

Dokumentation und Auswertung der Lego-Wesen

Verhalten kybernetischer Vehikel
Informatikprojekt I00-2

Inhaltsverzeichnis

Inhaltsverzeichnis.....	2
1 Aufgabenstellung	3
2 Anpassung der Wesen an die Realität.....	3
2.1 Sensorempfindlichkeit.....	3
2.2 Auswirkung auf das Verhalten der Wesen.....	3
2.3 Motorengeschwindigkeit	3
2.4 Sensorwinkel	3
3 Dokumentation der Wesen	3
3.1 Class control	3
3.1.1 initSensor()	3
3.1.2 setSpeed()	3
3.1.3 mSpeed()	3
3.1.4 LookAround()	3
3.1.5 goGo()	3
3.1.6 pause()	3
3.2 Wesen2.....	3
3.2.1 Wesen2a	3
3.2.2 Wesen2b	3
3.3 Wesen3.....	3
3.3.1 Wesen3a	3
3.3.2 Wesen3b	3
3.4 Wesen4.....	3
3.4.1 Wesen4a	3
3.4.2 Wesen4b	3
4 Fazit Theorie / Praxis	3
5 Anhang.....	3
5.1 Source-Code.....	3
5.1.1 Control	3
5.1.2 Wesen2a	3
5.1.3 Wesen2b	3
5.1.4 Wesen3a	3
5.1.5 Wesen3b	3
5.1.6 Wesen4a	3
5.1.7 Wesen4b	3

1 Aufgabenstellung

Konstruieren Sie mit Lego einen Roboter, der die im Braitenberg beschriebenen Verhalten ausführen kann.

2 Anpassung der Wesen an die Realität

Das Anpassen der Wesen an die Realität ist einfacher gesagt als getan. Die exakte, theoretische Implementation der nach Braitenberg beschriebenen Wesen erzielte nicht das gewünschte Resultat.

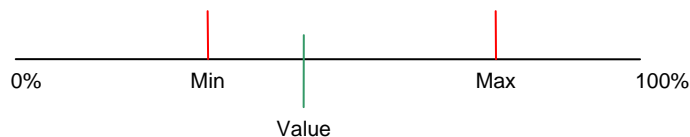
Wie bereits in der Analysephase erkannt, spielte die wenig geeignete Hardware eine hauptsächliche Rolle. Wir mussten die Wesen ein wenig anpassen. Zum Beispiel sind die Sensoren nicht empfindlich genug für grosse Einfallswinkel. Zudem ist der Verlauf der Empfindlichkeit zur Distanz nicht linear. Weiter können wir keinen Raum zur Verfügung stellen, welcher keine Fremdlichtquellen hat beziehungsweise die Lichtquelle nie nur direkt bestrahlt, sondern immer irgendwo gespiegelt wird (weisse Wand). Somit ist der Messwertebereich der Sensoren nicht zwischen 0 und 100%, was natürlich die lineare Kopplung mit den Motoren erschwert und dabei nicht das gewünschte Verhalten resultiert.

Nachstehend haben wir in einzelne Kapitel gegliedert, die Probleme der Hardware erläutert.

2.1 Sensorempfindlichkeit

Wir haben die Geschwindigkeit von 0 – 7 in einem Min/Max-Bereich der gemessenen Werte zugeordnet. Min und Max werden laufend aktualisiert, Value ist der gemessene Sensorwert pro Sensor. Die Min/Max – Werte sind global und gelten für beide Sensoren.

$$Speed = \text{round}\left(\frac{7 (Value - Min)}{Max - Min}\right)$$



Initialwerte der Min/Max-Werte werden zu Beginn bei der „Geburt“ eines Wesens gesetzt, indem der Roboter eine 360°-Drehung durchführt, welche Lichtquellen als Maximum und dunkle Ecken als Minimum setzt.

2.2 Auswirkung auf das Verhalten der Wesen

Dies verändert das Verhalten der Wesen geringfügig. Die Min/Max-Werte werden laufend angepasst, was natürlich zu einer leichten Verschiebung der errechneten Geschwindigkeiten führen kann und somit das Wesen zu Beginn seines Lebens anders auf die Lichtintensität reagiert, als nach einer gewissen Reife.

2.3 Motorengeschwindigkeit

Wie wir in der Analysephase unseres Projektes gesehen haben, ist die Geschwindigkeitsdifferenz zwischen den Werten 1-7 zu wenig unterscheidend. Die Wesen würden so unmerkliche Richtungsänderungen machen und an den Lichtquellen vorbeifahren obschon sie den Motoren korrekte Impulse liefern.

