PS Einführung in die Bio-Informatik SS 1999
Projekt: Roboter-Simulation

Christian Spielberger
spielberger@vpn.at

Rainer Trummer
rtrummer@cosy.sbg.ac.at

Ferdinand Schober
fschober@cosy.sbg.ac.at

Version: 13. April 1999
Uhrzeit: 15:45 - 16:15

LV-Leiter: Schwaiger Roland, Dipl.Ing. Mag.

Zusammenfassung

Ein Roboterarm soll durch ein Artificial Neural Network gesteuert werden. Ein Genetischer Algorithmus übernimmt die Anpassung und somit die Lernphase des ANN. Nach einer gewissen Zeit ist die Lernphase beendet, und die Trainingseinheit kann abgehängt werden.

Kapitel 1

Spezifikation

1.1 Simulierte Welt

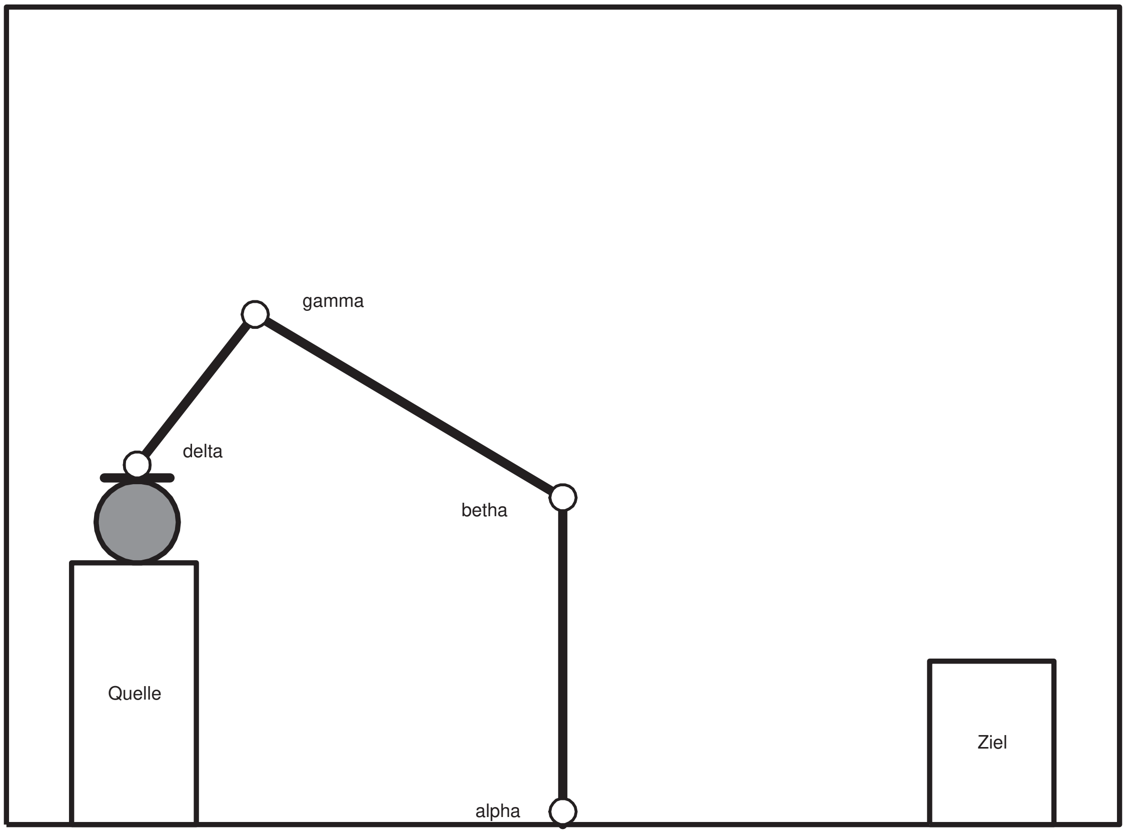

Abbildung 1.1: Simulierte Welt

Der Roboterarm hat drei Glieder und vier Gelenke. Das erste Gelenk ist am Boden befestigt, am letzten ist eine Magnetscheibe befestigt. Die Gelenke können jeden beliebigen Winkel annehmen. Auf beiden Seiten befindet sich ein Podest mit beliebiger Höhe. Auf einem der beiden Podeste liegt eine Eisenkugel.

1.1.1 Aufgabenstellung

Aufgabe für den Roboterarm ist es, die Kugel vom einen Podest auf das andere zu heben. Dazu kann er seine Magnetscheibe benutzen, um die Kugel aufzuheben. Die Aufgabe muß getrennt werden in

- Kugel holen
- Kugel von Podest A nach Podest B bewegen
- Kugel auf Podest B ablegen

Nebenbedingungen sind:

- Die Kugel sollte möglichst in der Mitte des Podests abgelegt werden.
- Der Arm bzw. die Kugel sollte nicht mit dem Podest kollidieren.
- Es soll möglichst wenig Energie verbraucht werden. Dh. der Roboterarm sollte den direktesten Weg anstreben.

1.1.2 Motorik

An jedem Gelenk befindet sich ein Motor. Jeder der Motoren hat einen Eingang, der die Werte $\Delta w = -1, 0, +1$ annehmen kann und eine entsprechende Änderung des Winkels w hervorruft.

1.1.3 Sensorik

Es gibt Sensoren die folgende Meßwerte liefern.

