Einführung

Als wir uns am Freitag, den 27.9., nach Adelsheim zur Science Academy 2004 aufmachten, waren wir wohl alle mit gemischten Gefühlen und ganz unterschiedlichen Vorstellungen und Erwartungen auf dem Weg zu unserem Kurs. Natürlich hatten wir schon einige erste Eindrücke während des Vorbereitungs-Wochenendes gesammelt, doch blieben immer noch viele offene Fragen: Komme ich mit meinen Kursmitgliedern und den Leitern auch über zwei Wochen gut aus? Werde ich diese lange Zeit überhaupt Spaß an unserem Projekt haben? Habe ich wirklich den richtigen Kurs gewählt? Und je weiter man sich Adelsheim näherte, desto mehr wuchs die Aufregung und Spannung... .
Zu diesem Zeitpunkt war uns allen noch nicht klar, dass wir unseren Kurs allein organisieren können und unsere Ziele selbst stecken würden. Außerdem dachte keiner, dass wir von unseren Kursleitern jegliche Freiheit der Kreativität bekämen und in allem unterstützt werden würden.

Wir trafen uns am gleichen Abend nach dem Abendessen das erste Mal im Kursraum. Nach langer Zeit waren wir endlich wieder unter uns Robotikern. Die meisten kamen sich ziemlich fremd vor, obwohl man sich in der Zeit, die zwischen dem Vorbereitungswochenende und der eigentlichen Akademie lag, im extra angelegten Forum der Science Academy, austauschen konnte. Doch dieses Gefühl der Fremdheit schwand bei uns allen sehr schnell wieder.

Wir wuchsen - in den leider viel zu kurzen zwei Wochen - zu einer richtigen Familie zusammen, genauso wie es Frau Greenway in ihrer Begrüßung am ersten Tag erwähnt hatte. Der ganze Robotikkurs, Teilnehmer sowie Leiter, standen einander immer mit Rat und Tat zur Seite und halfen, wo sie konnten. Das Wort „TEAMWORK'' war bei uns ganz großgeschrieben.

Unser Team bestand insgesamt aus 15 Robotikern: zwölf Kursteilnehmern, unsere drei Mädchen, Heidi Bersi, Jessica Matthias und Rebecca Zelt und unseren 9 Jungs, Christoph Amann, Marius Blaesing, Peter Holz, Daniel Lippert, Andreas Mayer, Andreas Müller, Pascal Weinert, Martin Wurditsch und Matthias Zimmermann. Natürlich nicht zu vergessen unsere drei hervorragenden und sehr kompetenten Kursleiter: Helge Peters, Matthias Taulien und Georg Wilke.

Wir begannen am ersten Tag sofort mit dem vollen Programm. Es wurde mit Lego hantiert und gebastelt. Schon am Abend hätte sich jeder, der nicht zum Robotikkurs gehörte, nicht mehr auf den Arbeitstischen zurecht gefunden. Für uns aber verlief alles sehr koordiniert, denn der Aufbau der Roboter erfolgte nach Anleitung, die uns unsere Kursleiter vorgelegt hatten. So bauten wir in Kleingruppen à zwei oder drei zunächst alle die gleichen Lego-Roboter, da wir uns erst einmal einarbeiten mussten und Erfahrungen im Programmieren sammeln wollten. Wir schlossen mit dieser Arbeit mehr oder weniger am nächsten Vormittag, Samstag den 28.9., ab.

Doch innerhalb der nächsten Tage sollten diese etwas primitiven Roboter nicht mehr zu erkennen sein. Denn die einzelnen Gruppen begannen schon kleine, kompakte Programme zu schreiben. Dafür wurde der eigene "Gruppen"-Roboter optimiert, also umgebaut. Doch denkt nicht, dass es bei drei oder vier Umbauten blieb! Nein! Würde man ein und denselben Roboter zu Beginn eines Tages neben den des Nachmittags stellen, könnte man den veränderten Roboter nicht mehr wieder erkennen. Dies zeigt nur mit wie viel Eifer und Spaß wir alle bei der Sache waren. Es wurden immer wieder neue, bessere Ideen in den Raum geworfen und oft auch umgesetzt.
Hatte jemand von uns ein offenes, ungeklärtes Problem, konnte er damit rechnen, dass sich mindestens einer der anderen neben ihn setzte und sich mit ihm so lange den Kopf zerbrach, bis eine Lösung auf dem Tisch lag. Diese war meist nicht nur zufriedenstellend, sondern perfekt. Es ist ganz klar, dass sich nicht jeder im Robotikbereich gleich gut auskennen kann. So bildeten sich schnell für alle verschiedenen Themenbereiche Spezialisten, die immer bereit waren, sich eines Problems anzunehmen.

Und aufgrund dieser „Rollenverteilung“ unterschieden wir nach einigen Tagen in unserem Kurs zwischen Lego-und FischerTechnik-Konstrukteuren, da wir beide Bauarten zur Verfügung hatten und die Interessen der Teilnehmer in beide Richtungen gingen. Ebenso wurde innerhalb dieser Gruppen nochmal unterschieden zwischen denen, die lieber programmierten und denen, die eher ihr handwerkliches Können nutzen wollten, also die Roboter bauten. So steuerte jeder von uns etwas zu allen Bereichen bei. Diese „Rollenverteilung“ hieß nicht, dass es bei uns im Kurs kleine Gruppen gab, die sich voneinander abschotteten. Im Gegenteil, wir arbeiteten alle unterstützt, jedoch nicht geleitet durch unsere Kursleiter, an einem gemeinsamen Projektziel auf eine uns selbst gesetzte Aufgabe hinaus. Die Teilung der Arbeit bedeutete für uns möglichst

effektives und schnelles Arbeiten mit den größtmöglichen Erfolgen.

Der Spaß kam in unserem Kurs nie zu kurz. Man kann mit Worten nicht einmal ansatzweise beschreiben, wie wir uns des öfteren vor Lachen gekrümmt haben oder was für Scherze und kleine liebenswerte Neckereien begangen wurden.

Es war immer eine total entspannte Arbeitsatmosphäre ohne jeglichen Zwang. Man hatte immer die Möglichkeit sich zurückzulehnen und ein paar mal kräftig und tief durchzuatmen, ohne Angst vor irgendeinem Druck zu haben. Doch jeden ergriff nach diesen wenigen Sekunden der eigene Ehrgeiz und Arbeitseifer wieder und er machte sich aufs Neue an seine Arbeit.

Schon nach den ersten Tagen begannen wir zu planen, jedoch erst am Donnerstag, den 2.9. wurde unsere Projektziel komplett ausgefeilt und fertig besprochen. Wir setzen uns selbst die Latte an Erwartungen sehr hoch und halsten uns viel Arbeit und Mühen auf.

Es entstanden zwei Lego-Gruppen, die aus Rebecca und Jessica und aus Andreas Ma. und Pascal bestanden, sowie zwei FischerTechnik-Gruppen, die sich aus Heidi, Matthias, Martin sowie Peter und Christoph, Andreas Mü., Marius und Daniel zusammensetzten.

Einer der vielen großen Höhepunkte unserer zwei Wochen war unter anderem die Rotation. Ein Tag, an dem jeder Kurs, also auch wir Robotiker, insgesamt vier Präsentationen am Vormittag des 4.9. abhielten. Wir verteilten uns also auf vier Gruppen, die sich gründlich auf ihre Vorträge vorbereiteten. Es wurde eine Power-Point-Präsentation für alle Gruppen erstellt, die von den Ideen aller lebte. Ganz klar gab es trotz dieser intensiven Vorbereitungen Erfolge wie auch Rückschläge. Mal funktionierte der Roboter nicht mehr so wie noch zwei Minuten zuvor, mal kam man einfach nicht so weit mit dem Programmieren, wie man sich es erwünscht und geplant hatte.
Zu wenigen, aber trotzdem einigen Zeitpunkten hätte man gerne die beiden Legoteile, die zusammenpassen sollten- es aber einfach nicht taten- hingeschmissen oder einfach auf das kleine Kreuzchen am rechten oberen Bildschirmrand geklickt und den Raum verlassen, um sich auf sein Bett zu legen und abzuschalten. Doch es blieb nur ein Gedanke.

Am Tag der Rotation waren viele Probleme noch nicht geklärt. Aber das war auch gar nicht nötig. Dem Publikum, also den anderen Teilnehmern der verschiedenen Kurse und ihren Leitern, sollte nur

ein kleiner Einblick in den derzeitigen Stand unserer Arbeit geben werden, was nach der erhaltenen Resonanz auch sehr gut geklappt hat.

Die richtige Aufregung gab es jedoch erst vor der Abschlusspräsentation. Die Vorbereitungen dieser Vorträge gingen schon einen Tag vor der Abschlusspräsentation am 7.9 los und wurden am 8.9. weitergeführt. Je mehr wir uns dem Ende der Akademie näherten, desto größer wurde unser Engagement und es wurden immer häufiger Überstunden bis in die späten Nachtstunden eingelegt. Für diese Vorträge teilten wir uns abermals ins Gruppen ein. Wie bei der Rotation war alles wohl überlegt. Es gingen immer zwei Spezialisten für Lego und zwei für FischerTechnik zusammen, damit jede Gruppe für sich in allen Themengebieten der Präsentation tiefgreifendes Wissen repräsentierte.

Da wir in den zwei Wochen so zusammengewachsen waren und uns perfekt aufeinander abgestimmt hatten, erreichten wir am Ende der Akademie unser Projektziel. Dazu waren wir sehr gut auf die Präsentation am vorletzten Tag vorbereitet, sodass sie bei allen drei Vorträgen immer stark besucht war. Diesmal zählte alles, da nicht nur die „eingeweihten“ Besucher der Rotations-Präsentation anwesend waren, sondern auch Eltern, Geschwister, Verwandte, Freunde und Bekannte.

Spätestens jetzt, als jeder einzelne seine Eltern nach den letzten zwei Wochen wiedersah, wusste er, dass es mit der Akademie bald vorbei sein würde. Es blieb uns noch der Mittwochabend, 8.9. und der nächste Tag bis mittags, an dem wir den ganzen Vormittag damit verbrachten unser organisiertes Chaos von Lego- und FischerTechnik-Teilen, Blättern und Unterlagen und Überresten der Präsentation aufzuräumen bzw. zu sortieren. Diese Zeit nutzten wir – und mit „wir“ sind schon im ganzen Artikel Teilnehmer UND Leiter angesprochen – noch einmal, um gemeinsam über die vergangen zwei Wochen zu schwärmen, nachzudenken, darüber mit den anderen zu reden und zu lachen... . Man kann behaupten, dass keiner das letzte Mal den Robotikraum verließ, ohne sich noch mal umzudrehen und dabei ein Gefühl des Verlusts gespürt zu haben.

Innerhalb der Akademie haben wir alle so viel dazu gewonnen, dass nichts mit dem Abschied nach den zwei Wochen verloren ging. Es wurden dicke Freundschaften innerhalb und natürlich auch außerhalb unseres Kurses geschlossen. Es wurden Erfahrungen in Robotik und der Programmierung

gesammelt, sowie in der Zusammenarbeit, Kooperation und Arbeitsteilung mit anderen. Jeder war ziemlich traurig als es dann endgültig „Tschüß“ hieß und es war ein komisches Gefühl, alle neu gewonnenen Freunde ein letztes Mal zu umarmen und dabei zu wissen, dass man sie erst in zwei Monaten und dann nur noch selten wiedersehen würde.

Unsere einstimmige Meinung ist auf jeden Fall, dass die zwei Wochen viel zu rasch vergangen sind und dass es schade war, keine Verlängerung buchen zu können.

Das Team des Robotik-Kurses bestand aus 3 Leitern, 9 Jungen und 3 Mädchen.

Am Fr., den 27.8., bauten wir in kleinen Gruppen Lego-Roboter nach Anleitung.

Am Sa., den 28.8., freundeten wir uns ein wenig mit Java an und programmierten kleine Programme, so dass unsere Roboter mit Hilfe von Lichtsensoren an einem Abgrund halten oder zurückfahren konnten.

Am So., den 29.8., haben wir unsere Roboter so umgebaut und programmiert, dass sie mit Hilfe der Lichtsensoren an einer Kante entlang fahren und mit Hilfe eines Tastsensor Banden erkennen und danach wenden konnten.

Am Mo., den 30.8., trennten sich die Gruppen nach Lego und Fishertechnik. Dann erfolgten wieder Umbauten, so dass unsere Roboter auch die neuen Aufgaben, die wir nun programmieren konnten, technisch erfüllten. Außerdem sammelten wir Ideen für unser Projektziel.

Am Di., den 31.8., bauten wir unsere Roboter erneut um, perfektionierten unsere Programme oder schrieben sie weiter.

Am Mi., den 1.9., gingen wir unseren Arbeiten nach und sammelten erneut Ideen für unser Projektziel.

Am Do., den 2.9., entschlossen wir uns für unser endgültiges Projektziel, wobei wir uns noch nicht sicher waren, ob unsere Roboter diese Aufgaben überhaupt erledigen konnten.

Am Fr., den 3.9., bauten und programmierten wir weiter, damit unsere Roboter eben dieses Projektziel auch erledigen konnten. Es wurden die Namen der Roboter festgelegt. So entstanden 2 Lego-Roboter: „Snoop“ und „Sekus“ (Spedition es kracht und scheppert) sowie 2 Fishertechnik-Roboter namens „Gustav mit der Hupe“ und „Frank“
Die Präsentation vor den anderen Kursen (Rotation) wurde vorbereitet.

Am Sa., den 4.9., Rotationstag mit Besuch der anderen Kursen.

Am So., den 5.9., gaben wir unseren Robotern noch die letzten Feinschliffe.

Am Mo., den 6.9., Ausflug nach Heidelberg mit Besuch der MS-Technik und der Uniklinik.

Am Di., den 7.9., wurden die letzten Probleme gelöst, da am folgenden Tag die Abschluss-Präsentation stattfand.

Am Mi., den 8.9., bereiteten wir morgens die Präsentation vor. Präsentationstag, an dem wir allen, auch unseren Familien, alles vorstellten.

Am Do., den 9.9., mussten wir die Lego- und Fishertechnik-Teilchen sortieren.