Um dies zu verhindern haben wir im Programm die Geschwindigkeit nachträglich angepasst, indem wir den Motor verfrüht stoppen. So wird zum Beispiel der Motor schon ab Geschwindigkeiten kleiner 4 gestoppt. Leider muss dies bei den verschiedenen Wesen zu einem anderen Zeitpunkt geschehen, damit das Verhalten einigermaßen stimmt.

Durch diese schlechten Voraussetzungen wirkt das Verhalten ein wenig digital. Die Wesen scheinen nun entweder zu stehen, oder Vollgas zu fahren.

Wie die einzelnen Wesen anzupassen sind ist zu Beginn des Robocodes des jeweiligen Wesens dokumentiert (Kapitel 3).

2.4 Sensorwinkel

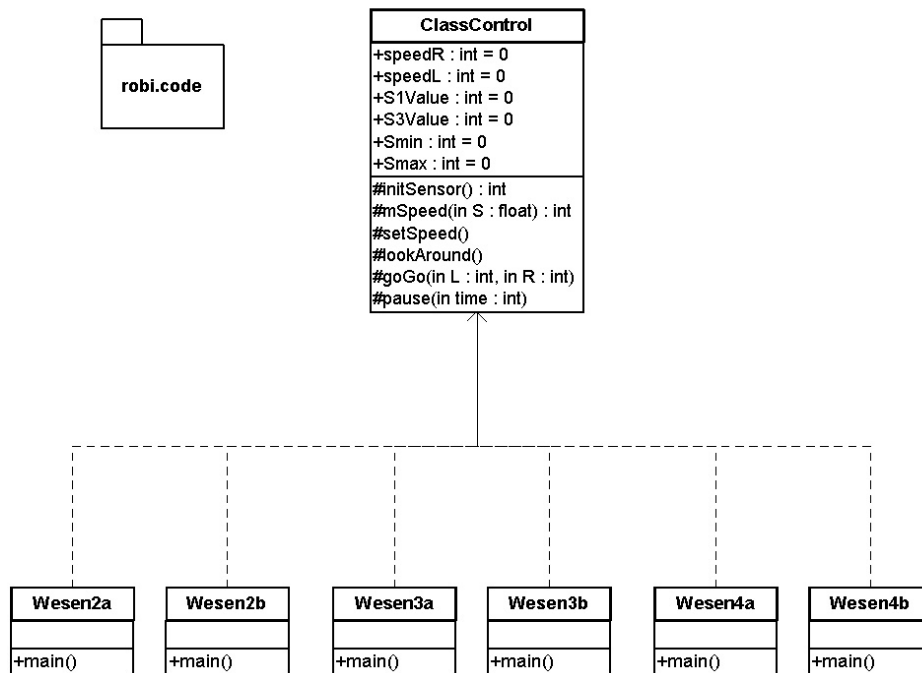
Durch die gedämpfte Arbeitsweise der Sensoren (siehe Dokumentation: Analyse und Design) waren wir gezwungen eine ähnliche Strategie wie bei der Anpassung der Motorengeschwindigkeit zu verfolgen.

Das Ändern des Winkels der beiden Sensoren vorne am Roboter erzielt ein unterschiedliches Verhalten gegenüber der Lichtquelle des jeweiligen Roboters. Wir versuchten auch hier im Kapitel 3 beim Beschrieb des Robocodes Informationen darüber zu liefern, wie die Winkeleinstellung sein muss. In der Realität sind diese Einstellungen leider nur sehr schwierig zu messen und sind von Faktoren wie; Startposition, Art der Lichtquelle, Abstrahlungen, etc. beeinflussbar.

3 Dokumentation der Wesen

In diesem Abschnitt wird der „Robocode“ dokumentiert.

3.1 Klassendiagramm



3.2 Class control

Die Klasse Control wird an alle Wesen vererbt. Sie enthält die lebenswichtigen Funktionen, welche die Bewegungen, die Berechnung und für die Sensoren zuständig sind.

3.2.1 initSensor()

InitSensor konfiguriert die Sensoren: Lichtsensoren mit prozentueller Angabe. Zudem werden die Min/Max-Werte gesetzt, damit bei der Berechnung keine Division durch 0 entstehen kann.

```

protected static void initSensor() {
    Sensor.S1.setTypeAndMode(SensorConstants.SENSOR_TYPE_LIGHT,
        SensorConstants.SENSOR_MODE_PCT);
    Sensor.S1.activate();
    Sensor.S3.setTypeAndMode(SensorConstants.SENSOR_TYPE_LIGHT,
        SensorConstants.SENSOR_MODE_PCT);
    Sensor.S3.activate();
    S1Value=Sensor.S1.readValue();
    S3Value=Sensor.S3.readValue();
    Smax=S1Value+3; Smin=S1Value-3;
}

