Dr. Jürgen Buchham

RoSi

Ein Framework
zur
Simulation
von
NXT-ähnlichen Robotern

Grundsätzliches

Rosi-Simulationen sind von der Klasse Rosi abgeleitete Java-Klassen.
Die Klasse Rosi ihrerseits ist abgeleitet von der Klasse JApplet, Somit ist jede Rosi-Simulation ein Applet und lässt sich über HTML-Dokumente publizieren.

Die Klasse Rosi verkapselt die gesamte komplizierte Geometrie und Darstellung des Roboters mit allen seinen Komponenten und dessen Bewegungsabläufe und Verhaltensweisen.
Sie stellt eine Reihe von Methoden und Klassen (mit deren Methoden) bereit, mit denen auch ein Programmier-Anfänger Steuerungsaufgaben für den Roboter bewältigen kann.

Man erkennt die nach außen tretenden Klassen und Methoden des Frameworks daran, dass sie allesamt mit Wörtern der deutschen Sprache bezeichnet werden.

```
public class RosiBeispiel extends Rosi
{
      ....
      void los()
      {
         ...
      }
}
```

Die Klasse RoSi

Hinweise:

Die Spielfläche (Rosi-Klasse *Parcours*) entspricht einem Koordinatensystem mit x von -400 bis +400 und y von 300 bis -300.
Die Positionsangaben beziehen sich auf dieses Koordinatensystem.

Winkelangaben verstehen sich im Gradmaßstab (modulo 360). Die X-Achse zeigt in Richtung Null Grad.
Die Geschwindigkeitsangaben entsprechen etwa Pixel pro Sekunde.

Das Display (Rosi-Klasse *Display*) ist wie beim echten NXT ein Punktraster mit den Ausmaßen 100x64 Pixel.
Es wird der besseren Lesbarkeit wegen in doppelter Vergrößerung neben dem Parcours dargestellt.

Methoden zur Fortbewegung

positionieren(int nx, int ny, int w)
Aufstellen der Fahrzeugs an der Stelle nx,ny mit dem Winkel w.

fahren(int vr, int vl)
Fahren mit Geschwindigkeit *vr* auf dem rechten Motor und Geschwindigkeit *vl* auf dem linken Motor.

fahren(int v)
Fahren mit Geschwindigkeit *v* auf beiden Motoren.

links(int nProzent)
Erhöhen der Geschwindigkeit des rechten Motors um *nProzent* Prozent und Vermindern der Geschwindigkeit des linken Motors um *nProzent* Prozent.

rechts(int nProzent)
Erhöhen der Geschwindigkeit des linken Motors um *nProzent* Prozent und Vermindern der Geschwindigkeit des rechten Motors um *nProzent* Prozent.

linksDrehen(int nLinks)
Verändern der Geschwindigkeit des linken Motors.
Werte zwischen 0 und 100.
0 bedeutet geradeaus (links gleich rechts),
50 bedeutet drehen um das linke Rad (links = 0)
100 bedeutet drehen um den Mittelpunkt (links = -rechts)

rechtsDrehen(int nRechts)
Verändern der Geschwindigkeit des rechten Motors.
Werte zwischen 0 und 100.
0 bedeutet geradeaus (rechts gleich links),
50 bedeutet drehen um das linke Rad (rechts = 0)
100 bedeutet drehen um den Mittelpunkt (rechts = -links)

gerade()
Rücksetzen auf die ursprünglich mit fahren() gesetzten Werte.

drehen(int gradRechts, gradLinks)
drehen des rechten Rades um *gradRechts* Grad und des linken Rades um *gradLinks* Grad

halt()
Anhalten.
Gleichbedeutend mit fahren(0,0);

Methoden zur Ablaufsteuerung

fertig()
Anzeige nach den Anfangseinstellungen.

starten()
Bewegungsablauf starten.

warten(int mSec)
Den momentanen Bewegungsablauf mSec Millisekunden 'laufen lassen'.

programmBeenden();
Das Rosi-Applet beenden.

