# **Table of Contents**

[**Table of Contents** 2](#_Toc111487181)

[**Main idea of the project** 3](#_Toc111487182)

[**Show Board** 3](#_Toc111487183)

[**Make Move** 3](#_Toc111487184)

[**Is Legal Move** 3](#_Toc111487185)

[**Winner** 3](#_Toc111487186)

[**Code Snippets** 4](#_Toc111487187)

[**Board.hs** 4](#_Toc111487188)

[**Game.hs** 8](#_Toc111487189)

[**Screenshots** 9](#_Toc111487190)

[**First snip of the running program** 9](#_Toc111487191)

[**Wrong move** 10](#_Toc111487192)

[**Wins left diagonal** 10](#_Toc111487193)

[**Wins right diagonal** 11](#_Toc111487194)

[**Wrong move on a full column** 11](#_Toc111487195)

[**Win horizontal row** 12](#_Toc111487196)

[**Win vertical column** 12](#_Toc111487197)

# **Main idea of the project**

* In Connect-Four, a two-player game resembling tic-tac-toe, players alternately arrange pieces on a vertical board that is 6 rows high by 7 columns wide.
* Being the first to line up four pieces in a row, whether it be horizontal, vertical, or diagonal, is the goal.
* Each player utilises pieces of a specific hue. Pieces entered in a particular column always fall to that column's lowest open row since the board is vertical. A column is considered full and cannot accept any more pieces once it has 6 pieces.
* The first person to complete a line of four connected pieces wins the game. Each player starts the game with 21 identical pieces. The game is drawn if all 42 men have been used and no player has placed four pieces in a row.

**There are four functions that are expected to design and test:**

* showBoard: Create a String representation of the board.
* isLegalMove: True if the attempted move is legal. Used to prevent players from moving in full columns or playing in column -1.
* winner: True if the game is over (either player has four in a row).
* makeMove: Gives a new Board with a piece added. Assumes that the move is legal.

## **Show Board**

This should draw with no issues, showing the board each row on its own line. I do not anticipate problems with this function.

## **Make Move**

* 5 points: piece falls to the bottom of the row when empty (basic move)
* 5 points: piece falls to the appropriate place (follows all rules)

## **Is Legal Move**

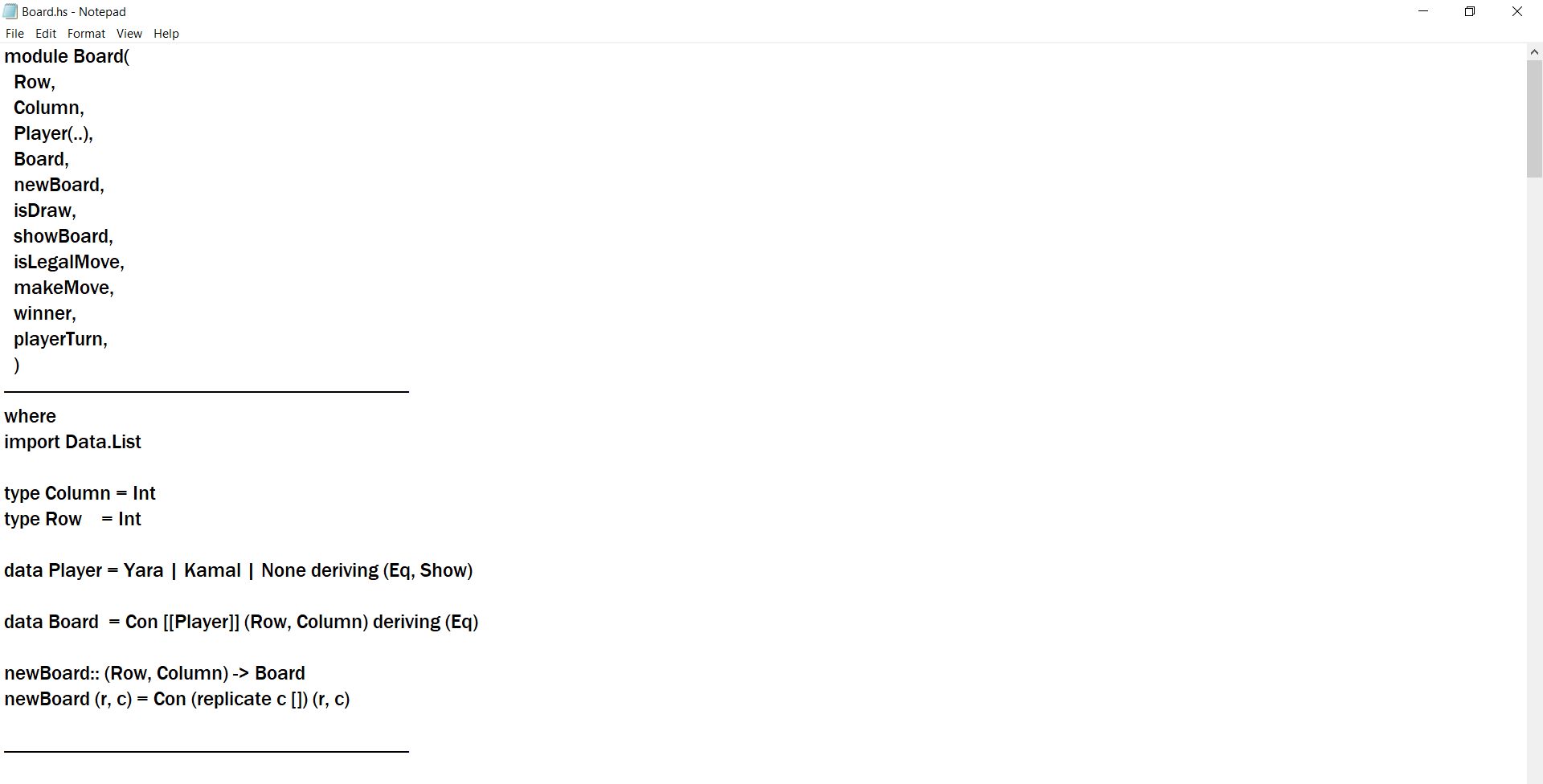
* 5 points: moves outside of board are not allowed (column negative or too large)
* 5 points: moves not permitted in full column

## **Winner**

* 5 points: horizontal and vertical wins detected
* 5 points: diagonal wins detected

# **Code Snippets**

## **Board.hs**



Haskell modules are a practical approach to organise several functions that could share the same names and bundle a set of related functionality into a single package. The first element in your Haskell file is the module definition. There is only one module per file, and its name starts with a capital letter.

newBoard :: (Row, Column) -> Board: returns an empty board with the specified number of rows and columns

isDraw :: Board -> Bool: checks if board is full

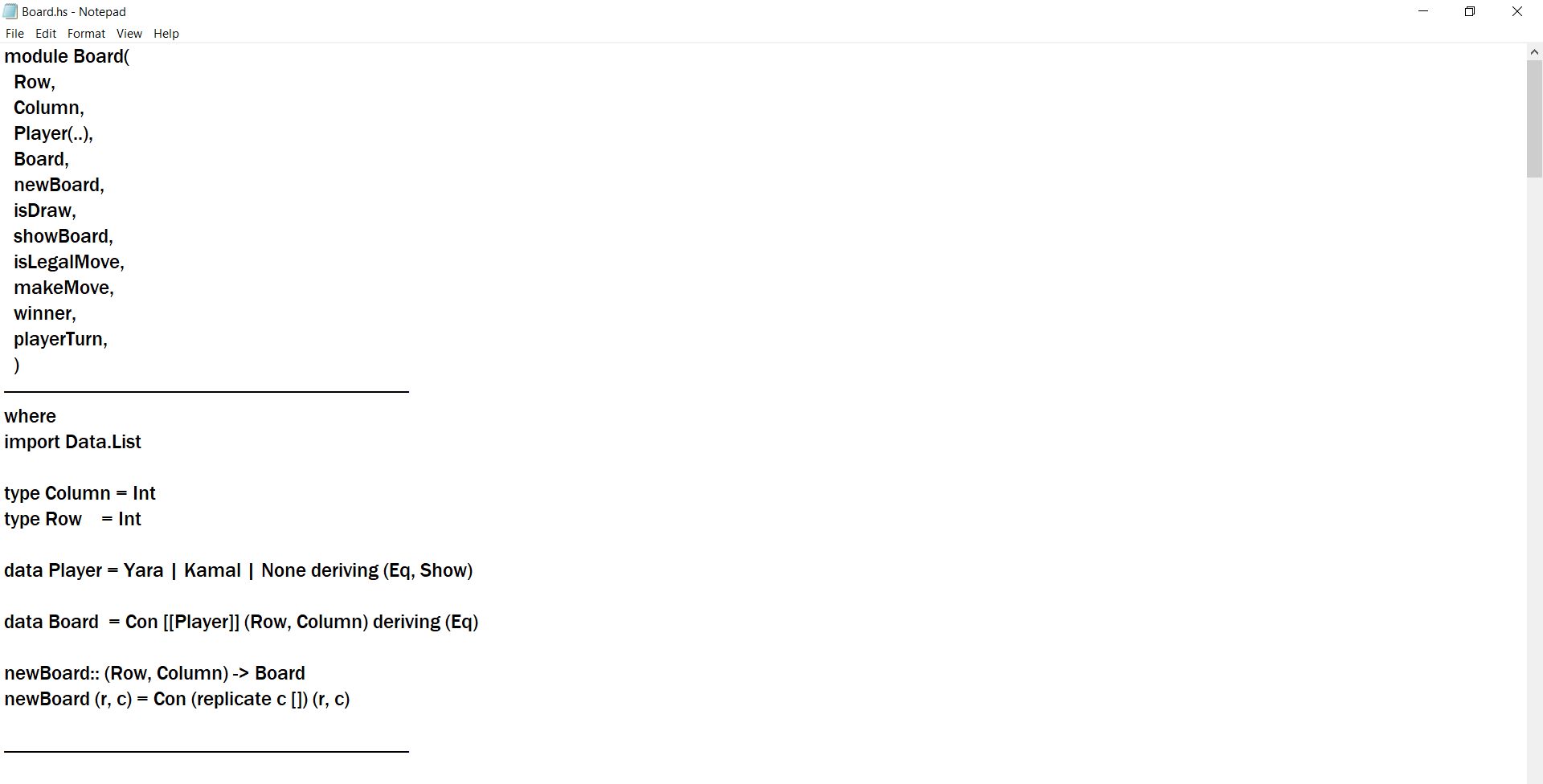
showBoard :: Board -> String: converts Board to String to show to user

