Genetisches Programmieren zur Verhaltens-optimierung von Artificial-Life-Agenten

- Diplomarbeit -

im

Fachbereich Informatik
der Universität Dortmund

von

Jens Ch. Lisner

13. Juli 1998

Erstgutachter : Prof. Dr. W. Banzhaf
Zweitgutachter : Dipl. Inf. R. Keller

Universität Dortmund
Fachbereich Informatik
Lehrstuhl für Systemanalyse
D-44221 Dortmund

Inhaltsverzeichnis

Einleitung

Philia ist das Ergebnis der Projektgruppe „Realisierung und Anwendung eines Genetic-Programming/Artificial-Life-Systems", an der Universität Dortmund im SS 95 und WS 95/96. Das Ziel war es die verschiedenen Ansätze Genetic-Programming und Artificial-Life in einem Modell zu verbinden. Dabei sollte das Modell die Beobachtung von aus der Natur bekannten Phänomene ermöglichen. Dazu wurde eine Umgebung geschaffen, die kooperatives[1] und konkurrierendes Verhalten ermöglicht (s. [1]).

Die Frage nach explizit modellierten Räuber-Beute-Systemen kam auf, nachdem sich in der Projektgruppe (kurz: PG) gezeigt hat, daß einfachere Modelle nicht von selbst zur Bildung solch komplexer Strukturen in der Lage sind. Konkurrierender Spezies soll nun eine Entwicklung zu elaboriertem Systemverhalten erlauben. Dies soll durch Koevolution auf Programmebene ermöglicht werden. Koevolution ermöglicht die kontinuierliche Weiterentwicklung über ein bestimmtes Systemverhalten hinaus. Damit werden immer neue Formen der Organisation ermöglicht.

In Kapitel 1 sollen die grundlegenden Begriffe und Disziplinen kurz erörtert werde. Kapitel 2 beschäftigt sich mit dem Philia-Modell, und seiner Funktionsweise. Es stellt gleichsam eine Einordnung in schon vorhandene Disziplinen dar, und stellt einige Besonderheiten vor, die sich aus seiner Rolle als „Zwittersystem" ergeben.

Kapitel 3 stellt das dieser Arbeit zugrunde liegende Räuber-Beute-Modell vor, und zeigt grundlegende Gemeinsamkeiten in der Entwicklung solcher Systeme auf. Darauf aufbauend werden in Kapitel 4 verschiedene Simulationsreihen durchgeführt, die vor allem der Frage nach einer funktionierenden Koevolution in Räuber-Beute-Systemen nachgehen sollen.

In Kapitel 5 werden die Ergebnisse nach den Hauptgesichtspunkten Emergenz, Koevolution und implizite Fitneß zusammengefaßt.

[1] „Philia" ist das griechische Wort für „Freundschaft":

Kapitel 1

Grundlagen

1.1 Artificial Life

Es gibt verschiedene Versuche das Leben zu charakterisieren. In den klassischen philosophischen Lehren ist „leben" gleichbedeutend mit dem Besitz einer Seele. Bei Aristoteles liegt dem Leben der Entelechiegedanke zugrunde (s. [21] und [37]). Eine Entelechie besitzt die Eigenschaften des Selbstseins, der Selbstbewegtheit und der Unsterblichkeit. Die Entelechie ist notwendigerweise mit der Materie verknüpft. Im Rationalismus wurde der Seele (und damit dem Leben) eine eigene Substanz zugesprochen, die entweder von der Materie getrennt (z.B. DÉCARTES [12]), oder mit ihr identifiziert wurde, insofern als ihnen der gleiche „Urstoff" (die Monaden s. LEIBNIZ [33]) zugrunde liegen. Der Materialismus trennt sich von der Vorstellung des Lebens als autonome Substanz, und stellt die physikalischen und chemischen Prozesse der Materie in den Vordergrund. Die Biologie interessiert sich für natürlich gewachsenes Leben, dem unter anderem die Eigenschaften Stoffwechsel, Fortpflanzung, Vererbung, Reizbarkeit und der Aufbau bestimmter Strukturen zugrundeliegt.

Dem Artificial Life (AL) liegt eine andere Betrachtungsweise zugrunde, die den Begriff des Lebens von den Eigenschaften der Materie selbst trennt, und es als Eigenschaft der Struktur des Systems, in dem die Materie organisiert ist, sieht. Im Artificial Life steht das Systemverhalten im Vordergrund. Damit wird im Prinzip Leben[1] auf Basis von Turing-Maschinen möglich.

AL bedeutet „life made by humans rather than by nature" ([32]). Dabei dienen dem AL aber nicht nur biologische Systeme als Vorbild (z.B. [36]). Für LANGTON steht vor allem der synthetische Forschungsansatz im Mittelpunkt. Danach ist AL „an attempt to increase vastly the role of syntheses in the study of biological phenomena", also der Versuch, neben der klassischen Analyse empirischer Daten, eine Erforschung des Phänomens „Leben" durch Synthese komplexer Organismen.

[1] Der Begriff „Leben" wird im folgenden immer im Sinne der Betrachtungsweise der Artificial Life gebraucht.

Im Vordergrund steht dabei der Begriff der Emergenz. Emergentes Verhalten entsteht aus der Zusammenwirkung einzelner Elemente des Systems, die nach einfachen Regeln gesteuert werden. Kennzeichnend ist, daß emergentes Verhalten nicht in das System hineinmodelliert wird. durch wen? durch ZB Programmierer;

1.2 Genetic Programming

Beim Genetic Programming geht es um eine zielgerichtete Codeoptimierung zur Lösung durch eines, von einer expliziten Fitneßfunktion, vorgegebenen Problems (s. KOZA [27]).

Die Lösungen werden in einer Programmiersprache codiert. Die Programme werden zusammengesetzt aus einer Menge von Funktionen und Terminalen. Funktionen sind dadurch ausgezeichnet, daß sie mindestens ein Argument besitzen. Beispiele dafür sind:

- boolesche Funktionen
- arithmetische Funktionen
- Kontrollstrukturen (z.B. if, while, seq)

Terminale dagegen haben keine Argumente. Beispiele sind:

- Konstanten
- Variablen

In diesem Zusammenhang definiert man die Arität einer Funktion als die Anzahl ihrer Argumente. Folglich haben Terminale eine Arität von Null. ist also ein Terminal eine Fkt.

Der Algorithmus beginnt mit der Erzeugung einer Initialpopulation, also einer Menge von vorläufigen Lösungen, die zufällig generiert werden*. Danach werden die Lösungen anhand der Fitneßfunktion evaluiert. Dadurch wird ihnen ein Fitneßwert zugeordnet. Aus der Population werden nun die besten Lösungen aussortiert, und durch Crossing-Over, Reproduktion und ggf. Mutation dieser Lösungen eine neue Population generiert. Dabei ist es möglich auch einige schlechter bewertete Lösungen in den Crossing-Over Prozeß mit aufzunehmen, um eine größere Streuung der Kindpopulationen im Lösungsraum zu erhalten. Auf diese Weise kann ein lokales Maximum überwunden werden. Ist die Fitneß groß genug wird der Prozeß abgebrochen.

*) oder gezielt generiert;

1.3 Koevolution

Dieses Verfahren stammt Ursprünglich aus dem Bereich der Genetic Algorithms (GA). Es handelt sich um ein Verfahren den evolutiven Prozeß zu optimieren. Das Verfahren hebt sich insofern von der klassischen GA ab, als daß zwei Populationen generiert werden, von denen eine neue Eingaben für die Programme der jeweils anderen Population produziert, mit dem Ziel, in diesen Programmen Schwachstellen ausfindig zu machen. Diese Form der Koevolution stellt bereits ein klassisches Räuber-Beute-Szenario dar.

Beispielsweise versuchte Hillis (s. [24]) einen Sortieralgorithmus mittels Koevolution zu optimieren. Während eine Population Sortierverfahren enthielt, bestand die andere Population aus Permutationen, die als Trainingsdaten dienten. Dabei wurden die Sortieralgorithmen nach der Menge der gelösten Aufgaben, und die Permutationen, nach der Menge der Programme, die nicht in der Lage waren, die Permutation richtig zu sortieren, bewertet. Dieses Verfahren führt in kürzerer Zeit zu qualitativ besseren Ergebnissen. (wahrscheinlich ein Bsp.)

Kapitel 2

Philia: Ein Ansatz mit GP- und AL-Elementen

In Philia wurden die beiden Ansätze GP und AL in einem Modell vereint. Aus der AL wurde die Idee der Weltmodellierung, der Entwicklung von Populationen autonomer Agenten und der Emergenz übernommen. Aus der GP wurde die Idee der Codeoptimierung durch Crossing-Over mehrerer Instanzen von Lösungen in einer Generation, und Mutation in das Modell übernommen. In Philia sind beide Ansätze durch eine simulierte AL-Welt und durch Agentenprogramme, die die Steuerung individueller Agenten übernehmen, vertreten. Beide Komponenten werden einerseits zusammengefügt, durch die Einwirkungen der Funktionen im Agentenprogramm auf die Umwelt, andererseits durch die als Antwort erzeugte Rückkopplung der Umwelt auf die Agenten und die ständig wechselnde Umgebung bedingt durch die unabhängigen Regeln der AL-Welt. Die für GP typische explizite Fitneßfunktion ist in Philia durch die Regeln der AL-Komponente ersetzt. Die Auswahl der Eltern für die nächste Generation erfolgt durch das Zusammenspiel der AL-Welt und den individuellen Agentenprogrammen. Die Fitneßfunktion liegt somit implizit im Zusammenspiel der Systemkomponenten, also im System selbst. Die sich ständig ändernden Bedingungen verändern im Idealfall auch das Optimierungsziel, und damit die implizite Fitneß. Die Fitneß wäre so einem ständigem Wechsel unterworfen (siehe Abschnitt 2.1.6).

In Philia können mit Hilfe von Agententypen Gruppen mit unterschiedlichen Eigenschaften gebildet werden. Dieses Verfahren macht es möglich den Funktionen Constraints (siehe Abschnitt 2.1.9) aufzuerlegen, um Agententypen eine Rolle als Räuber oder Beute zuzuweisen.

Die Populationen in Philia sind in der Lage Strategien zu entwickeln, wie sie auch in der Natur vorkommen. Die Funktionen und Terminale wurden so gewählt damit dies möglich wird. (siehe Abschnitt 2.1.4) Beispiele für mögliche Emergenzphänomene in einem Räuber-Beute-Szenario wären:

- Herdenbildung (auf Seiten der Beute)

- Rudelbildung (auf Seiten der Räuber)
- Jagdstrategien der Räuber
- Warn- und Fluchtmechanismen der Beute

Im ggs. zu ähnlichen Systemen, die auf GA basieren (z.B. [2], [39] und [50]), gibt es in Philia keine simulierte DNS. Es wird durch die Programme kein äußeres Erscheinungsbild, oder keine Fähigkeiten abgebildet, sondern statische Verhaltensweisen. Eine Fähigkeit zur Mimikry z.B. ist ausgeschlossen. In dem System Tierra werden die Programme als Bitstrings abgebildet. Es gibt hier allerdings keinen Operator, der das Crossing-Over ermöglicht.

2.1 Elementare Regeln in Philia[1]

Die Welt wurde Zeit- und Raumdiskret modelliert. Sie ist eine Ebene und sowohl vertikal als auch horizontal begrenzt. Es gibt drei verschiedene Objektklassen: Food, Carrion und Agenten. Auf jedem Feld darf sich nur ein Objekt befinden. Jedes Objekt stellt für einen Agenten ein unüberwindliches Hindernis dar. Während der Simulation kann zwischen den Objekten der verschiedenen Klassen Energie ausgetauscht werden. Alle Objekte enthalten eine bestimmte Menge Energie.

2.1.1 Food

Die Energiemenge der Foodobjekte ist für alle gleich, und verändert sich nicht. Ein Foodobjekt kann aber von einem Agenten „gefressen" werden. in diesem Fall wird das Objekt entfernt, und seine Energie wird dem Agenten zugeschlagen. Das „Fressen" von Food durch einen Agenten eines bestimmten Typs kann durch Constraints verboten werden. Food kann zu Beginn eines Zeitschrittes auf einem leeren Feld entstehen. Dann ist es sofort für die Agenten verfügbar.

2.1.2 Carrion

Carrion enthält die Energiemenge, die nach dem Tod eines Agenten übriggeblieben ist. Auch Carrion kann von Agenten gefressen werden, und der Konsum durch Constraints eingeschränkt werden. Auch beim Fressen von Carrion wird die darin erhaltene Energiemenge dem Agenten zugeschlagen.

[1] Einige Möglichkeiten von Philia, die hier nicht benutzt werden, sind im folgenden nicht aufgeführt. Nähere Informationen in [1].

2.1.3 Agenten

Agenten sind die komplexesten Objekte in Philia. Sie bestehen aus drei Teilen:

1. Attribute sind statische Parameter, die direkt von den Eltern an die Kinder weitervererbt werden, und sich so während der ganzen Simulation nicht verändern. Sie definieren so die Agententypen und legen ihre charakteristischen Eigenschaften fest. Einige Attribute sind für alle Agententypen gleich, und definieren auf diese Art allgemeine Eigenschaften von Agenten in der ganzen AL-Welt. Die Attribute sind:

   MinMatingAge
   Das Mindestalter des Agenten um sich paaren zu dürfen.

   CostOfLiving
   Die Energie, die der Agent pro Zeitschritt verliert.

   MaxEnergy
   Gibt an, wie hoch der Statuswert **Energy** maximal werden darf.

   MinEnergyFactor
   Prozentualer Wert, bezogen auf **MaxEnergy**, bei dessen Unterschreitung ein Agent stirbt.

   HungerFactor
   Prozentualer Wert, bezogen auf **MaxEnergy**, bei dessen Unterschreitung das Prädikat *Hungry?* erfüllt ist.

   NrOfChildren
   Gibt die Anzahl der Agenten an, die bei einer Paarung erzeugt werden.

   OffspringRadius
   Bei einer erfolgreichen Paarung wird versucht, die in einem Rechteck der Kantenlänge **OffspringRadius** die Anzahl von **NrOfChildren** Agenten einzufügen.

   MatePause
   Anzahl der Zeitschritte, die nach einem erfolgten *Mate* verstreichen muß, bis ein erneutes *Mate* erfolgreich sein kann.

   MaxAge
   Maximale Anzahl an Zeitschritten, die der Agent leben kann.

   MaxFriends
   Maximale Anzahl möglicher Friends.

   RangeOfHearing
   Hörweite des Agenten.

RangeOfVision

Sichtweite des Agenten.

Strength

Stärke des Agenten. Dieses Attribut gibt an, ob der angegriffene Agent bei einer Attacke unterliegt.

2. Der Status der Agenten gibt die Situation des Agenten zu einem bestimmten Zeitpunkt wieder. Die Statuswerte ändern sich dynamisch von Zeitschritt zu Zeitschritt, je nach verhalten des Agenten, und werden daher maßgeblich sowohl vom GP-Teil durch die Strategieprogramme und vom AL-Teil durch die vom Agenten in den Aktionen wahrnehmbare Welt, mit der er interagieren kann, beeinflußt. Die folgenden Parameter legen den Status fest:

XPosition

Die Horizontale Position des Agenten in der Welt.

YPosition

Die Vertikale Position des Agenten in der Welt.

Orientation

Die Orientierung des Agenten auf einem Feld. Dieser Wert kann sich in 45° Schritten durch Aktionen des Agenten ändern.

Energy

Die dem Agenten zur Verfügung stehende Energie. Sie kann den Wert **MaxEnergy** nicht überschreiten. Wird dagegen der Wert $MaxEnergy * MinEnergyFactor$ unterschritten stirbt der Agent und wird in Carrion umgewandelt. Die Energie wird am Ende jedes Zeitschritts um den Wert **CostOfLiving** erniedrigt.

Age

Das Alter der Agenten. Die Agenten sterben, sobald ihre Alter den Wert **MaxAge** überschritten hat.

Memory

Ein 1-Bit-Speicher, den die Agenten zur Ablaufsteuerung in ihren Programmen nutzen können, und der auch von den Agenten selbst gesetzt wird.

LastMate

Enthält die Anzahl der Zeitschritte die noch vergehen müssen, bis der Agent wieder einen Paarungsversuch unternehmen kann.

Screams

Anzahl der Schreie die der Agent bei der letzten Ausführung seines Strategieprogramms ausgestoßen hat. Dieser Wert wird vor einer erneuten Ausführung wieder gelöscht.

Friends

Eine Liste von „befreundeten" Agenten.

3. Das Strategieprogramm, welches für jeden Agenten individuell und statisch ist. Das Programm wird vererbt durch Crossing-Over der Elternprogramme und ggf. Mutation. Es wird implementiert in einer Philia eigenen Strategiesprache. Das Programm ist als dreiadrige Baumstruktur implementiert. Die Tiefe des Baums ist beschränkt auf maximal acht und mindestens vier Ebenen. Jeder Knoten enthält genau eine Funktion. Die Strategien sind für alle Agenten zwar individuell, trotzdem können innerhalb eines Typs Gruppen von Agenten entstehen, die ähnliches Verhalten zeigen, das sich deutlich von dem anderer Gruppen unterscheidet.

In diesem Zusammenhang sind noch einige Begriffe wichtig, die zwar nicht im Status des Agenten festgehalten werden, sich aber aus der Position, Orientierung, Sicht- und Hörweite des Agenten ergeben:

Aktionsfeld

Dieses Feld, das genau vor dem Agenten in seiner Orientierung liegt, ist das Feld auf das sich die meisten einfachen Aktionen (z.B. Eat, Mate und alle Here? Funktionen) beziehen.

Sehstrahl

Alle Felder bis zu einer maximalen Sichtweite, die in der Orientierung des Agenten liegen. Wird von den Ahead? Funktionen verwendet.

Hörbereich

Alle Felder um den Agenten, die im Radius der Hörweite des Agenten liegen.

2.1.4 Die Strategiesprache in Philia

Die Strategiesprache in Philia ist eine Sprache mit LISP-ähnlicher Syntax. Die Ausdrücke sind vollständig geklammert, die Funktionen haben maximal drei Argumente, und sind semantisch so gewählt, das auch bei zufallsgenerierten Programmen keine syntaktischen Fehler auftreten können. Jede Funktion liefert einen booleschen Wert zurück. Jeder Funktion sind Kosten zugeordnet. Ein Programm wird terminiert, wenn die Summe der Kosten aller bisher ausgeführten Funktionen den Wert **MaxCost** (Kostentabelle siehe Abschnitt 3.1) überschreitet, oder das Programm vollständig abgearbeitet ist.

Die Funktionen lassen sich nach booleschen und Strukturfunktionen, Prädikaten und Aktionen gruppieren.

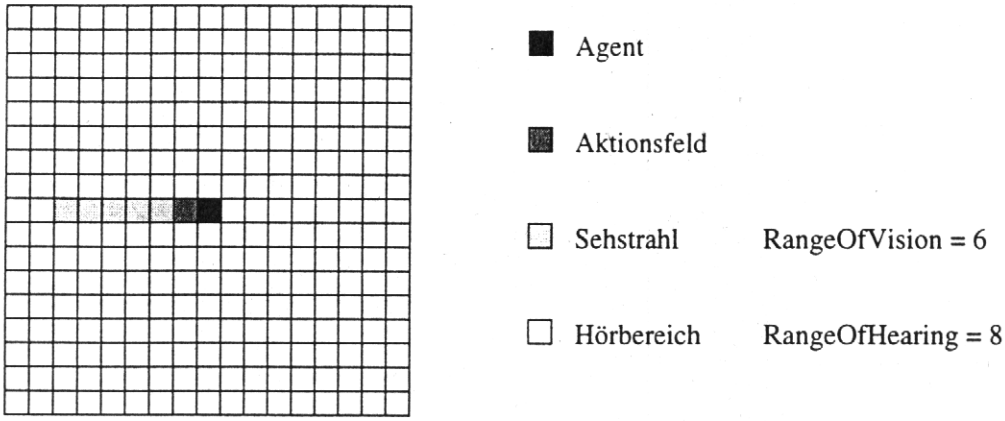

Abbildung 2.1: Aktionsfeld, Sehstrahl und Hörbereich um den Agenten, mit Beispielwerten.

