COMP5450 Assessment 3

Introduction

This assessment is about implementing the game of Tic Tac Toe (also known as: Noughts and Crosses).

In case you do not know it: the game is played on a 3x3 grid that is initially empty. Two players are playing, by alternatingly making moves. A move by a player places their token (an X for player 1, an O for player 2) into a cell that was empty.

We are using algebraic notations for indexing the positions in the board, with A,B,C indexing the columns and 1,2,3 the rows. Specifically, these coordinates would be used in the implementation for moves made by a human player. If the

		X
X		
	O	

Figure 1: Sample board position

same token appears 3 times in any of the three columns, three rows or two main diagonals the game is over and that player wins. If the grid is filled without that happening the game is a draw.

For the depicted board, we have Xs in positions C3 and A2, and an O in position B1. It would be O's turn to make a move; a legal move would be C2, but it is not a good move, because X can force a win by responding A1. O cannot force a win, but B2 keeps the game balanced.

1 The Board and its Logic

1. Define appropriate types for a `Board`, a `Move`, and an `Outcome`. You can use algebraic types or type synonyms — this is your choice.

2. Define a function `printBoard :: Board -> IO()` that prints the board in a human-readable format.

3. Define a function of type `Board -> Move -> Maybe Board` that checks whether the move is legal in the board. If it is legal, the function should return the updated `Board`, otherwise `Nothing`.

4. Define a function of type `Board -> Maybe Outcome`, which returns the information about the outcome of the game: do we have a win for player X, a win for player O, a draw, or should the game continue? A return value of `Nothing` indicates that the game should continue.

2 Communications, Players and Processes

There are two kinds of communication messages going on in this game: the invitation to make a move (which will communicate a `Board` to a player), and the actual `Move` itself, which the player communicates to the game manager.

There are three processes in every game: the game manager, and the two players. A player is either a bot, or a human player. A player has three parameters: their game token (O or X), an input channel to read move invitations and an output channel to write chosen moves.

5. Define the human player. When a move is requested from the human player, the board should be on display, and a move should be read from the keyboard. If the move is illegal an appropriate error message should be printed, and we try again. If the move is legal it should be send back on the player's output channel.

6. Define a bot player. This player should make some legal move by just investigating the board. There should be an element of randomness (use the `PickRandom` module on moodle), unless there is only one "good" move. The bot does not have to be clever, but it could be.

7. Define the game managing process behaviour. This needs to keep track of the board position, the players' input channels and whose turn it is. It should keep inviting the players in turn to make their moves, and announce a winner (or a draw) when the situation occurs.

8. Define a game start. This is an IO action parameterized by two players. This action should create the necessary channels, fork the player processes, and then continue managing the game. The purpose of the two parameters is to allow different match-ups (humans or bots vs. humans or bots).

Marks: 5 marks each for questions 1 and 2. 10 marks each for questions 3 and 4. 15 marks each for questions 5 and 6. 20 marks each for questions 7 and 8.