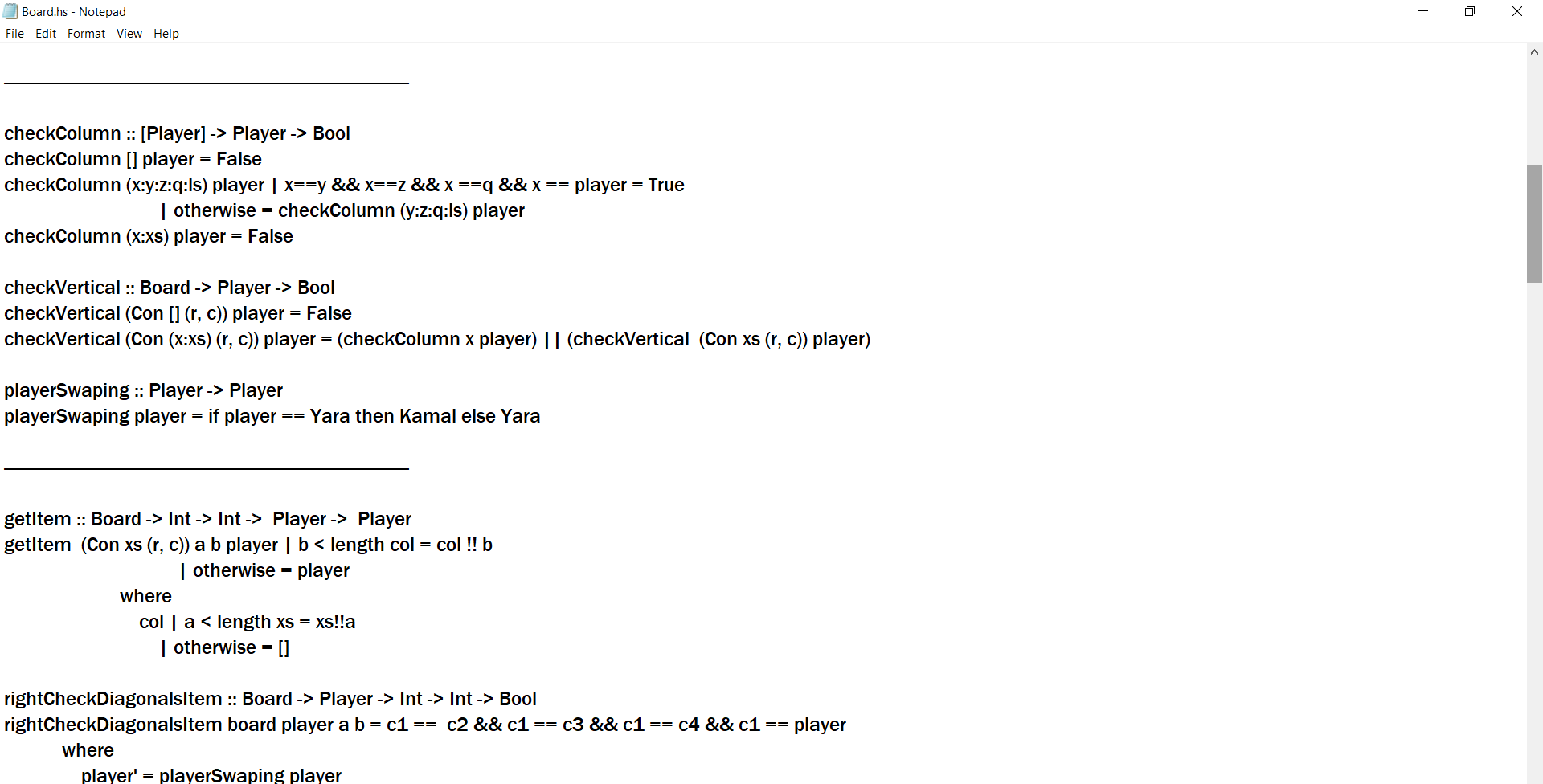
isLegalMove :: Board -> Column -> Bool: checks if the move is legal (i.e., column not full yet, col <= no. of cols of board)

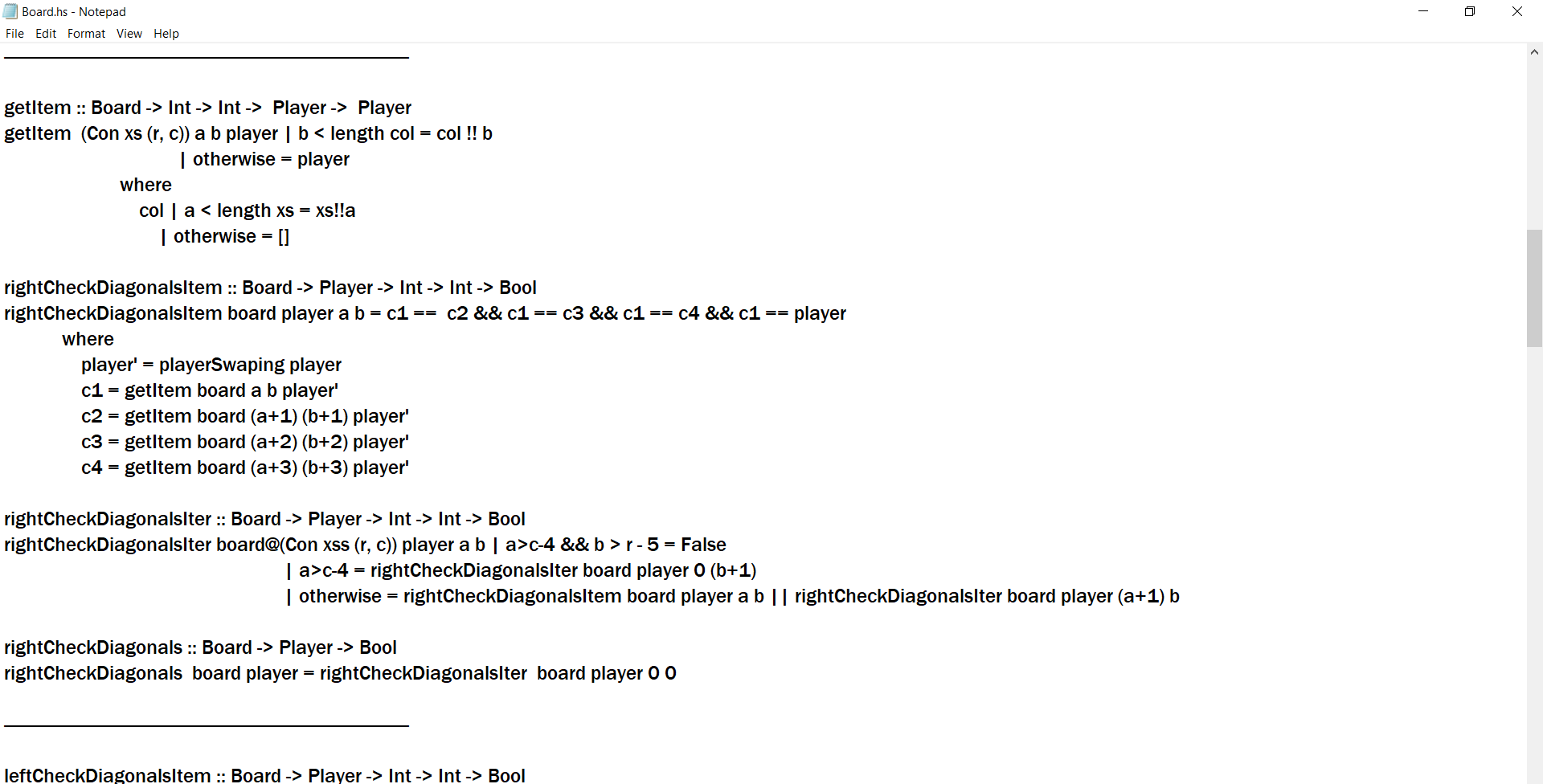
makeMove :: Board -> Column -> Player -> Board: make one move, raise runtime error if move is illegal

