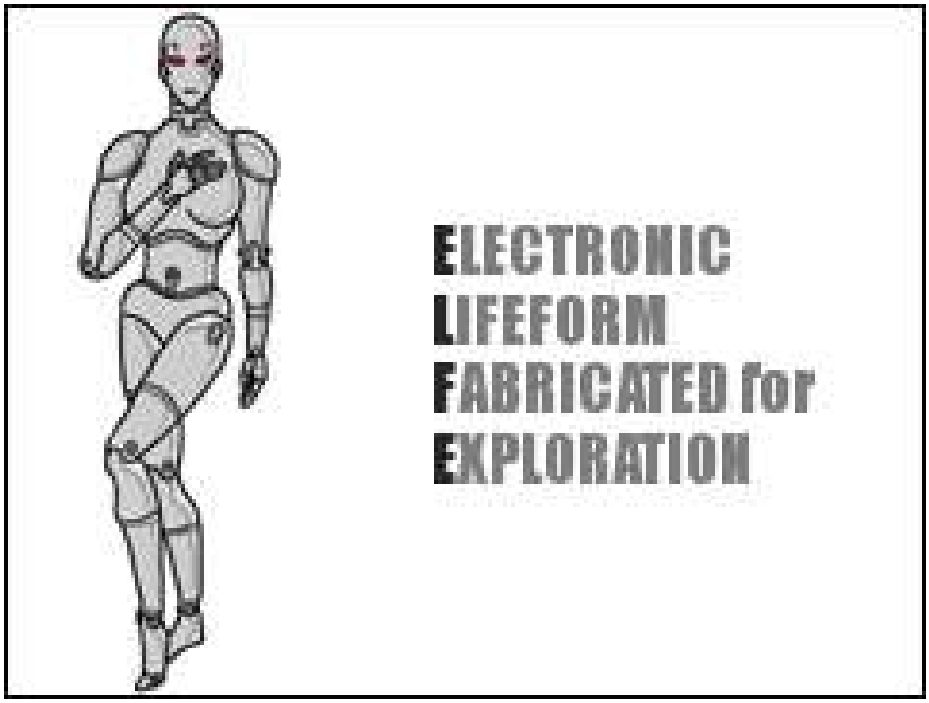
Dokumentation

Ein elfbeiniger Roboter lernt laufen

- Simulierte Evolution macht's möglich -



Teammitglieder: Dylan Baar, Andy Grabow, Jesko Nordlohne, Chavdar Papazov

Betreuer: Dipl.-Inform. Ingo Boersch

Inhaltsverzeichnis

Aufgabenstellung

Dem Projekt "Elfe" lag die Fragestellung zugrunde, ob es möglich ist, dass ein elfbeiniger Roboter (mit Namen "Elfe") eigenständig lernt, wie er seine elf Servomotoren ansteuern muss, um sich möglichst schnell vorwärts zu bewegen. Dabei sollte der Algorithmus zur Fortbewegung nicht durch das Programmierer-Team vorgegeben werden. Vielmehr war es dem Team freigestellt, beliebige Verfahren aus dem Gebiet der künstlichen Intelligenz zu implementieren, um den Roboter so in die Lage zu versetzen, eigenständig die optimale Ansteuerung der Servomotoren (Rotationswinkel, Verzögerungszeiten im Bewegungsmuster, Reihenfolge der Ansteuerung der "Beine") zu finden.

Hardware

Elfe

- Lauf-Roboter mit 11 Servo-Motoren (auf 0 – 180° positionierbar)
- Aksen-Board mit Microcontroller der 8051-Familie, serieller Schnittstelle mit angeschlossenem Bluetooth-Modul zum Flashen der Software und zur Kommunikation mit dem PC
- Gewöhnliche Com-Port Maus als Odometrie-Messsystem

PC

- Entwicklungsumgebung für C++ (PC-seitige Software) und C (Elfe-seitiger Interpreter)
- zwei serielle Schnittstellen mit angeschlossenem Bluetooth-Modul zum Übertragen der Software und zur Kommunikation mit Elfe

Planung

Grundidee

Das Team entschied sich, das Projekt mithilfe genetischer Algorithmen (GA) zu realisieren. Genetische Algorithmen sind eine Form der direkten Evolution, bei der eine Fitnessfunktion spezifiziert wird, mit welcher die Leistungsfähigkeit eines Individuums berechnet werden kann. Ein Individuum ist dabei zunächst ein zufällig zusammengewürfelter, beliebig langer Befehlssatz, der die Ansteuerung der Servo-Motoren beschreibt. Dieser Befehlssatz wird vom

Roboter interpretiert und ausgeführt. Über eine an den Roboter montierte gewöhnliche Computermaus wird nun gemessen, ob sich der Roboter bewegt hat und wenn ja, wohin. Anhand dieser Messwerte wird anschließend die Fitness des zugrundeliegenden Individuums (Befehlssatzes) berechnet. Ein Individuum, bei dem der Roboter ein Stück vorwärts gekommen ist, ist fitter als ein Individuum, bei dem der Roboter stehen geblieben ist oder sich rückwärts bewegt hat.