Display-Methoden

löschen()
Das Display löschen.

punkt(int x, int y)
Zeichnen eines Punktes mit den Koordinaten *x* und *y*

linie(int x1, int y1, int x2, int y2)
Zeichnen einer Linie vom Punkt (*x1*,*y1*) zum Punkt (*x2*,*y2*)

rechteck(int x, int y, int b, int b)
Zeichnen eines Rechtecks mit der linken oberen Eckpunkt (*x*,*y*) , der Breite *b* und der Höhe *h*

kreis(int x, int y, int r)
Zeichnen eines Kreises mit dem Mittelpunkt (*x*,*y*) und dem Radius h

text(String s, int x, int y)
Schreiben des Textes *s* an die Stelle (*x*,*y*)

text(int n, int x, int y)
Schreiben der Zahl *n* an die Stelle (*x*,*y*)

text(double a, int x, int y)
Schreiben der Zahl a an die Stelle (*x*,*y*)

Sonstige Methoden

schreiben(String text)
schreiben(int n)
schreiben(double x)
Schreiben in den System-Ausgabebereich in Eclipse (Meldungsfenster am unteren Rand der Eclipse – IDE)

int **calcR(double vai)**
Ermittelt den Radius bezogen auf den Fahrzeugmittelpunkt bei vorgegebenem Verhältnis der äußeren zur inneren Geschwindigkeit *vai*.

double **calcVai(int r)**
Ermittelt das Verhältnis von äußerer zu innerer Geschwindigkeit bei vorgegebenem Radius bezogen auf den Fahrzeugmittelpunkt.

int **x()**
liefert die momentane x-Koordinate des Fahrzeugmittelpunkts.

int **y()**
liefert die momentane y-Koordinate des Fahrzeugmittelpunkts.

int **winkel()**
liefert den momentanen Richtungswinkel des Fahrzeugs in Grad.

long **laufzeit()**
liefert die Zeit in Millisekunden seit Programmstart.

Innere Klassen

Prozess

Der Konstruktor *prozess(int mSec)* erzeugt einen eigenständigen Thread, der alle *mSec* Millisekunden die Methode *Aktion* aufruft.
Diese ist in den abgeleiteten Klassen zu überschreiben, wie z.B.

```
Prozess pr = new Prozess(10)
{
      void Aktion()
      {
            ...
      }
};
```

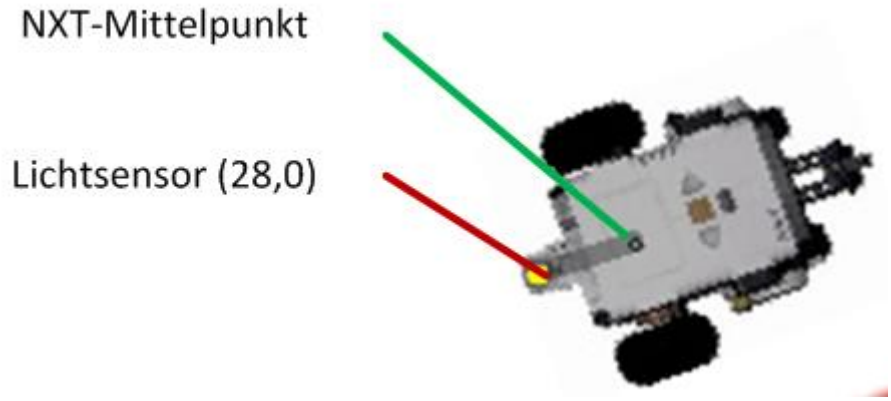
Der Prozess wird mit der Methode starten() gestartet.

```
pr.starten();
```

Die Sensoren und Gegenstände werden mit dem jeweiligen Konstruktor erzeugt und damit automatisch bei Rosi angemeldet.
Sie können nach der Erzeugung konfiguriert werden, z.B. bezüglich ihrer relativen Position zum Fahrzeugmittelpunkt.

LichtSensor

Der Konstruktor der LichtSensor-Klasse erwartet einen Integer-Parameter, der beim realen NXT die Portnummer angibt, an welcher der Sensor angeschlossen ist. In Rosi hat die Portnummer keine Bedeutung, muss aber mit angegeben werden.
Der Lichtsensor liefert beim realen NXT einen Helligkeitswert. Je nach Betriebsart repräsentiert dieser Wert das Umgebungslicht oder das Licht, welches von der Umgebung reflektiert wird. In Rosi wird standardmäßig der Farbwert des Untergrundes an der Sensorposition geliefert. Für den Farbwert gibt es in Java verschiedene Darstellungsmöglichkeiten, im einfachsten Fall setzt sich die Farbe aus drei Integer-Werten im Bereich von 0 bis 255 für die drei Grundfarben Rot, Blau und Grün zusammen. Zur leichteren Handhabung als Helligkeitswert, wird der Durchschnittswert der drei Farben verwendet (Summe durch 3). Hell sind hierbei Werte größer gleich 200.
Während beim realen NXT, bedingt durch die 4 Eingänge, nur maximal 4 Sensoren möglich sind, kann Rosi eine beliebige Anzahl Sensoren verwalten.



