Fortgeschrittene funktionale Programmierung in Haskell

Übungszettel 3

Aufgabe 3.1:

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
```

Aufgabe 3.2:

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 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
```