

Debugging

RoboCode Special Interest Topic

Summer 2011



Christian Riess, Eva Eibenberger

Pattern Recognition Lab (Computer Science Dep. 5)

Friedrich-Alexander-University Erlangen-Nuremberg

Was tun, wenn der Roboter sich „anders“ verhält?



- Jeder Programmierer macht Fehler – auch bei sorgfältiger Vorgehensweise
- Daher ist es wichtig, ein Programm sorgfältig zu prüfen
- Die Prüfung besteht im allgemeinen aus zwei Teilen:
 - Den Quellcode gegenlesen (**wird oft unterschätzt**)
 - Das Verhalten zur Laufzeit überprüfen (**Thema dieser Folien**)
- Möglichkeiten der Fehlersuche zur Laufzeit (Debugging)
 - Text-Konsole
 - Graphische Debugging-Möglichkeiten
 - Separate Programme
 - Debug-Bot



Fehlersuche in der Text-Konsole

- In der ersten Programmierübung wurden Anweisungen mit `System.out.println()` in der Konsole ausgegeben.
- In RoboCode wird die Ausgabe in die Konsole des jeweiligen Roboters umgeleitet
- Die Konsole kann mit einem Klick auf den Roboternamen geöffnet werden

```

printing at time 120
noticedCollision
(x: 119, 120, 782.0000000000003, 581.9999999999997, 782.0000000000003, 581.9999999999997, power = 2.0
120
didFire = true
collisionToFireCondition = 119, 120, 2.0, c.postLower = chriess.HalfMoon$Point@500b2175, chriess.HalfMoon$Point@a1b161
fired!
registering x = 782.0000000000003, y = 581, time = 119, speed = 14.0
SYSTEM: Exception occurred in robocode.CustomEvent
java.lang.NullPointerException
    at chriess.HalfMoon.onCustomEvent(HalfMoon.java:102)
    at robocode.CustomEvent.dispatch(CustomEvent.java:108)
    at robocode.Event$HiddenEventHelper.dispatch(Event.java:244)
    at net.sf.robocode.security.HiddenAccess.dispatch(HiddenAccess.java:194)
    at net.sf.robocode.host.events.EventManager.dispatch(EventManager.java:487)
    at net.sf.robocode.host.events.EventManager.processEvents(EventManager.java:460)
    at net.sf.robocode.host.proxies.BasicRobotProxy.executeImpl(BasicRobotProxy.java:412)
    at net.sf.robocode.host.proxies.BasicRobotProxy.execute(BasicRobotProxy.java:323)
    at robocode.AdvancedRobot.execute(AdvancedRobot.java:565)
    at chriess.HalfMoon.run(HalfMoon.java:52)
    at net.sf.robocode.host.proxies.HostingRobotProxy.run(HostingRobotProxy.java:220)
    at java.lang.Thread.run(Thread.java:62)
printing at time 121
122
  
```



- Maßnahmen, um nicht mit Ausgaben des Roboters geflutet zu werden:
 - Kampfgeschwindigkeit drosseln (Regler Mitte unten am Spielfeld)
 - Nur Ausgaben schreiben, wenn Bedingungen erfüllt sind, die für die Fehlersuche relevant sind



Fehlersuche in der Text-Konsole: Beispiel

- Angenommen, wir haben offensichtlich einen Programmierfehler, und vermuten ihn in der Berechnung der x- und y-Koordinate des Gegners.
- Dann kann man die geschätzte Position des Gegners in der Konsole ausgeben:

```

200 public void onScannedRobot(ScannedRobotEvent e) {
201     double angle = (fourPI + (getHeadingRadians() + e.getBearingRadians())) % twoPI;
202     double enemyX = getX() + Math.cos(angle) * e.getDistance();
203     double enemyY = getY() + Math.sin(angle) * e.getDistance();
204
205     System.out.println("Enemy sighted at x = " + enemyX + ", y = " + enemyY);
206 }

```

- Tatsächlich widerspricht $y=4.6$ ganz offensichtlich den Koordinaten des Gegners

```

-----
Round 1 of 10
-----
Enemy sighted at x = 753.1631198692834, y = 4.609539868349515
Enemy sighted at x = 748.1631198692835, y = 4.609539868349543
Enemy sighted at x = 695.1631198692835, y = 4.6095398683496285
delta predicted enemy pos and real (18 steps): 0.0/0.0
Enemy sighted at x = 631.1631198692835, y = 4.609539868349557

