University of Sheffield, Department of Computer Science

COM2001: Advanced Programming Techniques
Semester 1: Functional Programming
Assessment 2017-18
Assignment 3: Domino Players

This exercise counts for 25% of the assessment for the COM2001 module

1. Introduction

In this assignment you will implement 5s-and-3s dominoes players and test them against each other by playing simulated matches.

In section 5 are some hints about good play. **You should experiment with your dominoes players to see if implementing each hint does indeed improve the player's game.** You are not expected to programme all the hints and you may try other tactics.

2. 5s-and-3s Dominoes Matches

A 2-player 5s-and-3s dominoes match is organised as follows:

- A match consists of N games, typically 3. **The winner is the player who has won most games.** With our tireless computer players N can be much greater.
- Each game involves a sequence of rounds (as defined in assignment 2). The two players take it in turns to drop first in each round.
- A game terminates when one player or the other wins by achieving an accumulated score, over the rounds, of **exactly 61**.
- If at the end of a round neither player has reached 61, a new round starts and the scores obtained within this round are added on to the scores after the last round.
- **As with all good pub games, you must finish exactly**, i.e. if your score is 59 you need to play a domino which scores 2 to win the game. If you score more ('going bust') your score remains the same (i.e. if your score is 59 and you play a domino which scores 4 your score remains at 59).

3. Match-playing Software

A Haskell module **DomsMatch** is provided for you. Its top-level function looks like so:

domsMatch :: DomsPlayer->DomsPlayer->Int->Int->(Int,Int)

Here the 3rd argument is the number of games to be played and the 4th argument is an initial seed for the random number generator. **domsMatch** returns the number of games won by the first player (P1) and the second (P2).

To use **domsMatch** your data structures must be compatible. This is what **domsMatch** uses:

data DomBoard = InitBoard|Board Dom Dom History
deriving (Show)

- **InitBoard** is the initial state of a round, with nothing dropped
- In **Board**, the 2 **Doms** are those at the left and right ends, with pips lined up, e.g if the board is [(1,5),(5,5),(5,6)] the left end is (1,5) and the right is (5,6).
- The **History** allows the course of the round to be reconstructed (skilled dominoes players work things out from this):

  type History = [(Dom,Player,MoveNum)]

  History is a left-to-right list of triples representing the dominoes on the board, who played each domino and the play number, starting at 1.

data Player = P1|P2 -- player 1 or player 2
deriving (Eq,Show)
data End = L|R -- left end or right end
deriving (Eq,Show)
type Scores = (Int,Int) – P1's score, P2's score
type MoveNum = Int
type Hand = [Dom]
type Dom = (Int,Int) – a domino

Type DomsPlayer = Hand->DomBoard->Player->Scores->(Dom,End)
i.e. a **DomsPlayer** is a function which takes

- The player's **Hand**
- The DomBoard
- The player's identity (P1 or P2)
- The current **Scores**

And returns the **Dom** to play and the **End** to play it

4. Example of a Match

2 **DomsPlayer**s are provided for you (in **DomsMatch**)

- randomPlayer chooses a random play form its hand
- hsdPlayer always plays the highest scoring domino

So

```
domsMatch hsdPlayer randomPlayer 100 42 ~> (93,7)
```

plays a match of 100 games between these players. hsdPlayer wins 93 of these games.

5. Skilled Play

Here are some hints about good play:

- Some players normally play the highest scoring dom unless it risks the opponent scoring more (e.g. if the ends are 6 and 0 and your hsd is (6,6), play it unless the opponent could play (0,3) or (0,6).

- Beware of playing a dangerous Dom unless you can knock it off, e.g. playing (6,6) if there are no more 6s in your hand.

- The History tells you what doms remain and therefore what to guard against.

- You can use the History to work out what doms your opponent may have (e.g. if the ends are 5 & 5 & the opponent doesn't play (5,5) s/he doesn't have it.

- Knowledge about what your opponent is knocking on is particularly useful.

- If you have the majority of 1 spot value (e.g. 4 6s) it's a good idea to play that value, especially if you have the double.

- If you have a weak hand, try to 'stitch' the game, i.e. make both players knock.

- If you are getting close in the end game, you should obviously see if you have a winner. If not, try to get to 59, because there are more ways of scoring 2 than scoring 1,3 or 4.

- If the opponent is getting close in the end game, look for a dom which will prevent her/him winning on the next play, or reduce the chances of this happening.

- If you have 'first drop' onto an empty board, a popular choice is (5,4), because it scores 3 but the maximum you can score in reply is 2.

6. Design for the DomsPlayers

There will be a briefing for this assignment in class. In the briefing the problem of how to design a dominoes player which can use these hints will be discussed. However, you may implement your DomsPlayers in any way you like provided you explain it.

7. What to hand in

Hand in 3 documents, in a single zip archive:

1. Your **design**, preferably as a diagram (an example will be given in the lectures) and explanatory notes
2. Your **implementation**: commented code as a .hs file, ready to run
3. Your experimental **results**: you should try to demonstrate that changes you make to the domino players code give them an advantage.

8. Marking Scheme

20% of the available credit is for the design, 50% for the implementation and 30% for the results.

6. How to hand in

Hand in by MOLE

DEADLINE: Midnight, Friday 15th December (Week 12)