Methode positionieren(int vor, int seit)
Gibt die Position des Lichtsensors auf dem NXT in Rosi an. Der Mittelpunkt des NXT befindet sich etwa auf dem orangenen Button, von dort aus wird die Position in Pixeln angegeben. Der Lichtsensor erscheint im NXT als kleiner Kreis an einem grauen Strich. Der Mittelpunkt des NXT ist etwa auf der Mitte des Displays.

Methode liesWert()
Liefert den aktuellen Helligkeitswert als Integer im Bereich 0 bis 255 zurück.

Methode liesRot()
Liefert den aktuellen Rotanteil als Integer im Bereich 0 bis 255 zurück.

Methode liesBlau()
Liefert den aktuellen Blauanteil als Integer im Bereich 0 bis 255 zurück.

Methode liesGrün()
Liefert den aktuellen Grünanteil als Integer im Bereich 0 bis 255 zurück.

Methode hell()
Liefert true wenn der Sensor aktuell einen Helligkeitswert größer gleich 200 misst.

Methode dunkel()
Liefert true wenn der Sensor aktuell einen Helligkeitswert kleiner 200 misst.

EntfernungsSensor
Der EntfernungsSensor bildet den NXT – Ultraschall-Sensor in Rosi ab. Während beim realen NXT der Sensor allein durch seinen mechanischen Aufbau eine „Blickrichtung“ hat, kann in Rosi ein Winkel angeben werden. Ohne Winkelangabe werden 0° angenommen. Der Ausgangspunkt für die Messung ist von der Position des Sensors beim Aufruf von positionieren( ) abhängig. 0° Blickwinkel bedeuten dann gerade nach vorn, die Orientierung des Winkels ist im Uhrzeigersinn.

Methode positionieren()
Position des Entfernungssensors ausgehend vom NXT-Mittelpunkt, siehe LichtSensor.

Methode liesWert() liesWert(int Winkel)
Liefert den Abstand zum nächsten Gegenstand in Pixeln wahlweise mit Angabe eines Blickwinkels.

Methode getWert(int Winkel) getWert(int Winkel, Gegenstand g)
Liefert einen RadarWert mit Blickwinkel oder mit Blickwinkel und angegebenem Gegenstand.

RadarWert
Datenstruktur-Klasse zum Zusammenfassen aller relevanten Informationen einer Radar-Messung.
Beinhaltet **Winkel** und **Abstand** als Integer, das **Ziel** als Gegenstand und eine **Kennung** für den Zielgegenstand, also die Bezeichnung des Zielgegenstandes.

Radar
Die Radar-Klasse verwendet den EntfernungsSensor um einen kompletten 360°-Scan der Umgebung des NXT im Rosi durchzuführen.

Methode abtasten()
Führt einen kompletten 360°-Scan durch und speichert die Messergebnisse in einem Array mit 360 RadarWert-Einträgen.

Methode ZielMessen(ziel)
Diese Methode liefert den kürzesten Abstand zum angegebenen Ziel. Das Ziel wird als Objekt der Klasse Gegenstand angegeben. Intern wird die Methode abtasten() verwendet mit nachfolgender Auswertung des Arrays.

Methode liesWert(winkel)
Misst mit dem Abstandsensor im angegebenen Winkel. Hierbei wird überprüft, ob in der Richtung ein Gegenstand oder der Rand der Simulationsfläche auftaucht. Ergebnis ist ein RadarWert.

Gegenstand
Mit den Gegenstand-Klassen können Hindernisse oder Ziele für den Ultraschallsensor erzeugt werden. Beim Rechteck wird im Konstruktor die Position angegeben, x- und y-Koordinate der linken unteren Ecke sowie Breite und Höhe des Rechtecks. Beim Kreis wird im Konstruktor der Mittelpunkt und der Radius angegeben.

Methode hit(x,y)
Überprüft, ob der angegebene Punkt auf dem Gegenstand liegt.

Methode verschTo(x,y)
Verschiebt den Gegenstand um die angegebenen Abstände. Diese Methode wird auch beim Verschieben mit der Maus benutzt.

- **Rechteck** extends Gegenstand
- **Kreis** extends Gegenstand