winner :: Board -> Int: (helper function) find the winner and display as Int

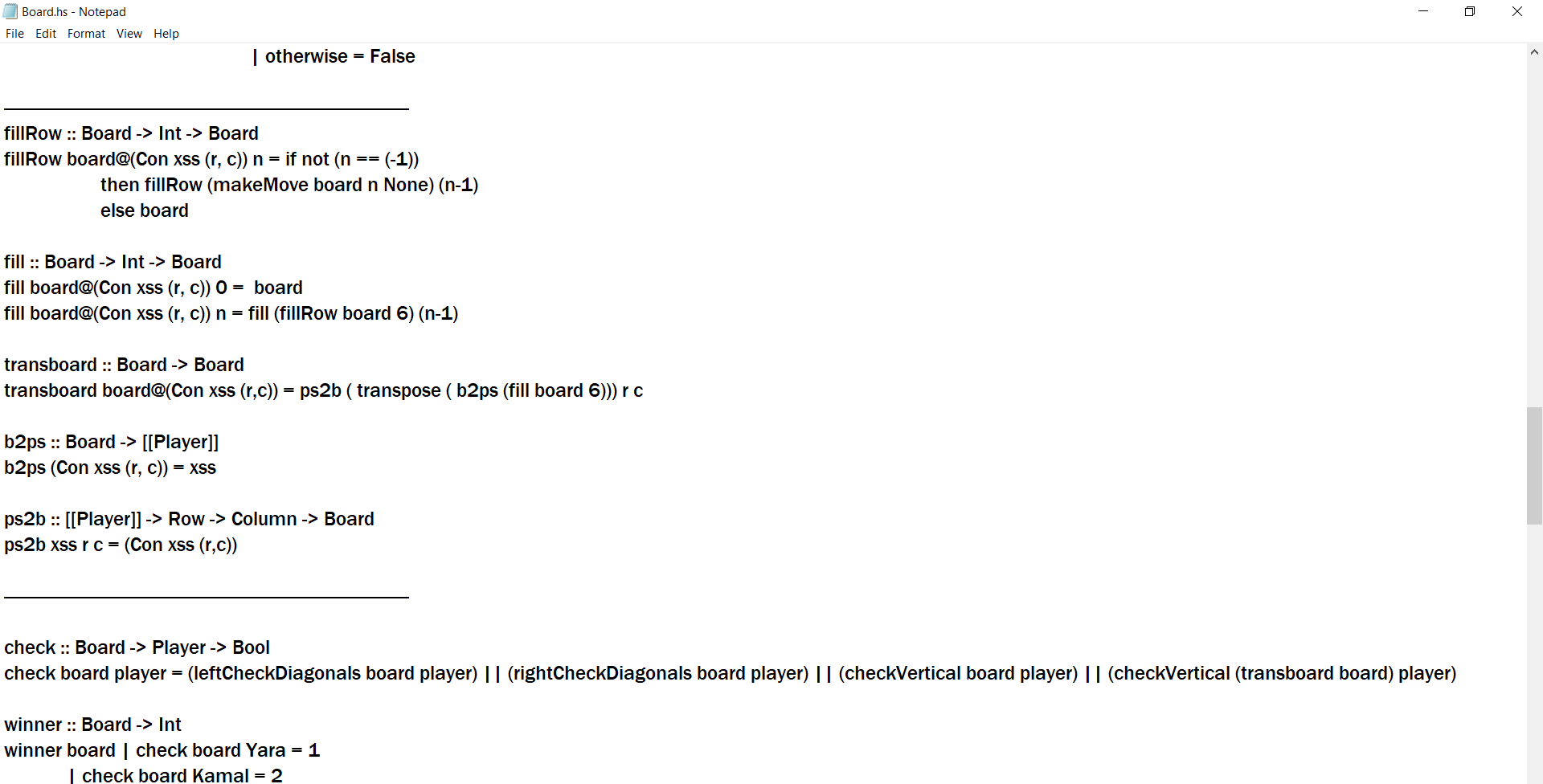
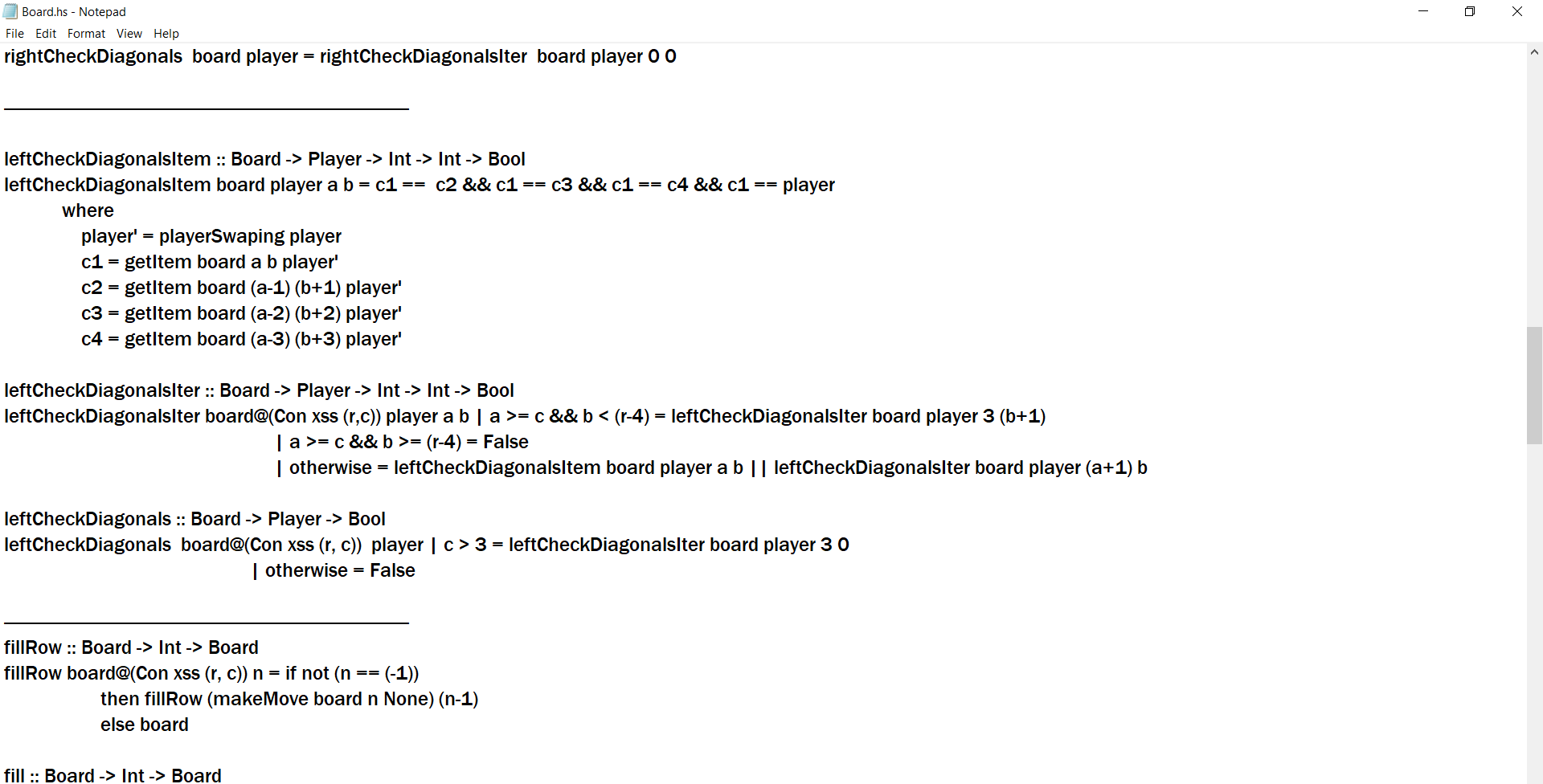
playerTurn :: Board -> Player: (helper function) find out whose turn it is

Here, defining the types of data that’s going to be used. Definition of abstract data types (define Board as an ADT). Creates new board.