- Position der Kugel Kugel.x, Kugel.y
- Position der Magnetscheibe Magnet.Mx, Magnet.My
- Zielkoordinaten Ziel.x, Ziel.y
- Status der Kugel

  -1 ..... Kugel liegt irgendwo
  +1 ..... Kugel hängt am Magneten

- Winkel der Gelenke α , β , γ , δ

1.1.4 Ausgabe

Am Bildschirm erscheint das Modell, sowie es Abbildung 1.1 zeigt. Wenn sich zumindest einer der Winkel um mehr als ϵ Grad ändert, wird dieses Bild aktualisiert. Zusätzlich könnte eine Ausgabe über den Status des Genetischen Algorithmus erfolgen:

- der beste Fitneßwert
- Anzahl der Generationen
- Anzahl der Simulationsdurchläufe
- ...

1.2 Artificial Neural Network

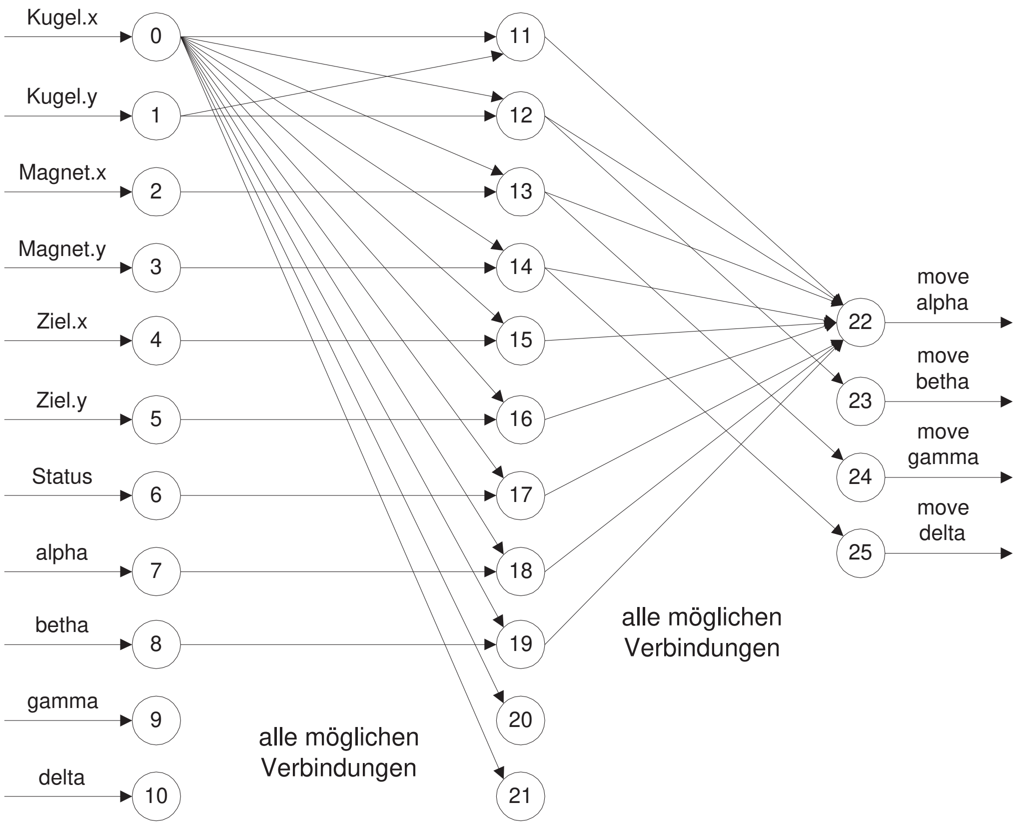

Abbildung 1.2: Artificial Neural Network

1.2.1 Neuronen

Das Neuronale Netz besteht aus dem Eingangslayer, mit elf Neuronen, dem Ausgangslayer mit vier Ausgängen und dem Hiddenlayer. Die Neuronen haben beliebig viele Eingänge, deren Signale bei der Berechnung aufaddiert werden. Sie können einen positiven bzw. negativen Sättigungswert erreichen (Standardmäßig = ± 1000).

Eingangsneuronen

Die Eingangsneuronen müssen der Sensorik wie unter Abschnitt 1.1.3 schon beschrieben entsprechen.

Kugel.x	X-Koordinate des Kugelmittelpunkts
Kugel.y	Y-Koordinate der Kugel
Magnet.x	X-Koordinate des Magnetmittelpunkts an der Unterseite
Magnet.y	Y-Koordinate der Magnetscheibe
Ziel.x	X-Koordinate zu der sich der unterste Punkt der Kugel hinbewegen soll
Ziel.Y	Y-Koordinate des Ziels
Status	Gibt an ob die Kugel geholt oder transportiert wird. (siehe Abschnitt 1.1.3)
α	aktueller Winkel des Gelenks am Boden.
β	aktueller Winkel des zweiten Gelenks.
γ	aktueller Winkel des dritten Gelenks.
δ	aktueller Winkel zwischen Magnetscheibe und letztem Arm.

