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

Projekt-Dokumentation

1.1 Simulierte Welt

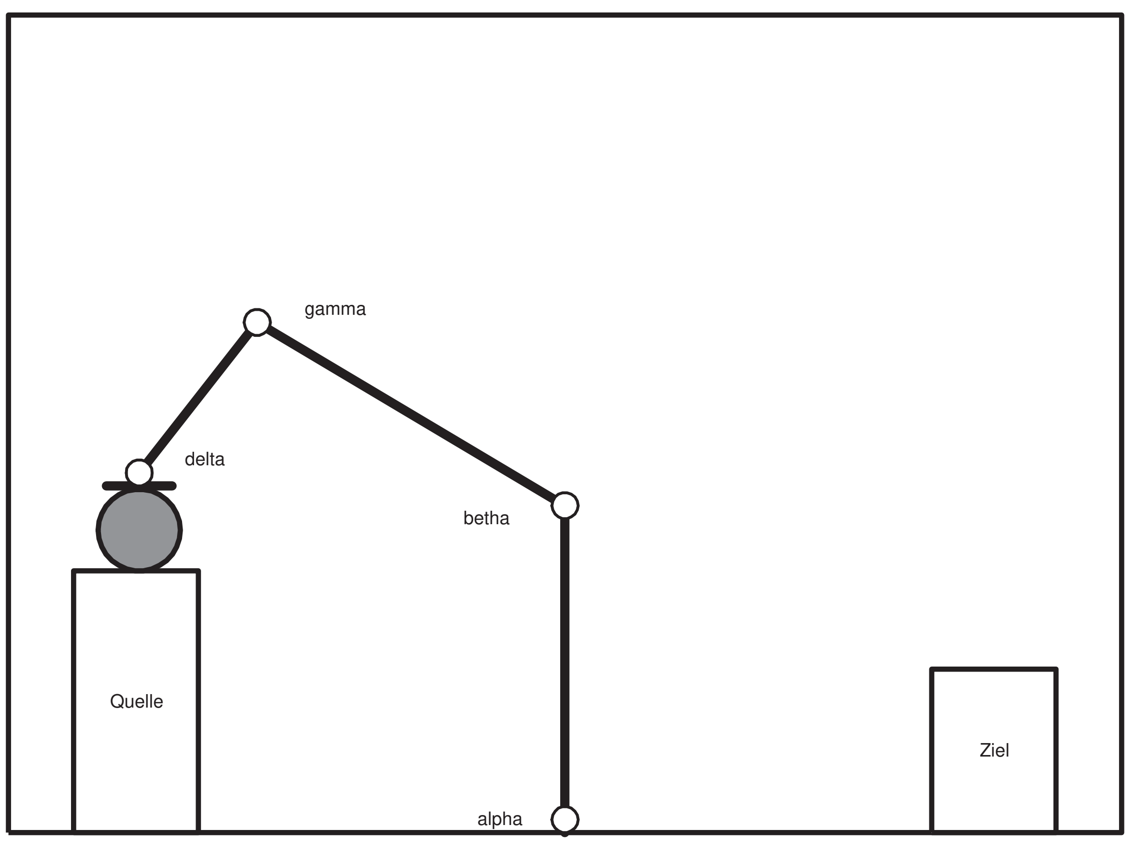

Abbildung 1.1: Simulierte Welt

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 positive und negative Werte Δw 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 bzw. des untersten Punkts der Kugel, falls die Kugel bereits dranhängt (Quelle.Mx, Quelle.My)
- Zielkoordinaten Ziel.x, Ziel.y
- Einen Status, der die Richtung des Transports angibt:

  -1 ..... nach links
  +1 ..... nach rechts

- Winkel der Gelenke α , β , γ , δ

1.2 Oberfläche

Die Oberfläche wird als Fenster realisiert, von dem aus alle Einstellungen des Programms entweder über die Menüleiste oder durch direkte Tastatureingabe möglich sind. Der simulierte Greifarm wird im Fenster dargestellt und wird entweder von dem ANN (soweit dieses bereits trainiert ist) oder vom Benutzer direkt gesteuert. Dadurch kann der Benutzer erkennen, wie schwierig es ist die Kugel von einem Podest zum anderen Podest zu bewegen und dabei den kleinsten Weg zurückzulegen.