- Als boolesche Funktionen stehen die Operatoren *And*, *Or* und *Not* zur Verfügung. Für die *And* und *Or* Operationen gilt das sie abweisend implementiert wurden.
- Die *Seq*-Funktion ist ein reiner Strukturbefehl, und führt seine Argumente sequentiell aus. Der Rückgabewert ist gleich dem Rückgabewert des letzten Argumentes.
- *If* evaluiert ihr erstes Argument. War es erfüllt, wird das zweite Argument evaluiert, andernfalls das Dritte. Der Rückgabewert ist vom ausgeführten Teilbaum abhängig.

Für die im folgenden aufgeführten Prädikate gilt, das sich die *Here?*-Funktionen auf das Aktionsfeld, und die *Ahead?*-Funktionen auf den Sehstrahl beziehen. Für diese Funktionen gilt, das jedes andere Objekt wonach Ausschau gehalten wird ein Hindernis darstellt, und somit der Sehstrahl verkürzt wird.

- *Hungry?* ist erfüllt falls die Energie unter $MaxEnergy * HungerFactor$ fällt.
- *UnderAttack?* ist erfüllt, falls der Agent angegriffen wurde. Durch die hier zugrundegelegten Parameter ist das Prädikat allerdings niemals erfüllt.[2] *UnderAttack?* wurde Ursprünglich für einen anderen Kampfmodus eingeführt, der einen gegenseitigen Schlagabtausch vorsah. Dieser Modus erwies sich allerdings als schwer zu erlernen, und wird hier aus Gründen der Komplexität nicht benutzt.
- *Sound?* ist erfüllt, wenn im Hörbereich des Agenten ein Friend einen Schrei ausgestoßen hat.
- *Mem?* gibt den Wert von **Memory** im Status des Agenten zurück.

- *AgentHere?* und *AgentAhead?* sind erfüllt, falls ein Agent ausgemacht wurde.
- *FoodHere?* und *FoodAhead?* sind erfüllt, falls ein Food- oder Carrionobjekt entdeckt wurde.
- *FriendHere?* und *FriendAhead?* sind erfüllt, falls ein Agent entdeckt wird der in der Friendliste des ausführenden Agenten vorkommt.
- *NothingHere?* und *NothingAhead?* sind erfüllt, falls nichts gefunden werden konnte.

Die Aktionen sind die Terminale, die keine Prädikate sind. Der Rückgabewert der Aktionen ist fast ausschließlich vom Erfolg bestimmt.

- *Attack* greift den Agenten auf dem Aktionsfeld an. Ist die Aktion erfolgreich, so erhält der Agent die Energie des Angegriffenen, bis auf seine Minimalenergie erniedrigt um 1. Die Maximalenergie des Angreifers wird jedoch nicht überschritten.
- *Mate* strengt einen Paarungsversuch mit dem Agenten auf dem Aktionsfeld an. Ist diese Aktion erfolgreich, so wird ein neuer Agent generiert, dessen Programm durch die GP-Mechanismen Crossing-Over und Mutation entsteht. Die Aktion mißlingt, falls kein Agent auf dem Aktionsfeld vorhanden ist, die Paarung mit einem Agenten dieses Typs durch Constraints verboten wurde, oder daß im Radius **OffspringRadius** um den Agenten kein Platz mehr für den Child ist.
- *Eat* nimmt den Wert der Energie eines auf dem Aktionsfeld liegenden Food- oder Carrionobjektes auf und zerstört es.
- *MakeFriend* nimmt einen auf dem Aktionsfeld stehenden Agenten gleichen Typs in die Friendlist auf. Überschreitet die Anzahl der Friends den Wert **MaxFriends**, so wird der älteste Friend aus der Liste entfernt.
- *Scream* stößt einen Schrei aus. Diese Aktion ist immer erfolgreich.
- *Move* läßt den Agenten die Position auf das Aktionsfeld wechseln, sofern es frei ist.
- *TurnLeft/-Right* bewirken beide eine Links- oder Rechtsdrehung um 45°. Diese Aktionen sind immer erfolgreich.

[2]In der Implementierung aus der PG verbarg sich hier eine Ambiguität: In manchen Fällen war es möglich, daß ein Agent nach seinem Tod noch in der Lage war sein Programm auszuführen, da der Umwandlungsprozeß des toten Agenten in Carrion erst zum Ende des Timesteps erfolgte. Dies galt jedoch nur für Agenten, die von einem älteren Agenten attackiert wurden. In der hier zugrundeliegenden Version können Agentenprogramme nach dem Tod nicht mehr ausgeführt werden. Dadurch verliert *UnderAttack?* seine Funktion.

- *TurnSound* führt eine Drehung in Richtung desjenigen Friends aus, der im Hörbereich den höchsten **Screams**-Wert hat.
- *Nop* ist immer erfüllt, und hat keine Auswirkungen.

Zusätzlich gibt es noch die Funktion *SetMem*, die den Wert **Memory** im Status des Agenten auf den Wert des Unterbaumes setzt, und den alten Wert zurückgibt.

Im Gegensatz zu anderen GP-Systemen verwendeten Sprachen kennt die Strategiesprache in Philia keine Konstanten im eigentlichen Sinne. Da der Rückgabewert von einigen Funktionen allerdings konstant ist, lassen sich diese Funktionen als Quasi-Konstanten verstehen. Die einzige Variable die ein Philia-Programm direkt unter Kontrolle hat ist **Memory** im Status des Agenten. Alle anderen Funktionen lassen sich als fremdbeeinflußte Variablen verstehen.

2.1.5 Interaktionsmöglichkeiten

Aus der Strategiesprache ergeben sich Interaktionsmöglichkeiten, die in den folgenden Konzepten zusammengefaßt werden. Dabei sind die das jeweilige Konzept unterstützenden Funktionen aufgelistet.

Sichtkonzept

Agenten können andere Agenten, Food oder leere Felder erkennen, und mit entsprechenden Aktionen reagieren. Funktionen die das Konzept unterstützen sind: *AgentHere?*, *AgentAhead?*, *FriendHere?*, *FriendAhead?*, *FoodHere?*, *FoodAhead?*, *NothingHere?*, *NothingAhead?*.

Zum Sichtkonzept gehören nur Prädikate, obwohl ein negativer Rückgabewert von z.B. *Move* auch als *Not NothingHere?* interpretiert werden kann. Dies gilt allerdings im allgemeinen nicht mehr für einen positiven Rückgabewert, da der Agent dann seine Position gewechselt hat, weshalb diese Zuordnung seine Berechtigung hat.

Friendkonzept

Agenten können versuchen eine begrenzte Anzahl von Freunde zu gewinnen, und sie als solche zu erkennen. Diese sich daraus ergebende Interaktionsmöglichkeit ist auf Agenten eigenen Typs beschränkt. Die unterstützenden Funktionen sind: *MakeFriend*, *Scream*, *TurnSound*, *Sound?*, *FriendHere?*, *FriendAhead?*.

Soundkonzept

Agenten können mit Friends über größere Entfernungen in Kontakt treten. Die unterstützenden Funktionen sind: *Scream*, *TurnSound*, *Sound?*.

Das Soundkonzept ist Teil des Friendkonzepts, da die Soundfunktionen sich ausschließlich auf Friends beziehen.

2.1.5.1 Der GP-Teil von Philia

Mit der Funktion *Mate* wird der GP-Teil von Philia, der für die Entstehung eines neuen Agenten verantwortlich ist, in Gang gesetzt. Ein Child der an dem Prozeß beteiligten Parents erhält ein eigenständiges Programm, das durch Crossing-Over der Parentprogramme entsteht. Dabei ist eine Mutation möglich.

Beim Crossing-Over wird ein Knoten aus dem Baum des ersten Parent ausgewählt, und der Unterbaum durch den Unterbaum eines zufällig gewählten Knoten im Baum des zweiten Parent ersetzt. Wird die Maximaltiefe von acht Ebenen überschritten, werden die überzähligen Ebenen abgeschnitten, und die Unterbäume durch *Nop* ersetzt, es sei denn es kommt zur Mutation.

Die Mutation in Philia ist optional. Es gibt zwei Verfahren:

1. Mit einer gewählten Wahrscheinlichkeit dazu kommen, daß ein ganzer Teilbaum des Codes bei der Entstehung des Agenten nach dem Crossing-Over durch einen zufällig erzeugten Teilbaum ersetzt wird. Dabei wird das oben beschriebene Verfahren mit einem zufällig erzeugtem Baum wiederholt.[3] Dieses Verfahren wird im folgenden Baummutation genannt.

2. In einem zweiten Verfahren, wird nach jedem Crossing-Over die Abgeschnittenen Teilbäume durch zufällige Funktionen ersetzt. Dies ist der klassische Mutationsoperator in Philia, und wird im folgenden Punktmutation genannt.

Der Status des Agenten wird mit Default-Werten belegt, die im Worldfile festgelegt sind (siehe Abschnitt 3.1).

Der zweite Teil eines GP-Systems, die Auswahl von Agenten, die als erfolgreich gelten können, ist durch die implizite Fitneß im System inhärent bereitgestellt (siehe Abschnitt 2.1.6).

2.1.6 Implizite Fitneß

Codeoptimierung in Philia läßt sich nicht in der gleichen Art verstehen wie in normalen GP-Systemen. Als erfolgreich wird eine Strategie verstanden, die zur Entwicklung von emergentem Verhalten beiträgt. Die Strategien werden aber nicht bezüglich einer vorgegebenen Fitneßfunktion funktional bewertet. Es wird lediglich die Bedingung gestellt, daß die Population nicht allzu früh ausstirbt.[4] In einen reinen GP-System wird eine Lösung funktional bewertet und ihr eine Fitneß zugeordnet. In Philia stehen dagegen die ausgeführten Aktionen im Mittelpunkt, die den Agenten mit der

[3] Diese Option ist erst nach Ende der PG eingeführt worden.

[4] Dadurch soll sichergestellt werden, daß sich überhaupt eine Ökologie entwickelt hat, die die Rahmenbedingungen für eine mögliche Codeoptimierung zur verfügung stellt.

Umwelt interagieren lassen. Durch diese Interaktion entsteht emergentes Verhalten. Die Evolution in Philia ist also keinesfalls zielgerichtet. Das besondere an dem Philia Modell ist somit, daß die Fitneß sich mit den Bedingungen der Umwelt verändern, und somit immer wieder neue Optimierungsziele für die einzelnen Agenten entstehen können. Das heißt aber auch, daß zu bestimmten Zeitpunkten unterschiedliche Strategien eine Lösung darstellen können.

Der Fokus soll hier gerichtet werden auf Strategien, die zu bestimmten Zeitpunkten häufig verwendet werden, und sich über der Zeit den veränderten Umweltbedingungen anpassen. Dies ist besonders dann sinnvoll, wenn verschieden Agententypen miteinander konkurrieren. Da eine Vermischung des genetischen Programms unter den Typen nicht möglich ist, können sich die Agenten gegenseitig unter Evolutionsdruck setzen. Daraus könnte eine tatsächliche implizite Fitneß motiviert sein.

Es besteht in einem GPAL-System allerdings auch die Möglichkeit, daß sich für die einzelnen Agententypen Standardlösungen etablieren, d.h. faktisch keine qualitativen Änderungen im Programmcode der Agenten mehr auftreten. In diesem Fall würde keine Evolution mehr stattfinden, und es hätte sich ein Gleichgewicht etabliert, vorausgesetzt die Population bleibt auch weiterhin stabil. Ein solcher Fall ist in der PG tatsächlich eingetroffen (s. [1]).

2.1.7 Ablauf der Simulation

Zu Beginn eines Zeitschritts werden neue Foodobjekte erzeugt. Ein Foodobjekt wird mit einer bestimmten Wahrscheinlichkeit, die gegen jedes Feld getestet wird, generiert. Danach werden die Agenten sequentiell, vom ältesten zum jüngsten, abgearbeitet. Wenn die Energie eines Agenten auf den Minimalwert gefallen ist, z.B. durch die Attacke eines anderen, wird das Programm des Agent in diesem Zeitschritt nicht mehr ausgeführt. Ein das Programm eines Childs dagegen, das in diesem Zeitschritt erzeugt wurde, wird auch in diesem Zeitschritt noch ausgeführt. Am Ende des Zeitschritts wird das Alter der Agenten erhöht, und die toten Agenten in Carrion umgewandelt.

2.1.8 Einfache Weltmodelle

In der PG wurde an einem einfachen Weltmodell mit nur einem Agententyp und ohne Constraints statische Parametertests durchgeführt, die bereits einigen Aufschluß über das System erlauben.[5] Dies einfache Modell hat die Eigenschaft sehr stabil zu sein, wenn sich eine Population erst einmal etabliert hat. Der Verlauf der Populationskurve ist für jede Simulation charakteristisch. (s. Abb. 3.1) Das System schwingt sich auf einen chaotischen Attraktor ein, um einen Gleichgewichtspunkt, dessen Wert sich

[5] Das zugrunde liegende Worldfile ist in [1] ab S. 94 ff. beschreiben.

abschätzen läßt. Das Gleichgewicht wird bestimmt von der Welt zugeführten, und von den Agenten verbrauchten Energie. Die ungefähre Populationsgröße P ergibt sich durch die folgende Formel:

$$P = \frac{size_x size_y}{1 + \frac{CostOfLiving}{prob_f energy_f}}$$

Bei dieser Gleichung stellen $size_x$ und $size_y$ die Größe der Welt in beiden Richtungen dar. Die Parameter $prob_f$ und $energy_f$ stehen für die Wachstumswahrscheinlichkeit, und den Energiegehalt von Food.

Die Populationsgröße kann aber noch durch weitere Parameter beeinflußt werden (z.B. **MinMatingAge** oder **MatePause**).

Weitere Beobachtungen zeigten, daß Agenten in der Lage waren einfache Überlebensstrategien zu entwickeln. die auf eine Evolution schließen lassen. Erste Versuche mit Räuber-Beute-Systemen scheiterten aber an der Instabilität eines solchen Systems in Philia.

2.1.9 Verschiedene Agententypen

In Philia gibt es die Möglichkeit verschiedene Agententypen zu bilden, die durch unterschiedliche Attribute gekennzeichnet sind. Zusätzlich lassen sich Constraints definieren, mit deren Hilfe eine einzelne Funktionen in ihren Möglichkeiten eingeschränkt werden können. Diese Möglichkeit wird bei der Modellierung von Räuber-Beute-Systemen genutzt. In Philia lassen sich grundsätzlich vier verschiedene Arten von Constraints definieren:

1. **MATE-CONSTRAINTS** werden benutzt um Agenten eines bestimmten Typs explizit die Paarung mit Agenten anderen Typen zu gestatten, oder zu verbieten. So wird eine Paarung zwischen Räuber- und Beuteagenten verhindert.

2. **FRIEND-CONSTRAINTS** verbieten oder erlauben Friendbezogene Funktionen gegenüber bestimmten Agententypen. So kann z.B. ein Räuberagent nie ein Friend eines Beuteagenten werden.

3. **EAT-CONSTRAINTS** legen fest welche Art von Food oder Carrion von Agenten gefressen werden kann. So könnten Beuteagenten z.B. gezwungen werden nur Food als Nahrung zu akzeptieren. Dies hat allerdings keine Auswirkungen auf die Prädikate *FoodAhead?* und *FoodHere?*, die jede Form von Food oder Carrion als Food betrachten.

4. **ATTACK-CONSTRAINTS** legen fest welche Agenten attackiert werden können oder nicht. So können Beuteagenten z.B. gar keine Attacken durchführen.

Kapitel 3

Räuber-Beute Systeme in Philia

3.1 Das Weltmodell

Die Welt wurde rechteckig und nach allen Seiten abgeschlossen gewählt, und hat eine Ausdehnung von 89 Feldern in X-Richtung und 86 Feldern in Y-Richtung. Die Wahrscheinlichkeit das Food auf einem Feld wächst wurde mit 0.01 festgelegt. Food hat dann einen Energiewert von 30.

Desweiteren werden einige Attribute definiert, die von allen Agententypen geteilt werden. Alle Agenten sterben spätesten nach 1200 Zeitschritten. Neue Agenten können nach der Paarung in einem Radius von 2 Feldern um den Agenten abgelegt werden. Ein Agent kann maximal 1000 Energieeinheiten speichern. Der Agent muß zum Überleben mindestens 10 Energieeinheiten besitzen. Das Prädikat *Hungry?* ist erfüllt, wenn der Energiewert unter 100 Einheiten fällt. Mit jedem Zeitschritt verliert der Agent eine Einheit Energie.

Die im Worldfile angegebenen Statusparameter beziehen sich nur auf die Erschaffung der Welt zu Beginn der Simulation. Dabei erhalten die Agenten volle Energie, ihr Alter ist 0 Zeitschritte, und die Flags **Memory**, **Attacks** und **Screams** sind gelöscht. Die Position und die Orientierung der Agenten werden auf zufällige Werte gesetzt.

Zusätzlich wird die Mindesttiefe der Programme auf vier Ebenen, und die Maximaltiefe auf acht Ebenen festgelegt. Dies ist notwendig um die Rechnerlast in einem vertretbarem Rahmen zu halten. Die Punktmutation wurde abgeschaltet. Die Wahrscheinlichkeit für eine Baummutation wurde mit $5 * 10^{-5}$ angesetzt.

Zu den globalen Parametern gehört auch die Kostentabelle. Das komplette Worldfile ist ab S. 17 dargestellt. Die bisher nicht genannten Parameter im Worldfile, werden nicht genutzt, und haben keinen Einfluß, auf das bisher beschriebene Modell.

```
PARAMETERS

                Topology = 0                    # Rechteck
                MaxAge = 1200
                CheapMating = 0
                MateProb = 1
                VMutation = 0
                RMutation = 0
                CMutation = 0.00005
                Mutation = 0
                FastAttack = 1
#               AttackFactor = 1
#               DefenseFactor = 0.25
                WorldWidth = 89
                WorldHeight = 86
                FoodPropMap = 0.01
                FoodEnergy = 30

                Energy = 1000
                OffspringRadius = 2
                MaxEnergy = 1000
                MinEnergyFactor = 0.01
                HungerFactor = 0.1
                Age = 0
                Decay = 0
                MaxProgramDepth = 8
                MinProgramDepth = 4
                LastMate = 0
                Screams = 0
                Orientation = -1
                XPosition = -1
                YPosition = -1
                Memory = 0
                Attacks = 0

                MaxCost = 100
                Cost-Nop = 1
                Cost-Seq = 1
                Cost-If = 1
                Cost-And = 1
                Cost-Or = 1
                Cost-Not = 1
```

```
                Cost-Attack = 20
                Cost-Mate = 14
                Cost-Eat = 12
                Cost-MakeFriend = 10
                Cost-Scream = 10
                Cost-Move = 29
                Cost-TurnLeft = 7
                Cost-TurnRight = 7
                Cost-TurnSound = 15
                Cost-SetMem = 2

                Cost-Hungry? = 1
                Cost-UnderAttack? = 1
                Cost-Sound? = 3
                Cost-Mem? = 1
                Cost-AgentHere? = 5
                Cost-AgentAhead? = 5
                Cost-FoodHere? = 5
                Cost-FoodAhead? = 5
                Cost-FriendHere? = 5
                Cost-FriendAhead? = 5
                Cost-NothingHere? = 5
                Cost-NothingAhead? = 5
END_PARAMETERS

AGENTS

# Beuteagenten
AGENT A
                MATE-CONSTRAINTS        +A-B
                FRIEND-CONSTRAINTS      +A-B
                EAT-CONSTRAINTS         +P+B+A
                ATTACK-CONSTRAINTS      -*
                NUMBER = 4000
                ATTRIBUTES
                                MaxFriends = 20
                                NrOfChildren = 2
                                MinMatingAge = 100
                                MatePause = 5
                                CostOfLiving = 1
                                Strength = 50
                                RangeOfVision = 5
                                RangeOfHearing = 10
```

```
                END_ATTRIBUTES
END_AGENT

# Räuberagenten
AGENT B
                MATE-CONSTRAINTS            +B-A
                FRIEND-CONSTRAINTS          +B-A
                EAT-CONSTRAINTS             -P+A+B
                ATTACK-CONSTRAINTS          +A-B
                NUMBER = 1000
                ATTRIBUTES
                                MaxFriends = 6
                                NrOfChildren = 1
                                MinMatingAge = 120
                                MatePause = 8
                                CostOfLiving = 1
                                Strength = 80
                                RangeOfVision = 20
                                RangeOfHearing = 20
                END_ATTRIBUTES
END_AGENT

END_AGENTS
```

Die Kosten sind sehr niedrig angesetzt, was die Stabilität der Population fördern soll. Allerdings ist es auch hier schwer „funktionierende" Parameterwerte zu finden.

