Christoph Lüth
Martin Ring

2. Übungsblatt

Ausgabe: 13.05.14
Abgabe: 27.05.14

In diesem Übungsblatt lernen wir die Simulation kennen, die uns für den Rest der Veranstaltung begleiten wird. Während für dieses Übungsblatt die Simulation noch sehr einfach gehalten ist, wird sie dannim weiteren Verlauf der Veranstaltung ständig erweitert.

Als erstes benötigen Sie das Tool `sbt`. Installationsanweisungen dazu finden sich unter

`http://www.scala-sbt.org/release/docs/Getting-Started/Setup.html`

Nun laden Sie das Archiv `uebung-02.zip` von der Webseite herunter, und entpacken es. In der Kommandozeile starten Sie `sbt` im Verzeichnis `uebung-02`. In der `sbt`-Konsole kann durch den Befehl `eclipse` ein Eclipse Projekt generiert werden, was dann mit der Scala-IDE zur Bearbeitung geöffnet werden kann. Diese finden Sie hier:

`http://scala-ide.org/download/sdk.html`

Die für die Simulation benötigten Bibliotheken sind in das Projekt bereits eingebunden. (Sie benötigen eine Internetverbindung, damit `sbt` die Abhängigkeiten herunterladen kann)

In der Datei `/src/main/scala/uebung02/Uebung02.scala` ist ein minimaler Einstiegspunkt zu finden. `Simulation.start(teams = 1)` erzeugt eine neue Simulation mit einem Team. (In der vorliegenden Version sind mehrere Teams noch nicht unterstützt.) Pro Team stehen 25 Roboter zur Verfügung. Diese erhalten Sie über das Callback, das `Simulation.onReady` übergeben werden kann:

```
val mine = Simulation.start(teams = 1)
mine.onReady { robots => assert(robots.size == 25) }
```

Die einzelnen Roboter implementieren das Interface `RobotControls` über das sie auch steuerbar sind:

```
trait RobotControls {
  val name: String
  def availableSensors: Traversable[String]
  def addEventListener(sensorId: String, handler: String => Unit)
  def removeEventListener(sensorId: String, handler: String => Unit)
  def availableOutputs: Traversable[String]
  def setPowerLevel(outputId: String, level: Double)
}
```

Jeder Roboter hat einen eindeutigen Namen (`name`). Über `availableSensors` erhalten Sie eine Liste der verfügbaren Sensoren (bzw. deren IDs), denen man mit `addEventListener` zuhören kann. Ein Callback kann mit `removeEventListener` wieder entfernt werden. Über `availableOutputs` erhalten Sie die Liste der Ausgänge die mit `setPowerLevel` gesetzt werden können. Folgende Sensoren stehen zur Verfügung:

`"gps"` Die Koordinaten des Roboters. (Format `"00.0000N 00.0000W"`)

`"batteryLevel"` Der Ladestand der Batterie. (Format `"0.0%"`)

`"gyroscope"` Die aktuelle Winkelgeschwindigkeit. (Format `"0.0000"`)

`"accelerometer"` Die lineare Beschleunigung. (Format `"0.0000 0.0000"`)

`"laser"` Die Werte der drei Laser (links, mitte, rechts). Liegen zwischen 0 und 1 und zeigen an, wie schnell das Licht reflektiert wurde, und damit, in welcher Entfernung ein Hindernis detektiert wurde. Der Wert 0 bedeutet, dass der Laser unmittelbar auf ein Ziel getroffen ist, der Wert 1 bedeutet, dass keine Reflektion wahrgenommen wurde. Die Werte sind relativ zur Stärke der Laser die über die entsprechenden Ausgänge gesetzt wird. (Format `"0.0000 0.0000 0.0000"`)

`"goldDetector"` Zeigt an wie viel Gold in der unmittelbaren Umgebung des Roboters ist; höhere Werte bedeuten mehr Gold. (Format `"0.0000"`)

Über folgende Ausgänge kann ein intakter Roboter gesteuert werden:

`"leftWheel"`, `"rightWheel"` Steuern das linke bzw. rechte Rad des Roboters über Werte zwischen -1 und 1

`"laserL"`, `"laserC"`, `"laserR"` Steuern den linken, mittleren bzw. rechten Laser des Roboters über Werte zwischen 0 und 1

Alle Ausgänge verbrauchen Batterie je nach gesetztem `powerLevel`. Wenn mindestens ein Callback an einem Sensor registriert ist, verbraucht dieser ebenfalls Strom.

2.1 *Ab in die Mine* *12 Punkte*

Machen Sie sich mit der Simulation vertraut, indem Sie eine Steuerung für die Roboter implementieren, welche möglichst viel Gold in die Basis schiebt. Ihre Basis ist der grün markierte Bereich von `80N 80W` bis `50N 50W`. Über `onGoldCollected` können Sie nachvollziehen, wie viel Gold sich in der Basis befindet.

Verkapseln Sie je Roboter alle veränderlichen Zustände in einem Objekt, auf welches Sie über Methoden zugreifen, die die nächsten Aktionen berechnen. Versuchen Sie dabei, die Sensordaten intelligent zu verknüpfen und denken Sie daran, dass die Sensoren alle Strom verbrauchen, solange sie ihnen zuhören. Sie sollten also auch versuchen, bestimmte Sensoren nur zu aktivieren, wenn die Sensordaten auch aktuell benötigt werden.

- Ob sich der Roboter in der Basis befindet können sie beispielsweise überprüfen, indem sie entweder die GPS Daten nutzen oder den Ladestand der Batterie beobachten. Wenn dieser steigt befindet sich der Roboter wahrscheinlich in der Basis.
- Die Geschwindigkeit des Roboters können sie sowohl über das GPS als auch über den Accelerometer approximieren.
- Versuchen sie die Orientierung des Roboters durch Kombination von GPS und Gyroskop zu ermitteln.
- Da der Golddetektor in alle Richtungen ausschlägt, müssen sie weitere Beobachtungen über den Laser als auch über den Wiederstand machen (Wenn der Roboter sich bei gleicher Stromzufuhr langsamer bewegt schieben wir etwas vor uns her).

Zum Einstieg sollten sie ein Verhalten implementieren, welches Kollisionen vermeidet und die Batterien nicht leer werden lässt. Dieses Verhalten können sie dann langsam erweitern.

Wenn Ihre Implementierung Sie nicht zufrieden stellt, dann verzweifeln Sie nicht! Wir lernen in der Vorlesung noch viele Konzepte kennen, die es uns ermöglichen, aus dem Callback-Zustands-Wust ein schickes, sicheres und skalierbares Programm zu machen.

2.2 *ScalaCheck* *8 Punkte*

Spezifizieren Sie, wie sich die Roboter verhalten sollen indem Sie entsprechende Properties in ScalaCheck implementieren. (Ein Gerüst dafür ist im Projekt zu finden.) Viele Eigenschaften (wie beispielsweise Kollisionsfreiheit) sind schwierig zu überprüfen, aber sie sollten insbesondere spezifizieren, wie der Roboter auf bestimmte Eingaben reagieren soll. Dabei sollten sie beliebige Roboterzustände generieren, indem sie einen eigenen Generator dafür schreiben.