```

3.2.2 setSpeed()

Diese Methode liest die Sensorwerte und berechnet daraus die Geschwindigkeit für die Motoren mit oben erklärter Formel. Zudem werden die Min/Max-Werte aktualisiert. Smin und Smax sind globale Variablen und gelten für beide Sensoren.

```
/* Reads sensor values and calculates forward speed */
protected static void setSpeed(){
    S1Value=Sensor.S1.readValue();
    S3Value=Sensor.S3.readValue();
    if (S1Value>Smax) Smax=S1Value;
    if (S3Value>Smax) Smax=S3Value;
    if (S1Value<Smin) Smin=S1Value;
    if (S3Value<Smin) Smin=S3Value;
    speedL=Math.round(7*((float)S1Value-(float)Smin)/((float)Smax-(float)Smin));
    speedR=Math.round(7*((float)S3Value-(float)Smin)/((float)Smax-(float)Smin));
}
```

3.2.3 mSpeed()

Die Methode mSpeed wird ausschliesslich von den Wesen 4a und 4b benutzt. Der Unterschied zu setSpeed ist, dass die Geschwindigkeiten nur einmal, nicht immerwährend berechnet und abgespeichert werden.

```
protected static int mSpeed(float S){
    if (S>90){ return 7;
    }else if (S>78){ return 6;
    }else if (S>65){ return 5;
    }else if (S>52){ return 4;
    }else if (S>39){ return 3;
    }else if (S>26){ return 2;
    }else if (S>13){ return 1;
    }else { return 0;
    }
}
```

3.2.4 LookAround()

Damit das Wesen an die Lichtverhältnisse in einem Raum angepasst werden kann, macht es in dieser Methode einen 360°-Kreis und merkt sich dabei die Minimum- und Maximum-Werte. Zusätzlich merkt sich das Wesen während des Drehens die ungefähre Position der stärksten, bei mehreren, der resultierenden Lichtquellen. Nach der Drehung kann sich das Wesen dann nach dieser ausrichten und wird so sicher auch Lichtquellen „im Rücken“ finden.

```
protected static void lookAround() throws InterruptedException{
    int tmpDir = 0;
    Motor.A.setPower(0);
    Motor.C.setPower(7);
    Motor.A.stop();
    Motor.C.forward();
    for(int i=0;i<27;i++){
        S1Value=Sensor.S1.readValue();
        S3Value=Sensor.S3.readValue();
        if (S1Value>Smax) { Smax=S1Value; tmpDir=i;}
        if (S3Value>Smax) { Smax=S3Value; tmpDir=i;}
        if (S1Value<Smin) Smin=S1Value;
        if (S3Value<Smin) Smin=S3Value;
        pause(200);
    }
    Motor.A.setPower(0);
    Motor.C.setPower(7);
    Motor.A.stop();
    Motor.C.forward();
    tmpDir=tmpDir-3;
    for(int i=0;i<tmpDir;i++){
        pause(200);
    }
    Motor.A.stop();
    Motor.C.stop();
    pause(2000);
}
```

3.2.5 goGo()

Die Methode goGo verwaltet die Motoren. Hier wird entsprechend den errechneten Werten die Geschwindigkeit für den Linken und Rechten Motor gesetzt oder der Motor wird gestoppt.

```
protected static void goGo(int L, int R){
    if (R==0 && L==0) {
        Motor.A.stop();
        Motor.C.stop();
    }
    else if (L==0){ Motor.A.stop();
        Motor.C.setPower((int)R);
        Motor.C.forward();
    }
    else if (R==0){ Motor.C.stop();
        Motor.A.setPower((int)L);
        Motor.A.forward();
    }
    else {
        Motor.A.setPower((int)L);
        Motor.C.setPower((int)R);
        Motor.A.forward();
        Motor.C.forward();
    }
}
```

3.2.6 pause()

Pause-Methode für den RunningThread.

```
protected static void pause(int time) throws InterruptedException{
    try { Thread.sleep(time); }
    catch (InterruptedException ie) { }
}
```

3.3 Wesen2

Verhalten der Wesen2a,b – Furcht und Aggression:

Das Wesen2a verhält sich „ängstlich“, das heisst, es fährt einer Lichtquelle davon.

Wesen2b ist der aggressive Typ. Er fährt auf die Lichtquelle zu.

3.3.1 Wesen2a

Die Berechnung der Sensorempfindlichkeit erfolgt wie in Punkt 2.1 beschrieben. Die Motorgeschwindigkeit ist grösser, je heller die Lichtquelle. Bevor die Methode goGo aufgerufen wird, wird nun eine Anpassung vorgenommen. Der Motor wird somit früher, bereits bei einem Wert kleiner 3 gestoppt.

