Das TSH Projekt

Projektbericht

Jahn-Takeshi Saito, Timo Steffens und Joachim Wagner

Datum: 29. Februar 2000
Version: 0.11

Vorwort

Der vorliegende Text ist eine lückenhafte Vorabfassung, deren Inhalt noch nicht mit Jahn-Takeshi Saito und Timo Steffens abgesprochen ist. Die Entscheidung auf eine Projektbezeichnung ist auch noch nicht gefallen. Des weiteren ist zu beachten, daß für die meisten Versuchen noch nicht genug Messungen durchgeführt wurden, um alle Einflußgrößen zu erkennen. Auch sind nicht alle Versuche mit unseren Programmen sofort nachvollziehbar, da die Funktionen laufend erweitert werden. Unsere Programmmodule sind aber bereits so angelegt, daß dies später sehr einfach nachgeholt werden kann. Für jeden Versuch wird es dann ein kurzes Hauptprogramm geben, mit dem die Versuche mit verschiedene Parametern durchgeführt werden können.

Entwicklung

Am 18. Februar 1999 haben wir, das sind Jahn-Takeshi Saito, Timo Steffens und Joachim Wagner, uns getroffen und die Grundzüge unseres AI-Ansatzes festgelegt. Nach weiterer Planung am 09. März wurde dann bis zum 08. April programmiert und Details des Ansatzes überarbeitet. Am 16. April konnten wir dann erste Experimente mit unserer AI durchführen. Da gleichzeitig das Semester begann, hatten wir aber nicht genug Zeit, um den Erfolg der AI unter verschiedenen Bedingungen in der simulierten Welt zu messen. Danach folgten nur noch kleine, programmiertechnische Änderungen und ab dem 11. Mai 1999 das Quellenstudium für die Bewertung unserer Ergebnisse.

Nach einem weiteren Treffen am 28. Mai begann dann die Arbeit an diesem Projektbericht. Im Juni haben wir zusätzliche Versuche mit der AI durchgeführt. Dazu wurde u.a. ein Modul zur genetischen Optimierung erstellt. Am 30. Juni 1999 haben wir unser Projekt und die ersten Ergebnisse im Rahmen der CogSci-Referate-AG vorgestellt. Erst ab dem 28. Juli wurde weiter an dem Bericht gearbeitet, um am 09. August eine vorläufige Version im WWW zur Verfügung stellen zu können. Erst im Februar 2000 setzten wir die Arbeit fort.

Osnabrück, den 29. Februar 2000

Joachim Wagner

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In der Veranstaltung "Methoden der KI", an der wir, Jahn-Takeshi Saito, Timo Steffens und Joachim Wagner, mit Erfolg teilgenommen haben, wurden die Grundlagen der KI und viele Ansätze vorgestellt. Diese Ansätze unterscheiden sich sehr, und keiner von ihnen kann für sich in Anspruch nehmen, alle Probleme der KI zu lösen. Wir meinen daher, daß die vorhandenen Lösungen zusammen eingesetzt und durch neue Ansätze ergänzt werden müssen.

Bei dem Aufstellen neuer Ideen greift man immer auf bekannte Muster zurück. Bei der Lösung von Teilproblemen, die sich bei der Konkretisierung der eigenen Idee stellen, geht man gerne bereits ausgearbeitete Wege. Häufig wird die eigene Idee dabei verändert. Sie hat schließlich starke Ähnlichkeit mit einem anderen, älteren Lösungsweg.

Natürlich ist dies in der Wissenschaft gewollt. [Beleg?] Man braucht einen Bezug zu den vorhandenen Theorien, um vergleichen und bewerten zu können. Des weiteren können die verschiedenen Ansätze so leicht voneinander profitieren, da für die Teilprobleme ein großer Vorrat von Lösungen bereitsteht. Der unter mehr oder weniger großem Erfolgsdruck stehende Wissenschaftler kann unter diesen Bedingungen kaum riskieren, viel Arbeit in losgelöste Ideen zu investieren. Völlig neue Ideen haben es daher sehr schwer.

Von einem Studenten werden dagegen keine Beiträge erwartet, die das oben erwähnte Lösungsrepertoire vergrößern, insbesondere nicht im Grundstudium. Seine Leistungen beschränken sich in der Regel auf das Zusammenfassen und Gegenüberstellen vorhandener Ansätze. Wir haben dadurch Freiheiten, die im späteren Leben praktisch nicht mehr existieren. Uns bietet sich also die Gelegenheit, unsere Ideen ausprobieren zu können, ohne hohen Erwartungen gerecht werden zu müssen.

1.2 Zielsetzung

Wir wollen eine AI entwickeln, die sich in einer einfachen, simulierten Welt zurechtfindet. Dazu stellen wir einen eigenen Ansatz auf und schreiben ein Computerprogramm, das unseren Ansatz verwirklicht. Dabei interessieren uns besonders die Probleme, die bei der Ausarbeitung des Ansatzes und bei seiner Umsetzung auftreten. Anschließend wollen wir unsere Erkenntnisse mit denen anderer Veröffentlichungen vergleichen.

Unser Projekt hat damit mehrere Ziele. Zum einen wollen wir einen eigenen, neuen Ansatz entwickeln, der möglichst wenig durch die bekannten KI-Methoden geprägt ist. Selbst wenn unsere Lösung nicht "besser" sein sollte als die vorhandenen, wird sie neue Wege zeigen und auch neue Fragen aufwerfen.

Zum anderen steht in diesem Projekt nicht im Vordergrund, den gewählten Ansatz zu verteidigen. Vielmehr sollen mit ihm die Probleme der Entwicklung einer AI deutlich werden.

1.3 Überblick

Dieser Projektbericht ist wie folgt aufgebaut:

Kapitel 1 lesen Sie gerade. Es beschreibt, warum wir dieses Projekt durchgeführen, was der Anlaß war und welche Ziele wir mit diesem Projekt erreichen wollen.

Im Kapitel 2 gehen wir auf den von uns gewählten Ansatz und die Rahmenbedingungen unserer Umsetzung näher ein.

Im Kapitel 3 stellen wir die Versuche vor, die wir mit unsere AI-Umsetzung durchgeführt haben, und versuchen, sie auszuwerten.

Kapitel 4 soll helfen, unseren Ansatz einzuordnen. Wir erläutern einige Lernverfahren, die uns hierzu geeignet erscheinen. Gleichzeitig ist dieses Kapitel Grundlage für die Diskussion im Kapitel 5.

Kapitel 5 beschreibt die Probleme, die bei der Ausarbeitung des AI-Ansatzes auftraten, und zeigt die Parallelität zu Problemen anderer AI-Ansätze.

Im Kapitel 6 erläutern wir, welche Verbesserungsmöglichkeiten wir bei unseren Ansatz sehen.

Anhang A enthält eine Kurzbeschreibung der Programmmodule unserer Implementaion und erklärt, wie die Versuche durchgeführt wurden.

Kapitel 2

Unser Ansatz

In diesem Kapitel beschreiben wir den von uns gewählten Ansatz. Zuerst wird er kurz umschrieben. Dabei führen wir die Begriffe ein, die sich während der Projektarbeit gebildet haben. Orientiert an diesen Begriffen folgt dann eine Beschreibung, wie wir unseren Ansatz ausgestalltet haben. Dazu gehört, wie die im Computer simulierte Welt aussieht, in der sich unsere AI zurechtfinden muß, wie die AI ihr Wissen verarbeitet und wie sie mit der Welt in Kontakt tritt.

2.1 Grundzüge

Die AI speichert bei jeder Handlungswahl ab, welche Sinneseindrücke vorliegen, welche Ziele sie zu der Zeit hat und auf welche Handlung die Wahl fiel. Dieses "Gedächtnis"ist also ein zeitliches Protokoll der sensorischen Eingaben, des Zustands der AI (z.B. Ziele) und der gewählten Handlungen. Die Gesamtheit der gespeicherten Informationen zu einem Zeitpunkt nennen wir eine *Zeitscheibe*.

Um zu entscheiden, welche Handlung die AI ausführen soll, ruft sie ihren Gedächtnisinhalt ab. Dazu werden zur aktuellen Situation und zum Ziel der AI Zeitscheiben generiert und im Gedächtnis ein ähnliches Paar gesucht, wobei natürlich die Zielscheibe zeitlich hinter der Ausgangsscheibe liegen muß, der die Handlung entnommen wird. (Die Idee, auf ähnliche Situationen zurückzugreifen, hatte einer von uns, der ein Vortrag über Case-Based Reasoning gehört hatt.)

Die Ziele der AI sollen sicherstellen, daß sie in gleichen Situationen nicht immer die gleiche Handlung ausführt, sondern mitbewertet, ob sie ihren Zielen näher kommt. Die AI soll ihre Ziele selbst bestimmen können, z.B. Zwischenziele generieren. Uns ist aber noch nicht klar, wie dies geschehen soll. Daher sind die Ziele vorerst statisch von außen vorgegeben.

Im folgenden wird dieser Ansatz weiter konkretesiert, indem wir auf die einzelnen Aspekte näher eingehen. Dabei sollten die oben verwendeten Begriffe klarer werden. Zwar orientiert sich die Erläuterung an unserer Implementation der AI. Aber wir werden hier trotzdem allgemein bleiben. Deatails zur Implementation findet man im Anhang A und natürlich im Quellcode.

2.2 Simulierte Welt

Wir haben versucht, die AI möglichst unabhängig von der Darstellung und den Gestztmäßigkeiten der Welt zu halten, indem nur das Sensorik-Modul und die elemtaren Handlungen als Schnittstelle dienen. Entscheidend für die AI ist aber, daß die Welt keine Echtzeitanforderungen an sie stellt. Der Zustand der Welt ändert sich nicht, während die AI ihre nächste Handlung berechnet. Die Dauer der Berechnung hat also keine Auswirkungen.