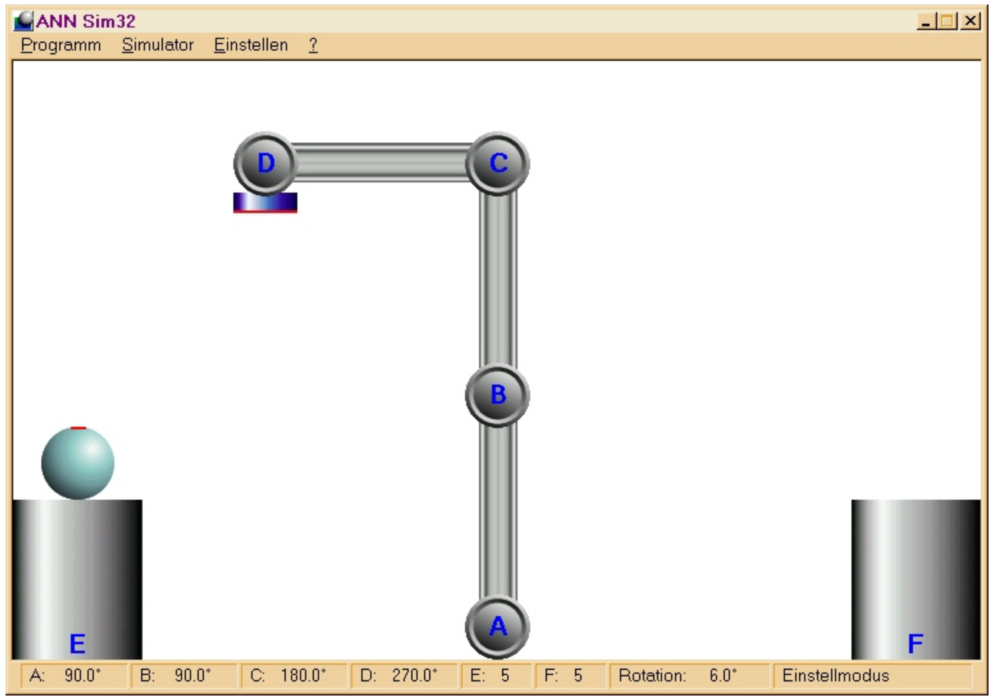

Abbildung 1.2: Benutzeroberfläche - Anfangsstatus

Bei jedem Start des Programmes befindet sich der Greifarm in der Ausgangsposition, wie in Bild 1 zu sehen. Danach kann der Benutzer über den Menüpunkt Simulator/Trainiere ANN oder mit F3 das Training 1 oder F4 Training 2 für das ANN aktivieren. Nach abgeschlossenem Training kann über den Menüpunkt Simulator/Aktiviere ANN oder mit F5 die Steuerung des Greifarmes an das ANN übergeben werden. Weiters kann der Greifarm direkt gesteuert werden, solange das ANN nicht trainiert wird oder die Steuerung des Greifarmes übernommen hat. Die Steuerung erfolgt entwerder über den Menüpunkt Einstellen oder durch die jeweiligen Tasten. Mit dem Menüpunkt Simulator/Zurücksetzen wird der Greifarm sowie die Podesthöhen auf die Startwerte zurückgesetzt. Über Simulator/Umrißmodus wird die gesammte Darstellung des Simulators nur in Umrissen gezeichnet, wodurch die Bewegungen, besonders bei langsamen Rechnern, weitaus schneller und flüssiger als in der Standarddarstellung sind.

1.2.1 Training

Sobald eine der beiden Trainingsarten gestartet wird, erscheint das Statistikfenster für das Training des ANN. Angezeigt werden die Gewichtsdaten des besten ANN, sowie weitere, von der Trainingsart abhängige Daten. Das Ende des Trainings erfolgt über den Knopf Abbrechen.

Trainingsart 1

Die Trainingsart 1 bewertet die einzelnen Mikrobewegungen des Greifarmes. Eine Bewegung in die richtige Richtung ergibt hohe Fitneßwerte. Dies hat zur Folge, daß sofort Erfolge für den Benutzer sichtbar sind und die ANN sehr schnell trainiert ist. Allerdings wird auf diese Weise dem ANN der Gesammtüberblick über das Problem entzogen, wodurch es innerhalb einer Ablegephase zu 'Vergessen' bereits bekannter Bewegungen kommen kann. Bei dieser Trainingsart kann der Benutzer die Bewegungen des Balles während des Trainings direkt im Grafikfenster beobachten. Hier werden zusätzlich die Distanz zum Zielpunkt, die beste Fitness und die Anzahl der besten verwendeten ANN im Trainingsfenster angezeigt.

Trainingsart 2

Bei der Trainingsart 2 des ANN werden gesammte Bewegungsabläufe des Greifarmes bewertet, die zum Aufnehmen, bzw. Ablegen der Kugel führen. Dabei dauert es, im Vergleich mit Trainingsart 1, bedeutend länger zufriedenstellende Ergebnisse zu erzielen. Allerdings wird an einer Gesamtlösung gearbeitet. Das Training wird auch für verschiedene Höhen ausgeführt. Im Trainingsfenster werden zusätzlich die durchschnittliche und beste Fitness, die Anzahl der Generationen und Individuen von ANNs angezeigt.

1.2.2 ANN Steuerung

Sobald das Training des ANN abgeschloßen ist, kann der Greifarm von der ANN gesteuert werden. Dadurch ist für den Benutzer sofort ersichtlich, ob mit den vorhandenen Einstellungen des Trainings das ANN ausreichend für diese Aufgabenstellung ist. Der Greifarm führt alle Winkelveränderungen der ANN durch und sollte den Ball vom linken zum rechen Podest bewegen. Da das ANN auf verschiedene Höhen der Podeste trainiert worden ist, sollte es für das ANN nach längerem Training möglich sein, auch verschiedene Höhen der Podeste zu meistern.

1.3 Artificial Neural Network

1.3.1 Neuronen

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

$\overrightarrow{X} = (x_1, x_2, \ldots, x_n)$	Eingangsvektor des Neurons
y	Ausgang eines Neurons

