Real World Haskell

Projekte 1

Julian Fleischer, Alexander Steen

Donnerstag, 25. Juli 2013

Heute gilt es ein Projekt auszuwählen und in Gruppen von 2 bis 4 Studenten zu bearbeiten. Für alle Projekte gilt, dass sie am Montag um 10 Uhr c.t. vorzustellen sind, inklusive einer kleiner Demo-Vorführung und einem kurzen Code Review.

Ihre Projekte müssen alle als cabal-Projekt vorliegen und am Montag als Tarball-Archiv (`.tar.gz` / `.tgz`) abgegeben werden. Sie können ein solches Archiv mit Hilfe des Befehls `cabal sdist` herstellen. Orientieren Sie sich an den vorhandenen Beispielen auf der Homepage:

BouncingBalls
`http://page.mi.fu-berlin.de/scravy/realworldhaskell/tag4/BouncingBalls-1.0.tar.gz`

ChatServer
`http://page.mi.fu-berlin.de/scravy/realworldhaskell/tag4/ChatServer-1.0.tar.gz`

EchoServer
`http://page.mi.fu-berlin.de/scravy/realworldhaskell/tag4/EchoServer-1.0.tar.gz`

Netcat
`http://page.mi.fu-berlin.de/scravy/realworldhaskell/tag4/Netcat-1.0.tar.gz`

In allen Projekten werden die Grafik-Bibliothek `gloss` oder das `network` package verwendet. Bitte konsultieren Sie dafür die offizielle Dokumentation. Auch zu dem `base` package werden Fragen auftreten (dieses enthält beispielsweise die `Prelude` und `Control.Monad` oder `System.IO`)!

base
`http://hackage.haskell.org/package/base`

gloss
`http://hackage.haskell.org/package/gloss`

network
`http://hackage.haskell.org/package/network`

Sollten Sie Fragen zu `Cabal` oder `GHC` haben, konsultieren Sie bitte das entsprechende Benutzerhandbuch:

The Glorious Glasgow Haskell Compilation System User's Guide
`http://www.haskell.org/ghc/docs/7.6.3/html/users_guide/`

Cabal User Guide
`http://www.haskell.org/cabal/users-guide/`

Am Freitag findet keine Sitzung in der Uni statt. Es steht Ihnen und Ihrem Team frei einen oder mehrere Termine zu wählen und sich an einem von Ihnen gewählten Ort zu treffen. Wir erwarten durchaus, dass sie insgesamt wenigstens 6 Stunden in die Projekte investieren.

Game Of Life

Programmieren Sie Conways Game of Life mit Hilfe des Grafikframeworks `gloss`. Informationen zu Conways Game of Life finden sie in der Wikipedia:
`http://en.wikipedia.org/wiki/Conway's_Game_of_Life`

Gehen Sie kleinschrittig vor:

1. Schreiben sie eine GUI Anwendung mit Hilfe der `simulate` Funktion aus dem Paket `Graphics.Gloss.Interface.Pure.Simulate`
2. Zeichnen Sie zuerst mehrere Quadrate in ihre Zeichenfläche. Überlegen Sie sich dazu einen geeigneten Datentyp für ihre World!
3. Aktualisieren Sie nun ihre World und ihre Quadrate entsprechend den Regeln des Game of Life! Sie können zunächst Quadrate manuell in ihrem Code als lebendig markieren.
4. Erweitern Sie ihre Simulation indem sie statt `simulate` die Funktion `play` aus dem Paket `Graphics.Gloss.Interfac` verwenden. Durch das Klicken eines Quadrates soll man nun den Zustand (lebendig oder tot) verändern können.

Snake

Programmieren Sie Snake! Sollten Sie Snake nicht kennen und niemals ein Nokia-Handy besessen haben, Informationen zu Snake finden Sie in der Wikipedia unter `http://en.wikipedia.org/wiki/Snake_(video_game)`.

Gehen Sie kleinschrittig vor:

1. Entwickeln Sie zunächst mit Hilfe von `play` aus dem Paket
   `Graphics.Gloss.Interface.Pure.Game`
   eine Anwendung in der ein Rechteck mit Hilfe der Pfeiltasten durch die Gegend bewegt werden können.