Erwähnenswertes zu Java:

Java gibt es offiziell seit dem 23. Mai 1995. D.h. es ist ein sehr junge und moderne Sprache, die immer populärer wird. Sie wurde von der Firma „Sun“ entwickelt.
Deshalb haben wir uns entschieden, unsere selbstkonstruierten Roboter mit Java zu programmieren.
Obwohl wir Laien waren, konnten wir uns schnell in Java einfinden.
Hinzu kommt, dass diese Sprache eine plattformunabhängige Programmiersprache ist, d.h. die mit Java erstellten Programme laufen auf allen gängigen Betriebssystemen

Grundwissen zur Java-programmierung:

Unsere Programme haben wir mit JCreator, einem einfachen Editor geschrieben. Dieser beinhaltet kleine Programmierhilfen und die nötigen Schaltflächen, zum Kompilieren (Übertragen des Programms).
Beim JCreator wird der Bildschirm in vier Bereiche eingeteilt (Abbildung 1) :

- Links oben werden die zum Projekt oder „Workspace“ gehörenden Klassen (Dateien) angezeigt.
- darunter befindet sich die Anzeige aller Methoden und Variablen der momentan ausgewählten Klassen.
- am unteren Fensterrand entlang laufend werden Statusmeldungen einer kompilierten Klasse oder eines Projekts ausgegeben.
- ganz rechts im größten Feld, auch als Editorenfenster bekannt, wird der „Quellcode“ hineingesetzt.

Als „Quellcode“ bezeichnet man ein Programm, das für uns Menschen lesbar geschrieben ist. Ist der Quellcode fertig gestellt, wird er kompiliert, d.h. der Text wird in eine für den Computer lesbare Zeichenabfolge übersetzt.

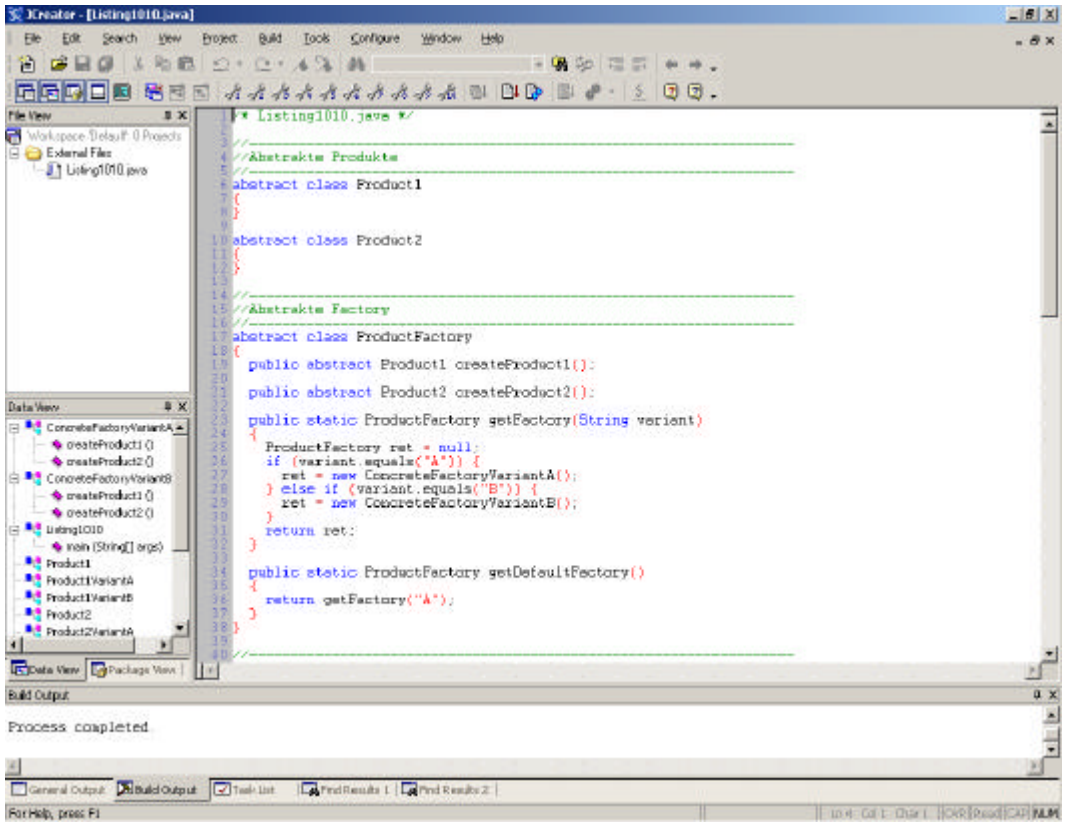


Abbildung 1

Diese wird in einer sogenannten class-Datei gespeichert. Mit dem „Java-Interpreter“, einem weiterem Programm, kann diese class- Datei gelesen, und anschließend ausgeführt werden.

Objektorientierte Java-Programme:

Ein solches Programm besteht zunächst einmal aus einer oder mehreren Klassen. Manchmal fasst man mehrere Klassen zu einem Package (Packet) zusammen.
Eine Klasse kann man am besten mit einem

Bauplan für ein Objekt auffassen. Darin wird beschrieben, welche Eigenschaften und Funktionsweisen ein Objekt hat. So wird beispielsweise im Bauplan festgelegt, wie der Motor eines Autos zu bauen ist und zu funktionieren hat.

Ein Objekt kann verschiedene Arten von Variablen und Methoden enthalten.

Verwendung von Variablen:

Variablen dienen zum Abspeichern von Werten.
Jede Variable muss von einem bestimmten Datentypen sein. Diese Typen geben an, aus welcher Art von Wert eine Variable besteht. Es gibt beispielsweise „Ganzzahl"- Typen, „Gleitkomma"- Typen, „Wahrheitswerte"- Typen und „Zeichen"- Typen.

Baut man einen realen Motor nach diesem Bauplan, kann man ihn z. B. laufen lassen. Ebenso erzeugt man in Java Instanzen von Objekten, mit denen man dann arbeiten kann.

Jede Variable muss genau einem Datentyp zugeordnet sein. Man nennt das Deklarieren. Ebenso muss jede Variable einen Anfangs- oder Startwert haben. Dies nennt man Initialisieren.

Beispiel für Deklaration:
```
int breite;
```
nt (= Integer) besagt, dass die Variable einen Wert von den Ganzzahl- Typen annehmen soll und dass sie den Namen „breite" erhält.
Das Semikolon muss immer am Ende des Ausdrucks stehen. Es besagt, dass hier ein Befehl endet. Anschließend wird die Variable initialisiert:
```
breite = 2;
```
Das bedeutet, dass der Variable namens „breite" ein Wert 2 zugewiesen wird.
Man kann beide Vorgänge auch zusammenfassen.
Statt:
```
int breite;
breite = 2;
```
schreibt man kürzer:
```
int breite = 2;
```

Verwendung von Methoden:

In Methoden werden die Aktionen von Objekten festgelegt.
Um Methoden verwenden zu können, muss man sie deklarieren.
Beispiel:
```
void rechteckErzeugen() {
     Anweisungen
}
```

Im Klammernpaar könnte man Parameter an die Methode übergeben. Die einzelnen Anweisungen, die in der Methode ausgeführt werden sollen, setzt man in den Zwischenraum der geschweiften Klammern, die sich immer hinter der Deklaration befinden.

Textausgabe auf den Bildschirm:

Möchte man jedoch nur einen Text in der Konsole ausgeben, so gibt es die Möglichkeit die schon existierenden Methoden

```
System.out.println() ;
```

bzw.

```
System.out.print();
```

zu verwenden.
Beide Methoden können sowohl Zeichenketten zwischen zwei Anführungszeichen als auch Variablen ausgeben.
Der Unterschied beider besteht allerdings darin, dass die „println“- Methode bei der Ausgabe im Gegensatz zur „print“- Methode automatisch einen Zeilenumbruch vornimmt.
Den Text, bzw. die Variable, übergibt man an die Methode als Argument.

Beispiel:

```
System.out.println("Hallo");
```

Oder:

```
int i = 2 ;
System.out.println(i) ;
```

Die Textausgabe lautet hierbei :

```
Hallo
```

bzw.

```
2
```

Kommentare:

Kommentare können an beliebigen Stellen im Quellcode auftauchen. Sie geben uns Menschen Anhaltspunkte, was im darauf folgende Abschnitt im Programm geschieht.
Die Kommentare spielen beim Kompilieren keine Rolle, denn sie werden vom „Compiler“ vollkommen ignoriert.

Es gibt drei Arten von Kommentaren:

- Einzeilige Kommentare:
- sie beginnen mit //
- Mehrzeilige Kommentare :
- diese beginnen mit /* und enden mit */
- Dokumentationskommentare:
- sie beginnen mit /** und enden mit */ . Diese Kommentare dienen dazu, Programme im Quelltext zu dokumentieren. Mit Hilfe von „javadoc“, einem Programm, werden die Kommentare aus dem Quellcode entnommen und in ein HTML Dokument umgewandelt. Das erleichtert anderen sichtlich die Arbeit, wenn sie sich mit einem unbekannten Programm vertraut machen wollen.

```
001 public class TestQueue
002 {
003   public TestQueue()
004   {
005     /*Intialisierungen, z.B. Erzeugen
006       eines zu testenden Queue-Objekts*/
007   }
008
009   public void test1()
010   {
011     //Erste Testmethode
012   }
013
014   public void test2()
015   {
016     //Zweite Testmethode
017   }
018
019   //...
020 }
```

Die main-Methode:

Um ein Programm starten zu können, benötigt der Javainterpreter die „main“- Methode.
Das ist das erste Ziel, dass der Interpreter ansteuert. Sie sieht wie folgt aus:

```
public static void main (String [] args) {
       ...
}
```

Nun kann man die Methoden, die aufgerufen werden sollen, der Hauptmethode übergeben, indem diese in das Feld zwischen den geschweiften Klammern gesetzt werden.

So schnell können die ersten Gehversuche in Java unternommen werden!

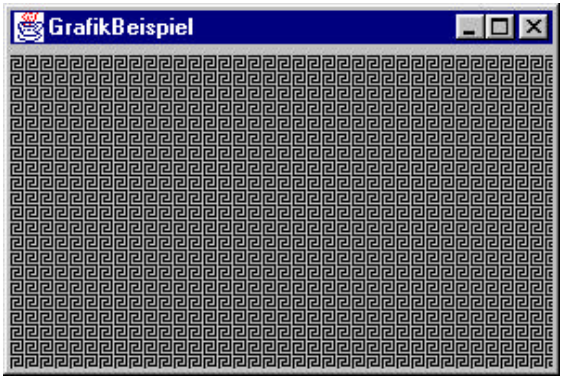
```
001 /* Hello.java */
002
003 public class Hello
004 {
005  public static void main(String[] args)
006   {
007     System.out.println("Hello, world");
008   }
009 }
```

Grafische Oberfläche von Java-Programmen

Fenster

Natürlich kann man mit Java auch normale Windowsfenster programmieren. Standardmäßig ist in der linken oberen Ecke, das Icon von Sun. Dann folgt der Fenstertitel und in der rechten oberen Ecke befinden sich die 3 Buttons zum Minimieren, Verkleinern und zum Schließen. Bei dem Button zum schließen ist aber zu beachten, dass man seine Funktion extra programmieren muss und diese nicht schon festgelegt ist. Denn es können verschiedene Ereignisse auftreten, wenn man auf ihn klickt, z. B. kann nur das Fenster oder das ganze Programme geschlossen werden. Dann kann man auch noch die Größe, Position und die Hintergrundfarbe dieses Fensters programmieren. Dieses Fenster kann natürlich als ein vorgefertigtes Package importiert werden. Solch ein Fenster bietet natürlich viel einfachere Möglichkeiten zur Ein- und Ausgabe von Daten, als die vorhin verwendete Textausgaben.

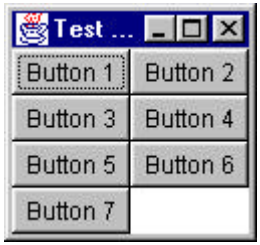
Das Fenster könnte jetzt schon Grafiken ausgeben:



Aber noch fehlt eine Eingabemöglichkeit für solch ein Fenster.

Buttons

Nun kann man in solch ein Fenster auch noch Buttons einfügen. Über Buttons kann man dem Computer Befehle erteilen. Auch bei Buttons gibt es Standardpakete, mit denen sich auch hier wieder an Programmierarbeit sparen lässt. Buttons haben eine Aufschrift und man kann auf sie klicken. Doch bis jetzt passiert noch nichts. Dazu braucht man zuerst einmal einen „Action Listener“, der den Button abhört. Wenn er merkt, dass z. B. der Button gedrückt wird, gibt er ein Signal an das Programm weiter. Dieses kann nun ein bestimmtes Ereignis auslösen, z .B. die Farbe des Hintergrunds ändern. Die Größe und Position eine Buttons kann natürlich auch verändert werden.



Textfelder

Auch in einem Fenster kann man Textfelder erzeugen, in die Informationen mit der Tastatur eingegeben und verarbeitet werden können. Hier braucht man einen „Event Listener“, welcher die Ereignisse im Textfeld überwacht, wenn nach der Eingabe etwas Bestimmtes geschehen soll. Bei Textfeldern kann man auch einstellen, ob diese Textfelder „visible“ oder „invisible“ sein sollen, also sichtbar oder unsichtbar. Wenn das Textfeld visible ist, kann man etwas hineinschreiben. Wenn es aber invisible ist, kann man beim Programmieren einen Text vorgeben. Dieser kann aber, während das Programm läuft, nicht mit der Tastatur verändert

werden. Sie sehen eigentlich aus wie Textfelder, nur dass sie grau unterlegt sind.
Jedoch sind in dem Beispiel hier keine invisiblen Textfelder, denn
Name, Vorname und Ort sind „Label“.

Label

Ein Label dient zur Beschriftung von Dialogboxen. Ein Label enthält eine Zeile Text, der vom Programm, aber nicht vom Benutzer geändert werden kann.

Panel

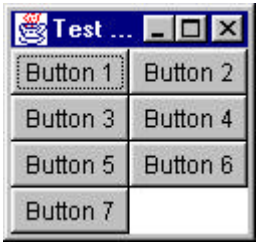
Panel sind kleinere Fenster , die ein großes Fenster unterteilen. Sie ermöglichen eine schönere Gliederung innerhalb des Fensterinhaltes und erhöhen damit die Anschaulichkeit eines Programms.

