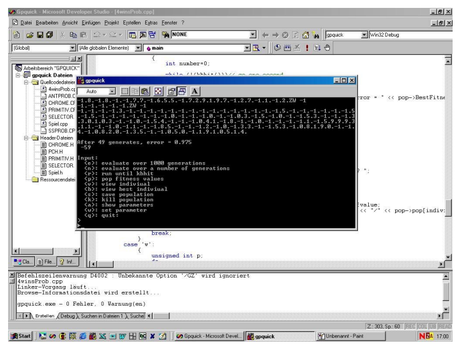
FACHHOCHSCHULE BRANDENBURG
FACHBEREICH INFORMATIK UND MEDIEN

Genetisches Programmieren von Spielstrategien

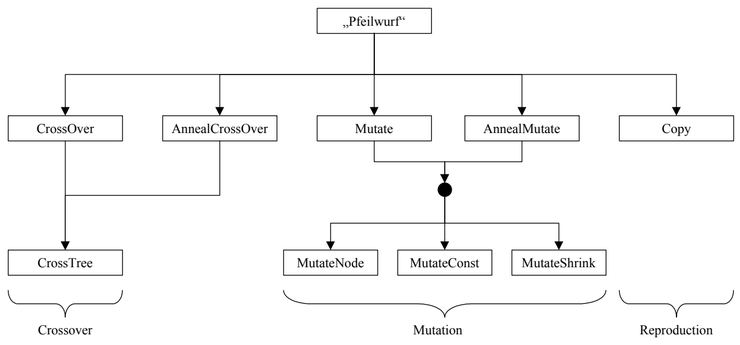
4 Gewinnt mit GP Quick

Semester-Arbeit von
Patrik Falk, Oliver Kroll, Stefan Müller, Tino Schonert & Marco Sorich



Aufgabenstellung

Zielstellung dieses Themas ist es, das allseits bekannte Spiel "4 Gewinnt" mit GP Quick umzusetzen. Dazu sind im Vorfeld folgende Fragen zu beantworten.
1. Vertrautmachen mit der Funktion und Arbeitsweise von GP Quick (d.h. mit dessen Objekten, Methoden und Übergabeparametern)
2. Welche Parameter sind für das Projekt erforderlich?
3. Wie funktioniert das Spiel an sich?



Darstellung einer Population

Populationsgröße
20

Bestes Individuum
4

Individuum-ID	Fitnesswert	Code
0	0.690476	(ADD (ADD 107 (get -72 -128)) (ADD (get (MUL (ADD -128 -128) (MUL -128 90)) -61) (MUL (get -128 -87) (get (SetVar -128 -128) -128))))
1	0.583333	(ADD (get (get -127 -128) -62) (ADD -126 (ADD (get -128 -128) (ADD (MUL 40 -124) (get -128 -41)))))
2	0.607143	(ADD -128 (ADD -128 (ADD (get -128 -128) (ADD (MUL 40 -126) (get -128 -43)))))
3	0.648148	(ADD (ADD -126 (ADD (get -128 -128) -128)) (ADD -128 -126))
4	0.706897	(ADD (get (get -25 (get (ADD -66 (get -128 -66)) 28)) -62) (ADD -126 (get -68 -128)))
5	0.5	(ADD (get (get -128 -128) -62) (ADD (ADD (get 73 -128) 42) (ADD (get -128 -128) (ADD (MUL 40 -126) (get -128 -43)))))
6	0.346154	(ADD -66 (ADD (get -128 -127) (ADD -128 (ADD 81 (ADD (get -126 -128) (MUL 120 -127))))))
7	0.447368	(get (ADD -96 -60) -62)
8	0.609375	(ADD (ADD -128 (get -72 -128)) (ADD (get (get (ADD (SetVar -128 (get (ADD -128 -128) (get (get 27 -116) -62))) -128) (get -128 -43)) -61) (get (get -128 -66) (get -128 -127))))
9	0.551282	(ADD -95 (ADD (get (get -69 (MUL -33 -91)) (ADD (get -128 -128) -128)) (get -96 -128)))
10	0.681818	(ADD (ADD -128 (get -128 -126)) (ADD (get -127 -80) (get (get -128 -127) (get -128 -128))))