Der Weltzustand wird in einer Matrix von Zellen gespeichert. Jede Zelle hat Attribute, wie z.B. *Farbe*, *Begehbarkeit* und *Verhaltensweise*. Ist der Wert letzteren Attributes "AI", dann befindet sich in der entsprechenden Zelle eine AI, die sich gemäß dem AI-Modul verhält. Die Welt benutzt also diesen Vermerk, um das Modul zu bestimmen, das die auszuführende Handlung berechnet. Die Welt durchläuft die Matrix zeilenweise und fordert jede Zelle zum Handeln auf. Ein Zähler stellt sicher, daß die AI nach einer Bewegung nach rechts oder unten nicht sofort wieder eine Handlung ausführen kann.[1]

Uns ist klar, daß dieses Modell Probleme hervorruft, wenn in mehreren, benachbarten Zellen Handlungen ausgeführt werden. Die Auswirkungen wären nicht intuitiv. Wir meinen aber, daß dies kein Nachteil für die Simulation ist. Der zeitliche Ablauf der Handlungen ist anders als in der realen Welt. Die AI sollte damit aber keine Schwierigkeiten haben und genausogut lernen können wie in einem aufwendigeren Modell.

2.3 Unsere AI

Die AI tritt immer dann in Aktion, wenn sie von der Welt aufgefordert wird zu handeln. Unsere AI fragt daraufhin die Sinndeseindrücke ab und baut eine neue Zeitscheibe, die in das Gedächtnis eingefügt wird. Dann wird eine Handlung gewählt, in die neugebaute Zeitscheibe (und damit in das Gedächtnis) eingetragen und der Welt als Antwort zurückgegeben.

2.3.1 Sinneseindrücke

Über das Sensorik-Modul kann die AI einen Teil der Welt erfassen. Die Informationen, die in der Welt gespeichert sind, werden auf verschiedene Weise gefiltert und aufbereitet. Abgesehen vom Zustand der AI, der zwei Zahlen für *Hunger* und *Schmerz* umfaßt, erhält die AI nur durch die Sensorik Informationen über der Welt.

Die Sinneseindrücke sind neben dem Weltzustand auch von der räumlichen Position und Ausrichtung der AI abhängig. Jeder Zugriff auf die Welt erfolgt relativ zu der AI, so wie man es erwartet. Es werden also nur einfache Drehungen und Verschiebungen der Koordinaten durchgeführt.

[1] Zu den bisher implemtierten Zellentypen siehe Abschnitt 3.2.1 und Anhang A.

Für die visuellen Eindrücke werden nur Weltzellen berücksichtigt, die im Sichtfeld der AI liegen, das bei unserer AI von -45 bis +45 Grad und vier Zellen weit reicht. Steine, die größer als zehn Einheiten sind, und Nahrungsquellen verdecken alles, was "hinter" ihnen liegt. Auf den visuellen Eindruck haben nur die Eigenschaften *Farbe*, *Glattheit* und *Größe* Einfluß. Die Genauigkeit der Werte wird vom Sensorik-Modul stark reduziert. Werte über hundert Einheiten für *Glattheit* oder *Größe* werden nicht mehr differenziert.[2]

Die AI "sieht" die drei Werte in jeweils 16 Stufen. Je Zelle sind also $16^3 = 4096$ unterschiedliche Eindrücke möglich. Des weiteren befinden sich wegen der Sichtreichweite von vier Zellen bis zu 16 Zellen im Sichtfeld. (Es sind entsprechend weniger Zellen, wenn eine Sichtblockade vorliegt.) Die Zahl der möglichen visuellen Sinneseindrücke ist also trotz der starken Informationsreduktion sehr groß ($4096^{16} \approx 10^{57}$).

Neben den visuellen Eindrücken erhält die AI auch vier auditorische Werte, die jeweils eine Amplitude (16 Stufen) und einen Charakter (Kombinationen von *klopfend*, *schrill* und *dumpf*) haben. Allerdings herrscht in unserer simulierten Welt bisher absolute Stille, so daß dieser Teil der Sensorik keine Auswirkung hat.

Betrachtet man die Implementation genau, so entdeckt man eine Stelle, an der die strikte Trennung von Welt- und AI-Modul verletzt wird. Für Hunger und Schmerz haben wir zur Vereinfachung eine direkte Verbindung zwischen diesen beiden Modulen gewählt. Sie können beide die entsprechenden Daten lesen und manipulieren. Dies ist bei unserer Moduleinteilung notwendig, da die Welt Hunger und Schmerz verändern können muß, z.B. wenn die AI gegen einen Stein läuft, aber auch die AI Zugriff auf die Werte haben muß. Das ganze ist aber nur ein theoretisches Problem, so lange wir im AI-Modul sicherstellen, daß die Werte für Hunger und Schmerz nicht verändert werden. Z.B. darf die AI den Hungerzähler nicht einfach erniedrigen anstatt etwas zu essen.[3]

2.3.2 Zeitscheiben

Eine Zeitscheibe bündelt Sensordaten, drei Ziele, den Zustand der AI (eigentlich zu den Sensordaten gehörend) und die Handlung. Diese Informationseinheit ist von zentraler Bedeutung für unsere AI. Sie ist die Grundlage für Gedächtnis, Ziele und Handlungswahl unserer AI.

Im AI-Modul wird eine Metrik über den Zeitscheiben definiert. So kann der Unterschied zwischen zwei Zeitscheiben skalar, also mit einer Zahl, ausgedrückt werden. Für die Distanz zweier Zeitscheiben s_1 und s_2 schreiben wir im folgenden $d(s_1, s_2)$. Die zweite Scheibe s_2 darf dabei Joker enthalten. Die betroffenen Komponenten werden dann so behandelt, als ob sie in s_1 und s_2 gleich sind. Enthält s_2 nur Joker, so ist $d(s_1, s_2) = 0$. Andersherum ist d aber nicht definiert. s_1 darf also keine Joker enthalten. Abgesehen davon ist d aber symmetrisch, es gilt also $d(s_1, s_2) = d(s_2, s_1)$ für alle Zeitscheiben ohne Joker.

[2] Für die genauen Regeln siehe Source-Code.

[3] Alternativ könnten wir jeder Zelle der Welt weitere Eigenschaften zuschreiben, die vom Zellentyp abhängig sind. Diese Daten könnten dann in der Sensorik in *Hunger* und *Schmerz* umgesetzen werden. Auf die zellentypspezifischen Daten anderer Zellen als der eigenen Zelle dürfte die Sensorik der AI natürlich keinen Zugriff haben. (Außer wenn man Telepathie der AI ermöglichen will.)

2.3.3 Gedächtnis

Das Gedächtnis der AI besteht aus einer chronologischen Liste aller Zeitscheiben, die bei jeder Handlungswahl gespeichert wurden, und einer Angabe, auf welche Abschnitte der Zeitscheibenliste und in welcher Reihenfolge zugegriffen werden soll. Die erste Version unserer AI hat im Gedächtnis nur einen solchen Abschnitt angelegt, der die gesamte Zeitscheibenliste umfaßt. Da die Suche in großen Abschnitten sehr rechenintensiv ist, ist aber so die Erfassung von Messwerten nicht durchführbar. Daher wird nun die Zeitscheibenliste von kürzeren, sich überlappenden Abschnitten abgedeckt. Es wird also immer die vollständige Zeitscheibenliste verwendet.

2.3.4 Ziele

Die AI berücksichtigt bei ihrer Handlungswahl drei unterschiedlich gewichtete Ziele. Jedes Ziel ist eine Zeitscheibe, die Jokerzeichen enthalten darf. Wir haben noch keine Idee, wie unsere AI ihre Ziele eigenständig ändern könnte. Daher sind die Ziele derzeit fest vorgegeben.

2.3.5 Handlungswahl

Zur Wahl einer Handlung wird das Gedächtnis abschnittsweise nach einem Zeitscheibenpaar durchsucht, das zu der aktuellen Situation s_n und den Zielen z_1 bis z_3 am besten paßt. Die Distanzfunktion d (siehe 2.3.2) bildet dafür die Grundlage. Jedes Paar $(s_i, s_j), i \leq j$ wird mit

$$g_0 d(s_i, s_n) + g_1 \sum_{k=1}^{3} g_{k+1} d(s_j, z_k) + g_5 \frac{j-i}{n}$$

bewertet. Je kleiner die Summe ist, desto dichter liegt das Paar an der aktuellen Situation. Der Vektor $\vec{g} = (g_0, g_1, g_2, g_3, g_4, g_5)$ bestimmt die Gewichtung der einzelnen Abweichungen.

Zwei Schwellenwerte beeinflussen die Suche. Der erste Schwellenwert ist sehr niedrig (0.0005). Wird bei der Suche ein Paar gefunden, für das die Bewertung kleiner als dieser Wert ist, dann wird die Suche sofort abgebrochen und das gefundene Paar als bestes Paar ausgegeben. Der zweite Schwellenwert schließt Paare von der Suche aus, die eine zu schlechte Bewertung haben. Ist sie größergleich dem zweiten Schwellenwert, dann wird das Paar ignoriert. Es ist also möglich, daß trotz gefüllter Gedächtnisabschnitte kein "bestes" Paar gefunden wird.

Wurde ein "bestes" Paar (s_i, s_j) gefunden, dann wird aus der Zeitscheibe s_i die Handlung entnommen. Es wird also die gleiche Handlung gewählt, auf die schon zur Zeit t_i die Wahl fiel. Im Fall, daß kein Zeitscheibenpaar gefunden wurde, wird eine Zufallszahl erzeugt und die Handlung nur von ihr abhängig bestimmt. Positiv ausgedrückt bedeutet dies, daß die AI die Welt exploriert. Zusätzlich haben wir die Handlung *Vorangehen* mit einer größeren Wahrscheinlichkeit belegt, damit die AI leichter große Strecken zurücklegen kann.

2.3.6 Handlungen

Unsere AI kann die Welt nicht direkt manipulieren. Nur das Welt-Modul kann bestimmen, welche Auswirkung die versuchte Durchführung einer Handlung hat. Z.B. muß die Handlung *Vorangehen* nicht immer dazu führen, daß die AI sich bewegt. Sie könnte vor einer Wand stehen oder in Leim getreten sein.