3.3.1.1 Code

```
import josx.platform.rcx.*;
public class Wesen2a extends Control{
    public Wesen2a(){
        public static void main(String []args) throws InterruptedException{
            initSensor();
            lookAround();
            //Do while true ...
            while (true){
                setSpeed();
                // Anpassung an Lichtquelle
                if (speedL<3) speedL=0;
                if (speedR<3) speedR=0;
                goGo(speedL,speedR);
                pause(100);
            }
        }
    }
}
```

3.3.1.2 Auswertung

Das beste Verhalten erzielen wir mit einer Taschenlampe immer direkt in den Sensor zündend, die Sensoren ca. 10°, nur leicht geöffnet. Das bedeutet die Lichtquelle ist nicht rundum Abstrahlend, sondern auf den Roboter gerichtet und der Schwellenwert des Sensors entsprechend gross.

Der Roboter funktioniert auch bei einer rundum Abstrahlenden Lichtquelle, wobei hier dann auf die Startposition und Winkel zur Lichtquelle geachtet werden muss. Strahlt die Lichtquelle exakt vor dem Roboter, kollidiert dieser aufgrund der gleichen Werte beider Sensoren, mit der Lichtquelle.

3.3.2 Wesen2b

Die Berechnung erfolgt wie in Punkt 1.1 beschrieben. Als einziger Unterschied zu Wesen2a wurden die errechneten Geschwindigkeiten (speedL, speedR) der Motoren vertauscht. Damit erreicht man die Eigenschaften von Wesen2b, welche ja nur die Sensoren mit den Motoren kreuzt. Dafür muss vor dem Aufruf der Methode goGo der Schwellenwert angepasst werden (je nach Lichtquelle und Raum).

3.3.2.1 Code

```
import josx.platform.rcx.*;
public class Wesen2b extends Control{
    public Wesen2b(){
    }
    public static void main(String []args) throws InterruptedException{
        initSensor();
        lookAround();
    }
}
```



```

//Do while true ...
while (true){
    setSpeed();
    // Anpassung an Lichtquelle
    if (speedL<5) speedL=0;
    if (speedR<5) speedR=0;
    goGo(speedR, speedL);
    pause(100);
}
}
}

```

3.3.2.2 Auswertung

Da dieses Wesen eine Abwandlung des Wesens 2a ist, gelten hier exakt die gleichen Bedingungen für Sensorwinkel und Motorengeschwindigkeit.

3.4 Wesen3

Verhalten der Wesen3a,b – Liebe und Neugier:

Das Wesen3a verliebt sich in eine Lichtquelle, das heisst, es richtet sich nach einer Lichtquelle aus und bleibt stehen. Der Typ 3b fährt daran vorbei und hält nach neuen Quellen Ausschau.

3.4.1 Wesen3a

Der Code entspricht dem von Wesen2 mit dem Unterschied, dass die errechneten Geschwindigkeitswerte invertiert werden. Das heisst aus einer grossen Geschwindigkeit wird eine kleine. Auch hier können und müssen vor dem Aufruf der Methode goGo die Werte an die Umwelt angepasst werden.

3.4.1.1 Code

```

import josx.platform.rcx.*;
public class Wesen3a extends Control{
    public Wesen3a(){
        public static void main(String []args) throws InterruptedException{
            initSensor();
            lookAround();
            //Do while true ...
            while (true){
                setSpeed();
                // Anpassung an Lichtquelle
                if (speedL<5) speedL=0;
                if (speedR<5) speedR=0;
                speedL=7-speedL;
                speedR=7-speedR;
                goGo(speedL, speedR);
            }
        }
    }
}

```

3.4.1.2 Auswertung

Hier erzielten wir ein besseres Verhalten bei einer starken, rundum Abstrahlenden Lichtquelle, die Sensoren noch weniger, ca. 5-10° zur Fahrzeugrichtung auseinander. Leider knallt der Roboter auch hier in die Lichtquelle, wenn er die Sensorwerte nicht unterscheiden kann.

3.4.2 Wesen3b

Die Kreuzung der Sensoren/Motoren wird auch hier nur mit dem Vertauschen der Geschwindigkeitswerte bei der Übergabe an die Methode goGo realisiert. Die Werte zur Anpassung müssen ebenfalls individuell gesetzt werden. Alles andere ist identisch zum Wesen 3a.