```





Fehlersuche per graphischer Ausgabe

- Ein Roboter kann Zeichenoperationen in der Arena durchführen (Kreise, Rechtecke, ...)
- Eignet sich hauptsächlich, um geometrische Berechnungen visuell zu überprüfen
- Individuelle Zeichnungen werden in der Roboterkonsole unter „paint“ aktiviert



- Im Program muss `import java.awt.*;` angegeben sein, und die Funktion `void onPaint(Graphics2D g)` implementiert sein.
`g` kann man sich als Zeichenfläche oder Folie vorstellen, die über die Arena gelegt wird. Alles was in dieser Funktion auf `g` gezeichnet wird, erscheint in der Arena.



Fehlersuche per graphischer Ausgabe: Beispiel

- Nochmal die Position des Gegners: Diesmal auf dem Spielfeld eingezeichnet
- enemyX und enemyY müssen Klassenvariablen sein, damit sie in onPaint zur Verfügung stehen
- onPaint zeichnet ein rotes Rechteck mit grünem Kreis um die geschätzte Position

```

200 double enemyX;
201 double enemyY;
202 public void onScannedRobot(ScannedRobotEvent e) {
203     double angle = (fourPI + (getHeadingRadians() + e.getBearingRadians())) % twoPI;
204     enemyX = getX() + Math.cos(angle) * e.getDistance();
205     enemyY = getY() + Math.sin(angle) * e.getDistance();
206
207     System.out.println("Enemy sighted at x = " + enemyX + ", y = " + enemyY);
208 }
209 public void onPaint(Graphics2D g) {
210     g.setColor(java.awt.Color.RED);
211     g.fillRect((int)enemyX-3, (int)enemyY-3, 7, 7);
212     g.setColor(java.awt.Color.GREEN);
213     g.draw(new Ellipse2D.Double(enemyX-10, enemyY-10, 20, 20));
214 }

```

- Auf dem Spielfeld sieht man, dass die Positionsschätzung in einer ganz anderen Richtung liegt als der Gegner.



Fehlersuche mit präProgrammen



- Das Überprüfen von Berechnungsergebnissen mit unterschiedlichen Fällen ist es oft zu langwierig, auf eine bestimmte Situation in der Arena zu warten.
- Oft ist es effektiver, eine separate, selbständig ausführbare Klasse zu schreiben, in der man gezielt verschiedene Eingaben berechnet und das Ergebnis ausgibt.
- Dies ist besonders wichtig, wenn verschiedene Fallunterscheidungen abgeprüft werden sollen.
- Wir haben ein „Übungsblatt“ vorbereitet, um diesen Ansatz zu illustrieren:
Auf der Projekthomepage -> Aufgabenblatt 2

Fehlersuche mit separaten Programmen



- Das Überprüfen von Berechnungsergebnissen mit unterschiedlichen Fällen ist es oft zu langwierig, auf eine bestimmte Situation in der Arena zu warten.
- Oft ist es effektiver, eine separate, selbständig ausführbare Klasse zu schreiben, in der man gezielt verschiedene Eingaben berechnet und das Ergebnis ausgibt.
- Dies ist besonders wichtig, wenn verschiedene Fallunterscheidungen abgeprüft werden sollen.
- Wir haben ein „Übungsblatt“ vorbereitet, um diesen Ansatz zu illustrieren:
Auf der Projekthomepage -> Aufgabenblatt 2

Debug-Bot



- Um das Verhalten für ein bestimmtes Gegner-Verhalten zu überprüfen, schafft man sich am besten einen Gegner, der sich genau so verhält.
- Die Sample-Bots sind schon recht gut als Trainingspartner, z.B.:
 - Tracker rückt dem eigenen Roboter auf die Pelle
 - Crazy ist schwierig zu treffen
 - Walls ist auch schwierig zu treffen
 - ...
- Für alles, was nicht über die Sample-Bots abgedeckt ist, kann man eine Kopie des eigenen Bots anlegen, die genau das Testverhalten implementiert

- Eine zweite Möglichkeit ist die Ausgabe von internen Daten.

Wir können die Fehlersuche auf den Folien 4 und 6 folgendermaßen erweitern:

- Lege eine Kopie von Walls an
- Lass die Kopie von Walls auf seiner Konsole jede Runde seine Position ausgeben: Das ist unsere „ground truth“
- Damit vergleichen wir die Ausgabe unseres Schätzers