G52AFP, year 2015-2016
Coursework 1: Noughts and Crosses

Abstract

The aim of this coursework is to write a Haskell program to play noughts and crosses, which uses game trees and the minimax algorithm to ensure that it never looses, i.e. always achieves a win or a draw.

Instructions

- This coursework counts for 10% of the assessment for the module, and may either be solved on your own, or jointly with **ONE** other student taking the module. Larger teams are not permitted.
- Solutions must be in the form of a literate Haskell script (.lhs document), and include a brief explanation for each definition. Bonus marks are available for particularly clear, simple, or elegant solutions.
- For identification purposes, your script must begin as follows:

  ```
  G52AFP Coursework 1 - Noughts and Crosses

  Your full name(s)
  Your full email address(es)
  ```

  In the case of jointly produced solutions, only one copy should be submitted, containing the names of both students.
- The deadline for submission is 3pm on **Tuesday 8 March**. Submission is electronic, via Moodle:

  `http://moodle.nottingham.ac.uk/mod/assign/view.php?id=1876162`

The board

The 3×3 board is represented as a list of list of player values:

```
type Board = [[Player]]
```

In turn, a player value is either a nought, a blank, or a cross, with a blank representing a space on the board that is not yet occupied:

```
data Player = Nought | Blank | Cross
              deriving (Ord, Eq, Show)
```

For example, here is a typical board:

```
[[Blank, Nought, Blank], [Cross, Nought, Cross], [Cross, Blank, Blank]]
```

The user interface

The following code displays a board on the screen:

```
showBoard          :: Board → IO ()
showBoard          = putStrLn ∘ unlines ∘ concat ∘ interleave bar ∘ map showRow
                     where
                       bar = [replicate 18 '-']

showRow            :: [Player] → [String]
showRow            = beside ∘ interleave bar ∘ map showPlayer
                     where
                       beside = foldr1 (zipWith (++))
                       bar = replicate 5 "|"

showPlayer         :: Player → [String]
showPlayer Nought  = ["     "," +-+ "," | | "," +-+ ","     "]
showPlayer Blank   = ["     ","     ","     ","     ","     "]
showPlayer Cross   = ["     "," \\ / ","  X  "," / \\ ","     "]

interleave         :: a → [a] → [a]
interleave x []    = []
interleave x [y]   = [y]
interleave x (y : ys) = y : x : interleave x ys
```

For the purposes of this coursework, how *showBoard* works is not important; just use this function in your own program. For example, applying *showBoard* to the board from the previous section gives the following output:

```
     |     |
     | +-+ |
     | | | |
     | +-+ |
     |     |
-----------------
     |     |
 \ / | +-+ | \ /
  X  | | | |  X
 / \ | +-+ | / \
     |     |
-----------------
     |     |
 \ / |     |
  X  |     |
 / \ |     |
     |     |
```

Exercise

Write a Haskell program

$$main :: IO\ ()$$

that allows a human player to play noughts and crosses against the computer, using game trees and the minimax algorithm (which will be explained in the lectures) to ensure that the computer never looses.

Hint: construct your program from the bottom-up, starting by defining a number of utility functions on boards, and only then proceeding to think about game trees, minimax, and the user-interface.