Hidden-Neuronen

Die Anzahl der Neuronen im Hiddenlayer wird nach der Implementierung experimentell ermittelt oder in einer späteren Version eventuell automatisch angepaßt. So könnte man ein Wachstum des Neuronalen Netzes bzw. eine minimale Größe ermitteln, indem man verschiedene Versuche mit unterschiedlicher Neuronenanzahl in der Hiddenschicht anstellt. Eventuell könnte man diese Anzahl als negative Eigenschaft in die Fitneßfunktion des Genetischen Algorithmus aufnehmen und so eine optimale Anzahl ermitteln.
Wird ein Hidden-Neuron hinzugefügt, so müssen jedenfalls alle Eingangsneuronen mit diesem Neuron verbunden werden. Für die Ausgangsneuron gilt das gleiche.

Ausgangsneuronen

Es gibt vier Ausgänge des Neuronalen Netzes, somit gibt es auch vier Ausgangsneuronen. Die Ausgänge sind mit der Motorik (siehe Abschnitt 1.1.2) verbunden und bedeuten folgendes:

$\Delta\alpha$	Änderung des ersten Winkels
$\Delta\beta$	Änderung des zweiten Winkels
$\Delta\gamma$	Änderung des dritten Winkels
$\Delta\delta$	Änderung des Magnet-Winkels

1.2.2 Verbindungen

Wie schon erwähnt geht von jedem Eingangsneuron eine gerichtete, bewertete Verbindung zu jedem Hiddenneuron.

Speichermodell der Verbindungen:
Das gesamte Neuronale Netz wird als gerichteter bewerteter Graph angesehen und die Verbindungen somit durch die Adjazenzmatrix repräsentiert. Wird ein Wert der Matrix gleich Null gesetzt, so bedeutet dies, daß die entsprechende Verbindung nicht existiert. Da

in der Adjazenzmatrix ja alle möglichen Verbindungen berücksichtigt werden, bleiben viel der Einträge gleich Null. Das sind:

- Eingangsneuronen werden untereinander nicht verbunden.
- Hiddenneuronen werden untereinander nicht verbunden. Später vielleicht doch, als Rückkopplungen.
- Ausgangsneuronen werden untereinander nicht verbunden.
- Eingangsneuronen werden nicht direkt mit Ausgangsneuronen verbunden.

Die Gewichtungen der Verbindungen geben an, wie stark das Signal des sendenden Neurons verstärkt bzw. abgeschwächt wird. Die Gewichtungen können auch negativ werden. Sie haben aber auf jedem Fall einen maximalen Wert der für den posiviten und den negativen Bereich gilt. Die Werte werden als short-integer codiert. Sie sind allerdings bei der Berechnung durch einen Genauigkeitsparameter (standardmäßig = 100) zu dividieren.

1.3 Genetic Algorithm

Die Gewichtungen des ANN werden durch einen Genetischen Algorithmus so eingestellt, sodaß im Endeffekt der Roboter auf Werte, die die Sensorik liefert richtig reagiert. Das erste Struktogramm beschreibt den Grundablauf des Genetischen Algorithmus (Abb: 1.3).

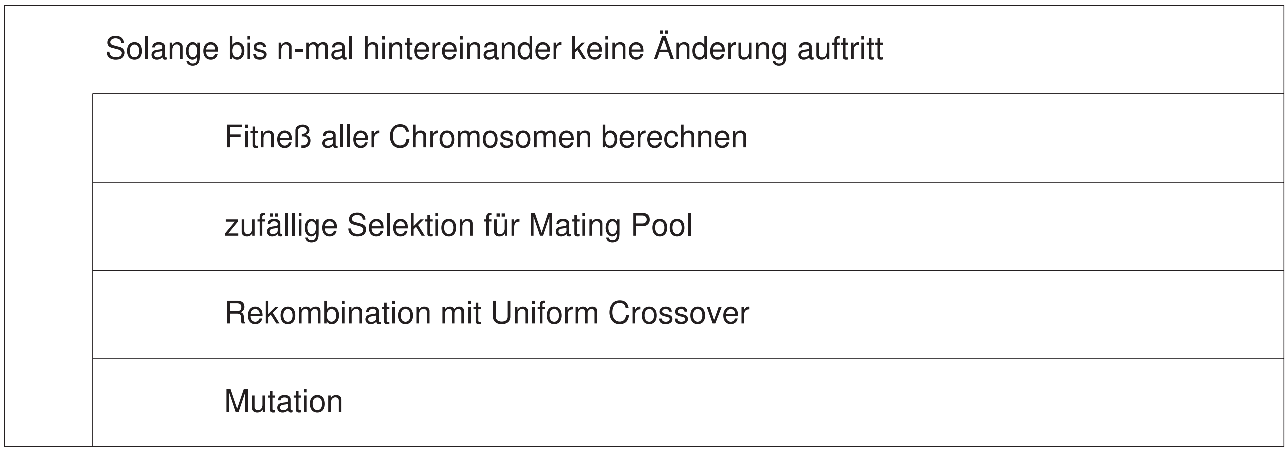

Abbildung 1.3: Genetischer Algorithmus - grob

1.3.1 Chromosomen

Wie üblich bei GA bestehen Chromosomen aus Genen. Jedes Gen entspricht in unserem Falle einer Gewichtung. Da jede Verbindung mit einer Gewichtung bewertet ist entspricht die Anzahl der Gene pro Chromosom (=:g) der Verbindungsanzahl. Die Größe der Adjazenzmatrix ist n Zeilen mal n Spalten. Die Gewichtungen werden mit Short-Integerwerten codiert, die jeweils 2 Byte im Speicher belegen.

