Functional Programming: Minesweeper

Laurent Christophe, Wolfgang De Meuter

Programming Project: Assignment #1 (2014 - 2015)

1 Introduction

The final mark for the Functional Programming course is based 50% on a programming project (25% on the first programming assignment, 25% on a second programming assignment) and 50% on the oral exam. This document describes the first programming assignment. The second assignment will be handed out later in the semester.

Submission is done by sending your raw code files to the teaching assistant: Laurent Christophe (`lachrist@vub.ac.be`). The assignment is due 24 November 2014 (8AM). You will have to defend both parts of your project individually on a date that will be communicated later. Notice that the execution of the project is strictly individual and no plagiarism shall be tolerated!

The project will be marked according to how well you fulfil the functional requirements and according to how well you apply the concepts explained during the lectures and the lab sessions. If you encounter any problem or if you have a precise question, feel free to contact the assistant at `lachrist@vub.ac.be`.

2 Requirements

The goal of this programming assignment is to create a text based Minesweeper[1] game. Our version of Minesweeper is played on a two dimensional board of variable size. While playing Minesweeper, the player has to *flag* the cells of which he/she believes that they contains a mine. The player can also *click* on a cell which reveals the number of adjacent mines in response (two cells are considered adjacent if and only if they share one side or one corner). However if the player clicked on a mined cell he/she loses the game.
Your code should implement the following rules:

1. A cell can either be flagged, clicked or masked (i.e. left intact).
2. It should be possible to specify the size (two dimensions) of the board.
3. The mines should be randomly placed on the board based on a seed provided by the user.
4. The number of mines should depend on the size of the board.
5. The first click cannot lead to losing the game since that would be unfair.
6. When displaying the board, flagged cells should be displayed differently from masked cells.
7. When displaying the board, clicked cells should display the number of adjacent mines.
8. When displaying a won / lost board, the position of the mines should be revealed.
9. If the user has flagged all the mines and clicked on all the clear cells, then the game is won.
10. If the user has clicked on a mined cell then the game is lost.
11. If the user has clicked on a cell without adjacent mines (i.e. the number of adjacent mines is equal to zero) then the click is propagated to all the adjacent cells (eight cells in total).

[1] `http://en.wikipedia.org/wiki/Minesweeper_(video_game)`

Here is a fragment of a possible run:

```
> Place a flag???
Just(3,1)
> Place a flag???
Nothing
> Click...
(2,1)
>
+-+-+-+-+
|F|1|0|0|
+-+-+-+-+
|1|1|0|1|
+-+-+-+-+
| | |2|F|
+-+-+-+-+
| |2|F|2|
+-+-+-+-+
> Place a flag???
Just(1,1
```

3 Facilities

For this programming assignment, your are allowed to use a Haskell file[2] that contains code for helping you to interact with the user as well as templates that will force you towards a well-designed solution. A.o., the file exposes the `Board` type class which you need to implement:

```
-- The show instance must be highly customized to display a board in ASCII
class Show b => Board b where
  -- Create a board from seed, dimension and first click (avoid immediate loosing)
  initialize :: Int -> (Int, Int) -> (Int, Int) -> b
  -- Click a cell on the board (no effect if out-of-bounds)
  click :: (Int,Int) -> b -> b
  -- Flag a cell on the board (no effect if out-of-bounds)
  flag :: (Int,Int) -> b -> b
  -- Test if all the mines have been flagged and all the clean cells clicked
  won :: b -> Bool
  -- Test if a mined cell has been clicked
  lost :: b -> Bool
```

The file also exports a higher-order function called `top` that needs to be called with your concrete `initialize` implementation in order to form the `main` function of your program. For instance:

```
main :: IO ()
main = top (initialize :: Int -> (Int,Int) -> (Int,Int) -> MyBoard)
```

[2]Available on the Poincarre platform in the "Assignment" subdirectory