Vorüberlegungen

Zunächst stellte sich dem Team die Frage, aus welchen Komponenten die zu übertragenden Befehle zusammengesetzt sein sollten, damit sie eine eindeutige und effektive Ansteuerung der Servomotoren ermöglichen? Das gefundene Format hat die Form `{Servonr; Gradzahl; Verzögerung}`. Ein Beispielbefehl, der diesem Format entspricht, würde lauten: {drehe Servo7 auf 20 Grad, warte 30 ms, drehe Servo2 auf 32 Grad, warte 12ms}

Ferner wurde im Laufe des Entwicklungsprozesses die Frage geklärt, wo die Aussonderung kranker Individuen erfolgen solle?
Das Ergebnis: Individuen werden nicht auf dem AksenBoard als krank ausgesondert, (krank heißt, sie überschreiten die zulässigen Grenzen der Werebereiche für Zeit, Servonr und Winkel), sondern schon von dem evolutionären Algorithmus aussortiert bzw. mit einer sehr schlechten Fitness bestraft. Das spart Zeit, da das Individuum nicht erst auf dem AksenBoard evaluiert werden muss. Der evolutionäre Algorithmus generiert so lange neue Individuen, bis sie in den vorgegebenen Wertebereichen liegen. Nur gesunde Individuen werden weiter ans AksenBoard übertragen. Dort ist dann weder Clipping (Abschneiden zu großer Werte oder Hochschieben zu kleiner Werte) noch Modulo-Rechnung o.ä. nötig und es kann mit den direkt empfangenen Werten unmittelbar gearbeitet werden.

Eine weitere Befürchtung des Entwicklerteams stellte die Messungenauigkeit dar: Sie könnte zu groß werden, wenn ein sehr kurzes Individuum ausgeführt wird, bei dem die Kugel der Maus nur kurz und unspezifisch zuckt, die berechnete Geschwindigkeit des Individuums also sehr unzuverlässig ist. Lösungsansatz: Jedes Programm muss mindestens eine bestimmte Länge (gemessen in Zeit oder Anzahl der Befehle) haben, bevor es übertragen wird. Dieser Lösungsansatz wurde jedoch verworfen, weil nach Meinung des Entwicklerteams ohnehin die kurzen, kleinen Individuen, die nur kurz ziellos „herumzucken", sehr schnell aussterben und die manuellen Eingriffe in den GA möglichst minimal gehalten werden sollten.

Gewünschter Ablauf

Auf einem PC wird ein Programm ausgeführt, welches ein Zufallsindividuum erzeugt, z.B. der Form {`drehe Servo7 auf 20 Grad, warte 30 ms, drehe Servo2 auf 32 Grad, warte 12ms`}. Dieses Individuum wird in eine Zeichenkette übersetzt, die per Funk (Bluetooth) an das Board auf dem Roboter übertragen wird.

Die empfangenen Befehle werden in "Elfen-Bewegungen" umgesetzt und die

Bewegungssignale der Maus über ein serielles Kabel an den Computer zurück übertragen. Nach erfoglreicher Ausführung der Servobefehle meldet Elfe sich beim PC zurück. Aus den Mausdaten und der Kommunikation mit Elfe kann der PC die zurückgelegte Strecke und dafür benötigte Zeit ermitteln. Darauf basierend kann er die Fitness des soeben ausgetesteten Individuums bestimmen. Mit Hilfe evolutionärer Verfahren (Selektion, Mutation, Crossover) wird eine neue Population erzeugt (vgl. obige Abbildung).

Auf dem PC ist ein umfassendes Logging eingerichtet, wodurch es möglich ist, jeden einzelnen Evolutionsschritt für spätere Auswertungen festzuhalten bzw. bereits zuvor generierte Individuen auf das Elfe-Board nachzuladen.