Zwischen AI- und Welt-Modul sind Codes für die möglichen Handlungen definiert. Sie sind von null an aufsteigend numeriert. Dem AI-Modul wird über eine Konstante die Anzahl der so definierten Handlungen mitgeteilt. Derzeit sind nur vier Handlungen, Vorangehen, Linksdrehung, Rechtsdrehung und Essen, vorgegeben. Wie gesagt, für die AI sind dies nur Zahlen, deren Bedeutung sie nur über die Auswirkung der Handlungen erfahren kann. Das Modell, nach dem die Welt arbeitet, ist der AI nicht bekannt.

Kapitel 3

Versuche mit der AI

In diesem Kapitel beschreiben wir die Versuche, die wir mit unsere AI gemacht haben.

3.1 Erste manuelle Versuche

Zuerst haben wir versucht, der AI schrittweise elementare Vorgehensweisen beizubringen. Dazu haben wir die AI in eine sehr einfache Welt gesetzt und die Welt dann langsam vergrößert. In folgender Abbildung sind die ersten Welten dargestellt. `S` kennzeichnet Steine und `E` Nahrungsquellen.

```
a) SSS  b) SSSS  c) SSSSS  d) SSSSSS
   E S     E  S     E   S     E    S
   SSS     SSSS     SSSSS     SSSS S
                                  SSS
```

Abbildung 3.1: Schrittweises Lernen in einfachen Welten

In a) soll die AI lernen, Nahrung aufzunehmen und sich zu drehen, wenn sie vor einer Wand steht. In b) und c) geht es darum, einen Gang entlangzugehen. d) deutet an, wie der Lernprozeß weitergehen könnte.

Durchgeführt haben wir den Versuch, indem wir die Datenstrukturen, mit denen die Welt repräsentiert wird, per Hand angepaßt haben. Wir haben die AI nacheinander in die Welten a) bis d) möglichst weit von der Nahrungsquelle ausgesetzt. Die AI wurde jeweils so lange laufen gelassen, bis sie von der Nahrungsquelle gegessen hat. In jedem Schritt hat sich der Inhalt des Gedächtnisses entsprechend vergrössert. Der beobachtete Zuwachs war aber erstaunlich groß, bis zu 100 Zeitscheiben bereits bei b). Ein zielstrebiges Zulaufen der AI auf die Nahrungsquelle ist nicht zu erkennen. Die große Dauer bis zum Erreichen der Nahrungsquelle läßt vermuten, daß die AI nur zufällig handelt.

3.2 Messreihen

Betrachtet man die graphische Darstellung des Simulationsablaufs, dann scheint dem Beobachter das Verhalten der AI selbst dann intelligent zu sein, wenn sie nur zufällsbasiert Handlungen wählt. Dies und die unklaren Ergebnisse der ersten Versuche zeigen, daß bessere Messmöglichkeiten erforderlich sind.

3.2.1 Unabhängige Variablen

In unserer simulierten Welt und bei der AI gibt es sehr viele Freiheitgrade. Jede Zelle der Welt kann beliebig belegt werden, und die Breite und Höhe der Welt kann frei gewählt werden. Um das Kreieren neuer Welten zu vereinfachen, bietet das Welt-Modul eine Funktion, die zufallsbasiert Welten erzeugt. Eine Liste von Belegungsschablonen steuert die Zellenbelegung. Jeder Schablone ist dabei eine Wahrscheinlichkeit zugeordnet. Für jede Zelle der Welt wird zufällig, aber unter Berücksichtigung der angegebenen Wahrscheinlichkeit, eine Schablone ausgewählt. Die Schablone legt fest, von welchen Typ (Stein, Nahrungsquelle, AI etc.) der Zelleninhalt sein soll und weist den weiteren Freiheitsgraden Werte zu.

Das Beispiel in Abbildung 3.2 zeigt eine solche Liste von Belegungsschablonen. Mit ihr kann eine Welt kreiert werden, die etwa 60% leere Felder, 15% Nahrungsquellen, 15% AIs und etwa 10% Steine enthält. Dabei werden für die leeren Felder und für die Steine Standardwerte benutzt, da keine Schablone angebenen ist.

Wahrscheinlichkeit	*Typ*	*Schablone*
0,60	leeres Feld	—
0,10	Nahrung	S_1
0,05	Nahrung	S_2
0,10	AI	S_3
0,05	AI	S_4
0,10	Stein	—

Abbildung 3.2: Schablonen für die Zellenbelegung

Unsere AI hat die meisten Freiheitsgrade. Neben den in 2.3.5 genannten gibt es noch sechs weitere Gewichtungen für visuelle und auditorische Sinneseindrücke, Hunger und Schmerz und für diese beiden Gruppen untereinander.[1] Bei Nahrungsquellen kann im wesentlichen gesteuert werden, wie schnell sie nachwachsen. Weitere Parameter begrenzen die Anzahl der Früchte, die die Nahrungsquelle tragen kann, und die Schwankungsbreite der Nachwachsgeschwindigkeit.

Im Beispiel (Abbildung 3.2) könnten z.B. bei S_1 und S_2 unterschiedliche Schwankungsbreiten der Nachwachsgeschwindigkeit angegeben sein, wodurch eine ungleichmässige Wahrscheinlichkeitsverteilung für die Nachwachsgeschwindigkeit der Nahrungsquellen erreicht werden würde.

[1] noch geplant: einstellbare Gedächtnisabschnittsüberlappung

Die Typen AI und Stein erhalten eine Sonderbehandlung. Am Rand der Welt werden immer Steine gesetzt. Da sie eine eventuell vorhandene Schablone nicht auswerten, erübrigt sich die Frage, welche Schablone für Randsteine benutzt werden würde, wenn mehrere Schablonen für Steine angegeben sind. Über einen boolschen Parameter kann verhindert werden, daß mehr als eine AI in die Welt gesetzt wird. In obigen Beispiel würden, nachdem eine Zelle mit einer AI belegt wurde, die Zeilen vier und fünf gelöscht und die übrigen Wahrscheinlichkeiten entsprechend erhöht werden.

3.2.2 Abhängige Meßgrößen

Im Interesse unserer Messungen steht der Erfolg der Strategie, nach der die AI ihre Handlungen auswählt. Eine geeignete Meßgröße wäre die Anzahl der Handlungen, die mit der jeweils optimalen Handlung übereinstimmen. Die optimale Handlung kann mit Kenntnis der Gesetzmässigkeiten und des Zustands der Welt berechnet werden. Dieser Ansatz ist hier aber nicht anwendbar, da der Rechenaufwand bereits bei etwa zehn Schritten zum Erreichen des Ziels mit je vier Handlungsmöglichkeiten zu groß wird. (Mit diesen Zahlen sind schon eine Million Wege zu untersuchen.)

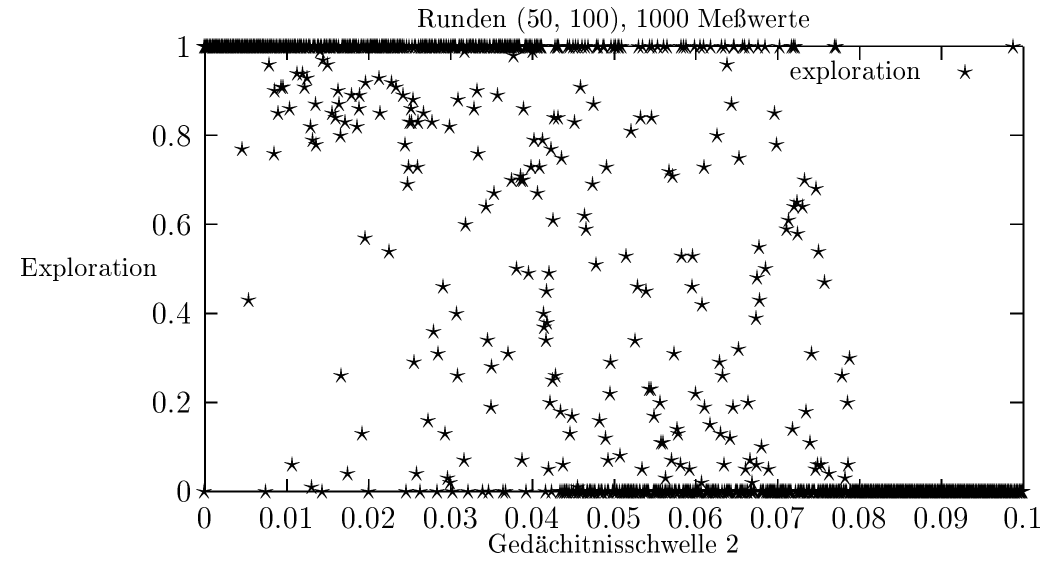
Es gibt aber Meßgrößen, die leicht ermittelt werden können: Bei der Handlungswahl kann protokolliert werden, wie viele Handlungen dem Gedächtnis entnommen werden und wie oft die Entscheidung zufällig getroffen wird. Im ersten Fall spricht man von Exploitation, im letzteren von Exploration. Ist die Explorationsrate groß oder sogar 1, dann unterhält unsere AI vergeblich ein Gedächtnis, da die Handlungen der AI hauptsächlich vom Zufall bestimmt sind.

Die Zähler für Hunger und Schmerz bieten weitere Messmöglichkeiten. Bei jeder Bewegung erhöht sich der Hunger. Ißt die AI, wird der Hungerzähler um einen konstanten Wert erniedrigt. Schmerz fügt sich die AI zu, wenn sie gegen einen Stein läuft oder in einen Stein beißt. Da beide Werte besonders von den letzten Schritten der AI abhängen, bilden wir zwei weitere Meßgrößen, Hungersumme und Schmerzsumme, indem wir die jeweiligen Werte bei jedem Schritt aufsummieren. Alle vier Werte werden so normiert, daß sie beim dauernden, ungehinderten Geradeauslauf 1 sind.