bei 11 Neuronen im Hiddenlayer:
$g = 11 \cdot 11 + 11 \cdot 4 = 165$
$n = (11 + 11 + 4) = 26$
$\text{MemSize} = n \cdot n \cdot 2\text{Byte} = 1,4\text{KB}$

bei 22 Neuronen im Hiddenlayer:
$g = 11 \cdot 22 + 22 \cdot 4 = 330$
$n = (11 + 22 + 4) = 37$
$\text{MemSize} = n \cdot n \cdot 2\text{Byte} = 2,7\text{KB}$

1.3.2 Fitneßberechnung

Jedes der Individuen wird hier auf ihre Fitneß getestet. Das heißt das ANN, des Individuums errechnet aus den aktuellen Werten der realen Welt die Ausgangsparameter, die nun an die reale Welt zurückgegeben werden. Die reale Welt führt jetzt lediglich alle Berechnung durch, führt aber die Änderungen nicht aus. Diese Berechnung liefert dem Fitneßalgorithmus nun neue Parameter der realen Welt zurück. Die Fitneßfunktion hat nun zwei Parameterlisten, die sie vergleichen kann. Das Struktogramm in Abb 1.4 veranschaulicht diesen Algorithmus.

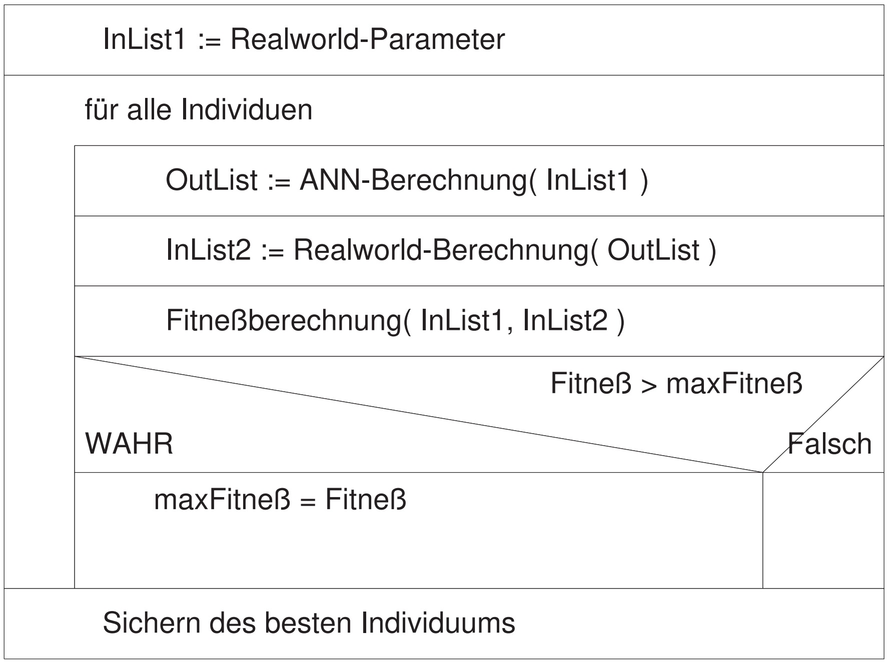

Abbildung 1.4: TWorld::calcFitness

Fitneßfunktion

Die Sensorik und folgende Kriterien wurden so gewählt, daß die Fitneßfunktion nach jeder Mikro-Bewegung errechnet werden kann.

- Gute Bewertung wenn Magnetscheibe waagrecht ist.
- Gute Bewertung, wenn sich die Magnetscheibe genau in Richtung Ziel bewegt.
- Beim Holen ist das Ziel der oberste Punkt der Kugel.
- Beim Bewegen zum anderen Podest ist das Ziel der Mittelpunkt der Plattform. Der unterste Punkt der Kugel soll dem obersten Punkt der Plattform angenähert werden.
- Beim Nähern an das Podest werden zusätzliche Pluspunkte vergeben, wenn den Ecken ausgewichen wird, damit die Kugel nicht kollidiert.

Die Fitneßfunktion übernimmt zudem die Trennung der unterschiedlichen Schritte, die notwendig sind um die Aufgabe zu lösen. (siehe Absatz 1.1.1) Während der Berechnung der Fitneß, wird die maximale Fitneß ermittelt und das entsprechende Chromosom gesichert.

1.3.3 Selektion

Als Selektionsmethode wurde hier eine spezielle Variante der Tournamentselektion gewählt. Es werden immer zwei Individuen zufällig gewählt, verglichen und das bessere der beiden überschreibt das schlechteste der ganzen Population. (siehe Abb 1.5)