Umsetzung

Allgemein

Interpreter

Der Roboter ist mit dem an der FH Brandenburg entwickelten Aksen Board ausgestattet. Zunächst wurde ein Interpreter in der Programmiersprache C entwickelt und in compilierter Fassung auf das Board übertragen. Der Interpreter hat die Aufgabe, auf dem seriellen Port des Boards zu lauschen, ob Servomotor-Befehle ankommen. Ein Three-Way-Handshake-Verfahren stellt sicher, dass die Übertragung per Funk fehlerfrei gelingt.

Das auf dem Elfe-Board implementierte ThreeWayHandshake-Protokoll funktioniert wie folgt: Der PC sendet ein syn-Signal an Elfe, um einen Verbindungsaufbau anzufordern. Die Anfrage wird durch die Elfe mit einem Acknowledgement bestätigt. Daraufhin beginnt der PC mit der Übertragung der Kommando-Triplets, die von Elfe mit einzelnen Acknowledgements bestätit werden. Hat der PC alle Befehle übertragen, signalisiert er das durch die Übertragung eines Fin-Signals, woraufhin Elfe mit der Zahl der erfolgreich empfangenen Pakete antwortet. Nach einem abschließenden Acknowledgement des PCs beginnt Elfe mit der Abarbeitung der Befehle. Wurde der Befehlssatz des Indiduums erfolgreich ausgeführt, signalisiert Elfe das mit einem Fin, welches den PC-seitigen Timer stoppt und die Berechnung des Fitnesswertes anstößt.

Elfe und PC können Fehler jederzeit über ein Nack-Signal anzeigen, was zu einem erneuten Kommunikationsaufbau führt.

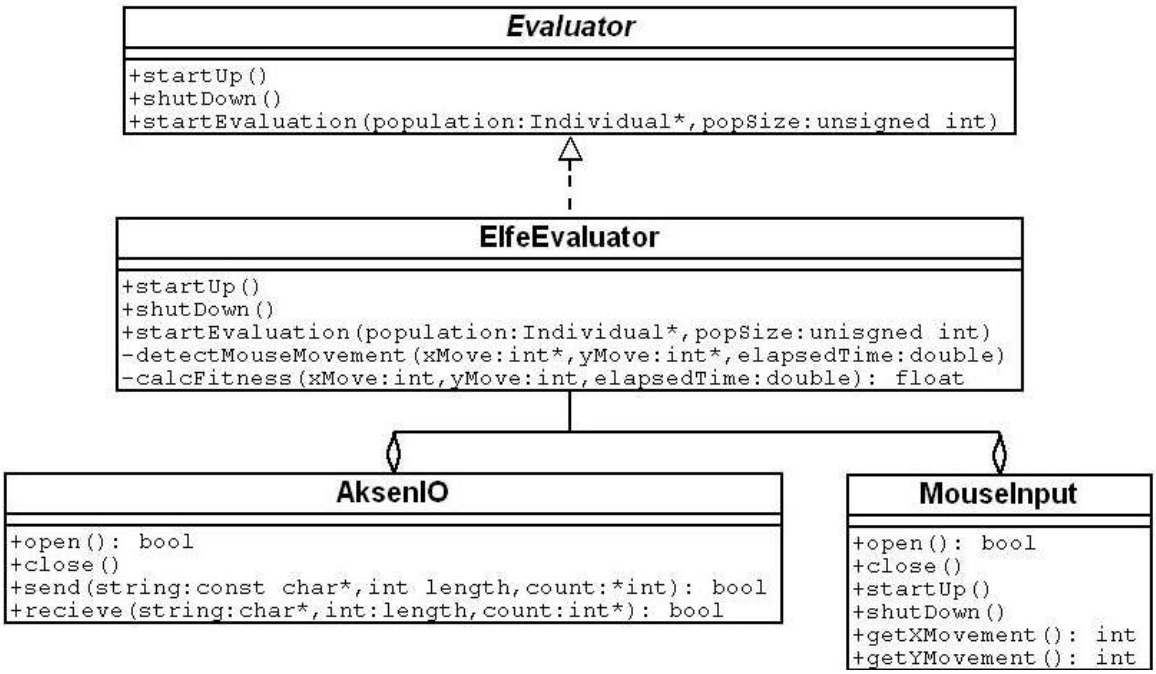
```
PC <==> AKSEN          BIDIRETIONALE KOMMUNIKATION
   ==>> s              syn Verbindungsaufbau anfordern
   <<== a              ack Verbindungsaufbau bestaetigen
   ==>> {1,2,3,}       Kommando-Triplet (Trennzeichen Komma)
                       {Servonr, Winkel, Zeit}
   <<== a              Empfangenes Triplet bestaetigen
                       (bei ungueltigem Triplet <<== n
                       und Sprung  zu syn
   ==>> f              fin Verbindung terminieren
   <<== x              ack: Alle Packete empfangen
   ==>> a              ack erfolgreiche Uebertragung bestaetigen
   <<== f              board bestaetigt erfolgreiche
                       Programm-Abarbeitung

   ==>> n              n (nack) dient jederzeit als
                       Fehlermeldung an PC
```