Die einzelnen Agententypen sind entsprechend ihrer Rolle als Räuber bzw. Beute modelliert. Sie sind bezüglich Paarung und Freundschaft Typeninvariant, um eine klare Unterscheidung zwischen beiden Typen zu gewährleisten. Beuteagenten dürfen alles fressen, aber keine Agenten angreifen. Räuberagenten sind beim Fressen dagegen auf Carrion beschränkt, können aber, um ihrerseits ihren Energiebedarf zu decken, Beuteagenten angreifen.

Die Agententypen werden weiterhin durch einige unterschiedliche Attribute gekennzeichnet. So erzeugen Beuteagenten, im Gegensatz zu Räubern, bei einer Paarung zwei Kinder. Dadurch ist eine erhöhte Beutepopulation möglich. Es hat sich gezeigt, daß eine erhöhte Beutepopulation für die Stabilität der Gesamtpopulation notwendig ist. In diesem Sinne sind die Möglichkeiten der Fortpflanzung bei den Räubern weiter eingeschränkt. So beträgt die Pause die ein Räuber zwischen zwei Paarungsversuchen machen muß 8 Zeitschritte, während die Beuteagenten schon nach 5 Zeitschritten wieder paarungsbereit sind. Auch das Mindestalter für die Paarung ist unterschiedlich. Während Räuber 120 Zeitschritte warten müssen, können Beuteagenten sich bereits

nach 100 Zeitschritten paaren. Beide Einschränkungen können aber auch zum Vorteil für die Agenten umgemünzt werden, denn da der Agent bei einer Paarung die Hälfte seiner Energie verliert ist sie eine sehr kostspielige Aktion für die Agenten. Eine große Wartezeit zwischen den Paarungen kann den Agenten also helfen ihre Energiereserven zu füllen. Ähnlich verhält es sich mit der Anzahl der Kinder. Da eine größere Anzahl auch bedeutet das jedes Kind nur einen entsprechenden Bruchteil der möglichen Energie zugeteilt bekommt. Aus diesem Grund ist es nicht unüblich, daß ein Kind direkt nach der Geburt wieder stirbt. Diese Beispiele zeigen die Problematik auf, plausible Werte für diese Attribute zu finden. Im Vergleich haben sich diese Werte allerdings bewährt.

Um das Verhältnis zwischen Räuber und Beute richtig darzustellen, müssen die Räuber in der Lage sein die Agenten bei einem Angriff auch zu töten, und damit ihre Energie zu absorbieren. Deshalb erhalten die Räuber einen größeren Strength Wert als die Beuteagenten.

Weitere Unterschiede sind bei der Sinnesstärke, und bei der Anzahl der Freunde zu finden. Sie wurden Vorbildern aus der Biologie angepaßt. Während die Beute sich in Gruppen von 20 Agenten zusammenfinden können (wobei zu berücksichtigen ist, das jeder Agent Mitglied in jeder Gruppe sein kann, sich also eher ein Netzwerk von Freunden als mehrere geschlossene Gruppen bilden), sind bei den Räubern nur 6 Freunde möglich. Sie sind also gegenüber den Beuteagenten eher Einzelgänger. Die maximale Sichtweite ist mit 20 für die Räuber viermal so hoch wie die der Beuteagenten. Der Hörradius ist mit ebenfalls 20 doppelt so hoch, und erstreckt sich bei günstiger Position über fast ein Viertel der Welt.

3.2 Stabilitätskriterien

Um eine Optimierung des Codes durch konkurrierende Spezies beobachten zu können, wurde ein Mindeststabilitätskriterium festgelegt, bei dem beide Populationen mindestens den 10001. Timestep erreichen mußten (schwaches Stabilitätskriterium). Als quasi-stabil gelten diejenigen Simulationen, in denen beide Populationen einen weitaus größeren Zeitrahmen, der zwar von Simulation zu Simulation teilweise unterschiedlich war, aber mindestens 50000 Timesteps betrug, überleben (starkes Kriterium). Es gab in allen Simulationsreihen nur wenige Simulationen, die diese Kriterien erfüllten. In einigen Simulationsreihen wurde nur das abgeschwächte erstere Kriterium erfüllt. Da die Simulationen sich zu diesem Zeitpunkt meist noch in der Einschwingphase befanden, war es problematisch aufgrund der hohen Streuungsbreite der verschiedenen Strategien, eindeutige Tendenzen auszumachen. Deshalb steht bei den Analysen nicht immer die Entwicklungsdynamik im Vordergrund, sondern konzentrieren sich auf Probleme die aus vorhergehenden Simulationen hervorgegangen waren.

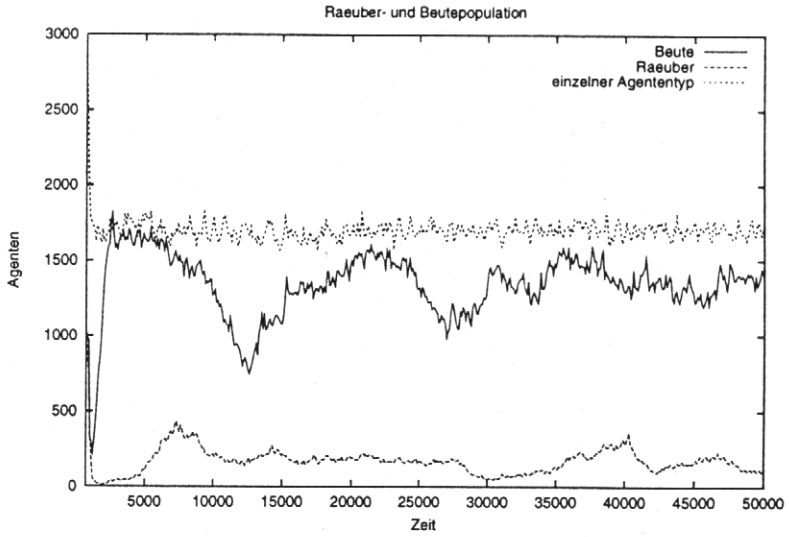

Abbildung 3.1: Räuber- und Beute-Populationen im Vergleich mit Population aus einer Simulation mit nur einem Agententyp.

3.3 Gemeinsamkeiten in der Populationsdynamik

Es wurden insgesamt 4 Simulationsreihen mit Räuber-Beute-Modellen durchgeführt. Dabei war die Entscheidung für eine Simulationsreihe nie planvoll vorgegeben, sondern stellt, durch Veränderung einzelner Parameter, immer einen neuen Blickwinkel aufgrund einer vorhergehenden Analyse dar.

Die Abb. 3.1 zeigt die Räuber und Beutepopulationen in einer quasi-stabilen Simulation. Zum Vergleich ist der Verlauf der Population mit einem Agententyp zu sehen. Die Art und Weise der Entwicklung ist in allen Simulationsläufen ähnlich. Dabei wurden die ersten 2000 Timesteps unberücksichtigt gelassen, da die Startpopulation für den weiteren Verlauf der Kurve unerheblich ist. Der in der PG bereits beobachtete Einschwingeffekt findet sich auch hier wieder. Die Beutepopulation erreicht ihren Tiefstand bei Timestep 1000. Die zu Beginn hohe Anzahl an Räubern läßt die Beutepopulation schnell auf ein globales Minimum sinken. Als Folge davon sinkt auch die Räuberpopulation. Hier findet der erste Selektionsprozeß statt. In Simulationen mit nur einem Agententyp, strebt das System nun einen Gleichgewichtspunkt an, der sich schätzen läßt Gleichgewichtspunkt (s. Abschnitt 2.1.8). In Räuber-Beute-Systemen beginnt sich nun das Verhältnis zwischen Räuber- und Beutepopulation einzuschwingen. Auffällig ist vor allem, daß die Standardabweichung der Räuber- und Beutepopulationen gegenüber den einfachen Agentenpopulationen deutlich größer ist. Sie betragen $\sigma = 48.65$ für das einfache Modell, und $\sigma_{Beute} = 169.4$ bzw. $\sigma_{Raeuber} = 76.44$ für die in Abb. 3.1 gezeigten Simulationen. Die Einführung unterschiedlicher Agententypen führt also zu einem qualitativ anderen Verhalten der Simulation. Der Durchschnitt der beiden Populationen bleibt (in etwa) stabil. Da-

mit sind die ersten beiden Volterraschen Gesetze erfüllt, wonach neben dem stabilen Durchschnitt (zweites Volterrasches Gesetz), auch die periodischen Schwankungen der Populationskurven für Räuber und Beute phasenverschoben sind (s. [46]). In der Abb. 3.1 fällt auf ein Maximum in der Räuberpopulation, fast genau ein Minimum der Beutepopulation und umgekehrt.

Kapitel 4

Simulationen und Analyse

4.1 Vorgehen

4.1.1 Ziel der Analyse

Das Ziel der Analyse ist, Anzeichen dafür zu finden, daß das Räuber-Beute-Szenario zu einer wahrnehmbaren Weiterentwicklung des Codes der Agenten führt. Dahinter steht als weitere Fragestellung welche Vorbedingungen erfüllt sein müssen, damit es zu einer permanenten Evolution der Strategien durch Konkurrenz kommt. Die Strategien stellen die lokalen Regeln der AL-Welt dar aus denen sich Emergentes Verhalten ergibt. Eine Änderung der Strategien, sollte sich auch eine Änderung im emergenten Verhalten des Systems mit sich bringen.

4.1.2 Begriffe

Für die Analyse sollen hier einige Begriffe zur Klärung angeführt werden:

Umgebung

Die Umgebung ist der Teil der Welt, den der Agent konkret von der Welt wahrnimmt. Dazu gehört Food auf dem Sehstrahl der durch *FoodAhead?* wahrgenommen wird, Geräusche im Hörbereich durch *Sound?* oder *TurnSound* wahrgenommen, ein Agent auf dem Aktionsfeld durch *AgentHere?*, *AgentAhead?* oder *Mate* wahrgenommen, etc.

Strategie

Das Verhalten des Agenten in jeder möglichen Umgebung.

Sequenz

Damit ist die Sequenz der Funktionen gemeint, die von den Agenten in einem bestimmten Timestep ausgeführt wird. Da später von den Sequenzen auf die Strategien zurückgeschlossen werden soll (siehe Abschnitt 4.1.3), ohne den konkreten Entscheidungsprozeß nachvollziehen zu müssen, da die Möglichkeit z.B. boolesche Ausdrücke zu bilden, unendlich komplex werden können, wird von allen parametrisierten und Funktionen und der Funktion *Nop* abgesehen.

Verhalten

Dieser Terminus bezieht sich auf das was der Verhaltensforscher sieht, also die Seiteneffekte der Sequenz, die sich durch die Zustandsänderungen in der Welt manifestiert, die der Agent vorgenommen hat. Diese werden durch die nicht-parametrisierten nicht-Prädikate ausgelöst.

Genpool

Der Genpool umfaßt alle Funktionen, die zu einem bestimmten Zeitpunkt in den Programmen der Agenten enthalten sind. Eine für die Analyse wichtige Größe ist die Anzahl der *einzelnen* Funktionen, die im Genpool noch vorhanden sind.

Konzept

Sicht-, Friend- und Soundkonzept, wie in 2.1.5 bereits beschrieben.

Kombination

Gemeint sind häufig wiederkehrende kurze Sequenzenabschnitte, die fast immer ein und dieselbe Teilstrategie charakterisieren. Diese Definition wurde im Nachhinein vorgenommen, und nimmt ein Ergebnis der Analyse vorweg. Die wichtigste Kombination ist die bei der Beute immer wiederkehrende *FoodAhead? Turn*[1] Sequenz, die in fast allen Fällen den Agenten sich nur dann wegdrehen läßt, wenn kein Food gesehen wurde.

Programmfluß

Der Programmfluß bezeichnet hier den Programmfluß, der sich ergeben würde, wenn alle Parametrisierten Ausdrücke vereinfacht würden. In diesem Sinne haben z.B. Konstanten keinen Einfluß auf den Programmfluß.

Lösung

Eine Sequenz, die gegen Ende der Simulation besonders erfolgreich ist. Dieser Begriff trägt der Tatsache Rechnung, daß gegen Ende der Simulation immer mehr Agenten immer wenige Sequenzen gebrauchen. Dies trifft vor allem für die Beute zu.

[1] Im folgenden steht *Turn* immer für *TurnRight* oder *TurnLeft*.

4.1.3 Vorgehen und Überblick

Im Unterschied zu GP mit expliziter Fitneßfunktion ergibt sich im Philia-Modell ein wesentliches Problem: Es gibt keine Fitneß, die erfolgreiche Agenten kennzeichnen würden. Der Terminus „erfolgreich" ist in diesem Modell für einzelne Agenten nur unscharf definiert. Das führt unmittelbar zu der Frage wonach denn nun eigentlich gesucht werden soll. Es wird daher nicht versucht eine „beste" Strategie unter vielen zu finden, sondern die Strategien die zur Stabilität der bestehenden Ökologie beitragen. Da die Anforderungen an das System hier vor allem Stabilität ist, ist dies nur Konsequent. Betrachtet werden also Strategien, die von möglichst vielen Agenten eingesetzt werden. Andere Strategien tragen zwar auch zur Systemstabilität bei, aber sie tun dies auf eine nur schwer nachvollziehbaren Weise.

Da es keine Vorgabe betreffend der Semantik der Programme (z.B. anhand einer Fitneßfunktion) gibt, muß die Semantik in der Analyse hergeleitet werden. Dies ist das Kernproblem der Analyse, und mit eines der komplexesten Probleme im ganzen GP Bereich, da das Programm in Struktur und Ablauf verstanden werden muß. Aus Gründen der Komplexität wurde von einer Betrachtung der Programme selbst abgesehen. Die Analyse stützt sich also auf das Verhalten einer großen Menge von Agenten zu einem bestimmten Zeitpunkt. Das Problem besteht darin, daß das Verhalten noch nichts über die zugrundeliegende Strategie aussagt. Nun zeigt sich aber, daß die Umstände, in denen bestimmte Verhaltensweisen entstehen, sehr ähnlich, in den meisten Fällen sogar identisch sind, so daß oft nur nur immer derselbe Programmfluß entsteht, was eine mögliche Analyse durch Vergleich verschiedener Sequenzen in den meisten Fällen fast unmöglich macht. Es ist also äußerst schwierig eine allgemeingültige Aussage über die Strategie zu treffen. Trotzdem ist dies der noch am ehesten gangbare Weg, weshalb sich die Analyse auf den Vergleich verschiedener Sequenzen, aus einzelnen Stichproben, stützen wird. Auf diese Weise sollen Hinweise auf die verwendete Strategie gewonnen werden.

Zuerst werden nach dem in Abschnitt 3.2 genannten starkem Stabilitätskriterium verschiedene stabile Simulationsläufe gesucht. Alle 2500 Timesteps werden Stichproben genommen, wobei die Aktionen der Agenten gedumpt werden. Im einzelnen werden die Sequenzen aller Agenten gesammelt und miteinander verglichen. Identische Sequenzen werden gezählt, sowie die Umgebungen in denen sie entstanden sind, und nach Räuber und Beute getrennt. Um besonders häufige Sequenzen herauszufiltern, werden die fünf erfolgreichsten Sequenzen jeder Stichprobe erfaßt, um dann diejenigen herausfiltern zu können, deren Vorkommen irgendwann während der Simulation eine bestimmte Anzahl, die für Räuber und Beute aufgrund der Unterschiedlichen Populationsgrößen unterschiedlich ist, überschritten haben. Der Wert für die Räuber lag dabei bei zehn, für die Beute bei fünfzig. Dazu wird der zeitliche Verlauf der Anzahl jeder dieser Sequenzen über dem Simulationszeitlauf festgehalten. Auf diese Art und Weise erhält man ein erstes Bild von der Entwicklung des Verhaltens der Agenten unter bestimmten Umständen.

Um nun überhaupt etwas über die zugrundeliegende Strategie aussagen zu können, hat es sich als nützlich erwiesen die Sequenzen miteinander zu vergleichen, und besonders häufige Kombinationen zu lokalisieren. Diese Kombinationen haben die Eigenschaft, auch bei verschiedenen Simulationen, ein Teil derselben Strategie zu sein. Dies war ein Ergebnis früher Analysen, wie sie in den ersten Simulationsläufen der ersten Simulationsreihe dargestellt werden. Gestützt wird diese Hypothese durch die starken Ähnlichkeiten der Sequenzen in allen Simulationen. Aus diesem Grund wurde diese Hypothese als Arbeitshypothese, und als Hilfe für weitere Analysen verwendet. In den ersten Simulationen mußte daher auch genau untersucht werden, welche Funktion die einzelnen Kombinationen haben.

Nachdem nun ein ungefähres Bild über die Funktionsweise der Programme erstellt wurde, stellte sich die eigentliche Frage nach der Evolution der Agentenprogramme. Hier greift eine zweite Arbeitshypothese, die man das „Prinzip Einfachheit" nennen könnte. Sequenzen unterscheiden sich häufig nur dadurch das z.B. zwei Funktionen in der Reihenfolge vertauscht wurden, oder irgendwo in der Sequenz plötzlich ein Prädikat auftaucht, die in einer vergleichbaren Sequenz nicht vorhanden ist. Diese Familienähnlichkeiten deuten häufig auf ein und dieselbe Strategie hin. Im ersten Fall ist es häufig völlig egal, ob der Agent zuerst versucht sich zu paaren, und dann versucht zu fressen oder umgekehrt. Im zweiten Fall passiert es regelmäßig, daß eine neue Informationsquelle eingeführt wird, deren Wert aber gar nicht benutzt wird. Es gibt viele Möglichkeiten in Philia wie Information verloren gehen kann. Einer der häufigsten dürfte wohl sein, daß ein Teilbaum in dem eine Information ohne Seiteneffekte erzeugt wird, als erstes oder zweites Argument der Funktion *Seq* vorkommt. Tatsächlich sind diese Vereinfachungen gefährlich, da natürlich sehr leicht wichtige Zusammenhänge, die nicht einfach intuitiv erfaßbar sind, übersehen werden können. Auffällige Unterschiede sollten allerdings erklärt werden können.