Bis auf Hunger und Schmerz sind unsere Meßgrößen davon abhängig, welche Schritte der AI in die Messung einbezogen werden. Unser Meßprogramm erlaubt es, die AI einige Runden ohne Messung laufen zu lassen. Anschliessend erfolgt die Messung. Insgesammt läuft die AI $Vorabrunden + Messrunden$ Runden lang. In den Graphen sind diese Werte als Paar $(Vorabrunden, Messrunden)$ angegeben.

3.2.3 Gedächtnisschwelle 2

Sehr schnell stellten wir fest, daß unsere AI entweder nur zufällig handelt oder nur das Gedächtnis intensiv benutzt. Da das zweite Verhalten sofort nach der ersten zwangsläufig zufällig zu wählenden Handlung einsetzt, wird immer die gleiche Handlung ausgeführt. Die AI dreht sich dann im Kreis, rennt dauernd gegen einen Stein oder ähnliches.



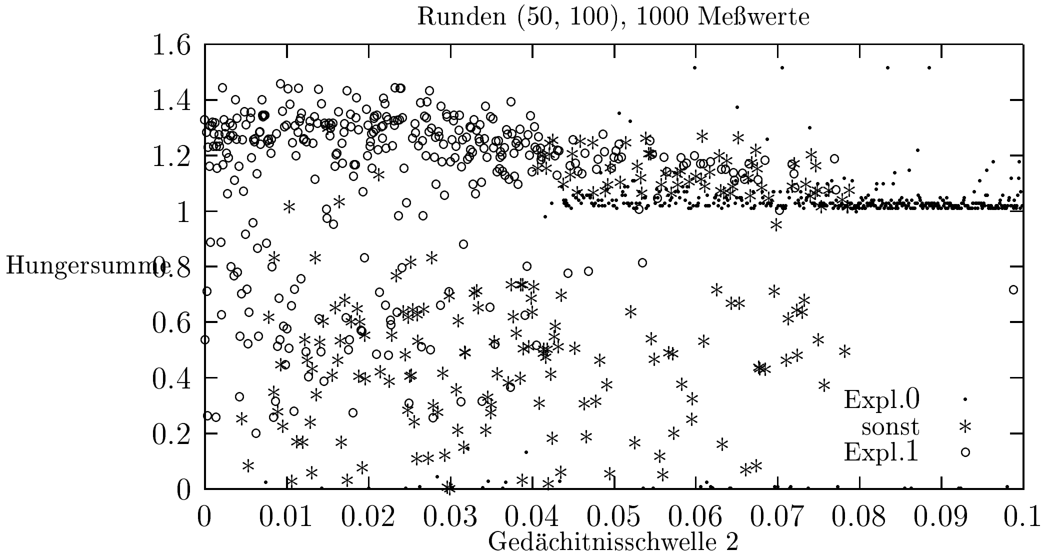


Abbildung 3.3: Explorationsrate und Hungersumme in Abhängigkeit von der Gedächtsnisschwelle 2

Betrachtet man Abschnitt 2.3.5, so wird klar, daß die Gedächtnisschwelle 2 von entscheidender Bedeutung ist. Sie bestimmt, wie stark ein im Gedächtnis gespeichertes Zeitscheibenpaar mit dem aus der aktuellen Situation gebildeten Paar übereinstimmen muß, damit es zur Handlungswahl herangezogen werden kann. Ziel des Versuchs ist, ein Wert zu finden, bei dem die meisten Handlungen gedächtnisbasiert sind aber bei dem die AI auch gelegendlich die Welt exploriert.

In der Abbildung 3.3 oben[2] ist die Explorationsrate abgetragen, die angibt, wie viele Handlungen zufällig gewählt werden. Sie ist bei einem kleinen Wert für die Gedächtnisschwelle 2 meistens 1. Die AI wählt dann in vielen Simulationsläufen alle Handlungen zufällig aus. Ist dagegen die Gedächtnisschwelle 2 groß, werden während der Meßrunden alle Handlungen dem Gedächtnis entnommen. Wir können daraus aber nicht schließ, daß immer die gleiche Handlung ausgeführt wird, da wir für die Vorabrunden keine Messungen vorgenommen haben.

Die starke Streuung der Werte kommt vermutlich auch dadurch zustande, daß für jede Messung eine neue Welt zufallsbasiert generiert wird. Dies soll verhindern, daß die Meßergebnisse nur für eine Welt gültig sind. Einzelne, herausrangende Werte könnten hierauf zurückzuführen sein. Man beachte auch, daß von den 1000 Messungen nur 209 eine Explorationsrate ungleich null und eins ergaben.[3]

Die Abbildung 3.3 unten zeigt für den gleichen Meßvorgang die jeweilge Hungersumme. Sie variiert für kleine Werte von Gedächtnisschwelle 2 stark. Für große Werte dagegen ist sie fast konstant 1. Es gibt mehrere Häufungen, die scheinbar von der Explorationsrate abhängen, die mit eingezeichnet wurde. Ohne Exploration ("$\cdot$ ", 433 Werte) ist die Hungersumme meistens 1 oder nur wenig größer. Die wenigen Ausreißer sind aber ungleich verteilt. Mit großer Gedächtnisschwelle 2 treten sowohl Hungersummen deutlich über 1 als auch dicht bei 0 auf. Ist die Gedächtnisschwelle 2 klein, so treten mit Explorationsrate gleich 0 nur noch kleine Hungersummen auf. Große Werte kommen nicht vor. Allerdings muß beachtet werden, daß das Auftreten des Falls Explorationsrate gleich 0 von der Gedächtnisschwelle 2 abhängt. Es wäre also gewagt, zu folgern, daß wenn soviel gelernt wurde, daß keine Exploration mehr von der AI für nötig gehalten wird, mit hoher Gedächtnisschwelle 2 häufig das Gelernte keinen Erfolg (Hungersumme klein) bringt, während dies mit niedriger Gedächtnisschwelle 2 immer eintritt.

Ist die Explorationsrate größer 0 aber kleiner 1 ("$*$", 209 Werte), so nimmt die Hungersumme meistens Werte zwischen 0.0 und 0.8 an. Dies ist nur möglich, wenn die AI gelegentlich von den Nahrungsquellen ißt. Bei großer Gedächtnisschwelle 2 tritt eine weitere Häufung um 1.2 herum auf. Dies ist geringfügig weniger als mit andauernder Exploration. Vermutlich verringert das Gelernte die Wahrscheinlichkeit, gegen eine Wand zu laufen oder Steine anstatt Nahrung zu essen.

Für Exploration gleich 1 ("$\circ$", 358 Werte) liegen viele Hungersummen zwischen 1.2 und 1.4. Bis hinunter zu 0.2 sind weitere Werte gleichmäßig verteilt.

[2]Das Kreuz bei $(0,0)$ ist ein Verarbeitungsfehler.

[3]noch zu untersuchen: typische Graphen für feste Welten.

3.2.4 Fläche der Welt

Diese Messung dient im wesentlichen als Kontrollmessung. Sie soll zeigen, ob es ausreicht, die anderen Messungen in kleinen Welten (28×11 Zellen) durchzuführen. Gleichzeitig würden sich andere Abhängigkeiten, z.B. von dem zeitlichen Ablauf des Versuchs, zeigen.

In der Abbildung 3.4 oben ist dargestellt, wie sich die Explorationsrate in Abhängigkeit von der Fläche der Welt verhält. Bei diesen Messungen ist die Gedächtnisschwelle 2 konstant 0.05 und das Seitenverhältnis der Welt etwa 28:11. Eine Abhängigkeit ist nicht zu erkennen. Die Verteilung entspricht der von Abbildung 3.3 oben beim Wert 0.05. Die scheinbare Häufung im linken Bereich auf die Werte 0 und 1 liegt daran, daß im rechten Bereich viele Werte übereinander liegen, da die Fläche unter Einhaltung des Seitenverhältnisses nicht beliebig gewählt werden kann: Im Versuch lief die Soll-Fläche A in gleich großen Schritten von 25 bis 4950. Mit dem Seitenverhältnis 28:11 berechnen sich dann die ganzzahlige Breite b und die Höhe h zu

$$b = \left\lfloor \sqrt{\frac{28}{11}A} + 0.5 \right\rfloor, \qquad h = \left\lfloor \sqrt{\frac{11}{28}A} + 0.5 \right\rfloor.$$

In der Abbildung eingezeichnet wurde die tatsächliche Fläche $A' = bh$, deren maximale relative Abweichung von A für große A annähernd proportional zu $\sqrt{A}$ ist. ($A' - A = bh - (b+0.5)(h+0.5) = 0.5(b+h) + 0.25 \approx c\sqrt{A}$, wobei $c = 0.5\left(\sqrt{\frac{28}{11}} + \sqrt{\frac{11}{28}}\right)$) [4]

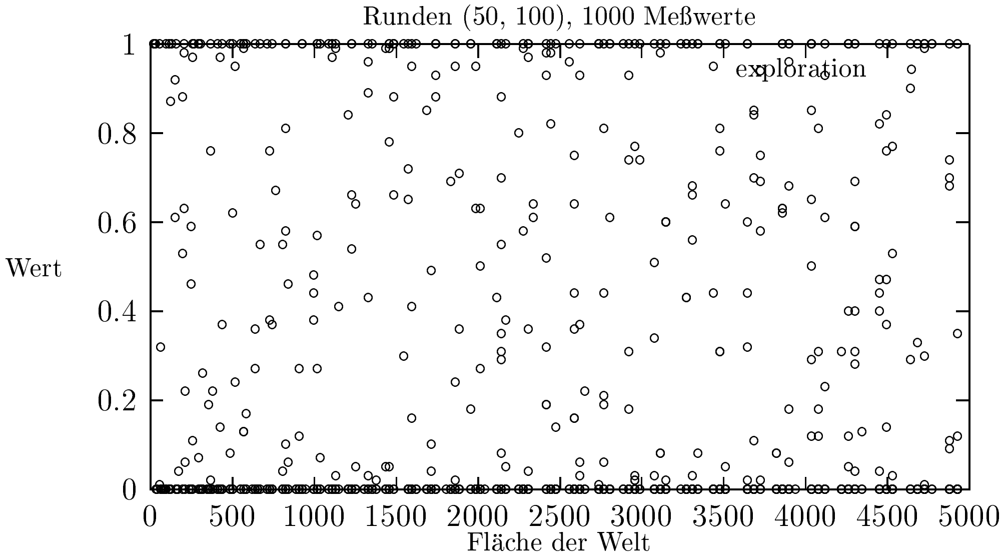
Fläche	0.0	$]0,1[$	1.0
$[0, 1000[$	58%	25%	17%
$[1000, 2000[$	65%	22%	13%
$[2000, 3000[$	60%	28%	12%
$[3000, 4000[$	66%	25%	9%
$[4000, 5000[$	62%	25%	13%

Tabelle 3.1: Häufigkeit der Explorationsraten

Die Tabelle 3.1 bestätigt, daß die Fläche der Welt keinen Einfluß auf die Explorationrate hat. Da auf jede Tabellenzeile nur durchschnittlich 200 Meßwerte kommen, lassen sich die vorhandenen Abweichungen mit den entsprechenden Binomialverteilungen erklären.

