Project Sheet

FritzLight Project - Controlling 30x12 RGB Pixels Individually

STS Software Technology Systems

Functional Programming - Winter 2022/2023 - January 09, 2023 - Schupp/Lübke

Introduction

We offer the FritzLight project as an interactive way for you to get in contact with the input and output capabilities of Haskell, and to solidify your current Haskell knowledge in a more practical way. In this project, you will learn how to control the pixels of a screen individually in a functional way, based on user-given input. The FritzLight system consists of two parts: The emulator, which is fed with RGB[1] data for each pixel to display it on a screen, and the user-defined program which handles input as well as the actual computation of pixel data. Your task is to write such a custom program in groups, creating a small game or other visual effects with interactive components (including input to the system).

What do we expect for the course of this project?

Generally spoken, we expect you to be motivated to actually *contribute* to your group project. Be prepared to explain your concept during the mandatory appointment sessions, as well as the final program on presentation day. Come up with your own ideas, try out whatever comes to your mind, be creative! Use the 360 pixels to create something new, exciting, astonishing (or boring, it is all up to you!).

How do I start with my own project?

The back-end (e.g., events creation, timing behaviour,...) is already implemented in the provided project library. To help you getting started with your own FritzLight project, we provide a sample program which allows the user to move a single pixel over the screen (output) by pressing the arrows keys (input). Please see *sdl2.pdf* for detailed setup instructions.

When you have completed the setup, you can run the sample program shown in Listing 1. In the `main` function, four parts are of importance for your work: The IP address (which you will have to adapt according to the emulator device you want to communicate with[2], the delay between each sent frame (choose a value between 33000 (33ms) and 500000 (500ms)), the actual event function for your own processing of input and generation of output (repeatedly called after the desired delay), and the initial state of the system. The system state is not necessarily a two-element-tuple, but can be any data type that you want to use for data which is supposed to be persistent between events, such as the positions of dynamic pixels or letters (only make sure that the type corresponds to the one you are expecting in the contract of your event function).

[1] https://en.wikipedia.org/wiki/RGB_color_model

[2] online emulator: 134.28.70.51 | emulator in room E4.036 (via switch): 192.168.23.2 | FLight hardware in room E4.036 (via switch): 192.168.23.1

