Übungsblatt 5

Vorwort

Dieses mal wollen wir ausgehend von unserem Parser (selbstgeschrieben oder von der Tutoren zur Verfügung gestellt) eine Visualisierung schreiben.

Set-Up

Wir legen zunächst wieder ein `stack`-Projekt an und tragen dort die benötigten Dependencies ein:

- `attoparsec`, falls wir den Parser selbst schreiben wollen
- `text`, weil aus den Parsern nur "Text" und nicht "String" herausfällt
- `gloss` zur Visualisierung.

Sie können zunächst einmal den in der Vorlesung besprochenen Code ausführen und so nachvollziehen, ob alles richtig installiert ist:

```
import Graphics.Gloss
import Graphics.Gloss.Data.Color (makeColor, red)
main :: IO ()
main = display (InWindow "Hello World" (500,500) (100,100))
               (makeColor 0.9 0.9 0.9 1)
               (Pictures [ Color red (Circle 1000)
                         , Text "Hello World Text"
                         ])
```

Aufgabe 1

Parsen sie die mitgegebene Datei "data.csv" in einen geeigneten Datentypen, den sie definieren. Der CSV-Parser liefert meist nur `[[Text]]`, sie sollen aber so etwas wie `[DataRow]` daraus erstellen (für ein passend Definiertes `DataRow`). Geben sie dies mittels `print` auf der Konsole aus

Aufgabe 2

Wir möchten wissen, wie viele Zugriffe in welcher Stunde auf unserer Website es gab. Sortieren sie die Daten in 24 Buckets ein und stellen sie dies mittels `display` als einfaches Balkendiagramm dar.

Aufgabe 3

Wir möchten gerne eine Animation der Zugriffe über Zeit haben. Teilen sie hierzu den Datensatz in 5-Minuten-Buckets ein und erzeugen sie für jeden Bucket ein Bild aus 16x16 Kacheln, deren Farbe von Weiss nach Rot geht - je nachdem, wie viele Zugriffe in einer "Kachel" waren. Ein Zugriff auf eine Kachel ist definiert über die erste Zahl in der angegebenen IP (0-255). Die Kachel oben links entspricht der 0, oben rechts der 15, unten links der 240, unten rechts der 255. `ip "div" 16` gibt ihnen also die Zeile und `ip "mod" 16` die entsprechende Spalte.

Skalieren sie die Zeit so, dass 1h aus dem Log 10 Sekunden in der Animation entspricht.

Aufgabe 4

Denken sie sich weitere Visualisierungen des Datensatzes aus. Evtl. können sie z.b. mittels `play` den Datensatz interaktiv erkundbar machen.

Alternative aufgaben für gloss (ohne Parser)

Aufgabe 1

Zeichne mit GLOSS die folgenden Formen:

- Kreis
- Rechteck
- gefüllter Kreis (Farbe der Füllung: Rot)
- Linie

Die Hintergrundfarbe soll hierbei weiß sein, und das Bild eine Auflösung von 600x600px besitzen. Die Position des Fensters soll (300,100) betragen.

Aufgabe 2

```
module Main where

import Graphics.Gloss

data Ball = BallData { ball         :: Float -> Picture
                     , ballRadius   :: Float
                     , ballColor    :: Color
                     , ballposition :: (Float, Float)
```

```
                    , ballTempo    :: (Float, Float)
                    }
initialData :: Ball
initialData = BallData { ball         = circleSolid
                       , ballRadius   = 100
                       , ballColor    = red
                       , ballposition = (150, 150)
                       , ballTempo    = (40, 40)
                       }

window :: Display
window = InWindow "Main Window" (600, 600) (300, 100)

background :: Color
background = black

-- AUFGABE: Implementiere die Funktion render
render :: Ball -> Picture
render = undefined

-- AUFGABE: Implementiere die Funktion moveBall, sodass der Ball sich bewegt.
moveBall :: Ball -> Float -> Ball
moveBall = undefined

main :: IO ()
main = animate window background frame
  where
    frame :: Float -> Picture
    frame = render . moveBall initialData
```

Aufgabe 3

```
module Main where

import Graphics.Gloss
import Graphics.Gloss.Data.ViewPort

type Punkt  = (Float, Float)
type Radius = Float

(width, height) = (600, 600)


data Ball = BallData { ball         :: Float -> Picture
             , ballRadius   :: Float
```

```
              , ballColor    :: Color
              , ballposition :: (Float, Float)
                      , ballTempo    :: (Float, Float)
                      }
initialData :: Ball
initialData = BallData { ball         = circleSolid
                , ballRadius   = 100
                , ballColor    = red
                , ballposition = (20, 0)
                        , ballTempo    = (140, 140)
                        }

window :: Display
window = InWindow "Main Window" (600, 600) (300, 100)

background :: Color
background = black

fps :: Int
fps = 120

-- render Funktion aus der zweiten Aufgabe
render :: Ball -> Picture
render = undefined

-- moveBall Funktion aus der zweiten Aufgabe
moveBall :: Float -> Ball -> Ball
moveBall = undefined

-- -- AUFGABE: Implementiere die Funktion touchHori, sodass der Ball vom Oben und Unten zurü
touchHori :: Punkt -> Radius -> Bool
touchHori = undefined

-- AUFGABE: Implementiere die Funktion touchVert, sodass der Ball von Rechts und Links zurue
touchVert :: Punkt -> Radius -> Bool
touchVert = undefined

-- AUFGABE: Implementiere mithilfe der Funktionen touchHori und touchVert die Funktion check
checkTouchWall :: Ball -> Ball
checkTouchWall = undefined

update :: ViewPort -> Float -> Ball -> Ball
update _ s = checkTouchWall . moveBall s

main :: IO ()
main = simulate window background fps initialData render update
```