Gleiches gilt für die Abbildung 3.4 unten. Daß nur bestimmte Flächen gewählt wurden, kann man gut in dem vertikal gestreuten Bereich erkennen.

[4] Es wurden auch dann unterschiedliche Welten erzeugt, wenn die Fläche auf gleiche Werte A' gerundet wurde. Dazu wurde der Weltname (= Startwert des Zufallszahlengenerators) aus Namensprefix, Fläche und einer fortlaufenden Nummer zusammengesetzt.



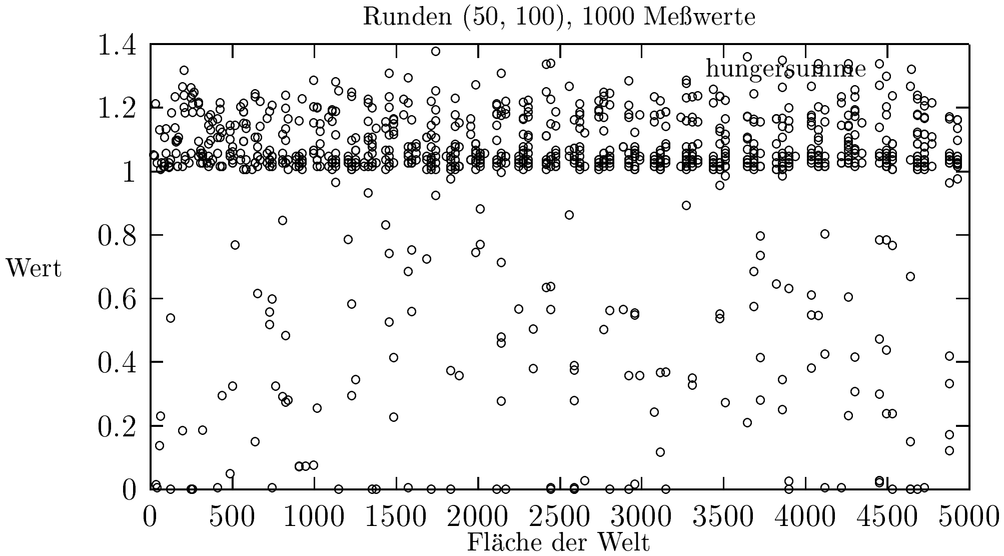


Abbildung 3.4: Exploration und Hungersumme in Abhängigkeit von der Fläche der Welt

3.2.5 Orientierung an Steinen

In den Versuchen 3.2.3 und 3.2.4 waren 85% der Zellen leer und nur 10% Steine vorhanden. Da unsere AI nur vier Zellen weit "sehen" kann, muß die Orientierung in der Welt sehr schwer sein. Mit 18,5% Wahrscheinlichkeit befindet sich kein Stein in Sichtweite.[5]

In der Abbildung 3.5 sind die Meßwerte aus einer Wiederholung des Versuchs 3.2.3 mit 20% Steinen dargestellt. Obwohl hier nur 250 Messungen gemacht wurden, kann man erkennen, daß sich die Graphen lediglich nach rechts verschoben haben. Dies deutet darauf hin, daß die durchschnittliche Distanz zwischen den Zeitscheibenpaaren größer geworden ist.[6]

3.2.6 Nachwachsende Nahrungsquellen

Für die bisherigen Versuche wurden Welten benutzt, die nur unerschöpfliche Nahrungsquellen enthalten können. Die AI kann ihre Hungersumme also niedrig halten, indem sie bei einer Nahrungsquelle verharrt und fortlaufend von ihr ißt. Ein weitere Versuch soll klären, wie sich das Verhalten der AI verändert, wenn man ihr diese Möglichkeit entzieht. Dazu bekommen alle Nahrungsquellen jeweils einen Energiezähler. Wird ein variables Energieniveau erreicht, kann eine weitere Frucht von der Nahrungsquelle gebildet werden. i Falls genügend Früchte vorhanden sind, verfällt ein Teil der Energie.

Die Abbildung zu diesen Versuch lassen wir weg, da sich die Unterschiede leicht mit Worten beschreiben lassen. Die Explorationsrate verhält sich im wesentlichen wie vorher, nur endet der Bereich, in dem die Exploration größtenteils gleich null ist, fast abrupt. Werte mit hungersumme gleich null oder unter 0.2 treten nicht mehr auf. Die Verteilung der restlichen Werte ist aber vergleichbar mit denen der vorangegangenen Versuche.

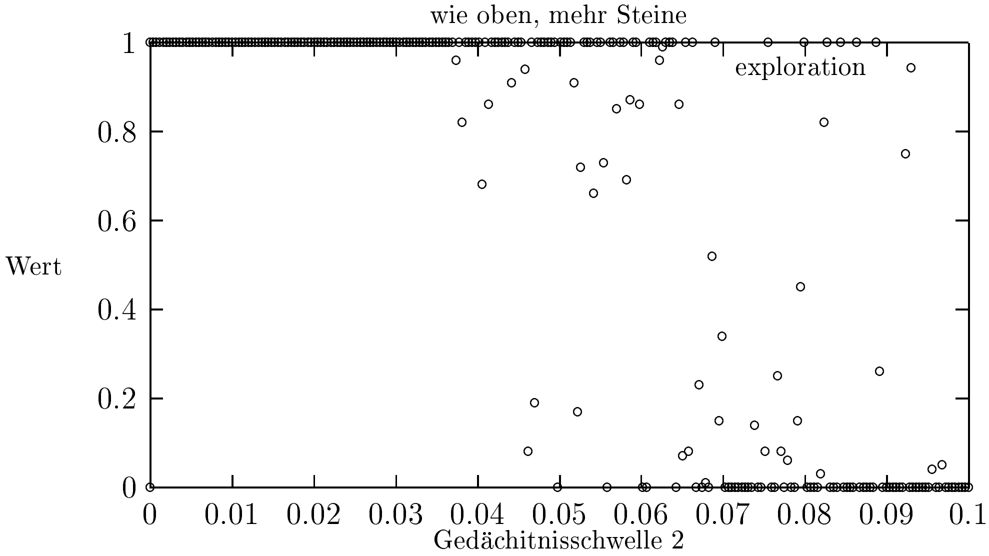
Da wie beim Versuch mit mehr Steinen nur 250 Meßwerte ermittelt wurden und nur bei einem geringen Teil die Explorationrate ungleich null und eins ist, läßt sich nicht mehr erkennen.

3.2.7 Schlußfolgerungen

Die Versuche zeigen, daß es keinen Wert für die Gedächtnischwelle 2 gibt, der für alle Welten ein einheitliches Verhalten im Bezug auf die Gedächtnisnutzung garantiert. Meistens wählt unsere AI nach den 50 Vorabrunden einer der Totalen dauerende Exploration und Exploitation. (Eine mögliche Lösung wäre die automatische Anpassung der Gedächtnisschwelle 2.)

[5]Die Wahrscheinlichkeit für genau i Steine ist $\frac{16!}{i!(16-i)!}0.1^i(1-0.1)^{16-i}$.

[6]Mit ziel.sd[] = { Nichts, Stein, Nichts, Stein, * } kann dies nicht erklärt werden.



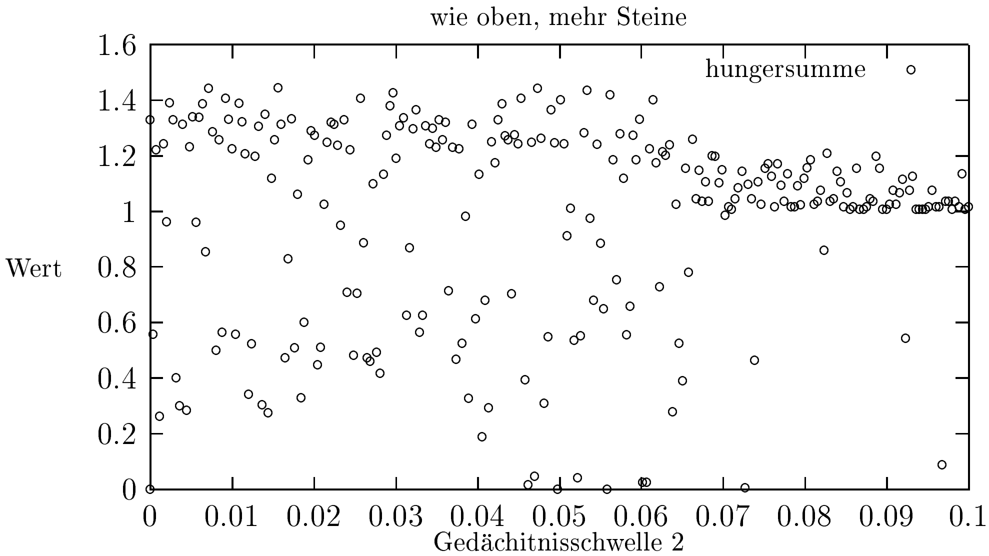


Abbildung 3.5: Orientierung an Steinen

3.3 Parameteroptimierung

In den bisherigen Versuchen haben wir die meisten Parameter im Quellcode der AI fest vorgegeben. Während der Implemtentation der AI trat an vielen Stellen die Notwendigkeit auf, Parameter festzulegen, die in der Planung nicht vorgesehen bzw. bedacht wurden, z.B. die Gewichtung von Glattheit, Größ und Farbe. Wie haben dann Werte eingesetzt, die uns sinnvoll erschienen. Dieses etwas willkürliches Vorgehen wollen nun durch eine gezielte Suche nach optimalen Parametern ersetzen.

