Christoph Lüth
Martin Ring

3. Übungsblatt

Ausgabe: 27.05.14
Abgabe: 12.06.14

3.1 *Pimp the Future* — *8 Punkte*

Schreiben sie folgende Erweiterungsmethoden für Futures:

```
// continue the computation with the provided function and return the result
// as a new Future
def continue[S](cont: Try[T] => S): Future[S]

// returns the result if available or throws an exception otherwise
def now: T

// returns either the result of this or that future, depending on
// which one completes first. (don't use firstCompletedOf)
def |(that: Future[T]): Future[T]
```

Schreiben sie außerdem folgene Methoden (nicht als Erweiterungen):

```
// retries to execute the block up to n times before it returns a
// failed future.
def retry(n: Int)(block: => Future[T]): Future[T]

// returns a future that will never complete
def never[T]: Future[T]

// turns a sequence of futures into a single future of the sequence of results
def sequence[T](futures: Seq[Future[T]]): Future[Seq[T]]

// returns a future that completes after the provided timespan expires
def timeout(t: Duration): Future[Unit]

// adds a timeout to a future: if it does not complete in provided timespan,
// complete with a Failure
def withTimeOut(t: Duration)(f: Future[T]): Future[T]
```

3.2 *Minenarbeiten* — *12 Punkte*

Achtung: Diese Aufgabe können Sie erst nach der Vorlesung am 03.06. bearbeiten.

Dieses Mal wollen wir etwas Ordnung in unsere Mine bringen!

- Finden sie geeignete Datenstrukturen um die einzelnen Sensordaten zu repräsentieren.
- Implementieren sie Methoden für alle Sensoren der Roboter, welche ihnen Observables mit den Sensordaten zurückgeben.
- Verwenden sie die Kombinatoren auf Observables, um die Sensordaten zu verknüpfen.
- Reduzieren sie die Datenmenge weiter durch sinnvolle Zusammenfassung von Werten. Implementieren sie zum Beispiel eine Kollisionserkennung auf Grundlage des Accelerometers.
- Implementieren sie mindestens folgende asynchrone Methoden auf ihren Robotern.

```
// moves the robot to the specified position while avoiding collisions
def moveTo(pos: Position): Observable[Unit]

// watches the batteryLevel as well as the position of the robot
// and reports when the battery level runs critically low (i.e.
// it is estimated as just enough to move back to the base)
def criticalBattery: Observable[Unit]

// lets the robot search for gold and report all positions of
// detected nuggets
def findGold: Observable[Position]

// brings the robot back to the base as quick as possible
def goToBase: Observable[Unit]
```

- Lassen sie einige Roboter patroullieren, um Gold zu suchen. Solche Kombinationen sollten nun möglich sein:

  ```
  (findGold takeUntil criticalBattery) ++ goToBase
  ```

- Implementieren sie mindestens ein weiteres Verhalten für einige Roboter, welches das Gold einsammelt.