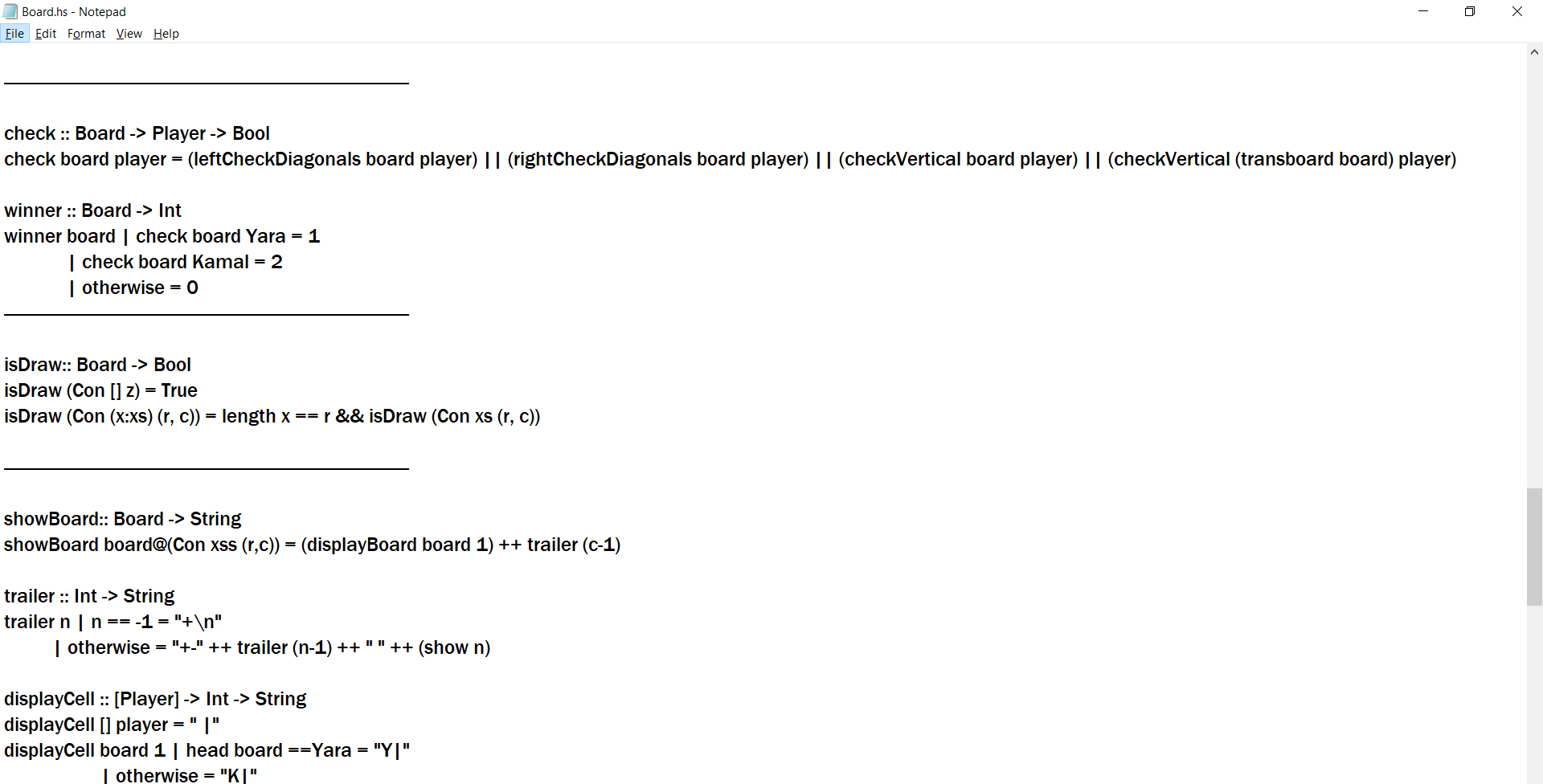
Check the board for winner in the columns. Checks the column. Check vertical column win or not. Generates player swapping, to put in empty spaces.



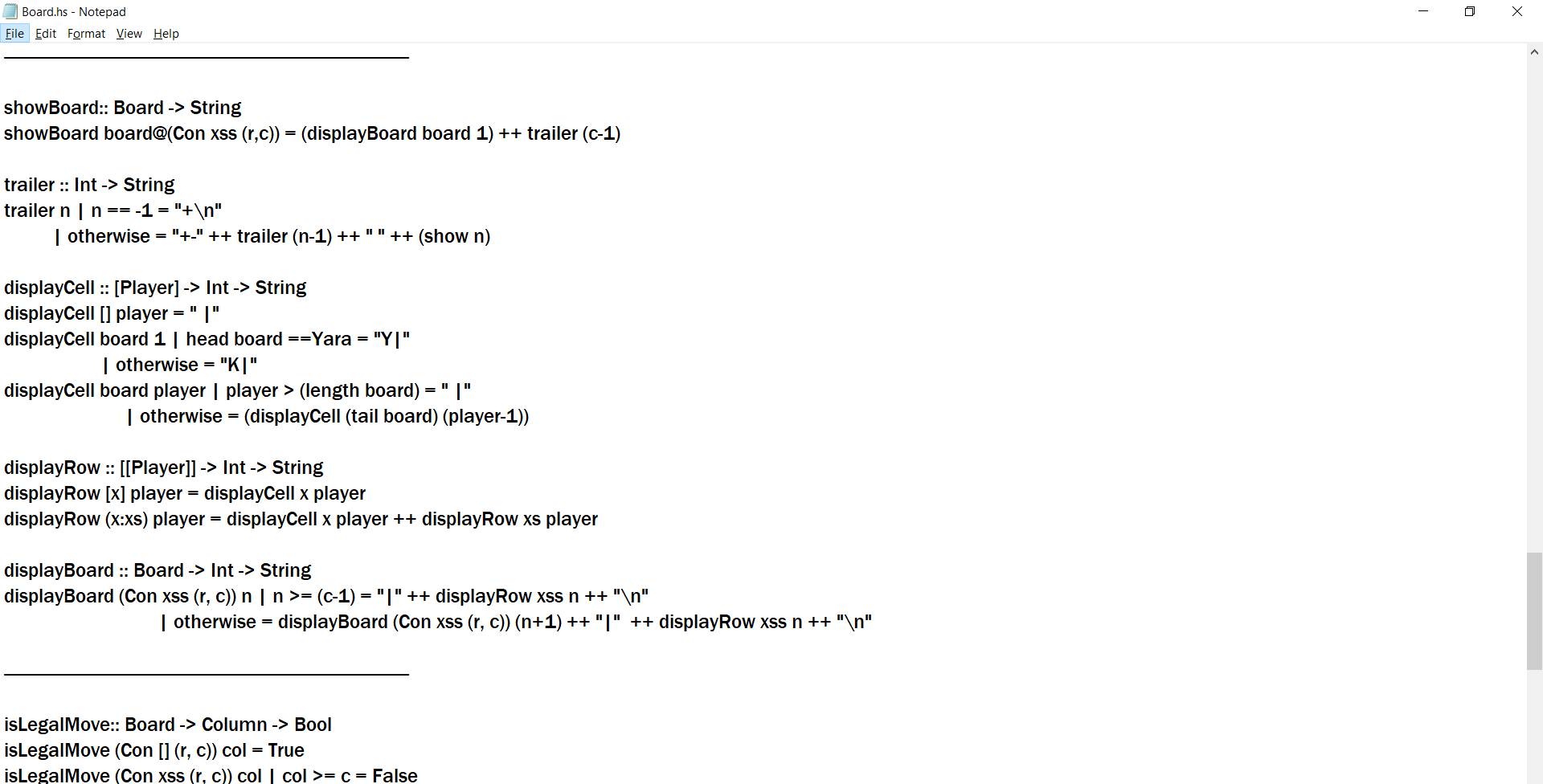
Gets item at position a b. Checks right diagonals for winner.

Checks left diagonals for winner.

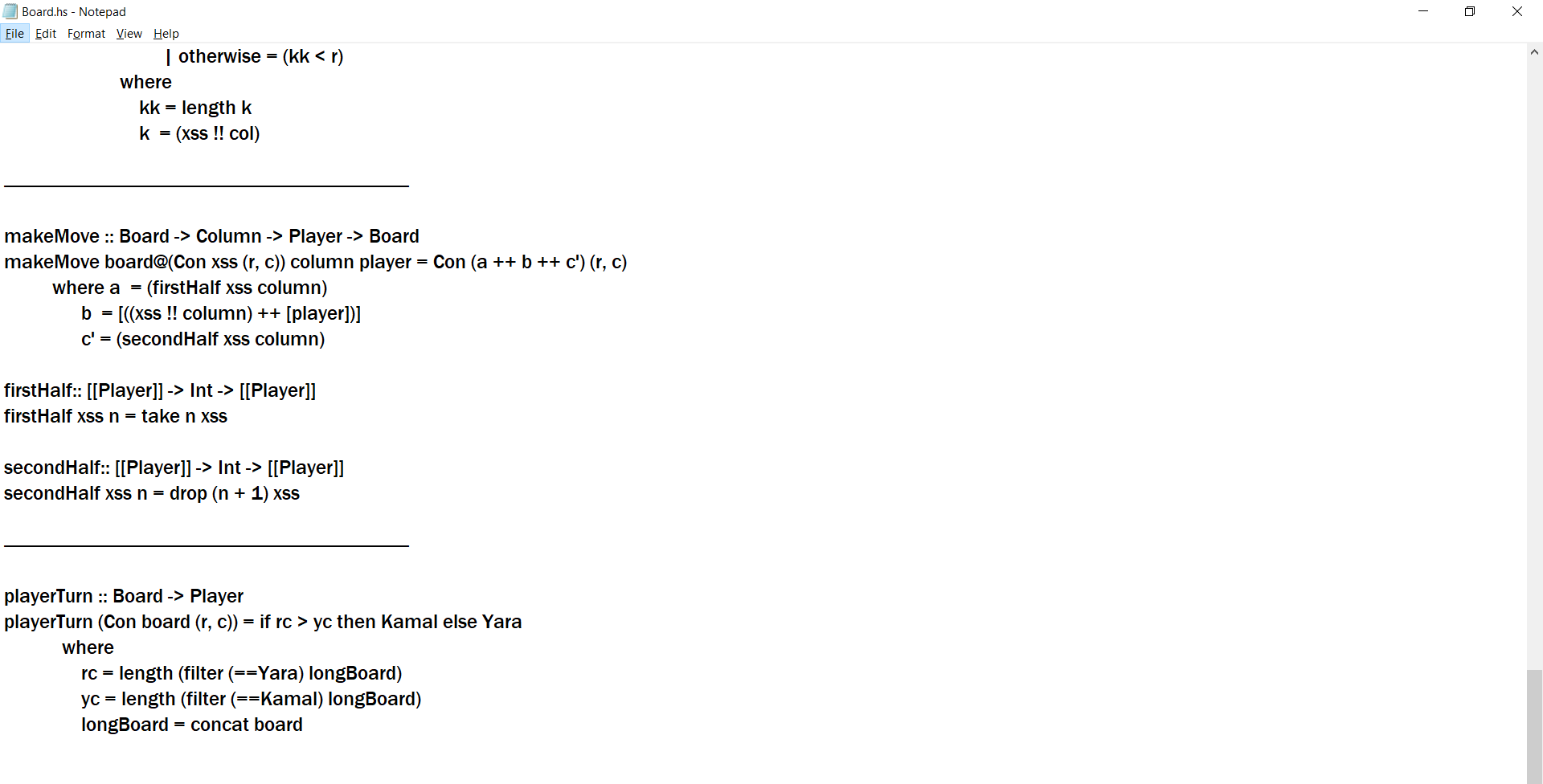
Transposes the board.   
b2ps: generates array of players from board.  
ps2b: generates board from array of players.