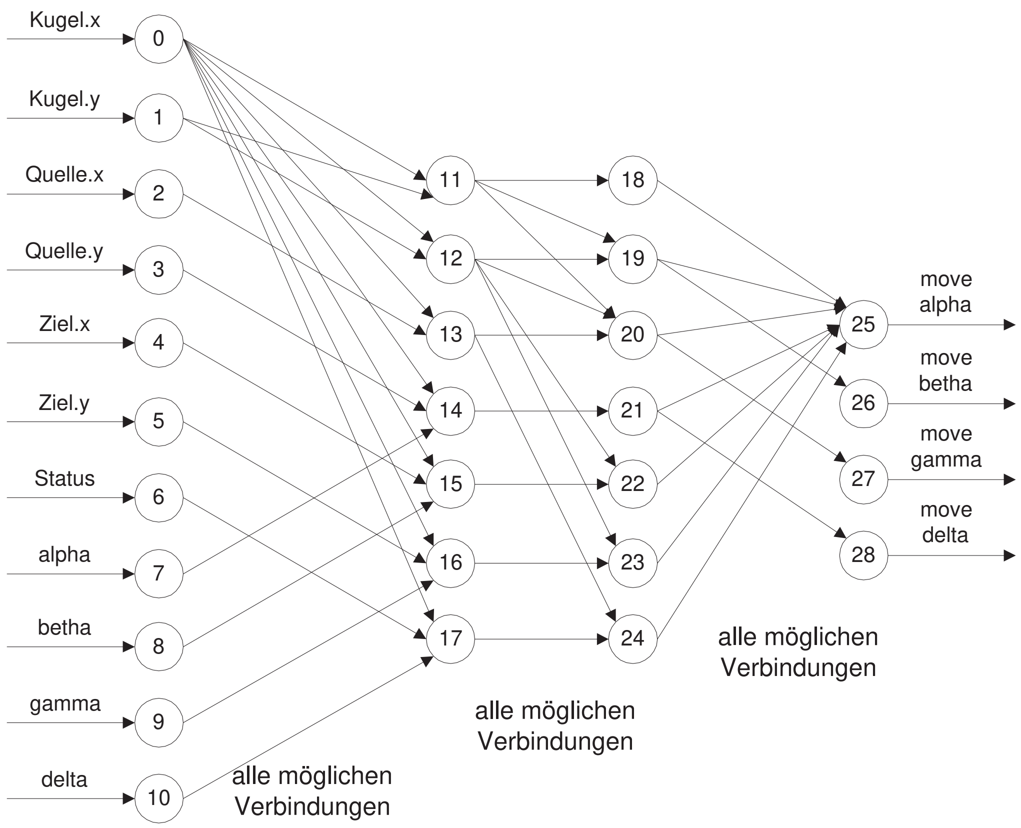

Abbildung 1.3: Artificial Neural Network

Somit ergibt sich für den Ausgang y:

$$y = \sum_{i=1}^{n} x_i$$

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
Quelle.x	X-Koordinate der Quelle
Quelle.y	Y-Koordinate der Quelle
Ziel.x	X-Koordinate zu der sich der unterste Punkt der Kugel hinbewegen soll
Ziel.Y	Y-Koordinate des Ziels
Status	gibt die Richtung des Transportes an (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 wurde in der Testphase experimentell ermittelt. Zudem wurde eine zweite Schicht eingebaut und die Möglichkeit eingeräumt, die Anzahl der Hiddenlayer bei der Instanzierung der Klasse TANN frei zu wählen. Die Anzahl der Hiddenknoten pro Layer ist für alle Hiddenlayer gleich, und kann ebenfalls bei der Instanzierung angegeben werden.

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

1.3.2 Verbindungen

Wie schon erwähnt geht von jedem Eingangsneuron eine gerichtete, bewertete Verbindung zu jedem Hiddenneuron der ersten Hiddenschicht . Von jedem Hiddenneuron der ersten Hiddenschicht geht eine Verbindung weiter zu jedem Neuron der zweiten Hiddenschicht, usw. bis zu den Ausgängen.

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 viele der Einträge gleich Null. Das sind:

- es gibt keine Verbindungen zwischen Neuronen einer Schicht.
- es gibt keine Verbindungen von Schicht i nach Schicht $i-1$.
- es gibt keine Verbindungen von Schicht i nach Schicht $i+\Delta i$, wobei $\Delta i > 1$.

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 werden allerdings beim Einsatz des Netzes durch einen Genauigkeitsparameter (standardmäßig = 100) dividiert.

1.3.3 Einsatz des ANN

Am Eingang werden zu einem Zeitpunkt t bestimmte Werte anliegen. Das Netz soll nun von diesen Werten abhängig, die Ausgangswerte berechnen.

$\overrightarrow{I} = (i_1, i_2, \ldots, i_l)$	Eingangsvektor des Netzes
$\overrightarrow{O} = (o_1, o_2, \ldots, o_m)$	Ausgangsvektor des Netzes
$Y = \{y_1, y_2, \ldots, y_k, \ldots, y_n\}$	Menge aller Neuronenausgänge (Axone)
y_k	Ausgang des Neurons k
w_{jk}	Gewichtung der Verbindung von Neuron j nach k

Zu einem Zeitpunkt t liegen am Eingang bestimmte Werte $\overrightarrow{I}$ an. Diese bewirken entsprechende Werte $\overrightarrow{O}$ am Ausgang. Es gibt also die Funktion:

$$\overrightarrow{O} = f(\overrightarrow{I})$$

Der Ausgangsvektor ist unabhängig von der Zeit t, da keine Rückkopplungen zugelassen werden. Die Funktion selbst wird durch die Gewichtungen an den Verbindungen des ANN bestimmt. Betrachtet man ein Neuron genauer ergibt sich für dessen Ausgang:

$$y_k = \sum_{j=1}^{n} y_j \cdot w_{jk}$$

1.4 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: 1.4).