Nun muß die Entwicklung mit Ereignissen in der Welt in Zusammenhang gebracht werden, da davon ausgegangen wird, daß drastische Veränderungen häufig Reaktionen auf veränderte Umweltbedingungen darstellen. Dabei kommen besonders drastische Veränderungen der Populationsgröße für beide Agententypen in betracht, als auch die dominanten Verhaltensweisen des jeweils anderen Agententyps. Das Ergebnis ist eine Skizze von der Evolution der Agentenstrategien.

Es hat sich während der Analyse ergeben, daß es sinnvoll ist, den Genpool mit in die Betrachtungen einzubeziehen. Hier spielen zwei Fragestellungen eine Rolle:

1. Wieviele der Funktionen befinden sich noch im Genpool?
2. Welche Funktionen sind noch Genpool vorhanden?

Diese Fragestellungen traten durch die Feststellung auf, daß einige Funktionen nach einiger Zeit gar nicht mehr benutzt wurden. Im allgemeinen verschwindet eine solche Funktion nach einer Zeit auch aus dem Genpool. Mit der Beantwortung der zweiten

Name	Seed	letzter Timestep	Durchschnitt		Standardabweichung	
			Räuber	Beute	Räuber	Beute
n4	885302809	100009	199	1354	111	228
n8	891244383	70000	185	1288	118	259
n11	891611427	70000	185	1211	119	272
n12	891596169	70000	161	1270	120	273
n14	891875523	50000	119	1396	136	313
n15	891859123	50000	185	1260	144	331

Tabelle 4.1: Stabile Läufe in der ersten Simulationsreihe.

Frage kann auch festgestellt werden ob einzelne Funktionen in den Sequenzen als Konstanten auftreten. So kann *Sound?* nie erfüllt sein, wenn kein *Scream* mehr benutzt wird. Die Ergebnisse können so auch als zusätzliches Analysemittel genutzt werden. Alle 10000 Timesteps wird ein kompletter Dump durchgeführt, aus dem sich der Genpool herleiten läßt. Ob eine Funktion konkret benutzt wird oder nicht läßt sich aus den Sequenzen ersehen.

4.2 1. Simulationsreihe

In der ersten Simulationsreihe ergaben sich 6 stabile und 135 instabile Populationen. Sie sind in Tab. 4.1 zusammengefaßt.

Die Entwicklung der Population verläuft wie in Abschnitt 3.3 beschrieben[2].

Bei den Beuteagenten finden sich erfolgreiche Sequenzen nach den auf S. 25 genannten Kriterien z.T. schon bei der ersten Stichprobe (Timestep 2501) auf. Spätestens ab 22501 tauchen sie auch über einen längeren Zeitraum auf, und werden immer häufiger gezählt. Die Abb. 4.2 zeigt ein Beispiel. Nur wenige Sequenzen sind letztendlich so erfolgreich. In Abb. 4.1 wird dies besonders deutlich. Dargestellt sind die erfolgreichen Sequenzen, und ihre Summe. Die Kurve erreicht zwischen 30001 und 32501 ein lokales Maximum, daß zum Zeitpunkt 50001 knapp übertroffen wird. Während am ersten Maximum allerdings 5 Sequenzen maßgeblich zum Erfolg. beitragen, haben am zweiten Maximum alleine zwei Strategien zu 80% Anteil.

4.2.1 Genpool

In der ersten Simulationsreihe zeigte sich bereits ein interessanter Trend des Systems. War eine Funktion einmal, über eine kurze Zeitspanne von meist wenigen hundert

[2]Die Laufzeit wurde später auf 50000 Timesteps verkürzt, da immer weniger Veränderungen in den Sequenzen auftraten.

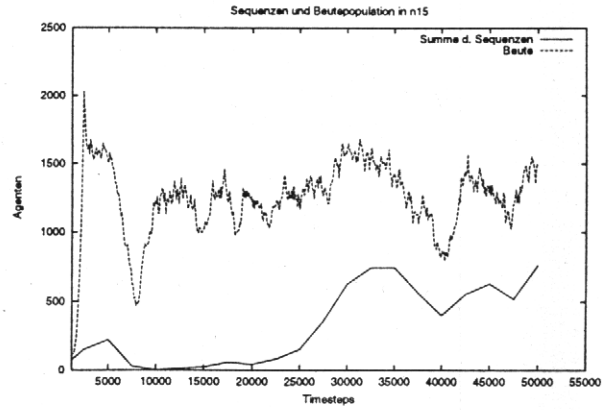

Abbildung 4.1: Erfolgreiche Sequenzen konvergieren gegen die Population.

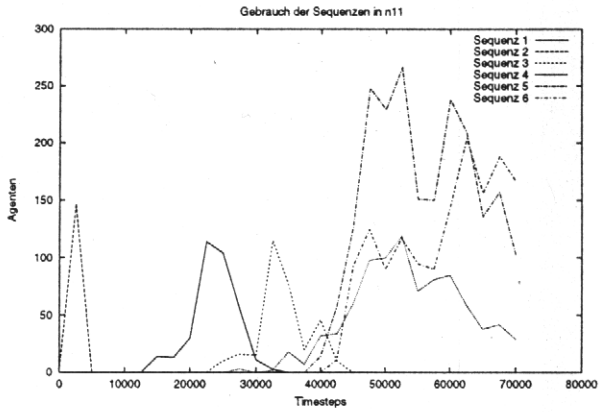

Abbildung 4.2: Später auftretende Sequenzen sind erfolgreicher.

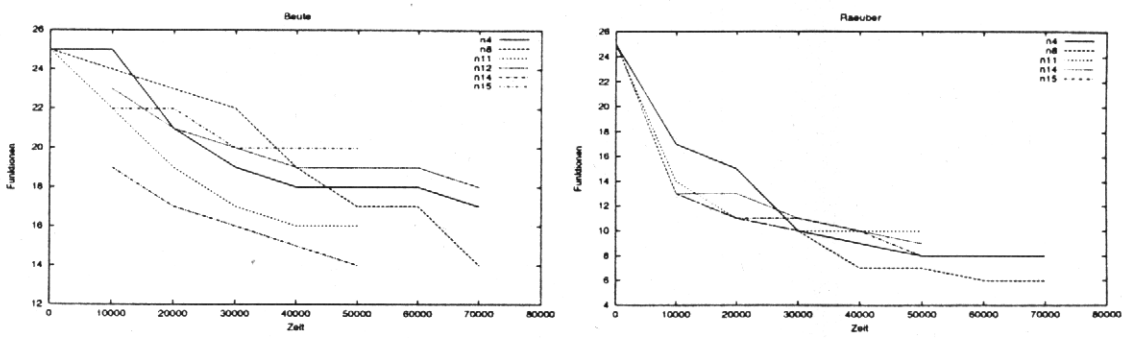

Abbildung 4.3: Die Anzahl der einzelnen Funktionen im Genpool fällt monoton über der Zeit.

Zeitschritten, unbenutzt, so wurde sie auch im folgenden nicht mehr benutzt. Tatsächlich schlug sich dies auch auf Programmebene nieder. Die Abb. 4.3 zeigt wie die Anzahl der einzelnen im Genpool enthaltenen Funktionen monoton fällt. Tatsächlich fiel der Gebrauch der meisten Terminale, die nicht unmittelbar zum Erhalt der Population notwendig waren, sehr schnell. Dieses Problem rührt von der Codierung der Funktionen aus dem GP-Teil des Systems her. Eine Funktion die nicht mehr im Genpool vorkommt kann nicht wiederhergestellt werden, es sei denn durch Mutation. Wären die Funktionen als Bitstrings (wie in der GA) codiert, so könnten jederzeit durch reines Crossing-Over alle Funktionen wieder hergeleitet werden. Die erste Simulationsreihe zeigt also deutlich, das die Wahrscheinlichkeit für eine Baummutation mit $5 * 10^{-5}$ zu klein gewählt war.

Zu den Funktionen die aus der Simulation herausfielen gehörten vor allem die Funktionen, die ein Konzept auf Basis der Friend-Funktionen realisieren sollten. Die Gründe für dieses Verhalten liegen in der Fähigkeit des Systems begründet Funktionen auszusondern, die für das Überleben der Populationen nicht benötigt werden. Heraus fallen überwiegend solche Funktionen, die relativ teuer sind, und damit Cost-Ressourcen verschwenden, weil sie lebensnotwendigeren Funktionen (allen voran *Eat* und *Mate*) weichen müssen. Dies gilt z.B. bei der Beute für die Funktionen *Attack* (Cost: 20), *MakeFriend* (Cost: 10), *Scream* (Cost: 10), *TurnSound* (Cost: 15).

Diese Eigenschaft schränkt den Entwicklungsspielraum des Systems ein. Die späteren Sequenzen ähneln sich daher mit zunehmender Laufzeit (s. Abschnitt 4.2.2), da immer weniger verschiedene Funktionen bei begrenzter Länge, in den Sequenzen vorkommen. Sie entsprechen immer den gleichen einfachen Schemata, und „konvergieren" von ihrem Verhalten her gegen eine Standardstrategie (mit leichten Variationen).

Einige Entwicklungen im Genpool können nachvollzogen werden:

Die Tatsache, daß die Räuber die Funktion *Eat* nicht mehr zur Aufnahme von Carrion nutzen, hat zwei Gründe. Zum einen ist die Energie die durch das Angreifen lebender Agenten gewonnen wird höher da die Agenten zum leben eine Energie größer zehn haben müssen, während Carrion einen Energiewert von neun hat. Zum anderen

entwickeln die Beuteagenten einen z.T. sehr effizienten Mechanismus zur Suche nach Food (s.u.). Die zentralen Funktionen *FoodAhead?*, *-Here?* und *Eat* aber machen keinen Unterschied zwischen Food und Carrion.

4.2.2 Sequenzanalyse: Beute

Da die erfolgreichen Sequenzen z.T. sehr ähnlich sind, werden hier vor allem diejenigen Sequenzen analysiert, die später als Lösungen auftreten. Um genaueres über die Strategien zu erfahren wurden weitere Stichproben in dem Zeitraum gemacht, in dem die Sequenz besonders stark vertreten war. Dabei wurden 10000 Timesteps lang alle 200 Timesteps eine Stichprobe genommen.

4.2.2.1 Simulation n4

Die erfolgreichste Sequenz in n4 ist:

FoodAhead? TurnRight FoodAhead? TurnRight Hungry? Move Eat TurnRight Eat

Sie tritt in dieser Form bereits ab Timestep 17501 auf. In Timestep 22501 wird sie bereits von mehr als 100 Agenten benutzt. In Timestep 75001 wird sie von fast 500 Agenten benutzt. Als Variation tritt eine Sequenz auf, die bis auf das fehlende *Hungry?* identisch ist. Diese wird in Timestep 75001 von über 250 Agenten genutzt. Zusammen benutzen also in etwa zwei Drittel aller Agenten zwei fast identische Sequenzen.

Die Umgebung in der diese Sequenz gewählt wird ist zu Beginn fast immer dieselbe. In 96.2% der beobachteten Fälle schlagen beide *FoodAhead?* fehl, *Hungry?* ist erfüllt, der Agent kann einen Schritt nach vorne machen und nimmt keine Energie auf. Fällt das *Hungry?* aus, bleiben die Rückgabewerte in 97.4% der anderen Funktionen gleich. Der Agent findet kein Food auf dem Sehstrahl und dreht sich weg. Das ganze wiederholt sich noch einmal. Beim dritten mal versucht der Agent nicht erst festzustellen ob Food vorhanden ist, sondern unternimmt einen blinden Fressversuch. Dabei wird berücksichtigt das das Food nicht unbedingt direkt vor dem Agenten liegen muß. Dem trägt der Agent Rechnung in dem er versucht einen Schritt zu gehen, woraus ihm auch dann keine Nachteile entstehen, wenn ihm der Weg versperrt ist. Da auch ein gescheitertes *Eat* keine Nachteile mit sich bringt, ist der Versuch einer Nahrungsaufnahme auf gut Glück sinnvoll. Dieser wird in einer anderen Richtung wiederholt, was die Möglichkeit der Energieaufnahme steigert.

Es stellt sich die Frage inwiefern beide verwandt sind, und ob sie die gleiche, und vor allem welche Strategie sie repräsentieren. Um die zweite Frage zu klären, soll zuerst die Rolle der *FoodAhead?* Prädikate zu Beginn der Sequenzen geklärt werden. Um dies zu klären wurden Agenten gesucht bei denen das *FoodAhead?* zu einem anderen Zeitpunkt erfüllt war:

Funktionen (1)	Anzahl	ohne Turn
AgentAhead?	104	104
FoodAhead?	1157	1157
MakeFriend	13977	13977
Mate	1550	1550
Eat	556	556

Funktionen (2)	Anzahl	ohne Turn
Eat	150	150
FoodAhead?	6	6
FriendHere?	1651	1651
Hungry?	14381	14381
MakeFriend	1507	1489
Mate	107	107

Tabelle 4.2: Funktionen nach dem ersten und dem zweiten *FoodAhead?* zu Beginn der Sequenz. Die zweite Spalte zeigt die Fälle, in denen in der restlichen Sequenz bis zum nächsten *Turn* ein *Eat* ausgeführt wurde.

Tabelle 4.2 zeigt welche Funktionen ausgeführt wurden, nachdem die entsprechende *FoodAhead?*-Funktion erfüllt war. Dabei finden sich auch die Fälle, bei denen im folgenden ein *Eat* ausgeführt wird, ohne das sich der Agent zwischenzeitlich wegdreht. Da die Agenten nur eine Lebenserwartung von maximal 1200 Timesteps haben, mußte sich die Analyse in diesem Punkt auf einen engeren Zeitraum beschränken. Gezählt wurde exemplarisch im Bereich 50002 – 60002.

An beiden Stellen führt ein erfülltes *FoodAhead?* dazu, daß der Agent sich nicht vom gefundenen Food wegdreht. Fast immer versucht der Agent dieses Food auch zu fressen. Dies wird durch Funktionen erreicht, die im momentanen Zusammenhang keinen Sinn ergeben (z.B. Funktionen die sich auf Agenten beziehen). In jedem Fall führt der Agent aber ein *Eat* aus, egal ob er Food fand oder nicht. da sich der Agent bei nicht gefundenem Food aber vor dem Fressen noch wegdreht, ist dieses Verhalten sinnvoll, da er immerhin ein Feld vor sich hat über das er noch keine Informationen besitzt. Nebenbei erhöht der Agent durch diese Taktik seine Überlebenschancen in anderer Hinsicht weiter: Dadurch, daß er sich wegdreht, wenn kein Food gefunden wird, dreht er sich auch von möglichen Räubern weg.

Abb. 4.4 zeigt die Entwicklung dieses Verhaltens über den Zeitraum der Simulation, und ist ein klares Zeichen für den Erfolg dieser Strategie. Gezählt wurden die Agenten, die obige Initialsequenz benutzt, und nachher *Eat* ausführten. Dies zeigt auch deutlich, daß sich diese Teilstrategie als Gesamtlösung des Systems durchsetzt.

Bei den gleichen Agenten stellt sich auch die Frage, unter welchen Umständen die arterhaltende Funktion *Mate* ausgeführt wird. Ein Teil der Agenten tut dies direkt nach erfüllten *FoodAhead?*, wie aus obigen Tabellen hervorgeht. Diese Versuche führen aber niemals zum Erfolg, da ja in diesem Fall kein anderer Agent vor dem Agenten steht.

Im nächsten Test soll also die Frage im Vordergrund stehen in welchem Maße die einzelnen Funktionen das Auftauchen von *Mate* beeinflussen. In der Tabelle wurden Agenten gezählt, die unter der angegebenen Bedingungen die Funktion *Mate*

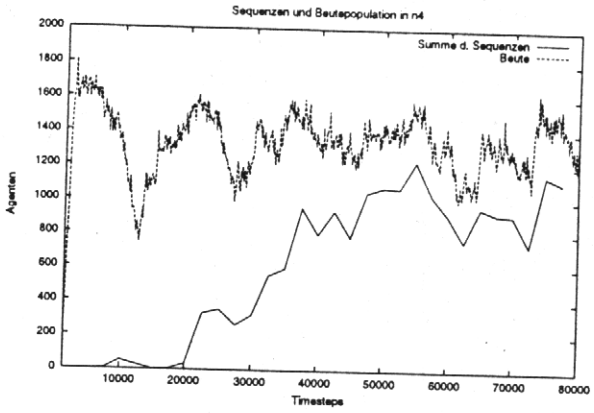

Abbildung 4.4: Die Strategie in n4 konvergiert gegen die Gesamtpopulation.

ausführen (unabhängig ob das *Mate* zum Erfolg führen kann oder nicht).

Diese Werte lassen vermuten, daß das bestehen der Population in den meisten Fällen durch *Hungry?* gesteuert wird, im ggs. zur zweiten Strategie, die dies durch das erste *FoodAhead?* zu steuern scheint. Dies macht durchaus Sinn, da die *Mate* Funktion im Erfolgsfalle dem Agent die Hälfte seiner Energie raubt. Auf diese Art und Weise ist es erheblich wahrscheinlicher, daß der Agent noch mindestens 50 Energieeinheiten hat, damit seine Kinder mit jeweils 12 gerade noch lebensfähig sind. Man kann Agenten, die die erstere Sequenz benutzen, also als fitter betrachten.

4.2.2.2 Simulation n8

Auch hier gibt es zum Schluß zwei Lösungen die fast identisch sind, die sich aber beide im Vergleich zu n4 erst spät entwickeln. Die ältere der beiden Sequenzen lautet:

Eat TurnLeft Hungry? FoodAhead? TurnLeft Move FoodAhead? TurnLeft Eat

Sie tritt erst ab Timestep 42501 auf, während die zweite, die bis auf das fehlende *Hungry?* identisch ist, sogar erst ab Timestep 50001 auftaucht. Die Ähnlichkeit zur Lösung in Lauf n4 ist nicht zu übersehen. Die Umgebung ist in 98.7% der Fälle ähnlich. Das *Eat* schlägt fehl, die Energie des Agenten ist unter 10%, es liegt nach der Linksdrehung kein Food auf dem Sehstrahl, nach einer weiteren Linksdrehung kann der Agent einen Weg nach vorn machen. Auch dann ist kein Food zu sehen, und das *Eat* (nach erneuter Linksdrehung) schlägt fehl. Praktisch hat der Agent, von seinem Ausgangspunkt aus gesehen, einen Rückschritt gemacht, und eine Rechtsdrehung vollführt. 97.7% der Agenten, die die zweite Sequenz benutzen finden sich in genau der gleiche Lage wieder, mit dem Unterschied, daß eine Aussage über den Energiestand der Agenten nicht vorhanden ist. Wieder stellt sich die Frage, welchen Einfluß die Rückgabewerte der Funktionen auf das Verhalten der Agenten haben. Der Vergleich mit gefundenen Agenten in einer anderen Umgebung in der *Eat* erfüllt war ergab, daß die darauf folgende Funktion immer *TurnLeft* war, was ein Hinweis darauf

Funktion (1)	Stelle	Wert	Gesamt	Mate
FoodAhead?	1	1	17344	7200
FoodAhead?	3	1	17473	4885
Hungry?	5	0	222067	114915
Move	6	0	65943	0
Eat	7	1	2853	0
Eat	9	1	2492	0

Funktion (2)	Stelle	Wert	Gesamt	Mate
FoodAhead?	1	1	1876	651
FoodAhead?	3	1	2888	42
Move	5	0	12363	0
Eat	6	1	429	0
Eat	8	1	0	0

Tabelle 4.3: Funktionen mit ihren Positionen in der ersten und zweiten Sequenz, der Gesamtzahl, der Agenten in denen an der angegebenen Stelle die Funktion den angegebenen Wert lieferte, und die Anzahl der Agenten, die unter diesen Bedingungen im folgenden ein *Mate* ausführten.

ist, daß *Eat* den Programmfluß nicht weiter beeinflußt, es sei denn der Programmfluß wird in einem anderen Teilbaum fortgesetzt, in dem die nächste Funktion ebenfalls *TurnLeft* ist. Dies gilt nicht für *Hungry?*. Ändert sich hier der Wert, so wird in 56.3% der Fälle *Mate* ausgeführt. Offenbar richtet auch diese Strategie ihren Paarungstrieb nach der noch vorhandene Energie im Agenten. Auch die *FoodAhead?*s funktionieren wie in n4. Ein weiterer *Turn* wird in fast allen Fällen bis zum nächsten *Eat* verhindert. Interessant ist das Verhalten falls das *Move* mißlingt: Der Agent versucht in allen beobachteten Fällen ein *Eat*, was sinnvoll sein kann, denn es könnte sich bei dem Objekt ja um Food handeln. Insgesamt läßt sich sagen, daß die Strategien fast identisch mit denen aus n4 sind, sieht man einmal davon ab, daß die Reihenfolge eine andere ist. Hier zeigen sich auch schon zwei wichtige Kombinationen. *FoodAhead? Turn* läßt den Agenten wegdrehen, falls kein Food gesehen wurde. Ansonsten wird in den meisten Fällen bis zum nächsten *Eat* eine Drehung verhindert. *Eat Turn* funktioniert ähnlich, allerdings scheint hier kein Entscheidungsprozeß stattzufinden. Wird also kein Food gefunden, so Orientiert der Agent sich einfach in eine andere Richtung.

