xo haskell game ui

# Main idea of the project

The two Haskell modules in this code that define the game of tic tac toe are the main concept. The "Board" module, which establishes the game logic and a game loop for the game of tic-tac-toe, uses a set of utility functions provided by the "Helper" module.

The "Helper" module contains functions for inserting and retrieving items from arrays, splitting a character array into single-length strings, determining the next move (either "x" or "o"), determining whether all items in a list are identical, and retrieving the diagonal elements of a 2D array.

The Move type is described as a pair of numbers, and the Board type is defined as a list of strings in the Board module. Additionally, it defines a gameLoop function that manages the game loop for a game of tic-tac-toe, as well as an emptyBoard that is a 3x3 board filled with space characters. With either one or two players, the gameLoop function offers logic for the computer to move when playing with just one player. Additionally, the Board module contains functions for processing player move input, receiving the computer's move, applying a move to the board, and rendering the board as a string. Additionally, it includes functions for determining whether a player has won the game or whether a tie resulted.

A practical way to group together several functions that may have the same names is to use Haskell modules, which group together a number of related features into a single package. The module definition is the first item in your Haskell file. Each file only contains one module, and it has a capital letter at the beginning of its name.

# Code Snippets

## Helper.hs

The Board module makes use of a variety of utility functions that are defined in the Helper module.

Text

Description automatically generated

**These functions include:**

**put** is a function that takes in three arguments: an integer **pos** representing the position at which to insert the new value, a value **newVal** of any type, and a list **list** of values of the same type as **newVal**. It returns a new list with **newVal** inserted at the specified position.

**put2d** is a function that inserts a value into a 2D list (a list of lists) at a given position. It takes in four arguments: two integers **x** and **y** representing the position at which to insert the value, a value **newVal** of any type, and a 2D list **mat** of values of the same type as **newVal**. It returns a new 2D list with **newVal** inserted at the specified position.

**get2d** is a function that retrieves a value from a 2D list (a list of lists) at a given position. It takes in three arguments: two integers **x** and **y** representing the position of the value to retrieve, and a 2D list **mat** of values. It returns the value at the specified position.

**strArr** is a function that takes in a string and returns a list of single-length strings, where each element of the output list is a string containing a single character from the input string.

**nextChar** is a function that takes in a character and returns the next character in the sequence 'x', 'o', 'x', 'o', etc. If the input character is not 'x' or 'o', the function will return 'x'.

**allSame** is a function that takes in a list and returns **True** if all elements of the list are equal, and **False** otherwise.

**diagUR** is a function that returns a list of the elements of the main diagonal of a 2D list (a list of lists), starting from the upper right corner and moving to the lower left corner. The function assumes that the input is a square 2D list (i.e., a list of lists with the same number of rows and columns).

**diagUL** is a function that returns a list of the elements of the main diagonal of a 2D list (a list of lists), starting from the upper left corner and moving to the lower right corner. The function assumes that the input is a square 2D list (i.e., a list of lists with the same number of rows and columns).

## Board.hs

Graphical user interface, text, application

Description automatically generatedText

Description automatically generated

Graphical user interface, text, application

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text, letter

Description automatically generated

Text

Description automatically generated

A picture containing text

Description automatically generated

This code defines a module called **Board** which exports several functions related to a tic-tac-toe game.

The module also imports several other modules: **Helper**, **Data.List**, **Data.Char**, and **Data.Time.Clock.POSIX**. The functions from these modules are used throughout the code to perform various tasks such as inserting elements into lists, generating random numbers, and converting between types.

Here is a detailed explanation of the code in the **Board** module:

**Board** is a type synonym for a 2-dimensional list of characters. It is used to represent the state of the game board in a tic-tac-toe game.

**Move** is a type synonym for a tuple of two integers. It is used to represent a move on the game board, with the integers representing the x and y indices of the move.

**emptyBoard** is a constant value representing an empty 3x3 tic-tac-toe board, with all spaces. It is defined as **[" ", " ", " "]**.

====================================================================

**gameLoop** is a function that runs the game loop for a tic-tac-toe game. It takes a board, the current player's character, and a flag indicating single-player or two-player mode as arguments. It does the following:

* If the game is single-player and it is the CPU's turn (**playerChar** is 'o'), it generates a random move for the CPU using the **getCPUMove** function. It then applies the move to the board using the **doMove** function, and checks for a win or a tie using the **isWinner** and **catsGame** functions. If there is a win or a tie, it prints a message to the console and terminates the game. If there is neither a win nor a tie, it continues the game by calling itself with the updated board and the next player's character.
* If the game is two-player or it is the player's turn in single-player mode, it prompts the player to enter a move and reads the input from the console. If the input is 'exit', it terminates the game. If the input is a valid move, it applies the move to the board using the **doMove** function and checks for a win or a tie using the **isWinner** and **catsGame** functions. If there is a win or a tie, it prints a message to the console and terminates the game. If there is neither a win nor a tie, it continues the game by calling itself with the updated board and the next player's character. If the input is not a valid move, it prints an error message and calls itself with the unchanged board and the same player's character.