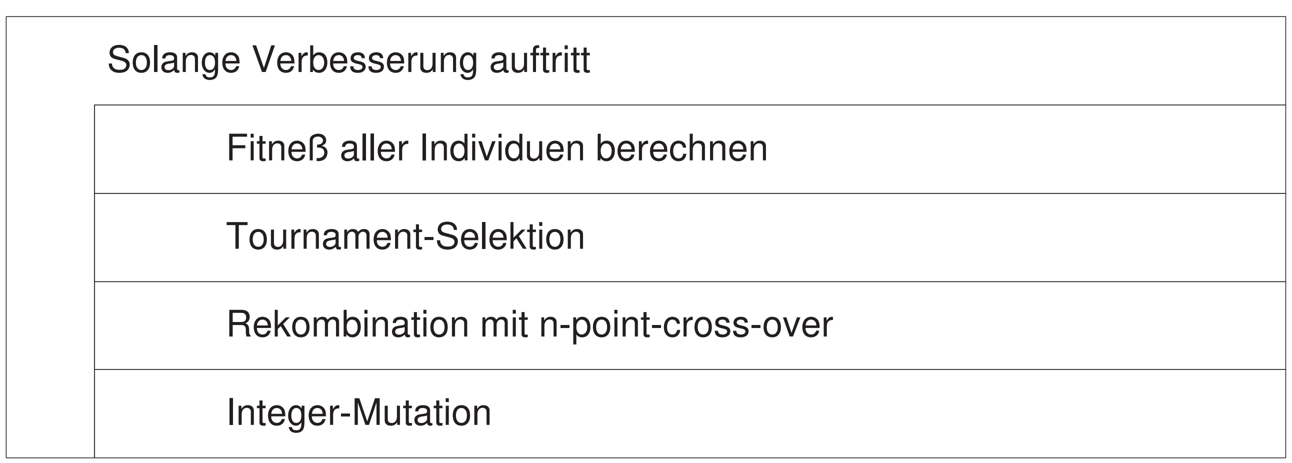

Abbildung 1.4: Genetischer Algorithmus

1.4.1 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 Anzahl aller möglichen Verbindungen $g = n^2$. Die Größe der Adjazenzmatrix ist n Zeilen mal n Spalten. Die Gewichtungen werden mit Integerwerten codiert, die jeweils vier Byte im Speicher belegen.

n	Anzahl der Knoten im Netz = Zeilen/Spalten-Anzahl der Adjazenzmatrix
g	Anzahl aller möglichen Verbindungen
l	Anzahl der Eingangsneuronen
m	Anzahl der Ausgangsneuronen
h	Anzahl der Hiddenneuronen pro Hiddenlayer
k	Anzahl der Hiddenlayer
v	Anzahl tatsächlich verwendeter Verbindungen
s	Speicherbelegung der Adjazenzmatrix

Die Anzahl der Verbindungen v, die wirklich vorhanden sind, das heißt wo die Gewichtung ungleich Null ist, ergibt sich zu:

$$v = l \cdot h + h^2 \cdot k + h \cdot m$$

bei 7 Neuronen pro Hiddenlayer:

$$v = 11 \cdot 7 + 7^2 \cdot 2 + 7 \cdot 4 = 203$$

mit:

$$n = l + h \cdot k + m$$

$$n = (11 + 7 \cdot 2 + 4) = 29$$

$$s = n^2 \cdot 4 \text{ Byte} = 29^2 \cdot 4 = 3,4 \text{ KB}$$

1.4.2 Fitneßberechnung Variante 1

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 1.5 veranschaulicht diesen Algorithmus.

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.

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.

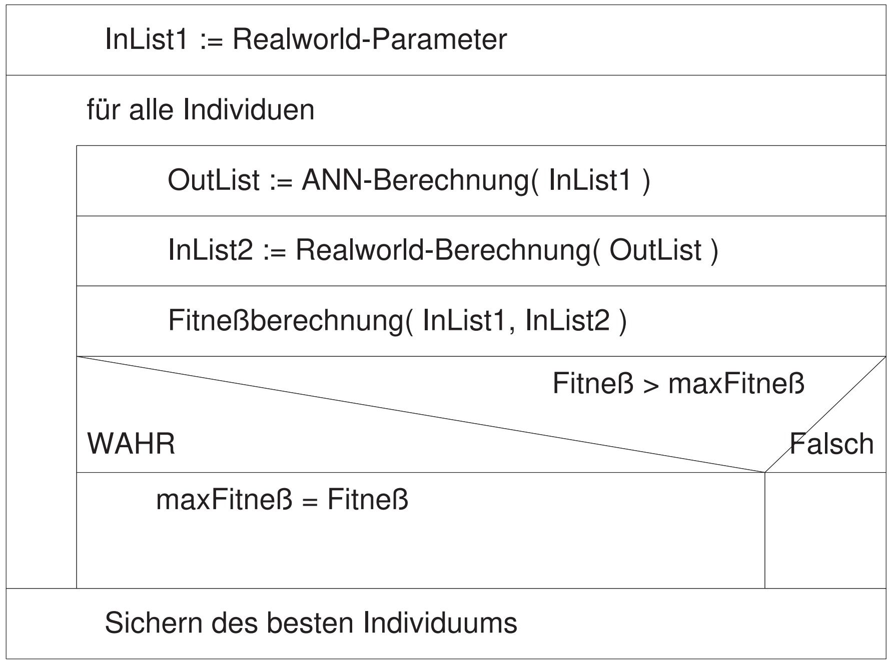

Abbildung 1.5: TWorld::calcFitness

1.4.3 Fitneßberechnung Variante 2

Beim Testen wurde erkannt das die Bestimmung der Fitneß nach jeder Mikrobewegung folgende Nachteile birgt:

- Die Entwicklung führt leicht zu lokalen Maxima, und zwar in der Nähe des rechten Podests.
- Das ANN ändert sich ständig und erreicht nie einen konstanten Zustand. Das kommt daher, da das Netz an jeder Stelle neu lernt sich in Richtung Ziel zu bewegen. Es verlernt also alte Stellungen.