4.2.2.3 Simulation n11

Auch hier kommen als Lösung zwei Sequenzen in betracht, die sich sehr ähnlich sind. Die gegen Ende erfolgreichere Sequenz ist:

FoodAhead? TurnLeft AgentAhead? TurnLeft Eat Move FoodAhead? TurnLeft

Sie taucht erst ab Timestep 40001 auf. Die zweite unterscheidet sich nur dadurch, daß das zweite *FoodAhead?* unmittelbar wiederholt wird. Sie tritt ab Timestep 30001 auf. Da bei zwei unmittelbar hintereinanderfolgende *FoodAhead?*s keine funktionalen Unterschiede zum einfachen *FoodAhead?* bestehen, kann man die erste Sequenz als kompaktere Version der zweiten ansehen. In 84.1% der Fälle findet der Agent kein Food auf dem Sehstrahl, dafür aber einen Agenten. Das *Eat* schlägt fehl, der Schritt nach vorn gelingt, und danach ist immer noch kein Food zu sehen. In der zweiten

Variante sieht es in 82.2% der Fälle genauso aus. Auffälligerweise scheint auch hier wieder die gleiche Strategie zugrunde zu liegen. Allerdings ist hier die Kombination *FoodAhead? TurnLeft* an das Ende des ausführbaren Teils kopiert. Hier stellt sich die Frage nun ganz besonders ob der Agent tatsächlich noch in der Lage ist ein *Eat* auszuführen, falls das Prädikat erfüllt ist. Tatsächlich wird nach einem erfolgreichem *FoodAhead?* in fast 100% der Fälle die Funktion *Eat* direkt ausgeführt. Die *FoodAhead? TurnLeft* Kombination zu Beginn der Sequenz funktioniert ebenfalls genau wie in den anderen Simulationsläufen zuvor.

Auffällig ist auch der Gebrauch von *AgentAhead?*. Hier ist zu klären ob es sich tatsächlich um eine Interaktionssuche mit einem anderen Agenten handelt, oder die Information anderweitig verwertet wird.

In allen Fällen wird nach dem mißlungenem *AgentAhead?* ein *Move* ausgeführt. In 92.5% der Fälle folgt darauf ein *Eat*, ohne das sich der Agent vorher wegdreht. Eine Interaktion mit Agenten ist sehr unwahrscheinlich, da er ja reagiert wenn kein Agent vorhanden ist. Diese Variante scheint also eine weitere Spielart der *FoodAhead? Turn* Kombination mit umgekehrten Vorzeichen zu sein. Ist kein Agent zugegen, so ist die Chance gegeben, daß sich vielleicht Food dort befindet.

Bei *Eat* findet sich eine Neuheit. In den meisten Fällen dreht sich der Agent nach gelungenem *Eat* weg. Er hat also gelernt, daß wenn ein Foodstück gefressen ist, sich auf dem Feld kein Food mehr befinden kann.

Der Rückgabewert von *Move* hat keinen Einfluß auf den Programmablauf.

4.2.2.4 Simulation n12

Die erfolgreichste Lösung im Lauf n12 ist die Sequenz:

FoodAhead? TurnLeft AgentAhead? TurnLeft Move Eat TurnLeft Eat

In 83.5% der Fälle kann kein Food gefunden werden, dafür steht aber ein Agent im Sehstrahl, der Schritt nach vorn gelingt, und beide *Eat* schlagen fehl.

Diese Sequenz tritt quasi gleichzeitig mit einer zweiten Sequenz, die zum Ende noch über hundert mal vorkommt auf:

FoodAhead? TurnLeft AgentAhead? Move Eat TurnLeft Eat AgentAhead?

Sie zeigt eine Besonderheit durch das fehlende *Turn* nach dem *AgentAhead?*. Dadurch stellt sich hier wieder die Frage nach der Bedeutung des *AgentAhead?*. Bei den Umgebungen sind in dieser Variante zwei Fälle dominant. Zu 53.4% sieht es so aus, daß sowohl *FoodAhead?* als auch das erste *AgentAhead?* fehlschlagen. Das *Move* kann ausgeführt werden, und beide *Eat* schlagen fehl. In 31% der Fälle sieht es aber so aus, daß auch das letzte *AgentAhead?* nicht erfüllt ist.

Semantisch zeigen sich keine Neuerungen. *Move* und *Eat* tragen nicht zum Programmfluß bei. In der zweiten Sequenz zeigt sich, daß ein erfülltes *AgentAhead?*

mit einer hohen Wahrscheinlichkeit von 96.3% zu einem *TurnLeft* führt. Ist dagegen *AgentAhead?* nicht erfüllt zeigt sich das obige Verhalten. Es scheint als seien die oben genannten 31% auf eine andere Strategie zurückzuführen, deren Sequenz nur zufällig dieselbe ist, und in der *AgentAhead?* keinen Einfluß auf den Programmablauf hat.

4.2.2.5 Simulation n14

In n14 gibt es eine eindeutige Lösung die sich deutlich von den Lösungen in den anderen Läufen abhebt. Bereits ab Timestep 5001 treten diese und weitere Lösungen, die bis Timestep 50001 erfolgreich sind, auf:

TurnRight Mem? Eat TurnRight Eat TurnRight Move Eat

In 84.6% der Fälle ist *Mem?* nicht erfüllt, alle *Eat* schlagen fehl, und *Move* ist erfolgreich.

Das Auffälligste Merkmal ist, das in dieser Sequenz keinerlei sensorische Fähigkeiten eingesetzt wird. Es bleibt also zu prüfen ob diese Funktion von anderen Terminalen, also in diesem Fall *Eat*, übernommen wird.

In dieser Sequenz hat nur das erste *Eat* Einfluß auf den Programmablauf. Es verhindert kurioserweise das wegdrehen des Agenten falls es erfüllt ist. Stattdessen wird mit dem nächsten *Eat* fortgefahren (das dann als Konstante auftritt, und doch noch ein wegdrehen erzwingt). Zu *Mem?* konnte kein Fall gefunden werden in dem es erfüllt war.

4.2.2.6 Simulation n15

Ähnlich wie in n14 verläuft die Entwicklung auch in n15. Die erfolgreichste Sequenz war:

Eat TurnRight Move Eat TurnRight Eat TurnRight

Diese Sequenz ist fast identisch zu der aus n14. Auch hier schlagen (in 93% der Fälle) alle *Eat* fehl, und *Move* ist erfolgreich. Sie repräsentiert im Grunde auch die gleiche Strategie. *Eat* hat keinen Einfluß auf den Programmablauf. Auch in 78.9% der Fälle ändert ein mißlungenes *Move* nichts am Programmablauf. In 14.9% der Fälle führt es allerdings zu einer Linksdrehung.

4.2.2.7 Schlußfolgerungen

Die Sequenzanalyse hat einige Gemeinsamkeiten in den Simulationsläufen aufgezeigt.

- Gegen Ende der Simulation finden sich immer eine oder zwei erfolgreiche Sequenzen, die dieselbe Strategie repräsentieren. Sieht man sich die Entwicklung

ähnlicher Sequenzen über den gesamten Zeitablauf an, so stellt man fest das die Zahl der Agenten die diese Strategie nutzen gegen die Gesamtpopulation konvergiert. Insofern kann diese Strategie als Lösung bezeichnet werden.

- Es gibt zwei verschiedene Arten von Lösungen. Die erste macht effektiv Gebrauch von Prädikaten, um Food aufzuspüren. Dabei treten zwei Varianten auf:

  1. Ist *FoodAhead?* nicht erfüllt, so drehe dich weg, ist es erfüllt, so führe *Eat* aus.
  2. Steht ein Agent auf dem Sehstrahl, so weiche dem Agenten aus (Drehung), ansonsten Versuch zu fressen.

  In zwei Fällen konnten die Agenten auch den Gebrauch von *Hungry?* erlernen, um ihren Energiehaushalt bezüglich *Mate* zu regeln.

- Die zweite Art benutzt keinerlei sensorische Fähigkeiten. Die Rückgabewerte haben nur wenig Einfluß auf den Programmablauf. Zentrales Moment scheint hier das *Eat* im Zusammenhang mit einer *Turn*-Funktion zu sein. Das *Eat* hat auch keinen Einfluß darauf ob *Turn* ausgeführt wird oder nicht. Falls Food gefunden wurde ist das also genauso uninteressant für den Agenten, wie wenn keines gefunden wurde.

- *Mate* tritt z.T. nur sporadisch auf. Außer bei n4 scheint es in keinem Lauf ein System zu geben, welches die Fortpflanzung steuert. Es scheint als sei es nicht nötig sich an anderen Agenten zu orientieren um ein fortbestehen der Population zu gewährleisten. Da *Mate* sehr kostenintensiv ist, muß der Gebrauch von *Eat* mit dem Gebrauch von *Mate* über die Durchschnittsenergie ausbalanciert werden. Deshalb kann *Mate* in den meisten Fällen kein fester Bestandteil einer erfolgreichen Sequenz sein.

- Alle Agenten sind Mobil. *Move* ist bestandteil aller Sequenzen. Die Mobilität scheint für die Beuteagenten ein wichtiger Überlebensfaktor zu sein.

- Es werden im Normalfall keine Friend-Funktionen benutzt. Sie haben in einer normalen Umgebung keine Funktion.

4.2.3 Sequenzanalyse: Räuber

Weil das Erfolgskriterium für die Räuber mit 10 Agenten sehr niedrig angesetzt war, ergab die Analyse eine größere Auswahl. Wie sich herausstellte war dies aber kein größeres Problem, da die Räuber mehrheitlich nur bis zu drei Terminale, die nicht *Nop* waren, benutzten, was praktisch keinen Spielraum für Innovationen ließ. Allerdings hieß das häufig auch, daß keine Analyse der Strategien möglich war, da keine Fälle gefunden wurden, in der die Umgebung zu einem anderen Zeitpunkt abwich.

4.2.3.1 Simulation n4

Als erfolgreich kommen drei Sequenzen in betracht.

1. *Attack TurnRight Mate Attack*

   Diese Sequenz tritt bereits ab Timestep 12501 auf, überschreitet ab 22501 die Grenze von 10 Agenten und steigt in Timestep 40001 auf 75 Agenten. In Timestep 50001 passiert sie ein lokales Minimum (weniger als fünf Agenten) und steigt dann wieder. Die jeweiligen Umgebungen lassen alle Funktionen, außer die Konstante *TurnRight*, zu 98% scheitern.

2. *TurnRight Mate Attack Attack*

   Diese Sequenz ist zum Schluß zur ersten Sequenz fast gleichauf. Sie wird bereits ab Timestep 10001 gebraucht. In 99.1% der Fälle scheitern alle Funktionen.

3. *TurnRight Attack Attack Attack*

   Diese Sequenz wird ab Timestep 15001 benutzt, und ist die am wenigsten erfolgreiche von den dreien. In allen Fällen scheitern alle Funktionen.

Auffällig ist der einfache Aufbau der Sequenzen. Es werden nur die allernötigsten Funktionen gebraucht. Ab Timestep 10001 wird die Funktion *Move* nicht mehr benutzt. Im Gegenteil zur Beute sind die Räuber also nicht mobil. Statt dessen werden häufig Kombinationen ausgeführt, die keinen Sinn ergeben. Dazu gehört z.B. die mehrfache Ausführung von *Attack* hintereinander.

Um den Einfluß der Rückgabewerte auf den Programmablauf zu testen wurden wieder Fälle mit alternativen Umgebungen gesucht. In der ersten Sequenz haben andere Rückgabewerte keinen Einfluß auf das weitere geschehen. In der zweiten Sequenz konnten aufgrund des geringen Vorkommens insgesamt keine alternativen Sequenzen festgestellt werden. Das gleiche gilt auch für die dritte Sequenz. Trotz allem drängt sich die Vermutung auf, daß der Ablauf völlig Sequentiell festgelegt ist. Die Agenten tuen also nichts anderes als sich drehen, versuchen sich zu paaren, und zu attackieren. D.h. wenn vor dem Agenten ein Beuteagent steht wird er gefressen, ansonsten paart sich der Agent falls ein Räuber vor ihm steht. Interessanterweise sind die Agenten, die Sequenz drei benutzen nicht selbständig in der Lage sich fortzupflanzen. Sie sind also darauf angewiesen, das andere Agenten die Funktion *Mate* auf sie anwenden.

4.2.3.2 Simulation n8

Hier ragen zwei Sequenzen besonders hervor.

1. *Attack TurnRight Mate Attack TurnRight Mate*

   Diese Sequenz taucht ab Timestep 45001 spontan einige male auf, wird aber schlagartig ab Timestep 62501 mit 39 Agenten zur erfolgreichsten Sequenz. In 97.6% der Fälle scheitern alle Funktionen.

2. *Attack TurnRight Mate Attack Mate*

   Diese Funktion bildet sich schon ab Timestep 20001 und erreicht gegen Ende 14 Agenten. Die Entwicklung verläuft auch hier nicht kontinuierlich. Hier scheitern die Funktionen in allen Stichproben.

Auch hier zeigt sich die Einfachheit der Lösung. Es konnten keine Fälle beobachtet werden in denen eine Funktion erfüllt war, weshalb der Einfluß der einzelnen Rückgabewerte nicht getestet werden konnte.

4.2.3.3 Simulation n11

Hier ragen zwei Sequenzen besonders hervor.

1. *Attack TurnRight Attack Mate Attack*

   Hier fiel die Sequenz zwar gegen Ende weit unter 10 Agenten, aber eine Stichprobe vorher war sie mit 30 Agenten eine der Erfolgreichsten überhaupt.

2. *Attack TurnRight Attack Mate Attack TurnRight*

   Diese Sequenz löst die erste ab, und steigt auf 23 Agenten an.

In beiden Sequenzen scheitern alle Funktionen in allen Stichproben. Es konnten keine Fälle beobachtet werden in denen eine Funktion erfüllt war.

4.2.3.4 Simulation n12

In n12 gibt es zum Schluß keine Sequenz, die die Grenze von 10 Agenten überschreitet. Die erfolgreichsten wurden von 4 bzw. 2 Agenten verwandt.

1. *Attack*

2. *Attack Attack TurnRight Mate Attack*

Beide entstehen bereits früh in Timestep 5001 bzw. 25001.

In der ersten Sequenz schlägt die Funktion immer fehl. In der zweiten Sequenz ist dies in 96.7% der Fall. Es konnten keine Fälle beobachtet werden in denen eine Funktion zu einem anderen Zeitpunkt erfüllt war.

4.2.3.5 Simulation n14

Mit 16 Agenten ist hier zum Schluß folgende Sequenz am erfolgreichsten:

TurnLeft Mate Mate Mate Mate Mem? Attack

Sie tritt erst ab Timestep 45001 auf. In allen Fällen scheitern alle Funktionen. Es konnten keine Fälle beobachtet werden in denen eine Funktion zu einem anderen Zeitpunkt erfüllt war.

4.2.3.6 Simulation n15

Hier gab es gegen Ende nur eine erfolgreiche Sequenz:

Attack Attack TurnRight Mate Attack.

Sie wird bereits ab Timestep 5001 benutzt, und gewinnt deutlich an Popularität ab Timestep 37501. Auch hier schlagen alle Funktionen fehl. Es konnten keine Fälle beobachtet werden in denen eine Funktion zu einem anderen Zeitpunkt erfüllt war.

4.2.3.7 Schlußfolgerungen

In allen Simulationen bietet sich ein einseitiges Bild. Die Räuber benutzen sehr einfache und kurze Sequenzen. Ihre Strategie beschränkt sich auf drehen, attackieren und paaren. Dabei treten kaum Unterschiede zwischen den Lösungen verschiedener Simulationen auf. Einwirkungen der Rückgabewerte auf den Programmfluß, können aufgrund der geringen Population nicht nachgewiesen werden.

4.2.4 Entwicklung

Wie auf S. 26 bereits erwähnt, wird ab hier davon ausgegangen, daß beobachtbare Änderungen in der Umwelt eine Anpassung im Code der Agenten verlangen. Wiederum wirkt diese Anpassung zurück auf die so veränderten Umweltbedingungen. Auch haben die Codes der beiden unterschiedlichen Agentenklassen aufeinander Einfluß. Hier wurden aber wiederum nur die auf S. 25 genannten Kriterien für erfolgreiche Agenten berücksichtigt, da die Situation sonst unüberschaubar Komplex geworden wäre. Um eine Entwicklung nachvollziehen zu können, wurden Sequenzen verglichen, die zu einem Zeitpunkt während der Simulation mit über fünfzig Agenten vertreten waren. Bei den Räubern galt die Grenze von zehn Agenten.

4.2.4.1 Simulation n4

Hier wurden 11 Sequenzen auf Seiten der Beute, und 27 auf Seiten der Räuber, als erfolgreich bewertet.

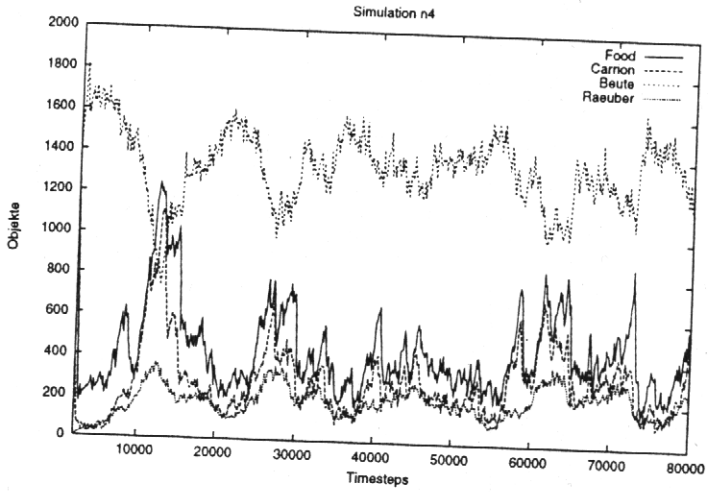

Abbildung 4.5: Simulation n4. Auffällig eine Änderung im System ab Timestep 12000.

