Disseny d'Aplicacions Telemàtiques

Pràctica 1B

Aniol Martí i Espelt

2 de maig de 2020

Índex

1 Introducció

En aquesta segona part de la pràctica 1 es pretén realitzar una simulació del Joc de la Vida, inventat per John Conway l'any 1970. Per tal de poder interactuar amb el joc crearem una aplicació CGI interactiva. El desenvolupament de la pràctica se separa en 4 parts: a cada part s'aniran afegint funcionalitats fins a arribar a la versió final.

2 Exercicis

En aquest apartat comentaré com s'han desenvolupat i com funcionen els diferents passos que han donat lloc a l'aplicació final.

Observació: A cada exercici només s'inclou el codi que he desenvolupat jo per aquell pas.

2.1 Exercici 1

2.1.1 Comentaris

Aquest primer exercici simplement consisteix en el taulell de joc. Per part de l'estudiant cal completar la funció `drawBoard :: Board -> Drawing` del fitxer `src/Life/Draw.hs`. Per fer-ho, haurem d'usar la funció `drawCell :: Board -> Pos -> Drawing` que hem de definir i implementar.

La funció `drawCell` dibuixa un quadrat negre de dimensions 1x1 a la posició indicada. A partir d'aquesta funció s'implementa `drawBoard`, que mitjançant la funció `foldMap` dibuixa una cèl·lula a cada posició de la llista `liveCells board`. El funcionament d'això ho hem vist a la part A, però en lloc de caselles dibuixàvem semàfors.

2.1.2 Codi font

```
drawBoard :: Board -> Drawing
drawBoard board = foldMap (drawCell board) (liveCells board)

drawCell :: Board -> Pos -> Drawing
drawCell board (x,y) = translated (fromIntegral x) (fromIntegral y) (colored black (
    solidRectangle 1 1))
```

2.2 Exercici 2

2.2.1 Comentaris

En aquest segon exercici es pretén afegir una graella que es pugui mostrar i ocultar prement la tecla **G** del teclat. La graella té 3 estats:

- Oculta.
- Només a l'entorn de les cèl·lules.
- Tot el taulell.

Primer de tot cal afegir el tractament de l'esdeveniment "prémer la tecla G". Per fer-ho, afegim un nou cas a la funció `handleEvent :: Event -> Game -> Game` que ja se'ns dona definida. Aquí cal destacar que el paràmetre de tipus `Game` està definit amb el que es coneix com a *Field labels*. Això fa que el compilador defineixi una funció `gmGridMode :: Game -> GridMode` que ens dona el tipus de graella del joc que li passem. A partir

d'aquí, fent servir aquesta funció i una expressió `case` es fa el canvi de graella, passant del tipus de graella actual al següent.

A part del tractament de l'event també cal modificar la funció que dibuixa la pantalla. Ara no només cal dibuixar el taulell, també cal dibuixar la graella. En aquest cas, per dibuixar-ho tot, es fa una composició del dibuix del taulell i de la graella. Per saber quin tipus de graella cal dibuixar es defineix i s'implementa una funció local `gridDraw :: GridMode -> Draw` que donat un tipus de graella fa un dibuix blanc (és a dir, sense graella), al voltant de les cèl·lules (a partir de la posició de la primera cèl·lula i de l'última) o a tot el taulell (a partir de l'altura i l'amplada d'aquest).

2.2.2 Codi font

```
-- The compiler defines a function gmGridMode :: Game -> GridMode
handleEvent (KeyDown "G") game =                -- Change grid
    let nextGrid = case (gmGridMode game) of NoGrid    -> LivesGrid
                                             LivesGrid -> ViewGrid
                                             ViewGrid  -> NoGrid
    in setGmGridMode nextGrid game


-- Drawing
draw game =
    drawBoard (gmBoard game) <> gridDraw (gmGridMode game)
    where gridDraw NoGrid = blank
          gridDraw LivesGrid = drawGrid (minLiveCell (gmBoard game)) (maxLiveCell (gmBoard
              game))
          gridDraw ViewGrid = drawGrid (round (-viewWidth/2), round (-viewHeight/2)) (round (
              viewWidth/2), round (viewHeight/2))
```