Aus diesen Gründen wurde die Fitneßberechnung umgestellt. Jedes Individuum bekommt eine bestimmte Zeit lang eine Chance zu zeigen was es kann. Die meisten werden natürlich irgendwie sinnlos herumfahren. Aber eines von hundert wird vielleicht dabei sein, daß die Kugel erwischt und dann von links nach recht transportiert.

Das Individuum bekommt 200 Schritte zur Verfügung. Wird die Kugel getroffen, so wird es mit Fitneß belohnt und bekommt diesmal etwas weniger Schritte dazu, um vom linken zum rechten Podest zu gelangen. Erreicht es dieses Ziel so, bekommt es wieder einige Schritte um das nächste Ziel zu erreichen, usw... Die Größe der Belohnung ist umgekehrt proportional zu den Schritten, die dazu benötigt werden.

Gehen die Schritte aus, so ist die Fitneßberechnung, für das Individuum beendet und das selbe Spiel beginnt für das nächste. Das Individuum mit der größten Fitneß wird extra gespeichert. Es wird später bei der Steuerung verwendet. Für die Statusausgabe wird außerdem die durchschnittliche Fitneß berechnet. Sie gibt Auskunft über die Entwicklung der Population.

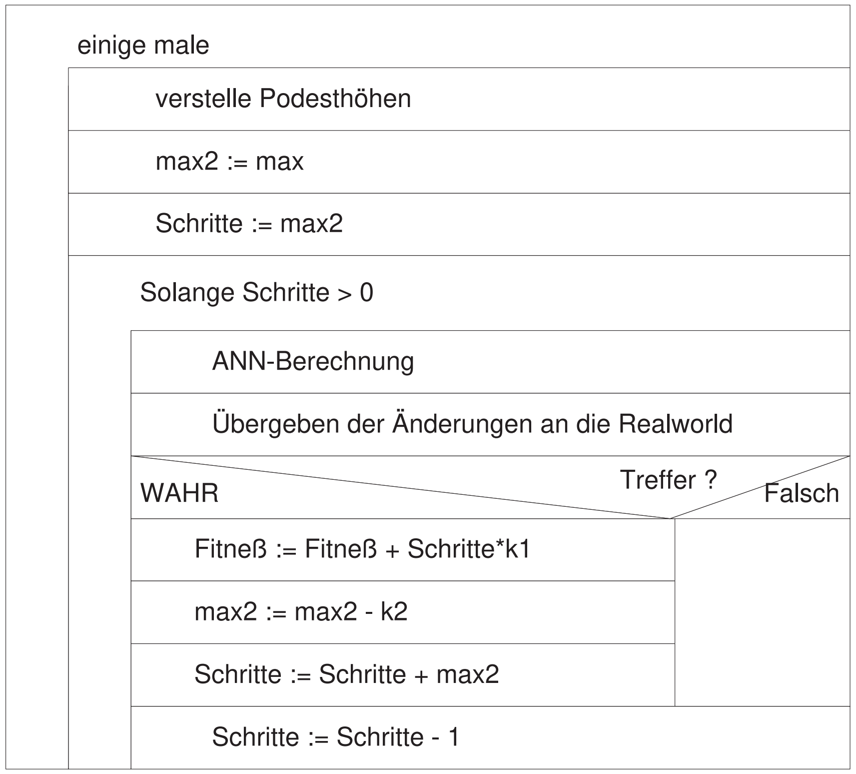

Abbildung 1.6: TChromosom::calcFitness2

1.4.4 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 1.7)

1.4.5 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 überhaupt stattfindet. Es wird nun n-Point-Crossover verwendet. Die Anzahl der Crossover-Punkte wird mit dem Zufallsgenerator ermittelt, die n Crossover-Punkte ebenfalls. Zwischen jedem zweiten Crossover-Punkt werden nun die Gene ausgetauscht.

1.4.6 Mutation

Jeder Wert der Adjazenzmatrix, der eine mögliche Verbindung im ANN darstellt, wird mutiert. Dazu werden die Integerwerte in ihrer Binärdarstellung betrachtet. Die Bits werden

