Robby Projekt

Erweiterung des Greenfoot Roboter-Szenarios um Sensorik, Speicher und Hindernisumgehung

Adrian Schrader Moritz Jung Alexander Riecke

12. Dezember 2015

Zusammenfassung

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Inhaltsverzeichnis

1 Sensorik

1.1 Aufgabenstellung

Um Robby Anhaltspunkt für seine Aktionen zu geben, sollen zwei verschiedene Arten von Sensoren eingeführt werden, mit denen Robby seine Umgebung abtasten kann. Robby kann nicht durch eine Wand laufen, also sollte er erkennen können, ob in seiner Umgebung solch ein Hindernis auftaucht.

Eines der Hauptaktionen eines Roboters in diesem Szenario ist das Sammeln von Akkus für Energie. Um gezielter nach Akkus suchen muss Robby seine Umgebung nach ihnen abtasten.

wandHinten() Wenn, in Bezug auf Robbys Laufrichtung gesehen, ein Actor der Klasse Wand ein Feld hinter Robby steht, soll die Methode **`true`** zurückgeben, ansonsten **`false`**.

akkuVorne() [akkuRechts(), akkuLinks()] Wenn, in Bezug auf Robbys Laufrichtung gesehen, ein Actor der Klasse Akku ein Feld vor [rechts von, links von] Robby steht, soll die Methode **`true`** zurückgeben, ansonsten **`false`**.

1.2 Problematik

Diese vier Fälle können auf das Problem reduziert werden aus der Blickrichtung des Roboters und dem spezifischen Suchwinkel einen Vektor vom Roboter zum Suchfeld zu konstruieren, damit überprüft werden kann, ob die Methode `this.getOneObjectAtOffset(int v_x,int v_y,Class<?> c)` ein Objekt übergibt oder nicht. Die Mutterklasse Roboter löst die Aufgabe, in dem sie jeden einzelnen Suchvektor als einzelne Methode implementiert und in ihr die vier Blickrichtungen abfragt, um daraus einen fest einprogrammierten Vektor auszuwählen.

Diese Herangehensweise funktioniert zwar, ist jedoch für eine schlanke, wiederverwendbare, nachvollziehbare und skalierbare Klasse nicht geeignet. Um die Klasse evtl. später um Abfragen zusätzlicher Aktoren erweitern zu können, muss die Abfrage in einer einzigen Methode stattfinden. Diese errechnet dynamisch aus den Faktoren den gewünschten Vektor.

1.3 Lösung

Um die Lösung für dieses Problem zu verstehen ist es hilfreich den gesuchten Vektor $\vec{v}$ als Zeiger zu verstehen, dessen Betrag immer auf $|\vec{v}| = 1$ genormt ist. Aus dem Winkel θ von der Horizontalen lässt sich dann die Komponente in x und y-Richtung mithilfe von Sinus und Kosinus errechnen.

$$v_x = |\vec{v}| \cdot cos(\theta) \qquad v_y = |\vec{v}| \cdot sin(\theta) \tag{1.3.1}$$

Für unseren Anwendungsfall interessieren uns nur ganzzahlige Werte von v_x und v_y zwischen -1 und 1. Daher können wir die Domäne für θ enger eingrenzen.

$$\theta \in \left\{ k \cdot \frac{\pi}{2} \,|\, k \in \mathbb{N}_0 \right\} \tag{1.3.2}$$

Für die in Abschnitt1.2 besprochene Methode müssen wir jedoch zuerst den Winkel für die Laufrichtung und den Suchwinkel addieren, um auf den gesuchten Vektor zu kommen. Dieser soll dann in das Bogenmaßumgerechnet und zu den Komponenten verarbeitet werden. Über eine Abfrage der Methode `this.getOneObjectAtOffset(int v_x,int v_y,Class<?> c)` lässt sich dann der überprüfen, ob der gesuchte Actor existiert oder nicht.

```
/**
 * Der Sensor überprüft, ob sich neben der
 ↪  Laufrichtung von Robby ein
 * anderer Actor befindet.
 * @param direction Winkel von der Laufrichtung
 ↪  zum Suchfeld.
 * @param class Klasse des gesuchten Actors
 * @return boolean
 */
public boolean istObjektNebendran(int
 ↪  direction, Class<?> cl)
{
   double angle = (this.getRotation() +
    ↪  direction) / 180.0 * Math.PI;

   return (this.getOneObjectAtOffset(
          (int)Math.cos(angle),
          (int)Math.sin(angle), cl) != null);
}
```

Quellcode 1.1: Die implementierte Methode `istObjektNebendran(int,Class<?>)` aus Robby.java

2 Speicher

2.1 Aufgabenstellung

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.2 Problematik

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

2.3 Lösung

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis po-

suere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

3 Hindernisse

3.1 Aufgabenstellung

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

3.2 Problematik

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

3.3 Lösung

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Literatur

Quellcode 3.1: Vollständige, dokumentierte Version der Klasse Robby.java, die alle oben beschreibenen Zusatzfuntionen enthält.

```
1  import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
2
3  /**
4   * Die Klasse Robby ist eine Unterklasse von Roboter.
5   * Sie erbt damit alle Attribute und Methoden der Klasse Roboter.
6   */
7
8  public class Robby extends Roboter
9  {
10
11     /**
12      *
13      */
14     public Robby()
15     {
16
17     }
18
19     /**
20      * In der Methode "act" koennen Befehle / andere Methoden angewendet werden:
21      * Die Methoden werden dort nacheinander "aufgerufen", wenn man
22      * nach dem Kompilieren / uebersetzen den Act-Knopf drueckt.
23      */
24     public void act()
25     {
26
27     }
28
29     /**
30      * Der Sensor ueberprueft, ob sich in Laufrichtung des Roboters
31      * ein Akku befindet.
32      * @return boolean
33      */
34     public boolean akkuVorne()
35     {
36        return this.istObjektNebendran(0, Akku.class);
37     }
38
39     /**
40      * Der Sensor ueberprueft, ob sich rechts der Laufrichtung des Roboters
41      * ein Akku befindet.
42      * @return boolean
43      */
44     public boolean akkuRechts()
45     {
46         return this.istObjektNebendran(90, Akku.class);
47     }
48
49     /**
50      * Der Sensor ueberprueft, ob sich links der Laufrichtung des Roboters
51      * ein Akku befindet.
52      * @return boolean
53      */
54     public boolean akkuLinks()
55     {
56         return this.istObjektNebendran(-90, Akku.class);
57     }
58
59     /**
60      * Der Sensor ueberprueft, ob sich entgegen der Laufrichtung des Roboters
61      * eine Wand befindet.
62      * @return boolean
63      */
```

```
64      public boolean wandHinten()
65      {
66          return this.istObjektNebendran(180, Wand.class);
67      }
68
69      /**
70       * Der Sensor überprüft, ob sich neben der Laufrichtung von Robby ein
71       * anderer Actor befindet.
72       * @param direction Winkel von der Laufrichtung zum Suchfeld.
73       * @param class Klasse des gesuchten Actors
74       * @return boolean
75       */
76      public boolean istObjektNebendran(int direction, Class<?> cl)
77      {
78         double angle = (this.getRotation() + direction) / 180.0 * Math.PI;
79
80         return (this.getOneObjectAtOffset(
81                 (int)Math.cos(angle),
82                 (int)Math.sin(angle), cl) != null);
83      }
84
85  }
```