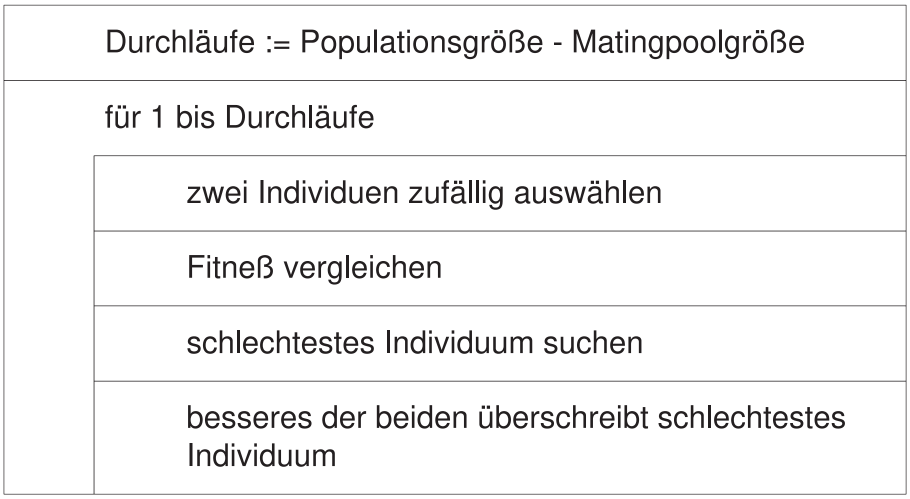

Abbildung 1.5: TWorld::Selection

1.3.4 Rekombination

Zunächst wird die Rekombination in zwei Teile unterteilt. Im ersten Teil wird die Hälfte der Population zufällig miteinander gepaart. Jedes gepaarte Individuum wird markiert, damit es nicht nocheinmal gepaart wird. Die zweite Hälfte wird dann der Reihe nach gepaart, damit der Zufallsgenerator nicht ständig auf die selben Individuen trifft und somit Rechenzeit verschwendet.
Vor dem Paaren zweier Individuen entscheidet wieder ein Zufallswert der mit einer Wahrscheinlichkeit p_R bewertet ist, ob die Rekombination zwischen diesen beiden Chromosomen

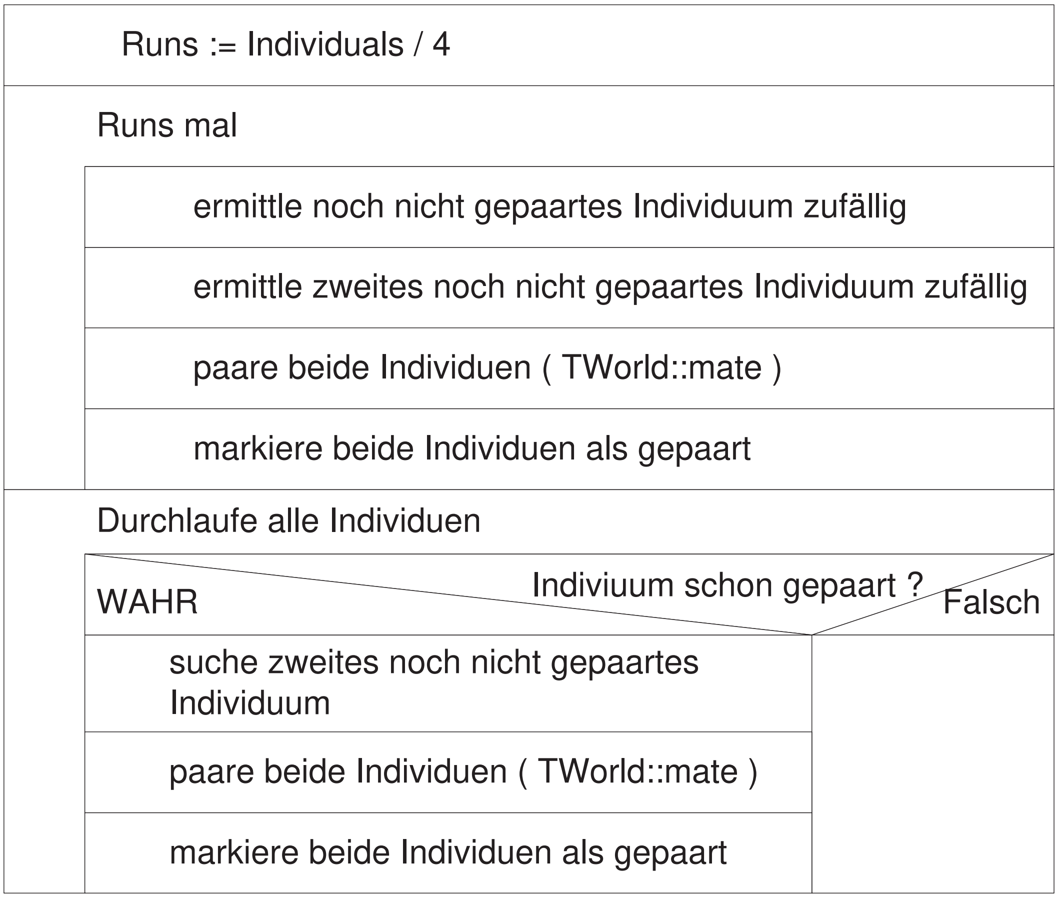

Abbildung 1.6: TWorld::Crossover

überhaupt stattfindet. Wenn ja, dann wird Onepoint Crossover verwendet. Das heißt es wird ein Crossoverpoint zufällig ermittelt, in unserem Fall eine Spalte und Zeile der Adjazenzmatrix. Ab der ermittelten Position werden nun alle Werte in den Matrizen der beiden Individuen ausgetauscht.

