Assignment 1, COMP3400, Semester 1 2020

Pick15

We're going to implement an API and the logic for a game. This game is known as Pick15 or Number Scrabble.

Game rules The rules are pretty simple:

- There are two players
- Each player takes a turn choosing a number from 1 to 9
- Once a number has been selected, it can never be selected again
- The winner is the first player to have exactly three of their numbers adding to 15
- If (and only if) all nine numbers have been selected, and neither player has three numbers adding to 15, the game is a draw
- Some of the consequences of these rules are
  - The game has a maximum of 9 moves
  - Neither player can win before move 3
  - A player may have selected 4 or 5 numbers, but requires **exactly 3** (not 2, 4 or 5) of those add to 15 to win

Examples of game play

```
P1: 9
P2: 1

P1: 4
P2: 8

P1: 2
```

At this point, P1 is the winner since 9+4+2 sums to 15.

```
P1: 4
P2: 3

P1: 1
P2: 9

P1: 2
P2: 6

P1: 5
P2: 8

P1: 7
```

At this point, the game is a draw since since neither player has exactly 3 numbers adding to 15, and there are no more numbers to select.

Goals

1. Write an Application Programming Interface (API) using the Haskell programming language to play the game

- Aside from the data structures and functions given below, you will need to write your own data structures and functions to achieve the higher-level API.
- Use this API for implementing the terminal program.

2. Provide an ability for two players to play the game interactively using a terminal. The following functions will help:

- `getChar :: IO Char`
- `putStr :: String -> IO ()`
- `putStrLn :: String -> IO ()`
- Experiment with these functions and look up their documentation to understand what they do.
- The following are examples of input/output sessions.

  Winning:

```
ghci> playPick15
press 'q' to Quit
[1 2 3 4 5 6 7 8 9]
Player 1 to move
>>> 7
[1 2 3 4 5 6   8 9]
Player 2 to move
>>> x
Invalid input
[1 2 3 4 5 6   8 9]
Player 2 to move
>>> 3
[1 2   4 5 6   8 9]
Player 1 to move
>>> 8
[1 2   4 5 6     9]
Player 2 to move
>>> 7
Already selected
[1 2   4 5 6     9]
Player 2 to move
```

```
>>> 4
[1 2     5 6     9]
Player 1 to move
>>> 5
[1 2       6     9]
Player 2 to move
>>> 9
[1 2       6      ]
Player 1 to move
>>> 2
Player 1 wins! 8 + 5 + 2 = 15
ghci>
```

Quitting:

```
ghci> playPick15
press 'q' to Quit
[1 2 3 4 5 6 7 8 9]
Player 1 to move
>>> 9
[1 2 3 4 5 6 7 8  ]
Player 2 to move
>>> q
Bye!
ghci>
```

Draw:

```
ghci> playPick15
press 'q' to Quit
[1 2 3 4 5 6 7 8 9]
Player 1 to move
>>> 5
[1 2 3 4   6 7 8 9]
Player 2 to move
>>> 8
[1 2 3 4   6 7   9]
Player 1 to move
>>> 9
[1 2 3 4   6 7    ]
Player 2 to move
>>> 1
[  2 3 4   6 7    ]
Player 1 to move
>>> 6
[  2 3 4     7    ]
Player 2 to move
>>> 4
```

```
[  2 3       7    ]
Player 1 to move
>>> 3
[  2         7    ]
Player 2 to move
>>> 7
[  2              ]
Player 1 to move
>>> 2
The game is a draw
ghci>
```

Data Types and Functions

Data Types Implement any appropriate type-class instances for the following data types, using either the `instance` keyword or the `deriving` keyword.

```
-- a data type that denotes the numbers that each player may select from
data Number_1to9 =
  N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9

data List a =
  Nil | Cons a (List a)

-- starts off empty and records each move as a player selects a number
data GameState =
  GameState (List Number_1to9)

data Solution =
  Solution
    Number_1to9
    Number_1to9
    Number_1to9

-- the possible results of a valid number selection
data ValidSelection a =
  Player1Wins Solution
  | Player2Wins Solution
  | Draw
  | KeepPlaying a

-- the possible results of any number selection
data NumberSelected =
  InvalidSelection
  | AlreadySelected Number_1to9
  | Selected (ValidSelection GameState)
```

Functions These functions form the API to play the game

```
-- start a new game
newGame :: GameState
-- select a number from the game state
selectNumber :: Number_1to9 -> GameState -> NumberSelected
-- returns whether or not the game state is a draw
isDraw :: GameState -> Bool
```

This function is for playing the game interactively using the terminal

```
playPick15 :: IO ()
```

Considerations

- Consider that some other (non-terminal) interface may be implemented in the future, which will use your API.
- If you make a design decision in your data types or functions, it is helpful to provide a comment explaining that decision. Redundant or superficial comments are discouraged.

  For example, this comment is redundant, since the type already specifies what the comment says:

  ```
  -- Function that returns a Bool
  isDraw :: GameState -> Bool
  ```

  This comment is helpful:

  ```
  -- Determines if the game is finished by any state.
  -- Useful function to implement isDraw.
  isGameOver :: GameState -> Bool
  ```

- It is acceptable to change `GameState` to use Haskell's built-in list data type (called `[]`) instead of our own list data type. Whichever decision you choose here, provide a short comment as to why. This comment could be as simple as, "I find my own list data type easier to work with and the reason is . . . "

Assessment

- This assignment contributes to 10% of the total marks for COMP3400.
- The marks for this assignment out of 20 will be distributed as follows:
  - `newGame` function **2 marks**
  - `selectNumber` function **8 marks**
  - `isDraw` function **3 marks**
  - `playPick15` function **3 marks**
  - Overall design, including helpful program comments **4 marks**

In considering overall design

- Reduce the API usage area by which an API user can put the game into an invalid state
  - For example, selecting a number after the game has finished, puts the game into an invalid state
  - This also includes return values. For example, `isGameOver` returns a `Bool` and not say, an `Int`, since this function always returns one of two values.
- Writing any additional functions for the API, which you might consider helpful to playing the game, is encouraged
- Helpful documentation
  - Useful comments for the API user
  - Example usages that help demonstrate how the API is to be used

Submission

Submit your `.hs` file(s) to the assessment folder on Blackboard. If you have a bunch of files you can zip them up and submit them.