Das Aksen-Board gibt während der Programmabarbeitung laufend Statusmeldungen auf seinem LC-Display aus. Außerdem setzt es die in Grad-Einheiten übertragenen Servobefehle in normalisierte AksenBoard-spezifische Pulsweitenmodulationsbefehle um. Eine Normalisierung wurde notwendig, weil die Servomotoren nicht geeicht sind.

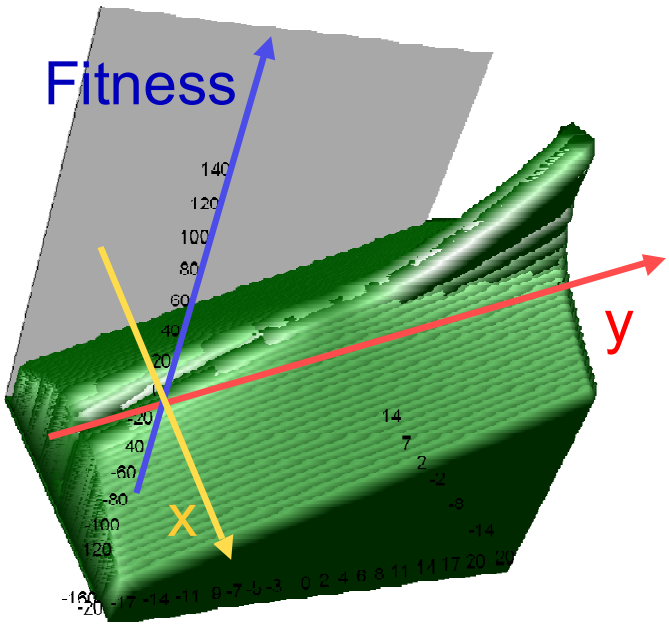
Evaluator

Der Evaluator berechnet die Fitness des jeweils zuletzt ausgeführten Individuums.



Der Berechnung der Fitnessfunktion liegt die Fortbewegungs-Geschwindigkeit eines Individuums zugrunde. Der PC berechnet jene Geschwindigkeit, indem er die gemessene Mausfortbewegung durch die Dauer der Programmausführung teilt.

Die Dauer wird wie folgt großzügig gemessen: Der Timer auf dem PC wird vor Übertragung des letzten Befehls des ganzen Individuums gestartet und wieder gestoppt wenn der PC das FIN empfangen hat, welches das AksenBoard nach Ausführung des letzten Befehls sendet. Mit Start und Stop des Timers wird auch die Bewegungsmeldung der Maus gestartet oder gestoppt.



Die zugrundegelegte Fitnessfunktion:

$$\frac{\frac{y}{|x|} * \sqrt{x^2 + y^2} + m * y - n * |x|}{t}$$

Die wesentlichen Charakteristika jener Gleichung lassen sich dem oben dargestellten Kurvenplot anschaulich entnehmen: Bewegt sich der Roboter in y-Richtung vorwärts ohne stark in x-Richtung vom Kurs abzuweichen, resultiert eine hohe Fitness. Bewegt sich der Roboter jedoch rückwärts bzw. zu stark in x-Richtung, ergibt sich ein negativer Fitnesswert.

Obige Formel stellt ferner sicher, dass zwei Individuen A und B, welche real gleich fit sind, weil sie z.B. den gleichen Fortbewegungs-Algorithmus benutzen, auch dann den gleichen Fitnesswert zugewiesen bekommen, wenn sie unterschiedliche Zeiten zur Abarbeitung des Algorithmus zur Verfügung gestellt bekommen.