1.3.5 Mutation

Jeder Wert der Adjazenzmatrix, der nicht Null ist wird mutiert. Dazu werden die Integerwerte in ihrer Binärdarstellung betrachtet. Die Bits werden der Reihe nach durchgegangen und kippen mit einer Wahrscheinlichkeit von p_Bit, wobei diese Wahrscheinlichkeit abhängig ist von der Wertigkeit des Bits $BitValue$. Bits, die weiter links in der Binärdarstellung der Integerzahl stehen sind mit einer geringeren Wahrscheinlichkeit p_Bit behaftet. Begonnen wird die Mutation beim höchstwertigten Bit ganz links. Die Nummer des höchstwertigen Bits ist abhängig vom maximalen Wert, der in der Adjazenzmatrix auftreten kann:

$$\mathsf{HyBit} = log_2(\mathsf{maxValue}) \tag{1.1}$$

Die Mutationsrate soll die Kippwahrscheinlichkeit des mittleren Bits angegeben. Die Kippwahrscheinlichkeit für das vorderste Bit errechnet sich also mit:

$$\mathsf{BitPropabilityFirst} = \frac{\mathsf{MutationRate}}{\mathsf{BitFaktor}^{\frac{\mathsf{HyBit}}{2}}} \tag{1.2}$$

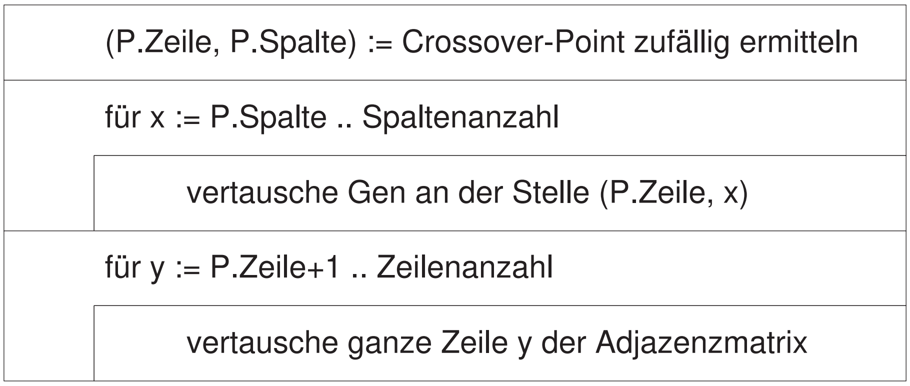

Abbildung 1.7: TWorld::mate

Die Wahrscheinlichkeit für jedes weitere Bit ergibt sich zu:

$$\mathsf{BitPropability}_i = \mathsf{BitPropabilityFirst} \cdot \mathsf{BitFaktor}^i \tag{1.3}$$

Wobei $BitFaktor$ die relative Änderung der Wahrscheinlichkeit von einem Bit zum nächsten angibt und i den Abstand zum vordersten Bit.

Die Wertigkeit der einzelnen Bits wird benötigt damit das Kippen mittels XOR-Funktion durchgeführt werden kann. Sie ist für das höchstwertige Bit:

$$\mathsf{BitValueFirst} = 2^{\mathsf{HyBit}} \tag{1.4}$$

Der Wert für jedes weitere Bit ergibt sich zu:

$$\mathsf{BitValue}_i = \frac{\mathsf{BitValueFirst}}{2^i} \tag{1.5}$$

Schleifenausstiegsbedingung ist: $\mathsf{BitValue} = 0$

Die Entscheidungszahl für das Kippen eines Bits ergibt sich nun zu:

$$p_{M,i} = \mathsf{BitPropability}_i + U \tag{1.6}$$

Wobei U hier eingführt wird als gleichverteilte Zufallszahl im Intervall $[0,1]$. Der linke Teil in der Summe ist die Wahrscheinlichkeit für die Änderung. Die Änderung wird durchgeführt, wenn die Summe $p_M \geq 1$.

Vorzeichen:
Die Wahrscheinlichkeit, daß das Vorzeichen kippt ist abhängig vom Absolutwert des Gens. Umso größer die Zahl desto unwahrscheinlicher ist, daß sich das Vorzeichen ändert.

$$p_M = \frac{\mathsf{MutationRate} \cdot (\mathsf{maxValue} - \mid Gen \mid)}{\mathsf{maxValue}} + U \tag{1.7}$$

1.3.6 Abbruchbedingung

Der Genetische Algorithmus wird dann beendet, wenn nach einer bestimmten Anzahl von Generationen n_G keine Änderung mehr aufgetreten ist.

Kapitel 2

Entwicklungsdokumentation

2.1 Datum: 14.05.1999

2.1.1 Festlegungen

- Das Programm soll in Borland C++ geschrieben werden.
- Verwendung von objektorientierter Programmierung.

2.1.2 class TANN

Deklaration einer Klasse TANN, die das ANN repräsentiert. Die Anzahl der Knoten ist variabel, und wird durch folgende **Eigenschaften** bestimmt:

Nodes	gesamte Knotenanzahl des ANN
InputNodes	Anzahl der Eingangsknoten
OutputNodes	Anzahl der Ausgangsknoten
HiddenNodes	Anzahl der versteckten Knoten

Weitere Eigenschaften der Klasse sind:
short maxWeight
Bereich der Gewichtung geht von -maxWeight bis + maxWeight.

short precWeight
Gibt die Genauigkeit der Gewichte an. Dabei bedeutet 100 z.B. das auf Hundertstel genau gerechnet wird.

short Treshold
Die vier Ausgangsknoten benötigen diesen Schwellwert, um von Null auf +1 bzw. -1 schalten zu können.

```
struct TNode
      {     int internValue
            short Output
      }
```

Der Knotentyp des ANN besteht aus zwei Unterfeldern, dem internen Wert, welcher der Summe der Eingänge entspricht und aus dem Ausgangswert. Der Ausgangswert wird durch eine Ausgangsfunktion aus dem internen Wert errechnet.

int maxIntern
Der interne Wert des Knotens kann eine Sättigung erreichen, bei der er einfach stehenbleibt, wenn die Summe der Eingänge größer wäre.

typedef TNode *PNode;
Zeiger auf Knotentyp

PNode NodeList;
Knotenliste als dynamischer Array

typedef short TWeight;
Typ der Gewichte auf den Verbindungen im ANN

typedef TWeight *PWeight;
Zeiger auf Gewichttyp

PWeight *AdjazenzMatrix;
Die Adjazenzmatrix des ANN legt die Verbindungen und deren Gewichte fest. Sie hat so viele Zeilen wie es Knoten gibt. Die Anzahl der Spalten ist gleich groß.

