Fortgeschrittene Funktionale Programmierung in Haskell

Übungszettel 5

Aufgabe 5.1:

In der Vorlesung wurde `MaybeT` (basierend auf der `Maybe`-Monade) explizit vorgestellt. Erstellen Sie analog einen Monadentransformer `EitherT`, basierend auf der `Either`-Monade.

Zur Erinnerung: `Either` ist definiert als:

```
data Either a b = Left a  -- "Fehler"
                | Right b -- "Erfolg"
```

Erstellen Sie hierzu folgende Instanzen:

- `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 diese als Ausgangsbasis.

Aufgabe 5.2:

Für die folgenden Aufgaben benötigen Sie die externe Bibliothek `mtl`[1], in der der `RWST`-Stack (Read-Write-State-Transformer, siehe Vorlesung) bereits implementiert ist.

Richten Sie sich hierzu mit dem Programm `cabal` in einem Verzeichnis eine lokale Arbeitsumgebung (genannt *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 gerne dabei.
$ cabal update                          # Paketliste aktualisieren
$ cabal init                            # Erstellen eines Paketes
$ cabal sandbox init                    # Initialisieren der Sandbox
$ nano <projektname>.cabal              # Hinzufügen von mtl > 2.2.0 && < 2.3
                                        # als Dependency
                                        # Einstellen der Main durch ändern von
                                        #      main-is: game.hs
$ cabal install --only-dependencies     # Installieren aller Dependencies
$ cabal build                           # Projekt bauen
$ cabal run                             # Projekt ausführen
$ cabal repl                            # Einen ghci laden, in dem alle Dependencies
                                        # bereits 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 die 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 sinnvoll durch einen weiteren Zähler, der durch die Tasten `r` und `l` für rechts und links erhöht bzw. ernidriegt wird.