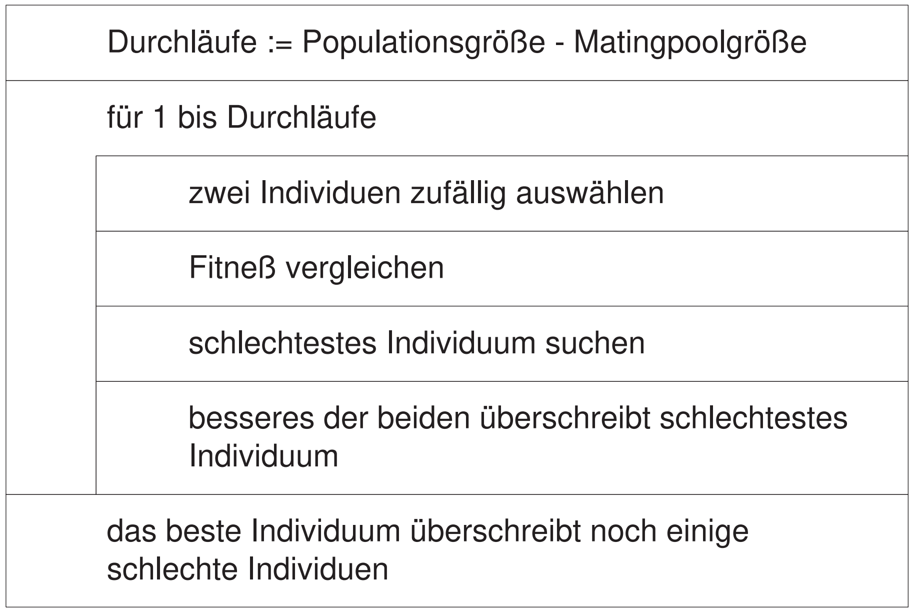

Abbildung 1.7: TWorld::Selection

der Reihe nach durchgegangen und kippen mit einer Wahrscheinlichkeit von p_{Mi}, wobei diese Wahrscheinlichkeit abhängig ist von der Wertigkeit des Bits 2^i. Bits, die weiter links in der Binärdarstellung der Integerzahl stehen sind mit einer geringeren Wahrscheinlichkeit p_{Mi} behaftet. Begonnen wird die Mutation beim höchstwertigten Bit ganz links. Die Mutationsrate soll die Kippwahrscheinlichkeit des Bits 0 ganz rechts angegeben. Die Kippwahrscheinlichkeit für das vorderste Bit errechnet sich also mit:

$$p_{Mn} \;=\; p_M \cdot k^{-n}$$

Die Wahrscheinlichkeit für jedes weitere Bit ergibt sich zu:

$$p_{Mi} \;=\; p_M \cdot k^{-i}$$

Wobei k die relative Änderung der Wahrscheinlichkeit von einem Bit zum nächsten angibt und i den Abstand zum least significant Bit.

Die Wertigkeit w der einzelnen Bits wird benötigt damit das Kippen mittels XOR-Funktion durchgeführt werden kann. Die Wertigkeit eines Bits i ist also:

$$w_i = 2^i$$

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

$$E = p_{Mi} + U(\,[0,1[\,)$$

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 $E \geq 1$.

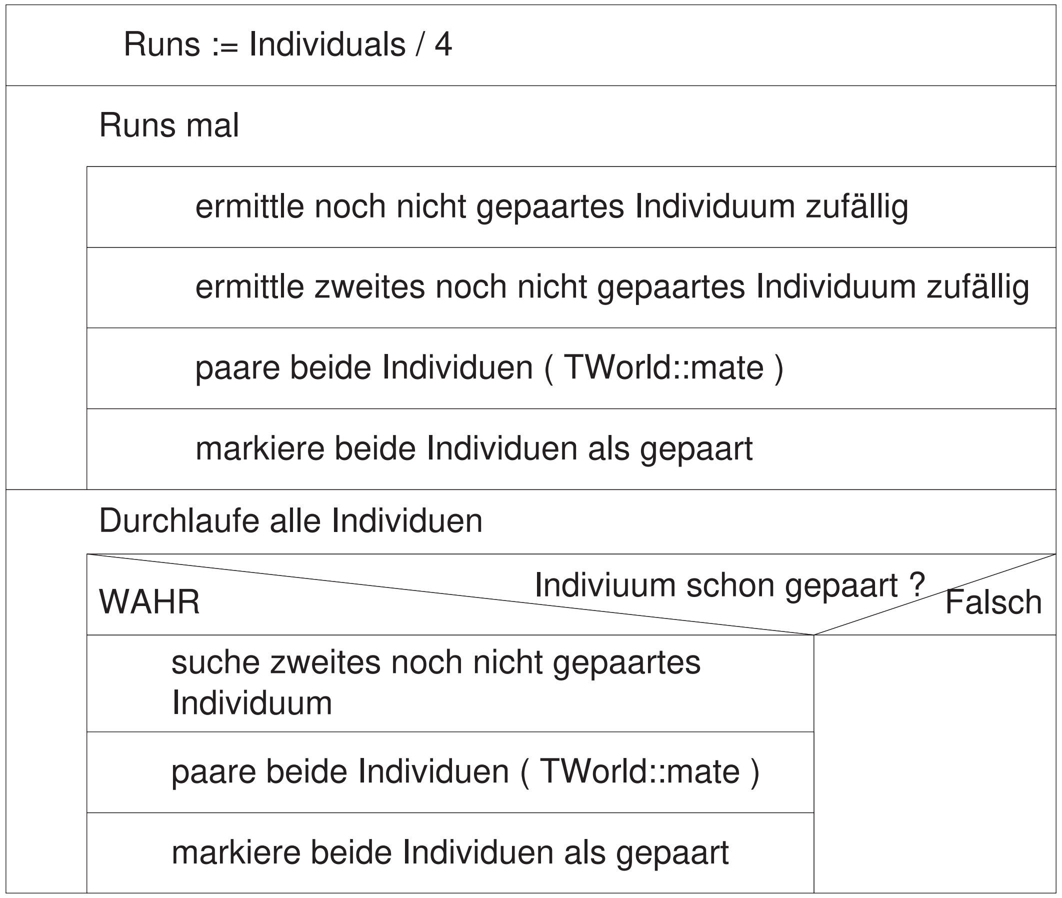

Abbildung 1.8: TWorld::Crossover

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.

$$p_{MV} = \frac{p_M \cdot (m - \mid g \mid)}{m} + U(\,[0,1[\,)$$

Wobei m der maximale Wert der auftreten kann ist und g der Wert des Gens.