Layouts

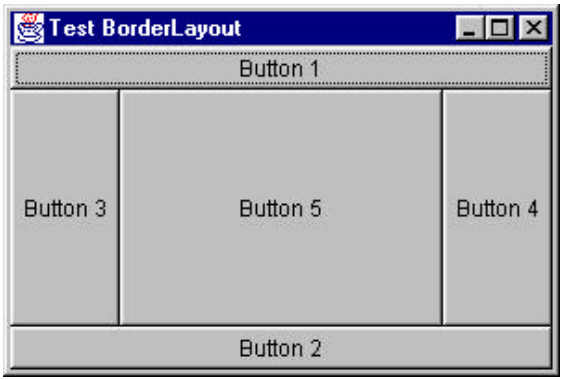
Ein Fenster kann verschiedene Layouts haben. Das heißt die Symbole innerhalb eines Fensters können auf verschiedene Arten angeordnet werden, z. B. als eine Tabelle. Auch hier gibt es wieder Standardlayouts, die einem die Arbeit der Layoutprogrammierung erleichtern:

Das „Flow“-Layout ordnet alle Elemente des Fensters in einer Reihe an. Wenn diese voll ist, wird in der nächsten Reihe fortgefahren.

Das „Grid“- Layout ordnet die Elemente des Fensters in einem rechteckigen Gitter an, wobei alle seine Gegenstände eine bestimmte Größe haben.



Das „Border“-Layout teilt das Fenster in 5 Bereiche auf, nämlich in die vier Ränderbereiche und in das Zentrum.
Hier kann man die Position der Buttons einfacher angeben, wie bei den vorhergehenden Layouts. Die Positionszuweisung erfolgt über die Wörter Norden, Süden, Westen, Osten und das Zentrum.



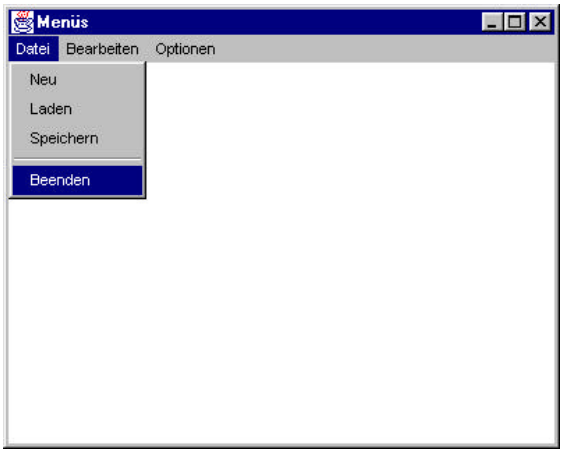
Mit diesen drei genannten Layouts und den verschiedenen Panels in einem Fenster kann man die Grafik und die Übersichtlichkeit eines solchen Programms um einiges verbessern. Natürlich gibt es noch mehr Layouts und man kann auch selbst welche programmieren, aber diese drei reichen eigentlich aus, um ein grafisch ansehnliches Programm zu erstellen.

Weitere Elemente eines Fensters

In einem Fenster kann es natürlich noch andere Gegenstände geben, wie die bereits genannten, z.B. eine „Scrollbar“ (Schiebregler), Kästchen zum Ankreuzen, Felder mit Auswahlmöglichkeiten, usw. Mit diesen Elementen können die Möglichkeiten eines Fensters noch erweitert werden.

Das Menü

Eine Menüleiste stellt das Hauptmenü eines Fensters dar. Sie befindet sich unterhalb der Titelleiste am oberen Rand des Fensters und zeigt die Namen der darin enthaltenen Menüs an. Auch Menüs geben mit Hilfe eines „Action Listeners“ Nachrichten an das Programm weiter. So können dann beispielsweise neue Fenster geöffnet werden…



Nun hätte man schon die wichtigsten Bestandteile, die man zur Programmierung von Fenstern benötigt und wäre in der Lage seine Programmierkenntnisse zu erweitern...

Vertiefung der Programmier-kenntnisse:

Sichtbarkeitskennzeichen von Methoden und Variablen:

Um Methoden oder Variablen nur in ganz bestimmten Bereichen aufzurufen zu können oder anders genannt sichtbar zu machen, können sie mit Adjektiven versehen werden.

public:
Steht vor einer Methode oder einer Variablen public, so kann immer auf sie zugegriffen werden.

private:
Steht vor einer Methode oder einer Variablen private, so ist diese nur innerhalb der Klasse selbst sichtbar. Nicht einmal eine Unterklasse (siehe Vererbung) kann auf sie zugreifen. Das heißt die Methode oder Variable kann nur innerhalb der Klasse aufgerufen werden , in der sie auch deklariert und initialisiert wurde.

protected:
Steht vor Methoden oder Variablen protected, ist nur der Zugriff auf sie von den denjenigen Klassen erlaubt, die sich im selben Paket befinden und von allen Unterklassen, die von dieser Klasse, in denen sich die Methoden bzw. Variablen befinden, abgeleitet sind.

Verzweigungen:

Diese dienen dazu, bestimmte Programmteile auszuführen, wenn die vorgegebene Bedingung eintritt. Die Verzweigungen stehen innerhalb einer Methode.

Die if – Anweisung :

```
if(Bedingung) {
     Anweisung1;
     Anweisung2;
     ...
}
else {
     Anweisung3;
     Anweisung4;
     ...
}
```

Der „Ausdruck“ wird als erstes bei der Laufzeit des Programms ausgewertet. Ergibt diese Auswertung einen wahren Wert, wird die darauf folgende „Anweisung“ ausgeführt. Ansonsten wird diese „Anweisung“ nicht ausgeführt und sondern die else Anweisung.

Schleifenanweisungen:

Um nicht ständig alles doppelt und dreifach abzutippen, falls eine Wiederholung erwünscht ist, gibt es Schleifenanweisungen, die ebenfalls innerhalb von Methoden stehen.

Die while- Schleife:

```
while (Bedingung) {
     Anweisung1 ;
     Anweisung2;
     ......
}
```

Der „Ausdruck“ wird wieder als erstes bei Laufzeit geprüft, ist das Ergebnis wahr, wird die „Anweisung“ ausgeführt, solange bis beim Testen des Ausdrucks ein wahres Ergebnis geliefert wird . Ergibt das Resultat einen falschen Wert, wird die Schleife nicht

wiederholt und gleich mit der ersten Anweisung nach der Schleife fortgefahren.

Die do- Schleife:

```
do {
     Anweisung1;
     Anweisung2;
     ...
}
while (Bedingung);
```

Diese Schleife wird mindestens einmal ausgeführt, denn hier wird zuerst die „Anweisung“ ausgeführt und erst danach wird der „Ausdruck“ geprüft. Ergibt dieser beim Durchlauf ein wahres Resultat, wird die „Anweisung“ wiederholt . Wird ein falsches Resultat geliefert, kann nach mindestens einem Durchlauf mit der ersten Anweisung nach der do- Schleife weiter gemacht werden.

Die for- Schleife:

```
for (init; test; update) {
     Anweisung1;
     Anweisung2;
     ...
}
```

Zuerst wird der „init “- Teil befolgt. Dieser dient dazu eine Initialisierung durchzuführen.
Darauf folgt der „Test “- Part. Dieser Ausdruck wird wieder getestet. Ist das Ergebnis daraus wahr, wird die „Anweisung“ darunter ausgeführt, ansonsten wird abgebrochen und mit der ersten Anweisung nach dieser Schleife weitergemacht. Solange allerdings ein wahrer Wert zurückgeliefert wird, wird die Schleife wiederholt bearbeitet.
Fehlt dieser „Test “- Teil , setzt der Compiler von sich aus ein konstantes wahres Ergebnis ein.
Der „update“ - Teil wird immer bei jeder Schleifenwiederholung nach dem „Test“-Teil durchlaufen.
Dieser Part der Schleife dient dazu, den Schleifenzähler zu verändern.

Konkretes Beispiel:

```
for (int i = 2; i <= 10; i ++) {
     Anweisung1;
     Anweisung2;
     ...
}
```

Die ganzzahlige Variable „i“ bekommt den Wert 2 zugewiesen. Solange i kleiner oder gleich 10 ist, soll die „Anweisung“ ausgeführt werden. i++ besagt, dass bei jeder Schleifenwiederholung die Variable um eins höher gesetzt werden soll.

Arrays:

In einem „Array“ (= Feld) können Listen von Werten des gleichen Datentyps gespeichert werden.

Arrays könnte man beispielsweise verwenden, wenn man mit einem Roboter an eine bestimmte Position fährt und diese abspeichern will. Diese Werte der Positionen könnten in den Arrays, also in den Feldern, abgespeichert werden.
Auch bei Arrays ist eine Deklaration erforderlich.
Bei dieser wird der Datentyp der Variable, mit der gearbeitet werden soll, ein eckiges Klammernpaar und die Bezeichnung des Feldes aufgelistet.

Beispiel:
```
int [ ] position;
```

Arrays werden wie Java Objekte behandelt.
Aber nur durch die Deklaration ist das Array noch nicht im Speicher angelegt. Das Array- Objekt muss erst noch erstellt werden. Dies geschieht mit Hilfe eines new-Operators.

Beispiel:
```
position = new int [10];
```

Hier wird also ein Feld namens `position` erzeugt, dass 50 freie Plätze zum einspeichern von ganzzahligen Variablen hat.
Die Deklaration und die Erstellung eines Array-Objekts können auch Gleichzeitig praktiziert werden.

Beispiel:
```
int [ ] position = new int [10] ;
```

Bei dieser Erstellung des Array-Objekts wird die Anzahl der Felder mit dem Grundwert des jeweiligen Datentyps initialisiert.

Bei dem obigem Beispiel wird ein „integer“ verwendet, dass heißt, dass der Grundwert des Datentyps 0 ist, also hat das erste Feld die Bezeichnung 0.
Der Zugriff auf die einzelnen Felder erfolgt über die Bezeichnung des Feldes. Die Zählung geht hierbei von 0 bis Anzahl minus 1, also 9.

Beispiel:
```
position[0] = 7;
position[1] = 5;
...
position[9] = 1 ;
```

Dem ersten Feld mit der Bezeichnung 0 von Position1 wird der Speicherwert 7 zugewiesen, dem zweiten Feld mit der Bezeichnung 1 von Position1 wird der Speicherwert 5 zugewiesen, etc.

Vererbung:

Vererbung ist sinnvoller und schneller, wenn man vorhandene und evt. schon getestete alte Programmteile für neue Programmteile verwenden kann, anstatt jedes Mal alles komplett neu zu schreiben.
Dies geschieht dadurch, dass bereits vorhandene Klassen erweitert werden. Die erweiterten Klassen, die als Unterklassen bezeichnet werden, erben dabei alle Variablen und Methoden der abgeleiteten Klasse, die auch als Basis- oder Vaterklasse bezeichnet wird.

Die Deklaration einer Vererbung geschieht folgendermaßen:

Unterklasse `extends` Basisklasse

Konkretes Beispiel :
```
Schueler extends Mensch
```

Hier wird „Schueler“ von Mensch abgeleitet, „Schueler“ erbt demnach alle Eigenschaften, die auch ein Mensch hat.
Zusätzlich können in „Schueler“ aber noch neue Eigenschaften festgelegt werden, die nicht jeder Mensch besitz. Zum Beispiel kann in der Klasse „Schueler“ eine Klassenstufe angelegt werden, die angibt , welche Klasse der Schueler besucht. Bei einem Menschen ist es allerdings nicht festgelegt, ob er überhaupt noch in die Schule geht.

Java und unsere Roboter

Die Lego-Roboter

Bei den Lego-Robotern wird das gesamte Programm mit einem Infrarotsender auf einen RCX–Baustein übertragen. Dieser ist ein eigenständiger Microcomputer mit Prozessor und internem Speicher.
Das Programm muss jedoch zuerst auf einem richtigen Computer geschrieben werden, da der Microcomputer keine Möglichkeiten zur Texteingabe hat. Anschließend compiliert man das Programm und überträgt dann die so erzeugte class-Datei auf den RCX. Damit der RCX das Programm auszuführen kann, musste ihm zuvor ein Betriebssystem überspielt werden, das man als Firmware bezeichnet. Der RCX hat 3 Eingänge, an die man Sensoren anschließen kann und 3 Ausgänge, an die man Motoren oder Lämpchen anschließt. Ein Java-Programm muss auf Eingangssignale der Sensoren reagieren und sie entsprechend auf die Ausgänge weiterleiten. So kann das Programm z. B. dem linken Motor sagen, dass er sich nach vorne zu drehen hat. Nach und nach entwickelt sich so ein recht umfangreiches Programm für die Lego-Roboter.
Dieses Programm wird mit der Zeit natürlich immer komplizierter, wenn man z. B. den Roboter noch einige Sekunden warten lassen muss, bis er einen Befehl ausführt. Noch schwieriger ist es, wenn man mit Hilfe eines Lichtsensors die Farbe der Tonnen unterscheiden muss. Damit dies überhaupt funktionieren kann, muss der Lichtsensor ständig Prozentwerte der Lichthelligkeit zurückliefern. Wenn nun der Schwellwert , also eine bestimmte Prozentzahl, untertreten wird, muss ein Ereignis ausgelöst werden. Allerdings besteht hier ein großes Problem, denn der Schwellenwert kann sich verändern , wenn z. B. das Licht angeschalten weird oder die Sonneneinstrahlung an manchen Tagen stärker ist. Dies hat zur Folge, dass der Schwellenwert nicht untertreten und das Ereignis nicht ausgelöst werden kann.
Das Lego-Programm wird mit einem Knopf auf dem Baustein des Roboters gestartet und auch wieder beendet.

```
package verhalten;
import josx.robotics.*;

/**
 * Solange keine Tonne da ist, sucht Snoop.
 */
public class Suche implements Behavior,
SnoopConstants {

/**
 * Die Kontrolle soll nur übernommen werden, wenn keine
Tonne da ist.
 */
   public boolean takeControl() {
      return (!Init.schwarzeTonneDa &&
              !Init.gelbeTonneDa
              &&!Init.aufLinie);
       }

/**
 *Snoop fährt geradeaus.
 */
    public void action() {
        mLinks.forward();
        mRechts.forward();
    }

/**
 *Motoren werden auf Leerlauf geschaltet
 */
  public void suppress() {
        mLinks.flt();
        mRechts.flt();
    }
}
```

Die Fischertechnik-Roboter

Die Fischertechnik-Roboter sind im Gegensatz zu den Lego-Robotern ständig mit dem Laptop verbunden und brauchen auch noch eine grafische Oberfläche auf dem Computer zur manuellen Bedienung. Die Bewegungen des Roboters können gestartet, überwacht und auch beendet werden. Was hier natürlich die Feineinstellung der Roboter um einiges erleichtert. Denn man kann die benötigten Stellungen der Achsen einzeln anfahren und dann die jeweiligen Positionen speichern.