3.4.2.1 Code

```
import josx.platform.rcx.*;
public class Wesen3b extends Control{
    public Wesen3b(){
    }
    public static void main(String []args) throws InterruptedException{
        initSensor();
        lookAround();
        //Do while true ...
        while (true){
            setSpeed();
            // Anpassung an Lichtquelle
            if (speedL<3) speedL=0;
            if (speedR<3) speedR=0;
            speedL=7-speedL;
            speedR=7-speedR;
            goGo(speedR, speedL);
            pause(100);
        }
    }
}
```

3.4.2.2 Auswertung

Da dieses Wesen eine Abwandlung des Wesens 3a ist, gelten hier exakt die gleichen Bedingungen für Sensorwinkel und Motorengeschwindigkeit.

3.5 Wesen4

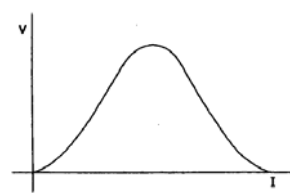
Wesen4a Wertung und Wesen4b Geschmack

Bei diesen Wesen verlassen wir die lineare Kopplung zwischen Sensor und Motor. Bei Wesen 4a wird eine nicht lineare Funktion zur Kopplung verwendet, bei Wesen 4b wird mit Schwellenwerten ein entsprechendes Verhalten generiert.

3.5.1 Wesen4a

Die nicht lineare Kopplung zwischen Sensor und Motor wird mit dem Array *param* realisiert.

Die Arraywerte entsprechen grob (nur 8 Werte) dem Verlauf dieser Funktion.



Die errechnete Geschwindigkeit wird mit dem Wert in *param* multipliziert. Das Wesen 4a bewegt sich nun nur bei gemässigten Lichtwerten. Ist es zu hell oder zu dunkel, bleibt es stehen.

Die Funktion `setSpeed()` wird bei den Wesen 4a und 4b nicht benutzt, die Geschwindigkeit wird linear verteilt und nicht fortlaufend berechnet.

3.5.1.1 Code

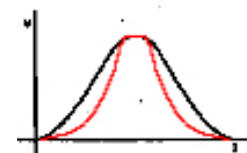
```
import josx.platform.rcx.*;
import java.lang.Math;

public class Wesen4a extends Control{

    public Wesen4a(){
    }
    static int[] param = { 0 , 0 , 5 , 7 , 7 , 5 , 0 , 0};
    public static void main(String []args) throws InterruptedException{
        initSensor();
        lookAround();
        //Do while true ...
        while (true){
            S1Value=Sensor.S1.readValue();
            S3Value=Sensor.S3.readValue();
            speedL=param[mSpeed(S1Value)];
            speedR=param[mSpeed(S3Value)];
            goGo(speedL, speedR);
            pause(100);
        }
    }
}
```

3.5.1.2 Auswertung

Das richtige Verhalten dieses Wesens konnten wir mit direktem blenden, oder löschen der Lichtquelle erkennen. Wählt man die Startposition und den Einfallswinkel der Lichtquelle zu Beginn ideal, bewegt es sich auch wie erwünscht. Wichtig ist, dass man den Schwellenwert der Sensoren im Code richtig einstellt (*param*). Je nach Raum verhält sich das Wesen „deutlicher“ wenn *param* diskreter gewählt wird (rote Linie).



Der Winkel der Sensoren sollte man so wie bei Wesen 2 (ca. 10°, nur leicht geöffnet) einstellen, sonst erreichen die Sensoren nie gleichzeitig 100% und das Wesen beginnt zu „schwänzeln“.



3.5.2 Wesen4b

Wesen 4b funktioniert genau gleich wie Kollege 4a, hat jedoch andere Werte in param, womit sein Verhalten auch anders ist. Es braucht einen bestimmten Schwellenwert, bis die Motoren reagieren.

3.5.2.1 Code

```
import josx.platform.rcx.*;
import java.lang.Math;
public class Wesen4b extends Control{
    public Wesen4b(){
    }
    static int [] param = { 0 , 0 , 0 , 0 , 7 , 7 , 7 , 7 };
    public static void main(String []args) throws InterruptedException{
        initSensor();
        lookAround();
        while (true){
            S1Value=Sensor.S1.readValue();
            S3Value=Sensor.S3.readValue();
            speedL=param[mSpeed(S1Value)];
            speedR=param[mSpeed(S3Value)];
            goGo(speedL,speedR);
            pause(100);
        }
    }
}
```

3.5.2.2 Auswertung

Da dieses Wesen eine Abwandlung des Wesens 4a ist, gelten hier exakt die gleichen Bedingungen für Sensorwinkel und Motorengeschwindigkeit.