2.3 Exercici 3

2.3.1 Comentaris

En aquest tercer exercici es demana afegir les funcions per poder fer zoom i desplaçar-se pel taulell. S'hauran d'afegir 6 nous tractaments: per les tecles **O**/**I** (*zoom out*/*zoom in*) i per les 4 fletxes. També s'ha de tenir en compte que en desplaçar-se o fer zoom caldrà convertir la posició del ratolí per la del taulell. Això ho farem amb la funció `pointToPos :: Point -> Game -> Pos`, que ens dona l'enunciat. Com podem veure, es modifica el tractament del ratolí i en lloc d'obtenir la posició directament aquesta es passa per la funció `pointToPos`.

Primer de tot, el tractament del zoom. Tant el *zoom in* com el *zoom out* tenen un tractament similar. Primer de tot es verifica que no passis el límit de zoom, és a dir, que per ampliar no sigui 2 vegades més gran i que per desampliar no sigui 16 vegades més petit. En cas que això no sigui així, es multiplica o es divideix per 2 el zoom, respectivament.

Pel que fa al tractament del desplaçament, els 4 casos també són pràcticament idèntics, només cal tenir en compte el sentit de desplaçament i restar-lo (o sumar-lo) al *shift* actual i tenint en compte el zoom. Per fer-ho s'utilitzen les funcions del mòdul `Drawing.Vector`.

Finalment, cal tenir en compte que per dibuixar la pantalla dependrà de la posició i del zoom. Per facilitar aquesta tasca es defineix una funció `zoomAndMove :: Double -> (Double, Double) -> Drawing -> Drawing` que donat un zoom, una posició i un dibuix, mou i escala aquest últim en funció dels dos primers. Per fer el dibuix final es torna a fer una composició de dos dibuixos: el del taulell i el de la graella, però aquest cop passats per la funció `zoomAndMove`. També cal tenir en compte que quan et desplaces o fas zoom les posicions extremes del taulell canvien, així que per dibuixar la graella sencera cal passar l'amplada i l'alçada per la funció `pointToPos` tal i com hem fet amb el ratolí.

2.3.2 Codi font

```
-- Event processing
handleEvent (MouseDown (x, y)) game =           -- Set live/dead cells
    let pos = pointToPos (x, y) game
        brd = gmBoard game
    in setGmBoard (setCell (not $ cellIsLive pos brd) pos brd) game

handleEvent (KeyDown "I") game =                -- Zoom in
    if gmZoom game < 2.0 then setGmZoom (gmZoom game * 2.0) game
    else game

handleEvent (KeyDown "O") game =                -- Zoom out
    if gmZoom game > 1/16 then setGmZoom (gmZoom game / 2.0) game
    else game

handleEvent (KeyDown "ARROWUP") game =           -- Down shift
    setGmShift (gmShift game ^-^  (1.0 / gmZoom game) *^  (0, 5)) game

handleEvent (KeyDown "ARROWDOWN") game =         -- Up shift
    setGmShift (gmShift game ^+^  (1.0 / gmZoom game) *^  (0, 5)) game

handleEvent (KeyDown "ARROWRIGHT") game =        -- Left shift
    setGmShift (gmShift game ^-^  (1.0 / gmZoom game) *^  (0, 5)) game

handleEvent (KeyDown "ARROWLEFT") game =         -- Right shift
    setGmShift (gmShift game ^+^  (1.0 / gmZoom game) *^  (0, 5)) game


-- Drawing
zoomAndMove :: Double -> (Double, Double) -> Drawing -> Drawing
zoomAndMove zoom (x,y) draw = scaled zoom zoom (translated x y draw)

draw game =
    zoomAndMove (gmZoom game) (gmShift game) (drawBoard (gmBoard game)) <> zoomAndMove (gmZoom
         game) (gmShift game) (gridDraw (gmGridMode game))
    where gridDraw NoGrid = blank
          gridDraw LivesGrid = drawGrid (minLiveCell (gmBoard game)) (maxLiveCell (gmBoard
              game))
          gridDraw ViewGrid = drawGrid (pointToPos ((-viewWidth/2), (-viewHeight/2)) game) (
              pointToPos (viewWidth/2, viewHeight/2) game)
```