3.3.1 Motivation

Die Gedächtnisschwelle 2 bestimmt nur indirekt, ob ein Zeitscheibenpaar zur Handlungswahl herangezogen wird. Die Distanzfunktion d spielt dabei wie in Abschnitt 2.3.5 beschrieben eine wesentliche Rolle. Es ist durchaus möglich, daß ihre Eigenschaften die die geeignete Wahl der Gedächtnisschwelle 2 behindern, wie folgende Vereinfachung zeigt.

Nehmen wir an, die Zeitscheiben ließen sich durch Punkte in der Ebene angemessen beschreiben. Weiter sei vernachlässigt, daß nicht nur Start- sondern auch Zielzeitscheiben und der zeitliche Abstand zwischen Start und Ziel berücksichtigt werden. Dann definiert die Gedächtnisschwelle 2 um die im Gedächtnis liegenden Punkte Umgebungen, die mit euklidischer Distanzfunktion d kreisförmig sind. Die prozentuale Überdeckung mit diesen Kreisen bestimmt die Explorationsrate, da in den nicht bedeckten Gebieten kein Punkt aus dem Gedächtnis weniger als Gedächtnischwelle 2 entfernt ist. Wenn die Explorationsrate also stark für verschiedene Welten variiert, dann muß dies auch für den Überdeckungsgrad gelten. In erster Linie wird vermutlich der Überdeckungsgrad von der unterschiedlichen Verteilung der Punkte in der Ebene gestreut, die von den Eigenschaften der jeweiligen Welt abhängt. Aber auch die Distanzfunktion kann eine Rolle spielen. In der Vereinfachung auf Punkte in der Ebene könnte z.B., wenn die verschiedenen Welten die Punkte nur in eine Richtung strecken, eine Distanzfunktion, die diese Richtung weniger stark gewichtet, die Streuung eliminieren.

3.3.2 Parameter unserer AI

Die Distanz zweier Zeitscheiben wird aus vier unterschiedlich gewichteten Einzelwerten, die aus den visuellen[7] und auditorischen Werten und Hunger und Schmerz hervorgehen, berechnet. Weiter gibt es wie die in 2.3.5 beschrieben Gewichte für Start- und Zielzeitscheiben und für die Zeitspanne zwischen ihnen.

Wir haben uns auf diese Parameter beschränkt, obwohl im Quellcode noch viele Parameter der AI zu finden sind, die fest vorgegeben sind. Der Aufwand, die entsprechenden Stellen zu lokalisieren und anzupassen, so daß sie zur Laufzeit des Programms eingestellt werden können, ist recht hoch.

[7] Idee: jede visuellen Zelle getrennt gewichten

3.3.3 Genetische Optimierung

Bei näherer Betrachtung ist unserer Optimierungsaufgabe ein sehr schwieriges Problem. Optimiert werden soll der Erfolg der AI in allen Welten in Abhängigkeit von einer großen Anzahl von Parametern, d.h. gesucht ist ein Parametersatz x, so daß

$$f(x) = \frac{1}{|W|} \sum_{w \in W} f_s(x, w)$$

maximal wird, wobei W die (endliche) Menge aller Welten und $f_s(x, w)$ der Erfolg der AI in der Welt w mit Parametern x ist. Die Funktion f kann nicht einfach analysiert, z.B. abgeleitet werden. Zwar kann man hoffen, daß die starke Streuung der Werte von f_s, die man in den bisherigen Versuchen sehen kann, durch die Mittelung über alle Welten aufgehoben wird. Der Zusammenhang zwischen Parametern x und Erfolg $f(x)$ bleibt aber völlig unklar. Wir sehen keine andere Möglichkeit, als $f(x)$ durch Berechnung von Einzelwerten $f_s(x, w)$ zu analysieren. Dies erfordert für jedes zu untersuchende x und w einen kompletten Simulationslauf unserer AI, der sehr rechenintensiv und damit langwierig ist.

Wie brauchen also ein Optimierungsverfahren, daß keine Ableitung zu $f(x)$ benötigt und sehr gut mit folgender spezieller Form der Näherung von $f(x)$ zurecht kommt, die wir zur Verfügung stellen können. Für sehr kleine Teilmengen W' von W läßt sich $f_s(x, w)$ für alle $w \in W'$ noch in annehmbarer Zeit berechnen. Bei jeder erforderlichen Näherung von $f(x)$ kann leicht eine andere, zufällig gewählte Teilmenge von W benutzt werden, so daß die Abweichung auf lange Sicht nicht einseitig ausfallen.

Eine Suche mit der Monte-Carlo-Methode scheidet aus, da sie unter diesen Bedingungen sehr lange dauern würde. Die Eigenschaften der Welt haben wesentlichen Einfluß auf den Erfolg der AI. Ergab eine Näherung zu $f(x)$ einen großen Wert, so wird es sehr unwahrscheinlich, einen neuen Kandidaten x' zu finden, der x überbietet, selbst wenn es viele tatsächlich bessere Kandidaten gibt. Eine geeignete Gegenmaßnahme wäre, die Teilmenge der Welten für die Näherung so groß zu wählen, daß die Streuung beschränkt wird. Dieser Weg ist aber die rechenintensiv. Eine andere Lösung des Problems könnte sein, für jeden Vergleich eine neue Näherung zu $f(x)$ basierend auf einer neuen Teilmenge von W heranziehen. Dies bewirkt aber, daß andere Parameter x' den aktuellen Parameter x leicht überbieten können, indem sie zufällig in günstigeren Welten geprüft werden. Dies Verfahren kommt also hier nicht in Frage.

Die sogenannte genetische Optimierung scheint uns besser geeignet zu sein. Dies Verfahren ist an den Behauptungen der Evolutionstheorie angelehnt. Entsprechend werden bei der Beschreibung dieses Verfahrens Begriffe von dort übernommen, z.B. heißt dort ein Parametersatz x ein Individuum. Wie die meisten Verfahren ist die genetische Optimierung ein iteratives. Gestartet wird mit einer initialen Population $P_1 = \{x_1^1, \ldots, x_n^1\}$. Im k-ten Schritt wird für jedes Individuum x_i^k $(i = 1, \ldots, n)$ eine Fitness (Bewertung) $b(x_i^k)$ bestimmt, aufgrund derer entschieden wird, welche Individuen einen Beitrag zur neuen Population P_{k+1} leisten. (Siehe dazu auch [5] Seite 250 - 252 und den Quellcode des Programs.[8]

[8]Von dem in [5] angegebenen Algorithmus weichen wir darin ab, daß nicht eine feste Anzahl von

Im Idealfall sollte $b = f$ gelten. Interessant für unser Vorhaben ist die Frage, wie stark oben beschriebene Näherung das Verfahren verschlechtert. Nehmen wir an, in der Population P_k haben m Individuen eine überdurchschnittlich gute, tatsächliche Fitness $f(x_{i_j})$ $(j = 1, \ldots, m)$. Ein einzelnes Individuum x_{i_j} $(j = 1, \ldots, m)$ kann zwar durch die Streuung der Näherung von einem Individuum x_{i_j} $(j = m+1, \ldots, n)$ überboten werden. Im statistischen Mittel werden aber mehr tatsächlich fittere Individuen besser bewertet als tatsächlich weniger fitte. Wenn die Population groß genug ist, dann ist dies auch in der Praxis mit hoher Wahrscheinlichkeit der Fall.

Mit den Bildern der genetischen Optimierung gesprochen bedeutet unser Vorgehen, daß wir nicht eine allein vom Individuum abhängende Fitness verwenden sondern daß die Fitness nur in einer konkret gegebenen Welt ermittelt werden kann. Dabei sind die Bedingungen in der Welt für jedes Individuum verschieden. *** Messreihen ***

3.3.4 Messreihen

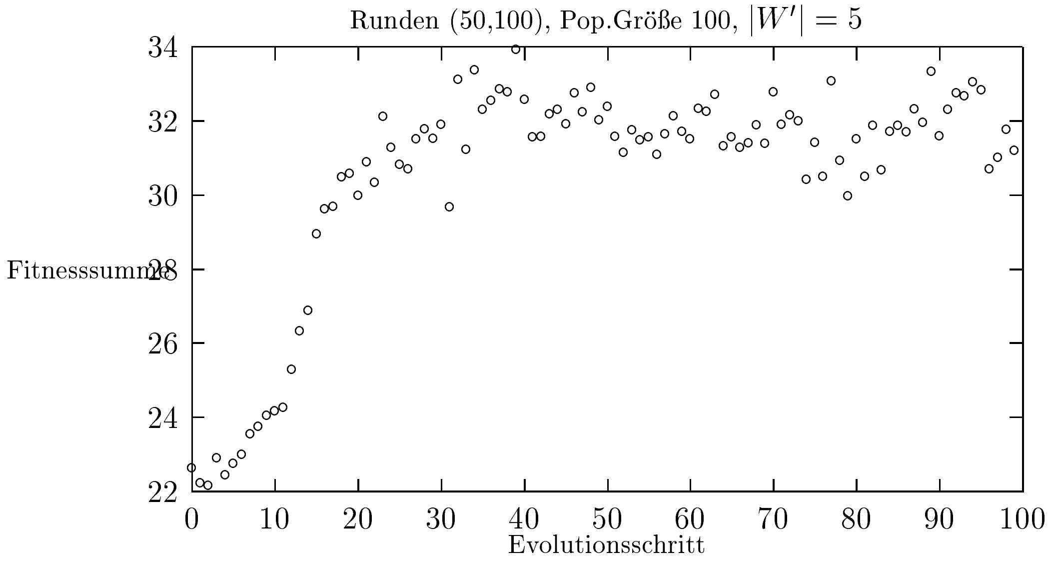
Die Startpopulation P_1 haben wir zu 80% zufällig initialisiert, 10% sind Individuen wie in den bisherigen Versuchen und weitere 10% sind zufallsbasiert 0-1-wertig initialisiert. Einige Flaggen steuern, ob und welche Werte normiert werden. Diese sind auß bei den default-Individuen ebenfalls zufällig gewählt. Als Fittness $f_s(x, w)$ haben wir den Kehrwert aus der Summe von Hungersumme und Schmerzsumme, wobei die Hungersumme doppelt gewichtet wird, definiert. Zusätzlich wird auf den Nenner ein Strafwert in Höhe von 2.0 aufaddiert, wenn die Explorationrate nicht zwischen 0.01 und 0.99 liegt.[9]

