Übungsblatt 2

Throat-Clearing

a.k.a. Imports, damit der Code funktioniert.

```
import Data.Functor
import Data.Monoid
```

Functor

Sie haben in der Vorlesung die Typklasse `Functor` kennengelernt. Zur Erinnerung:

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b
```

Nehmen sie an, sie hätten folgende Datentypen gegeben, für die alle eine `Functor`-Instanz existiert und eindeutig ist:

```
data Identity a = Identity { unIdentity :: a }

data Vielleicht a = Etwas a
                  | Nichts

data EntwederOder b a = Entweder a
                      | Oder b

data GameVector b a = V3 a a a
                    | VStrange [a]
                    | Neighbours [GameVector b a]
                    | EntwederOder b (GameVector b a)
```

Schreiben sie hierzu die jeweiligen `Functor`-Instanzen.

Besser und allgemeiner

Vereinfachen und verallgemeinern sie folgenden Ausdrücke so weit wie möglich und geben die sie dadurch entstehenden Typsignaturen an. Bedenken sie, dass wenn sie auf eine Typklasse abstrahieren, sie die gesamten Gesetze der Typklasse benutzen können.

Kann die Funktion nachher mehr als vorher?

Bonus: Hat sich an der Laufzeit etwas verändert?

```haskell
mystery1 :: [[a]] -> [[a]]
mystery1 = map (++[])

mystery2 :: (Eq a) => a -> a -> a -> Bool
mystery2 x y z
  | x == y || y == z = True
  | x == y && y == z = True
  | x /= z           = False
  | y /= z           = False
  | x == y || y /= z = True
  | otherwise        = False

mystery3 :: Eq a => (a -> a) -> a -> [a] -> [a]
mystery3 f a l = post a l'
   where
     l' = map f l
     post a (x:xs)
         | a == x    = x : post a xs
         | otherwise =     post a xs
     post _ [] = []

mystery4 :: (Int -> Bool)
         -> Vielleicht (EntwederOder String Int)
         -> Vielleicht (EntwederOder String Bool)
mystery4 f (Etwas (Entweder a)) = Etwas . Entweder . f $ a
mystery4 _ (Etwas (Oder b))     = Etwas (Oder b)
mystery4 _  Nichts              = Nichts
```

Bonus

Es gibt von dem bekannten Spiel 2048 eine Haskell-Implementation für die Kommandozeile in unter 100 Zeilen. Diese ist zu finde unter https://github.com/gregorulm/h2048/blob/master/h2048.hs

Sie können diesen Code mit `GHC` kompilieren oder im `ghci` ausführen (`main` ist die Start-Funktion).

Was für Prinzipielle Vorgehenspunkte können sie erkennen? Eine kleine Erklärung gibt es im Blog der Erstellers: http://gregorulm.com/2048-in-90-lines-haskell/

Keine Angst, sie müssen dies noch nicht verstehen, aber es soll verdeutlichen, wie viel man mit extrem wenig erreichen kann. Viele der Abgabeprojekte werden in dieser Größenordnung liegen (aber meist noch so etwas wie ein GUI o.ä. benötigen). Versuchen sie einfach den Code kaputtzuspielen (z.b. Tasten ändern, Siegbedingung ändern, Cheats einbauen, ...).

Viel Spass beim Spielen! :)