Bei einem Fischertechnik-Roboter werden bestimmte Arbeitsvorgänge in Klassen aufgefasst und programmiert. Zum Beispiel die Klasse „Tonne greifen“. In dieser Klasse können die speziellen Methoden und Variablen festgelegt werden, die dafür zuständig sind, dass letztendlich die Tonne vom Roboter gegriffen werden kann. Diese Klasse wird von einer, ihr übergeordneten, allgemeineren Klasse aufgerufen. Innerhalb der Unterklasse stehen Befehle in den Methoden , die zum Beispiel für die Steuerung der Motoren zuständig sind. Auch bei Fischertechnik werden Action Listener verwendet, welche die verschiedenen Taster überwachen und Meldung an das Programm bei deren Drücken weitergeben. Denn die Taster sind meist an Anschlagpunkten angebracht und beim Tasterdrücken ist meist ein Anhalten des Roboters erwünscht. Dabei hat der Action Listener die Aufgabe, das Drücken des Tasters zu beobachten und gegebenenfalls die Methoden aufzurufen, die für das Anhalten der Motoren zuständig sind.

Ein Fischertechnik-Roboter muss auch z. B. die Position einer Tonne speichern können. Dazu muss wie vorhin schon bei dem Thema „Arrays“ erwähnt, ein sogenanntes Speicherfeld erzeugt werden. So kann beispielsweise eine gelbe Tonne eingeliefert und deren Position gespeichert werden. Will man auf diese Tonne zugreifen , ruft
das Programm in einer speziell dafür programmierten Methode die Speicherplätze auf und der Roboter kann die Tonne ausliefern.
Neben diesen wichtigen Dingen können die Roboter zusätzlich mit Lichter ausgestattet werden und bei bestimmten Ereignissen, wie z.B. das Greifen einer Tonne, diese dann zu leuchten bringen. Dieses wird durch if-Anweisungen im Programm ausgelöst.

```
public void fasseTonneGelb(){
  int fachPos = 0;
  int pos = 0;
     System.out.println("fasse      gelbe
     Tonne");
      while(pos < 4 && fachBelegt[pos]) {
                pos++;
      }
  switch (pos){
    case 0: fachPos = 0; break;
    case 1: fachPos = -7; break;
    case 2: fachPos = -14; break;
    case 3: fachPos = -21; break;
    default:     System.out.println("REGAL
    SCHON VOLL!!! REGAL SCHON VOLL!!!
    BEFEHLSVERWEIGERUNG!!!    WARTE    AUF
    AUSLIEFERUNGSBEFEHL!!!");
      }
        nullpunkt();
        fahrezu(0, 0, 0, -1);
        fahrezu(0, 0,-130, -1);
        fahrezu(0,0,-130,1);
        fahrezu(0,-3,0,1);
        fahrezu(-15,-8,0,1);
        fahrezu(fachPos,-13, 0,1);
        fahrezu(fachPos,-13,-60,1);
        fahrezu(fachPos,-13,-60,-1);
        fahrezu(fachPos, -13, 0, -1);
        if(pos < 4) {
        fachBelegt[pos] = true;
        istgelb[pos] = true;
        System.out.println("    Pos:    "
        +pos);
        System.out.println(" FachPos: "
        +fachPos);
         System.out.println(" belegt?: "
        +fachBelegt[0]+fachBelegt[1]+fach
        Belegt[2]+fachBelegt[3]);
        System.out.println(" gelb?: "
        +istgelb[0]+istgelb[1]+istgelb[2]
        +istgelb[3]);
            }
        }
```

Weitere Möglichkeiten von Java

Java und Bilder und Sounds

Innerhalb eines Javaprogramms können auch Bilder und Sounds importiert werden, die dann bei bestimmten Ereignissen geöffnet und abgespielt werden. Sounds können beispielsweise abgespielt werden, wenn der Legoroboter eine Tonne mit einer bestimmten Farbe findet. Aber mit Java können

auch kleine Melodien und Bilder selbst programmiert werden. Bilder kann man mit Hilfe einer Turtle (Schildkröte) erstellen. Diese Turtle befindet sich in einem schon fertiggestelltem Programm. Wenn man dieses Standardpaket importiert hat, kann man angeben zu welcher Position des Koordinatensystems sich die Turtle von einer bestimmten Grundposition aus bewegen soll. Sie hinterlässt dann eine Linie in einer bestimmten Farbe, die einstellbar ist. Man kann hier auch Bögen und Kreise zeichnen, die Turtle an eine andere Stelle „springen“ lassen und sie dort weiterzeichnen lassen. Natürlich kann man auch programmieren, dass z. B. ein gezeichnetes Dreieck um einen bestimmten Winkel gedreht und verschoben wird. Das kann man beliebig oft wiederholen lassen. So können am Schluss sehr komplizierte Muster und Grafiken entstehen. Diese könnte man dann wieder in einem anderen Programm verwenden.
Die Melodien kann man mit Hilfe eines Synthesizers (Programm oder Gerät, das elektrische Klänge und Tonfarben erzeugt), der in das Programm importiert werden kann, Töne mit bestimmter Höhe programmieren. Man kann auch aus vorgegebenen Klangfarben auswählen. Beispielsweise ob es der Klang eines Klaviers oder der Klang einer Flöte sein soll.
So kann man mit Hilfe von Java ganz schnell zum Komponisten werden!

Java und das Netzwerk

Natürlich kann man mit Javaprogrammen auch ein ganzes Netzwerk von Computern miteinander korrespondieren lassen. So kann z. B. ein Programm, dass sehr viel Daten verarbeiten muss, um einiges schneller laufen, wenn es die Prozessoren anderer Computer mitbenutzen kann. Dies erhöht natürlich die Möglichkeiten der Computerindustrie enorm. Denn es ermöglicht beispielsweise die nächste unbekannte Primzahl zu ermiteln.
Außerdem können mit solchen Netzwerk-programmen Menschen auf der ganzen Welt viel schneller miteinander korrespondieren, Daten übertragen, Videokonferenzen führen, usw. Auch hier zeigt sich wieder der Vorteil, dass Java eine Plattform unabhängige Sprache ist.

Die Verwendung von Java

Java ist besonders im World Wide Web gefragt, denn hier braucht man solch eine Plattform unabhängiger Sprache, denn nicht jeder Computer, der auf das World Wide Web zugreift, hat die gleiche Plattform. Aber Java ist nicht nur für die Computertechnologie interessant. Denn mit Java können ebenso gut Handys oder andere Microdevices programmiert werden. Auf diese kann man dann Programme, beispielsweise kleine Spiele, übertragen. Aber man kann Java ebenso gut dafür verwenden um Roboter zu programmieren, wie wir es getan haben oder auch andere elektronische Geräte programmieren, wie zum Beispiel neuere Spülmaschinen oder Backöfen.
Aus diesem Grund wird sich Java als universelle Programmiersprache in der Zukunft durchsetzen.

Die Lego-Roboter

Aus unserem Kurs haben sich Jessica, Rebecca, Andreas Ma. und Pascal dazu entschlossen, Lego-Roboter zu bauen. Lego-Roboter sind aus den Teilen des beliebten Kinderspielzeugs gebaut und sehen deshalb wie Spielzeug aus. Lego-Roboter sind nicht stationär und können sich daher frei bewegen. Da sie kabellos sind, haben sie eine nur durch die Batterieleistung begrenzte Reichweite. Das Bauen der Roboter ist relativ einfach, jedoch sind sie nicht sehr stabil. Der zentrale Baustein eines jeden Lego-Roboters ist der sogenannte RCX (Robotic Command Explorer), welcher drei Eingänge für Sensoren und drei Ausgänge für Motoren oder Lampen besitzt. Zwei der Ausgänge werden oft schon nur zum Fahren und Lenken benötigt. Als Energiequelle benutzt der RCX Batterien, bei denen allerdings nach einiger Zeit die Leistung abnimmt und die Sensoren darauf andere Werte anzeigen, wodurch weitere Schwierigkeiten entstehen.

In den folgenden Abschnitten werden die Funktionen der beiden Roboter Snoop und SEKUS vorgestellt.

Erste Konstruktionsversuche

Die SEKUS-Gruppe bei der Entwicklung

Bau des Roboters Snoop

Snoop (R1)

Der Snoop von Rebecca und Jessica, ist ein auf Lego basierender Suchroboter.

Aufgabe:

Snoop sucht gelbe und schwarze Tonnen, die zufällig auf einer Platte verteilt wurden. Beim Suchen fährt er immer geradeaus, bis er an einer Wand anstößt und dann um einen Zufallswinkel dreht. Sobald er eine Tonne gefunden hat, schließt er den Greifer, ermittelt welche Farbe die Tonne hat und sucht dann eine schwarze Linie auf dem Boden. Wenn er diese gefunden hat, fährt er je nach Farbe der Tonne auf der linken oder an der rechten Kante der Linie entlang, bis er am Abladepunkt ankommt. Dann fährt er ein Stück zurück und lässt die Tonne liegen. Jetzt hat der Fischertechnikroboter R2 Gustav eine Tonne geliefert bekommen. Snoop beginnt nun wieder Tonnen zu suchen.

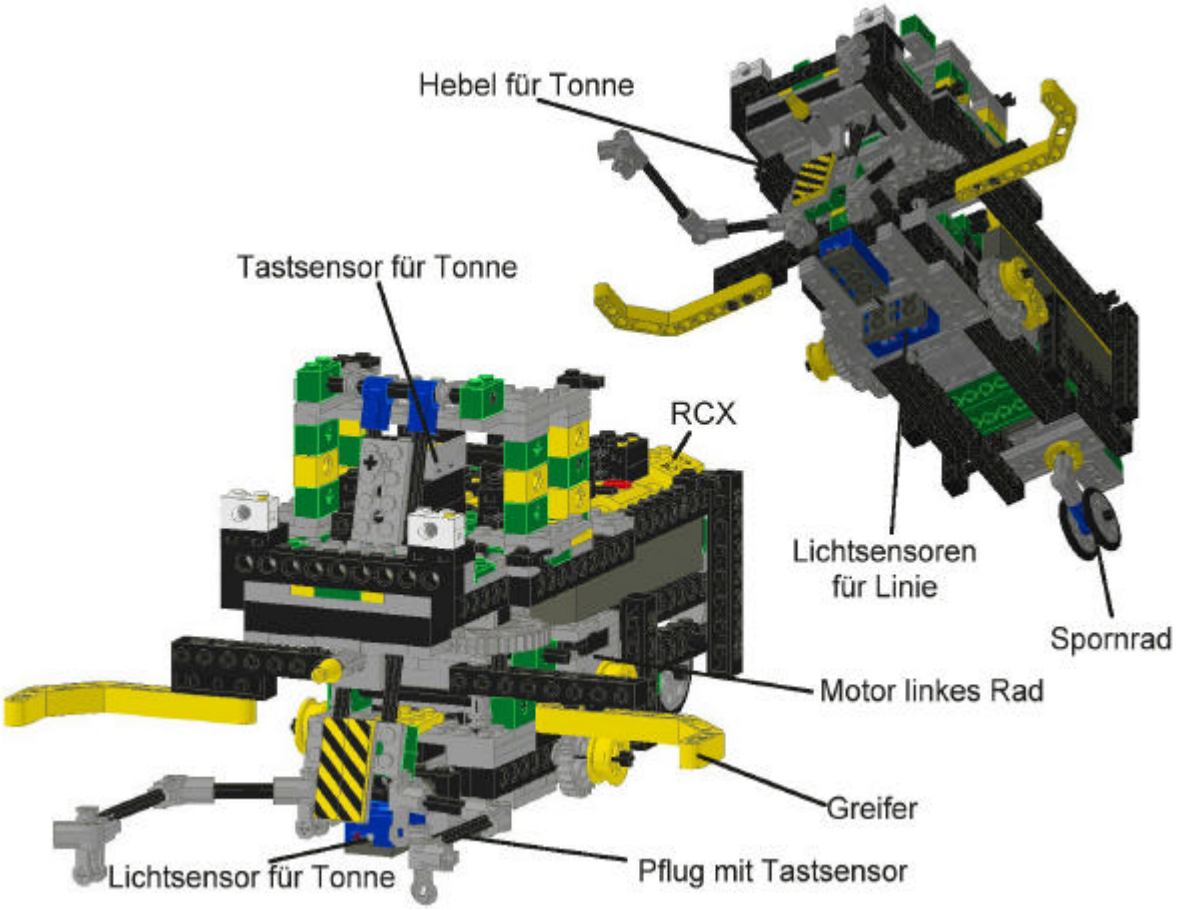
Bautechnische Umsetzung:

Das Wichtigste bei Snoop ist die Frontpartie. Das Zusammenspiel zwischen dem Greifer, dem Pflug, den Berührungs- und Lichtsensoren zu gewährleisten ist eine nicht einfache Aufgabe. Der Pflug zentriert die Tonnen und löst bei Berührung der Bande den „Wendemodus“ aus. Der Greifer schließt nach Auslösung eines zweiten Berührungssensors. Die Farbe der Tonne wird durch einen Lichtsensor registriert. Bei der Konstruktion gibt es folgende wichtigen Besonderheiten:

- Snoop besitzt nur drei Räder: je ein seitliches Antriebsrad und ein Spornrad, das sich frei drehen lässt. Jedes Antriebsrad hat einen eigenen Motor, was eine hohe Wendigkeit bewirkt.
- Da bei den Versuchen die Tonne oft im Greifer war, aber Snoop es nicht registrierte, musste ein Pflug zum Zentrieren der Tonne eingebaut werden, damit die Tonne den Berührungssensor auslöst.
- Im Pflug integriert ist ein Tastsensor, der dann gedrückt wird, wenn der Roboter an der Bande anstößt.
- In der Mitte der Frontpartie ist ein Tastsensor, der die Tonne registrieren soll.
- Da die Tonnen allerdings sehr leicht sind, musste man einen Hebel einbauen. Das Ende des Hebels ist schwarz-gelb gestreift.