2. Erweitern Sie ihre Anwendung, dass das Rechteck sich immer in die aktualle Richtung weiterbewegt und die Pfeiltasten nur noch eine Richtungsänderung ermöglichen.

3. Erweitern Sie ihre Anwendung, so dass (zunächst nach einem festen Muster) Kreise auf dem Spielfeld aufpoppen, die man mit dem sich bewegenden Rechteck einsammeln kann.

4. Erweiteren Sie nun noch ihre Anwendung, so dass ihre Schlange länger wird mit jedem aufgeschnappten Kreis. Hierfür müssen Sie ihren Welt-Datentyp entsprechend anpassen, da ja nun komplexe Formen entstehen können, die sich auch noch weiter bewegen. Was müssen Sie speichern?

Pong

Programmieren Sie Pong in Haskell unter Verwendung der Grafik-Bibliothek `gloss`. In diesem Spiel gibt es einen rechteckigen Ball und zwei Panele die durch zwei Benutzer einzeln nach oben oder nach unten gesteuert werden können. Informationen zu Pong finden sie u.a. in der Wikipedia:
`http://en.wikipedia.org/wiki/Pong`.

Gehen Sie kleinschrittig vor:

1. Entwickeln Sie zunächst mit Hilfe von `play` aus dem Paket
   `Graphics.Gloss.Interface.Pure.Game`
   eine Anwendung in der zwei Rechtecke nach oben und unten bewegt werden können, beispielsweise durch die Pfeil -auf und -ab Tasten für das rechte und `Q` und `W` für das linke.
2. Erweitern Sie ihre Anwendung durch einen Punktestand der an einer Stelle des Spielfelds angezeigt wird. Ihren Punktestand müssen Sie natürlich auch in ihrem World-Datentyp speichern.
3. Fügen Sie nun einen Ball hinzu der in der Mitte startet und zunächst schräg nach oben rechts (wie in der Bouncing Balls Anwendung) startet.
4. Nun müssen sie überprüfen ob der Ball eines der Panels trifft und ggf. die Richtung ändern.
5. Wenn Sie so weit gekommen sind haben Sie schon fast ein komplettes Pong-Spiel gebaut. Was ihnen jetzt noch fehlt ist etwas Spiellogik, wie etwa Starten, Zählen von Ausbällen, Beenden des Spiels. Ihnen sind keine Grenzen gesetzt das Spiel nach Ihrem Belieben zu erweitern!
6. Sie können noch hinzufügen, dass der Ball verschieden stark beschleunigt wird oder in eine andere Richtung abgelenkt wird, je nach Geschwindigkeit die ein Panel im Moment des zurückschlagens hat.

Hinweise:

Hier ist eine mögliche Datenstruktur für ihre World:

```
data World = World {
    paddleLeftPos :: Float,
    paddleRightPos :: Float,
    ballPos :: (Float, Float),
    ... weitere Eigenschaften ...
  }
```

Space Invaders

Programmieren Sie Space Invaders in Haskell unter Verwendung der Grafik-Bibliothek `gloss`. Bei diesem Spiel wandert eine Reihe von Monstern die in mehreren waagerechten Reihen angeordnet sind von oben nach unten und von links nach rechts. Je weiter die Monster nach unten gelangen, desto schneller werden sie.

Der Spieler kann durch die Links- und Rechts- Pfeiltasten ein dreieckiges Raumschiff nach links und rechts bewegen. Mit Hilfe der Leertaste kann der Spieler einen Schuss abfeuern der ein Monster verletzen oder komplett zerstören kann.

Weitere Informationen zu Space Invaders gibt es u.a. in der Wikipedia:
`http://en.wikipedia.org/wiki/Space_Invaders`

Gehen Sie kleinschrittig vor:

1. Zeichnen Sie zunächst 6x4 Rechtecke stellvertretend für Monster und ein Dreieck, stellvertretend für ihr Raumschiff. Benutzen Sie aber von Anfang an die `play` Funktion aus dem Paket `Graphics.Gloss.Interface.Pure.Game`

2. Erweitern Sie nun ihr Programm dahingehend, dass sie das Raumschiff durch Nutzung der Pfeiltasten nach links und rechts bewegen können.