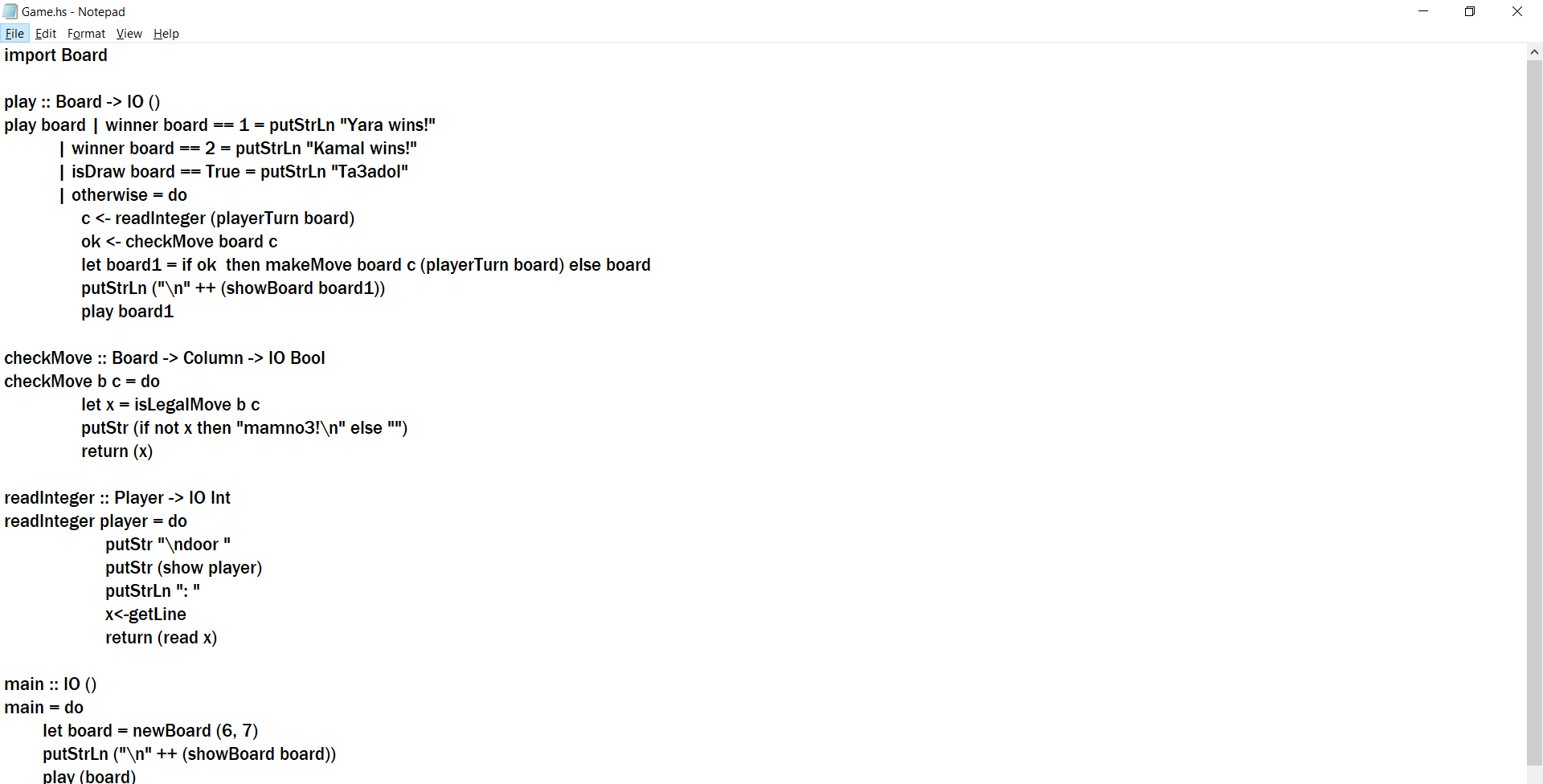
Checks diagonals and columns and columns of the transpose to see if Player is the winner.  
Finds the winner. Checks if board is full.

Convert board to string. Generates the trailing lines for the output. Creates output for each cell. Separated the cells of one row.  
Joins the rows together, original table is created from columns.

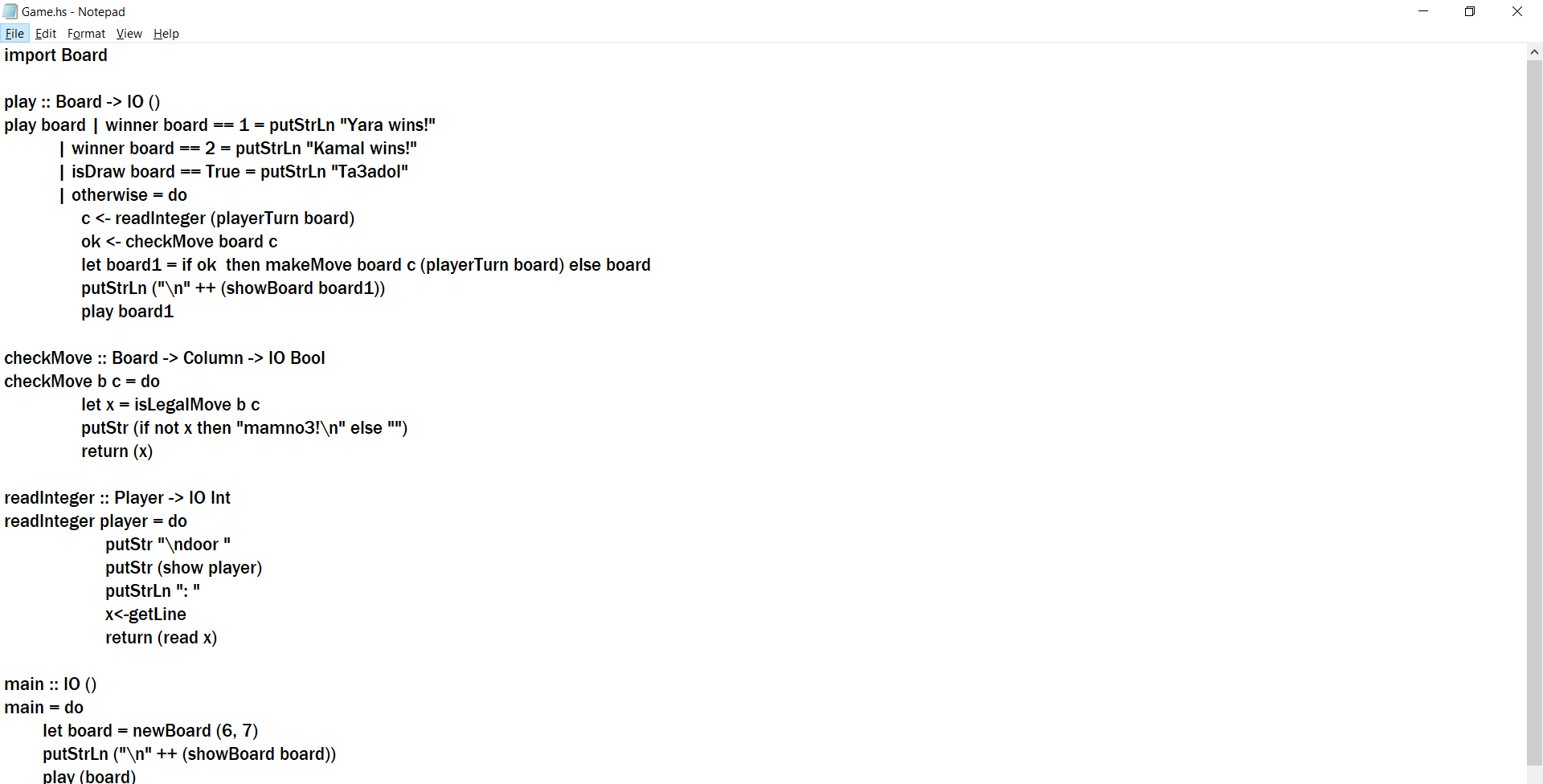
Checks if the move is legal.  
Make one move, raise errors for illegal move.  
First helper function to in makeMove, split to two-part, first half.  
Second helper function to in makeMove, split to two-part, second half.