- Direkt unter diesem Hebel ist ein Lichtsensor, der die Farbe der Tonne wahrnimmt. Da Gelb viel heller ist als Schwarz, erkennt der Lichtsensor, dass die Reflektion höher ist. Es gab auch zuvor Versuche, nur allein mit diesem Sensor wahrzunehmen, ob eine Tonne da ist, was aber scheiterte, da kein Unterschied zwischen gelber und keiner Tonne ausgemacht werden konnte.
- Zwischen den Vorderrädern sind zwei auf den Boden gerichtete Lichtsensoren angebracht für die Aufgabe, an der Kante der Linie zu fahren.

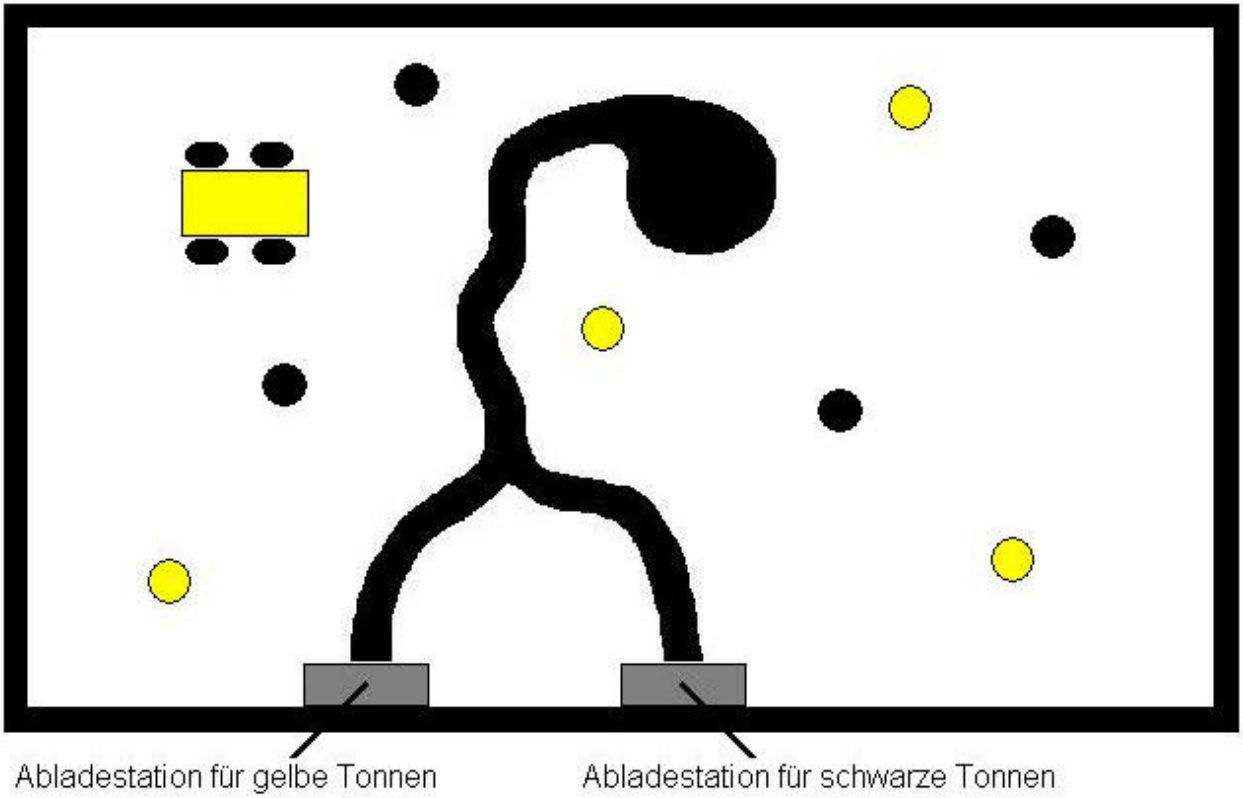
- Der Greifer ist das markanteste Merkmal. Er verhindert, dass die Tonne auf der Fahrt verloren geht. Er wird durch einen dritten Motor angetrieben. Es war knifflig, die Drehbewegung des Motors in eine Auf/Zu-Bewegung des Greifers umzuwandeln.



Programmiertechnische Umsetzung:

Das Programmieren von Snoop ist vom Grundgerüst das Gleiche wie das von Sekus.
Das Einzige was die Programmierung der beiden unterscheidet, sind die verschiedenen Verhalten auf Grund der verschiedenen Aufgaben, die im Folgenden näher erläutert werden:

1. **Stoppen**: Wenn der RUN-Button gedrückt ist, wird das Programm beendet
2. **Suche**: Wenn Snoop keine Tonne hat, fährt er die Platte im ziellosen Kurs ab.
3. **Wende**: Wenn der mit dem Pflug verbundene Berührungssensor gedrückt ist und keine Tonne im Greifer, fährt er ein kleines Stück rückwärts und dreht sich um seine eigene Achse.
4. **GreifeZu**: Wenn eine Tonne durch den Hebel den Berührungssensor auslöst, schließt sich der Greifer.
5. **SucheRechteKante:** Wenn eine Tonne vom Greifer gegriffen und als gelb erkannt



wird, muss Snoop die rechte Kante der schwarzen Linie suchen. Das heißt, der rechte Lichtsensor zwischen den Rädern muss weiß „sehen“ und der linke schwarz.

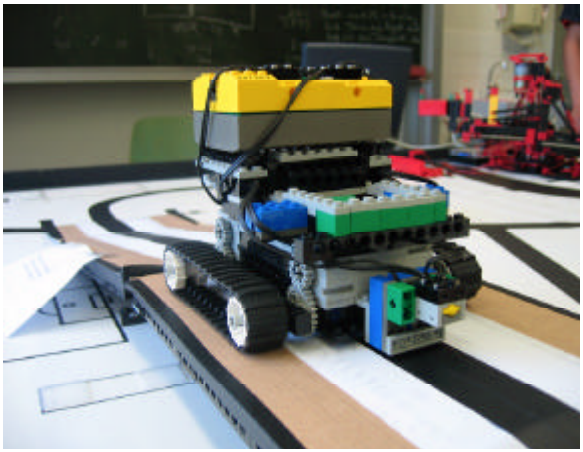
6. **SucheLinkeKante**: Wenn eine Tonne vom Greifer gegriffen und als schwarz erkannt wird, muss Snoop die linke Kante der schwarzen Linie suchen. Das heißt, der rechte Lichtsensor zwischen den Rädern muss schwarz „sehen“ und der linke weiß.
7. **Fahre**: Wenn Snoop durch seine beiden Lichtsenoren erkennt, dass er auf der Kante der Linie ist, auf der er auf Grund der Tonnenfarbe sein muss, fährt er geradeaus.
8. **DreheRechts**: Wenn der rechte der beiden unteren Lichtsensoren weiß anstatt schwarz oder der linke schwarz anstatt weiß „sieht“, muss Snoop nach rechts drehen.
9. **DreheLinks**: Wenn der linke der beiden unteren Lichtsensoren weiß anstatt schwarz oder der rechte schwarz anstatt weiß „sieht“, muss Snoop nach links drehen.
10. **LadeAb:** Stößt Snoop mit einer Tonne im Greifer gegen die Bande, setzt er zurück, öffnet seinen Greifer, fährt nocheinmal rückwärts, dreht um seine eigene Achse und beginnt von neuem mit dem Verhalten „Suchen“.

SEKUS (R3)

Der SEKUS von Pascal und Andreas Ma., benannt nach „Spedition S. Kracht & Scheppert“, ist ein auf Lego basierender Transportroboter.

Aufgabe:

SEKUS bekommt Tonnen vom Fischertechnikroboter R2 (Gustav mit der Hupe) aufgeladen. Daraufhin dreht er um und fährt auf einer schwarzen Linie über eine Brücke zum Fischertechnikroboter R4 (Frank). Dort fährt er gegen einen Tastsensor von Frank und fährt ein kleines Stück zurück, damit er besser entladen werden kann. Er hält an und wartet bis die Tonne entladen ist. Nun dreht er um, fährt auf der schwarzen Linie wieder zurück, stößt gegen eine Wand, fährt wieder ein kleines Stück zurück und wartet dann bis er beladen ist. Hierauf beginnt der ganze Vorgang wieder von vorne .

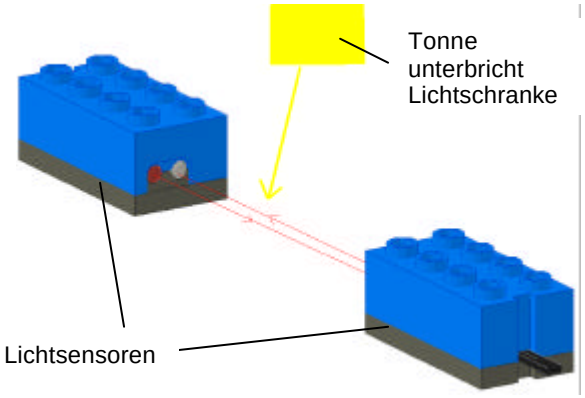


Bautechnische Umsetzung:

SEKUS muss so gebaut werden, dass er die Brücke überwinden kann. Außerdem muss er auf einer schwarzen Linie entlang fahren und erkennen, ob er eine Tonne aufgeladen hat oder nicht. Er sollt möglichst kompakt sein, möglichst vibrationsarm fahren und eine Ladefläche haben, die mit einem Greifarm gut zu erreichen ist und gleichzeitig aber auch den nötigen Halt für die Tonne bietet. Wichtig ist auch, dass er recht präzise fahren und auf der Stelle drehen kann. Dies alles ist natürlich nur mit einem Kompromiss möglich, der wie folgt aussieht:

- 4 Motoren in Verbindung mit zwei Raupen und einer Übersetzung von 2 : 1 vom Antrieb zur Raupe sorgen für die nötige Kraft und das Auf-der-Stelle-Drehen
- 2 Lichtsensoren ermöglichen das Fahren auf einer schwarzen Linie.
- Der Einbau des RCX erfolgte quer, was Platz spart und auch dafür sorgt, dass das Entladen erleichtert wird
- eine Art Führung, verwirklicht durch eine einen Art Trichter zur Ladefläche, sorgt für die gute Positionierung der Tonne
- ein Berührungssensor, der vorne vor den Lichtsensoren knapp über dem Boden angebracht ist für die Erkennung eines Zusammenstoßes mit der Wand
- Er besitzt eine Art Lichtschranke aus zwei gegenüberliegenden Lichtsensoren zum Erkennen ob eine Tonne da ist oder nicht. Da die zwei Lichtsensoren sich gegenseitig anstrahlen und beide an einem Eingang des RCX befestigt sind, d.h. ihre Werte addieren sich, ist der gemeldete Wert im Normalfall 100%. Wenn nun eine Tonne die Lichtschranke unterbricht, liegt der Wert nur noch bei ca. 80%, wodurch sich sehr gut entscheiden lässt, wann ein Roboter da ist und wann nicht.

Profil des Arbeitsbereichs von Sekus



Programmiertechnische Umsetzung:

SEKUS arbeitet mit dem Package namens Behavior („Verhalten“). In diesem Package ist ein Arbitrator („Schiedsrichter“) der entscheidet, welches „Verhalten“ aktiv wird. Unser Programm hat 8 Behaviors in aufsteigender Priorität: „fahre-Vorwaerts“, „dreheLinks“, „dreheRechts“, „drehe-Um“, „tonneAbgeladen“, „tonneAufgeladen“, „angestoßen“, „Stopp“, zu denen jeweils eine Datei gehört, in der festgelegt wird, wann dieses Verhalten aktiv wird (Methode „takeControl“). Außerdem sagt sie dem Roboter, was er machen soll (Methode „action“) und wie er sich zu verhalten

```
1  package behaviorctrl;
2
3  import josx.platform.rcx.*;
4  import josx.robotics.*;
5
6  public class DreheLinks implements Behavior, BehaviorConstants {
7
8      public boolean takeControl() {
9          return (!Status.istAngehalten &&
10                 sLinks.readValue() < S_SCHWARZ_WEISS &&
11                 sRechts.readValue() >= S_SCHWARZ_WEISS);
12     }
13
14     public void action() {
15         LCD.clear();
16         LCD.showNumber(5);
17         mLinks.backward();
18         mRechts.forward();
19     }
20
21     public void suppress() {
22         mLinks.forward();
23         mRechts.forward();
24     }
25 }
```

hat, bevor ein anderes Behavior aktiv wird (Methode „suppress“). Der Arbitrator „fragt“ die einzelnen Verhalten, ob die Bedingungen erfüllt sind, was dazu führt, dass das Verhalten aktiv wird. Nach absteigender Priorität, wenn die Bedingungen für das Verhalten „Stopp“ wahr sind und gleichzeitig für das Verhalten „fahreVorwaerts“ wird das Verhalten „Stopp“ aktiv. Wenn ein Verhalten aktiv ist, führt es die Bedingungen in Action so lange aus, bis die Bedingungen für dieses Verhalten nicht mehr wahr sind oder ein anderes Verhalten mit höherer Priorität die Kontrolle übernimmt. Darauf werden die Anweisungen in „Suppress“ ausgeführt. Danach wird dann erst das andere Verhalten aktiv. Der SEKUS hat 8 Verhalten die im folgenden näher erklärt werden:

1. **Stopp**: Wenn der RUN-Button gedrückt ist, wird das Programm beendet
2. **Angestossen**: Wenn der Berührungssensor gedrückt ist, fährt er ein kleines Stück rückwärts und speichert das Anstoßen in der Variable „istAngehalten“.
3. **TonneAufgeladen**: Wenn keine Tonne aufgeladen war und die Lichtschranke meldet, dass eine Tonne aufgeladen ist, wartet er noch fünf Sekunden, um ein Wegschwenken des leeren Greifarms des Fischertechnikroboters R2 zu ermöglichen. Er dreht danach um 190 ° nach links und setzt den Wert der Variable „istAngehalten’’ auf falsch.
4. **TonneAbgeladen**: Wenn eine Tonne aufgeladen war und die Lichtschranke meldet, dass keine Tonne mehr aufgeladen ist, wartet er weitere fünf Sekunden, um ein Wegschwenken des Greifarms mit der Tonne des Fischertechnikroboters R4 zu ermöglichen. Hierauf dreht dieser um ca. 190 ° nach links, setzt dabei den Wert der Variable „istAngehalten“ auf falsch.
5. **dreheUm**: Wenn der Wert der Variable „istAngehalten“ falsch ist und beide Lichtsensoren weiß sehen, dreht er rechts herum.
6. **dreheRechts**: Wenn der Wert der Variable „istAngehalten“ falsch ist, der linke Lichtsensor weiß sieht und der rechte schwarz, dreht er nach Rechts.
7. **dreheLinks**: Wenn der Wert der Variable „istAngehalten“ falsch ist, der linke Lichtsensor schwarz und der rechte weiß sieht, dreht er nach Links.
8. **fahreVorwaerts**: Wenn der Wert der Variable „istAngehalten“ falsch ist, fährt er vorwärts.