Methoden von TANN

initANN
Initialisiert obige Eigenschaften, erzeugt die Knotenliste und die Adjazenzmatrix, außerdem wird die Adjazenzmatrix mit Zufallswerten belegt. Dabei bleiben Verbindungen die nicht vorhanden sind (z.B.: zwischen Input-Layer und Output-Layer) gleich null.

runANN
übernimmt die Eingangssignale in den Inputlayer. Berechnet dann der Reihe nach die Werte für die Hiddenlayer, dann für die Ausgangslayer. Beim Ausgangslayer wird der Schwellwert berücksichtigt.

2.2 Datum: 15.05.1999

2.2.1 class TChromosom

Jedes Chromosom im genetischen Algorithmus soll ein mögliches ANN repräsentieren, zusätzlich einige Eigenschaften enthalten, wie z.B. Fitneß. Die Idee ist nun die Klasse TANN der Klasse TChromosom zu vererben.

Eigenschaften von TChromosom

int Fitneß
Die Fitneß des Chromosoms sagt aus wie gut das ANN, das es enthält funktioniert.

short ApproximationFactor
Der Näherungsfaktor ist ein entscheidender Parameter für die Fitneßfunktion und wird mit der Annäherung des Greifers zum entsprechenden Ziel multipliziert. Das Produkt ergibt den Fitneßwert.

short BallRadius
Kugelradius

float MutationRate
Mutationswahrscheinlichkeit p_M gibt die Wahrscheinlichkeit an, daß ein Bit im Bit-String kippt.

Methoden von TChromosom

init
Initialisiert die das ANN und die Eigenschaften von TChromosom.

calcFitness
Berechnet die Fitneß des Chromosoms. Die Parameter aus der realen Welt werden betrachtet. Es werden genauer zwei Parametersätze, der aktuelle Satz und ein älterer, verglichen. Die Änderung der Parametersätze wird bewertet.
Hier wird einstweilen nur die Entfernung errechnet. Die Differenz zur vorigen Entfernung mit ApproximationFactor multipliziert ergibt die Fitneß.

mutate
Mutiert das Chromosom. Genauer: Verändert die Gewichtungen des ANN wie folgt:

- Gewichtungen mit Wert 0 werden unverändert gelassen, denn das würde heißen, daß neue Verbindungen hinzukommen. ($\longrightarrow$ vielleicht spätere Erweiterung - wachsende ANN)

- Jeder der Werte wird nun gesondert für die Mutation betrachtet.

- Die Werte sind Short Integer und die vordersten Bits sind somit höherwertig als die hinteren. Die Asymmetrie der Integerzahlen wurde durch Anpassung der Bit-Mutationswahrscheinlichkeit teilweise ausgeglichen. Dabei wird die Anzahl der benötigten Bits für die Gewichtungen ermittelt (Standardmäßig: 9 Bits). Dann wird das mittlere Bit auf die MutationRate gesetzt. Die Bits links davon bekommen eine mit einer Konstante dividierte Wahrscheinlichkeit, und die Bits rechts davon bekommen eine mit der Konstante multiplizierte Wahrscheinlichkeit für die Mutation.

2.3 Datum: 15.05.1999 - 16.05.1999

2.3.1 class TWorld

Die Welt des Genetischen Algorithmus beinhaltet die Population, die aus einer konstant großen Liste von Induviduen besteht.
Problem: Wie deklariert man in C++ einen dynamischen Array von Objekten ? Wäre das möglich, dann könnte man die Anzahl der Individuen während des Programmablaufes auch verändern.
Es wird also ein Array deklariert mit 100 Objekten von der Klasse TChromosom. Die Objekte von TChromosom werden nun Individuen genannt und mit Population[idx] angesprochen.

Eigenschaften von TWorld

short bestIndividualIdx
Index des Individuums mit der größten Fitneß.

class TChromosom bestInduvidual;
bestes Individuum

int maxFitnessLast
höchste Fitneß in der vorigen Generation. Dieser Wert wird mit der aktuellen Höchst-Fitneß verglichen. Dieser Vergleich bildet die Abbruchbedingung des GA.

short NoImprovement
Anzahl der Generationen, die keine Verbesserung mehr hervorgebracht haben.

short maxNoImprovement
Erreicht NoImprovement den Wert in maxNoImprovement so wird der GA verlassen.

short MatingPoolSize
Größe des Matingpools. Gibt die Anzahl der unterschiedlichen Individuen an, die für die Rekombination selektiert werden.

int *InputList
Diese Liste von short-Integer-Werten dient dazu mit der realen Welt Daten auszutauschen. Zusätzlich wird diese Liste dem ANN übergeben.

int *InputListOld
Zweiter Speicher für InputList.

int *OutputList
Liste der Ausgangsparameter vom ANN. Wird der realen Welt übergeben, und gibt dort die Ansteuerung der Motoren an.