GP Quick

GPQUICK ist ein schon teilweise fertig programmierter Genetic-Programming-Kernel. Man muss in diesem nur noch sein eigenes durch genetisches Programmieren zu lösendes Problem formulieren und diesen Kernel auf die eigenen Vorganben und Wünsche anpassen. GPQUICK ist in C++ programmiert und somit voll objektorientiert. Es unterteilt sich in mehrere Klassen. Die Klasse Chrome enthält ein einzelnes Individuum inklusive aller seiner Parameter und Zugriffsfunktionen. Diese ist eingebettet in die Klasse Pop, welche eine Liste vieler Chromes enthält. Pop repräsentiert somit die gesamte Population. Die Klasse Function enthält die Menge der Operationen des eigenen zu implementierenden Problems, welches selbst wieder Subklasse der Klasse Problem ist. Die Zuordnung und Anmeldung der durch genetisches Programmieren zu trainierenden Funktionen geschieht mit Hilfe vordefinierter Makros.
Das System ist leicht an das eigene Problem anpassbar, wenn auch nicht von Anfang an leicht zu verstehen. Es nimmt jedoch dem Programmierer viel Arbeit ab, die ansonsten durch die Implementation des genetischen Trainierens anfallen würde.

Die Klasse SPIELFELD

In der Klasse SPIELFELD wird der Zugriff auf das virtuelle Spielfeld realisiert. Unser Spielfeld hat eine Größe von 7x6 Feldern. In der Funktion Init() wird die Matrix erstellt und mit Startwerten belegt. Neben der Initialisierung ist auch die Funktion Test_auf_Reihe() von großer Bedeutung. In diese Funktion ist unser Algorithmus eingebunden. Der Algorithmus durchsucht die Matrix nach vier zusammenhängenden, gleichartigen „Steinen“. Die Funktionen Spiel_beendet() und Zug_ausfuehren() realisieren das Ausführen eines Zuges und geben das Spielergebnis zurück. Die Funktion get() gibt dem Individuum die Möglichkeit auf das Spielfeld zuzugreifen und so Informationen über eine Spielsituation zu bekommen. Das Individuum erlangt somit nur Kenntnis darüber, welcher Art „Stein“ auf welcher Koordinate liegt.

Die Klasse ARENA

In dieser Klasse werden die unterschiedlichen Spielmöglichkeiten realisiert. Diese sehen so aus, dass zwei Individuen der Population gegeneinander spielen, um zu „lernen“ und ihre Fitness zu verbessern. Eine weitere Möglichkeit ist, dass ein Mensch gegen ein Individuum spielt, um zu testen, wie gut es denn eigentlich ist. Als dritte Variante können zwei Individuen unterschiedlicher Populationen gegen einander spielen, um die Wirkung der eingestellten Parameter zu testen.

1	2	3	4	5	6	7
O						
O		O				
O		O		O		X
X		X		O		X
O		X		O		X
X		X		X		X

Die Abbildung zeigt die Darstellung eines Spielendes, wie es sich dem Mensch am Monitor darstellt. Dieses wurde von zwei Individuen erzeugt, die schon viel Zeit hatten zum „Lernen“.