Find out whose turn it is.

## **Game.hs**



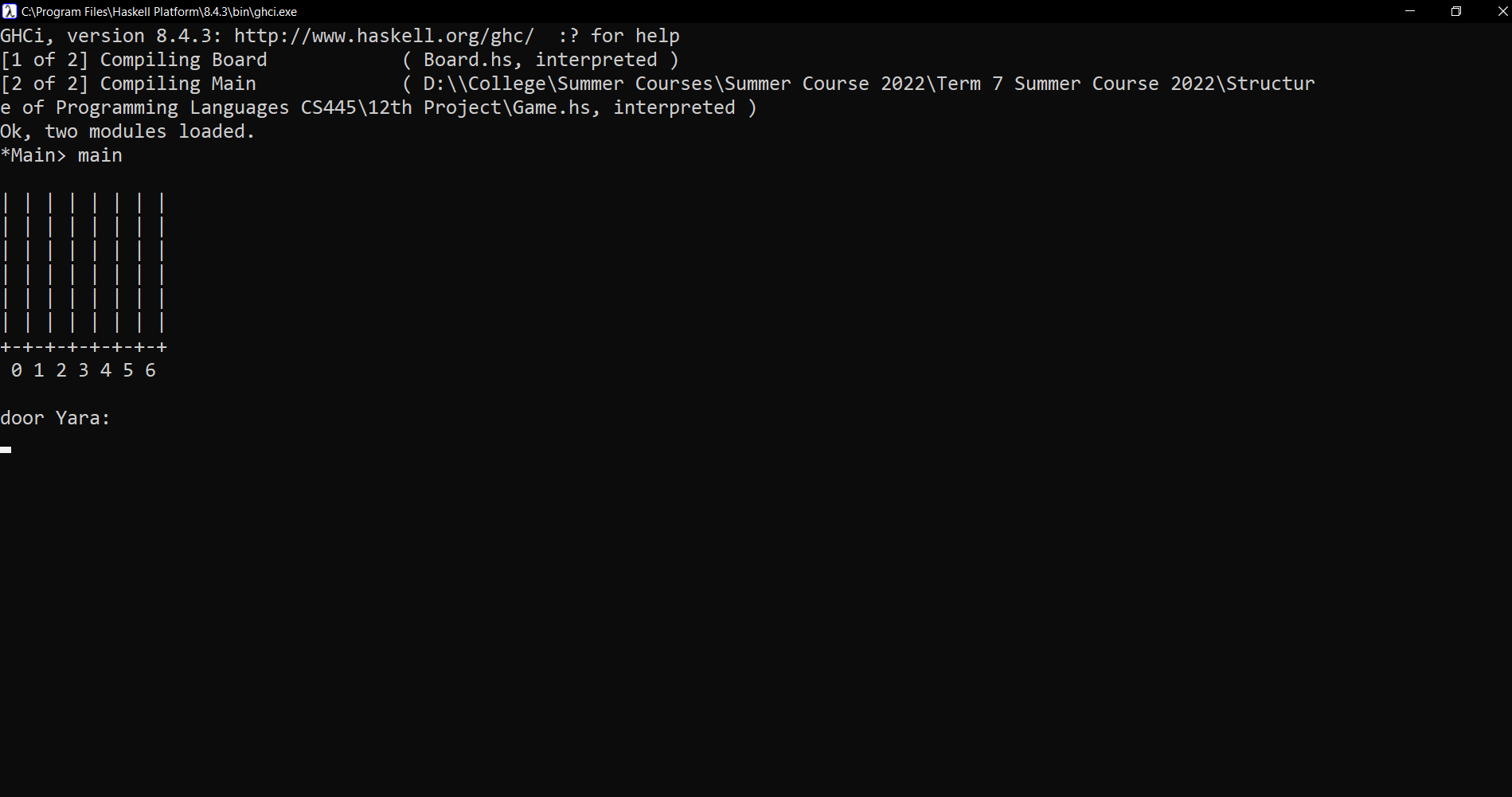
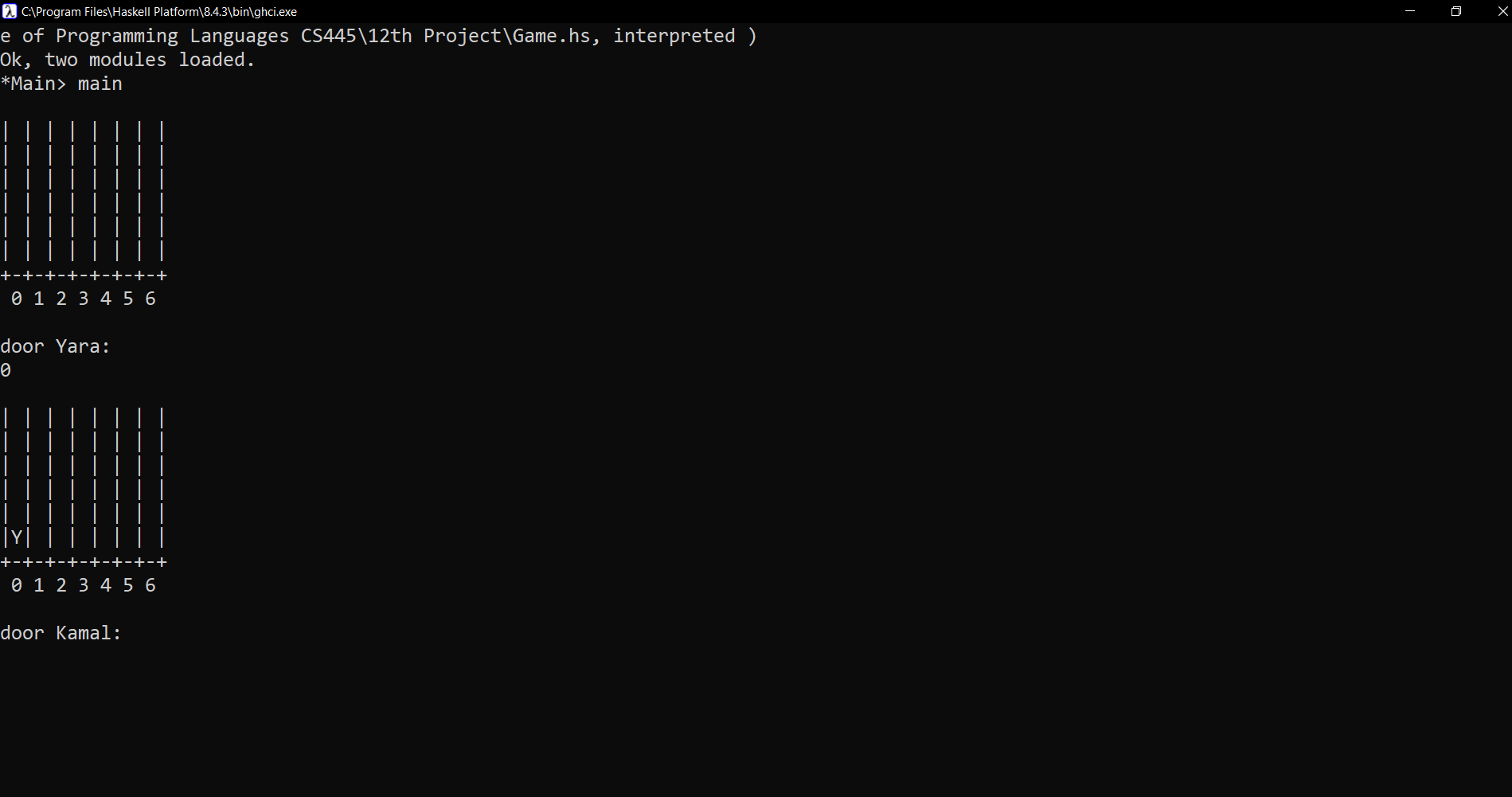
Core function, transform connect four into I/O, that ask users for input and display the board. Convert False to a String: Illegal Move! Announce wins and draws with the following strings.