Was passiert demnach? Wenn man ihn auf eine schwarze Linie setzt, fährt er los, da „fahreVorwaerts“ aktiv ist. Nun weicht er rechts von der schwarzen Linie ab, da der linke Lichtsensor weiß sieht. Auf Grund dessen wird „dreheRechts“ aktiv. So findet SEKUS die Spur wieder. Dadurch wird „fahreVorwaerts“ wieder aktiv. Er korrigiert sich ständig, und verlässt auch in Kurven die Spur nicht.

Diagramm zur Veranschaulichung des Auf-der-Linie-Fahrens.

Wenn er gegen die aufgestellte Wand stößt, welches den Berührungssensor auslöst, wird das Behavior „Angestoßen“ aktiv. Dies geschieht, weil es eine höhere Priorität hat als z.B. „fahreVorwaerts“, auch wenn sich SEKUS gleichzeitig noch auf der schwarzen Linie befindet. Nun fährt er ein kleines Stück zurück und setzt den Wert der Variable „istAngehalten“ auf falsch. Danach sind die Bedingungen für das „Angestoßen“ nicht mehr erfüllt, da der Berührungssensor nicht mehr gedrückt ist. Nun wird kein Behavior aktiv. SEKUS wartet.
Wenn er eine Tonne erhält bzw. abgeladen bekommt, wird „TonneAbgeladen“ bzw. „Tonne-Aufgeladen“ aktiv. Er dreht um. Am Ende dieser Behaviors wird der Wert der Variablen „istAngehalten“ wieder auf „false“ gesetzt, so dass wieder die Behaviors „dreheUm“, „dreheRechts“, „dreheLinks“, und „fahreVorwaerts“ aktiv werden können. Der SEKUS fährt daraufhin zum anderen Fischertechnikroboter, um sich be- bzw. entladen zu lassen, wobei sich die Verfahrensweise wiederholt.

Der RCX

Wie schon erklärt, ist der RCX der zentrale Baustein des Roboters.

Er besitzt 50kB Speicher, der zum Speichern der Programme und der Firmware (des „Betriebssystems“) nötig ist. Da die Firmware allein schon 17kB des Speicherplatzes beansprucht, sind die Programme in ihrer Größe etwas eingeschränkt, im Allgemeinen reicht der zur Verfügung stehende Platz aber aus, da die Programme nur etwa 10-15 kB groß sind.

Die Infrarotschnittstelle, die sich an der Vorderseite befindet wird beim Übertragen der Programme bzw. der Firmware benötigt. Außerdem kann der Roboter so mit anderen Robotern oder dem PC kommunizieren.
Auf der LCD-Anzeige bekommt man die Information, ob das Programm gerade läuft. Außerdem kann man im Programm festgelegte Texte einblenden lassen. Sie hat allerdings nur 5 Stellen, also benutzten wir nur Abkürzungen oder zugewiesene Zahlen.

Die Sensoren

Hier werden einige Sensoren vorgestellt, jedoch nur die, die bei uns jemals zum Einsatz kamen:

Der Lichtsensor sendet Licht aus und empfängt das reflektierte Licht wieder. Die Helligkeit des empfangenen Lichts gibt er in Prozent an den RCX weiter. Damit kann er dann Farben erkennen, die vorher im Programm durch bestimmte Helligkeitswerte definiert wurden.

Der Berührungssensor reagiert auf Druck und meldet dann „false“ oder „true“ an den RCX (also gedrückt bzw. nicht gedrückt).
Der Rotationssensor misst die Umdrehungen pro Achse und das Programm errechnet so die gefahrene Strecke. Wenn man an das linke und das rechte Rad einen Sensor baut, so kann man relativ genau z.B. ein Quadrat abfahren lassen.

Des Weiteren gibt es noch viele andere Sensoren wie Temperatursensoren und Neigungssensoren. Man kann sich auch mit genügend Fachkenntnissen eigene Sensoren bauen.

Fischer-Technik Roboter

Die Fischer-Technik Roboter ähneln in ihrer Bauweise und Aufgabenbewältigung schon sehr dem Vorbild der modernen Industrieroboter. Sie sind meist stationär gebaut und besitzen einen typischen Greifarm, der durch mehrere separat steuerbare Achsen flexibel und präzise bewegt werden kann.
Der zentrale Tower unserer Roboter (vgl. Abb.) ermöglicht die Dreh- und Höhenbewegung, der daran befestigte Seitenarm steuert die Bewegung nach vorne und hinten. An ihnen ist schließlich eine starke Greifzange angebracht, deren Reichweite teilweise durch eine weitere Höhenachse vergrößert wird.

Um eine genaue Position anfahren zu können, arbeitet der Roboter mit sogenannten Zählern, von denen an jeder Achse einer vorhanden ist. Vergleichbar mit dem Verteilerfinger eines Automotors, ist an allen Achsen eine 4-blättrige Schraube befestigt, die bei jeder vollen Umdrehung einen elektrischen Kontakt viermal schließt und wieder öffnet. Gibt man nun im Programm für die Achsen eine bestimmte Anzahl von Umdrehungen

an, so kann der Greifarm eine festgelegte Position wiederholt relativ präzise anfahren.

Die größtmögliche Reichweite wird durch an den beiden Enden jeder Achse angebrachte Taster begrenzt. Sie reagieren wie die Zähler auf das Öffnen bzw. Schließen eines Stromkreises, wenn sie gedrückt bzw. losgelassen werden. Auch in der Bauweise unterscheiden sie sich nicht von den Zählern, es wird ihnen lediglich im Programm eine andere Aufgabe zugeschrieben.

Der Lichtsensor ermöglicht dem Programm, Lichtwerte der Umgebung zu erfassen und dadurch eine durch vorher definierte Werte festgelegte Farbe zu erkennen.

Ein weiteres Hauptmerkmal der Fischer-Technik Robotik ist das Interface, welches die Anweisungen des Computers sozusagen in echte Aktionen des Roboters umwandelt. Es besitzt acht Eingänge für die verschiedenen Sensoren, über die dem Computer gemessene Werte übermittelt werden, sowie vier Ausgänge für Motoren oder Lichter. Das Interface ist mit einem USB-Kabel an den USB-Ports des Computers angeschlossen, wodurch im Gegensatz zum Lego-RCX keine Speicherplatz-probleme auftreten können, dagegen aber die Mobilität etwas eingeschränkt wird.
Auch die Bauweise unterscheidet die Fischer-Technik- von den mobilen Lego-Robotern. Durch eine Steck-/Schieb-Verbindung gewinnt der Kunststoff an Verwindungssteifigkeit, bewegliche Teile wie Achsen, Zahnräder, Getriebe und Motoren sind sogar teilweise aus Metall gefertigt. Dennoch hatten wir lange mit dem Problem der Stabilität zu kämpfen, weil dadurch auch die Genauigkeit beim Anfahren eines Punktes erheblich eingeschränkt wurde.

Programmierung

Der grobe Aufbau eines Java-Programmes für Fischer-Technik Roboter ist von der Grundidee für die verschiedenen Roboter sehr ähnlich.
Exakt gleich ist das Package (Paket) “fischer-technik“, in dem in zahlreichen kleineren Klassen die Eigenschaften der verschiedenen Bauteile wie Motor, Lichter, Sensoren und Interface sowie von weiteren grundlegenden Programmfunktionen programmiert werden.
In den Oberklassen wird nun die endgültige Funktionsweise und das Verhalten des Roboters festgelegt. Diese Aufgabe wird auf Haupt- und Unterklassen verteilt, wobei mit dem Prinzip der Vererbung gearbeitet wird. Durch Vererbung kann eine Klasse von ihrer festgelegten Unterklasse alle Eigenschaften übernehmen, was zu einem kürzeren und wesentlich übersichtlicherem Quellcode beiträgt.
In der Klasse “Programm.java“ bzw. “Controller.java“ werden alle zuvor in den Unterklassen programmierte Funktionen aufgerufen. Sie sind von den Interfaces “FtTasterListener“ und

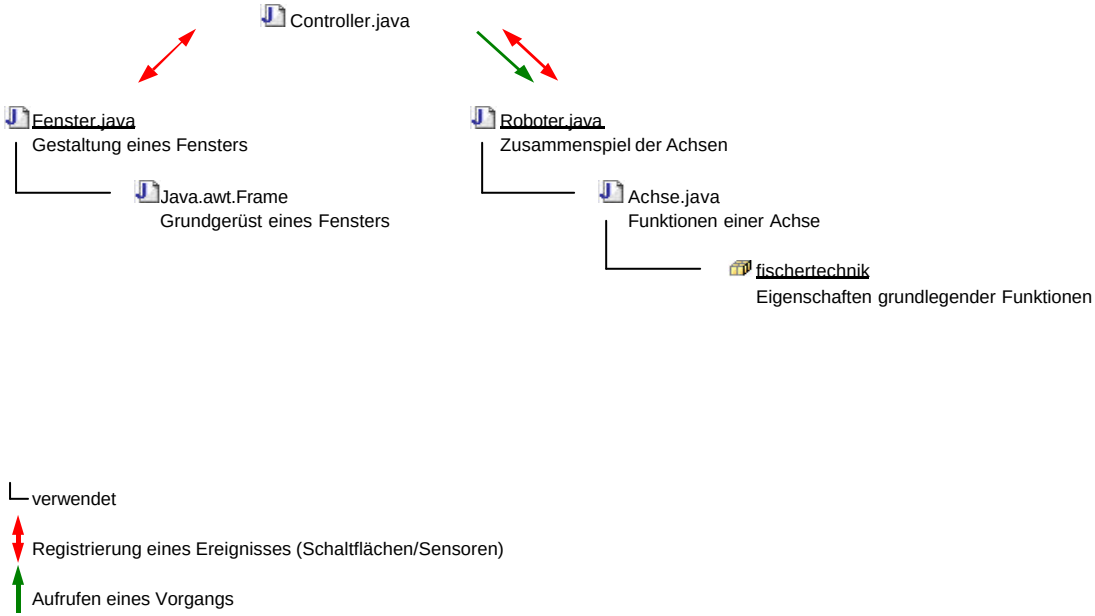
“ActionListener“ abgeleitet, was dazu führt, dass “Programm.java“ bzw. “Controller.java“ die von diesen Interfaces abgeleiteten Eigenschaften besitzten, bei Registrierung eines Ereignisses (“FtTasterListener“ für Fischer-Technik Sensoren bzw. “ActionListener“ für Schaltflächen oder Textfelder eines Fensters) einen Vorgang aufzurufen. Diese Aktion, in unserem Fall häufig eine Achsenbewegung, ist wiederum in einer Unterklasse definiert. In “Roboter.java“ wird das Zusammenspiel der Achsen festgelegt, beispielsweise in welcher Reihenfolge sie arbeiten; die genauen Koordinaten, d.h. die von den Zählern registrierten Umdrehungen, werden aber als Variablen noch offen gelassen und erst in der Klasse “Programm.java“ mit Zahlen belegt. “Achse.java“ enthält den Quellcode für die einzelnen Funktionen der Achsen, wie die Vorwärts- und Rückwärts-bewegung, das Stoppen bei Berührung eines Tasters und das Setzen eines Nullpunktes, was bewirkt, dass die Achsen an ihren Ausgangsort fahren und so die Zähler einen Punkt haben, von dem aus sie die Umdrehungen zählen können.

Anfangs wurden unsere Fischer-Technik Roboter von vielen verschiedenen Schaltflächen aus gesteuert, weshalb man nun auch noch eine weitere Klasse benötigte: in “Fenster.java“, abgeleitet von “java.awt.Frame“, einem Art Grundgerüst für ein Fenster, werden Fensterelemente wie Schaltflächen oder Textfelder programmiert.

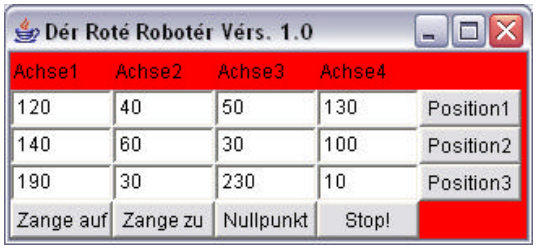
Was auf einen Mausklick hin passiert, wird, wie oben beschrieben, in “Programm.java“ bzw. “Controller.java“ festgelegt.

Nun wollten wir den Programmverlauf mehr und mehr automatisieren, sprich die Schaltflächen reduzieren, wobei unsere Endversionen immer noch von zwei Schaltflächen aus gesteuert werden mussten, die aber trotzdem eine größere Kette an Ereignissen beinhalteten.

MVC-Schema (Model View Control) eines FT-Programmes



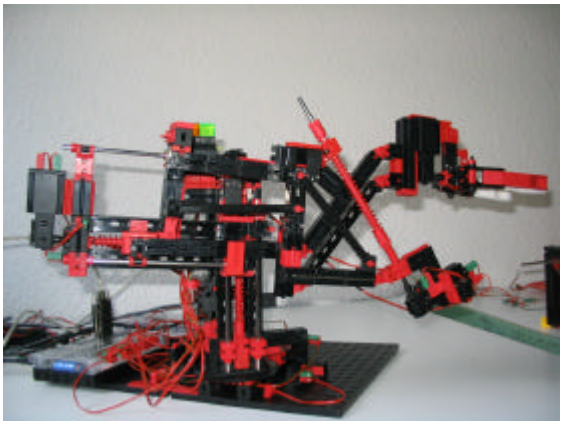
Entwicklung unseres Steuerungs-Fensters



Der Fischertecknik-Roboter „Frank“

Das Team

„Frank“ wurde von Andreas Mü., Christoph, Daniel und Marius gebaut bzw. programmiert.
Andreas, Christoph und Daniel übernahmen dabei zu einem größeren Teil die Programmierung des Roboters, während sich Marius eher auf den Bau von „Frank“ konzentrierte.