1.4.7 Abbruchbedingung

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

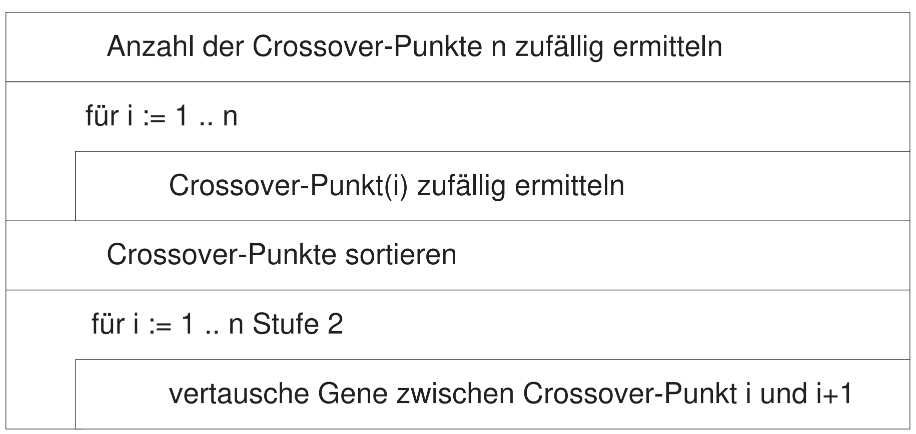

Abbildung 1.9: TWorld::mate

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

2.4.1 Parameter der realen Welt

Die Parameter sind im folgenden Array vom Typ Int gespeichert.

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

Array-Einträge	**Index**	**Parameter**	
	0	Kugel.x	Kugelmittelpunkt
	1	Kugel.y	
	2	Quelle.x	Magnetunterseite, wenn die Kugel auf der Plattform liegt
	3	Quelle.y	Kugelunterseite, wenn die Kugel schon dranhängt
	4	Ziel.x	Kugeloberseite, wenn sie noch liegt
	5	Ziel.y	Plattformoberseite, wenn sie dranhängt
	6	Status	-1 nach links, +1 nach rechts
	7	alpha	
	8	beta	
	9	gamma	
	10	delta	

2.4.2 GetCalcData

Deklaration:
int far* GetCalcData(int delta[4], bool keep, bool draw);

Diese Procedure sollte die Mikro-Bewegung mit den Ansteuersignalen für die Motoren ausführen.

Eingangsparameter:

delta[4]	Übernimmt die vier Winkeländerungen, die in die Winkel der Gelenke eingerechnet werden.
keep	Entscheidet, ob die berechneten Werte in die reale Welt übernommen werden sollen, oder ob sie lediglich an die aufrufende Funktion zurückgegeben werden sollen.
draw	Entscheidet, ob die aktuelle reale Welt dargestellt werden soll, oder nicht.

Ausgangsparameter:
Es wird ein Pointer auf die Parameter der Realworld zurückgegeben. Das ist also ein Pointer auf einen Array mit elf Integer-Einträgen.

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.

2.8 Datum: - 09.06.1999

- Korrektur der Übergabeparameter. Wie unter Abschnitt 2.4.1 ersichtlich.
- Es wurde ein Fenster eingebaut, daß aktuelle Daten des Programms anzeigen kann. Die jeweiligen Variablen, die darin angezeigt werden sollen, können durch Einfügen von zwei Zeilen an einer beliebigen Stelle im GA-Algorithmus hinzugefügt werden. In diesem Fenster wird auch die Adjazenzmatrix des besten Individuums dargestellt.
- Korrektur der Winkelwerte, sodaß sie zwischen -180 und $+180$ liegen.
- Da vermutet wird, daß im idealen Netz, falls es jemals gefunden wird, bestimmte Bereiche der Adjazenzmatrix für bestimmte Eingangsparameter zuständig sein werden, wurde das One-Point-**Crossover** durch ein n-Point-Crossover ersetzt. Die Anzahl n und die n Crossover-Points werden mittels Zufallsgen. erzeugt. Dann müssen die Crossover-Points sortiert werden, bevor die Bereiche zwischen zwei Punkte abwechselnd einmal vertauscht und dann wieder nicht vertauscht werden.

2.9 Datum: 10.06.1999

Mutation: Beim Testen wurde die Notwendigkeit erkannt sie anzupassen, weil manchmal eine Gewichting von null herauskam. Wenn eine Gewichtung einmal bei null angelangt ist, wird sie nämlich immer dabei bleiben.
Idee: Es wird die Möglichkeit eingebaut Nuller auch zu mutieren, allerdings nur zwischen Input- und Hidden-Layer und zwischen Hidden- und Output-Layer. Dafür werden kleine Werte null gleichgesetzt und die Anzahl der Null-Werte als positives Kriterium in die Fitneßberechnung aufgenommen. Somit wird zusätzlich zur Problembehebung ein Wachstum (Schrumpfen) der Verbindungen erreicht.

2.10 Datum: 14.06.1999

Fitneßfunktion: Beim Testen wurde erkannt das die Bestimmung der Fitneß nach jeder Mikrobewegung, dazu führt, daß:

- Die Entwicklung führt leicht zu lokalen Maxima, und zwar in der Nähe des rechten Podests.
- Das ANN ändert sich ständig und erreicht nie einen konstanten Zustand. Das kommt daher, da das Netz an jeder Stelle neu lernt sich in Richtung Ziel zu bewegen. Es verlernt also alte Stellungen.