4 Fazit Theorie / Praxis

Leider bewegt sich diese ganze Dokumentation grösstenteils um die Fehlerbehandlung der Sensoren und die Motorengeschwindigkeit. Wir haben zu Beginn diesen Faktoren deutlich zu wenig Wertung gewidmet. Die Wesen wie sie in Braitenberg beschrieben sind, sind sehr einfach und unkompliziert zu implementieren. Das Resultat war jedoch nie das Erwartete. Den Wesen musste Logik eingebaut werden von welcher Braitenberg nie sprach. Die Arbeit war teilweise auch demotivierend, weil durch die vielen Einflüsse und Fehlerquellen, trotz grösstmöglicher Phantasie und Anpassungsideen, kaum Fortschritte erzielt werden konnten. Während der Arbeit wünschten wir uns oft exaktere Laborbedingungen oder bessere Hardware zur Verfügung zu haben.

Das ganze Projekt ist in der Theorie interessant und hätte noch viel Entdeckungsraum.

5 Anhang

5.1 Source-Code

5.1.1 Control

```
/******
 *
 *      Project Röbi, Informnatikprojekt I00
 *      Control for Röbi
 *      Author: Gafner, Witschi, Reusser, Fleury
 *      Date: 19.05.2003
 *      Verison V1.0
 *
 *      Description of class Control:
 *      Steuermethoden für die Wesen
 *
 *
 *
 *****/

package robi.code;
import josx.platform.rcx.*;

public class Control{

    public static int speedR=0,
        speedL=0,
        S1Value=0,
        S3Value=0,
        Smin=0,
        Smax=0;

    public Control( ){

/* *****
Initialisierung der Sensoren
***** */

        protected static void initSensor(){
            Sensor.S1.setTypeAndMode(SensorConstants.SENSOR_TYPE_LIGHT,
SensorConstants.SENSOR_MODE_PCT);
            Sensor.S1.activate();
            Sensor.S3.setTypeAndMode(SensorConstants.SENSOR_TYPE_LIGHT,
SensorConstants.SENSOR_MODE_PCT);
            Sensor.S3.activate();
            S1Value=Sensor.S1.readValue();
            S3Value=Sensor.S3.readValue();
            Smax=S1Value+3; Smin=S1Value-3;
        }

/* *****
Vergeben der Geschwindigkeit abhängig von Lichtwert.
Fixe Werte für Wesen 4a und 4b
***** */

        protected static int mSpeed(float S){
            if (S>90){ return 7;
            }else if (S>78){ return 6;
            }else if (S>65){ return 5;
            }else if (S>52){ return 4;
            }else if (S>39){ return 3;
            }else if (S>26){ return 2;
            }else if (S>13){ return 1;
            }else { return 0;
            }

/* *****
Liest die Sensorwerte, setzt Minumum und Maximum-
Werte und berechnet die Geschwindigkeit für die
Motoren.
***** */

        protected static void setSpeed(){
            S1Value=Sensor.S1.readValue();
```

```

S3Value=Sensor.S3.readValue();
if (S1Value>Smax) Smax=S1Value;
if (S3Value>Smax) Smax=S3Value;
if (S1Value<Smin) Smin=S1Value;
if (S3Value<Smin) Smin=S3Value;
speedL=Math.round(7*((float)S1Value-(float)Smin)/((float)Smax-
(float)Smin));
speedR=Math.round(7*((float)S3Value-(float)Smin)/((float)Smax-
(float)Smin));
}

```

```

/* *****

```

Macht einen 360°-Kreis und setzt so Min und Max-Werte
Zudem wir das Wesen ungefähr auf die Quelle ausgerichtet

```

*****

```

```

protected static void lookAround() throws InterruptedException{
    int tmpDir = 0;
    Motor.A.setPower(0);
    Motor.C.setPower(7);
    Motor.A.stop();
    Motor.C.forward();
    for(int i=0;i<27;i++){
        S1Value=Sensor.S1.readValue();
        S3Value=Sensor.S3.readValue();
        if (S1Value>Smax) { Smax=S1Value; tmpDir=i;}
        if (S3Value>Smax) {Smax=S3Value; tmpDir=i;}
        if (S1Value<Smin) Smin=S1Value;
        if (S3Value<Smin) Smin=S3Value;
        pause(200);
    }
    Motor.A.setPower(0);
    Motor.C.setPower(7);
    Motor.A.stop();
    Motor.C.forward();
    tmpDir=tmpDir-3;
    for(int i=0;i<tmpDir;i++){
        pause(200);
    }
    Motor.A.stop();
    Motor.C.stop();
    pause(2000);
}