Abbildung 3.6 zeigt einen Optimierungslauf mit einer Population von 100 Individuen. Es wurden zur Bewertung der Fitness jeweils fünf Welten zufällig ausgewählt. Die Fitness steigt in den ersten 35 Schritten von durchschnittlich 2.2 auf 3.2. In der Abbildung unten sieht man, daß die Streuung innerhalb der Population stark zunimmt. (Damit die Abbildung nicht zu voll wird, wurden je Population nur 50 zufällig selektierte Werte dargestellt und der Bereich von P_40 bis P_69 weiß gelassen.) Bei den ersten zehn Populationen liegt die Fitness der meisten Individuen zwischen 0.2 und 0.25. Ab etwa P_20 herschen Werte zwischen 0.22 und 0.42 vor.

Wird für die Bestimmung der Fitness nur eine Welt benutzt, kann auf Explorationsrate, Hungersumme und Schmerzsumme geschlossen werden. In Abbildung 3.7 erkennt man mehrere Häufungen. Werte unter $\frac{1}{1.4*3} \approx 0.24$ deuten auf eine Explorationrate dicht bei 0 oder 1 hin. Solche Werte treten in allen Populationen auf. Bei Werten über $\frac{1}{2} = 0.50$ muß die Explorationsrate im Intervall $[0.01, 0.99]$ liegen. Ab etwa der zwanzigsten Population erreicht eine größere Anzahl von Individuen diese Fitness. Mit etwa der sechszehnten Population setzt es eine weitere Häufung um 0.40 herum ein.

Individuen für Select und Crossover verwendet wird, sondern für jedes Individuum der Population P_{k+1} zufällig bestimmt wird, wie viele Individuen der Population P_k Einfluß haben.

[9]Idee: stetige Straffunktion $s(Expl.Rate) = 2(2Expl.Rate - 1)^{12}$ verwenden.



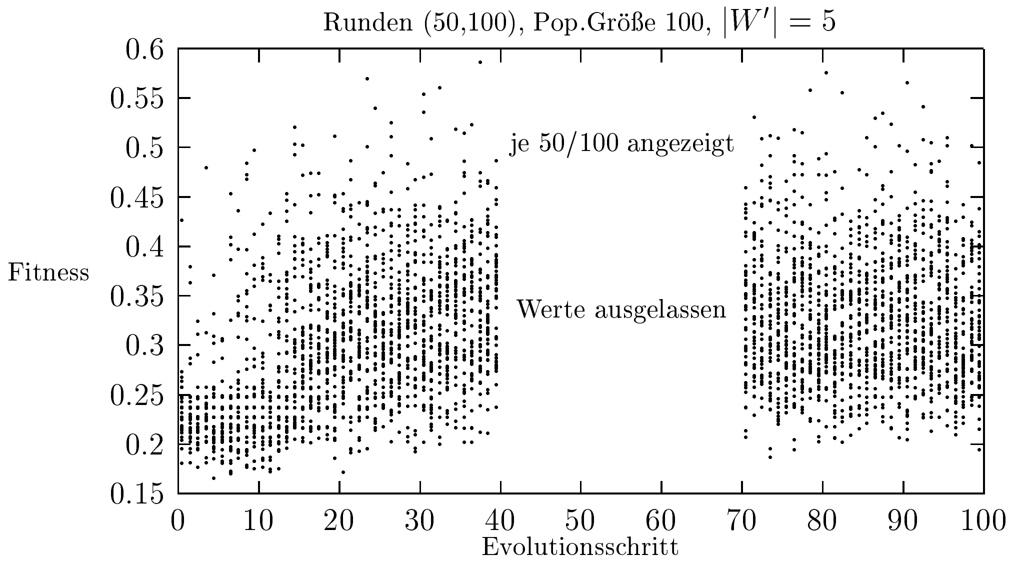
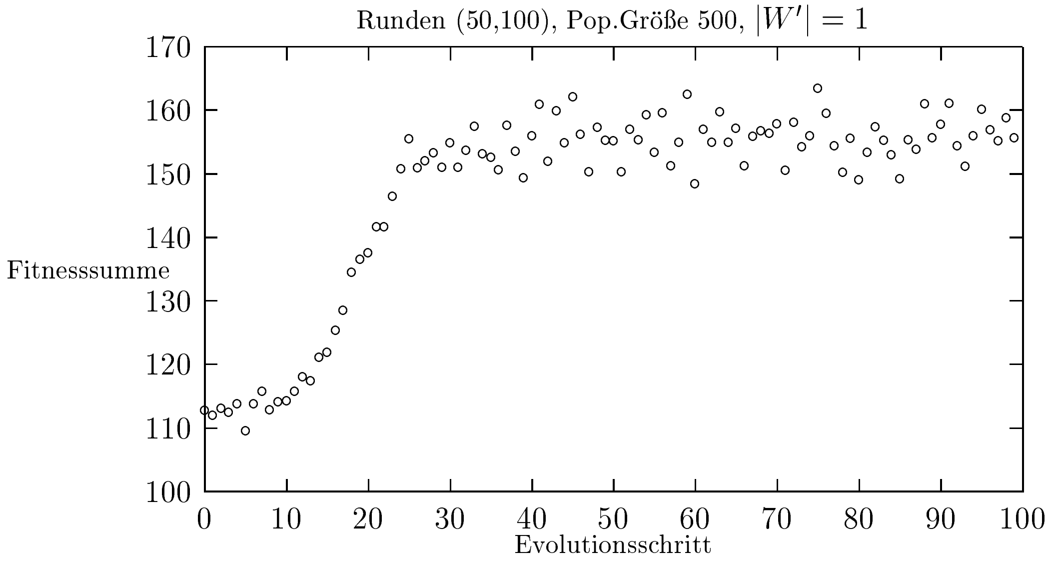


Abbildung 3.6: Genetische Optimierung der AI-Parameter (I)



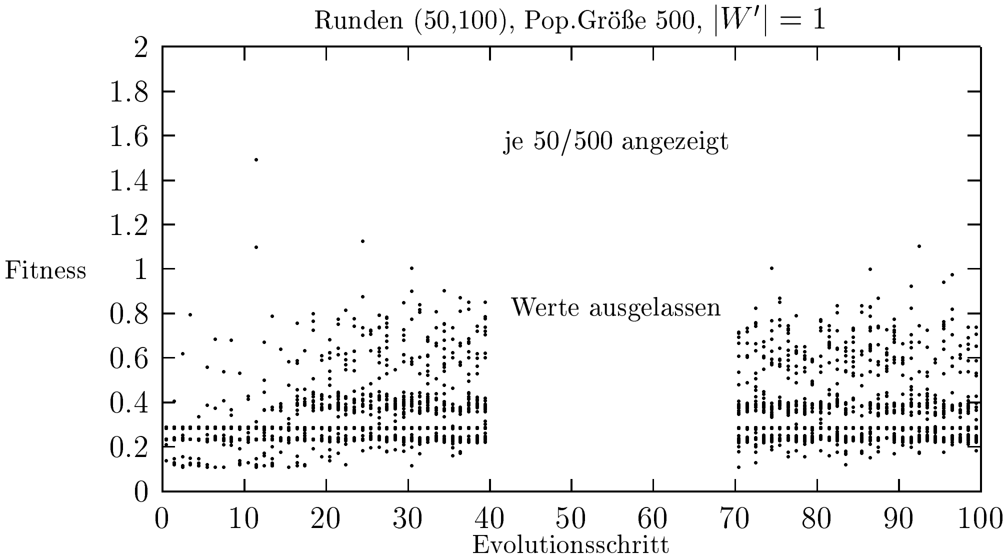


Abbildung 3.7: Genetische Optimierung der AI-Parameter (II)

3.3.5 Schlußfolgerungen

Die genetische Optimierung hat die durchnittliche Fitness erhöht. Ein ganze Population von Parametersätzen, die die Hungersumme und Schmerzsumme einer AI in vielen Welten klein halten können und gleichzeitig die Explorationsrate unseren Vorstellungen entsprechen lassen, ist aber nicht entstanden. Daß einzelne Idividuen erzeugt wurden, die dies leisten, müßte noch untersucht werden.[10]

3.4 Fazit

Unsere AI ist nur in wenigen Welten erfolgreich. Die Explorationsrate schlägt häufig gegen 0 oder 1. Die Parameterwahl hat sich als großes Problem herausgestellt.

[10]Idee: Graph zur Ergebnispopulation erstellen: X-Achse Individuumnummer, Y-Achse Fitness, je Individuum 50 mal Fitness mit $|W'| = 1$ bestimmen.

Anhang A

Programme

Wir haben die Programme in der Sprache C implementiert. Gleichzeitig haben wir bei der Gestaltung der Module und Schnittstellen versucht, die Ideen der objektorientierten Programmierung umzusetzen. Dadurch sollen unsere Programme sowohl leicht pfleg- und erweiterbar als auch auf vielen Systemen zum Laufen zu bringen sein.