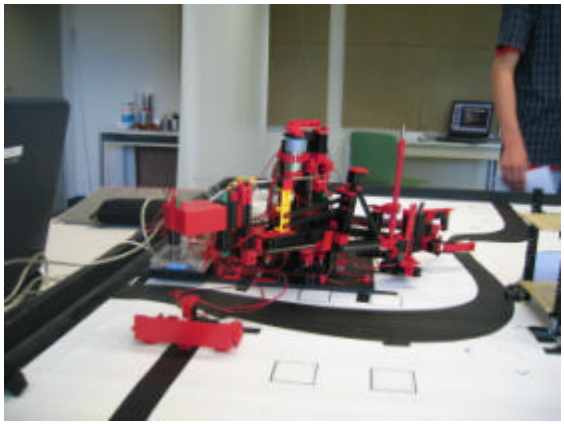


Der Fischertechnik-Roboter „Frank“ von Andreas, Christoph, Daniel und Marius.

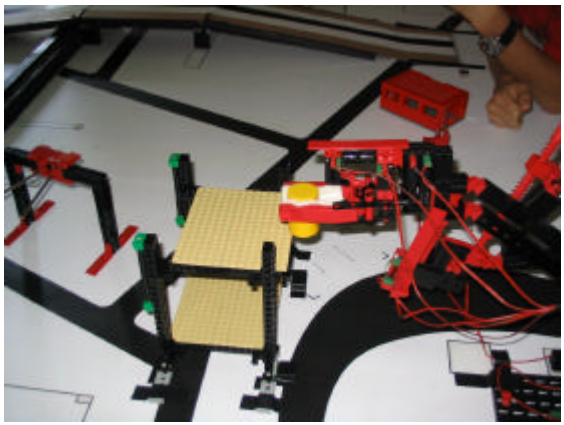
Die Aufgabe

Frank wurde die Aufgabe zugeteilt, gelbe bzw. schwarze Tonnen, die ihm der Lego-Transportroboter „Sekus“ liefert, in ein Regal einzuordnen, wobei die gelben Tonnen auf die untere Fläche des Regals und die schwarzen auf die obere gestellt werden sollte.

„Frank“ registriert, dass eine Tonne bereitsteht, wenn der Tastsensor (Abb. ganz vorne) von „Sekus“ gedrückt wird.



„Frank“ soll Tonnen, die hier im Vordergrund angeliefert werden, in das Regal rechts einordnen.



„Frank“ beim Versuch, die Tonne oben zu platzieren.

Der Aufbau

„Frank“ hat die Aufgabe, Tonnen in ein verhältnismäßig hohes Regal mit zwei Flächen einzuordnen. Dafür muss er also fähig sein, die Tonnen auf eine große Höhe zu bringen, was schon ein Problem mit der Konstruktion darstellt. Des Weiteren muss Frank in der Lage sein, die Tonnen auf der unteren Fläche des Regals zu platzieren und dann wieder mit der Greifzange aus dem Regal herauszukommen, ohne hängen zu bleiben. Dazu sind mehrere Achsen erforderlich.

„Frank“ besitzt deshalb vier Achsen, zusätzlich hat er eine Greifzange, die, rein technisch gesehen, nach dem Achsenprinzip funktioniert, also einen Motor und Sensoren benötigt.

Durch die hohe Zahl an Sensoren und Motoren – zu jedem Motor kommen zwei Tastsensoren und ein Zähler – sind für „Frank“ gleich zwei Interface-Module erforderlich. Eine weitere Konsequenz der großen Anzahl an Motoren und Sensoren sind die vielen Kabel, die mit den Interface-Modulen verbunden sind und den Roboter so in seiner Bewegungsfreiheit einschränken.

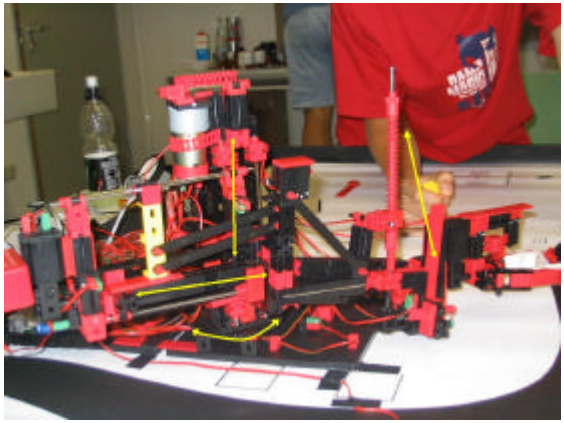
Die erste Achse (Abb. Pfeil ganz unten) sorgt dafür, dass sich „Frank“ seitlich drehen kann.

Die zweite Achse (Abb. linker senkrechter Pfeil) ist zur Regulierung des Armes. Mit ihrer Hilfe bewegt sich der gesamte Arm nach oben bzw. nach unten.

Die dritte Achse (Abb. waagerechter Pfeil) bewegt den Arm nach vorne bzw. nach hinten.

Die vierte Achse (Abb. rechter senkrechter Pfeil) bewegt die Greifzange nach oben bzw. nach unten.

Die Achsen können sich immer so lange bewegen,



Die gelben Pfeile zeigen die Bewegungen, die die Achsen „Frank“ ermöglichen.

bis sie einen Taster berühren, der, wenn er gedrückt wird, die Bewegung unterbricht.

Es gibt allerdings noch die Möglichkeit, die Dauer der Bewegung mittels eines Zählers zu kontrollieren. Die Greifzange entspricht der fünften Achse, allerdings benötigt sie keinen Zähler, sondern lediglich zwei Sensoren, die regulieren, wie weit sich die Zange öffnen oder schließen kann.

Die Programmierung

„Frank“ wurde wie alle drei anderen Roboter mit der Sprache Java (siehe zugehöriges Kapitel) programmiert.

Für Frank wurde, da er ein stationärer Fischertechnik-Roboter ist, mit Java ein Fenster mit Buttons programmiert, um ihn direkt vom PC aus steuern zu können. Dieses Fenster besitzt die Buttons „Motoren auf Nullpunkt“, „Stop“,

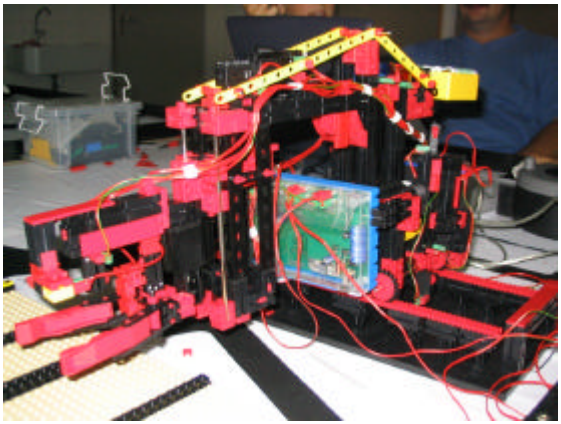
„Hole gelbe Tonne“, „Hole schwarze Tonne“.
Beim Druck auf den Button „Motoren auf Nullpunkt“ werden alle Achsen so lange in eine Richtung bewegt, bis sie den Tastsensor betätigen, der diese Bewegung stoppt. Mit dem Druck auf den Sensor wird für die Zähler an dieser Stelle der Nullpunkt gesetzt.
Allerdings dürfen nicht alle Achsen gleichzeitig den Nullpunkt setzen, da es sonst Berührungen mit dem Regal kommen würde, also wurden an dieser Stelle sog. Schleifen eingeführt, die verhindern, dass sich andere Achsen bewegen, während die eine noch aktiv ist.
Wenn der Button „Stop“ gedrückt wird, werden alle Achsen in der Position, in der sie sich gerade befinden, angehalten. Allerdings hat dies keine Wirkung, wenn gerade eine Schleife aktiv ist.
Der Button „Hole gelbe Tonne“ bewirkt im Prinzip das gleiche, wie der Button „Hole schwarze Tonne“, es werden nämlich die Achsen so bewegt, dass die Greifzange letztendlich an der Stelle angelangt, wo „Sekus“ die Tonne anliefert. Danach wird die Greifzange geschlossen, die Achsen bewegen sich zum Regal und stellen sie, je nach dem, ob gelb oder schwarz, unten bzw. oben ab.
Dabei ist die Reihenfolge der Achsen wieder sehr wichtig, denn alles muss so funktionieren, dass die Greifzange an die gewünschte Position kommt, dabei aber nirgendwo anstößt und aus der richtigen Richtung an die Position gelangt, wo die Tonne steht, um diese nicht zu verschieben.
Allerdings entstand dieses Programm mit all diesen Funktionen nicht auf einen Schlag, es wurden nach und nach immer mehr Methoden zu einer neuen zusammengefasst und diese dann einem bestimmten Button übergeben.
So wurden zum Beispiel die Positionen der Achsen zunächst so ermittelt, dass man die Zählerstände der einzelnen Achsen manuell eingab und so lange probierte, bis die gewünschte Position erreicht war. Dazu waren neben den Buttons im Fenster noch Textfelder nötig. Nachdem die Positionen ermittelt waren, wurden die Daten in den Quelltext direkt eingetragen und die Textfelder wieder aus dem Fenster gelöscht. Es wurden aber alle Achsen noch immer einzeln bewegt, mit separaten Buttons. Das sollte sich ändern und so wurden viele Buttons zu einem einzigen zusammengefasst.
So wurden letztendlich die Dateien „Programm9.java“, „MeinFenster.java“, „Roboter.java“ und „Achse.java“ von den Leuten des „Frank“-Teams erstellt. „Programm9“ ist das Hauptprogramm, das die Klassen „MeinFenster“, „Roboter“ und „Achse“ einsetzt.
Neben diesen Dateien waren aber noch viele andere nötig, um z.B. das Interface-Modul oder die Sensoren überhaupt benutzen zu können.
Diese Dateien wurden aber von den Kursleitern zur Verfügung gestellt.

Fischertechnik-Roboter „Gustav“

Das Team des Fischertechnik-Roboters „Gustav“ bestand aus Martin, Matthias, Peter und Heidi.

Die vier konstruierten und programmierten die zwei Wochen über einen Roboter, welcher, auf einer Schiene fahrend, über 4 Achsen verfügt.

Obwohl Fischertechnik Roboter hauptsächlich stationär gebraucht sind, ist er durch die besagte Schiene trotzdem recht beweglich. Der Hin- und Rückweg macht den Arbeitsraum deutlich größer, wodurch der Roboter viel mehr Arbeit verrichten kann. Auf den Schienen fährt ein Art Kran herum, welcher selber über 3 der insgesamt 4 Achsen verfügt.



Der Fischertechnik-Roboter „Gustav“ von Heidi Bersi, Martin Wurditsch, Matthias Zimmermann und Peter Holz

Mit der Zange kann der Roboter Tonnen und ähnliche greifbare Gegenstände aufheben. Durch die am Kran vorne angebrachte Achse, die auf und abfahren kann, wird der Gegenstand aus der Transportkiste heraus gehoben. Anschließend dreht sich der Kran und kann auf der anderen Seite der Schiene den Gegenstand ablegen. Falls notwenig bewegt er sich zuvor noch auf der Schiene zu einem anderen Punkt.

Die Tonnen werden zunächst nach Farbe erkannt und anschließend in ein Regal eingeräumt. Dort platziert merkt sich das Programm, wo schon Tonnen stehen und welche Farbe diese haben. Sollte ein Transportroboter kommen und eine gelbe Tonne verlangen, so weiß der Roboter auf die Weise, welche Tonne er abgeben muss.

Doch bevor man sich auf diese Probleme stürzen konnte, um sie zu realisieren, begann es zunächst mit einfacheren Sachen, wie zum Beispiel der grafischen Oberfläche, über die man zunächst die Motoren noch manuell steuern musste, was aber lange beibehalten wurde; die Automatik kam erst recht spät dazu. Diese Textfelder und Schaltflächen waren von Vorteil, da man ständig mit den Motoren Endpunkte der Drehungen herausfinden musste.

Die Sensoren, die die Umdrehungen der Achsen erfassen sollten taten dies jedoch manchmal nicht und so kam es öfters dazu, dass ein Motor einfach weiterdrehte und zum Teil dabei sogar den Roboter leicht beschädigte und man einige Teile wiederholt bauen musste. Auch die Zählung durch den Computer lief nicht immer erfolgreich ab, da die Sensoren anfangs oft noch die falschen Einstellungen hatten.

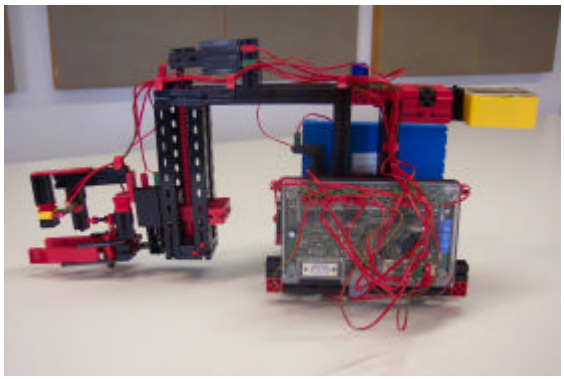
Bald waren auch diese Probleme beseitigt, sprich die Sensoren waren korrekt eingestellt und das Programm verzeichnete jede neue Meldung der

Zähler in der Konsole. Nun konnte man mit dem Programm zwar den Motor starten jedoch lief dieser dann in eine Richtung solange, bis man eine andere Richtung befahl. Da man so natürlich keinen Roboter präzise steuern kann, musste der Zähler überwacht werden, damit das Programm immer wusste, welcher Zählerstand gerade aktuell war.

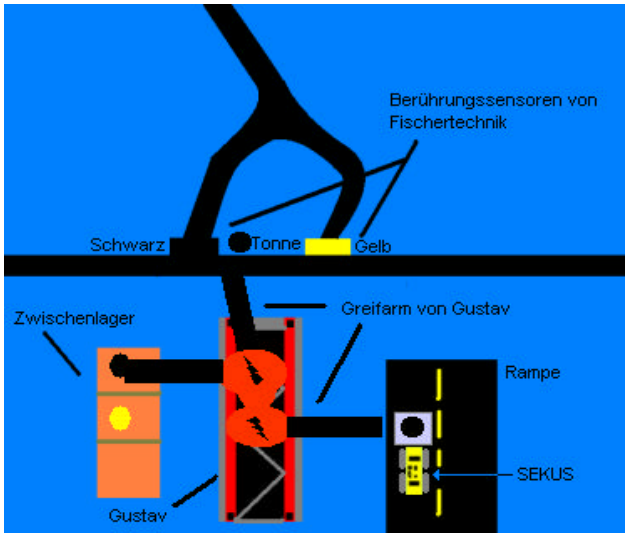
Hinzu kam dann noch eine Alarmfunktion die bei einem bestimmte Zählerstand den Motor stoppte. Nachdem der Alarm ausgelöst wurde, wurde der Wert gleich wieder gelöscht. Nun musste man im Programm einen neuen Wert eingeben und dem Motor den Befehl geben, zu diesem Wert zu fahren.