Aus diesen Gründen wurde die Fitnessberechnung umgestellt. Jedes Individuum bekommt eine bestimmte Zeit lang eine Chance zu zeigen was es kann. Die meisten werden natürlich irgendwie sinnlos herumfahren. Aber eines von hundert wird vielleicht dabei sein, daß die Kugel erwischt und dann von links nach recht transportiert.

Die neue Fitneßberechnung ist in den Funktionen TWorld::calcFitness2 und TChromosom::calcFitness2 implementiert. Die alte kann noch immer aufgerufen werden, und ist jetzt in den Funktionen TWorld::calcFitness1 und TChromosom::calcFitness1.

In TChromosom::calcFitness2 bekommt das Individuum 200 Schritte zur Verfügung. Wird die Kugel getroffen, so bekommt es wieder (fast) 200 Schritte um vom linken zum rechten Podest zu gelangen. Erreicht es dieses Ziel so, bekommt es wieder einige Schritte um das nächste Ziel zu erreichen, usw...
In TWorld::calcFitness2 werden alle Individuen durchlaufen, die maximale und die durchschnittliche Fitness errechnet.

Hiddenlayers: Da sich noch immer keine Besserung zeigte, wurde das ANN auf beliebige Anzahl der Hiddenlayer erweitert. Zur Zeit werden zwei Hiddenlayer mit jeweils sieben Knoten verwendet.

Fitneßfunktion: Das Problem liegt darin, daß wenn der Winkel der Plattform bewertet wird, wird als Ergebnis nicht nur die Plattform waagrecht gehalten sondern auch der mittlere Arm senkrecht. Deaktiviert man diese Bewertung, so lernt das Netz. Selbst die Tatsache, daß der Weg unter dem Boden bestraft wird, kann das Netz ausgleichen. Es kann schon nach ca. 5 Generationen die Kugel holen und zum linken Podest heben.

2.11 Datum: 15.06.1999

Darstellung: Die Darstellung mit den Bitmaps sieht zwar sehr professionell aus, ist aber sehr langsam. Deswegen wurde zusätzlich die Möglichkeit eingeräumt, auf Umrißmodus umzuschalten. Der Roboter und die Podeste werden dann nur mit Strichen gezeichnet.

2.12 Datum: 18.06.1999-19.06.1999

Fitneßberechnung: Bisher wurde das Netz nur auf eine bestimmte Podesthöhe berechnet. Jetzt werden während die Fitneß eines Individuums berechnet wird zusätzlich, die Podesthöhen verändert. Dies geschieht vor der Berechnung und nachdem ein Ziel erreicht wurde. Die Höhen werden auf Zufallswerte gesetzt. Jedes Individuum hat dann seinen eigenen Spezialfall, für den es gut funktioniert. Wir hoffen nun, daß sich die Individuen bei der Rekombination ihre Spezialfälle austauschen.

Wachstum der Hiddenneuronen: Die Hiddenneuronen sollten durch Mutation wachsen. Die Anzahl der Hiddenneuronen wird dann bei der Fitneßberechnung als negatives Kriterium gewertet. Diese Funktion wurde jedoch wieder deaktiviert, weil es Probleme mit der Speicher-Reallokation gibt.

2.13 Datum: 21.06.1999

Dokumentation auf neuesten Stand bringen und Referat vorbereiten.

2.14 Datum: 23. ,24.06.1999

Fitneßfunktion: Das Berechnen der Fitneß mit verschiedenen Höhen, führt dazu, daß die Fitneß stark schwankt. Mit der Rekombination läßt sich dieser Fehler nicht mehr (oder schon gar nicht) ausgleichen. Die Folgerung: Die Fitneß eines Individuums wird jetzt für mehrere Podesthöhen berechnet. Es können 1..10 verschiedene Fälle, das läßt sich durch eine Konstante festlegen, bewertet werden. Zum Schluß wird der Mittelwert verwendet. Außer einer der Werte ist kleiner als ein konsanter Mindestwert. In diesem Fall wird die kleinste Fitneß verwendet (das schwache Glied in der Kette).
Zusätzlich wird jetzt die Fitneß des besten Individuums, vor dem Durchrechnen einer Generation, neuberechnet. Es ergeben sich nämlich durch unterschiedliche Podesthöhen unterschiedliche Fitneßwerte !

Hiddenknoten: Die Anzahl der Hiddenknoten wurde testweise verringert. Es konnten schon mit nur zwei Hiddenneuronen pro Hiddenlayer gute Ergebnisse erziehlt werden.

Rekombination: Die Anzahl der Crossover-Punkte wurde nun fix festgelegt, und nicht mehr mit Zufallsgenerator ermittelt. Es wird vermutet, daß sie ein Vielfaches oder ein Teiler von der Knotenanzahl sein soll, damit ganze Zeilen der Matrix vertauscht werden können. Es werden jetzt eher weniger Cross-Overpunkte verwendet, damit gute Bereiche der Adjazenzmatrix nicht zerrissen werden.

2.15 Datum: 25.06.1999

Rekombination: Letzte Erkenntnis dazu ist, daß es am besten ist, man nimmt geleich viele Cross-over-Punkte, wie man Zeilen (=Spalten = Knotenanzahl) in der Matrix hat.

Mutation: Warum soll die MutationRate für das mittlere Bit gelten ? Es wurde umgebaut, daß sie für Bit 0 gilt. Sie Beträgt als für Bit i:

$$p_{Mi} = p_M \cdot 2^{-i}$$