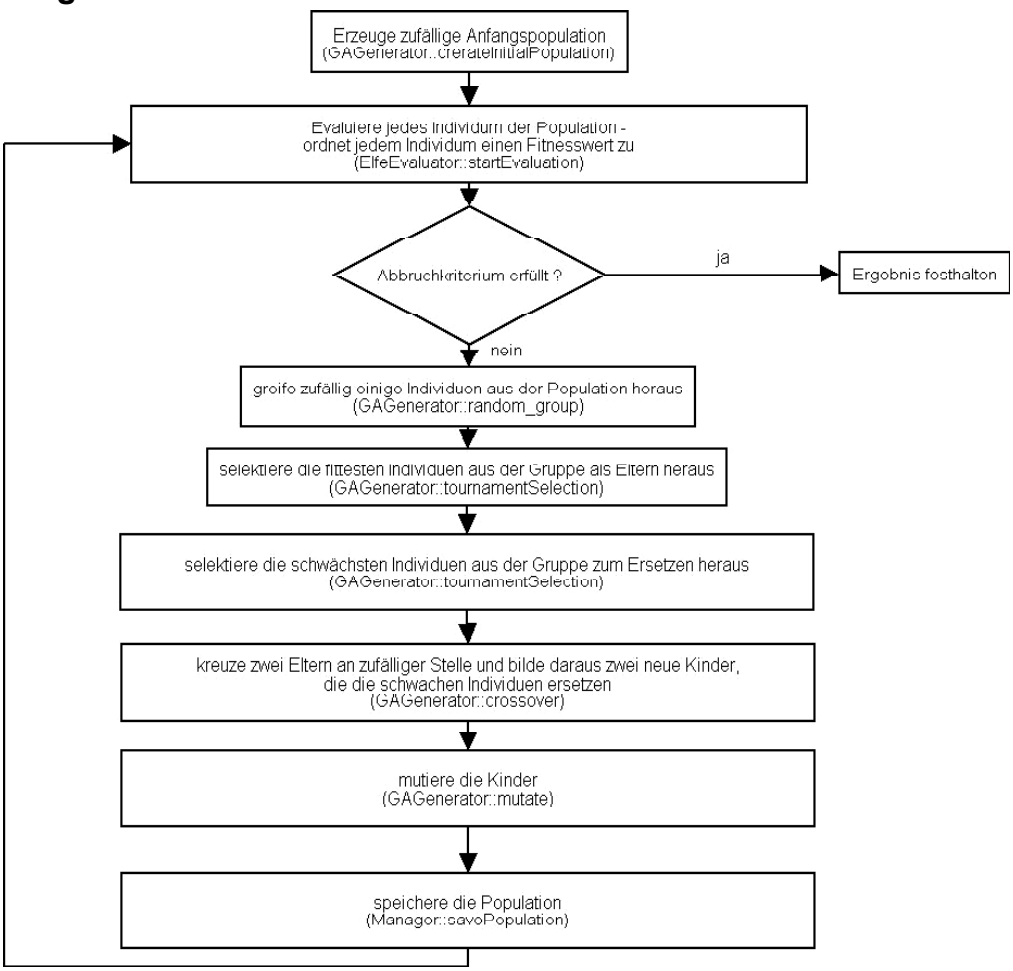
Bsp.: Individuum A hat 10 Sekunden zur Verfügung und schafft es so, den Fortbewegungsalgorithmus zwei mal hintereinander auszuführen. Individuum A legt in 10 Sekunden dadurch z.B. 10 Einheiten in x-Richtung zurück. Individuum B wird jedoch nur 5 Sekunden lang abgearbeitet, schafft demzufolge eine Strecke von 5 Einheiten in x-Richtung.

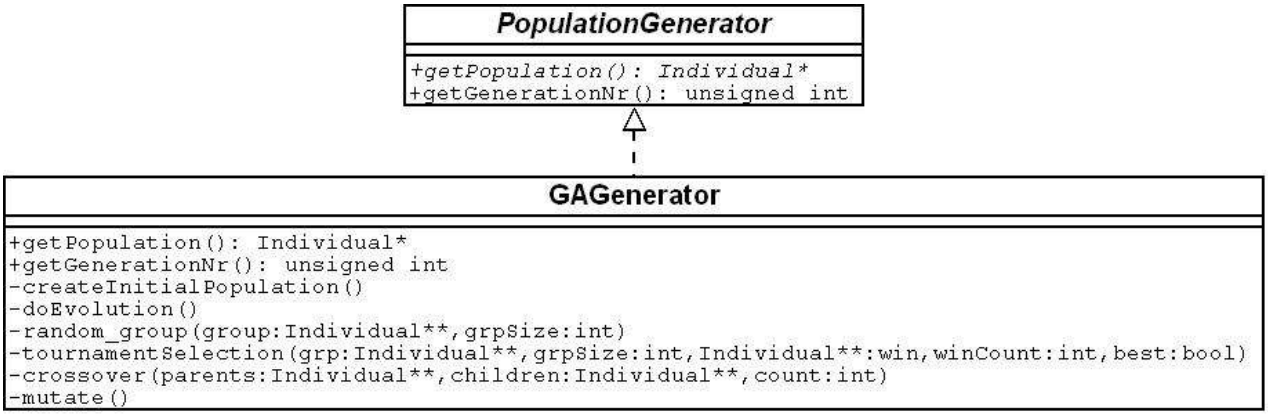
Rechenprobe 1:
X= 5, y=5, t=5 Fitness 1,41
X= 10, y=10, t=10 Fitness 1,41 => gleiche Fitness!