Listing 1: Excerpt from *SDLEventTest.hs*

```haskell
-- BEGIN MAIN FUNCTIONALITY

type MyState = (Int, Int)

--      wall dimens   input event  state      new state
move :: (Int, Int) -> KeyStatus -> MyState -> MyState
move (xdim, ydim) ("Pressed", "PO_Axis_1_DO", _) (x, y) = (x, (y - 1) `mod` ydim)
move (xdim, ydim) ("Held", "PO_Axis_1_DO", dur) (x, y) = if dur >= 100 then (x, (y - 1) `mod` ydim) else (x, y)
move (xdim, ydim) ("Pressed", "PO_Axis_0_DO", _) (x, y) = ((x - 1) `mod` xdim, y)
move (xdim, ydim) ("Held", "PO_Axis_0_DO", dur) (x, y) = if dur >= 100 then ((x - 1) `mod` xdim, y) else (x, y)
move (xdim, ydim) ("Pressed", "PO_Axis_1_D1", _) (x, y) = (x, (y + 1) `mod` ydim)
move (xdim, ydim) ("Held", "PO_Axis_1_D1", dur) (x, y) = if dur >= 100 then (x, (y + 1) `mod` ydim) else (x, y)
move (xdim, ydim) ("Pressed", "PO_Axis_0_D1", _) (x, y) = ((x + 1) `mod` xdim, y)
move (xdim, ydim) ("Held", "PO_Axis_0_D1", dur) (x, y) = if dur >= 100 then ((x + 1) `mod` xdim, y) else (x, y)
move (xdim, ydim) ("Pressed", "UP", _) (x, y) = (x, (y - 1) `mod` ydim)
move (xdim, ydim) ("Held", "UP", dur) (x, y) = if dur >= 100 then (x, (y - 1) `mod` ydim) else (x, y)
move (xdim, ydim) ("Pressed", "LEFT", _) (x, y) = ((x - 1) `mod` xdim, y)
move (xdim, ydim) ("Held", "LEFT", dur) (x, y) = if dur >= 100 then ((x - 1) `mod` xdim, y) else (x, y)
move (xdim, ydim) ("Pressed", "DOWN", _) (x, y) = (x, (y + 1) `mod` ydim)
move (xdim, ydim) ("Held", "DOWN", dur) (x, y) = if dur >= 100 then (x, (y + 1) `mod` ydim) else (x, y)
move (xdim, ydim) ("Pressed", "RIGHT", _) (x, y) = ((x + 1) `mod` xdim, y)
move (xdim, ydim) ("Held", "RIGHT", dur) (x, y) = if dur >= 100 then ((x + 1) `mod` xdim, y) else (x, y)
move _ _ (x, y) = (x, y)

--         wall dimens   state      generated frame
toFrame :: (Int, Int) -> MyState -> ListFrame
toFrame (xdim, ydim) (x', y') =
  ListFrame $
    map
      ( \y ->
          map
            (\x -> if x == x' && y == y' then Pixel 0xff 0xff 0xff else Pixel 0 0 0)
            [0 .. xdim - 1]
      )
      [0 .. ydim - 1]

--           input events      state      frame       new state
eventTest :: [Event String] -> MyState -> (ListFrame, MyState)
eventTest events state = (toFrame dim state', state')
  where
    state' =
      foldl
        ( \acc (Event mod ev) -> case mod of
            "SDL_KEY_DATA" ->
              foldl (flip (move dim)) acc (getKeyDataTuples (read ev :: KeyState))
            "SDL_JOYSTICK_DATA" ->
              foldl (flip (move dim)) acc (getButtonDataTuples (read ev :: ButtonState))
            _ ->
              acc
        )
        state
        events

-- END MAIN FUNCTIONALITY
-- BEGIN CONFIGURATION

ip :: String -- IP address of wall/emulator
ip = "134.28.70.51" -- "127.0.0.1"

delay :: Int -- delay between frames in microseconds
delay = 33000

initState :: MyState -- initial state
initState = (0, 0)

-- END CONFIGURATION

main :: IO ()
main = do
  window <- showSDLControlWindow
  Sock.withSocketsDo $
    runMate
      (Config (fromJust $ parseAddress ip) 1337 dim (Just delay) True [sdlKeyEventProvider,
↪  sdlJoystickEventProvider])
      eventTest
      initState
  destroySDLControlWindow window
```

The main functionality in this example is implemented via two functions: The `eventTest` function processes keyboard input (obtained from the list of input *events*), and moves the highlighted pixel accordingly. The `toFrame` function takes the information about the highlighted pixel and creates a complete assignment (`ListFrame`) of every single pixel to its desired RGB color data (in this case, the pixel to be highlighted is set to white (`0xff 0xff 0xff`), while the remaining ones are set to black (`0 0 0`).

If you need random data in any of your functions, replace the `runMate` function in `main` with `runMateRandom`, and add an additional first argument to your event function that will contain an infinite list of random `Int` values. The new contract would then be `eventTest :: [Int] -> [Event String] -> MyState -> (ListFrame, MyState)`.

Which possibilities do I have to output my pixel data graphically?

a) *Online* using the emulator at http://ledwall.sts.tuhh.de (only reachable via TUHH Network or VPN[3])

- The IP address of the online emulator is `134.28.70.51`
- Use a browser to view the available screens, click on your IP (to determine IP, see below)

b) In room *E4.036*: On the SmartBoard (Fig. 2b), or the FritzLight hardware. Connect (wired) to the *switch* (Fig. 1a, next to the FritzLight).

