Fortgeschrittene funktionale Programmierung in Haskell

Übungszettel 5

Aufgabe 5.1:

In der Vorlesung wurde MaybeT explizit vorgestellt. Erstellen sie einen Monad-Transformer EitherT für Either. Zur Erinnerung: Either ist definiert als:

```
data Either a b = Left a
                | Right b
```

Erstellen sie hierzu die Instanzen für

- Functor

```
instance Functor f => Functor (EitherT f) where
    fmap :: (a -> b) -> (EitherT f) a -> (EitherT f) b
```

- Applicative

```
instance Applicative f => Applicative (EitherT f) where
    pure  :: a -> f a
    (<*>) :: f (a -> b) -> f a -> f b
```

- Monad

```
instance Monad m => Monad (EitherT m) where
    return :: a -> (EitherT m) a
    (>>=)  :: (EitherT m) a -> (a -> (EitherT m) b) -> (EitherT m) b
```

Beispielcode mit Definitionen und Testfällen finden sie in der Datei `eitherT.hs`. Nutzen sie dies als Ausgangsbasis.

Aufgabe 5.2:

Für die folgenden Aufgaben benötigen sie die externe `mtl`[1] in der der RWST-Stack bereits implemntiert ist.

Richten sie sich hierzu ein einem Verzeichnis eine lokale Arbeitsumgebung (sandbox) ein, indem sie folgende Befehle verstehen und anschließend ausführen:

```
$ git init                                  #git initialisieren - falls gewuenscht.
                                            #alternativ: mit git clone ein bestehendes
                                            #repository klonen. Wir helfen dabei.
$ cabal init                                #erstellen eines paketes
$ cabal sandbox init                        #initialisieren der sandbox
$ nano <projektname>.cabal                  #hinzufuegen von mtl > 2.2.0 && < 2.3
                                            #als dependency
                                            #einstellen der Main durch aendern von
                                            #     main-is: game.hs
$ cabal install --only-dependencies         #installieren aller dependencies
$ cabal build                               #projekt bauen
$ cabal run                                 #projekt ausfuehren
$ cabal repl                                #einen ghci laden, in dem alle dependencies
                                            #schon geladen wurden
```

[1] https://hackage.haskell.org/package/mtl

Aufgabe 5.3:

In dieser Aufgabe geht es um die Verwendung eines Monad-Stacks. Hierzu schreiben sie ein (sehr!) simples Spiel:
Durch drücken von `u` (up) bzw. `d` (down) wird ein interner Counter hoch- bzw. runtergezählt. Arbeiten sie sich in den gegebenen Code (`game.hs`) ein und erstellen sie das Game-Loop

```
mainLoop :: RWST Env () State IO ()
```

und die Tasteneingabe

```
getInput :: RWST Env () State IO Input
```

Benutzen sie hierzu die gegebene pure Hilfsfunktion

```
getInputfromEnv :: Char -> Env -> Input
```

Aufgabe 5.4:

Erweitern sie ihr Spiel durch einen weiteren Zähler, der durch die Tasten `r` und `l` für rechts und links erhöht bzw. ernidriegt wird.