Methoden von TWorld

TWorld
Konsktruktor dieser Klasse. Sie initialisiert die Chromosome und somit auch die ANNs. Initialisiert die eigenen Eigenschaften.

calcFitness
Berechnet die Fitneß aller Individuen. Dazu werden die aktuellen Parameter der realen Welt zunächst ermittelt und in InputListOld gespeichert. Dann wird für jedes Individuum ein Testlauf durchgeführt. Ein Testlauf besteht aus:

- dem Aufruf des ANN, daß mir aus den Parametern in InputListOld die Ausgangsparameter in OutList liefert,
- dem Übergeben von OutList an die Realworld-Funktion. Ein spezieller Parameter dieser Funktion ist ein Anzeiger, der hier false gesetzt wird. Dieser steuert ob nur die Berechnung durchgeführt wird, oder ob die Änderung tatsächlich ausgeführt wird. Diese Funktion liefert die neuen Realworld-Parameter in InputList zurück.
- Die Parameter in InputList und InputListOld werden nun von der Methode calcFitness verglichen. Diese Methode liefert den Fitneß-Wert des Individuums zurück.
- In der Schleife wird die maximale Fitneß ermittelt.

getWorst
Liefert den Index des schlechtesten Individuums. Dies wird für die Selektion benötigt.

Selection
Selektion der besten Individuen. Hier wird Tournament-Selektion verwendet. Es werden immer zwei Individuen zufällig ausgewählt und verglichen. Das bessere überschreibt das schlechteste in der ganzen Population. Bei jedem Vergleich fällt also ein schlechtes Individuum weg. Die Anzahl der Vergleiche ist somit:
Populationsgröße - Matingpoolgröße

Nach der Selektion sind Matingpoolgröße viele verschiedene Individuen in der Population. Es gibt also mehrfach gleiche Individuen.

crossOver
Führt die Rekombination durch (Genetischer Operator). Verwendet wird Onepoint - Crossover. Der Crossoverpoint kann hier nur zwischen zwei Werten der Adjazenzmatrix liegen, nicht mitten in einem Integerwert. Das würde auch wenig Sinn machen. Die Auswahl der beiden Individuen erfolgt anfangs zufällig. Die gepaarten Individuen werden markiert, damit sie nicht nocheinmal gepaart werden. Nachdem die Hälfte der Population zufällig miteinander gepaart wurde, werden die restlichen der Reihe nach gepaart, damit der Zufallsgenerator nicht ständig bereits gepaarte Individuen liefert und somit Zeit verschwendet.

mutate
Mutiert alle Individuen.

useBestIndividual
Verwendet das beste Individuum um die Bewegung in der realen Welt auszuführen.

2.4 Schnittstelle zwischen GA/ANN und simulierter Welt

Dieser Abschnitt muß später, wenn alles feststeht, korrigiert werden.

2.4.1 Function getRealworld

Diese Procedure soll die Eingangs-Parameter für das ANN, wie in der Tabelle Array-Einträge ersichtlich, in einem Feld liefern.
Der Parameter ist somit folgender Array vom Typ Int.

Typ	**IN/OUT**	**Beschreibung**
Array[11] of Int	OUT	Darin werden die Eingangsparameter für das ANN übernommen.

Array-Einträge	**Index**	**Parameter**
	0	Kugel.x
	1	Kugel.y
	2	Greifer.x
	3	Greifer.y
	4	Ziel.x
	5	Ziel.y
	6	Status
	7	alpha
	8	beta
	9	gamma
	10	delta

Hier gibt es Änderungen:
Parameter 4,5,6

Ziel.x, Ziel.y	.....	geben die Zielkoordinaten für die Kugel an.
Status	.....	-1 - Die Kugel liegt irgendwo.
		+1 - Die Kugel hängt am Greifer.

2.4.2 Function runRealWorld

Diese Procedure sollte die Mikro-Bewegung mit den Ansteuersignalen für die Motoren ausführen. Dabei gibt es zwei Möglichkeiten:

- Die Bewegung wird tatsächlich ausgeführt.
- Die Bewegung wird nur berechnet und die Realworld-Parameter werden zurückgegeben.

Die Parameter:

Typ	**IN/OUT**	**Beschreibung**
Array[4] of Int	IN	Hier werden die Ausgangsparameter des ANN übergeben. Das sind die Antriebssignale für die Motoren.
Array[11] of Int	OUT	Darin werden die neuen Eingangsparameter für das ANN übernommen.
boolean	IN	true - Berechnung mit Ausführung; false - nur Berechnung.

2.4.3 Variable BallRadius

Beinhaltet den Radius der Kugel und wird für die Fitneßfunktion benötigt.

2.5 Datum: 21.05.1999

2.5.1 Testphase

Getestet wurden:

- Class TANN
- TWorld::Selection
- TChromosom::mutate somit ist auch TWorld::mutate ok.
- TWorld::mate, später muß noch TWorld::CrossOver getestet werden.
- TChromosom::calcFitness später muß noch TWorld::calcFitness getestet werden.

2.6 Datum: 22.05.1999 - 23.05.1999

2.6.1 Dokumentation

Dokumentation in Latex vervollständigen.

2.7 Datum: 31.05.1999

2.7.1 Projekt zusammenfügen

Verbinden der beiden Teile, ann-ga.h und annsim.h.