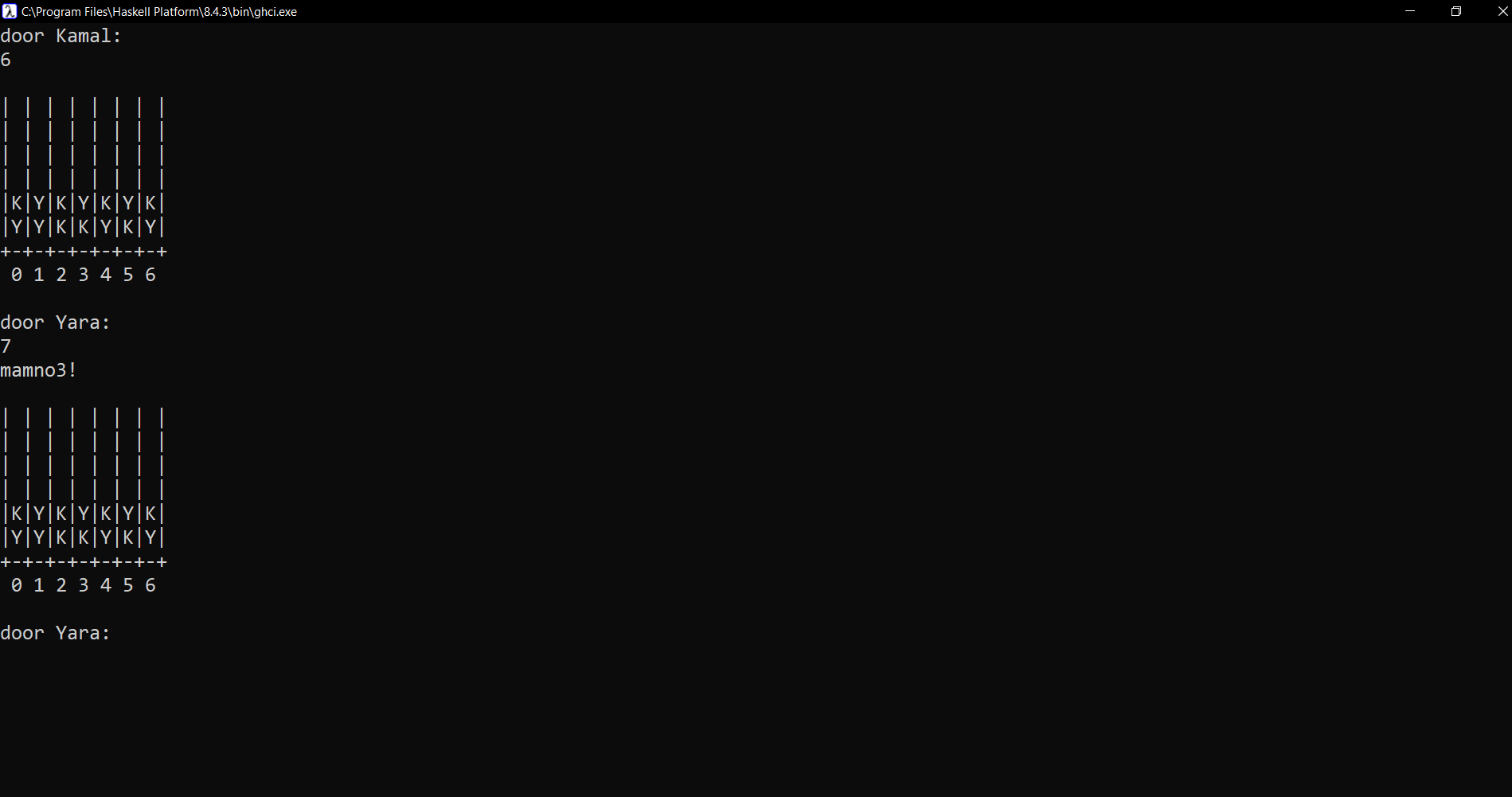
checkMove, check the move is valid or not, return True or False. Prompt for user input for each turn.

**Screenshots**

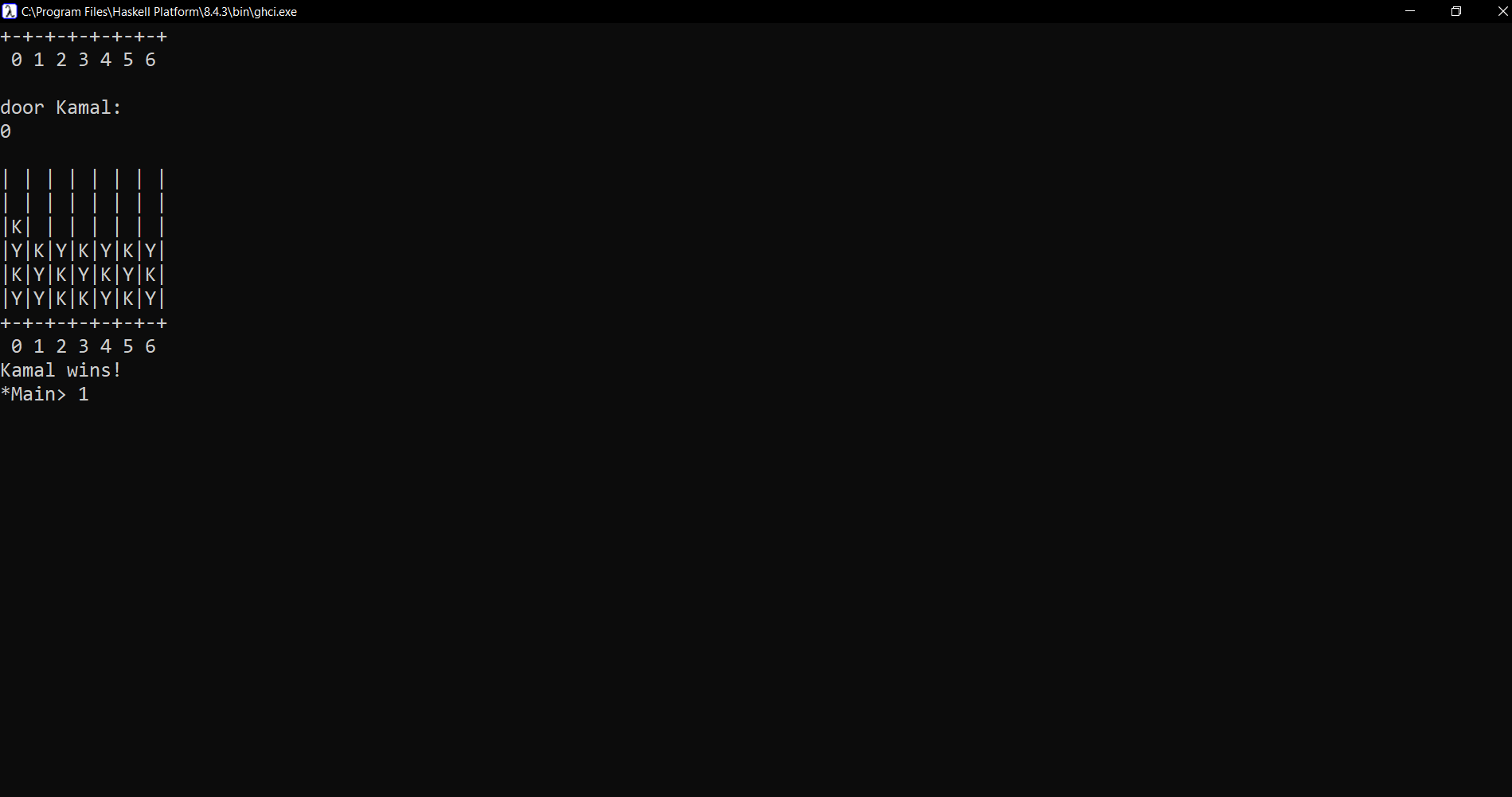
### **First snip of the running program**

