Reactive Programming

Deze week schrijven we een *reactive interface* voor een bestaande library, MBot. Deze bibliotheek werd eerdere jaren ontwikkeld speelgoedrobotten aan te sturen vanuit Haskell. Je kan hem terugvinden op Hackage en in je stack-project importeren door hem als volgt toe te voegen aan je "extra-dependencies" in `stack.yaml`:

```
extra-deps:
- MBot-0.2.4.1
- reactive-banana-1.2.0.0
```

In je *cabal*-bestand moet je hem natuurlijk ook nog toevoegen. We schrijven een bibliotheek, dus iets gelijkaardigs aan het onderstaande zou moeten volstaan:

```
library
  hs-source-dirs:      lib
  build-depends:       base >= 4.7 && < 5
                     , MBot >= 0.2 && < 0.3
                     , reactive-banana >= 1.2 && < 1.3
  exposed-modules:     ReactiveMBot
```

- Op Windows zou MBot gewoon moeten werken.
- Op Osx voeg je volgende vlaggen toe voor compilatie: `-framework IOKit -framework CoreFoundation`.
- Op sommige Linux Distro's zal je om met de MBot hardware te verbinden, mogelijks nog de "HID API" bibliotheek moeten installeren op je computer (`hidapi`-package of iets gelijkaardig). Eventueel voeg je de lijn `KERNEL=="hidraw*", ATTRS{busnum}=="1", ATTRS{idVendor}=="0416" ATTRS{idProduct}=="ffff", MODE="0666"` toe aan je udev rules om als gewone gebruiker met de HID API te kunnen verbinden.

Je kan src/Minimal.hs gebruiken om de verbinding met de MBot te testen.

De startcode van de bibliotheek zelf, met documentatie, vind je terug in src/ReactiveMBot.lhs. Hieronder vind je een mogelijke manier om deze te gebruiken.

```
module Week11Opgave where

import ReactiveMBot
import Reactive.Banana.Frameworks (MomentIO, reactimate)
```

Onze main methode zal een reactieve connectie opzetten en het `avoidWall` netwerk uitvoeren.

```
main = do mbot <- newMBot
          runMBot (avoidWall mbot) mbot
```

Wat doen we met de wielen?

```haskell
drive :: ReactiveMBot -> Float -> IO ()
drive mbot dist | dist > 20 = forward mbot
                | otherwise = left mbot
```

Wat doen we met de lichten?

```haskell
setLights :: ReactiveMBot -> Float -> IO ()
setLights mbot dist | dist > 20 = green leftLight mbot
                    | otherwise = blue leftLight mbot
```

Combinatie van beide. We `<$>`-mappen onze stroom van afstanden naar een `Event (IO ())`, of een stroom van reacties op die afstanden. Met `reactimate` brengen we dit in het `MomentIO ()` netwerk.

```haskell
avoidWall :: ReactiveMBot -> MomentIO ()
avoidWall mbot = do dist <- distance mbot
                    reactimate $ (setLights mbot) <$> dist
                    reactimate $ (drive mbot) <$> dist
```