G52AFP Coursework 1
Connect 4

Graham Hutton
University of Nottingham

Abstract

The aim of this coursework is to write a Haskell program to play the familiar game of connect four, which uses game trees and the minimax algorithm to implement the computer player.

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. Students must not make their code publically available online.

- Solutions must be in the form of a literate Haskell script (.lhs document), and include an explanatory comment for each definition. Bonus marks are available for particularly clear, simple, or novel solutions.

- For identification purposes, your script must begin as follows:

```
G52AFP Coursework 1 - Connect 4

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 **WEDNESDAY 7th MARCH**. Submission is electronic, via the University moodle system:

```
http://tinyurl.com/G52AFP-2018
```

The board

For flexibility, we define constants for the row and column size of the board, length of a winning sequence, and search depth for the game tree:

```
rows :: Int
rows = 6

cols :: Int
cols = 7

win :: Int
win = 4

depth :: Int
depth = 6
```

The board itself is represented as a list of rows, where each row is a list of player values, subject to the above row and column sizes:

```
type Board = [Row]

type Row = [Player]
```

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

```
data Player = O | B | X
              deriving (Ord, Eq, Show)
```

For example, here is a typical board:

```
test :: Board
test = [[B,B,B,B,B,B,B],
        [B,B,B,B,B,B,B],
        [B,B,B,B,B,B,B],
        [B,B,B,X,X,B,B],
        [B,B,O,O,X,B,B],
        [B,O,O,X,X,X,O]]
```

The user interface

The following code displays a board on the screen:

```
showBoard :: Board -> IO ()
showBoard b =
   putStrLn (unlines (map showRow b ++ [line] ++ [nums]))
   where
      showRow  = map showPlayer
      line     = replicate cols '-'
      nums     = take cols ['0'..]

showPlayer :: Player -> Char
showPlayer O = 'O'
showPlayer B = '.'
showPlayer X = 'X'
```

For example, `showBoard test` gives the following output:

```
.......
.......
.......
...XX..
..OOX..
.OOXXXO
-------
0123456
```

Exercise

Write a Haskell program

```
main :: IO ()
```

that allows a human player to play connect four against the computer, using game trees and the minimax algorithm (as explained in the lectures) to implement the computer player. Your solution should aim to be independent of the precise values of the constants `row`, `cols`, `win` and `depth`.

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