```

```

/* *****

```

Setzt Geschwindigkeit für die Motoren anhand
der Sensorwerte.

```

*****

```

```

protected static void goGo(int L, int R){
    if (R==0 && L==0) {
        Motor.A.stop();
        Motor.C.stop();
    }
    else if (L==0){ Motor.A.stop();
        Motor.C.setPower((int)R);
        Motor.C.forward();
    }
    else if (R==0){ Motor.C.stop();
        Motor.A.setPower((int)L);
        Motor.A.forward();
    }
    else {
        Motor.A.setPower((int)L);
        Motor.C.setPower((int)R);
        Motor.A.forward();
        Motor.C.forward();
    }
}

```

```

/* *****

```

Pause-Funktion. Während dieser Zeit bleibt der Zustand
des Wesens unverändert (fährt, steht, ...)

```

*****

```

```

protected static void pause(int time) throws InterruptedException{
    try { Thread.sleep(time); }
    catch (InterruptedException ie) { }
}

```



·robi

5.1.2 Wesen2a

Project Röbi, Informatikprojekt I00
Type of Character: Wesen2a
Author: Gafner, Witschi, Reusser, Fleury
Date: 19.05.2003
Version V1.0

Description of character:
Wesen2a nach Valentin Braitenberg verhält sich "ängstlich". Es flieht vor Lichtquellen.
Die Sensoren sind
proportional mit den Motoren gekoppelt. Je heller die Lichteinstrahlung, desto
schneller dreht der Motor. Somit
wendet sich das Wesen von Lichtquellen ab.

Anpassungen:
Vor dem Aufruf goGo werden Geschwindigkeiten an Umgebung angepasst

*****/

```
import josx.platform.rcx.*;

public class Wesen2a extends Control{

    public Wesen2a(){
    }

    public static void main(String []args) throws InterruptedException{
        initSensor();           //Sensoren initialisieren
        lookAround();           // Min/Max- Werte setzten
        //Do while true ...
        while (true){
            setSpeed();          // berechnet Geschwindigkeit
            if (speedL<3) speedL=0; // Anpassung an Umgebung
            if (speedR<3) speedR=0; // Anpassung an Umgebung
            //LCD.showNumber(S1Value);
            goGo(speedL,speedR); // Steuert Motoren
            pause(100);
        }
    }
}
```

5.1.3 Wesen2b

/******

Project Röbi, Informatikprojekt I00
Type of Character: Wesen2b
Author: Gafner, Witschi, Reusser, Fleury
Date: 19.05.2003
Version V1.0

Description of character:

Wesen2a nach Valentin Braitenberg verhält sich "aggressiv". Es greift Lichtquellen an. Die Sensoren sind proportional mit den Motoren gekoppelt. Je heller die Lichteinstrahlung, desto schneller dreht der Motor. Die Sensoren und die Motoren sind gekreuzt, das heisst der linke Sensor bedient den rechten Motor, der rechte Sensor den linken Motor. Somit fährt das Wesen auf Lichtquellen zu.

Anpassungen:

Vor dem Aufruf goGo werden Geschwindigkeiten an Umgebung angepasst

*****/

```
import josx.platform.rcx.*;
public class Wesen2b extends Control{
    public Wesen2b(){
    }
    public static void main(String []args) throws InterruptedException{
        initSensor();           //Sensoren initialisieren
        lookAround();           // Min/Max- Werte setzten
        //Do while true ...
        while (true){
            setSpeed();          // berechnet Geschwindigkeit
            if (speedL<5) speedL=0; // Anpassung an Umgebung
            if (speedR<5) speedR=0; // Anpassung an Umgebung
            //LCD.showNumber(S1Value);
            goGo(speedR,speedL); // Steuert Motoren
            pause(100);
        }
    }
}
```


5.1.4 Wesen3a

Project Röbi, Informatikprojekt I00
Type of Character: Wesen3a
Author: Gafner, Witschi, Reusser, Fleury
Date: 19.05.2003
Version V1.0

Description of character:
Wesen3a nach Valentin Braitenberg verhält sich "verliebt".
Es sucht die Lichtquellen und bleibt davor stehen. Die Sensoren sind
proportional mit den Motoren gekoppelt. Je heller die Lichteinstrahlung, desto
langsamer dreht der Motor. Somit
wendet sich das Wesen von Lichtquellen ab.