Auffällig ist ein Wandel zu Beginn der Simulation. Zwischen Timestep 5001–8001 lernen die Agenten die *FoodAhead?-TurnRight*-Kombination zu nutzen. Ab 12501 setzt sich die Kombination *Eat TurnRight* durch. Ab Timestep 20001 setzt sich das zweite *FoodAhead? TurnRight* durch, allerdings am Ende des Programmcodes. Bereits ab 12501 wird die Teilsequenz *Hungry? Move* in die Sequenzen eingefügt. Dies Zusammen ergibt die zweiterfolgreichste Sequenz. Sie hat ihren Höhepunkt in 22501–40001. Zeitgleich entsteht die insgesamt erfolgreichste Sequenz, die später in Timestep 75001 von doppelt so vielen Agenten benutzt wird. In dieser Version ist die zweite *FoodAhead? TurnRight* Kombination in den vorderen Programmcode gelangt. Die dritterfolgreichste Sequenz entsteht ebenfalls zu diesem Zeitpunkt, schafft aber erst ab 35001 erfolgreich zu werden, und bricht ab 57501 wieder bis fast auf Null, ab 65001 wieder dramatisch zu steigen. Dieser Sequenz fehlt das regulierende *Hungry?*. Der Erfolg dieser drei Sequenzen beginnt nach einer Veränderung im System (s. Abb. 4.5). Um Timestep 12000 gab es weitaus mehr Food als Beute. Dieser Zustand ändert sich im folgenden allerdings drastisch. Kurz bevor die Populationskurve das lokale Maximum erreicht, beginnen sich diese drei Strategien zu etablieren. In eben dieser Zeitspanne fällt jedoch die Food-Kurve drastisch ab. Die Agenten haben also ihre Fähigkeit Food aufzuspüren verbessern müssen. Während nun die Anzahl des Foods wieder steigt, steigt auch die Anzahl der Agenten, die eine dieser Strategien verfolgen. Folglich bricht die Food-Kurve ein weiteres mal zusammen. Gleichzeitig verschwinden in Timestep 15001 diejenigen Räuber, die noch Prädikate benutzen. In Timestep 10001 benutzten die Räuber noch hauptsächlich Funktionen, die im späteren Verlauf verschwinden. Die erfolgreichste Sequenz ist hier:

FriendHere? TurnRight TurnSound Mate Attack FriendHere? Hungry? TurnSound Mem?

Hier fällt die Präsenz der Friend-Funktionen *FriendHere?* und *TurnSound.* Da diese Sequenz am erfolgreichsten an einer steigenden Flanke der Räuberpopulation zu sehen ist, könnte sie zu diesem Entwicklungsschub beigetragen haben. Vor allem die Teilsequenz *TurnSound Mate* unterstützt diese These, wenn ausreichend *Scream* benutzt wird. Dies ist jedoch nicht der Fall. *Scream* wird seit Timestep 7501 nicht mehr gebraucht. *FriendHere?* dagegen ist niemals eine Konstante, da die Eltern eines Agenten automatisch Friends sind, unabhängig von *MakeFriend.* 2500 Timesteps später zeigen die Räubersequenzen ihre spätere Form, die nur noch in Details variiert wird. Dabei werden effektiv zwei *Attack* versucht, zwischendurch eine Rechtsdrehung vollführt und ein *Mate.* Dagegen steht zu diesem Zeitpunkt die Strategie der Beuteagenten einmal nach Food Ausschau zu halten, darauf zuzulaufen, und *Eat* auszuführen. Da dieses Verhalten sich schon ab Timestep 10001 abzeichnet, scheint neben der dadurch verursachten Verknappung an Food-Ressourcen, auch die zunehmende Agressivität der Räuber Grund für den Zusammenbruch der Beutepopulation in Timestep 12501 zu sein. Warum es nun genau zum Rückgang der Räuberpopulation kommt ist schwer feststellbar. Sicher ist jedoch, daß die hohe aufkommen an Food dazu beigetragen hat, da es für die Räuber ein unüberwindbares Hindernis ist. Ab hier ändert sich die Räuberstrategie effektiv nicht.

Nun scheint eine ruhigere Phase zu kommen. In dieser Phase entwickelt sich ein anderer Typ von Beuteagent. Er scheint eine Mischung aus dem ersten und den anderen beiden zu sein, da er sowohl im vorderen Teil zweimal als auch einmal im hinteren Teil der Sequenz die Kombination *FoodAhead? Turn* benutzt. Ab ca. Timestep 55000 bricht die Beutepopulation ein. Dies kann mehrere Ursachen haben: Zum einen ist durch den geringen Bestand an Food eine so hohe Population nicht mehr zu halten. Mit dem anhaltenden Erfolg der Strategie wurde das Food langsam immer weniger. Zum anderen steigt auch die Chance, daß die umherwandernden Beuteagenten auf Räuber treffen. Dies ist ein Zeichen für die Komplexität der Zusammenhänge zwischen den drei Größen. Zwar treten diese Kombinationen häufiger auf, wirken sich aber selten drastisch aus. Dieses Synergiephänomen allerdings hat folgen. Durch den Einbruch der Beutepopulation steigt das Food unverhältnismäßig schnell an. Diese Entwicklung wird allerdings sofort wieder gestoppt, denn die Anzahl der Agenten, die die neue aggressivere Art benutzt steigt, was zu einem dramatischen Fall der Foodvorräte führt. In der nachfolgenden Zeit bricht die Beutepopulation wieder ein, und die Population der Agenten, die die neue Strategie benutzen fällt wieder zusammen. Hier hat offenbar ein äußerer Umstand eine Entwicklung zu einer effizienteren Strategie verhindert. Dies zeigt auch, das Effizienz nicht unbedingt (bezogen auf die Gesamtpopulation), auch Fit heißen muß. Die Frage bleibt offen ob diese spontane Reaktion die Simulation wieder ins Gleichgewicht gebracht hat, oder ob sie von selbst wieder ins Gleichgewicht gefunden hätte. Allerdings scheint sich der Bestand dieser Agenten nach einem erneuten Zusammenbruch der Foodpopulation ab 73000 wieder zu erholen.

4.2.4.2 Simulation n8

In n8 wurden11 Sequenzen auf Seiten der Beute und 17 Sequenzen auf Seite der Räuber als erfolgreich bewertet..

Alle erfolgreichen Sequenzen haben eine ähnliche Struktur. Es ergibt sich das folgende Bild:

Bereits in Timestep 2501 ist die Grundstruktur ähnlich wie sie später in 70001 sein wird.

Alle Sequenzen beginnen mit *Eat TurnLeft FoodAhead?* bzw. *FoodHere?*. In Timestep 5001, 7501 und 27501 taucht zwischen *Eat* und *TurnLeft* das Prädikat *Sound?* auf. Die Benutzung der Funktion *Scream* fällt in den ersten 10000 Timesteps stark ab (1% in 5001 bis etwa 0.3% in 7501). Spätestens in Timestep 27501, wo die Scream bei etwa 0.1% liegt, ist eine Interaktion mittels des *Sound*-Konzepts eher fragwürdig. Bis Timestep 35000 schwankt der Foodvorrat stark. Ab da sinkt die Amplitude, und der Durchschnitt fällt ab. In diesem ersten Teil entwickelten sich Agenten, die die Kombination *FoodHere? TurnLeft* benutzten, also anstatt auf den Sehstrahl sich auf das Aktionsfeld konzentrierten. Die Foodkurve stabilisiert sich erst während sich langsam Strategien durchsetzen, die eine zusätzliche *FoodAhead? TurnLeft Eat* Kombination am Ende der Sequenz einsetzen. Dazu ist anzumerken das alle Sequenzen *Move* einsetzen. Ab Timestep 50000 werden Agenten populär, in denen *FoodHere?* durch *FoodAhead?* ersetzt ist. Dies führt unmittelbar zu der in Abschnitt 4.2.2.2 gezeigten Sequenz.

Die Räuber sehen strukturell genauso aus wie in n4, und verhalten sich auch so. Die erfolgreichen Räubersequenzen enthalten genau die Funktionen *Mate*, *TurnRight* und zweimal *Attack*. Diese Entwicklung setzt bereits bei 15001 ein und ändert sich nicht mehr. Davor scheint die Streuungsbreite so hoch zu sein, daß keine erfolgreichen Sequenzen gefunden werden konnten. Ein Zusammenhang mit der Beutepopulation, die über das im vorhergehenden Abschnitt gesagte hinausgeht, ist nicht erkennbar.

4.2.4.3 Simulation n11

In n11 kommen 6 Sequenzen auf Beuteseite, und 9 Sequenzen auf Räuberseite in betracht.

Die Streuungsbreite ist über den gesamten Einschwingprozeß so hoch, daß sich bei der Beute erst ab Timestep 22501 (abgesehen von einer sehr frühen Sequenz um 2501), und auf Räuberseite erst ab Timestep 12501, danach erst wieder ab 37501 auftaucht.

Auffällig in diesem Lauf ist vor allem ein Einbruch der Beutepopulation um 38000. Dies führt zu übermäßig viel Food, was durch das System bereits ab 40000 wieder ausgeglichen wird. Das besondere ist, das ab da die Gruppe der Agenten, die die oben beschriebene Lösung benutzen immer größer wird, obwohl die Population in

etwa gleichbleibt. Der Ausgleich des Systems scheint diese Sequenzen gefördert zu haben.

Bei den Räubern fällt ziemlich exakt in diesen Zeitraum von Timestep 37501 bis Timestep 52501 eine besonders starke Benutzung von *Mem?* auf. Allerdings wird *SetMem* ab Timestep 12501 nicht mehr benutzt, so daß dieses Prädikat als Konstante auftritt.

4.2.4.4 Simulation n12

In n12 kommen 9 Sequenzen auf Seiten der Beute, und 24 Sequenzen auf Seiten der Räuber in betracht.

In diesem Lauf ist ein Zusammenhang besonders deutlich. Immer wenn eine Sequenz erfolgreich wird, sinkt das Food auf ein lokales Minimum, daraufhin sinkt die Benutzung dieser Sequenz wieder. Das setzt sich über die ganze Simulation fort. Dabei ändert sich an den Sequenzen nicht viel. Sie benutzen ab 5001 schon eine *FoodAhead? TurnLeft AgentAhead? TurnLeft* Kombination, was sich nur noch ändert nach einem lokalen Minimum der Foodkurve. Die Folge ist, das eine weitere *FoodAhead? TurnLeft* Kombination in den Code aufgenommen wird. Dies führt letztendlich beim Food zum globalen Minimum, was wiederum zur Folge hat, das diese zusätzliche Kombination durch das noch effizientere *Eat TurnLeft Eat* ersetzt wird.

Auch in diesem Lauf entwickeln sich die Sequenzen der Räuber nach keinem erkennbarem System, das über die Einschwingphase hinausgeht. Ab Timestep 10001 sind die nicht zum Kern gehörenden Funktionen entfernt.

4.2.4.5 Simulation n14

In n14 erfüllen sowohl auf Seiten der Beute, und auf Seiten der Räuber 5 Sequenzen die Kriterien.

In dieser Simulation entsteht der Eindruck als fände gar keine Entwicklung statt. Alle erfolgreichen Sequenzen, haben einen gleichen Aufbau. Es gibt drei *TurnRight*, und drei *Eat* jeweils zwischen den *TurnRights* verteilt. Dazwischen steht ein *Move*. Außer der Reihenfolge der Funktionen ändert sich nichts. Es gibt keinen erkennbaren Zusammenhang zwischen den Sequenzen und der Population. In dieser Simulation gibt es nur zwei auffällig starke Schwankungen in der Population, die aber nicht zu einer qualitativen Veränderung des Codes führt.

Bei den Räubern finden sich erstmalig eine Besonderheit, da in Timestep 37501 *Scream*, und in Timestep 50001 *Mem?* benutzt wird. Letzteres hat allerdings wahrscheinlich keinen Einfluß auf den Programmfluß, da es am Ende der Sequenz steht. Zwar wird insgesamt von den Räubern *Scream* noch massiv eingesetzt (über 5%), aber ab Timestep 10001 werden sowohl *Sound?* als auch *TurnSound* nicht mehr benutzt. Eine Interaktion mittels des Sound-Konzepts ist also ausgeschlossen.

4.2.4.6 Simulation n15

In n15 kommen 9 Sequenzen auf Seiten der Beute, und 21 Sequenzen auf Seiten der Räuber in betracht.

In n15 sind die Populationen wieder wesentlich instabiler. Hier findet sich eine Besonderheit gegenüber n14. Ab 27501 etablieren sich zwei Sequenzen, die zusätzlich vor ihr *Move* noch eine *FoodAhead? TurnRight* Kombination gesetzt haben. Dies führt wahrscheinlich zu einer höheren Effizienz, da das Food im folgenden Verlauf um ca. Timestep 31000 ein globales Minimum erreicht. In dem folgendem Zusammenbruch der Population verschwindet diese Variante wieder.

Die Räuber entwickeln sich wie in den anderen Simulationen auch, ohne Besonderheiten. Auch hier erscheint in Timestep 17501, und Timestep 35001 bis 37501, wieder die Funktion *Mem?* am Ende der Sequenzen.

4.2.5 Emergenz

Die Situation in der Welt verändert sich während der Simulation kaum. Die Räuber liegen am Rande kleinerer Food-Felder, während die Beute über das Feld verstreut ist. In Zeiten wo die Räuberpopulation geringer ist, ist an den Rändern dieser Felder vermehrt Carrion zu sehen. Die Räuber warten also auf die Beute am Rande der Energiefelder, während die Beute nach Food Ausschau hält, und sich dabei über das Feld bewegt. Als zusätzliche Energiequelle stehen der Beute noch das Carrion zur Verfügung. Dieses emergente Verhalten läßt sich leicht erklären: Die Räuber stehen, um sich fortpflanzen zu können beisammen. Die Beuteagenten, die in der Nähe ein Foodobjekt sehen, werden angelockt, und unter Umständen gefressen. Dadurch befinden sich im Umfeld um die Räuber weniger Beuteagenten, und es entsteht Platz für Food, das nicht sofort gefressen wird. Wird das Food zuviel, so werden auf der einen Seite mehr Beuteagenten zum Food kommen, auf der anderen Seite die Räuber behindert.

4.2.6 Zusammenfassung

Insgesamt scheint es genügend Anhaltspunkte zu geben um die These zu stützen, wonach starke Abweichungen des Gleichgewichts im System zu wirklichen qualitativen Änderungen im Code führen können. Dieses Verhalten findet sich auch in der natürlichen Evolution wieder. Es zeigt aber auch das nicht jede „subjektiv" bessere Strategie auch wirklich die fitteste Strategie ist, in dem Sinne, das sie sich längerfristig durchsetzt. Dies ist auf das Kriterium der Stabilität zurückzuführen, das die Simulationen zu erfüllen hatten. Die Lösungen können durchaus unterschiedlich sein, aber es finden sich immer die gleichen Elemente. Ein großer Teil der Funktionalität

Name	Timestep		Durchschnitt		Standardabweichung	
	erster	letzter	Räuber	Beute	Räuber	Beute
p4	10001	60002	161	707	39	102
p8	4001	54002	59	934	64	133
p11	6001	56002	148	793	49	112
p12	6001	56002	133	697	56	138
p14	10001	60002	1	1004	6	49
p15	10001	60002	155	708	24	66

Tabelle 4.4: Simulationsläufe aus der ersten Simulationsreihe mit erhöhtem Evolutionsdruck.

wird nicht ausgenutzt. Dies könnte sich unter höherem Evolutionsdruck ändern. Eine zu geringe Mutationswahrscheinlichkeit führt zur Verarmung des Genpools. Eine Koevolution ist schwer nachweisbar. Die Räuber verändern ihre Strategie meist ab dem Timestep 15001 effektiv nicht mehr, sorgen aber weiterhin für eine, im Vergleich zu einfachen Modellen, Destabilisierung der Beutepopulation. Da sich die Beutestrategien zum Teil stark geänderten Umweltbedingungen anpassen, ist ein schwacher indirekter Zusammenhang möglich. Diese Frage wird in Abschnitt 5.2 diskutiert.

4.3 2. Simulationsreihe

Die 2. Simulationsreihe sollte klarstellen, ob eine Erhöhung des Evolutionsdrucks zu qualitativen Verbesserungen führen, d.h. ob kooperative Elemente zum Einsatz kommen würden oder sich eine wirkliche Koevolution einstellen würde. Dazu wurde die Futterwachstumswahrscheinlichkeit um die Hälfte reduziert. Um einen einfachen Vergleich zu ermöglichen, und um nicht neue stabile Simulationsläufe umständlich suchen zu müssen, wurde auf die bereits vorhandenen Läufe zurückgegriffen. Ansonsten war es schwierig, bei der geringen **FoodEnergy**, Simulationsläufe zu finden, die über der Einschwingphase hinaus stabil bleiben. Es wurden die vollständigen Dumps der Läufe zu einem bestimmten Zeitpunkt als Worldfiles benutzt. Dabei zeigte sich, daß in 2 Simulationen die Räuberpopulation nicht überlebte. In Tab. 4.4 sind die Simulationen dargestellt.

Bei der Analyse stellte sich heraus, daß die Lösungen mit denen aus der ersten Simulationsreihe vergleichbar war. Deshalb wurde die Bedeutung der Kombinationen aus der Sequenzanalyse der ersten Reihe entliehen, und hier nur einzelne Besonderheiten, die während der Entwicklung auftraten, näher betrachtet.

4.3.1 Entwicklung

Bei der Angabe der Timesteps wurde berücksichtigt, daß zum Beginn der Simulation eine bestimmte Zeitspanne bereits vergangen war.[3] Die hier betrachteten Sequenzen konnten also frühestens nach dem in Tab. 4.4 angegebenen Start entstehen.

4.3.1.1 Simulation p4

Hier gab es 7 erfolgreiche Beutesequenzen, und 22 erfolgreiche Räubersequenzen.

Im Simulationslauf p4 fallen zwei Dinge sofort auf. Zum einen gibt es unter den Erfolgreichen eine Sequenz, die schon von Anfang an existiert, ab 12502 die Toleranzgrenze überspringt, und zum Schluß die Erfolgreichste ist. Zum zweiten läßt sich kaum ein Zusammenhang zwischen den Populationen und den verwendeten Sequenzen erkennen. Die am häufigsten eingesetzte Sequenz ist

FoodAhead? TurnRight Move Hungry? Move Eat TurnRight

In einer ebenfalls erfolgreichen Variante ist das *Hungry?* entfernt. Interessanter ist eine dritte Variante:

FoodAhead? TurnRight Move Hungry? Eat TurnRight FriendHere? Mate

Dies sieht zunächst sehr vernünftig aus, besonders die Kombination *FriendHere? Mate* am Schluß, legt möglicherweise eine echte Interaktion durch Friend-Funktionen nahe. Allerdings ist in allen beobachteten Fällen dieses Prädikat nicht erfüllt. Das *Hungry?* ist zu 95% nicht erfüllt. Auch das ergibt keinen erkennbaren Sinn. Eine zwischen Timestep 32502 und Timestep 42502 auftauchende Sequenz läßt sogar den Eindruck aufkommen die Agenten seien dümmer geworden. In der folgenden Sequenz ist in 99.8% der Fälle das zweite *FoodAhead?* nicht erfüllt.

FoodAhead? TurnRight FoodAhead? Move Eat TurnRight Eat

Das heißt, das es sich bei diesem Prädikat und den folgenden zwei Funktionen, in den allermeisten Fällen um überflüssigen Code handelt. Insgesamt läßt sich also sagen, daß die Fähigkeiten der Beuteagenten trotz erhöhtem Evolutionsdruck anscheinend nicht verbessert wurden. In der vorliegenden Simulation läßt sich dies möglicherweise dadurch erklären, daß durch die geringere Beutepopulation die Anzahl des Food häufig höher war als die Anzahl der Agenten. Dadurch traten periodisch immer wieder kurze Zeiträume auf, in denen es sich nicht lohnte mehr nach Food Ausschau zu halten.