A.1 Module

Service stellt einige allgemeine, projektunabhängige Funktionen zur Verfügung. Das Modul enthält einen Pseudozufallszahlengenerator, dessen Zustand gespeichert werden kann und der auf jedem System die gleiche Folge von Zufallszahlen liefert, eine Hashtabelle, die das effiziente Ablegen und Auffinden von Informationen erleichtert, und eine Speicherverwaltung, die unseren objektorientierten Ansatz unterstützt.

Im AI-Modul wird das Verhalten der AI definiert. Die zentrale Funktion ist dabei ailaufen. Sie wird in jeder Runde der Welt für jede AI einmal aufgerufen. Man beachte, daß zur besseren Übersicht alle Funktionen, die die Ein-/Ausgabe betreffen, in eine gesonderte Quelldatei gestellt wurden.

Das Welt-Modul steuert den Ablauf in der simulierten Welt. Es verwaltet den Zustand der Welt und berechnet die Auswirkungen von Handlungen.

Das Sensorik-Modul vermittelt zwischen Welt- und AI-Modul. Es erlaubt der AI begrenzte Einblicke in den Zustand der Welt. Dabei ändert das Modul die Repräsentation und reduziert die Daten.

Im Sonstige-Modul sind alle weiteren Module zusammengefaßt, wie z.B. für Steine, Nahrungsquellen und leere Felder.

Verschiedene Hauptprogramme steuern die Durchführung der Experimente. Abhängig von Kommandozeilenparametern erzeugen sie Welten und protokollieren das Geschehen.

A.2 Algorithmen

Im Quellcode finden sich neben den Abschnitten, die aus dem Blickwinkel der AI interessant sind, viele lange Codestücke, die z.B. der Verwaltung der Datenstrukturen dienen. Im folgenden werden einige der dort auftretenden Algorithmen erläutert, damit der Leser sich im Quellcode leichter zurechtfindet.

Der Pseudozufallszahlengenerator (Service-Modul) setzt zwei Verfahren ein. Der erste Teil ist eine Kombination aus zwei sogenannten linear-kongruenten Generatoren (LCGs). Diese Generatoren zeigen unter bestimmten Bedingungen laut [8] ein gutes statistisches Verhalten. Die Folge von Zufallszahlen eines LCGs berechnet sich mit

$$x_i = (ax_{i-1} + b) \bmod m$$

und ist abhängig von den Konstanten a, b und m sowie vom Startwert x_0. Offensichtlich ist die so definierte Folge periodisch und wiederholt sich nach spätestens m Folgengliedern. Hier werden zwei LCGs mit Perioden benutzt, die keine gemeinsamen Teiler haben. Die Gesamtperiode ist dann das Produkt der Einzelperioden. In unserem Programm, das mit 32 Bit arbeitet, erhöht sie sich damit von ca. 10^9 auf ca. 10^{18}. Dieser Generator hat aber den Nachteil, daß nicht jedes Bit mit gleicher Wahrscheinlichkeit 0 oder 1 wird. Dies liegt daran, daß die m-Werte der LCGs die erzeugbaren Zufallszahlen einschränken. Daher verwenden wir zusätzlich einen additiven Generator. Die Definition der Zufallszahlenfolge ist eine Verallgemeinerung der Fibonacci Rekursion:

$$x_i = (x_{i-a_1} + \ldots + x_{i-a_n}) \bmod 2^m,$$

wobei a_1 bis a_n paarweise verschieden, ganzzahlig größer Null und kleiner gleich a_n und x_0 bis x_{a_n-1} vorgegeben sind. Für die gewohnte Fibonacci-Folge ist $n = 2$, $a_1 = 1$ und $a_2 = 2$. (Natürlich ohne modulo-Rechnung. Man beachte aber, daß dies für die unteren Bits der Folge keinen Unterschied macht, da bei der Addition die höheren Bits keine Einfluß auf niedriegere Bits haben.) Unter sehr speziellen Bedingungen ist die Periode $2^m - 1$ oder ein Vielfaches. [7] diskutiert diesen und andere Generatoren. Der Einsatz von zwei unabhängigen Verfahren hat zusätzlich den Vorteil, daß Implementationsfehler in einem der Generatoren die statistischen Eigenschaften der Zufallszahlen möglichst wenig beeinträchtigen. Wie leicht kleine Fehler die Qualität der Zufallszahlen beeinflussen kann, zeigt [6].

Ebenfalls im Service-Modul befinden sich Funktionen zum Hashing. Mit Hashing können Daten mit beliebigen Suchschlüsseln in ein begrenztes Datenfeld abgelegt werden. (Siehe [9] und [10].) Zum Beispiel speichert das Welt-Modul die Positionen der AIs in einer Hashtabelle, um sie schneller auffinden zu können. Als Schlüssel dient die Speicheradresse der jeweiligen AI, die sie eindeutig identifiziert. Da die Adressen beliebig verteilt sein können, ist es nicht sinnvoll, die Adressen direkt als Index zu verwenden. Eine Hashfunktion berechnet daher aus dem Schlüssel einen vorläufigen Index. Das Hashing überprüft dann, ob der Index bereits von einen anderen vorher verwendetetn Schlüssel, der zum gleichen Index umgerechnet wurde, belegt ist. Ggf. wird dann ein neuer Index bestimmt (Sondierschritt). Schließlich steht ein neuer Index zur Verfügung, oder es ist kein Index mehr frei.

Wir haben das Speichern der Benutzerdaten nicht ins Hashing mit aufgenommen, weil dies in C effizienter typabhängig implementiert werden kann. Trotzdem unterstützt unser Hashing Einfügen, Auffinden und Löschen.

In einem der Hauptprogramme tritt das Problem auf, daß aus einer großen Menge von Objekten einige zufällig ausgewählt werden sollen, wobei jedem Objekt eine eigene Wahrscheinlichkeit zugeordnet ist. Für ein Objekt ist folgendes Vorgehen naheliegend. (Die Objekte seien durchnumeriert von 1 bis n, und die Wahrscheinlichkeiten seien $p_1, \ldots, p_n$.) Man wählt eine Zufallszahl $x \in [0,1[$. Ist $x < p_1$, so wird das Objekt 1 gewählt. Sonst wird Objekt 2 gewählt, falls $x - p_1 < p_2$. Usw. Also wird das Objekt i genau dann ausgewählt, wenn $x - \sum_{j=1}^{i-2} p_j \geq p_{i-1}$ und $x - \sum_{j=1}^{i-1} p_j < p_i$. Dies läßt sich umstellen zu

$$\sum_{j=1}^{i-1} p_j \leq x < \sum_{j=1}^{i} p_j$$

oder mit $s_i := \sum_{j=1}^{i} p_j$

$$s_{i-1} \leq x < s_i.$$

Berechnet man also einmal die s_i, so muß man nicht linear alle Objekte abtesten sondern kann eine binäre Suche verwenden, die nur eine logarithmische Laufzeit hat, also wesentlich schneller ist, falls viele Objekte vorliegen.

Einen großen Teil im Quellcode nimmt die Ein- und Ausgabe ein. Alle Objekte können ihre relevanten Datenfelder ausgeben und selbständig wieder einlesen. Mangels Sorgfalt bei der Planung wiederholen sich hier Codestücke immer wieder. Daß wir das Konzept, daß Aggregate ihre Komponenten nicht selbst speichern oder einlesen sondern dies von ihnen erledigen lassen, intensiv benutzt haben, kann man hoffentlich trotzdem erkennen.

A.3 Hauptprogramme

Das Programm ai lädt eine Welt oder erzeugt eine neue, zufällige Welt, läß sie eine einstellbare Anzahl Runden laufen und speichert den neuen Zustand wieder. Der Ablauf kann auf dem Bildschirm verfolgt werden.

messwert ermittelt Meßwerte. Nicht alle Einstellungen können auf der Kommandozeile vorgenommen werden. Z.B. wird im Quellcode festgelegt, ob die Gedächtnisschwelle 2 oder die Fläche der der Welt variiert werden soll.

paramopt setzt Verfahren der genetischen Optimierung ein, um für die Parameter der AI, die in den anderen Hauptprogrammen fest vorgegeben sind, bessere Werte zu finden. Der Optimierungsprozess kann sehr ausführlich protokolliert werden. Shell-Skripte stehen zur Verfügung, die bestimmte Informationen extrahieren und für gnuplot aufbereiten.

Das Programm rtest testet den Zufallszahlengenerator des Service-Moduls. Es werden Fließkommazahlen ($[0,1[$) erzeugt und sowohl dezimal als auch binär ausgegeben. Letztere Ausgabe erfolgt mit X für 1 und . für 0, damit Regelmäßigkeiten mit dem Auge leichter erkannt werden können.

Literaturverzeichnis

[1] David W. Aha (1997): Lazy Learning. *AI Review*, (11) Seite 7-10 (Editorial)

[2] Christopher M. Bishop: *Neural Networks for Pattern Recognition*, Oxford University Press, New York 1995 ISBN 0-19-853864-2

[3] Achim Hoffmann (1998): *Paradigms of Artificial Intelligence*, Springer-Verlag Singapur 1998 ISBN 981-308397-2

[4] Pattie Maes (1997): *Modeling Adaptive Autonomous Agents*, Beitrag in *Artificial Life* ISBN 0-262-12189-1, Seite 135 - 162

[5] Tom M. Mitchell: *Machine Learning*, WCB/McGraw-Hill 1997 ISBN 0-07-042807-7

—

Zur Implementation:

[6] Edgar H. Sibley (1988): *Random Number Generators: Good One Are Hard to Find*, Communications of the ACM, v31 n10 Seite 1192-1201

[7] Knuth: *The Art of Computer Programming*, Volume 2, Seminumerical Algorithms, zweite Ausgabe, Addison-Wesley 1981, Seite 26-33

[8] Pierre L'Ecuyer (1988): *Efficient and Portable Combined Random Number Generators*, Communications of the ACM, v31 n6 Seite 742-749

[9] Michael T. Goodrich und Roberto Tamassia (1998): *Data Structures and Algorithms in Java*, Seite 283-291

[10] Oliver Vornberger und Frank M. Thiesing (1998): *Algorithmen*, Vorlesungsskript WS'97/98, Seite 131-141