Anpassungen:
Pause eingebaut

*****/

```
import josx.platform.rcx.*;
import java.lang.Math;
public class Wesen3a extends Control{
    public Wesen3a(){
    }
    public static void main(String []args) throws InterruptedException{
        initSensor();
        //Sensoren initialisieren
        lookAround(); // Min/Max- Werte setzten
        //Do while true ...
        while (true){
            setSpeed(); // berechnet Geschwindigkeit
            if (speedL<3) speedL=0; // Anpassung an Umgebung
            if (speedR<3) speedR=0; // Anpassung an Umgebung
            speedL=7-speedL;
            speedR=7-speedR;
            LCD.showNumber(S1Value);
            goGo(speedL,speedR); // Steuert Motoren
            pause(100);
        }
    }
}
```

5.1.5 Wesen3b

```

/*****
Project Röbi, Informatikprojekt I00
Type of Character: Wesen3b
Author: Gafner, Witschi, Reusser, Fleury
Date: 19.05.2003
Version V1.0

Description of character:
Wesen3b nach Valentin Braitenberg verhält sich "neugierig". Es sucht die Lichtquellen und neigt
sich davon ab.
Es bleibt stehen und hält nach weiteren ausschau. Die Sensoren sind
proportional mit den Motoren gekoppelt. Je heller die Lichteinstrahlung, desto langsamer dreht
der Motor. Somit
wendet sich das Wesen von Lichtquellen ab.

Anpassungen:
Sensorwinkel gross
Pause eingebaut
*****/
import josx.platform.rcx.*;

public class Wesen3b extends Control{
    public Wesen3b(){
    }
    public static void main(String []args) throws InterruptedException{
        initSensor();
        //Sensoren initialisieren
        lookAround();
        // Min/Max- Werte setzten
        //Do while true ...
        while (true){
            setSpeed(); // berechnet Geschwindigkeit
            if (speedL<3) speedL=0; // Anpassung an Umgebung
            if (speedR<3) speedR=0; // Anpassung an Umgebung
            speedL=7-speedL; // Invertierung Mehr-Weniger
            speedR=7-speedR; // Invertierung Mehr-Weniger
            //LCD.showNumber(S1Value);
            goGo(speedR,speedL); // Steuert Motoren
            pause(100);
        }
    }
}

```

5.1.6 Wesen4a

Project Röbi, Informatikprojekt I00
Type of Character: Wesen4a
Author: Gafner, Witschi, Reusser, Fleury
Date: 19.05.2003
Version V1.0

Description of character:

Anpassungen:

Da die Legomotoren leider nur sehr minim auf eine Änderung der Geschwindigkeit reagieren, werden die Motoren bei kleinen Geschwindigkeiten auf "Floating" (param) gesetzt, um eine grössere Abstufung zu erreichen.

Zurück auf int, dafür eher diskret, länger 0% und früher 100%

*****/

```
import josx.platform.rcx.*;
import java.lang.Math;

public class Wesen4a extends Control{
    public Wesen4a(){
    }
    static int[] param = { 0 , 0 , 5 , 7 , 7 , 5 , 0 , 0 }; // Nicht lineare Funktion
    public static void main(String []args) throws InterruptedException{
        initSensor(); //Sensoren initialisieren
        lookAround(); // Min/Max- Werte setzten
        //Do while true ...
        while (true){
            S1Value=Sensor.S1.readValue();
            S3Value=Sensor.S3.readValue();
            speedL=param[mSpeed(S1Value)]; // Korrektur der Motorenwerte
            speedR=param[mSpeed(S3Value)]; // Korrektur der Motorenwerte
            goGo(speedL, speedR); // Steuert Motoren
            pause(100);
        }
    }
}
```

5.1.7 Wesen4b

Project Röbi, Informatikprojekt I00
Type of Character: Wesen4b
Author: Gafner, Witschi, Reusser, Fleury
Date: 19.05.2003
Version V1.0

Description of character:

Anpassungen:

Da die Legomotoren leider nur sehr minim auf eine Änderung der Geschwindigkeit reagieren, werden die Motoren bei kleinen Geschwindigkeiten auf "Floating" (param) gesetzt, um eine grössere Abstufung zu erreichen.
Zurück auf int, dafür eher diskret, länger 0% und früher 100%

*****/

```
import josx.platform.rcx.*;
import java.lang.Math;

public class Wesen4b extends Control{
    public Wesen4b(){
    }

    static int[] param = { 0 , 0 , 0 , 0 , 7 , 7 , 7 , 7 };
    public static void main(String []args) throws InterruptedException{
        initSensor();
        //Sensoren initialisieren
        lookAround();
        // Min/Max- Werte setzten
        //Do while true ...
        while (true){
            S1Value=Sensor.S1.readValue();
            S3Value=Sensor.S3.readValue();
            speedL=param[mSpeed(S1Value)];
            speedR=param[mSpeed(S3Value)];
            goGo(speedL, speedR);
            pause(100);
        }
    }
}
```

// Korrektur der Motorenwerte
// Korrektur der Motorenwerte
// Steuert Motoren