Die Lösungen bei den Räubern in p4 unterscheiden sich ebenfalls nicht von denen der ersten Simulationsreihe. Allerdings gibt es im Zeitraum 10002 – 37002 Räuber, die eine ganze Reihe untypischer Funktionen benutzen. Diese Funktionen sind im einzelnen *FriendHere?*, *FoodHere?*, *TurnSound*, *Hungry?*, *Mem?* und *NothingAhead?*.

[3] Der Simulator erlaubt in dieser Version nicht die Zeit kontinuierlich mitzuführen.

Dabei hat *TurnSound* keine Funktion, da *Scream* von keinem Agenten mehr benutzt wird. Die Analyse stellt aufgrund der niedrigen Datenmenge (wg. der kleinen Population) ein Problem dar, aber alle Rückgabewerte dieser Funktionen sind so gut verteilt, das es sich hier wahrscheinlich eher um verschiedene Strategien handelt, die in verschiedenem Umfeld gleiche Sequenzen hervorgebracht haben. Dies könnte zwar als Hinweis auf eine Interaktion gewertet werden, aber ein Zusammenhang ist weder zu anderen Sequenzen, noch in den Populationen erkennbar.

4.3.1.2 Simulation p8

Hier gab es 10 erfolgreiche Beutesequenzen, und 3 erfolgreiche Räubersequenzen.

In p8 gibt es um Timestep 18500 eine Auffälligkeit Sowohl Räuber als auch Food erreichen ihr globales Maximum, während die Beute ihr globales Minimum erreicht. Danach verläuft die Simulation eher ruhig und ausgeglichen, während sie nach dem Aussterben der Räuber ihren Gleichgewichtspunkt findet. Mit einem Blick auf die erfolgreichen Sequenzen, sieht man das nach diesem Vorfall sich sehr schnell Sequenzen etablieren die bis zu drei *FoodAhead? Turn* Kombinationen enthalten. Dabei kommt es zunächst zum Erfolg der Sequenz

FoodAhead? TurnLeft Move Move FoodAhead? TurnLeft

Ab 25702 fällt diese Sequenz wieder, und es werden Sequenzen stark in denen drei (in einem Fall vier) *FoodAhead? TurnLeft* Kombinationen auftauchen, also sehr gründlich nach Food Ausschau halten. Nach dem Aussterben der Räuber in Timestep 38202 sterben diese Agenten wiederum aus, und eine Sequenz etabliert sich, die mit der obigen identisch ist, abgesehen davon, daß die zwei *Move* an das Ende der Sequenz verschoben sind. Es ist möglich, daß dieser Wechsel zum Aussterben der Räuber beigetragen hat. In diesem Fall ist der Zusammenhang, der in allen Simulationen durch die frappierende Ähnlichkeit beider Kurven in Erscheinung getreten ist, zwischen Räubern und Food interessant. Die Räuber bringen drei erfolgreiche Sequenzen hervor, die genau zwischen Timestep 21502 und Timestep 34002 liegen. Semantisch gibt es allerdings keine Besonderheiten.

4.3.1.3 Simulation p11

Hier gab es 9 erfolgreiche Beutesequenzen, und 21 erfolgreiche Räubersequenzen.

In p11 gibt es besonders zwei Stellen in denen sich sehr erfolgreiche Sequenzen häufen. Dies ist zwischen Timestep 24502 und Timestep 32002 sowie zwischen Timestep 37002 und Timestep 49502 der Fall. Daran sind vier Sequenzen beteiligt von denen drei fast identisch sind. Sie beginnen alle mit der folgenden Teilsequenz:

FoodAhead? AgentHere? TurnLeft FoodAhead? TurnLeft FoodAhead? FoodAhead?

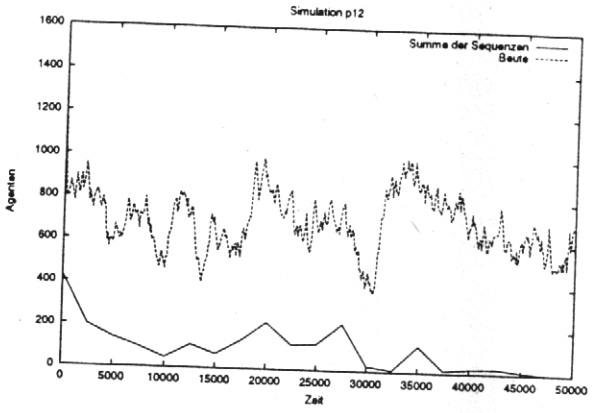

Abbildung 4.6: Die Summe der Sequenzen nimmt in p12 ab.

Abgesehen davon, daß man nicht genau sagen kann welche Rolle das *AgentHere?* spielt, da es in zuwenig Fällen erfüllt war, ist diese Sequenz nicht stärker auf die Foodsuche optimiert als die Sequenzen in der ersten Simulationsreihe. Im Gegenteil, sie scheinen sogar weniger fit zu sein, da die Agenten der ersten Reihe nach einem mißlungenen Versuch, gezielt zu suchen, noch einmal nach einer Drehung versuchten, zu fressen. Dies tun diese Agenten allerdings nur in Ausnahmefällen. Stattdessen wird, nach obiger Eingangssequenz, viel häufiger *Mate* ausgeführt. Die erfolgreichste Sequenz in p11 ist allerdings eine Sequenz die *FoodAhead?* überhaupt nicht einsetzt. Stattdessen kommt hier wiederum verstärkt *Eat* zum Einsatz. Diese Sequenzen werden besonders dann populär, wenn die Häufigkeit von Food relativ hoch sind, aber die Beutepopulation relativ niedrig. Dieses Verhältnis kehrt sich mit dem Erfolg dieser Sequenzen um. Hier ist vermutlich ein Zusammenhang gegeben. Die Räuber entwickeln sich wie in den Simulationen zuvor. Auffällig ist auch hier der Variantenreichtum, der möglicherweise daher rührt, das sich die Räuber in ihren einfachen Strategien keine Strukturen innerhalb des Codes zu etablieren brauchen.

4.3.1.4 Simulation p12

Hier gab es 10 erfolgreiche Beutesequenzen, und 23 erfolgreiche Räubersequenzen.

In p12 gibt es eine ununterbrochene Entwicklung der Sequenzen hin zu Agenten mit vier *FoodAhead? TurnLeft*-Kombinationen. Diese treten ab 23502 auf. Dies ist auch der Punkt in dem die Beutepopulation stark steigt, so daß davon ausgegangen werden kann, daß das Potential bis dahin bereits entwickelt war, sich aber erst jetzt durchsetzt? Die Lösungen, die hier entstehen, sind mit Lösungen aus Simulationen mit nur einem Agententyp vergleichbar (s. Abschnitt 4.6).

Entgegen allen bisher beobachteten Regeln, nimmt die Summe der erfolgreichen Sequenzen nicht zu, sondern ab (s. Abb. 4.6). Deshalb wurde die Grenze für erfolgreiche Sequenzen auf 20 reduziert. Diese Sequenzen unterschieden sich aber nicht

* Warum?

Grundlegend von den vorher beobachteten.

Die Räuber entwickelten sich ohne Besonderheiten.

4.3.1.5 Simulation p14

Hier gab es 13 erfolgreiche Beutesequenzen, und keine erfolgreichen Räubersequenzen.

In p14 sterben die Räuber direkt nach 2201 Timesteps aus. Dementsprechend entwickelt sich ein Gleichgewicht zwischen Beute und Food. Es kommt zu ähnlich einfachen Lösungen, wie bei den Räubern in anderen Simulationen. Die einzigen regelmäßig gebrauchten Funktionen sind *Move*, *Eat*, *TurnRight* und *Mate*. Einfache Variationen dieser Agenten werden über lange Zeiträume immer erfolgreicher. Es kommt zu keinen Innovationen.

4.3.1.6 Simulation p15

Hier gibt es 10 erfolgreiche Beutesequenzen, und 32 erfolgreiche Räubersequenzen.

In p15 gibt es nach einem schwankenden Vorkommen an Food einen relativ ruhigen Teil ab Timestep 35002. In dieser Zeit etablieren sich Agenten, die zweimal *FoodAhead? TurnLeft*-Kombination einsetzen. Diese scheinen gegen Ende der Simulation, nach einem langsamen abgleiten der Foodkurve wieder zu verschwinden. Von den Agenten wurde auch zum ersten mal durchgängig *NothingAhead?* und *NothingHere?* verwendet. Eine Klärung war aufgrund der Datenlage allerdings nicht möglich. Die Räuber entwickeln sich wie gewohnt.

4.3.2 Zusammenfassung

Die Erhöhung des Evolutionsdrucks brachte wenig Fortschritte. Wenn eine Population ausstarb, so waren es immer die Räuber, aufgrund ihrer geringen Population in allen Simulationen. Die größten Veränderungen fanden im Code der Beuteagenten statt. Dabei wurden z.T. effektivere Suchstrategien gewählt. In vielen Fällen wurden allerdings auch schlechtere Suchstrategien verwendet. Dies kann vor allem zwei Gründe haben:

1. In einigen Fällen gab es periodisch mehr Food als Agenten. Es ist möglich, daß durch den permanenten Wechsel keine neuen Strategien etabliert werden konnten.

2. Es gibt ein eine Strategie, die mit dieser Analysemethode nicht erfaßt wird.

Name	Seed	letzter Timestep
m1	894814653	14101
m2	894886469	16201
m3	894915799	10901
m4	894970939	21001

Tabelle 4.5: Simulationsläufe in der dritten Simulationsreihe, die über den Timestep 10000 stabil waren.

Die Besonderheit in Simulation p12 ist eine Ausnahme. Sie wurde nur in dieser Simulation beobachtet. Es scheint als hätte es in diesem Fall nach dem Aussterben der Räuber möglich gewesen, die Simulation fortzuführen, ohne sich auf eine bestimmte Sequenz zu spezialisieren. Da sich die Sequenzen semantisch aber nicht Grundlegend ändern, scheint dahinter immer noch dieselbe Strategie zu stehen.

Die Situation in der Welt blieb unverändert. Eine Koevolution ist auch hier nicht deutlich nachzuweisen.

4.4 3. Simulationsreihe

Als Maßnahme gegen das Austrocknen des Genpools, was große Veränderungen im Verhalten der Agenten verhindert, sieht das Philia-Konzept die Mutation vor. Durch Mutationen können Funktionen neu in den Genpool hineingepumpt werden. Diese Simulationsreihe sollte klären, welche Entwicklungen möglich sind, wenn der Genpool durch Mutationen fortlaufend angereichert wird.

Um dies zu erreichen, wurden beide Mutationsparameter geändert. Zum einen wurde die Punktmutation zugelassen. Zum anderen wurde die Wahrscheinlichkeit für die Baummutation um eine Potenz auf $5 * 10^{-4}$ erhöht.

Nur 4 von 83 Simulationsläufen erfüllten das schwache Stabilitätskriterium.

Wie die Tabelle 4.5 zeigt, waren diese vier Läufe nicht annähernd so stabil wie in den vorhergehenden Simulationsreihen.

4.4.1 Genpool

Die Frage ist nun, ob die Erhöhung der Mutationswahrscheinlichkeit zu einer Bereicherung des Genpools geführt hat. Die Abb. 4.7 zeigt die Zahl der einzelnen Funktionen im Genpool über der Zeit.

Verglichen mit Abb. 4.3 fällt vor allem auf, daß die Zahl der Funktionen im Genpool im Timestep 10000 bereits wesentlich geringer war. Zwar ist vereinzelt eine Abwärts-

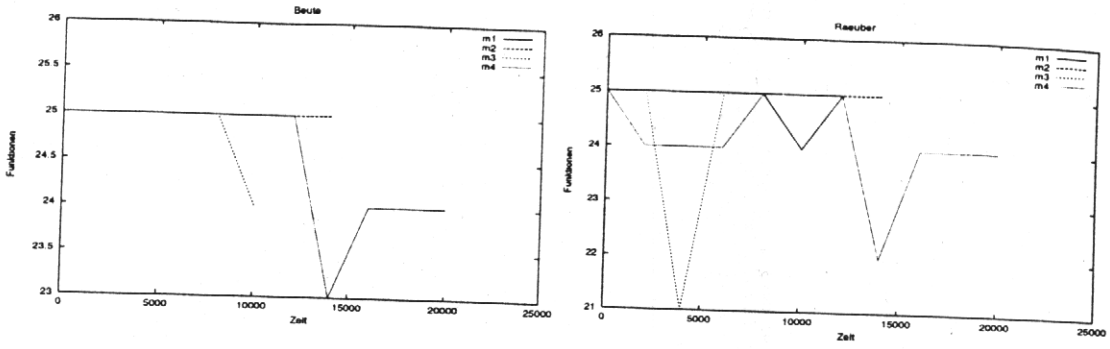

Abbildung 4.7: Anzahl der einzelnen Funktionen über der Zeit bei Beute- und Räuberagenten.

tendenz sichtbar, aber die Kurven fallen nicht monoton.[4] Dieser krasse Unterschied macht deutlich welchen starken Einfluß die Mutation auf den Genpool, und damit letztlich auf das Verhalten der Agenten hat.

4.4.2 Stabilität

Die Tatsache, das aus dem Genpool entfernte Funktionen sehr schnell wieder eingesetzt werden, liefert wahrscheinlich auch die Erklärung für die geringe Stabilität der Simulationen. Es drängt sich die Vermutung auf, daß besonders durch den starken Eingriff der Baummutation bereits bestehende erfolgreiche Strategien immer wieder zerstört werden.

Unbeachtet der Tatsache, daß die Datenmenge eigentlich zu gering ist um längerfristige Beobachtungen anzustellen, läßt sich doch zumindest sagen, daß in den Simulationen längerfristig Agenten mit guten Strategien (z.B. viele *FoodAhead? Turn* Kombinationen, und viele *Eat*), im Timestep 10001 noch sehr häufig sind, später aber immer weniger werden. Kurz vor dem Aussterben ist den Sequenzen jegliche Ähnlichkeit mit ihrem Ahnen abhanden gekommen. Diese Agenten sind nicht mehr überlebensfähig.

Eine weitere Möglichkeit ist, daß eine erhöhte Anzahl von Funktionen den Anteil wichtiger Funktionen im Genpool verringert. Für diese Erklärung gab es allerdings keine Anhaltspunkte. Abb.4.8 zeigt den Anteil einiger Funktionen am Genpool im Timestep 10001 für die Simulationen n4, n11, m1 und m3. Die Werte lassen keinen Rückschluß darüber zu, daß wichtige Funktionen im Vergleich zu anderen Läufen weniger vertreten sind.

[4]Hierbei sollte man allerdings die unterschiedlichen Zeitskalen beachten, die bedingt durch das frühe aussterben der Populationen eingesetzt werden mußten.

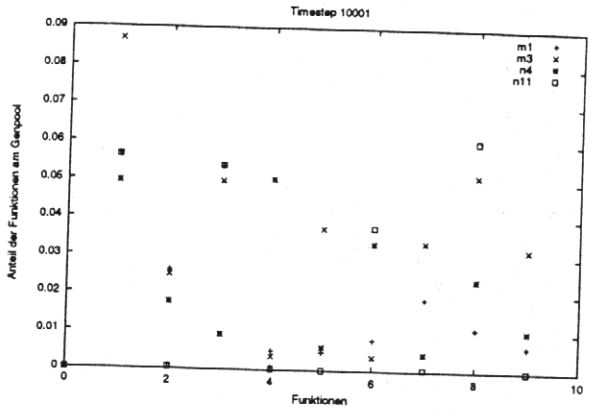

Abbildung 4.8: Anteil einiger Funktionen am Genpool bei der Beute. Dabei ist 1 = Mate, 2 = Eat, 3 = Move, 4 = TurnLeft, 5 = TurnRight, 6 = AgentHere?, 7 = AgentAhead?, 8 = FoodHere? und 9 = FoodAhead?.

4.4.3 Zusammenfassung

Von beiden Mutationsparametern, vollführt die Baummutation den stärksten Eingriff in die Programmstruktur, da hier ganze Teilbäume ersetzt werden. Es zeigt sich, daß beide Mutationsparameter zusammen sowohl zur Destabilisierung des Gesamtsystems, als auch zur Zerstörung einzelner Programmstrukturen beitragen. Es muß also eine Möglichkeit gefunden werden, diese Nachteile zu umgehen. Eine Möglichkeit ist, die Mutation auf reine Punktmutation zu beschränken.

4.5 4. Simulationsreihe

Nachdem sich gezeigt hatte, daß die Baummutation ein zu starker Eingriff in die Programmstruktur darstellt, sollte die Möglichkeit getestet werden, mit einer Punktmutation zu einem umfangreicheren Genpool zu kommen. Dazu wurde der Wert der Baummutation wieder auf seinen Ausgangswert gesetzt. Von 108 Simulationsläufen erfüllten 5 das schwache Stabilitätskriterium. Keine Simulation erreichte Timestep 50001. Tab. 4.6 faßt die Simulationen zusammen.

4.5.1 Genpool

Da in Simulationen mit einem Agententyp eine Punktmutation nicht zu einer sehr hohen Instabilität geführt hat, erhebt sich erneut die Frage nach dem Grund. Zunächst einmal ist es wichtig zu wissen, wie sich die Mutation auf den Genpool ausgewirkt hat. Abb 4.9 zeigt die Anzahl einzelner Funktionen im Genpool. Wie die Abb. zeigt, hat sich an der Situation gegenüber der dritten Simulationsreihe nichts geändert. Die

Name	Seed	letzter Timestep
k1	895228839	11601
k2	895238173	30501
k3	898854859	18401
k4	898949741	11001
k5	899060975	10678

Tabelle 4.6: Stabile Simulationsläufe in der vierten Simulationsreihe

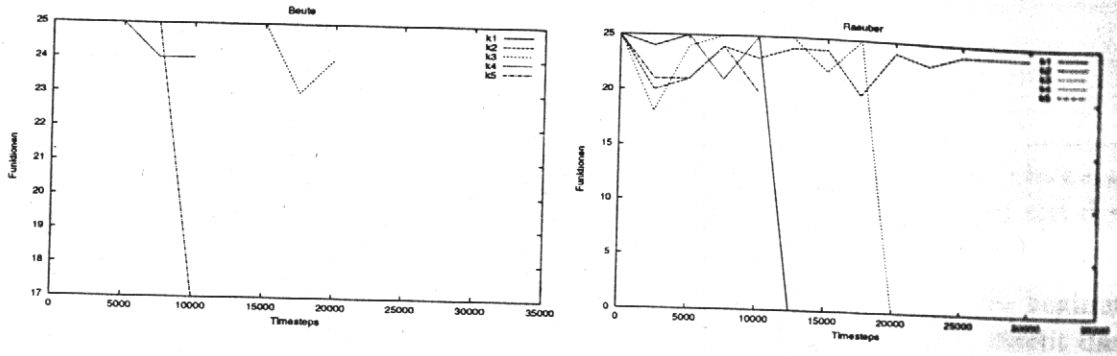

Abbildung 4.9: Anzahl der einzelnen Funktionen über der Zeit bei Beute- und Räuberagenten.

Vermutungen über die Stabilität scheinen auch hier zu gelten. Die Anzahl der einzelnen Funktionen im Genpool fällt nicht monoton, es findet also Mutation statt, die verloren gegangenes Genmaterial wieder zur Verfügung stellt. Diese Mutation fällt allerdings deutlich geringer aus, als in Simulationen mit erhöhter Wahrscheinlichkeit für eine Baummutation.