Im Moment war jedoch nur ein Motor dem Programm bekannt und so musste man weitere Motoren bzw. Achsen in das Programm einbauen. Um sich die Tipparbeit zu ersparen, für jede Achse jetzt erneut die Funktionen des Motors zu erklären arbeitete man mit Methoden. Dadurch musste man zwar die alten Funktionen etwas umschreiben, hatte danach aber einen schönen ordentlichen und vor allem übersichtlichen Quellcode.

Die Konstruktion des Roboters Gustav hatte anfangs noch leichte Probleme mit dem enormen Gewicht, das vor allem durch den Kran überwiegend in eine Richtung zerrte. Das Drehkreuz, auf welchem der Kran stand, wurde manchmal gar nicht erst gedreht, da das Zahnrad, welches das Drehkreuz drehen sollte, nicht in richtigem Kontakt mit diesem war und die Zähne des Drehkreuzes und des Zahnrads vom Motor nur aneinander knackten.



Roboter „Gustav“ aus der Seitenansicht



Aufgabe des FT-Roboters Gustav

Mit diesem Bild wird die Aufgabe von Gustav veranschaulicht. Man kann sehen, wie von oben der Lego-Roboter Snoop an eine Tonne heranfährt und diese vor der Mauer ablegt. Gustav nimmt sich nun diese Tonne und legt sie in sein Regal. Dabei weiß er, wie schon erwähnt, wo er die Tonne hinstellen darf, und an welcher Stelle im Regal noch Platz ist. Schließlich kommt der Lego-Roboter SEKUS an. Nachdem Gustav die Ankunft des Lego-Roboters mitgeteilt wurde, nimmt dieser die gewünschte Tonne aus dem Regal heraus und legt sie in die Transportkiste des Lego-Roboters.

Exkursion Heidelberg

MS-Technik

Nach den gemeinsamen Vorträgen im DKFZ gingen alle Kurse zu ihren eigenen Exkursionsplätzen und die Gruppe löste sich so langsam auf. Für unseren Robotikkurs wäre nun eigentlich eine Besichtigung des RobaCka - Roboters in der Kopfklinik geplant gewesen, doch leider war er gerade an diesem Tag zur Kontrolle völlig zerlegt worden.

So beschlossen wir spontan der MS-Technikausstellung, die als Bootmuseum auf dem Neckar ankerte, einen kurzen Besuch abzustatten.

„Unter Deck“

Nach kurzem Marsch durch die brütende Hitze erreichten wir das Boot. Unter Deck erwartete uns ein ungewohnter Anblick.

Der gesamte Laderaum des Schiffes war zu einem Museum umgebaut worden, in welches wir darauf sofort hineinspazierten.

Nun galt es zuerst ein Rätsel an verschiedenen Stationen zu lösen, was uns jedoch nicht sehr schwer fiel.

Die Ausstellung an sich war sehr interessant und einfallsreich aufgebaut. So gab es beispielsweise verschiedene Themenbereiche wie: Schall, Optik und Strom, die an Hand von Versuchen zum Ausprobieren logisch erklärt wurden.

Die Robotiker in der MS-Technik Ausstellung, beim Ausprobieren eines Kamera-Mikroskops.

Sicherlich eine der bekanntesten und interessantesten Methoden, Radioaktivität sichtbar zu machen: die Nebelkammer. Hier konnte man auf einem kleinen Podest stehend die trägen Alpha-Teilchen umherschwirren sehen. Zur Freude aller Robotikkursteilnehmer fanden wir auch einen kleinen, ferngesteuerten Roboter, den man von vier verschiedenen Stationen aus steuern konnte.

Für Räder und Schaufel gab es jeweils zwei Bedienungsstationen, dadurch wurde die Zusammenarbeit der einzelnen Funktionen des Roboters sehr gut hervorgehoben.

Mit Hilfe des Kameramikroskops (siehe Bild) untersuchten wir ganz verschiedene Materialien: Steine, Schwämme und auch unsere eigene Haut.
Auch ein Gerät, mit dem man Landschaften am PC untersuchen konnte, befand sich dort. Stufenloses Zoomen mit Hilfe einer Art Joystick machte das Betrachten von Bergen, Tälern und Flüssen erheblich einfacher.
Sogar unsere Lego-Roboter fanden wir in ähnlicher Form wieder.
Zum Abschluss des Rundgangs fand noch eine kleine Umfrage statt. Man konnte am PC Fragen über die Zukunft beantworten und die Ergebnisse der Abstimmung begutachten. Wer hätte gedacht, dass so viele Menschen schon in rund 25 Jahren mit einer bemannten Marslandung rechnen?
Schließlich traten wir hinaus in die pralle Sonne.

Nun stand jedoch unser Klinikbesuch auf dem Programm und mit ein klein wenig Eile traten wir den Rückweg am Neckarufer ins Neuenheimer Feld an.

Chirurgische Klinik Heidelberg

Unser nächstes Ziel nach der Besichtigung des Museumsschiffes war der Roboter
„da Vinci“ in der chirurgischen Klinik der Universität Heidelberg. Dort angekommen wurden wir von einer Krankenschwester in Empfang genommen und erst einmal in die Umkleiden geführt, da man den sterilen OP-Bereich nicht mit Straßenkleidung betreten darf. Dort entledigten wir uns unserer normalen Kleider und erhielten neue, sterile Kleider, Haube und Mundschutz. Frisch eingekleidet und ganz in grün wurden wir daraufhin in die Urologie zum Roboter „da Vinci“ geführt. Dort empfing uns auch schon ein Mitarbeiter von Intuitive Surgical (Herstellungsfirma von „daVinci“), welcher uns über die Möglichkeiten und Aufgaben des Roboters und dessen Geschichte informierte.

„da Vinci“ – ein Tele-Manipulator

Frontalansicht des „Roboters“.

„Da Vinci“ besteht aus zwei Teilen: An erster Stelle aus dem operierenden Teil und zweitens aus einer Bedienungseinheit, mit der er manuell vom Arzt gesteuert wird. Dadurch ist er aber streng genommen gar kein Roboter, da ein richtiger Roboter selbstständig agiert, unabhängig von irgendeiner manuellen Steuerung. „Da Vinci“ ist also kein Roboter, sondern ein Tele-Manipulator.
Der Manipulator besitzt drei Arme. An den beiden Äußeren ist an den Enden der Arme jeweils eine kleine Greifzange, und am Mittleren eine Kamera angebracht.
Über eine separate Bedienungseinheit wird der Manipulator vom operierenden Chirurg fern-

Wir bekommen den Ablauf einer OP erklärt.

gesteuert. Dieser übermittelt dann dem Roboter über ein Kabel tausende Male pro Sekunde, was er zu tun hat. Per Funk ist dies leider nicht möglich, da man von Gefahren ausgehen muss, sollte einmal der Funkkontakt unterbrochen werden.

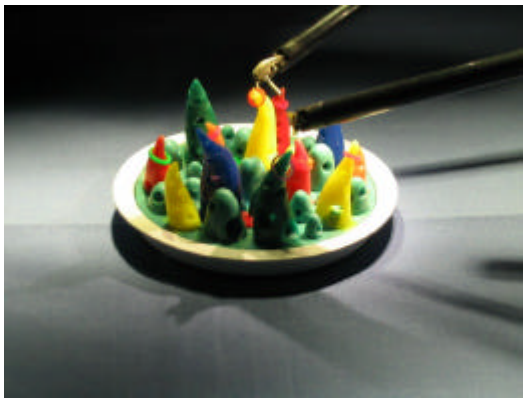
Um ihn zu bedienen, fährt der Chirurg mit den Fingern in kleine, an der Bedienungseinheit angebrachte Schlaufen, welche die Zangen des „da Vinci“ darstellen sollen. Die Bewegungen des Chirurgen werden dann auf den Manipulator übertragen. Über die Kamera am mittleren Arm sieht der Chirurg in 3D und stark vergrößert den Operationsbereich. Über Fußpedale kann er die Kameraeinstellungen ändern. Dies alles zusammen ermöglicht eine sehr präzise und genaue Arbeit.

Am meisten wird „da Vinci“ in der Chirurgie für Operationen an inneren Organen im Bauchbereich eingesetzt. Das besondere an „da Vinci“ ist, dass bei der Operation lediglich die drei kleinen Arme des Manipulators durch die Bauchdecke geführt werden müssen, sodass von der Operation lediglich drei kleine Wunden übrig bleiben. So werden größere Eingriffe vermieden und der Patient kann sich auf eine baldige Entlassung freuen.

Die separate Steuerung des „da Vinci“ ermöglicht sogar „Fernoperationen“ über große Entfernungen. Das größte Problem dabei ist noch die Übertragung. Ein kleiner Aussetzer des Funks, eine kleine Störung, keine Sekunde lang, könnten einen Patienten in eine lebensgefährliche Lage bringen. Diese Problem löst man mit Zeitverzögerung: Der Chirurg bewegt die Zangen auf die zu operierende Stelle zu, stoppt aber kurz davor und beginnt erst dann.

Arbeiten mit „da Vinci“

Am Ende durften wir uns dann auch noch an „da Vinci“ versuchen. Aufgabe war es, kleine Ringe über Hütchen zu stülpen.



Die Steuerung erforderte einiges an Geschick.

Es dauerte zwar einige Zeit, bis man sich an die Bewegungen, Stärke und Reaktion der kleinen Zangen gewöhnte, doch fanden es zumindest die meisten von uns erstaunlich einfach. Natürlich ist es etwas ganz anderes, einen Menschen zu operieren. Trotzdem war es eine tolle Erfahrung.
Wir erfuhren auch noch, dass es nur acht Exemplare dieses „Roboters“ in Deutschland gibt, was nur noch mehr bestätigte, dass unser Besuch in der Klinik, die Besichtigung des „da Vinci“ und auch die danach folgende Führung durch sämtliche OPs etwas ganz Besonderes war.

Das Team des Robotikkurses:

Die Kursleiter:

Name: **Helge Peters** (*1975)
Arbeitsstelle: Institut für Prozessrechentechnik, Automation und Robotik Universität Karlsruhe (TH)
Wir über ihn: hilft immer, total nett, grinst gern, unser Mädchen für alles

Name: **Matthias Taulien** (*1952)
Arbeitsstelle: Hector-Seminar Mannheim, Carl-Friedrich-Gauss-Gymnasium Hockenheim
Wir über ihn: erklärt gerne, zuverlässig, nett, immer voll konzentriert

Name: **Georg Wilke** (*1968)
Arbeitsstelle: Hector-Seminar Heidelberg, Bunsen-Gymnasium Heidelberg
Wir über ihn: total freundlich, hilft immer, man kann mit allen Fragen zu ihm kommen

Die Teilnehmer:

Name: **Christoph Amann** (*1988)
Schule: Schwarzwald- Gymnasium Triberg
Wir über ihn: immer hilfsbereit, super lustig, total nett, klasse Kumpel, cool, für gute Insiderwitze, da master, abgefahren
Spitzname: Chris, MasterBlaster

Name: **Heidi Bersi** (*1989)
Schule: Otto-Hahn-Gymnasium Böblingen
Wir über sie: lieb, zurückhaltend, einfach nur nett, hilfsbereit, hat alles unter Kontrolle, die Schreiberin, voll kooperativ

Name: **Marius Blaesing** (*1990)
Schule: Gymnasium am Hoptbühl Villingen
Wir über ihn: extrovertiert, Casanova, FT-Baumeister, macht jeden Spaß mit

Name: **Peter Holz** (*1989)
Schule: Dietrich-Bonnhoeffer-Gymnasium Eppelheim
Wir über ihn: super nett, hilft immer bei einem Problem, hört gut zu, Designer
Spitzname: Bender, Pit

Name:	**Daniel Lippert** (*1989)
Schule:	Matthias-Grünewald-Gymnasium Tauberbischofsheim
Wir über ihn:	schweigsam, Witzbold, die Spitze der Coolheit, Zahnpasta-werbungsgrinsen, der Sprüche-macher, toller Teamkamerad
Spitzname:	Sibat, Dan (the man)

Name:	**Pascal Weinert** (*1991)
Schule:	Herzog-Christoph-Gymnasium Beilstein
Wir über ihn:	super lieb, echt hilfsbereit und kooperativ, der Spammer, klasse Pianist

Name:	**Jessica Matthias** (*1989)
Schule:	Leibniz Gymnasium Östringen
Wir über sie:	Partymacherin, hört gern Witze, lacht gern und viel und laut (sehr laut), super Mitarbeit und einfach nur putzig!
Spitzname:	Jessy

Name:	**Martin Wurditsch** (*1990)
Schule:	Gymnasium Überlingen
Wir über ihn:	immer ganz korrekt, nie was falsch machen, immer einen netten Spruch auf Lager, diskussionsfreudig
Spitzname:	MVW

Name:	**Andreas Mayer** (*1989)
Schule:	Goethe-Gymnasium Ludwigsburg
Wir über ihn:	redet und postet gern und viel, total lieb, lustig
Spitzname:	Der Ma

Name:	**Rebecca Zelt** (*1990)
Schule:	Geschwister-Scholl-Gymnasium Mannheim
Wir über sie:	die Allerbeste, super lieb, immer für einen Scherz zu haben, hilft gerne, hört zu, nett und lustig, „Sündenbock“
Spitzname:	Becci, Eddie

Name:	**Andreas Müller** (*1990)
Schule:	Döchtbühl-Gymnasium Bad Waldsee
Wir über ihn:	ungewollt lustig, total nett, Checker, hat alles im Griff, überfreundlich, Kult, der Perfektionist
Spitzname:	Der µ (sprich: Der Mü)

Name:	**Matthias Zimmermann** (*1988)
Schule:	Martin- Gerbert- Gymnasium Horb
Wir über ihn:	sehr ruhig, nett, zurückhaltend, freundlich, der „Denker im Hintergrund“, auf ihn ist immer Verlass

„Robotics“ in der Uniklinik Heidelberg

„Robotics“ am Dokumentationswochenende in Donaueschingen: