird diese Anfangssituation mit dem Simulator in Folgesituationen überführt. Über i Schritte überführt der Simulator die Situation S_1 in die Situation S_i (Abbildung 3.3). Die Situation S_i besitzt eine gewisse *Wertigkeit*, die nach bestimmten Maßstäben ermittelt wird, wie z. B. die Anzahl gefundener und sich damit im Besitz von Agenten befindender Attraktionen. Diese Wertigkeit ist allerdings kein absoluter Faktor der eigentlichen Qualität bzw. dem *Potential* der Situation S_1.

Jeder Agent hat pro Zeiteinheit mehrere Möglichkeiten des Handelns, es gibt keine absolute Bedingung der Situation, aus der es sich ergibt, dass ein Agent eine bestimmte Handlung vollführen muss. Diese *nicht Determiniertheit* der Umgebung führt dazu, dass ein erneuter Durchlauf zu einem anderen Wert führen kann und sehr wahrscheinlich führen wird. Dieser Wert wird sich unter Umständen erheblich von dem vorherigen unterscheidet.

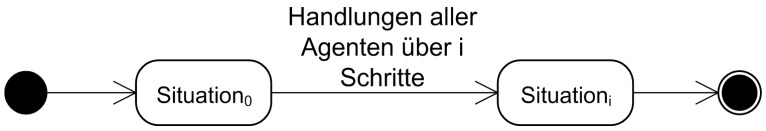


Abbildung 3.3.: Ermitteln der Wertigkeit eines Szenarios durch i Übergänge

Diese Eigenschaft macht es schwierig die eigentliche *Qualität* zu ermitteln. Angenommen es gibt nur zwei mögliche Folgesituationen pro Simulationsschritt, dann spannt sich schon nach vier Schritten ein Baum mit 16 Blättern auf. Unter Umständen besitzen diese 16 Endsituationen erheblich unterschiedliche Wertigkeiten. Allein in diesem Beispiel wäre es ziemlich teuer alle möglichen Endsituationen zu simulieren, da dies ja mit allen Individuen der Population durchgeführt werden müsste. Es wäre sinnvoller die Anzahl der Simulationen zu beschränken und einen Mittelwert über den besten und schlechtesten Wert der Situation zu bilden. Allerdings ist es auch hierbei schwierig vorher repräsentative Äste auszuwählen, da vorher nicht bekannt ist, welcher Ast zu einer repräsentativen Aussage der Situation führt. Somit ist es praktikabler n Simulationen einer Situation durchzuführen und zufällig die Äste auszuwählen. Die Wahrscheinlichkeit ist dann relativ hoch, dass für eine Situation mit einer niedrigen Qualität auch ein dementsprechend schlechter Wert ermittelt wird und genauso umgekehrt für Situationen mit einer hohen Qualität.

Da in die Simulation nicht aktiv eingegriffen werden kann, bedeutet dies konkret, anzunehmen, dass mehrere Simulationen eines Szenarios eine ausreichende Streuung verursacht.

Möglich wäre es die Anfangssituation eines Szenarios bei jedem Start leicht zu modifizieren, indem man die Startpositionen der Roboter verändert oder die Attraktionen an anderen Positionen platziert.

3.3. Der Simulator

Eine zentrale Bedeutung in dieser Arbeit besitzt der Simulator, mit dessen Hilfe die Fitness eines Settings ermittelt werden soll. Ein Teil des Simulator ist die simulierte Welt, die die Umwelt mitsamt der Attraktionen und der Roboter, die die Verkörperung der Agenten sind, simuliert. Sie wird in diesem Abschnitt als erstes betrachtet, anschließend werden die Eigenschaften der Kommunikation und des Agentenmanagers eingegangen. Die Eigenschaften der simulierten Welt sind eng verflochten mit den Fähigkeiten und Eigenschaften der Agenten. Auf diesen Aspekt wird unter Punkt 3.3.4 speziell eingegangen.

3.3.1. Simulierte Welt

Der erste Ansatz für den Simulator orientiert sich an der schon oben beschriebenen ct-bot Metapher.

Die Agenten sind einfache, fahrende Roboter mit einer Achse, sie können sich vorwärts und rückwärts bewegen und im Kreis drehen. Über einen Auffangschacht können sie Dinge einsammeln. Aus dieser Beschreibung wird deutlich, dass die Begriffe Agent und Roboter sich in ihrer Bedeutung überschneiden, daher sollen diese Begriffe hier mehr voneinander abgegrenzt werden. Wie in Abbildung 3.4 zu sehen, steht der Agent außerhalb der simulierten Welt, er kommuniziert mit seinem Körper, dem simulierten Roboter, um mit der Umwelt zu interagieren. Nachfolgend ist der Agent der Akteur und der Roboter die Simulation seiner Verkörperung.

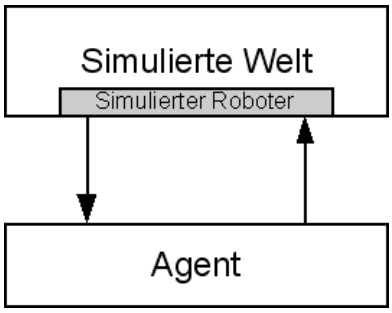


Abbildung 3.4.: Ein Agent kommuniziert mit dem simulierten Roboter in der simulierten Welt.

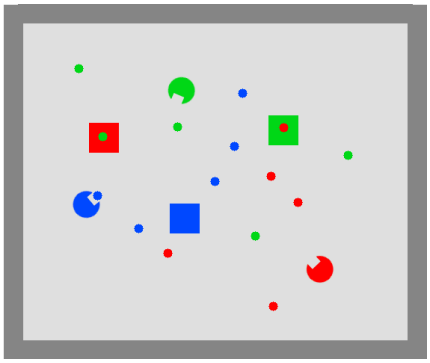


Abbildung 3.5.: Erster Ansatz für den Entwurf der Umwelt.

Agenten sollen fähig sein mit ihrer Umwelt, sowie mit anderen Agenten zu kommunizieren und interagieren. Ihre Umwelt nehmen sie über ein Satz an Sensoren wahr. Darunter fallen Sensoren die den Abstand zu anderen Gegenständen anzeigen und die angeben, ob ein farbiges Objekt in Sichtweite ist und welche Farbe es besitzt. Es ist nicht erforderlich, dass sich die Agenten untereinander erkennen können. Zudem sollte den Agenten ihr eigener Aufenthaltsort unbekannt sein.

Die Umwelt besteht sowohl aus Wänden und einer befahrbaren Ebene mit farbigen Attraktionen, die eingesammelt werden können, als auch aus Basisstationen. Die Umwelt bietet den Agenten die Möglichkeit, sich auf den Ebenen zu bewegen und die farbigen Attraktionen einzusammeln, wenn sie vorwärts auf sie zufahren. Ein Agent kann nur eine Attraktion zur Zeit transportieren. Die Attraktionen können abgelegt werden, wenn der Agent rückwärts fährt. In Abbildung 3.5 ist dieser Ansatz veranschaulicht. Die farbigen Rechtecke sind Basisstationen, die kleinen runden Flächen die Attraktionen und die großen die Roboter.

In den Basisstationen können die Roboter die Attraktion aus ihrem Transportschacht ablegen. Sie wird dann zu ihrem bisherigen Vorrat an Attraktionen hinzugefügt. Der Tauschvorgang könnte dann folgendermaßen ablaufen: Die Roboter sammeln in ihren Basisstationen Attraktionen, wenn sie tauschen möchten, so kommunizieren die Roboter über ihren Kommunikationskanal und handeln einen Tausch von Attraktionen aus. Die Attraktionen stehen nach dem Handel für den anderen Roboter bei der jeweiligen anderen Basisstation zum Abholen bereit.

Dieser Aspekt des Szenarios kann vereinfacht werden. Das Mitführen von Attraktionen und Ablegen auf den Basisstationen ist eine zusätzliche physikalische Eigenschaft, die für eine spätere Version der Simulation vorgesehen werden kann. Anstatt die Attraktionen in den Basisstationen ablegen zu müssen, damit sie zum eigenen Bestand zählen, werden Attraktionen automatisch eingesammelt, wenn sie in den Sichtbereich der Roboter geraten, und zu ihrem Bestand hinzu gezählt. Damit kann vorerst auf die Basisstationen verzichtet werden.

In diesem abgewandelten Szenario findet die Verhandlung an sich wie gehabt über den Kommunikationskanal statt, jedoch wechselt die Ware auf „magische“ Weise zu dem neuen Besitzer, ohne von ihm abgeholt werden zu müssen.

Diese Überlegungen führen zu den funktionalen Eigenschaften, die in der Tabelle 3.1 aufgeführt sind. Zum Beispiel gibt Tabelle 3.1 einen Überblick darüber welche Eigenschaften für die simulierte Welt unumgänglich, notwendig und welche wünschenswert sind. Eigenschaften in der selben Zeile zeigen den gleichen Aspekt in anderen Ausprägungen, z. B. ist es notwendig zumindest eine gridbasierte Welt zu besitzen, doch wünschenswert wäre eine kontinuierliche Welt. Ein anderes Beispiel ist, dass eine Kollisionserkennung unumgänglich für eine reibungslose Navigation durch die Welt ist, diese kann als eine Bumper-Metapher realisiert werden oder in Form eines Distanzsensor, der einen Abstand zum nächsten Gegenstand misst.

must	***should***	***nice to have***
	WELT	
grid		kontinuierlich
		Basisstationen
befahrbare und nicht befahrbare Felder		
nur korrekte Bewegungen der Objekte werden erlaubt		
grundlegende „physikalische“ Eigenschaften		reale physikalische Simulation
	ROBOTER	
Sichtradius		Sichtbereich vor Roboter
Bewegung vor / zurück / seitwärts	Bewegung vor / zurück / drehen	
		mehrere Geschwindigkeiten
Kollisionserkennung (Bumper)	Distanzsensor	
Attraktionen in Sichtweite werden dem eigenen Bestand zugerechnet		Transportschacht
	ATTRAKTIONEN	
farbig		
feste Position in der Welt		beweglich
einsammelbar		

Tabelle 3.1.: Funktionale Eigenschaften der simulierten Welt

Ein Spezialfall der simulierten Welt ergibt sich durch die Eigenschaften der Attraktionen. Wenn sie eine feste Position in der Welt haben und einsammelbar sind, dann verschwinden sie in dem Moment des Einsammelns aus der Umwelt-Simulation. Sie existieren nur noch als Parameter in den Agenten. Bewegliche Attraktionen würden die Simulation umfangreicher gestalten, da es dann Regeln bedarf, welche bestimmen, wann eine Attraktion bewegt wird und welche Auswirkungen dieses auf die anderen Objekte der Welt hat.

3.3.2. Kommunikation

Die Kommunikation stellt die Mittel zur Verfügung, damit die Agenten untereinander Verhandlungen zum Tausch von Attraktionen führen können. Denkbar wären hier verschiedene Metaphern. Einmal könnte es ablaufen wie in einem online Auktionshaus, vergleichbar mit eBay. Ein Agent bietet eine Attraktion zur Versteigerung an, alle anderen können Angebote einstellen. Das höchste Angebot bekommt den Zuschlag. Eine andere Metapher wäre die Ausschreibung. Ein Agent schickt an alle Agenten eine Anfrage für ein Angebot auf eine bestimmte Attraktion. Alle anderen Agenten können Angebote an den ausschreibenden Agenten schicken. Dieser Überprüft alle Angebote und entscheidet sich dann für eines.

Die Variante mit der Ausschreibung soll hier zum Einsatz kommen. Für diese Variante ist die Möglichkeit notwendig Punkt zu Punkt, sowie an einen Verteiler an alle Nachrichten zu verschicken, damit sich die Agenten untereinander Angebote zuschicken können. Um dies zu realisieren ist ein Vermittlungsmodul notwendig. Um Angebote zu erstellen und zu offerieren müssen verschiedene Arten von Nachrichten zur Verfügung stehen, z. B. die Möglichkeit Angebote anzufordern und zu unterbreiten, sowie Angebote zu akzeptieren und auszuschlagen. Die Tabelle 3.2 fasst diese Punkte zusammen.

must	***should***	***nice to have***
punkt zu punkt nachrichten		
Massen Verteiler		
Nachrichten in Form von Objekten		XML basiert
Nachrichtenklassen für „Angebote anfordern“, „Angebot unterbreiten“, „Angebot akzeptieren“, „Angebot ablehnen“		ACL konform

Tabelle 3.2.: Funktionale Eigenschaften der Kommunikation

3.3.3. Agentenmanager

Die Eigenschaften des Agentenmanagers zeigt die Tabelle 3.3. Notwendig ist das Übernehmen eines Settings und das erstellen der Agenten aus dem Setting. Außerdem muss er die Agenten beim Kommunikationsmodul anmelden und sie an die simulierten Roboter koppeln. Abschließend muss er den Simulationsprozess anstoßen können, um damit die Fitness feststellen zu können. Anschließend übergibt der Agentenmanager diese Fitness an den genetischen Optimierer.

must	***should***	***nice to have***
Erstellen von Agenten		
Aufnahme eines Settings durch den genetischen Optimierer		
Anmeldung der Agenten an Nachrichtenkanal und simulierte Roboter		
Anstoßen des Simulationsprozesses		
Rücklieferung der Fitness des Settings		

Tabelle 3.3.: Funktionale Eigenschaften des Agentenmanagers

Allgemein werden die Eigenschaften unter „Nice to have“ bei der Entwicklung außen vor gelassen, da diese zwar das Szenario komplexer gestalten würden, damit allerdings auch die Entwicklungsarbeiten aufwändiger werden ließen. Da in dieser Arbeit sichergestellt werden soll, dass das Szenario an sich funktioniert und angenommen wird, dass diese Eigenschaften voraussichtlich keinen prinzipiellen Einfluss auf das Gesamtergebnis haben würden, sollen diese zukünftigen Entwicklungen vorbehalten bleiben.

Hingegen werden die Eigenschaften unter „Should“ umgesetzt, da angenommen wird, dass sie die Aufgabestellung bereichern werden. So soll z. B. die Simulation es ermöglichen, dass der Roboter sich auch drehen kann, statt sich nur vor, zurück und seitwärts zu bewegen, wie es in ähnlichen Szenarien sonst der Fall ist.

3.3.4. Die Agenten und ihre Umwelt

Aus den besprochenen Eigenschaften der simulierten Welt geht hervor, dass Attraktionen keine Aktionsmöglichkeiten besitzen. Sie befinden sich unbeweglich auf Ebenen und können eingesammelt werden. Eingesammelte Attraktionen sind nicht mehr Teil der simulierten

Welt. Damit sind ihre Eigenschaften vollständig beschrieben, im Gegensatz zu den Eigenschaften der Agenten. Ihre Interaktionsmöglichkeiten mit der Umwelt bedarf einer genauen Differenzierung, aus diesem Grund wird in diesem Kapitel speziell auf Agenten eingegangen.

NEIGUNGEN	
1.	Hindernissen ausweichen
2.	möglichst viele Attraktionen einer bevorzugten Farbe besitzen

Tabelle 3.4.: Neigungen eines Agenten

Ein Agent besitzt die Neigungen aus Tabelle 3.4. Als Überlebensfunktion steht hier die Neigung Hindernissen auszuweichen. Sie dient ausschließlich dazu, dass der Agent Kollisionen vermeidet. Die zweite Neigung symbolisiert seinen eigentlichen Auftrag. Es ist überlegenswert, ob die Überlebensfunktion als ein fester Verhaltensbestandteil implementiert wird, der für alle Agenten gleich ist und damit unbeeinflusst von der genetischen Optimierung bleibt. Das hätte den Vorteil, dass dies den Suchraum einschränken würde.

AKTIONEN

Bewegungen

Aktion	Beschreibung
vorwärts	Bewegung in Blickrichtung
rückwärts	Bewegung entgegen Blickrichtung
Drehung rechts	Blickrichtung rechts herum rotieren
Drehung links	Blickrichtung links herum rotieren
stop	Roboter hält an

Sprechaktionen

Akt	Empfänger	Beschreibung
biete n Attraktionen der Farbe f zur Ausschreibung an	Alle	eine Ausschreibung wird an alle anderen Agenten verschickt
biete n Attraktionen der Farbe f_1 auf eine Ausschreibung a	Ausschreiber	ein Gebot wird an den Ausschreiber verschickt
akzeptiere Gebot von Bieter x	Bieter	Bieter erhält Zuschlag, Ware wird getauscht
Ausschreibung beendet	Alle	eine Nachricht zum Löschen einer Ausschreibung wird an alle verschickt, die Agenten können die Ausschreibung verwerfen

Tabelle 3.5.: Aktionen eines Agenten

Um ihr Ziel – die Neigung möglichst viele Attraktionen zu besitzen – zu erfüllen, dienen den Agenten die Aktionsmöglichkeiten, welche die in Tabelle 3.5 zusammen gefasst sind.

Zur Verfügung stehen den Agenten Aktionen zur Beeinflussung seiner Position in der Welt, hier die Vorwärts- und Rückwärtsbewegung, sowie die Drehbewegungen. Außerdem stehen Aktionen zum Ausführen von Verhandlungen zur Verfügung, die in der Tabelle unter Sprechaktionen aufgeführt sind. Mit diesen vier Sprechaktionen soll es möglich sein, dass Agenten Attraktionen zur Ausschreibung freigeben, Angebote der anderen Agenten abwarten und unter ihnen das für sie beste Angebot auswählen können. Jeder Agent kann seine Ausschreibungen zu jedem Zeitpunkt beenden, ob mit erfolgreichen Tausch oder ohne.

Um mit seiner Umwelt in Kontakt zu treten, besitzt der Agent die Sensoren aus Tabelle 3.6. Zu den Sensoren gehören externe, die Daten über die Umwelt bereitstellen, sowie interne, die interne Zustände des Agenten erfassen.

Die Umwelt, in der sich die Roboter und Attraktionen befinden, wird in Abbildung 3.6 skizziert. Dort sind die Wände und Ebenen als Raster dargestellt. Jede Rasterfläche kann ein Objekt aufnehmen, außer Wänden, auf denen sich kein Objekt befinden kann und unbefahrbar sind. Die genannte Abbildung zeigt ebenso eine Skizze einer Attraktion und eines Roboters. Sie werden dort auf einer befahrbaren Ebene dargestellt. Die Grundfläche des Roboters ist der des ct-bots nachempfunden.

SENSOREN	
Sensor	**Beschreibung**
Extern	
Distanz	liefert die Distanz zum nächsten Objekt vor dem Roboter (mit Maximum Grenze)
gefahrene Strecke (Encoder Schritte)	Angabe der gesamt gefahrenen Strecke
Farbe der gesichteten Attraktion	gibt die Farbe der zuletzt gesehen Attraktion an
Intern	
Anzahl Attraktionen der Farbe f	liefert den Status über die im Besitz befindlichen Attraktionen
Anzahl Attraktionen der bevorzugten Farbe	liefert den Status über die im Besitz befindlichen Attraktionen der bevorzugten Farbe
Anzahl Attraktionen der nicht bevorzugten Farben	liefert den Status über die Attraktionen, die sich im Besitz des Agenten befinden und dessen Farbe nicht zur bevorzugten Farbe gehören
Zuletzt gefundene Attraktion	gibt die Zeit an, die seit dem letzten Fund verstrichen ist

Tabelle 3.6.: Sensoren eines Agenten

Ein Roboter verfügt über Aktionsmöglichkeiten die in Abbildung 3.7 (links) dargestellt sind. Er soll die Möglichkeit besitzen, sich vorwärts und rückwärts zu bewegen, sowie nach links und rechts. Um ihn herum ist ein Sichtradius, in dem er Attraktionen erkennen kann. Alles außerhalb dieses Radius bleibt unerkannt. Der Sichtradius bewegt sich mit dem Roboter mit. Eine Vorwärts- oder Rückwärtsbewegung bzw. Seitwärtsbewegung versetzt den Roboter in das nächste Feld in Bewegungsrichtung.

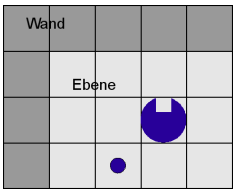


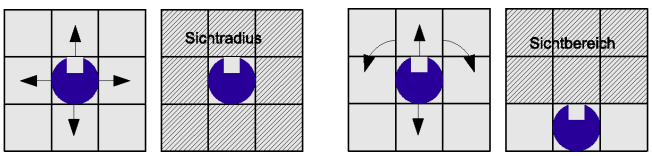
Abbildung 3.6.: Die Umwelt des Simulators. Die kleine blaue Kreisfläche ist die Attraktion, die große Kreisfläche ist der Roboter.

Die angestrebte Erweiterung zeigt Abbildung 3.7 (rechts), der Roboter kann sich drehen, anstatt sich seitwärts zu bewegen. Der Sichtbereich bleibt stets vor dem Roboter und bewegt bzw. dreht sich mit. Eine Drehbewegung wird den Roboter jeweils um 90 Grad auf dem Feld rotieren lassen. Eine Vorwärts- oder Rückwärtsbewegung versetzt den Roboter in das nächste Feld in Fahrrichtung des Roboters.

Außerdem verfügt der Roboter über einen Sichtbereich, der die Kamera simulieren soll. In diesem Bereich kann er Attraktionen erkennen. Der Sichtbereich befindet sich bei dem Modell mit Seitwärtsbewegungen in Form eines Sichtradius um den Roboter herum, bei der Implementierung mit Drehbewegungen vor dem Roboter, wie es in Abbildung 3.7 (a) und (b) skizziert ist. In der Realisierung kann die Größe des Sichtbereichs experimentell angepasst werden. Wie oben schon erwähnt, muss sich bei letzterer Implementierung der Sichtbereich mit dem Roboter mitdrehen.

Die externen Sensoren würden in diesem Fall wie folgt arbeiten:

Der *Distanzsensor* gibt die Anzahl an freien Ebenen vor dem Roboter an. Als Hindernisse



(a) einfache Bewegungsmöglichkeit mit Sichtradius

(b) erweiterte Bewegungsmöglichkeit mit Sichtbereich vor Roboter

Abbildung 3.7.: Bewegungsmöglichkeit und Sichtbereich eines Roboters

gelten Wände und andere Roboter, jedoch keine Attraktionen. Diese könnten natürlich auch als Hindernisse wahrgenommen werden, doch die Kollisionsvermeidung müsste überprüfen, ob eine Attraktion „den Weg versperrt“ oder ein wirkliches Hindernis. Zur Vereinfachung sollten deshalb Attraktionen für den Distanzsensor unsichtbar sein.

Der *Encoder* wird die Anzahl Rasterflächen wiedergeben, die der Roboter zurück gelegt hat. Mit Drehbewegungen kann für jedes „Rad“ ein Encoder vorgesehen werden und bei der Drehbewegung der eine Encoder runter und der andere rauf gezählt werden.

3.4. Genotypen und die Suche nach einer Lösung

In dem unter Punkt 3.2 beschriebenen Szenario wird zu der Optimierung eines einzelnen Agenten, die Optimierung einer Zusammenstellung von Agenten betrieben. Es reicht nicht, dass ein Agent so viele Attraktionen wie möglich einsammelt, wenn dabei die anderen Agenten nichts abbekommen. Ein Setting ist nur dann erfolgreich, wenn alle Agenten Attraktionen in ihren Besitz bringen können. Besonders erfolgreich ist es zudem, wenn die Agenten am Ende möglichst viele ihrer bevorzugten Attraktionen besitzen.

Diese Verschachtelung bedeutet, dass die Fitness eines einzelnen Verhaltens immer im Kontext der Umgebung zu sehen ist. Egoistische Verhalten sollten ein Setting nicht zum Erfolg führen können und mit der Zeit aus der Population verschwinden.

Die Genotypen sind auf oberster Ebene Zusammenstellungen (*Setting*) von Agentenverhalten. Die einzelnen Verhalten bilden keine festen Bausteine, sie sind ihrerseits aus Bedingungen und Aktionen zusammengesetzt. Im Abschnitt 2.3 des Grundlagenkapitels wurde angesprochen, dass Genotypen mutiert und gekreuzt werden, um neue Genotypen zu schaffen. Das setzt voraus, dass nach diesen Operationen wieder ein syntaktisch korrektes Individuum entsteht.

Auf oberster Ebene, die der Settings, gibt es in dieser Hinsicht wenig zu beachten. Egal welche Verhalten zusammen gestellt werden, bilden sie immer ein korrektes ausführbares Setting. Die einzige Bedingung ist hierbei, dass die Anzahl der Verhalten für ein Setting eingehalten wird.

Auf der Ebene der Verhalten wird es komplexer. Hier ist es notwendig eine eigene Sprache zu entwickeln, die so aufgebaut ist, dass der genetische Optimierer mit jeder Operation, die er darauf ausführt wieder ein syntaktisch korrektes Verhalten erhält.

In *Genetic Programming*[3] beschreibt John Koza wie Genotypen Repräsentiert werden können. Er propagiert dort eine hierarchische Struktur aus Bedingungen und Aktionen, die sich in der genetischen Programmierung gegenüber den Vektoren fester Länge durchgesetzt haben. Diese Darstellung entspräche einer Repräsentation von Computer Programmen. Im Ursprung wurden wie schon erwähnt Vektoren fester Länge eingesetzt, der Suchraum wird damit stark eingeschränkt, dennoch wird diese Methode heute noch verwendet. Hinter einem Vektor fester Länge verbirgt sich meist eine mathematische Formel, für die geeignete Parameter gesucht werden soll. Auf der Suche nach einem passenden Computerprogramm erscheint ein Vektor daher ungeeignet.

Für die Repräsentation eines Agentenverhalten bietet sich die Baumstruktur an. Die Knotenpunkte können aus Bedingungen bestehen, die in den Blättern in Aktionen münden. Der Pfad zu einer Aktion würde über eine Folge von Vergleichen verschiedener Sensorwerte führen. Dies entspräche in etwa einem Zustandsautomaten. Eine Veränderung des Sensorinputs hätte eine Zustandsänderung zur Folge, d. h. es würde einem anderen Pfad gefolgt werden und damit eine andere Aktion ausgeführt.

In EBNF ausgedrückt, sieht die Sprache folgendermaßen aus:

```
  Genotype = Condition Node Node.
      Node = Condition Node Node | Activity.
 Condition = Operand (DSens Digit | CSens Color).
  Activity = Action.
   Operand = "EQ" | "GE" | "LE".
    Action = "DRIVE" Parameter | "TRADE".
 Parameter = "FORWARD" | "BACKWARD" |
             "LEFT" | "RIGHT" | "STOP".
     DSens = "DISTANCE" | "DRIVENDISTANCE" |
             "LASTFOUND" | "OWNADDICTED" | "OWNNOTADDICTED".
     CSens = "LASTSEEN".
     Digit = "1" | "2" | "3" | "4" | "5" |
             "6" | "7" | "8" | "9" | "10".
     Color = "Red" | "Green" | "Blue".
```

Der Suchraum ergibt sich aus der Menge an möglichen Bäumen.

Nach Tabelle 3.5 (S. 39) kann der Agent 5 Bewegungsaktionen (BA) ausführen und 4 Sprechaktionen (SA). Damit sind $5BA + 4SA = 9$ Blätter möglich.

7 Sensoren stehen dem Agenten nach Tabelle 3.6 (S. 40) zur Verfügung. Diese werden mit Konstanten und Operanden zu Bedingungen zusammengesetzt und bilden dann einen

[3] Koza (1992) S. 63ff

Knotenpunkt. Wenn als Operanden $\leq$, $\geq$ und $=$ zur Verfügung stehen und die Konstanten, bzw. die Vergleichsdomäne, auf durchschnittlich 10 Möglichkeiten pro Sensor festgelegt wird, dann ergeben sich $7 \cdot 10 \cdot 3 = 210$ verschiedene Bedingungen. Die Vergleichsdomänen bestehen je nach Sensor aus einem Intervall ganzer Zahlen oder aus einer Auswahl an Farben, wie groß sie genau sind, hängt mit von der Implementierung ab, daher wird zur besseren Veranschaulichung der Größenordnung des Suchraumes eine Annahme getroffen.

Ein Baum aus einer Wurzel und zwei Kindern, bestünde demnach aus einer Bedingung und zwei Aktionen. Folgende Rechnung verdeutlicht, wie viele Bäume dieser Art existieren:

$$210\,Bedingungen \cdot 9\,Aktionen_{links} \cdot 9\,Aktionen_{rechts} = 17010$$

Die Menge der Bäume, die in den ersten beiden Ebenen aus Bedingungen (B) bestehen und deren dritte Ebene mit Aktionen (A) besetzt sind, beläuft sich auf:

$$\begin{array}{lll}
1.\,Ebene: & & 210\,B \\
2.\,Ebene: & \cdot & 210\,B_{links} \cdot 210\,B_{rechts} \\
3.\,Ebene: & \cdot & 9\,A_{B_{links}links} \cdot 9\,A_{B_{links}rechts} \cdot 9\,A_{B_{rechts}links} \cdot 9\,A_{B_{rechts}rechts} \\
\hline
 & = & 60.761.421.000
\end{array}$$

Eine systematische Suche ist bei dieser Größenordnung ausgeschlossen.

4. Entwurf und Realisierung

Aufbauend auf die Analyse (Kapitel 3) wird eine Anwendung entworfen und entwickelt. Dieses Kapitel widmet sich diesem Thema und zeigt die Architektur des Simulators, den Aufbau der Genotypen und am Ende die Realisierung des genetischen Optimierers.

4.1. Allgemein

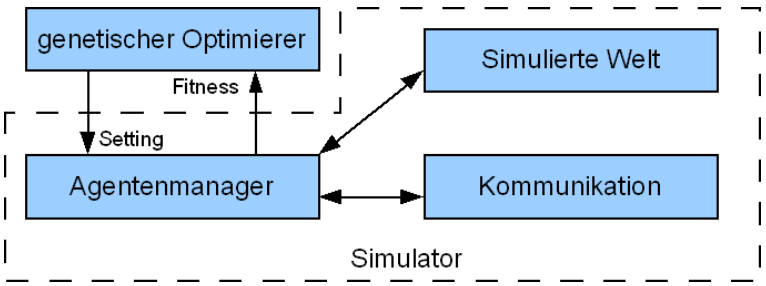


Abbildung 4.1.: erster Architekturansatz: 4 Module

Wie sich in der Analyse (3.1 Seite 30) herausgestellt hat, besteht die Anwendung aus vier Modulen (Abbildung 4.1):

Der genetische Optimierer ist dafür zuständig eine Population aus Genotypen zu erzeugen und jeweils nach erfolgter Evaluation eine Folgegeneration zu produzieren. Die Evaluation erfolgt mit Hilfe der anderen drei Komponenten, der simulierten Welt, dem Agentenmanager und der Kommunikation.

Die letzten drei Module bilden zusammen das Multiagentensystem und gleichzeitig die virtuelle Maschine, die die Fitness der Genotypen ermitteln wird. Diese Module werden daher in den folgenden Abschnitten zusammen betrachtet, während der genetische Optimierer in einem eigenen Abschnitt behandelt wird.

Um die Fitness eines Genotypen zu ermitteln wird das Multiagentensystem wie in der Analyse beschrieben, die Anfangssituation über n Schritte in Folgesituationen überführen. Dies wurde in der Analyse als Zeiteinheit definiert.

In dieser Zeiteinheit führt jeder Agent eine Handlung aus. Das heißt, das System triggert die Agenten $1 \ldots m$. Hat sich jeder Agent für eine Aktion entschieden, führt die simulierte Welt die Bewegungsaktionen aus und der Nachrichtenkanal vermittelt alle eingegangen Nachrichten.

Dieser Vorgang wird n mal durch das System ausgeführt.

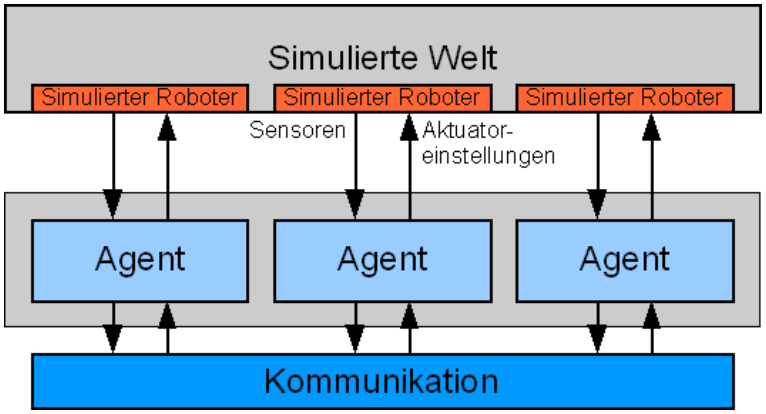


Abbildung 4.2.: Schematische Architektur der Simulationsumgebung

Im einzelnen besitzen die Module des Multiagentensystems folgende Aufgaben:

Agentenmanager Er verwaltet und Initialisiert die Agenten eines Settings. Holt vom genetischen Optimierer neue Settings zur Evaluation und startet den Evaluations-Vorgang.

simulierte Welt Sie umfasst situierte Roboter in einer Umwelt aus Wänden und Ebenen, sowie Attraktionen, die sich neben den Robotern auf den Ebenen befinden können.

Kommunikation Sie bietet einen Kommunikationskanal, damit die Agenten Handel betreiben können.

Das Zusammenwirken dieser Komponenten wird schematisch in der Abbildung 4.2 dargestellt. Die Agenten aus dem Agentenmanager tauschen mit dem Roboter des Simulators Sensordaten und Aktuatoreinstellungen aus. Desweiteren kommunizieren sie über die Kommunikationskomponente mit anderen Agenten.

Um die Agenten besser von der simulierten Welt zu trennen, um z. B. diese später gegen eine komplexere auszutauschen oder sogar echte Roboter einsetzen zu können, wird eine Zwischenschicht mit dem Namen *RoboterBoard* eingeführt (Abbildung 4.3). Diese ist eine Art schwarzes Brett, in das der Agent Aktuatoreinstellungen hineinschreibt und Sensordaten heraus liest. Der simulierte Roboter arbeitet umgekehrt, er schreibt Sensordaten hinein und liest Aktuatoreinstellungen heraus.

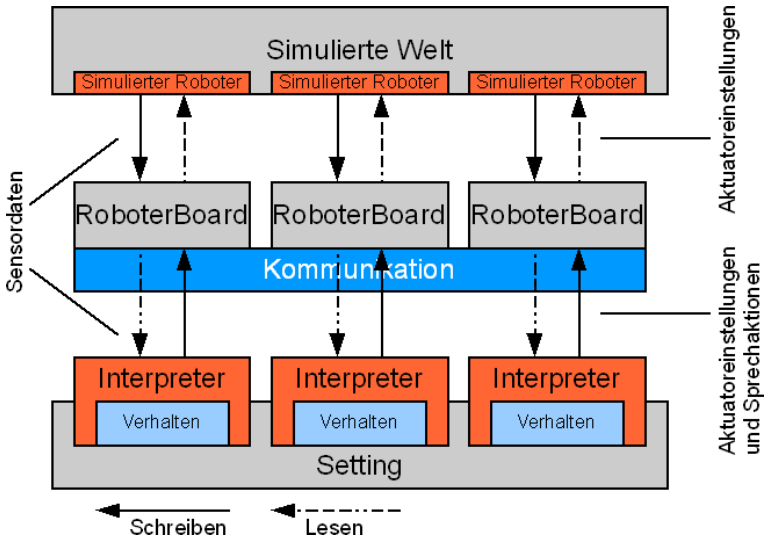


Abbildung 4.3.: Schematische Architektur der Simulationsumgebung. Agenten aufgeteilt in Interpreter und RoboterBoard.

Dieses *RoboterBoard* sichert zusätzlich den aktuellen Zustand des Agenten, da dort neben den externen Sensordaten auch die internen Sensordaten hinterlegt werden.

Da jeder Agent mit einem Verhalten ausgestattet wird und dieses interpretieren muss, besteht somit ein Agent aus einem Interpreter und diesem schwarzen Brett (Abbildung 4.3).

In den folgenden Abschnitten wird das vorgestellte Architekturkonzept im Einzelnen erläutert. Zur Steuerung und zur Visualisierung der Simulation, wird eine kleine GUI entwickelt. Auf sie wird unter Punkt 4.3.3 eingegangen.

4.2. Programmierung

Die Anwendung wurde in Java[1] Version 1.5 entwickelt. Gründe dafür waren:

- Plattformunabhängigkeit: Es gibt für Java virtuelle Maschinen für alle gängigen Plattformen, wie MS Windows, Linux, Solaris oder Mac OS.
- neue Eigenschaften der Version 1.5, wie z. B. der Enum Typ
- umfassende Kenntnis und Erfahrung mit der Java Umgebung

[1] URL zu Java im Literaturverzeichnis

Als IDE[2] kam Eclipse[3] in der Version 3.2 zum Einsatz. Eclipse ist eine ausgereifte Open Source Entwicklungsumgebung, die maßgeblich auf PlugIn Technik aufbaut. Zwar wird Eclipse hauptsächlich zur Java Entwicklung eingesetzt, was die ausgereifteste der verfügbaren Funktionalitäten ist. Doch es existieren inzwischen viele andere Entwicklungs-PlugIns, z. B. für C, C++ oder PHP. Zu den Vorzügen dieser IDE gehören z. B. die umfangreichen Refactoring Tools, die automatische Code Erweiterung, Code Templates, integrierter CVS Support und ein leistungsstarker Debugger. Es gibt neben Eclipse auch Netbeans[4] als freie Open Source IDE mit ähnlichen Eigenschaften. Letztendlich gab die persönliche Präferenz den Ausschlag für die Entscheidung.

Entwickelt wurde über weite Strecken anhand von Testfällen, die mit der JUnit[5] Bibliothek erstellt wurden. JUnit bietet ein Framework, das ermöglicht wiederholbare Tests für die zu entwickelnde Software zu schreiben. Entstanden ist die JUnit aus dem Extreme Programming[6] Paradigma um Kent Beck. Die Testfälle werden selber in Java implementiert und verkörpern die Erwartungen an die zu entwickelnde Software. Ein Test bedeutet in der Regel ein Methodenaufruf einer zu testenden Klasse. Der Rückgabewert oder die erfolgreiche Ausführung stellen den Testwert dar.

Entgegen der eigentlichen Entwicklung bedeuten Testfälle einen destruktiven Prozess, da mit den Tests versucht wird Fehler der Anwendung zu identifizieren. Diese Test zu schreiben ist für den Entwickler der Anwendung nicht besonders einfach, daher sollten die Tests in der Regel von einem Entwickler stammen, der nicht an der Anwendung selbst arbeitet.

4.3. Multiagentensystem

Wie unter Punkt 4.1 erwähnt, bilden die simulierte Welt, der Agentenmanager und die Kommunikation zusammen das Multiagentensystem, bzw. den Simulator. Es handelt sich hierbei um ein situiertes Multiagentensystem mit simulierten reaktiven Robotern.

Unter Punkt 4.3.1 wird dargestellt wie die simulierte Welt aufgebaut wurde, im folgenden Kapitel wird der Entwurf des Kommunikationsmodul vorgestellt. Der Agentenmanager und die GUI bilden das letzte Kapitel. Während der Entwurfsphase wurde der Agentenmanager Bestandteil der *Simulation*-Klasse und stellt kein eigenständiges Modul dar. Die GUI bietet eine komfortable Möglichkeit, den Evolutionsprozess zu starten, sowie die besten Lösungen grafisch darzustellen.

[2] Integrated Development Environment
[3] URL zu Eclipse im Literaturverzeichnis
[4] URL zu Netbeans im Literaturverzeichnis
[5] URL zu JUnit im Literaturverzeichnis
[6] URL zu Extreme Programming im Literaturverzeichnis

4.3.1. Simulierte Welt

Ausgehend von den funktionalen Eigenschaften aus Abschnitt 3.3 der Analyse, folgt hier der Entwurf und die Realisierung der simulierten Welt. Im Zentrum steht eine Welt, die aus Wänden und Ebenen besteht. Auf den Ebenen können Attraktionen und die Roboter stehen.

Die simulierte Welt wird aus drei Bestandteilen aufgebaut:

1. ein Raster aus Wänden und Ebenen; bilden die Umgebung (Klasse *Environment*)
2. ein Set an Attraktionen (Klasse *Attraction*)
3. ein Set an Robotern (Klasse *SimRobot*)

Zum einen verwaltet sie diese Bestandteile und zum anderen sind in ihr die Gesetze der Wechselwirkungen zwischen diesen Komponenten festgelegt.

Environment Die Umwelt wird als Raster aufgebaut, das pro Rasterfeld jeweils ein Element Wand oder Ebene aufnimmt. Dieses Raster wird mit einem zweidiemensionalen Array realisiert.

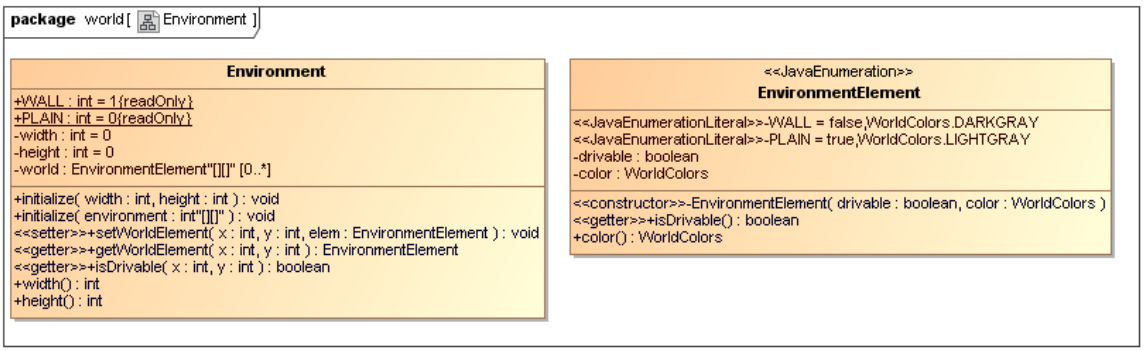


Abbildung 4.4.: UML Diagramm Environment

Wände und Ebene besitzen keine individuellen Eigenschaften, daher können sie als Literale repräsentiert werden. In Java 5 dient hierfür der Enum-Typ. Im UML Diagrammausschnitt (Abbildung 4.4) ist dieser Entwurf zu sehen.

Zu den speziellen Eigenschaften des Enum-Typ *EnvironmentElement* gehört die Abfrage, ob das Element befahrbar ist oder nicht. Jedem Element wird eine Farbe für die spätere Visualisierung zugewiesen.

Die *Environment* Klasse kann auf zwei Weisen initialisiert werden. Erstens kann über die gewünschte Höhe und Breite eine leere, nur aus Wänden bestehende Umgebung erschaffen

werden. Anschließend kann über die Setter-Methode *setWorldElement* die Umgebung gefüllt werden. Zweitens kann die Klasse auch mit einem zweidimensionalen Array initialisiert werden, dass die Umgebung in Einsen und Nullen darstellt. Eine „1“ steht für eine Wand und eine „0“ für eine Ebene. Dies ermöglicht einen einfachen Import, z. B. aus einer Textdatei.

Mit Hilfe der Methode *isDrivable* liefert ein initialisiertes *Environment* die Information, ob der Platz an einer bestimmten Koordinate befahrbar ist oder nicht.

WorldObject* und *WorldObjects Die *Attraktionen* und die *Roboter* sind Unterklassen von *WorldObject* (Abbildung 4.5). Um ein *WorldObject* in der Umwelt zu repräsentieren besitzt es eine Farbe und Position. Für Attraktionen (*Attraction*) reichen diese Eigenschaften aus, daher werden sie durch die *Attraction* Klasse nicht erweitert. Ein Roboter (*SimRobot*) hingegen benötigt außerdem noch die Informationen über die eigene Ausrichtung auf der Ebene, seinen Sichtbereich (*ViewabelArea*) und die Verbindung mit dem schwarzen Brett, über das Daten mit dem Agenten ausgetauscht werden. Der Roboter besitzt noch weitere zwei Methoden, die dazu dienen den Distanzsensor und die Kamera zu aktualisieren. Diese werden von der *SimulatedWorld* aufgerufen (siehe weiter unten).

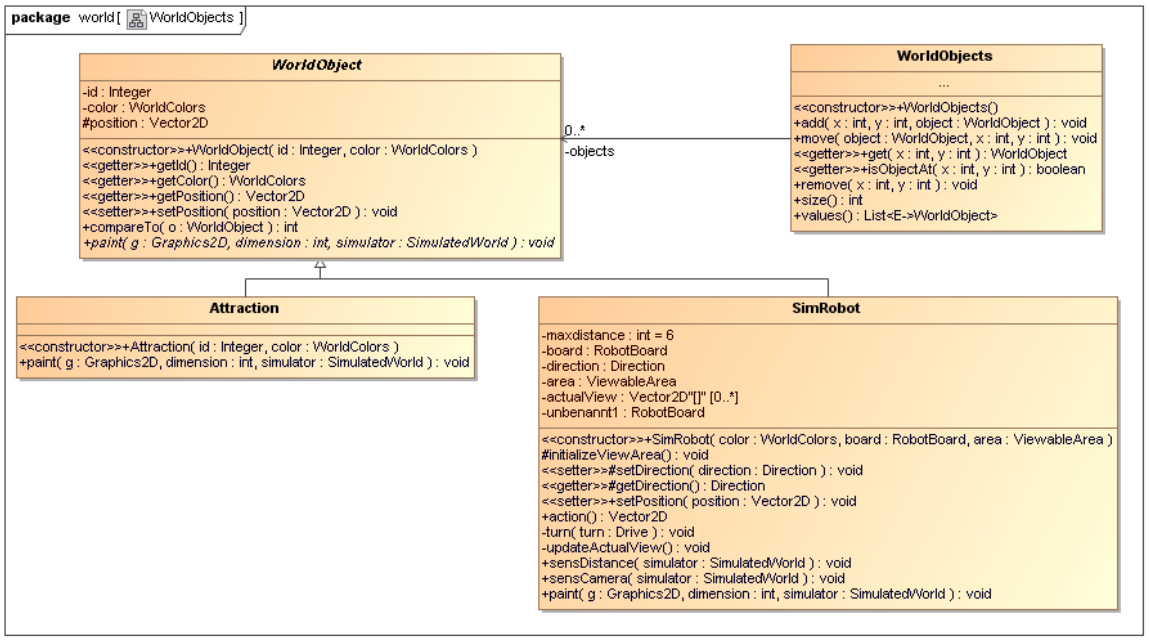


Abbildung 4.5.: UML Diagramm World Objects

In der Methode *action* werden die Aktuatoreinstellungen vom schwarzen Brett (*RobotBoard*) gelesen und ausgewertet. Anschließend liefert die Methode die gewünschte Zielposition des Roboters zurück.

Objekte der Klassen *Attraction* und *SimRobot* werden in der Klasse *WorldObjects* verwaltet. Dort kann ein *WorldObject* zu neuen Koordinaten bewegt und die Rückmeldung gegeben werden, ob sich ein Objekt an bestimmten Koordinaten befindet. Mit Hilfe der Klassen *Environment* und *WorldObjects* wird der Aufbau des Simulators (Abbildung 4.6) erheblich vereinfacht.

SimulatedWorld In dem UML Diagrammausschnitt 4.6 ist die *SimulatedWorld*-Klasse dargestellt. Sie verwaltet die Roboter und Attraktionen getrennt voneinander, weil dies den Zugriff auf die jeweilige Klasse an Objekten vereinfacht. Außerdem erleichtert dies, Kollisionen zwischen zwei Robotern abzufragen und Attraktionen ausfindig zu machen.

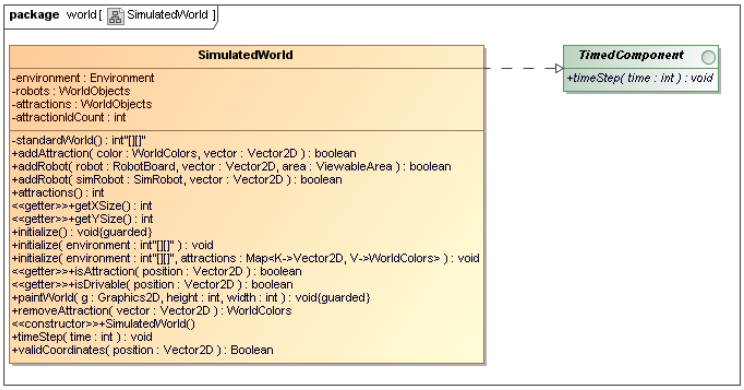


Abbildung 4.6.: UML Diagramm der simulierten Welt

Die wichtigsten Methoden der simulierten Welt werden hier erläutert:

addAttraction(color: WorldColors, vector: Vector2D) fügt dem Simulator eine Attraktion an der Position des übergebenen Vectors mit der zugehörigen Farbe hinzu.

addRobot(robot: RobotBoard, vector: Vector2D, area: ViewableArea) fügt dem Simulator einen Roboter an der Position des übergebenen Vectors hinzu. Dieser Roboter ist dann mit dem übergebenen *RobotBoard* verbunden. Die *ViewableArea* legt die Struktur und Größe des Sichtbereichs fest.

removeAttraction(vector: Vector2D): WorldColors entfernt eine Attraktion aus den angegebenen Koordinaten. Die Farbe der entfernten Attraktion wird zurück geliefert. Wenn sich an der Position keine Attraktion befand, wird eine NONE Farbe geliefert.

validCoordinates(position: Vector2D) gibt zurück, ob sich die Koordinaten im erlaubten Wertebereich, also innerhalb der Welt, befinden.

timeStep(time: int) bestimmt die Zeittaktung der Simulation. Alle Roboter werten in diesem Schritt ihre Aktuatoreinstellungen aus und berechnen ihre neue Zielposition. Außerdem werden ihre Sensoren aktualisiert und ggf. Attraktionen zu dem Bestand eines Roboters hinzu gefügt.

paintWorld(g: Graphics2D, height: int, width: int) zeichnet die Welt auf dem übergebenen Graphics Objekt.

Die grafische Darstellung der simulierten Welt ist in Abbildung 4.7 zu sehen. Die Roboter werden als große Kreisflächen mit einer Aussparrung, die die Vorderseite der Roboter markiert, dargestellt. Die Attraktionen sind die kleineren farbigen Kreisflächen. Die gelb/orange farbenen Flächen zeigen den Sichtbereich der Roboter an. In dieser Abbildung sind drei Roboter mit den Neigungen rot, grün und blau zu sehen, generell sind mit der Software aber beliebig viele Roboter möglich. Begrenzt wird die Anzahl durch die Fläche der Umwelt und dem Speicher.

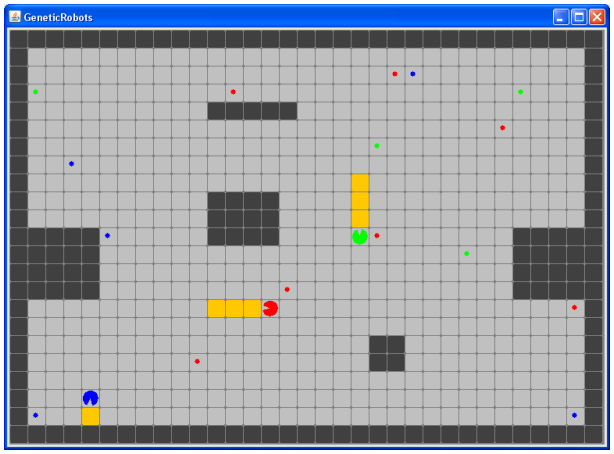


Abbildung 4.7.: grafische Darstellung der Welt des Simulators

4.3.2. Kommunikation

Damit die Agenten untereinander Attraktionen tauschen können, benötigen sie eine Kommunikationsmöglichkeit. In der Analyse wurden in der Tabelle 3.5 (S. 39) die verschiedenen Sprachaktionen ermittelt, mit deren Hilfe eine Verhandlung vonstatten gehen kann. Auf dieser Basis wird ein Kommunikationsmodul entworfen. Dieses Modul wird alle Handlungen für einen erfolgreichen Tauschhandel abdecken, so dass dies in einer Aktion *Handeln* zusammen gefasst für die Genotypen zur Verfügung gestellt werden kann.

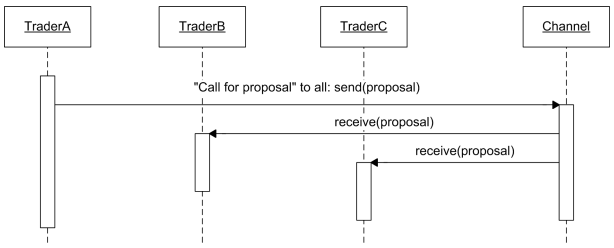


Abbildung 4.8.: Ein Agent sendet eine Ausschreibung an alle aus

Im Zentrum der Kommunikation steht der Nachrichtenkanal über den alle Nachrichten gehen. Wenn ein Agent eine Nachricht verschicken möchte, so übergibt er diese dem Nachrichtenkanal, dieser leitet die Nachricht an den jeweiligen Empfänger weiter. Geht die Nachricht an alle Agenten, so muss der Agent nicht extra alle Agenten einzeln ansprechen. Über den Empfänger „All“ verschickt der Nachrichtenkanal die Nachricht an alle verfügbaren Agenten.

Es gibt vier verschiedene Nachrichtentypen die ein Agent verschicken kann:

1. „Call for proposal“: Eine Ausschreibung für die Anzahl von Attraktionen einer bestimmten Farbe abschicken (Angebote erfragen).
2. „Offer proposal“: Für eine Ausschreibung ein Angebot unterbreiten.
3. „Accept proposal“: Ein Angebot akzeptieren.
4. „Close proposal“: Eine Ausschreibung schließen.

In Abbildung 4.8 wird der erste Fall dargestellt. Ein Agent „TraderA“ schickt ein „Call for proposal“ an alle anderen Agenten. Es wird mit dem Empfänger „All“ versehen, damit es alle Agenten erreicht.

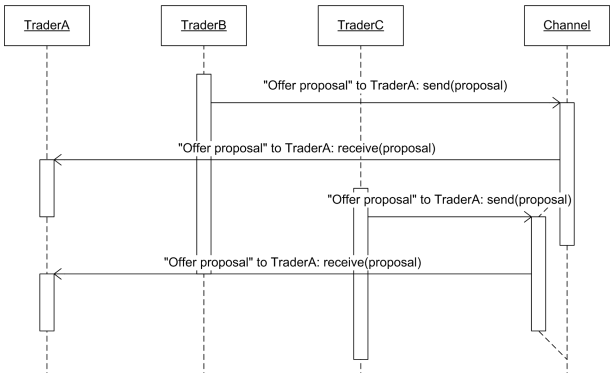


Abbildung 4.9.: Die anderen Agenten senden Angebote für die Ausschreibung von Trader A

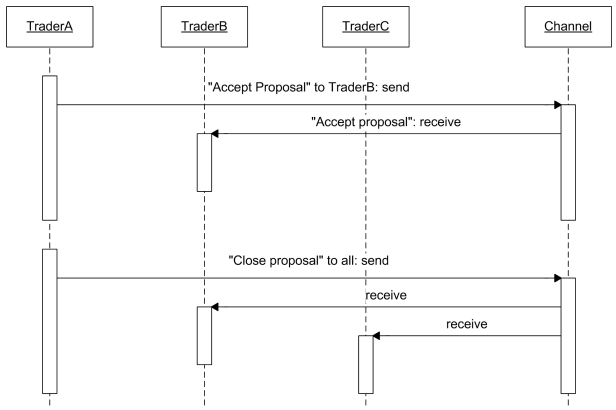


Abbildung 4.10.: Agent Trader A verschickt ein „Accept proposal“ und anschließend ein „Close proposal“.

Nachdem alle Agenten den „Call for proposal“ erhalten haben, können sie Angebote abgeben. Diese schicken sie direkt an den Fragenden. In Abbildung 4.9 wird dieser Vorgang dargestellt.

Dann kann sich der Agent „Trader A“ für ein Angebot entscheiden. Akzeptiert er ein Gebot, so sendet er dem jeweiligen Anbieter ein „Accept proposal“ wie es in Abbildung 4.10 dargestellt ist.

Abschließend muss der Agent „Trader A“ die Ausschreibung für alle schließen, damit die restlichen Agenten wissen, dass sie ihre Offerten wieder freigeben können.

Zu beachten ist, dass die Nachrichten der einzelnen Agenten der Abbildung zufolge zeitlich versetzt anzukommen scheinen. Diese Darstellung wurde zugunsten der Übersicht gewählt. Eigentlich laufen diese Aktionen in einer Zeiteinheit ab. In der Simulation läuft das folgendermaßen ab: Die Agenten versenden alle nacheinander ihre Nachrichten, diese werden bei dem Nachrichtenkanal zwischen gespeichert. Anschließend verteilt der Nachrichtenkanal die Nachrichten an die Empfänger.

Das fertige Modul tauscht Attraktionen immer eins zu eins. Eine Erweiterung dieses Verhaltens ist für eine spätere Version vorgesehen.

Im Anhang B.1 (S. 81) sind die UML Diagramme zum Modul und den Nachrichten abgebildet.

4.3.3. Agentenmanager, Evaluation und GUI

Wie in der Einleitung beschrieben, wurde der Agentenmanager in der *Simulation*-Klasse integriert. Diese Klasse kann Simulationsläufe mit Settings durchführen, dazu wird dieser

Klasse über die *start*Methode ein Setting übergeben. Durch diesen Methodenaufruf werden die Welt, mitsamt den Robotern und Attraktionen, die Agenten und der Kommunikationskanal initialisiert. Für den eigentlichen Simulationslauf wird die Klasse von außen über die Methode *timeStep* getaktet.

Diese Taktung geschieht über die *Evaluator*-Klasse. Sie steuert den eigentlichen Evaluationsprozess. Dazu liest sie die Parameter für den genetischen Optimierer und den Simulationslauf aus einer Konfigurationsdatei ein und initialisiert den genetischen Optimierer. Der Evaliationsprozess läßt sich über die Methoden *start* und *stop* kontrollieren. Über die *start*-Methode wird ein Thread gestartet, der den Prozess ausführt. Währenddessen protokolliert diese Klasse den Verlauf der Fitness, das Nachrichtenaufkommen im Nachrichtenkanal und die Verhalten des jeweils fittesten Setting, wie auch das fitteste Einzelverhalten.

Über eine GUI kann der Evolutionsprozess gestartet, gestoppt und überwacht werden. Außerdem ist es möglich, einen Simulationslauf des fittesten Settings der zuletzt evaluierten Generation grafisch darzustellen.

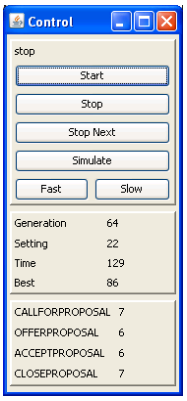


Abbildung 4.11.: Das Control Panel des Simulators

4.4. Genetik

Als erstes wird hier der Aufbau der Genotypen erklärt und wie diese interpretiert werden. Anschließend wird auf die Methoden des genetischen Optimierers eingegangen. Den Abschluss dieses Kapitels bildet die Fitness Funktion, die entscheidend den Lernerfolg steuert.

Das Verhalten eines Agenten ist ein Gen und wird als *Behavior* bezeichnet. Dieses wird als Baumstruktur aus Literalen aufgebaut. Hier wird wieder der Java Enum Typ verwendet. Als

Operanden	
EQ	(=) Gleichheit
GE	(>=) Größer gleich
LE	(<=) Kleiner gleich

Tabelle 4.1.: Genotypen - Operanden

Knoten für den Baum stehen Bedingungen (*Condition*) zur Verfügung, als Blätter dienen Aktivitäten (*Activity*). Bedingungen nehmen Operanden (*Operand*, Tabelle 4.1) und Sensoren (*Sensor*, Tabelle 4.2), sowie Konstanten für die Vergleichswerte auf. Die Aktivitäten verwenden Aktionen (*Action*, Tabelle 4.3), sowie Konstanten als Parameter. In diesem Fall besitzt nur die Aktion *Drive* einen Parameterwert.

Sensoren	Wertebereich	Beschreibung
DISTANCE	0 . . . 6	Abstand zum nächsten Hindernis vor dem Roboter
ADDICTION	ROT, GRÜN, BLAU	Abfrage, welcher Farbe der Agent zugeordnet ist
LASTSEEN	ROT, GRÜN, BLAU	zuletzt gesichtete Farbe
OWNADDICTED	0 . . . 15	Bestand der Attraktionen der eigenen Farbe
OWNNOTADDICTED	0 . . . 15	Bestand der übrigen Attraktionen
LASTFOUND	0 . . . 50	Zeit seit zuletzt gefundener Attraktion
OPENPROPOSAL	0 . . . 5	Anzahl offener Angebotsanfragen

Tabelle 4.2.: Genotypen - Sensoren

Aktionen	Parameter	Beschreibung
DRIVE	FORWARD, BACKWARD, LEFT, RIGHT, STOP	fahren
TRADE	-	Tauschhandel betreiben

Tabelle 4.3.: Genotypen - Aktionen

Jeder Sensor und jede Aktion besitzt eine Werte Domäne, die gültige Vergleichs- bzw. Parameter-Werte darstellt. Diese werden für die Erstellung der Genotypen verwendet. In den Bedingungen wird ein Sensor mit einem Wert aus seinem Wertebereich über einen Operanden verknüpft. Der Vergleich von Farben mit Operationen wie *größer gleich* oder *kleiner gleich* erscheint zunächst problematisch. Eine willkürliche Festlegung der Reihenfolge in einer Farben-Enumeration ermöglicht diese mit den Operationen zu verknüpfen und vereinfacht so den Interpreter.

Eine Darstellung dieser Komponenten als UML Digramme ist im Anhang B.2 (S. 83) zu finden.

Ein Genom könnte folgende Struktur aufweisen (Darstellung als String):

```
GE(DISTANCE,␣5)
␣DRIVE(FORWARD)
␣DRIVE(LEFT)
```

Dieses Genom besteht aus einer Bedingung und zwei Aktivitäten. Die erste Zeile steht für die Wurzel des Baumes und ist immer eine Bedingung. Eingerückt darunter befindet sich der linke und in der letzten Zeile der rechte Ast. Der linke Ast repräsentiert den Fall, wenn die Bedingung `GE(DISTANCE, 5)` wahr ist, andernfalls gilt der rechte Ast. Bedingungen können in beliebiger Reihenfolge auftreten. Die Tiefe der Bäume wird auf 20 Ebenen begrenzt. Damit sollte reichlich Spielraum für die Genomentstehung zur Verfügung stehen, ohne dass die Größe der Bäume ausartet. Die maximale Tiefe ist im genetischen Optimierer einstellbar.

Das Verhalten eines Agenten stellt in diesem Szenario ein Gen dar, ein Genotyp entsteht aus einem Setting, d. h. in diesem Versuchsaufbau die Kombination von genau drei Verhalten mit unterschiedlichen Farbneigungen.

Der **Interpreter** ist in der *Agent* Klasse integriert. Er besteht aus Switch Anweisungen, die die Literale aus den Genotypen auswertet.

Die Methoden des genetischen Optimierers:

rankingSelectionSetting(count: int) selektiert Settings aus der aktuellen Population über das Rankingverfahren. *Count* gibt die Anzahl der Settings an.

rouletteSelectionSetting() selektiert Settings mittels des Rouletteverfahrens. Der Selektionsdruck wird über den internen Parameter *selectionPressure* geregelt.

turnierSelectionSetting() selektiert Settings über das Turnierverfahren. Gesteuert wird dies über den internen Parameter *dipperSize*.

rankingSelectionGene(count: int) selektiert Gene, bzw. *Behavior*, analog zum Ranking von Settings.

crossover(gene1: Condition, gene2: Condition): Gene[ ] kreuzt zwei Gene miteinander und gibt die entstandenen Kinder zurück.

mutate(gene: Gene) verändert ein Gene an einem Knotenpunkt in beliebiger Tiefe, die restliche Struktur bleibt erhalten.

Parameter	Wert	Einheit
popSize	50	Settings
selectionPressure	10	% Settings
dipperSize (Schöpfkellen Größe)	3	Settings
crossoverRate	8	Behavior Paare
mutationRate	8	Behavior
maxLayer (maximale Tiefe)	20	Behavior Ebenen

Tabelle 4.4.: Parametereinstellungen des Optimierers

Über Parameter wird das Verhalten des Optimierers gesteuert. Die Anfangs gewählten Werte werden in der Tabelle 4.4 dargestellt.

Die Experimente müssen zeigen, in wie weit diese Werte zu Ergebnissen führen können.

4.4.1. Fitness Funktion

Die Fitnessfunktion benötigt besondere Aufmerksamkeit. Hat ein Agent möglichst viel Besitz erwirtschaftet, so soll er dafür belohnt werden. Befindet sich in seinem Besitzt die bevorzugte Farbe an Attraktionen, so soll er dafür besonders belohnt werden. Die folgende Fitnessfunktion dient als erster Ansatz, die bevorzugte Farbe geht hier zweifach in die Rechnung ein:

$$Fitness_1 = stockRed + stockGreen + stockBlue + stockAddicted$$

4.5. Evaluation (Versuchsdurchführung)

Auf den folgenden Seiten werden verschiedene Versuchsläufe mit der entwickelten Software dargestellt.

Erste Testläufe während der Entwicklungsphase mit der $Fitness_1$ aus Kapitel 4.4.1 auf Seite 58 zeigten zwar, dass die Software generell funktioniert, die Genotypen entwickelten allerdings keinerlei Ambitionen für Tauschhandel.

Daher wurde die Fitnessfunktion angepasst. Der Besitz von bevorzugten Attraktionen wurde noch stärker belohnt:

$$Fitness_2 = stockRed + stockGreen + stockBlue + 5 \cdot stockAddicted$$

Alle folgenden Versuche wurden mit dieser Fitnessfunktion durchgeführt. Außerdem wurde in jedem Versuch mit drei Agenten gearbeitet, mit den Farben rot, grün und blau.

Aus jedem Versuch sind im Anhang A (S. 72) die jeweils fittesten Agentenverhalten aus der 1. Generation, sowie einer Generation nachdem kein weiterer Anstieg der Fitness zu beobachten ist.

4.5.1. Versuch 1

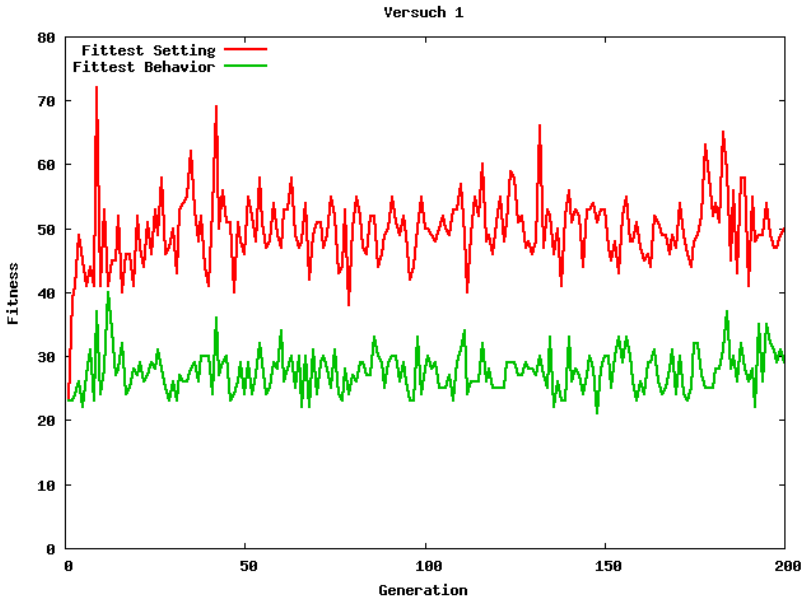


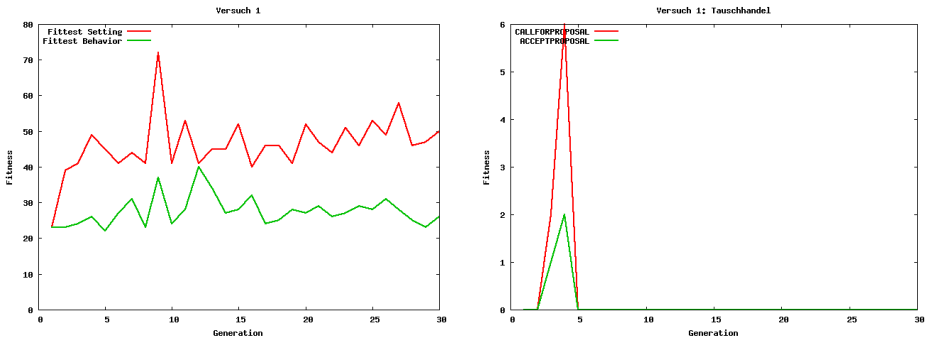
Abbildung 4.12.: Fitness Verlauf bei Versuch 1 bis Generation 200

Die Parameter für den ersten Versuchsdurchlauf:

Parameter	**Größe**	**Einheit**
Populationsgröße	100	Setting
Simulationsschritte	100	
Selektionsanteil	15	% Setting
Crossover	8	Behavior Paar
Mutation	8	Behavior

In dem Diagramm 4.12 sind sehr starke Schwankungen im Verlauf der Fitness zu erkennen. Dies läßt sich daraus erklären, dass jedes Setting nur einmal evaluiert wird und bei jeder neuen Generation die Attraktionen an anderen Positionen stehen. Ein vormals erfolgreiches Setting kann aus diesem Grund in der nächsten Generation ein wesentlich schlechteres Ergebnis erreichen.

Dennoch ist zu erkennen, dass die Fitness in den ersten Generationen stark ansteigt und anschließend trotz Schwankungen einen schwachen Aufwärtstrend besitzt. Ab Generation 50 bis 200 überwiegen die Schwankungen, so dass kein Trend mehr zu erkennen ist. Zu vermuten ist, dass die starken Schwankungen der Fitness von einer Generation zur Nächsten



(a) Fitness Verlauf (b) Kommunikation

Abbildung 4.13.: Versuch 1 bis Generation 30

wie ein Rauschen den Prozess, erfolgreichere Settings zu finden, überlagern. Feine Nuancen werden anscheinend unterdrückt. Ob hier eine Fitnessgrenze erreicht wurde, läßt sich dadurch nicht sagen.

In Abbildung 4.13 (a) sind die ersten 30 Generationen heraus gestellt. Es ist zu erkennen, dass es in diesen Generationen eine deutliche Aufwärtstendenz gibt. Allerdings wie in der Abbildung 4.13 (b) dargestellt, erfolgt kein Tauschhandel.

Zum Vergleich sind im Anhang A.1 (S. 72) das fitteste Verhalten aus Generation 1 und Generation 30 gedruckt. Ein Anfänglich sehr großer Verhaltenscode ist auf wenige Befehle zusammengeschrumpft und ist dabei erfolgreicher als der erste.

4.5.2. Versuch 2

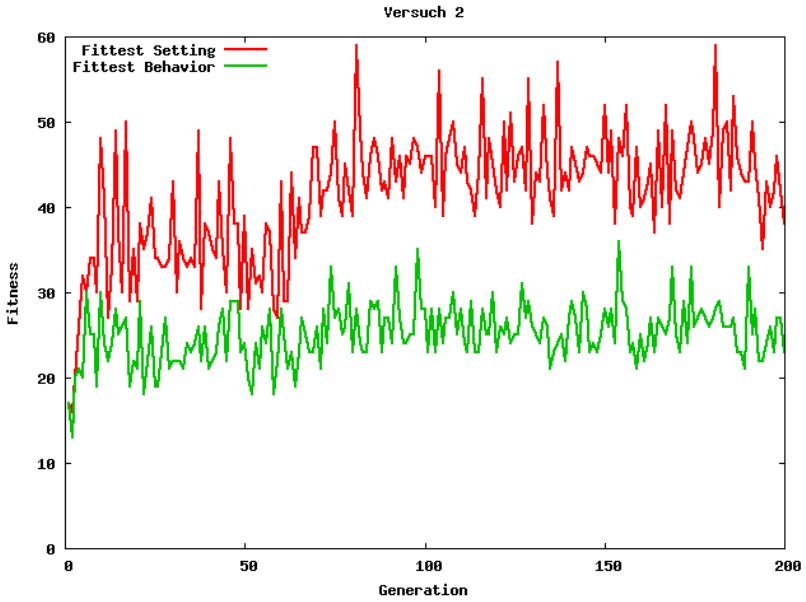


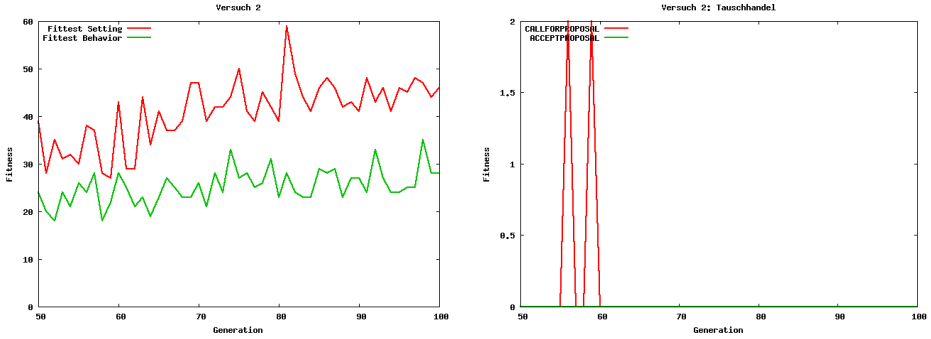
Abbildung 4.14.: Fitness Verlauf bei Versuch 2 bis Generation 200

Die Parameter für den zweiten Versuchsdurchlauf:

Parameter	**Größe**	**Einheit**
Populationsgröße	100	Setting
Simulationsschritte	100	Zeiteinheit
Selektionsanteil	15	% Setting
Crossover	8	Behavior Paar
Mutation	8	Behavior

Im zweiten Versuch (Abbildung 4.14) zeigt sich ein ähnliches Bild wie im ersten. Hier ist allerdings ein Sprung der Fitness zwischen der 50ten und der 100ten Generation zu erkennen. In der Vergrößerung (Abbildung 4.15 (a) S. 63)ist der Aufwärtstrend deutlich zu sehen. Allerdings gilt auch für diesen Versuch, dass kein Tauschhandel entstanden ist.

Hier kann vermutlich die Simulationszeit eine Rolle spielen. In den ersten beiden Versuchen wurden 100 Simulationsschritte ausgeführt. Die Settings können während dieser Zeitspanne anscheinend keinen Nutzen aus einem Tauschhandel ziehen, da sich noch ausreichend Ressourcen in der Umgebung befinden und der Aufwand für eine weitere Suche geringer ist, als einen Tausch anzustreben. Daher wird der nächste Versuch mit einer höheren Schrittanzahl ausgeführt.



(a) Fitness Verlauf (b) Kommunikation

Abbildung 4.15.: Versuch 2 Generation 50 bis 100

Für diesen Versuchslauf werden ebenfalls zum Vergleich die fittesten Verhalten nach Generation 1 und 100 im Anhang A.2 auf Seite 74) gedruckt. Nach der 1. wie auch 100. Generation zeigen sich zwei sehr umfangreiche Verhalten.

4.5.3. Versuch 3

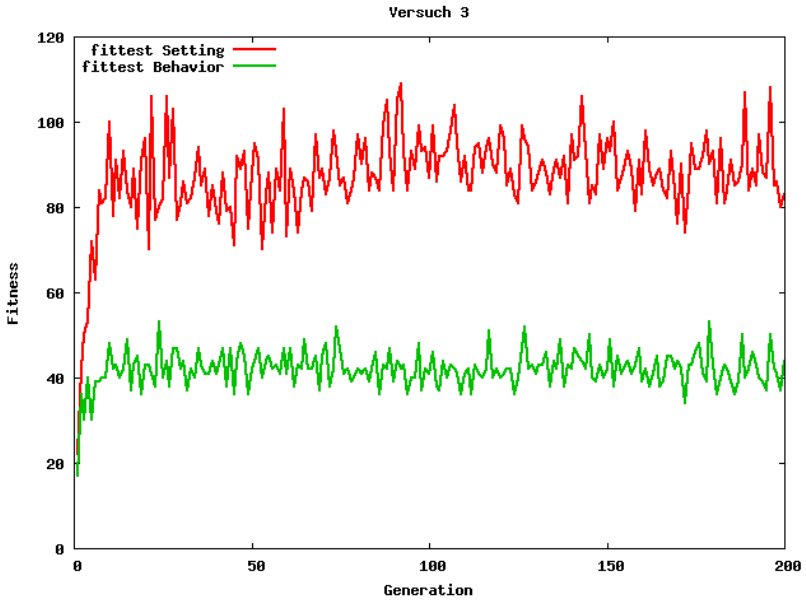


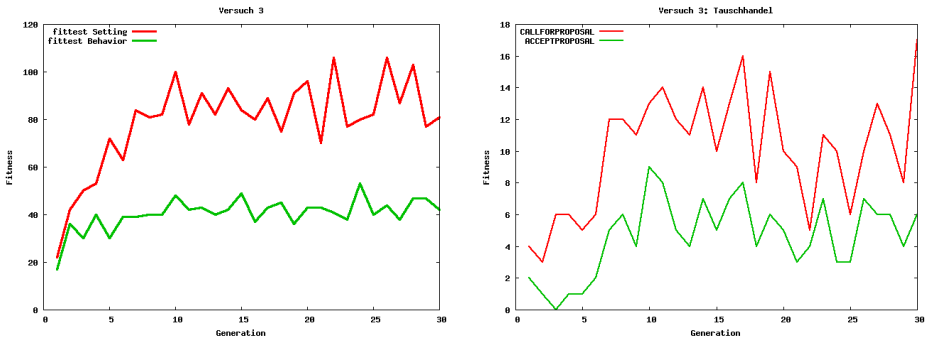
Abbildung 4.16.: Fitness Verlauf bei Versuch 3 bis Generation 200

Die Parameter für den dritten Versuchsdurchlauf:

Parameter	**Größe**	**Einheit**
Populationsgröße	100	Setting
Simulationsschritte	300	
Selektionsanteil	15	% Setting
Crossover	8	Behavior Paar
Mutation	8	Behavior

Abbildung 4.16 zeigt einen ähnlichen Verlauf wie im ersten Versuch (Abbildung 4.12 S. 60), allerdings mit einer höheren Gesamtfitness. Die höhere Fitness läßt sich auf die längere Simulationslaufzeit zurück führen, da die Fitness nicht normalisiert wird. Der Anstieg bis die Kurve in ein kontinuierliches Schwanken übergeht zieht sich über die ersten 10 Generationen hinweg und dauert damit etwas länger als in Versuch 1. In dieser Zeit finden anscheinend noch erfolgreiche Optimierungen der Settings und Behavior statt.

Die Abbildung 4.17 (b) zeigt allerdings einen ganz anderen Verlauf der Kommunikation über die ersten 30 Generationen, als es noch in den Versuchen 1 und 2 der Fall war. Die grüne Linie zeigt erfolgreichen Tauschhandel, die rote Linie zeigt die Versuche Attraktionen anzubieten. Damit wäre die Vermutung aus Versuch 2 bestätigt, dass sich Tauschhandel erst



(a) Fitness Verlauf (b) Kommunikation

Abbildung 4.17.: Versuch 3 bis Generation 30

lohnt, wenn über eine lange Simulationsstrecke die Ressourcen knapp werden und eine Verbesserung der Suche eine geringere Steigerung der Fitness nach sich zieht, als Attraktionen zu tauschen.

Auch in diesem Versuchslauf sind die starken Schwankungen wie in den anderen beiden Versuchsdurchläufen zu sehen. Wie in den Versuchen zuvor wird vermutlich das Rauschen einen weiteren Entwicklungsprozess stören. Das läßt sich aus dem rasanten Anstieg am Anfang und das anschließende Pendeln zwischen einer unteren und oberen Qualitätsgrenze schließen.

4.5.4. Versuch 4

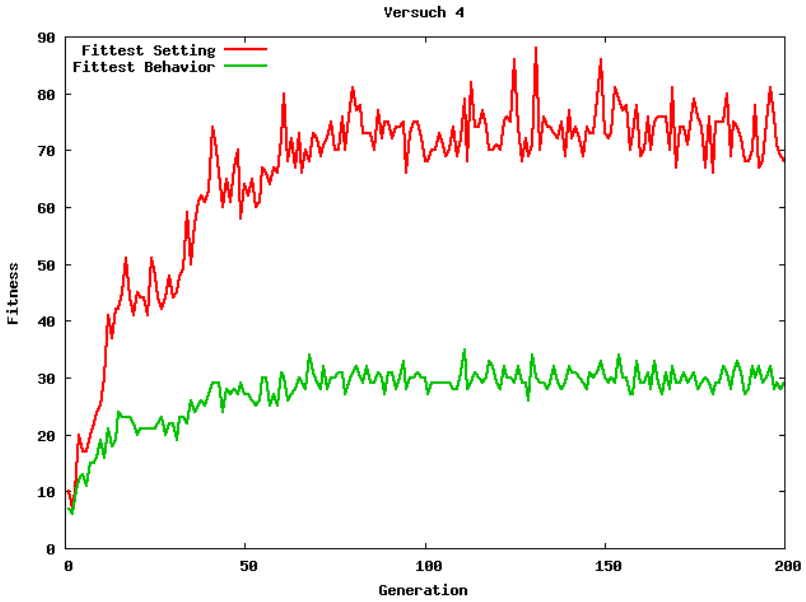


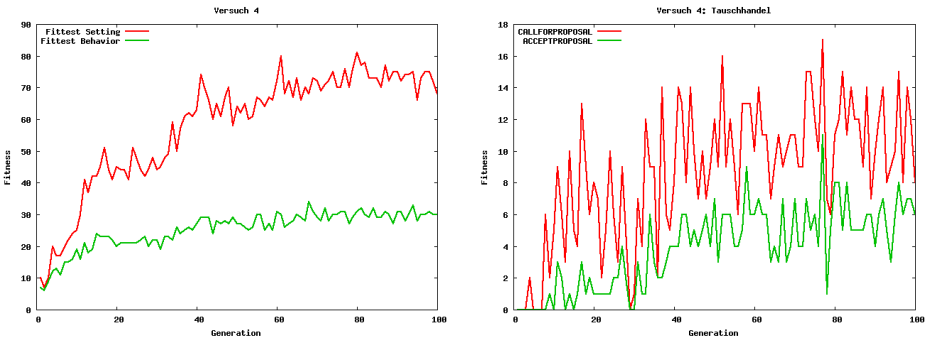
Abbildung 4.18.: Fitness Verlauf bei Versuch 3 bis Generation 200

Die Parameter für den vierten Versuchsdurchlauf:

Parameter	**Größe**	**Einheit**
Populationsgröße	100	Setting
Simulationsschritte	300	
Selektionsanteil	15	% Setting
Crossover	16	Behavior Paar
Mutation	16	Behavior
Oversampling	3	Simulationen

Für diesen Versuch wurde die Software leicht modifiziert, um die Möglichkeit zu schaffen, ein Setting für jeden Evaluationsdurchlauf mehrmals zu simulieren, um einen Durchschnitt der Wertigkeit über die Anzahl der Simulationsläufe zu bilden. Dieser Vorgang wird über den Parameter Oversampling gesteuert. Für diesen dritten Versuch wurden drei Simulationen pro Setting ausgeführt und davon die durchschnittlich erreichte Fitness als Qualität des Setting angegeben.

Zu dem wurden noch die Parameter für Anzahl der zu verheiratenden Behaviorpaare (Crossover) und die Mutationsrate erhöht.



(a) Fitness Verlauf (b) Kommunikation

Abbildung 4.19.: Versuch 4 bis Generation 100

Abbildung 4.18 zeigt, dass in diesem Versuch die Kurve viel flacher ansteigt als in den vorherigen Versuchen. Bis ca. zur 100. Generation steigt die Fitness beinahe kontinuierlich an. Die Verbesserung der Fitness fällt allerdings auch in diesem Versuch in den ersten Generationen am Stärksten aus.

Zu beobachten ist, dass die Schwankungen im Verlauf der Fitness flacher ausfallen. Die mehrfache Auswertung eines Setting hat sich demnach stabilisierend auf den Evaluationsprozess ausgewirkt.

5. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Multiagentensystem aus simulierten fahrenden Robotern aufgebaut, die farbige Objekte, Attraktionen genannt, suchen und untereinander tauschen sollten. Die Agenten, die die Logik der Roboter darstellten, sind reaktive Systeme, die eine rudimentäre numerische Repräsentation ihrer Umwelt besitzen.

Über genetische Programmierung sollte eine Zusammenstellung von Agentenverhalten, Setting genannt, gefunden werden, in der die Agenten sowohl suchen, als auch Tauschhandel betreiben. Jeder Agent bevorzugte eine bestimmte Farbe und der Besitz von Attraktionen dieser Farbe wurde in der Fitnessfunktion besonders belohnt.

Zum Aufbau der Behavior, wurde eine einfache Grammatik aus Bedingungen und Aktivitäten entwickelt, mit der sich Sätze in Form eines Baumes bilden ließen. Für die Bedingungen standen die externen Sensoren der Roboter, wie auch interne Sensoren der Agenten zur Verfügung, für Aktivitäten die Aktionen Handeln und Fahren. Während die Aktion *Handeln* ein autonomes Modul aufrief, bedeutete die Aktion *Fahren* ein direkte Ansteuerung der „Motoren“. Ein Agent konnte mit einem dieser Behavior ausgestattet werden und dieses interpretieren. Ein Setting bestand aus drei Verhalten, die jeweils einer der Farben Rot, Grün und Blau zugeordnet wurden. Selektion fand nach jeder Generation auf Settingebene statt. Die genetischen Operationen Crossover und Mutation wurden nur auf Behavior angewendet.

Es zeigte sich, dass Entwurf und Entwicklung einer geeigneten Software keine besondere Hürde darstellt. Genetische Verfahren mit einem eingeschränkten Satz an Kommandos, wie in dieser Arbeit, sind relativ einfach zu implementieren. Als wesentlich schwieriger erwies sich die Entscheidung, auf welcher Abstraktionsebene das Genom arbeiten soll. Gibt es Module für bestimmte Aufgaben oder soll diese Aufgabe auch Teil des Lernprozesses werden? So wurde hier die Möglichkeit des Tauschhandels in einem eigenen Modul implementiert, welches das Genom mit der Aktion *TRADE* aufrufen konnte. Die Suche war jedoch Teil der Optimierung.

Die ersten Versuche mit wenigen Zyklen zeigten, dass die Optimierung eher in Richtung individueller Optimierung ging, d. h. die Agenten eher damit beschäftigt waren die Ebene nach Attraktionen abzusuchen, als untereinander Tauschhandel zu betreiben. Allerdings zeigten Versuche mit höherer Zyklenanzahl, dass die Agenten schon in frühen Generationen anfingen untereinander mit Attraktionen zu handeln.

Es waren stets starke Schwankungen der Fitness von Generation zu Generation zu beobachten. Dies ist zum Teil darauf zurückzuführen, dass jedes Setting zur Bewertung nur einem Simulationslauf unterzogen wurde. Ein erster Versuchsaufbau mit mehreren Simulationsläufen pro Setting, in denen jedes Mal die Positionen der Attraktionen verändert wurden, zeigte in diesem Punkt eine leichte Verbesserung.

In der vorliegenden der Software besitzen die Agenten nur eine sehr eingeschränkte Sicht ihrer Umwelt. Außerdem haben sie so gut wie kein Gedächtnis, sie wissen lediglich wie lange der letzte Fund einer Attraktion zurück liegt.

Als Gedächtnis kommen z. B. Informationen über den Erfolg von Handelsbemühungen in Frage oder wie erfolgreich das Suchen in einem Teil der Umwelt war.

Das Handelsmodul könnte dahingehend modifiziert werden, dass der Agent entscheiden kann, ob das Modul nächstes Mal mit höheren oder niedrigeren Angeboten an die anderen Agenten herantreten soll.

Zudem stehen Suchen und Handeln noch auf unterschiedlichen Abstraktionsebenen, was ein Ungleichgewicht im Optimierungsprozess bewirkt. Beide Handlungen sollten auf eine Stufe gestellt werden. Dafür müssten Aktionen zur Verfügung stehen wie „Suche in Quadrant x“ oder dergleichen.

Derartige Veränderungen könnten den Grad der Optimierung steigern und die beobachteten Schwankungen im Fitnessverlauf verringern.

Ein genetisches Verfahren zu implementieren ist wenig problematisch, schwieriger ist es den Lernerfolg über die Parameter positiv zu beeinflussen. Allen voran die Fitnessfunktion, denn misslingt es hier die Qualität einer Lösung aussagekräftig zu kodieren, ist das ganze Verfahren reine Resourcenverschwendung. Aber auch die Selektionsverfahren, der Selektionsdruck und der Anteil an Crossover bzw. Mutationen sind von großer Bedeutung.

Es zeigte sich, dass zum Finden einer guten Lösung Experimente beim Einstellen dieser Parameter und deren Auswertung eine große Rolle spielten.

In diesem Stadium der Software ist es ungünstig sie auf reale Roboter wie den ct-bot zu übertragen. Die Ursachen für die starken Schwankungen im Optimierungsprozess müssten vorher lokalisiert und eliminiert werden. Die reale Welt hält viele zusätzliche Störfaktoren bereit, die den Optimierungsprozess negativ beeinflussen könnten und damit eine Optimierung verhindern würden. Hierzu wäre sicherlich notwendig die oben genannten Erweiterungen zu implementieren und damit verbunden weitere Experimente durchzuführen.

Insgesamt ist festzustellen, dass ein Einsatz genetsicher Lernverfahren in einem Szenario dieser Art grundsätzlich erfolgversprechend ist.

Literaturverzeichnis

[Burka 2007] BURKA, Florian: *Kameragestützte Objektverfolgung in Echtzeit im Kontext mobiler Roboter*. März 2007

[Darwin 1859] DARWIN, Charles ; MURRAY, John (Hrsg.): *Origin of species*. 1859

[Eclipse] ECLIPSE FOUNDATION: *Eclipse*. – URL http://www.eclipse.org. – Zugriff: August.2007

[Extreme Programming] JEFFRIES, Ronald E.: *Extreme Programming*. – URL http://www.xprogramming.com. – Zugriff: August.2007

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

[FIPA 2007] : *FIPA specifications*. 2007. – URL http://www.fipa.org/

[Hayes 1978] HAYES, P. J.: The Naive Physics Manifesto. In: MICHIE, D. (Hrsg.): *Expert Systems in the Micro-Electronic Age*. Edinburgh University Press, 1978, S. 242–270

[Heinsohn u. a. 2007] HEINSOHN, Jochen ; SOCHER, Rolf ; BOERSCH, Ingo: *Wissensverarbeitung*. Spektrum Akademischer Verlag, 2007

[Heise Zeitschriften Verlag GmbH & Co. KG 2006] *c't-Bot und c't-Sim*. 2006. – URL http://www.heise.de/ct/projekte/ct-bot/. – Zugriff: Juni.2007

[Hemelrijk und Kunz 2003] HEMELRIJK, Charlotte K. ; KUNZ, Hanspeter: *Artificial Fish Schools: Collective Effects of School Size, Body Size, and Body Form*. PDF. 2003. – URL http://www.rug.nl/biologie/onderzoek/onderzoekgroepen/theoreticalbiology/publications

[Java] SUN MICROSYSTEMS: *Java*. – URL http://java.sun.com. – Zugriff: August.2007

[Jung 2001] JUNG, Matthias: *Design und Realisierung einer Plattform für Informationsagenten*. Oktober 2001

[JUnit] BECK, Kent ; GAMMA, Erich ; MEADE, Erik G. H. ; SAFF, David: *JUnit*. – URL http://www.junit.org. – Zugriff: August.2007

[Koza 1992] KOZA, John R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992. – URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{\&}path=ASIN/0262111705. – ISBN 0262111705

[Lund 2006] LUND, Sven: *Optimierung von Neuronalen Netzen mit Hilfe Genetischer Algorithmen*. März 2006

[Maschmann 2007] MASCHMANN, Chrisitan: *Genetische Lernverfahren zur Optimierung einer Multiagentensimulation für marktwirtschaftliche Vorgänge*. Juli 2007

[Mondada u. a. 2002] MONDADA, F. ; GUIGNARD, A. ; COLOT, A. ; FLOREANO, D. ; DENEUBOURG, J.-L. ; GAMBARDELLA, L. ; NOLFI, S. ; DORIGO, M.: SWARM-BOT: A New Concept of Robust All-Terrain Mobile Robotic System / LSA2 - I2S - STI, Swiss Federal Institute of Technology. Lausanne, Switzerland, März 2002. – Technical Report

[Netbeans] NETBEANS: *Netbeans*. – URL http://www.netbeans.org. – Zugriff: August.2007

[Open Agent Architecture 2007] : *Open Agent Architecture*. 2007. – URL http://www.ai.sri.com/~oaa/

[Pfeiffer und Scheier 2001] PFEIFFER, Rolf ; SCHEIER, Christian: *Understanding Intelligence*. Bradford Books, 2001. – ISBN 0-262-16181-8

[Porschke 2002] PORSCHKE, Jan P.: *Genetische Programmierung von verhaltensbasierten Agenten*. Januar 2002

[Storjohann 2005] STORJOHANN, Timo: *Einsatz genetischer Lernverfahren für die Programmierung eines mobilen Roboters*. August 2005

[Swarm-Bots] : *Swarm Bots*. – URL http://www.swarm-bots.org/

[Weber 2005] WEBER, Max: *Soziologische Grundbegriffe*. S. 1–42. In: *Wirtschaft und Gesellschaft*, Zweittausendeins, 2005. – ISBN 3-86150-730-7

[Wikipedia 2007] : *Die freie Enzyklopädie*. online. 2007. – URL http://de.wikipedia.org/wiki/

[Wooldridge 2002] WOOLDRIDGE, Michael: *MultiAgent Systems*. Wiley, 2002. – ISBN 0-471-49691-X

A. Agentenverhalten

A.1. Versuch 1: fittest Behavior

Generation: 1
Fitness: 23

```
LE(DISTANCE, 3)
 DRIVE(RIGHT)
 EQ(LASTSEEN, GREEN)
  EQ(DISTANCE, 5)
   EQ(LASTSEEN, GREEN)
    EQ(DISTANCE, 4)
     LE(ADDICTION, GREEN)
      DRIVE(LEFT)
      DRIVE(LEFT)
     GE(OPENPROPOSAL, 4)
      GE(ADDICTION, RED)
       LE(OWNADDICTED, 12)
        GE(DISTANCE, 4)
         TRADE
         TRADE
        DRIVE(BACKWARD)
       EQ(OWNNOTADDICTED, 1)
        EQ(OWNADDICTED, 4)
         DRIVE(RIGHT)
         DRIVE(LEFT)
        EQ(OWNNOTADDICTED, 7)
         DRIVE(STOP)
         TRADE
      TRADE
    GE(LASTSEEN, BLUE)
    EQ(LASTFOUND, 20)
     TRADE
     EQ(LASTSEEN, RED)
      GE(OWNNOTADDICTED, 13)
       DRIVE(RIGHT)
       DRIVE(BACKWARD)
      DRIVE(BACKWARD)
```

```
␣␣␣␣␣LE(DISTANCE,␣0)
␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣GE(LASTSEEN,␣GREEN)
␣␣␣␣␣␣␣LE(LASTFOUND,␣3)
␣␣␣␣␣␣␣␣GE(OWNADDICTED,␣2)
␣␣␣␣␣␣␣␣␣DRIVE(RIGHT)
␣␣␣␣␣␣␣␣␣DRIVE(FORWARD)
␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣EQ(DISTANCE,␣3)
␣␣␣␣␣␣␣␣EQ(OWNNOTADDICTED,␣12)
␣␣␣␣␣␣␣␣␣DRIVE(LEFT)
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣LE(OWNNOTADDICTED,␣6)
␣␣␣␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣TRADE
␣␣GE(LASTFOUND,␣20)
␣␣␣LE(DISTANCE,␣1)
␣␣␣␣DRIVE(LEFT)
␣␣␣␣GE(OWNNOTADDICTED,␣0)
␣␣␣␣␣TRADE
␣␣␣␣␣EQ(OWNNOTADDICTED,␣13)
␣␣␣␣␣␣EQ(OWNADDICTED,␣7)
␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣DRIVE(FORWARD)
␣␣␣␣␣␣GE(ADDICTION,␣BLUE)
␣␣␣␣␣␣␣EQ(DISTANCE,␣0)
␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣DRIVE(RIGHT)
␣␣␣␣␣␣␣LE(LASTSEEN,␣RED)
␣␣␣␣␣␣␣␣LE(OPENPROPOSAL,␣5)
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣TRADE
␣␣␣EQ(OWNNOTADDICTED,␣5)
␣␣␣␣LE(ADDICTION,␣RED)
␣␣␣␣␣TRADE
␣␣␣␣␣TRADE
␣␣␣␣DRIVE(FORWARD)
```

Generation: 30
Fitness: 26

```
LE(DISTANCE,␣3)
␣DRIVE(RIGHT)
␣EQ(LASTFOUND,␣10)
␣␣DRIVE(LEFT)
␣␣DRIVE(FORWARD)
```

A.2. Versuch 2: fittest Behavior

Generation: 1
Fitness: 17

```
GE(LASTSEEN, BLUE)
 GE(OPENPROPOSAL, 2)
  LE(OWNADDICTED, 2)
   LE(OPENPROPOSAL, 0)
    EQ(OPENPROPOSAL, 2)
     DRIVE(BACKWARD)
     DRIVE(FORWARD)
    TRADE
   GE(LASTFOUND, 25)
    GE(LASTSEEN, BLUE)
     EQ(DISTANCE, 4)
      TRADE
      EQ(OWNNOTADDICTED, 0)
       GE(OPENPROPOSAL, 1)
        GE(OPENPROPOSAL, 4)
         DRIVE(BACKWARD)
         DRIVE(BACKWARD)
         EQ(LASTFOUND, 4)
         DRIVE(BACKWARD)
         TRADE
       TRADE
     DRIVE(STOP)
    EQ(OPENPROPOSAL, 0)
    TRADE
    EQ(ADDICTION, BLUE)
     GE(OWNNOTADDICTED, 5)
      DRIVE(BACKWARD)
      TRADE
     GE(LASTFOUND, 40)
      GE(OPENPROPOSAL, 5)
       EQ(LASTSEEN, GREEN)
        TRADE
        TRADE
        DRIVE(RIGHT)
       DRIVE(RIGHT)
  EQ(DISTANCE, 2)
   EQ(ADDICTION, GREEN)
   GE(DISTANCE, 1)
   TRADE
   GE(OWNADDICTED, 3)
    DRIVE(RIGHT)
    TRADE
```

```
␣␣␣␣DRIVE(LEFT)
␣␣␣DRIVE(FORWARD)
␣EQ(OWNADDICTED,␣3)
␣␣EQ(OWNNOTADDICTED,␣4)
␣␣␣DRIVE(RIGHT)
␣␣␣LE(LASTSEEN,␣BLUE)
␣␣␣␣TRADE
␣␣␣␣EQ(OWNADDICTED,␣0)
␣␣␣␣␣LE(OWNNOTADDICTED,␣7)
␣␣␣␣␣␣GE(ADDICTION,␣GREEN)
␣␣␣␣␣␣␣DRIVE(STOP)
␣␣␣␣␣␣␣EQ(LASTFOUND,␣2)
␣␣␣␣␣␣␣␣LE(ADDICTION,␣GREEN)
␣␣␣␣␣␣␣␣␣DRIVE(RIGHT)
␣␣␣␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣␣␣EQ(OWNNOTADDICTED,␣12)
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣␣DRIVE(LEFT)
␣␣␣␣␣␣TRADE
␣␣␣␣␣DRIVE(LEFT)
␣␣DRIVE(STOP)
```

Generation: 100
Fitness: 28

```
GE(LASTSEEN,␣RED)
␣GE(DISTANCE,␣5)
␣␣DRIVE(FORWARD)
␣␣EQ(LASTSEEN,␣GREEN)
␣␣␣LE(ADDICTION,␣GREEN)
␣␣␣␣TRADE
␣␣␣␣GE(OWNNOTADDICTED,␣14)
␣␣␣␣␣LE(ADDICTION,␣GREEN)
␣␣␣␣␣␣LE(LASTFOUND,␣1)
␣␣␣␣␣␣␣LE(LASTFOUND,␣10)
␣␣␣␣␣␣␣␣DRIVE(FORWARD)
␣␣␣␣␣␣␣␣DRIVE(LEFT)
␣␣␣␣␣␣␣LE(LASTSEEN,␣GREEN)
␣␣␣␣␣␣␣␣DRIVE(FORWARD)
␣␣␣␣␣␣␣␣GE(DISTANCE,␣4)
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣LE(LASTFOUND,␣1)
␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣TRADE
␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣GE(DISTANCE,␣5)
```

```
␣␣␣␣DRIVE(LEFT)
␣␣␣␣DRIVE(LEFT)
␣GE(LASTFOUND,␣25)
␣␣GE(LASTSEEN,␣BLUE)
␣␣␣EQ(DISTANCE,␣4)
␣␣␣␣TRADE
␣␣␣␣EQ(OWNNOTADDICTED,␣0)
␣␣␣␣␣GE(OPENPROPOSAL,␣1)
␣␣␣␣␣␣GE(OPENPROPOSAL,␣4)
␣␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣EQ(LASTFOUND,␣4)
␣␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣␣TRADE
␣␣␣␣␣TRADE
␣␣␣DRIVE(STOP)
␣␣EQ(OPENPROPOSAL,␣0)
␣␣␣TRADE
␣␣␣EQ(ADDICTION,␣BLUE)
␣␣␣␣GE(OWNNOTADDICTED,␣5)
␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣TRADE
␣␣␣␣GE(LASTFOUND,␣40)
␣␣␣␣␣GE(OPENPROPOSAL,␣5)
␣␣␣␣␣␣EQ(LASTSEEN,␣GREEN)
␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣DRIVE(RIGHT)
␣␣␣␣␣DRIVE(RIGHT)
```

A.3. Versuch 3: fittest Behavior

Generation: 1
Fitness: 17

```
LE(OPENPROPOSAL,␣0)
␣EQ(OPENPROPOSAL,␣1)
␣␣DRIVE(STOP)
␣␣LE(LASTFOUND,␣4)
␣␣␣DRIVE(RIGHT)
␣␣␣DRIVE(BACKWARD)
␣GE(OWNNOTADDICTED,␣15)
␣␣TRADE
␣␣GE(DISTANCE,␣3)
␣␣␣DRIVE(LEFT)
```

```
␣␣␣GE(OWNADDICTED,␣2)
␣␣␣␣LE(DISTANCE,␣0)
␣␣␣␣␣GE(OWNNOTADDICTED,␣6)
␣␣␣␣␣␣LE(DISTANCE,␣2)
␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣EQ(OWNNOTADDICTED,␣5)
␣␣␣␣␣␣␣␣EQ(OWNADDICTED,␣4)
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣LE(OPENPROPOSAL,␣5)
␣␣␣␣␣␣␣GE(OWNADDICTED,␣5)
␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣GE(OPENPROPOSAL,␣4)
␣␣␣␣␣␣␣␣GE(LASTSEEN,␣RED)
␣␣␣␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣␣␣␣DRIVE(FORWARD)
␣␣␣␣␣␣␣␣GE(OWNADDICTED,␣15)
␣␣␣␣␣␣␣␣␣DRIVE(LEFT)
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣DRIVE(FORWARD)
␣␣␣␣LE(OWNNOTADDICTED,␣12)
␣␣␣␣␣DRIVE(RIGHT)
␣␣␣␣␣TRADE
```

Generation: 30
Fitness: 42

```
GE(LASTFOUND,␣0)
␣EQ(LASTFOUND,␣15)
␣␣LE(LASTSEEN,␣BLUE)
␣␣␣LE(OWNNOTADDICTED,␣4)
␣␣␣␣TRADE
␣␣␣␣TRADE
␣␣␣DRIVE(RIGHT)
␣␣LE(DISTANCE,␣4)
␣␣␣GE(LASTFOUND,␣3)
␣␣␣␣EQ(OWNADDICTED,␣10)
␣␣␣␣␣EQ(OWNADDICTED,␣9)
␣␣␣␣␣␣DRIVE(STOP)
␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣EQ(DISTANCE,␣5)
␣␣␣␣␣␣LE(OWNNOTADDICTED,␣15)
␣␣␣␣␣␣␣DRIVE(FORWARD)
␣␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣LE(LASTSEEN,␣RED)
```

```
␣␣␣␣␣␣␣GE(OPENPROPOSAL,␣2)
␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣EQ(OWNADDICTED,␣5)
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣DRIVE(RIGHT)
␣␣␣␣TRADE
␣␣␣DRIVE(FORWARD)
␣LE(DISTANCE,␣4)
␣␣GE(LASTFOUND,␣3)
␣␣␣EQ(OWNADDICTED,␣10)
␣␣␣␣EQ(OWNADDICTED,␣9)
␣␣␣␣␣DRIVE(STOP)
␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣EQ(DISTANCE,␣5)
␣␣␣␣␣LE(OWNNOTADDICTED,␣15)
␣␣␣␣␣␣DRIVE(FORWARD)
␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣LE(LASTSEEN,␣RED)
␣␣␣␣␣␣GE(OPENPROPOSAL,␣2)
␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣EQ(OWNADDICTED,␣5)
␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣␣␣TRADE
␣␣␣␣␣␣DRIVE(RIGHT)
␣␣␣TRADE
␣␣DRIVE(FORWARD)
```

A.4. Versuch 4: fittest Behavior

Generation: 1
Fitness: 7

```
GE(OPENPROPOSAL,␣2)
␣LE(LASTFOUND,␣40)
␣␣TRADE
␣␣DRIVE(RIGHT)
␣GE(LASTFOUND,␣15)
␣␣DRIVE(BACKWARD)
␣␣GE(LASTFOUND,␣50)
␣␣␣GE(LASTSEEN,␣BLUE)
␣␣␣␣DRIVE(RIGHT)
␣␣␣␣TRADE
␣␣␣EQ(ADDICTION,␣RED)
␣␣␣␣LE(OPENPROPOSAL,␣1)
```

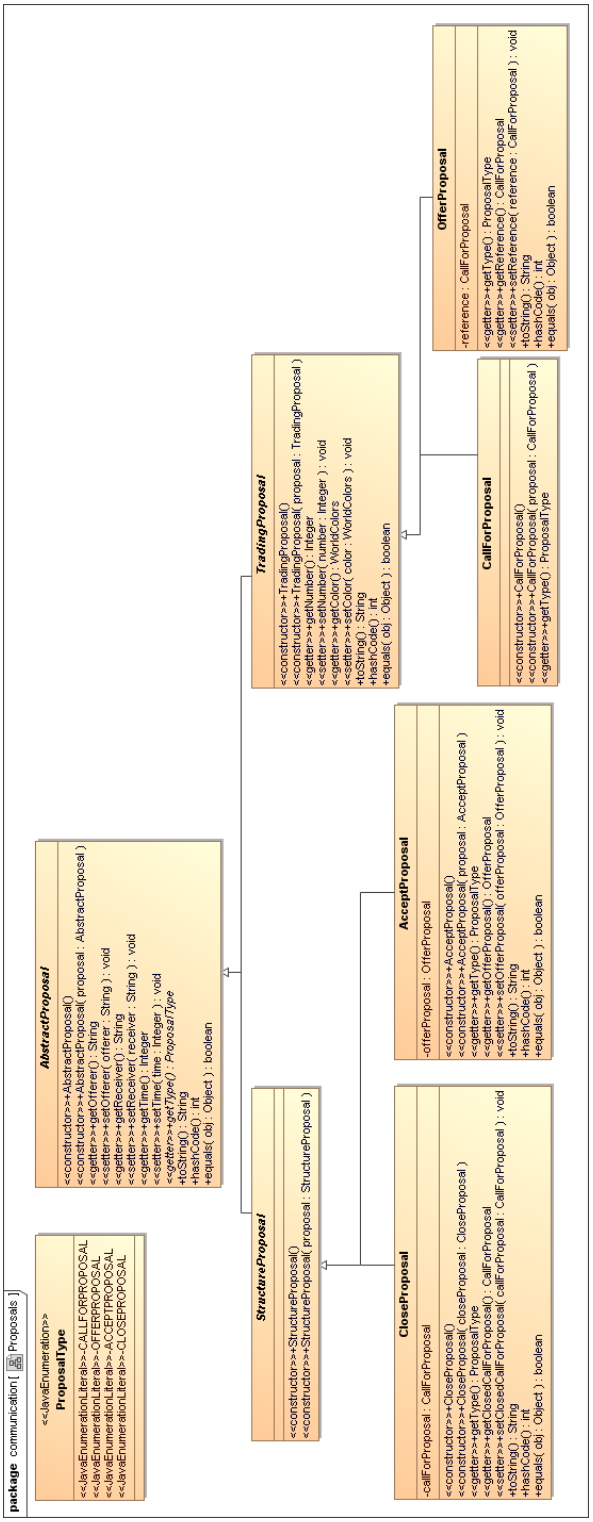
```
␣␣␣␣␣EQ(LASTFOUND,␣0)
␣␣␣␣␣␣DRIVE(BACKWARD)
␣␣␣␣␣␣TRADE
␣␣␣␣␣TRADE
␣␣␣␣DRIVE(LEFT)
```

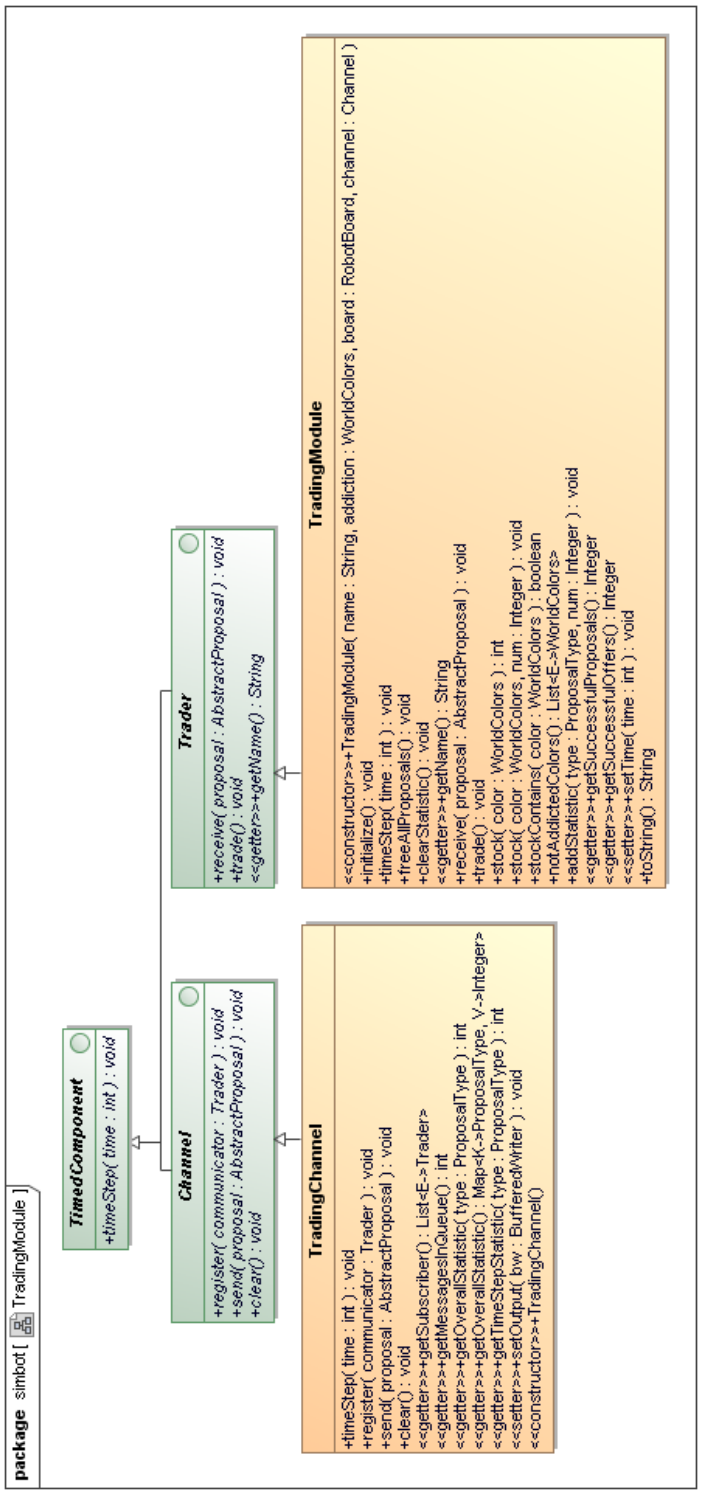
Generation: 100
Fitness: 30

```
LE(DISTANCE,␣4)
␣DRIVE(RIGHT)
␣EQ(ADDICTION,␣RED)
␣␣GE(LASTSEEN,␣BLUE)
␣␣␣TRADE
␣␣␣GE(OWNADDICTED,␣9)
␣␣␣␣DRIVE(RIGHT)
␣␣␣␣TRADE
␣␣LE(DISTANCE,␣0)
␣␣␣DRIVE(LEFT)
␣␣␣GE(LASTFOUND,␣20)
␣␣␣␣DRIVE(FORWARD)
␣␣␣␣TRADE
```

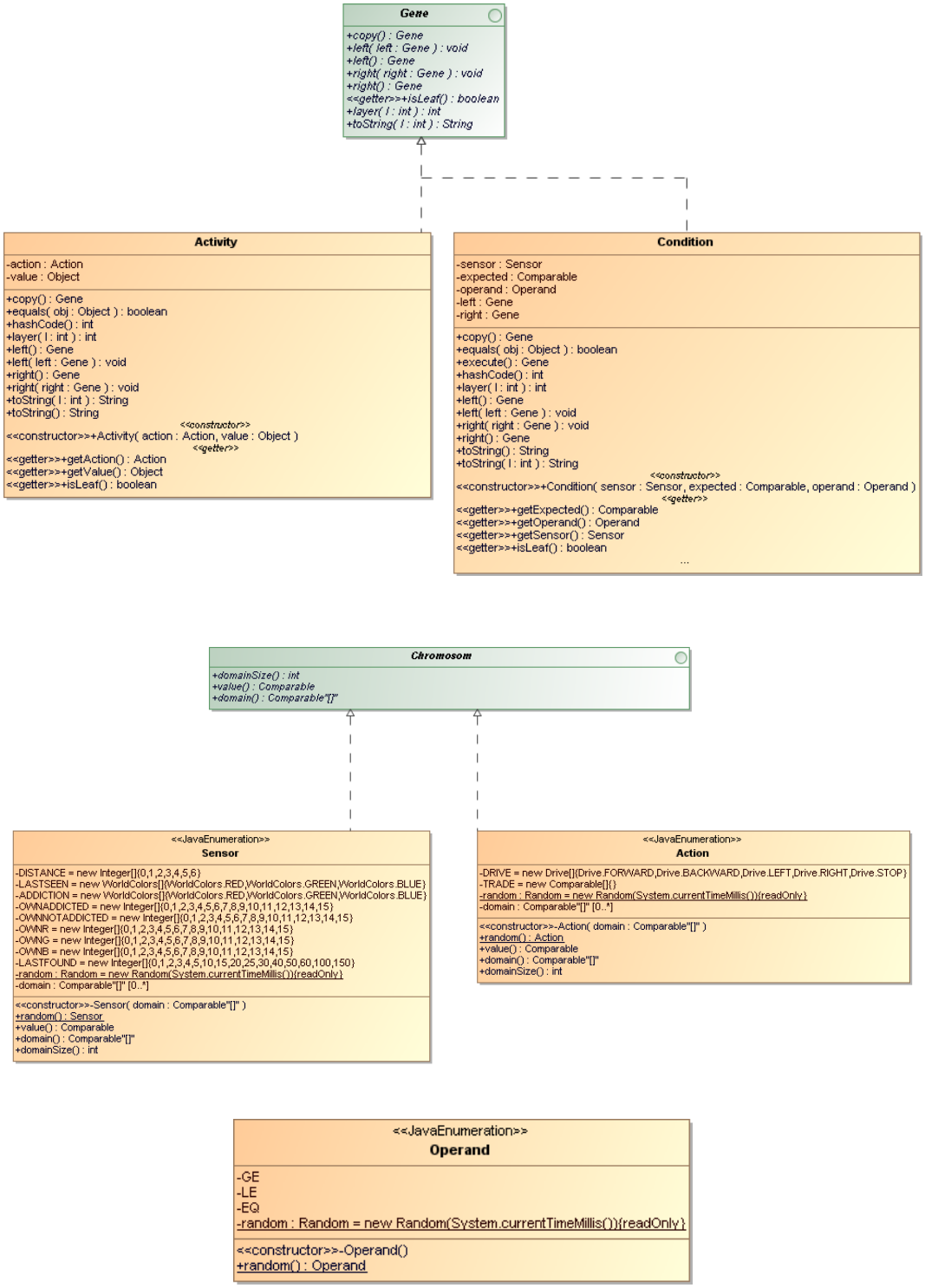
B. UML Diagramme

B.1. Trading


package communication [Proposals]
«JavaEnumeration»
ProposalType
«JavaEnumerationLiteral»CALLFORPROPOSAL
«JavaEnumerationLiteral»OFFERPROPOSAL
«JavaEnumerationLiteral»ACCEPTPROPOSAL
«JavaEnumerationLiteral»CLOSEPROPOSAL
AbstractProposal
«constructor»+AbstractProposal()
«constructor»+AbstractProposal(proposal : AbstractProposal)
«getter»+getOfferer() : String
«setter»+setOfferer(offerer : String) : void
«getter»+getReceiver() : String
«setter»+setReceiver(receiver : String) : void
«getter»+getTime() : Integer
«setter»+setTime(time : Integer) : void
«getter»+getType() : ProposalType
+toString() : String
+hashCode() : int
+equals(obj : Object) : boolean
StructureProposal
«constructor»+StructureProposal()
«constructor»+StructureProposal(proposal : StructureProposal)
TradingProposal
«constructor»+TradingProposal()
«constructor»+TradingProposal(proposal : TradingProposal)
«getter»+getNumber() : Integer
«setter»+setNumber(number : Integer) : void
«getter»+getColor() : WorldColors
«setter»+setColor(color : WorldColors) : void
+toString() : String
+hashCode() : int
+equals(obj : Object) : boolean
OfferProposal
-reference : CallForProposal
«getter»+getType() : ProposalType
«getter»+getReference() : CallForProposal
«setter»+setReference(reference : CallForProposal) : void
+toString() : String
+hashCode() : int
+equals(obj : Object) : boolean
CallForProposal
«constructor»+CallForProposal()
«constructor»+CallForProposal(proposal : CallForProposal)
«getter»+getType() : ProposalType
AcceptProposal
-offerProposal : OfferProposal
«constructor»+AcceptProposal()
«constructor»+AcceptProposal(proposal : AcceptProposal)
«getter»+getType() : ProposalType
«getter»+getOfferProposal() : OfferProposal
«setter»+setOfferProposal(offerProposal : OfferProposal) : void
+toString() : String
+hashCode() : int
+equals(obj : Object) : boolean
CloseProposal
-callForProposal : CallForProposal
«constructor»+CloseProposal()
«constructor»+CloseProposal(closeProposal : CloseProposal)
«getter»+getType() : ProposalType
«getter»+getCloseCallForProposal() : CallForProposal
«setter»+setCloseCallForProposal(callForProposal : CallForProposal) : void
+toString() : String
+hashCode() : int
+equals(obj : Object) : boolean


package simbot [TradingModule]
TimedComponent
+timeStep(time : int) : void
Channel
+register(communicator : Trader) : void
+send(proposal : AbstractProposal) : void
+clear() : void
Trader
+receive(proposal : AbstractProposal) : void
+trade() : void
<<getter>>+getName() : String
TradingChannel
+timeStep(time : int) : void
+register(communicator : Trader) : void
+send(proposal : AbstractProposal) : void
+clear() : void
<<getter>>+getSubscriber() : List<E->Trader>
<<getter>>+getMessagesInQueue() : int
<<getter>>+getOverallStatistic(type : ProposalType) : int
<<getter>>+getOverallStatistic() : Map<K->ProposalType, V->Integer>
<<getter>>+getTimeStepStatistic(type : ProposalType) : int
<<setter>>+setOutput(bw : BufferedWriter) : void
<<constructor>>+TradingChannel()
TradingModule
<<constructor>>+TradingModule(name : String, addiction : WorldColors, board : RobotBoard, channel : Channel)
+initialize() : void
+timeStep(time : int) : void
+freeAllProposals() : void
+clearStatistic() : void
<<getter>>+getName() : String
+receive(proposal : AbstractProposal) : void
+trade() : void
+stock(color : WorldColors) : int
+stock(color : WorldColors, num : Integer) : void
+stockContains(color : WorldColors) : boolean
+notAddictedColors() : List<E->WorldColors>
+addStatistic(type : ProposalType, num : Integer) : void
<<getter>>+getSuccessfullProposals() : Integer
<<getter>>+getSuccessfullOffers() : Integer
<<setter>>+setTime(time : int) : void
+toString() : String

B.2. Genotypen



Glossar

Attraktion ist ein Objekt in der realen / simulierten Welt, nach dem die Roboter suchen, bzw. das die 'Aufmerksamkeit' der Roboter auf sich zieht.

Behavior ist ein Agentenverhalten, aus Sicht der genetischen Optimierung stellt es ein Gen dar.

ct-bot Der ct-bot ist ein Projekt des ct-Magazins aus dem Heise Verlag (Heise Zeitschriften Verlag GmbH & Co. KG (2006)). Ein einachsiger Roboter mit einem 8 Bit Atmel Microcontroller und diversen Sensoren, wie Distanz-, Linien-, Tischkantensensor, als auch Radencoder. Dieser Roboter wird in C programmiert, der Heise Verlag bietet hierfür ein Framework zum besseren Einstieg an.

Emergenz ist das 'Erscheinen' von Phänomenen auf der Makroebene eines Systems, die erst durch das Zusammenwirken der Subsysteme (die Systemelemente auf der Mikroebene) zustande kommen (Wikipedia (2007))

Genotyp In der genetischen Programmierung ist ein Genotyp ein Programm oder Zustandsautomat. Die Sprache, mit der Genotypen beschrieben werden können, muss die Eigenschaft besitzen, nach genetischen Operationen wie Mutation oder Crossover einen syntaktisch korrekten Genotypen zu erhalten.

Setting ist ein Set von Agentenverhalten, aus Sicht der genetischen Optimierung stellt es einen Genotyp dar.

Sprechakt Theorie Die Sprechakt Theorie wurde von den Philosophen John Austin und Searle begründet. Austin hat die Grundlagen (1962) der Theorie geschaffen und sein Schüler John Searle hat diese Theorie später in seinem Buch 'Speech Acts' ausgebaut. Danach verändern Sprechakte die Welt im physikalischen Sinne. Nachrichten lösen eine Handlung in dem Zuhörer aus oder haben die Intention dieses zu tun. Genauer ist es zum Beispiel in Jung (2001) S. 23ff zu lesen.

Index

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 2007	
Ort, Datum	Unterschrift