CPSC 201: Introduction to Computer Science
Carsten Schürmann
Date: April 12, 2002

Homework 7

Due: midnight
Wednesday April 24, 2002.

In this assignment we implement an interactive and an automated version of the Connect Four game. The objective of this assignment is to let you design and implement a game search procedure based on techniques we discussed in class. We will have another tournament and you can get extra points if your system is among the very best players. I, of course, will be competing, too.

We have prepared a few files for you to get started on this assignment. They can be found in the course directory `/c/cs201/lib/code/ass7/*`. Copy all files in your working directory. From the SML toplevel (once SML is started in your working directory), simply type `CM.make ();` to load all files.

Connect Four is a game that two players play against each other. Each player picks one of two color Red and Yellow respectively. The player then take turns and insert coins into a vertical game board. The coins drop down to the bottom of the board which has seven slots. Each slot can maximally hold seven coins. The winner of the game is the player who first manages to line up four coins of his or her color in a straight line, vertically, horizontally, or diagonally.

Important: Do not edit the signature files. They must remain unchanged otherwise we cannot run the competition and let your implementations play against each other. If you do change them you are automatically disqualified from the competition.

Problem 1: Game setup (20 points)

Implement the structure `Game` in file `game.fun`. Take note of how we represent the configuration of the game. The board is represented by four different entities (r, y, t, l): r is the maximal number of red coins in a line and y the maximal number of yellow coins. t is a list of seven integers, each marks the top of an individual column, and l is a list of lines, including direction, color, length, and the left and right end points. The initial board can hence be represented by the following configuration.

```
val initialBoard  : board = ((0,0), [0,0,0,0,0,0,0], [])
```

Implement the following four functions:

```
makeMove  : board -> move -> board
validMove : board -> move -> bool
boardToString : board -> string
```

`makeMove` is used to insert a coin into the board, `validMove` decides if a move was valid, and `boardToString` converts a board into a string that can be printed on demand.

Problem 2: Interactive player (20 points)

Implement the function

```
play   : int -> Game.Color -> Game.board -> Game.move option
```

in functor `Interactive` in file `player.fun`. This function will print the current board, and prompt the user for input, a number between `1` and `7`. The user can also type in `quit`, and the game is to be aborted (which means), the opponent won. Make sure to ask the player as long as he or she mistypes or makes an invalid move.

Problem 3: Automatic player (40 points)

Complete the implementation of

```
play   : int -> Game.Color -> Game.board -> Game.move option
```

in functor `Automatic` in file `player.fun`. This function will look at the game board, apply analysis techniques to your liking and return a move (`SOME m`) where `m` is a move, or resigns (`NONE`). This is the meat of the assignment. The better you perform here, the better chances you have to win the tournament.

Implement the MINIMAX technique discussed in class with depth limited pruning and your choice of evaluation function for non-terminal symbols. Feel free to refine this technique as you see fit.

Problem 4: Connect Four game (20 points)

Complete the implementation of

```
playerRed : int -> string
playerYellow : int -> string
```

in functor `Connect4` in file `connect4.fun`. In essence you are implementing a referee, that passes moves from one player to the other. If `playerRed` is called this means that red makes the first move, otherwise with `playerYellow` yellow begins. We need this functionality for the competition.

Problem 5: Tournament (up to 40 extra credit points)

Your program will participate in the CS201 tournament. In addition to the ordinary credit for writing a valid implementation of the signature below, we offer a first prize of a tasteful Yale mug plus 40 extra credit points. You obtain 20 ec points for reaching the semi-final, 10 ec points for reaching the quarterfinals, and 5 ec points for reaching the round of the last sixteen. To this end, edit the file `connect4.sml` and replace the string `YourFirstName` by your first name. Hand-in all files in the directory, including `sources.cm`.

Your program loses a match if it raises an uncaught exception, attempts an illegal move, if the cumulative CPU time spent by your program for a match exceeds 300 seconds, or (the usual condition) it cannot make a legal move.

The tournament is played in single elimination format, with two matches (with either program making the initial move) between randomly paired programs in each round. In case of a tie the program which used less CPU time overall advances to the next round. (See structure `Timer`.) In each match, since there can be at most 49 moves, there can be no ties.

Hand-in Instructions

In the course directory /c/cs201/bin, there are five programs that support you in submitting your solution to the homework.

```
submit    assignment-number file(s)
unsubmit  assignment-number file(s)
check     assignment-number
protect   assignment-number file(s)
unprotect assignment-number file(s)
```

`submit` allows you to submit one of several files. For example

```
/c/cs201/bin/submit 7 game.sig game.fun ...
```

submits your file `game.sig` and `game.fun`. `check` can give you the peace of mind that your solution has really been submitted, and if you would like to make last minute changes to your solution, use `unsubmit` before submitting the updated version. `protect` and `unprotect` give you the power to protect/unprotect submitted solutions from deletion.