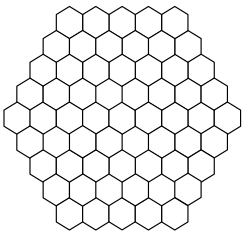
Functional Programming project

Due 24 April 2020

Your task in this coursework is to implement the game of Rokkakkei in Idris. This document introduces the game, and describes each of the tasks you are expected to complete.

Introducing Rokkakkei

Rokkakkei is played on a hexagonal board as follows:



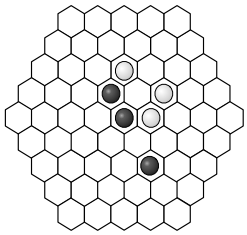
There are two players, player W(hite) and player B(lack). Player B moves first and puts a black stone on one of the positions. Next player W plays by putting a white stone on one of the empty positions, and the players alternate moves until the game finishes.

The goal is to connect four but *not* to connect three. The game finishes when one of three things happens:

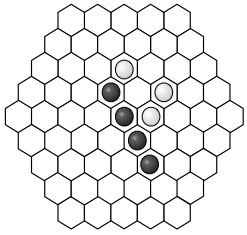
- a player connects three - in that case that player loses;
- a player connect four - in that case that player wins;
- the board is full and there are no connections of length three or four - in that case the game is a draw.

If a player connects four and three at the same time, winning beats losing: the player wins.

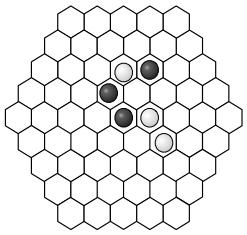
For example, consider the following game situation:



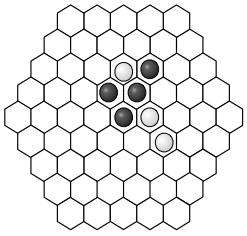
It is B's turn and B can win by connecting four:



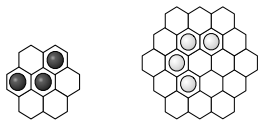
Consider also the following game situation:



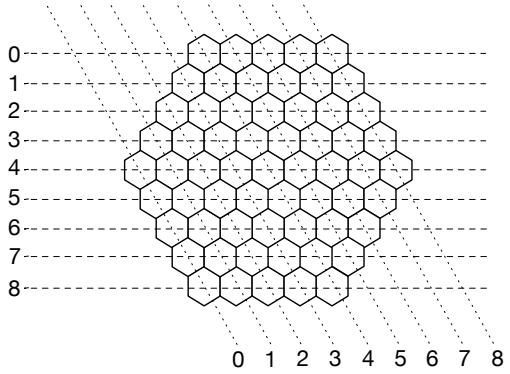
It is B's turn and she can try to stop W from connecting four.



In doing so, B had to connect three. But connecting three means automatic loss of the game: W is the winner! Note that only straight lines count as legal connections, the following do *not* count as legal connections of size three and four.



We will refer to individual board hexes by their coordinates:



So, for example, the first hex in the top row has coordinates $(0, 4)$. Notice that not all possible pairs of numbers give board positions, some are not *valid*. For example, the coordinates $(1, 1)$ do not correspond to any space on the board.

Task 1

Players are represented by datatype

```
data Player : Type where
  White : Player
  Black : Player
```

and a space on the board is either empty or it contains a Player stone

```
Space : Type
Space = Maybe Player
```

You are to define the datatype

```
Board : Type
```

that will hold the current state of the board, including all the moves played at any particular time.

You are to write three functions:

```
valid : Position -> Bool
get : Board -> Position -> Space
set : Board -> Position -> Space -> Board
```

The first of these, `valid` takes a `Position`, defined

```
Position : Type
Position = (Nat, Nat)
```

and return `True` exactly when the coordinates refer to a valid board position. Next, `get` takes a `Board`, a `Position` and returns the contents (`Space`) at that position. Finally `set` takes a `Board`, a `Position` and a `Space` and returns a modified board, with the new contents at the indicated position.

Task 2

Implement

```
winning_move : Position -> Player -> Board -> Bool
losing_move : Position -> Player -> Board -> Bool
```

that check whether the proposed move immediately wins the game (by connecting four) or immediately loses the game (by connecting three).

Task 3

Implement an interactive version of the game in the shell. The game should begin by printing an empty board. You should use the pretty printing `draw_board` routine provided to you in the file `draw_board.idr`.

Next it should print.

```
Player B?:
```

and expect an input of two non-negative integers, the coordinates of the next B move. You can expect the two integers to be separated by a single space.

Next, it should draw the board and print

```
Player W?:
```

take the coordinates for the next W move, and so on. If at any point a winning or losing move is made, you should print either

```
Player B wins!
```

or

```
Player W wins!
```

and exit. If the game reaches a draw, output

```
Draw!
```

and exit.

You are expected to provide appropriate error messages if the user tries to enter invalid coordinates, or make an illegal move – for example trying to move to an already occupied position. You are free to decide what to do after printing your error message, but for usability you should probably prompt the user again.

Successful completion of Tasks 1,2 and 3 earns a mark of 4. To get a mark of 5, you must pick *one* of the challenging problems below (Task 4a, Task 4b or Task 4c) and solve it satisfactorily. If you solve one your problems in a particularly elegant/impressive way, or solve more than one of the problems, you will be eligible for extra credit from this coursework.

Task 4a

Modify the game so that it can be played on a board of any size. The size can be given as a parameter to the game binary.

Task 4b

Modify the game so that moves are entered interactively with an animated cursor, that can be moved on the board using the arrow keys.

Task 4c

Implement an AI for the game so that a user can play against the computer.