### **Wrong move**



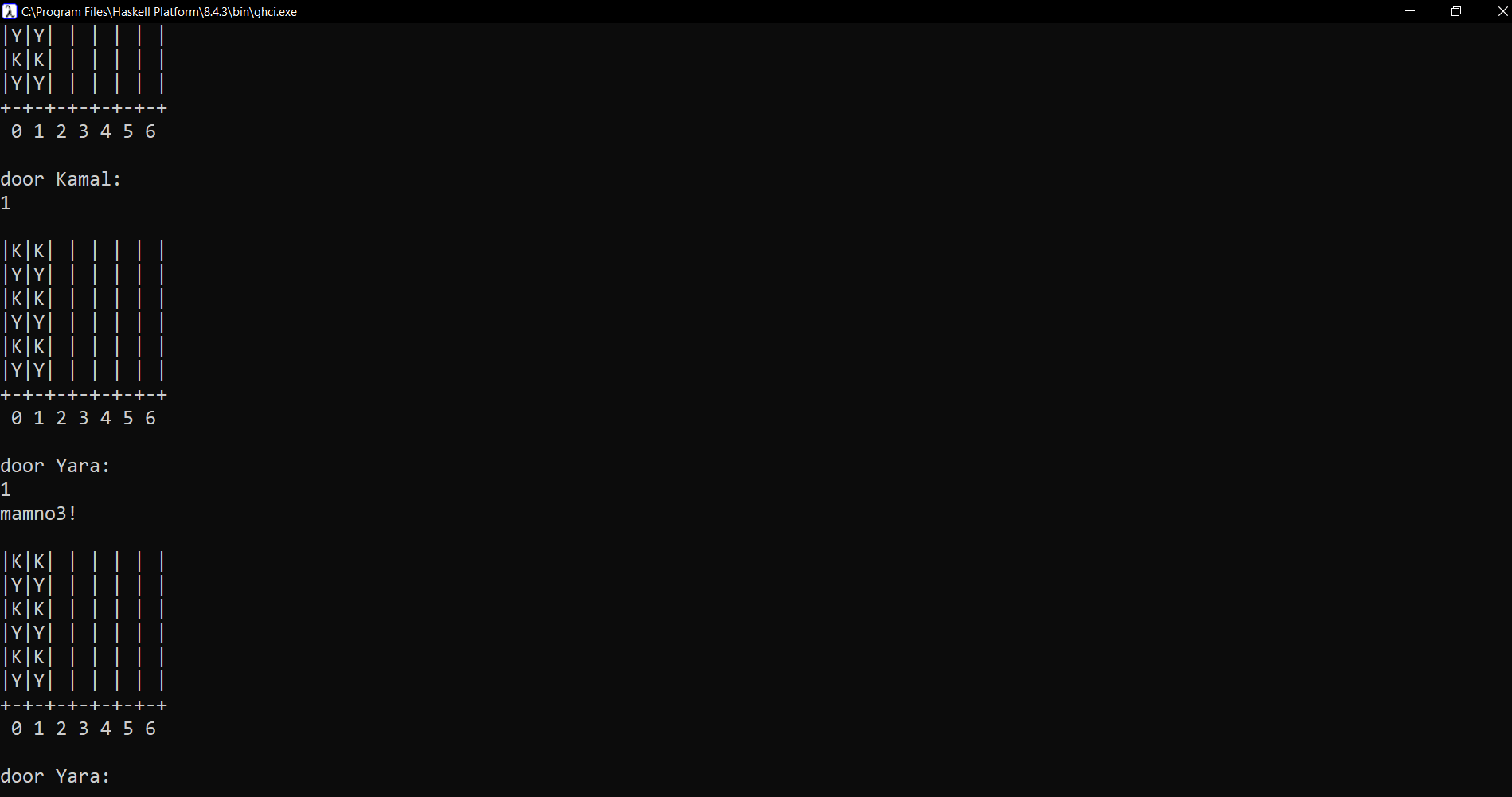
### **Wins left diagonal**



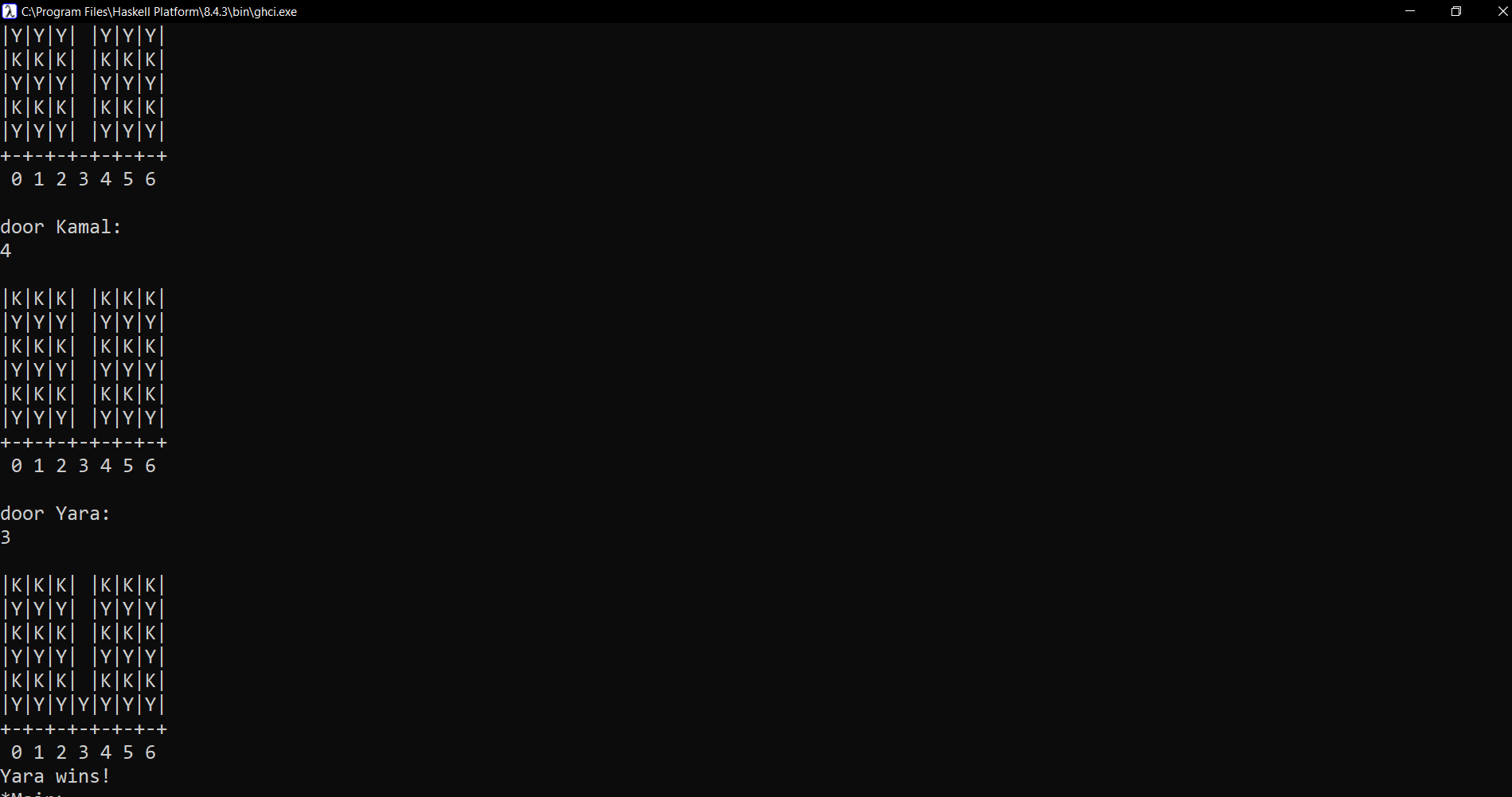
### **Wins right diagonal**



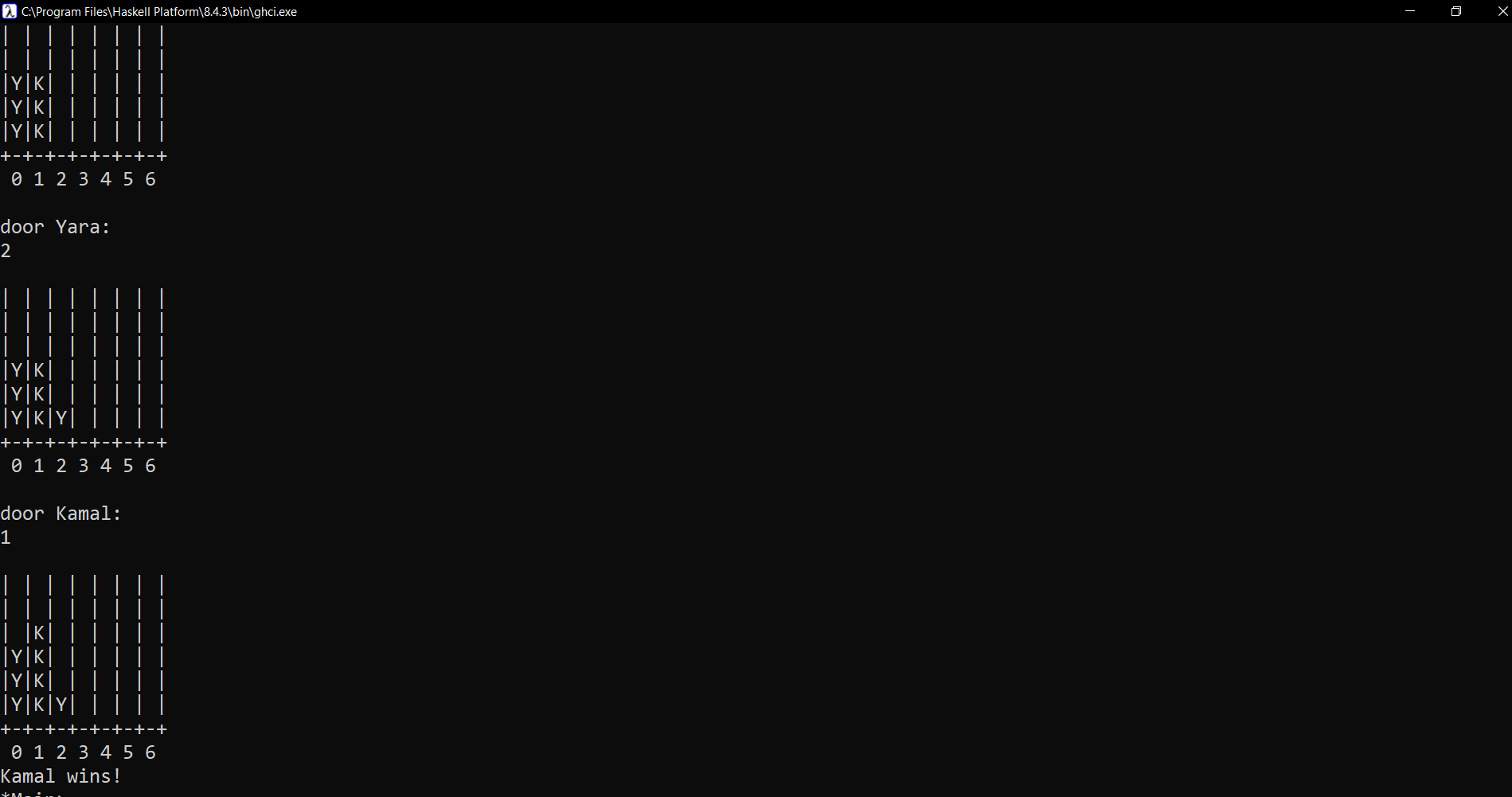
### **Wrong move on a full column**



### **Win horizontal row**



### **Win vertical column**



**Game Over – Draw**