Rechenprobe 2:
X= -10, y=6, t=5 Fitness -10,6

X= -20, y=12, t=10 Fitness -10,6 => gleiche Fitness!

Populationsgenerator





Die Evolutionsklassen

Im Folgenden werden die Klassen beschrieben, die in der Simulation der Evolution eine wesentliche Rolle spielen.

Command

Die Klasse „Command" ist der Hauptbestandteil eines Individuums (siehe unten „Individual"). Eine Instanz von „Command" stellt einen Befehl dar, der aus drei Datenelementen besteht: *servo_nr*, *grad* und *sleep*.

- *servo_nr* speichert die Nummer des Servomotors, der gedreht werden soll.
- *grad* bestimmt auf wie viel Grad der durch servo_nr bestimmte Servomotor gedreht werden soll
- *sleep* sagt aus, wann das Drehen beginnen soll, nachdem der entsprechende Befehl von der „Elfe" empfangen wird.

Individual

Die Objekte der Klasse Individual sind die „Hauptdarsteller" in der Evolution. Die GA-Methoden operieren hauptsächlich mit Instanzen von „Individual".
Ein Individuum besteht aus einer Folge von „Command"-en und einem Fitnesswert. Die Anzahl der Befehle ist nicht fest vorprogrammiert, und variiert von Individuum zu Individuum.
Damit ein Individuum bewertet werden kann wird er zur „Elfe" geschickt, die seine Befehle ausführt. Nach der Ausführung wird das Individuum bewertet und sein Fitnesswert wird bestimmt.

GAGenerator

Die Klasse GAGenerator ist das Hauptelement, das die „Evolution" vorantreibt. GAGenerator besteht aus drei Datenelementen und mehreren Methoden. Die wichtigsten davon werden nachfolgend erläutert.

Datenelemente

population – ein Zeiger auf die Population, bestehend aus Instanzen von „Individual". Die Populationsgröße ist konstant. Die meisten Methoden in „GAGenerator" arbeiten direkt mit den Instanzen, die durch *population* adressiert sind.

iteration – speichert die Iterationsnummer, also wie viel Mal wurde die Population schon zur „Elfe" geschickt und anschließend bewertet.

mutate_rate – bestimmt wie hoch die Mutationsrate in der Population ist. *mutate_rate* variiert von 0.0 bis 1.0. Bei 0.0 findet keine Mutation statt. Bei 1.0 werden so viele Mutationen ausgeführt, wie es Individuen in der Population gibt.

Methoden

createInitialPopulation()

Diese Methode erzeugt eine zufällige Anfangspopulation fester Größe. Alle Methoden, die mit Individuen arbeiten, arbeiten mit dieser Population und verändern diese.

Vorantreiben der Evolution

Im folgenden werden die Methoden aus der Klasse „GAGenerator" beschrieben, die im Herzen der Evolution stehen:

- random_group()
- tournamentSelection()
- crossover()
- mutate()

void random_group(Individual** group, int size)

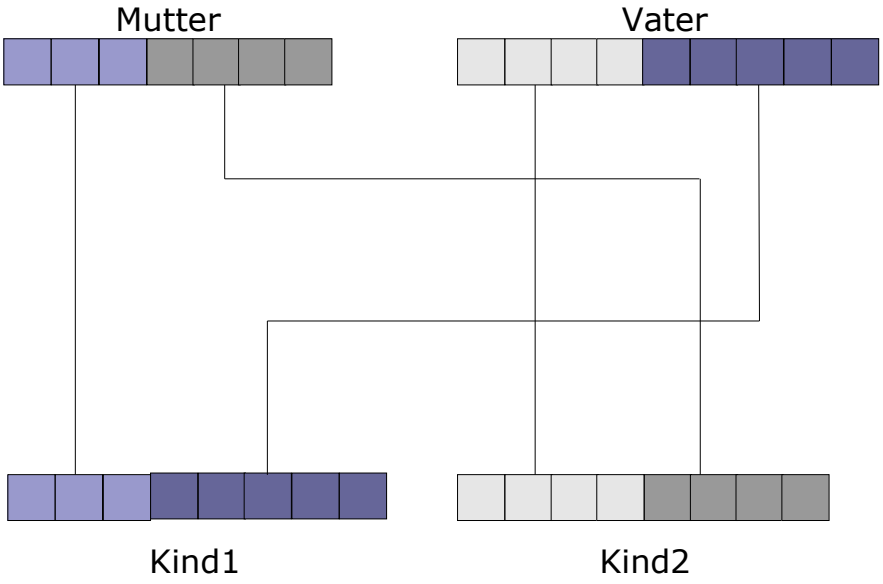
Die Methode „random_group" selektiert per Zufall eine Gruppe der Größe *size* aus Individuen aus der aktuellen Population. Die Adresse dieser Gruppe wird in *group* gespeichert. Sie ist der Pool, der zur Auswahl der Methode „tournamentSelection" steht.

void tournamentSelection(Individual** group, int size, Individual** winners, int n, bool parents)

Diese Methode selektiert eine Anzahl von *n* Individuen aus der Gruppe *group* der Größe *size* und speichert deren Adressen in *winners*.
Die Individuen werden nicht rein zufällig aus *group* ausgewählt. Die Wahrscheinlichkeit, dass ein Individuum ausgewählt wird wächst proportional bzw. umgekehrt proportional zu seinem Fitnesswert.
Die boolesche Variable *parents* legt fest, ob Eltern - starke Individuen, oder Loser - schwache Individuen ausgewählt werden.
Wenn *parents* den Wert true hat, haben Individuen mit höherem Fitnesswert bessere Chancen ausgewählt zu werden als solche mit niedrigerem Fitnesswert. Die Wahrscheinlichkeit ausgewählt zu werden wächst also proportional zum Fitness.
Hat *parents* den booleschen Wert false, sollen Loser aus der Gruppe *group* ausgewählt werden. D.h. schwächere Indidviduen (mit niedrigerem Fitness) werden vor den Stärkeren bevorzugt. Die Wahrscheinlichkeit ausgewählt zu werden wächst umgekehrt proportional zum Fitnesswert.

void crossover(Individual parents, Individual** kids, unsigned ind_count)**

Die Methode erzeugt aus *parents* „neue" Individuen und überschreibt mit diesen die Individuen in *kids*. Beide Pointer *parents* und *kids* zeigen auf eine Anzahl von *ind_count* Objekten. n Eltern erzeugen also n Kinder. *ind_count* muss gerade sein, sonst wird das Crossover abgebrochen.
Die „Paarung" erfolgt nach folgendem Schema:



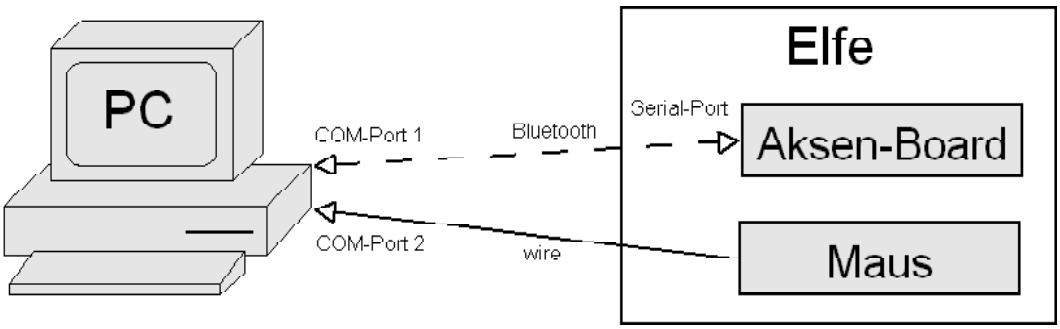
Ein Kästchen stellt einen Befehl dar (eine Variable vom Typ „Command").

Mutter und Vater werden in der Mitte durchtrennt. Die erste Hälfte der Mutter bildet die erste Hälfte von Kind1. Die zweite Hälfte der Mutter wird als zweite Hälfte Kind2 zugewiesen. Der Vater gibt seine erste Hälfte Kind2 und seine zweite Hälfte Kind1. Vater und Mutter bleiben unverändert. Ein Kästchen stellt einen Befehl dar (eine Variable vom Typ „Command").
Dieser Vorgang wird für alle Mutter - Vater Paare durchgeführt (deswegen muss *ind_count* gerade sein).

void mutate()

Die Methode "mutate" verändert die Population auf zufälliger Art und Weise. Die Klassenvariable *mutate_rate* (siehe oben **mutate_rate**) bestimmt wieviel Mal Individuen zur Mutation ausgewählt werden. Die Mutation erfolgt auf Bitebene. Bei jedem Individuum, das mutiert, wird zufällig aus jedem Befehl eine seiner Variablen *servor_nr*, *grad* oder *sleep* ausgewählt und ein zufälliges Bit dieser Variable wird gekippt.

Software-Architektur



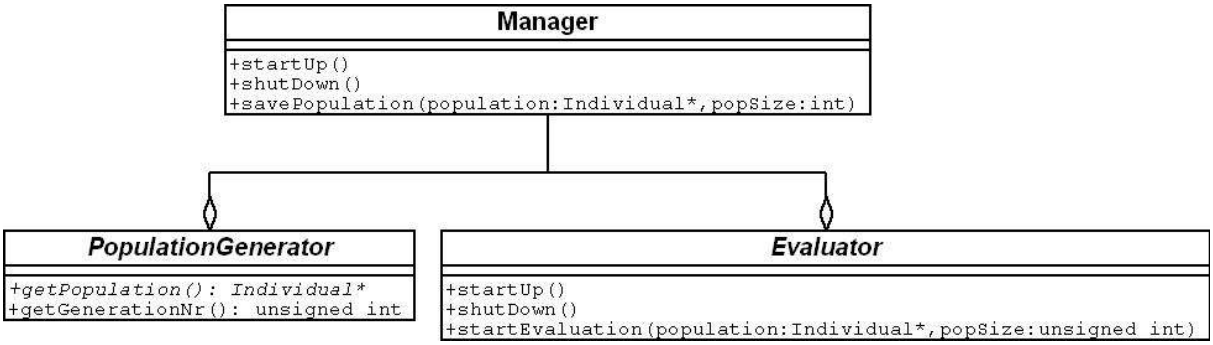
Bereits zu Beginn des Projekts stand fest, dass die Software in zwei Teilprojekten realisiert werden würde, zum Einen den Interpreter auf der Elfe, zum Anderen das GA-Programm auf dem PC. Weiterhin stellte sich heraus, dass eine Modularisierung des Projektes aus diversen Gründen von Vorteil sein würde. Das schließlich entwickelte Produkt untergliedert sich in folgende Bestandteile:

Auf der Elfe:
Interpreter [setzt Befehle in Bewegung um]

Auf dem PC:
Evaluator [berechnet Fitnesswerte]
GA-Generator [erzeugt Populationen]
Manager [managed Kommunikationsfluss]

Die Software-Architektur ist vollständig modular gehalten, sodass es zukünftig auf PC-Seite möglich ist, die Fitnessfunktion und den zugrundeliegenden evolutionären Algorithmus flexibel anzupassen bzw. auf der Elfe den Interpreter auszutauschen.

Der Manager ist das Bindeglied zwischen Populationsgenerator, Evaluator und Interpreter. Die Kommunikation zu Letzterem läuft über die serielle Verbindung, während Populationsgenerator und Evaluator direkt als lokale Klassen ansprechbar sind.



Praxiseinsatz

Das Logging ermöglicht es, dass Elfe tagelang der künstlichen Evolution ausgesetzt werden kann, ohne dass die Anwesenheit eines Entwicklers zwingend notwendig ist. Allerdings ist das Testareal für den Elfe-Roboter etwas klein, so dass ein gelegentliches manuelles Rücksetzen von Elfe auf die Startposition (Mitte des Testfeldes) erforderlich ist. Aufgrund von Kabelkonflikten entschied sich das Entwickler-Team, Elfe mit einem Aufbau zu versehen, der eine Stromzuführung über ein deckengeführtes Kabel ermöglicht.

Fazit

Die grundlegende Architektur konnte wie geplant realisiert werden. Mit dem erreichten Stand sind zukünftige Teams in der Lage, praktische Experimente mit Elfe durchzuführen und mit Hilfe des Loggings eine Auswertung auch nach mehrtägigen Tests vorzunehmen.

Verbesserungswürdig ist noch der GA-Generator, der aufgrund seiner Implementierung nicht mit negativen Fitnesswerten umgehen kann. Aus diesem Grund werden in der aktuellen Projektversion die real berechneten Fitnesswerte in den positiven Bereich gehoben.