4.6 5. Simulationsreihe

Diese letzte Simulationsreihe sollte Simulationsläufe ohne Räuber präsentieren, um einen Vergleich von Räuber-Beute-Systemen mit einfachen Modellen zu ermöglichen. Es wurden dabei nur 20 Stichproben gemacht, von denen gleich drei das starke Stabilitätskriterium erfüllten. Ein Überblick ist in Tab. 4.7 dargestellt.

Ein Beispiel für den Populationsverlauf ist in Abb. 3.1 zu sehen. Deutlich erkennbar ist der Unterschied in den Populationsgrößen im Durchschnitt, und in der Standardabweichung. Beides geht eindeutig auf den Einfluß der Räuber, und ihren wechselnden Populationsgrößen zurück. Gegenüber den anderen Simulationen lassen sich in den Populationskurven keine drastischen Veränderungen feststellen.

Name	Seed	Durchschnitt	Standard-abweichung
a1	897756927	1699	49
a2	897762157	1645	86
a3	897767553	1688	71

Tabelle 4.7: Stabile Simulationen in der fünften Simulationsreihe.

4.6.1 Entwicklung der Sequenzen

Es zeigt sich, daß die Konvergenz noch stärker ist als in den vorhergehenden Simulationsreihen. Alle Lösungen in den Simulationen ohne Räuber sind überraschenderweise hochspezialisiert. Es dominieren Varianten der Lösungen für Beuteagenten aus den Räuber-Beute-Systemen.

Für die Simulationsläufe a1 und a2 läßt sich die Lösung zusammenfassen. Sie beginnt in beiden Fällen mit drei *FoodAhead? Turn* Kombinationen. In Lauf a1 strebt das System sogar eine Lösung an, die mit einer weiteren Kombination dieser Art fortfährt. In allen anderen Fällen folgt ein *AgentAhead?*, was einer Klärung bedarf. In a1 gab es zwei Sequenzen in denen dies der Fall war. In a2 waren es drei Sequenzen. Leider konnte der Einfluß von *AgentAhead?* auf den Programmcode nicht überprüft werden, da in den jeweiligen Stichproben jeweils nur eine der beiden Bedingungen erfüllt war. In a1 folgen in der Sequenz nur noch *Eat*, *Turn* und *Move* Funktionen. In a2 kommt in jedem Fall noch eine weitere *FoodAhead?-Eat* Kombination hinzu. Betrachtet man die Sequenzen als stark verwandt, so läßt sich ihr Erfolg anhand der Abb nachvollziehen. Dabei wurden die Vorkommen der Erfolgreichen Sequenzen, die wie oben beschrieben aufgebaut sind, und vermutlich grundsätzlich dieselbe Strategie vertreten, aufaddiert, und der Population gegenüber dargestellt. Wie zu sehen ist erreicht die Strategie fast die Population, was darauf hin deutet, daß die Streuung noch geringer als bei den Räuber-Beute-Simulationen ist, da weniger Agenten von dieser Strategie abweichen.

In a3 scheint die Entwicklung auch geradlinig zu verlaufen. Allerdings scheinen in der Entwicklung zwei Teilsequenzen, die weitgehend unabhängig voneinander sind, wichtig zu sein. Zu Beginn steht dabei meist eine *FoodAhead? TurnLeft* Kombination, während danach zwischendurch immer die Folge *TurnLeft Move Eat* eine Rolle spielt. Dies ist wohl der kleinste gemeinsame Nenner, da die Streuung hier besonders gegen Ende der Simulation sehr hoch ist. In dieser Simulation spielt also die nicht zielgerichtete Suche eine größere Rolle.

4.6.2 Schlußfolgerungen

Die Unterschiedlichen Entwicklungen zeigen daß die hohe Spezialisierung der Agenten nicht zwingend ist. Möglicherweise ist die gerade durch die fehlende Konkurrenz zu

erklären, denn es gibt keinen intuitiv faßbaren Grund warum sich das Systemverhalten ändern sollte. Weiterhin läßt die starke Konvergenz vermuten, daß eine einmal eingeschlagene Entwicklungsrichtung endgültig ist, da sie sich nicht durch neue Gene ummodellieren lassen kann.

Kapitel 5

Schlußfolgerungen

5.1 Emergenz

Das Räuber-Beute-Modell zeigt in der AL-Welt einen komplexeren Zusammenhang, als in einfachen Modellen. Gibt es nur einen Agententyp, der seine Energie aus Food und Carrion bezieht, verteilen sich alle Agenten und anderen Objekte gleichmäßig über die Welt. Dabei bleiben sowohl die Population, als auch die Vorkommen an Food und Carrion fast konstant. Nach 10000 Timesteps sterben die Agenten so gut wie nie aus.

In Räuber-Beute-Modellen sieht das Verhältnis Grundsätzlich anders aus. Hier sind alle Objekte in hohem Maße voneinander abhängig. Meist liegen an verschiedenen Orten unbeweglich kleine Banden von Räubern, und warten darauf, daß Beuteagenten vorbeikommen, um das Food zu fressen, daß sich neben den Räubern ansiedelt. Verfolgt man die Entwicklung in der Einschwingphase, so stellt man fest, daß der Einbruch der Populationen um die Timesteps 1000 und 1200, von starkem Foodwachstum begünstigt wird. Vor allem Räuber haben nur dann eine Überlebenschance wenn sie sich zu diesem Zeitpunkt zu einer Gruppe zusammenfinden, wo es noch Platz für Nachwuchs gibt. Während die Beuteagenten häufig in der Lage sind, eine stabile Population aufzubauen, da Food für sie ja kein Hindernis sondern eine Energiequelle ist, müssen die Räuber darauf hoffen, daß durch eine erhöhte Beutepopulation ihre Energiequellen gesichert werden können. Dies ist der Grund weshalb Räuber es vergleichsweise schwerer haben sich durchzusetzen.

Dieses Verhalten des Systems ändert sich jedoch nie, da die Agentenstrategien sich immer ähnlicher werden.

5.2 Koevolution

Die Frage noch einer Koevolution ist aufgrund der vorliegenden Daten nur schwer zu beantworten. In Anbetracht der Ergebnisse, läßt sich feststellen, daß es immer wieder Punkte während der Simulation gab, in denen ein Zusammenhang naheliegt. Dieser Zusammenhang ist vor allem zwischen der Entwicklung des Foodvorkommens und den Beutecodes sichtbar. Ein direkter Zusammenhang zwischen Räuber-und Beutecode besteht bestenfalls zu Beginn der Simulation. Ein anderer Zusammenhang zwischen Räuber und Beute ist allerdings deutlich. Da die Räuber zur Destabilisierung des Systems beigetragen haben, traten immer wieder die Stellen auf, an denen sich neue Beutestrategien etablieren konnten.

Dies ist streng genommen aber noch keine Koevolution, da es zu keiner Optimierung auf Seiten der Räuberagenten kommt. Man kann also die Frage nach der Evolution bejahen, aber die Frage nach der Koevolution verneinen. Pointiert läßt sich sagen, daß die Räuber sich als Teil einer Steuerungsfunktion einer reinen AL Umgebung etablieren, die nach einer Etablierungsphase statisch bleibt. Eine qualitative Änderung des emergenten Verhaltens, die auf eine Koevolution der Strategien hinweist, war also nicht erkennbar.

Die Möglichkeit innovativere Strategien zu bilden wird längerfristig durch das Verarmen des Genpools verhindert. Da die Möglichkeiten der Mutation in Philia die Populationen zu stark destabilisieren können, wäre eine Möglichkeit vielleicht doch noch zu einer Koevolution zu finden, den Mutationsoperator zu modifizieren. Die einfachste Möglichkeit wäre dafür zu sorgen, daß die Population nicht ausstirbt, was allerdings auch die Konkurrenzsituation entschärfen würde. Eine zweite Möglichkeit, ist die Welt drastisch zu vergrößern, und damit eine erhöhte Population zu ermöglichen. Dadurch könnten möglicherweise, ein regionales Aussterben einer Population, durch eine andere Ausgeglichen werden. Praktikabel wäre auch die Mutation regional zu beschränken. Auf niedrigeren Evolutionsstufen gibt es hierzu Parallelen in der Natur.

5.3 Implizite Fitneß

Überraschenderweise zeigt ein Vergleich von Räuber-Beute-Systemen mit einfachen Modellen, daß in einfacheren Modellen eine höhere Spezialisierung möglich ist. „Fit" muß sich nicht unbedingt auf den Agenten selbst beziehen, wenn die Umstände (z.B. Konkurrenz) andere Stabilitätskriterien nahelegen. Es ist allerdings erstaunlich, daß die Agenten sich in dieser Situation ineffektiver Entwickelt haben.

Das Auftauchen immer wiederkehrender Kombinationen, sowohl bei Räuber-Beute-Systemen, als auch bei Systemen mit nur einem Agententyp, legt nahe, daß die implizite Fitneß zumindest vorübergehend nur eine Standardlösung zuläßt. Da die fehlenden Entwicklungsmöglichkeiten keine weiteren Innovationen mehr zuließen, konnte

nur festgestellt werden, daß keine kurzfristigen Veränderungen stattfanden, vor allem bei den Räubern, die schon nach sehr kurzer Zeit ihre Strategie nicht mehr änderten, während die Beutestrategien leicht variierten.

Im Gegensatz zu klassischen GP-Methoden mit Koevolution, muß aber berücksichtigt werden, daß zum einen die Populationsgröße stark schwankt, bedingt durch den Konkurrenzkampf, und damit die Auswahl der Agenten eben nicht durch eine Fitneßfunktion, sondern durch ein zufälliges Moment bestimmt ist, und zum anderen besonders spezialisierte Beuteagenten am ehesten zu den Räubern finden, da die Wahrscheinlichkeit dort auf Food zu treffen am höchsten ist. Beide Argumente zusammen bieten eine mögliche Erklärung für diesen Umstand. Allerdings liegt es auch im Bereich des Möglichen, daß die Agenten sich auch in einem Räuber-Beute-Szenario zu einem späteren Zeitpunkt noch stärker spezialisieren, wenn sie die Möglichkeit dazu haben. Trotzdem scheint, im Vergleich mit einfachen Modellen, das Attraktorgebiet über einen längeren Zeitraum verschoben worden zu sein.

Kapitel 6

Zusammenfassung

Diese Arbeit sollte zeigen, ob eine Codeoptimierung bezogen auf eine implizite Fitneß in einem GPAL-System, durch Koevolution möglich ist. Dabei wurden zwei verschiedene Gesichtspunkte beachtet:

1. Das Hauptaugenmerk richtete sich auf die Strategien, die die Agenten verwenden, um emergentes Verhalten zu erzeugen. Das Problem in diesem Zusammenhang bestand in der Herleitung der Strategien der Agenten aus ihrem Verhalten. Es wurden die Sequenzen von Funktionen die das Agentenprogramm erzeugt herangezogen um eine Strategie herleiten zu können. Zur Klassifizierung mußten äquivalente Strategien gefunden werden. Dies war nur teilweise möglich.

2. Das emergente Verhalten des Systems sollte Aufschluß auf Veränderungen der ihm zugrunde liegenden lokalen Regeln, in diesem Fall die Agentenprogramme, bringen. Dieser Ansatz scheiterte als klar wurde, daß sich das globale Systemverhalten nicht änderte. Hierbei bestand das Problem, das nur durch die Kenntnis der Grundlegenden Strategien ein Rückschluß möglich war. Deswegen mußte die Entwicklung des Systems anhand der Populationen mit der Entwicklungen der Strategien, in Zusammenhang gebracht werden.

Das Kapitel 4 zeigt, daß eine einfache evolutionäre Entwicklung zumindest auf Seiten der Beute möglich ist. Allerdings wird vor allem deutlich, daß eine Weiterentwicklung des Systems durch einige Eigenschaften des zugrundeliegenden Modells, wie z.B. den Mutationsoperator, verhindert wird. In weiteren Untersuchungen müßte eine Möglichkeit gefunden werden, diese Nachteile zu umgehen. Einige Möglichkeiten dazu sind in Kapitel 5 aufgeführt.

An diesem Punkt könnte dann ein erneuter Versuch ansetzen, eine Koevolution in Philia zu etablieren.

Literaturverzeichnis

[1] Endbericht der PG *Realisierung und Anwendung eines Genetic-Programming/ Artificial-Life-Systems*, WS 95/96, Universität Dortmund, Fachbereich Informatik, LS XI

[2] Ackley, D.H.; Littman, M. (1992). *Interactions Between Learning and Evolution*, Artificial Life II, SFI

[3] Allen, J. (1978). *Anatomy of Lisp* New York: McGraw-Hill

[4] Altenberg, L. (1994). *Emergent Phenomena in Genetic Programming*, In *Evolutionary Programming. Proceeddings of the Third Annual Conference*, World Scientific, Singapore

[5] Banzhaf, W.; Nordin, P., Keller, R.E.; Francone, F.D. (1998). *Genetic Programming. An Introduction*, Morgan Kaufmann, San Francisco, CA

[6] Belev, R.K.; Booker, L.B. (eds) (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Francisco, CA

[7] Cliff, D.;Husbands, P.; Meyer, J.-A.; Wilson, S.W. (eds.) (1994). *From Animals to Animats 3: Proceeding of the Third International Conferenceon Simulation of Adaptive Behavior*, MIT Press, Cambridge, Massachusetts

[8] Coveney, P; Highfield, R. (1990). *The Arrow Of Time*, W.H. Allen, London

[9] Dain, R.A. (1997). *Genetic Programming for mobile robot wall-following algorithms* In: Koza, J.R.; Deb, K.; Dorigo, M.; Fogel, D.B.; Garzon, M.; Iba, H.; Riolo., R.L., edit. textitGenetic Programming 1997. Proceedings of the Second Annual Conference, Stanford University, CA. Morgan Kaufmann, San Francisco, CA

[10] Darwin, C. (1860). *On the Origin Of Species by Means of Natural Selection, or the Preservation of Favoured Races inthe Struggle of Life*, Nachdruck, London: John Murray

[11] Dawkins, R. (1989). *The Selfish Gene* Oxford University Press, Oxford, UK

[12] Déscartes, R. (1637). *Von der Methode des richtigen Vernunftgebrauchs und der wissenschaftlichen Forschung*, Nachdruck 1978 (Dt. Übersetzung), Felix Meiner

[13] Dörner, D. (1989). *Die Logik des Mißlingens. Strategisches Denken in komplexen Situationen*, Rowohlt

[14] Eigen, M.;Schuster, P. (1982). *Stages of Emerging Life. Five Principles of Early Organisation*, J. Mol. Biol. 19, 47-61

[15] Fleischer, K.; Barr, A.H. (1994). *A Simulated Testbed for the Study of Multcellular Development: The Multiple Mechanisms of Morphogenensis*, Artificial Life III, SFI

[16] Fontana, W. (1990). *Algorithmic Chemistry: A New Approach to Functional Self-Organization*, Technical Report LA-UR 90-3431, Los Alamos National Laboratory

[17] Forrester, S; Jones, T.; (1994). *Modeling Complex Adaptive Systems with Echo*, Complex Systems: Mechanism of Adaption, R.J. Stonier and X.H. Yu ed., IOS Press, Amsterdam

[18] *Genetic Programming Mailing List.* genetic-programming@cs.stanford.edu

[19] Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, an Machine Learning*, Addison-Wesley

[20] Gustafson, D.; Barrett, W.; Bates, R.; Couch, J.D. (1986). *Compiler Construction: Theory and Practice* Science Research Assoc.

[21] Halder, A.; Müller, M. (1993). *Philosophisches Wörterbuch*, (2.A.) Herder Freiburg im Breisgau

[22] Harvey, I. (1994) *Evolutionary Robotics and SAGA: The Case for Hill-Crawling and Tournament Selection*, Artificial Life III, SFI

[23] Hillis, W.D. (1989). *Co-Evolving Parasites* In: Forrest, S. (ed.), Emergent Computation, MIT Press, Cambridge, MA

[24] Hillis, W.D. (1990). *Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure*, in: Physica D 42

[25] Holland, J.H. (1992). *Adaption in Natural and Artificial Systems* (2.A.), MIT Press Cambridge Massachusetts

[26] Iba, H. (1997). *Multiple-agent Learning for a Robot Navigation Task by Genetic Programming* In: Koza, J.R.; Deb, K.; Dorigo, M.; Fogel, D.B.; Garzon, M.; Iba, H.; Riolo, R.L.; (ed.) Genetic Programming 1997: Proceedings of the Second Annual Conference, Stanford University, CA. Morgan Kaufmann, San Francisco, CA

[27] Koza, John R. (1992). *Genetic Programming*, MIT Press Cambridge, Massachusetts

[28] Koza, John R. (1994). *Artificial Life: Spontaneous Emergence of Self-Replicating and Evolutionary Self-Improving Computer Programs*, Artificial Life III, SFI

[29] *John Koza's Home Page.* http://www-cs-faculty.stanford.edu/ koza

[30] Langton, C.G. (1989a). *Artificial Life*, in Langton (1989b)

[31] Langton, C.G. (Ed.) (1989b). *Artificial Life, SFI Studies in the Science of Complexity*, SFI

[32] Langton, C.G. (Ed.) (1997). *Artificial Life. An Overview*, MIT Press Cambridge, Massachusetts

[33] Leibniz, G.W. (1714). *Monadologie*, Nachdruck 1979 (Dt. Übersetzung, 2.A.), Phillipp Reclam jun., Stuttgart

[34] Lewin, R. (1992) *Complexity. Life at the Edge of Chaos*, New York: Macmillan

[35] Maes, P. (ed.) (1990). *Designing Automonous Agents*, MIT Press, Cambridge, Massachusetts

[36] Nagel, K., Rasmussen, S. (1994). *Traffic at the Edge of Chaos*, Artificial Life IV, MIT Press

[37] Nestle, W. (1977). *Aristoteles. Hauptwerke.* (8.A.), Kröner Stuttgart

[38] Pimm, S.L. et al. (1991). *Food web patterns and their consequences*, Nature vol. 350

[39] Ray, T.S. (1992). *An Approach to the Synthesis of Life*, Artificial Life II, SFI

[40] Ray, T.S. (1997) *An Evolutionary Approach to Synthetic Biology: Zen and the Art of Life*, in Langton (1997).

[41] Steele, G.L. (1982). *An Overview of Common Lisp.* In: Proceedings of the ACM Symposium on Lisp and Functional Programming

[42] Taylor, C.E.; Jefferson, D.R.; Turner, S.R.; Goldman, S.R. (1989). *RAM: Artificial Life for the Eploration of Complex biological systems* In: AL, SFI

[43] Törn, A.;Zilinskas, A. (1989) *Global Optimization*, Berlin, Springer

[44] Turning, A.M. (1936). *On Computable Numbers with an Application to the Entscheidungsproblem*, in: Proc. London Math. Soc. Ser. 2-42

[45] Varela, F.J.; Bourgine, P. (eds) (1991). *Towards a Practice of Autonomous Systems: Proceeding of the First Conference on Artificial Life*, MIT Press, Cambridge

[46] Vogel, G.; Hartmut (1985) *dtv-Atlas zur Biologie*, dtv

[47] von Neumann, J. (1966). *Theory of Self-reproducing Automata*, University of Illinois Press

[48] Williams, C.G. (1992). *Natural Selection: Domains, levels, and challanges* New York: Oxford University Press

[49] Wuketits, F.M. (1982) *Grundriß der Evolutionstheorie*, Darmstadt, Wissenschaftliche Buchgesellschaft

[50] Yaeger, L. (1994). *Computational Genetics, Physiology, Metabolism, NeuralSystems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context*, Artificial Life III, SFI