Die Fitness

Das ausschlaggebende beim genetischen Programmieren ist die Bewertung der einzelnen Individuen einer Population. Die Bewertung wird als Fitness bezeichnet und stellt, je nach Formel, das Verhältnis der „Intelligenz“ der einzelnen Individuen einer Population untereinander dar. Bei uns sieht die Formel folgender Maßen aus:

$$Fitnesswert = \left(\frac{GewonneneSpiele}{Max.Spiele}\right) + \left(\frac{1}{2 * Max.Spiele}\right)$$

Diese Formel empfanden wir deshalb als sinnvoll, weil sie einen größeren Bonus vergibt für Individuen, die noch nicht so viele Spiele in der Population absolviert haben. Es gibt denen dann die Möglichkeit häufiger spielen zu können und somit schneller zu „lernen“. Denn der Fitnesswert bei der Auswahl von Individuen ist sehr wichtig. Da die Fitness so wichtig ist, treten alle neuen Individuen als erstes 10mal gegen andere Individuen an. Damit kann sichergestellt werden, dass jedes Individuum einen Fitnesswert hat.

Bestimmung der realen Fitness

Um ein objektives Maß für die Fitness der einzelnen Chromosomen zu erhalten, wird ein realer Fitnesswert bestimmt. Dies geschieht indem jedes Individuum gegen mehrere definierte Zufallsspieler antritt. Hierzu wurden zehn verschiedene Zufallsspieler generiert. Der Quotient aus der Anzahl der gewonnenen Spiele und der Gesamtspielzahl ergibt die reale Fitness.

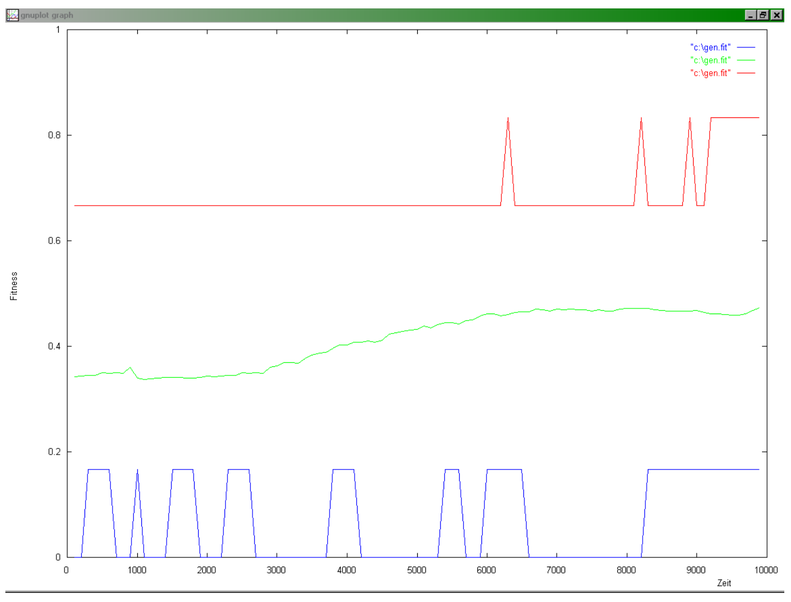


Abbildung:
Reale Fitness (Best, Average, Worst)
[Darstellung mit gnuplot unter Debian-Linux]

Auswertung, Fazit

Es ist uns gelungen, Programme entstehen zu lassen, die 4-Gewinnt spielen, auf dem Weg dorthin ist uns klar geworden, wie viele Parameter es gibt, die gut auf einander abgestimmt werden müssen. Schon bei der Wahl der Populationsgröße haben wir erst spät festgestellt, dass diese sehr viel größer sein sollte, was aber eine beträchtliche Erhöhung der Rechenzeit mit sich bringt. Allein mit der ungünstigen Einstellung eines Parameters, fällt das Ergebnis sehr viel schlechter aus. Unsere 4-Gewinnt Spieler besitzen zwar noch sehr primitive Verhalten, aber wir denken das mit der Überarbeitung einiger Faktoren, wie zum Beispiel die Implementierung von Variablen und Boolschen Funktionen das Ergebnis doch noch gesteigert werden kann.

Betreuer: Prof. Dr. J. Heinsohn
Dipl.-Inform. I. Boersch

Abgabe: 24.01.2002