Comparative Programming Languages

Project Assignment Functional Reactive Programming (Elm)

For this assignment, you are required to implement an adaption of the Tron game in Elm.

An example implementation can be found on this page. Your solution should mimic the behaviour of this implementation. If something is not clear in this assignment, you can consider the example implementation as part of the instructions.

Expected working time

We expect you to spend max. 20 hours on this assignment. You are allowed to turn in an incomplete assignment. If you do, try to implement features that are explained early on in the assignment first. If you finish early, there are some optional extra assignments you can make for better grades.

Features

We require you to implement the following features, in order to help you we provide a skeleton file that includes some functions that may be of use to you. You are not obliged to use these functions as long as you adhere to the feature descriptions below.

- The playing field

  There is a playing field of 1024x768 pixels, with a black background color.

- Game play

  A game starts by showing an instructional message: "Press space to start a new game". Games can be (re)started at any time by pressing space. If only one player remains, the field disappears and the text "Player _ has won!" appears and remains visible until space is pressed again.

- Light Cycles

  The players control a light cycle that travels around in the playing field leaving a trail that cannot be crossed.

  - Drawing

    To draw the light cycle for a player with a certain color `color` at a certain position `(x,y)` and orientation `o`, you can use the code present

in the skeleton file. It will draw a rectangle with the proper offsets and rotation as demonstrated in the following schematic:

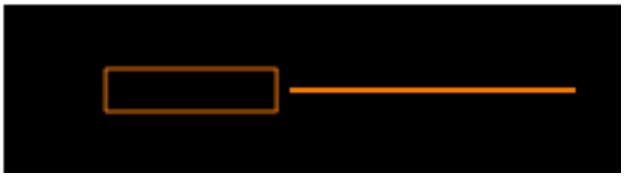

Figure 1: Light Cycle

```
import Color
playerW = 64
playerH = 16

data Orientation = N | E | S | W
showPlayer' : Color -> (Float, Float) -> Orientation -> Form
showPlayer' color (x, y) o =
    let fw = toFloat playerW
        (xOffset, yOffset, degs) = case o of
                    N -> (0, fw / 2, 90)
                    E -> (fw / 2, 0, 0)
                    S -> (0, -fw / 2, -90)
                    W -> (-fw / 2, 0, 180)
    in rect (fw) (toFloat playerH)
        |> outlined (solid color)
        |> move (x, y)
        |> move (xOffset, yOffset)
        |> rotate (degrees degs)
```

– Movement

  The light cycles start moving as soon as the game starts. Players start 50 pixels from their edge and move at a consistent speed towards the center. You're free to choose your own speed, as long as it's playable, our example moves at around 50 pixels per second.

– Steering

  Steering up, left, right and down are the only actions a player can perform. The default controls are the 'wasd' and arrow clusters. Turns are instantaneous and are not restricted, i.e., a turn happens as fast as a person can press the respective buttons. This implies that the playing field should be thought of as floats and not integers.

  A light cycle pivots around its back wheel:

– Collisions A player crashes when its light cycle hits the edges of the field or a tail. This means that hitting your own tail makes you lose

Figure 2: Moving upwards

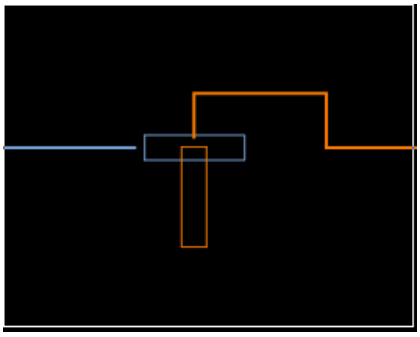

Figure 3: Orange wins

the game but hitting the opponent's *light cycle* is a valid move, thus going through a player and making your tail collide is an effective strategy:

- Tail

  When light cycles move they leave a trail behind that we call a 'tail'.

  - Drawing:

    Given a color and a list of segment points the function will return a form which includes the proper drawing. Tails have a width of 2.

```
tailWidth = 2

showTail : Color -> [(Float, Float)] -> Form
showTail color positions =
    traced { defaultLine |
        width <- tailWidth
        , color <- color
        } (path positions)
```

  - Location

    A tail starts appearing at the back of the light cycle so players can turn around a tail but doing a complete 180 into their own tail will result in suicide.
  - Length

    You can choose the tail's length but it should approximate the example and the game should be playable with your chosen length.

- Optional extra features

  If you have extra time, you can implement some extra features. These are not implemented in the example assignment.

  - Add a way for a player to increase his speed by driving near tails or walls. For example, when player 1 is within 5 pixels of a wall or tail his speed increases by 1% every tick while its color brightens.
  - Add support for 2, 3 or 4 player games.
  - Write a configuration menu that allows you to reconfigure the controls for each player.
  - Write a basic AI so that only 1 player is required to play the game. This AI would do the most basic and simple calculations:
    1. Look straight ahead and detect obstacles that are 50 pixels away or less
    2. Avoid detected obstacles (walls and tails) by turning towards the direction that has the most amount of space left For example, the AI notices a tail 50 pixels away or closer and looks at his own position (x, y), a calculation is made which direction (up, down, left, right) results in the largest area of the field (this does not take tails into account).
  - Write an advanced AI that takes tails into account and chooses the direction of least obstacles within a radius of 300 pixels.
  - Allow players to choose names (shown in the won/lost message after a game ends and possibly somewhere in the screen) and colors for their lightcycles and their trails.
  - Use Elm's Random Signal to create a variable amount of walls within the field when the game starts.

Practically

You must write your assignment in Elm 0.13 Elm is a young project and versions are released frequently. New versions may be released before the deadline of this assignment so before you panic about errors please check if you are not just using syntax from a different version, Elm releases are not yet backwards or forwards compatible!

Documentation

Here are some links with documentation material for Elm:

- Documentation for the Elm standard library.
- A document containing an overview of Elm syntax.

Turning in the assignment

You have to turn in your solution for this assignment as a single .elm file containing your full solution through Toledo. Please make sure that the file compiles and runs with Elm 0.13 before submitting.

The deadline is Sunday December, 21st at 23h59. If you have any questions, there is a discussion forum on Toledo that will be actively monitored by the assistants.