3. Erweitern Sie nun ihr Programm, so dass nach Drücken der Space-Taste ein kleiner Kreis an der Spitze des Dreiecks erscheint. Dieser Kreis muss dann natürlich weiter nach oben wandern (das ist das Geschoss mit dem Sie die Monster abschießen). Vergessen Sie nicht dieses Geschoss aus der Welt zu entfernen, wenn es den oberen Bildrand verlassen hat! Bedenken Sie, dass es mehrere Geschosse geben kann!

4. Nun müssen Sie noch implementieren, dass das GEschoss Monster treffen kann und dass die Monster sich alle n Sekunden entlang eines vorgegebenen Pfades (z.B. immer 10 Pixel nach unten) weiter auf den Spieler zu bewegen. Das Spiel ist zu Ende wenn ein Monster den unteren Bildrand (oder Spielfeld rand kurz über dem Raumschiff) erreicht.

GUI Chat

Programmieren sie einen grafischen Chatraum. Sie müssen hierfür *keinen* Server bauen sondern können den in der Vorlesung dargestellten Chat-Server verwenden. Sie müssen lediglich einen Chat-Client bauen, der die gezeichneten Formen an den Server sendet und empfangene Formen zeichnet.

Sie benötigen hierfür die Funktionen des Pakets
`Graphics.Gloss.Interface.IO.Game`
Dieses Paket bietet die Funktion `playIO` an, die es ihnen erlaubt IO-Aktionen zu benutzen um die Welt zu aktualisieren und Ereignisse zu behandeln.

Gehen Sie kleinschrittig vor:

1. Entwickeln Sie zunächst eine kleine Anwendung in der man beispielsweise Kreise durch point and click setzen kann. Verwenden Sie zunächst `play` aus dem Paket
   `Graphics.Gloss.Interface.Pure.Game`.
2. Erweitern Sie nun Ihren Canvas auf dem Sie Kreise zeichnen können indem sie `play` durch `playIO` austauschen und ihre Funktionen dergestalt modifizieren, dass sie einen monadischen return-Wert haben (siehe Signaturen von `play` vs. `playIO`).
3. Sie kennen bereits den ChatServer und den ChatClient Netcat. Sie müssen nun an Stelle von Textnachrichten Koordinaten über den ChatServer versenden. Sie müssen den Server hierfür nicht modifizieren. Es sollte sich jetzt auch erschließen, warum Ihre `playIO` Funktion monadisch ist: Sie wird Nachrichten aus dem Netzwerk lesen und empfangen.

Hinweise:

- Überlegen Sie sich welche Datentypen sie verwenden! Welcher Datentyp symbolisiert ihre Welt (Orientieren Sie sich an dem Bouncing Balls Beispiel)?
- Es ist hilfreich wenn Sie ihre Datentypen automatisch in einen String überführen können und aus einem String auslesen, um sie dann über den ChatServer (der ja Textnachrichten verarbeitet) zu verschicken und zu empfangen. Hierfür empfiehlt es sich die Typklassen `Read` und `Show` zu implenentieren. Das kann der Compiler glücklicherweise automatisch:
  `data World = World { objects :: [Object] } deriving (Eq, Show, Read)`

Schiffe versenken

Programmieren Sie eine Schiffe versenken Anwendung in der man Schiffe versenken über das Netwerk spielen kann. Implementieren Sie ein Spielfeld der Größe 10x10. Lesen Sie für dieses Projekt auch die Hinweise zu dem Projekt `GUI Chat` und `Game of Life`.

Gehen Sie kleinschrittig vor:

1. Lesen Sie die Hinweise `GUI Chat` und `Game of Life`. Welche Informationen müssen Sie über das Netzwerk vermitteln? Was müssen Sie zeichnen?