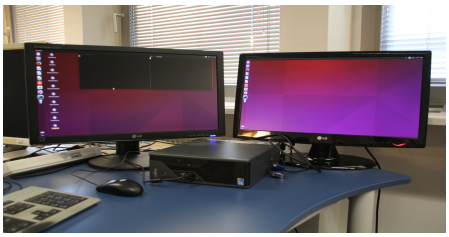
- Use the emulator on the SmartBoard, available at `192.168.23.2` (PC with mouse is in grey shelf, use guest account, open *firefox*)
- Control the FritzLight at `192.168.23.1`
- You may also execute sample programs using the FritzLight PC (Fig. 1b, use guest account)

To work in this room you need to make an appointment online[4]. If you're experiencing technical problems you can contact either Ole Lübke (office E4.059) or Thomas Sidow (office E4.050).

c) Install the emulator on your own system (advanced)

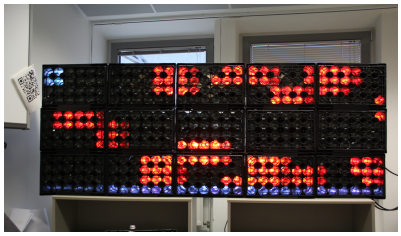
[3] https://www.tuhh.de/alt/rzt/netze/vpn.html
[4] https://nc20.sts.tuhh.de/index.php/apps/appointments/pub/IimnYz1YefGPFubBGKY%3D/form

(a) A Switch



(b) A Computer

Figure 1: The switch and the computer



(a) The FritzLight running an Obstacles game

(b) The emulator running a Pong game

Figure 2: Emulators in hard- and software

Useful links for the project

- How to determine your IP address (for Windows and macOS, there are similar ways for Linux, e.g., the `ip a` command)
  https://beebom.com/how-find-ip-address-windows-mac/

Tasks

Task 1 **Find a group and decide on a concept**

a) Find a group of four (4) members with whom you will work for the entire duration of this project.

b) Register your group on StudIP.

c) In your group, agree on a concept/an idea for your project, and make a plan for how you want to realize it.

d) Distribute the work equally between all group members.

If you can't seem to find a group or there aren't four people in your group yet, you have multiple options:

- Search for a group/additional members on the StudIP forums.
- Ask people during the lab sessions, or approach your tutors. We may be able to connect you with students from the other lab groups.
- Send an e-mail to ole.luebke@tuhh.de, maybe we can figure something out.

Deadline for group registration: Tuesday, January 17, 08:00

Task 2 The 1st intermediate appointment session is **mandatory** for all project participants. You need to book an appointment online[5], time slots are available on January 18 & 19 and need to be booked at least 24 hours in advance. The appointments take place in E4.059 (Zoom upon timely request).

[5] https://nc20.sts.tuhh.de/index.php/apps/appointments/pub/1QA9s2e6Jq85H3YYKKk%2BNw2/form

a) **Demonstrate your setup - 0.5 Points**
Compile the sample dot project, make sure that you can communicate with the online emulator, and show us that everything is working during the intermediate appointment session.

b) **Explain your concept - 1 Point**
For that session, you are supposed to be able to explain both the overall *project concept* of your group, and the individual parts that you have already contributed or will contribute to the final result. Additionally, you are supposed to write down your concepts in a `Readme`-file (e.g., what does your program do? How is it working? How is the user supposed to interact?, What are the key (user-defined) data types and functions?), which you will add to your final result submission. You can find suggestions on how to write a good `Readme` on the internet[6].

Deadline: Your booked appointment (on January 18 or 19)

Task 3 **Present your progress - 0.5 Points**
The 2nd intermediate appointment session on **Monday, January 23** (lecture hall exercise slot, i.e., H0.016, 8:00), is **mandatory** for all project participants. For that session, each group member is expected to have contributed (equally) to the current state of the project and explain their contributions.
Deadline: The 2nd intermediate appointment session on January 23

Task 4 **Present your final result - 1 Point**
The presentation session, most likely on **Wednesday, February 01**, but this is **subject to change**, is **mandatory** for all project participants as well. You will demonstrate your final project result, and explain the core parts of your implementations. At the latest by the given deadline (see below), your project needs to be uploaded as `FritzLight_[GroupNumber].zip` to the provided project folder on StudIP, containing the project files, the readme, and a list containing the group number as well as the names of all group members.

DEADLINE: 08:00, January 31, 2023
2023-01-31T08:00:00+01:00

[6] e.g., at https://www.makeareadme.com/