====================================================================

**getCPUMove** is a function that generates a random move for the CPU player in single-player mode. It takes a board and a seed value as arguments, and returns a **Move** tuple representing the move. It does the following:

* It generates a random number using the seed value as an argument to the **random** function from the **System.Random** module (which is imported by the **Helper** module).
* It uses the **mod** function to map the random number to the range 0-8, representing the indices of the spaces on the board.
* It uses the **div** function to map the random number to the range 0-2, representing the row and column indices of the move.
* It uses the **doMove** function to apply the move to the board and check if it is valid. If the move is invalid, it calls itself with the unchanged board and a new seed value (the current system time in milliseconds). If the move is valid, it returns the move.

**===================================================================**

**doMove** is a function that takes a board, a move, and a player character as arguments, and returns a new board with the player's move applied to it. It does the following:

* It checks if the move is out of bounds. If the move's x and y values are not between 0 and 2 (inclusive), it returns a tuple containing the original board and **False**.
* It checks if the space on the board at the specified x and y coordinates is already occupied. If the space is not a space character, it returns a tuple containing the original board and **False**.
* It creates a new board by replacing the space at the specified x and y coordinates with the player character. It does this using the **put2d** function from the **Helper** module, which takes an x and y coordinate, a new value, and a 2D array as arguments, and returns a new 2D array with the value inserted at the specified coordinate.
* It returns a tuple containing the new board and **True**.

This function is used to apply a player's move to the game board. It is called from the **gameLoop** function after a player enters a move.

====================================================================

**parseMove** is a function that takes a string as an argument and returns a tuple containing a **Move** and a boolean value. It does the following:

* It checks if the string is in the correct format. The correct format is a string consisting of an uppercase letter followed by a digit. If the string is not in the correct format, it returns a tuple containing **(-1, -1)** and **False**.
* It converts the letter part of the string to a number using the **ord** function, which takes a character and returns its ASCII code as an integer. It subtracts the ASCII code for 'A' from the ASCII code for the letter to get the x coordinate of the move.
* It converts the digit part of the string to a number using the **digitToInt** function, which takes a character and returns its numerical value as an integer. It uses this value as the y coordinate of the move.
* It returns a tuple containing the **Move** and **True**.

This function is used to parse a player's move from a string into a **Move** tuple. It is called from the **gameLoop** function after a player enters a move.

====================================================================

**boardStr** is a function that takes a board and returns a string representation of the board. It does the following:

* It uses the **intercalate** function from the **Data.List** module to join the rows of the board into a single string, with a newline character between each row.
* It replaces each 'x' character in the string with a red 'x' using the **setSGRCode** function from the **System.Console.ANSI** module (which is imported by the **Helper** module).
* It replaces each 'o' character in the string with a blue 'o' using the **setSGRCode** function.
* It resets the console colors using the **setSGRCode** function.
* It returns the modified string.

This function is used to display the current state of the game board to the player. The **setSGRCode** function is used to color the 'x' and 'o' characters to make them more visually distinct.

The **boardStr** function is called from the **gameLoop** function whenever it is time to display the board to the player. The string returned by **boardStr** is printed to the console using the **putStrLn** function.

====================================================================

**isWinner** is a function that takes a board and a **Move** as arguments, and returns a boolean value indicating whether the player who made the move is the winner. It does the following:

* It checks if the move is a winning move by checking if the x and y coordinates of the move align with one of the winning conditions. There are 8 winning conditions:
  1. All spaces in the same row as the move are occupied by the same player
  2. All spaces in the same column as the move are occupied by the same player
  3. All spaces in the top-left to bottom-right diagonal that the move belongs to are occupied by the same player
  4. All spaces in the top-right to bottom-left diagonal that the move belongs to are occupied by the same player
* If the move is a winning move, it returns **True**. If the move is not a winning move, it returns **False**.

This function is used to determine if a player who made a move is the winner. It is called from the **gameLoop** function after every player's move.

====================================================================

**catsGame** is a function that takes a board and returns a boolean value indicating whether the board is a "cat's game", meaning that it is a tie because all spaces on the board have been filled with 'x' and 'o' characters but there is no winner.

It does the following:

* It uses the **map** function to apply the **any** function to each row of the board. The **any** function takes a predicate (a function that returns a boolean value) and a list as arguments and returns **True** if the predicate is **True** for any element of the list, and **False** otherwise. In this case, the predicate is the lambda function **(\c -> c == ' ')**, which returns **True** if its argument is a space character and **False** otherwise.
* It uses the **foldr1** function to reduce the list of boolean values returned by **map** to a single boolean value. The **foldr1** function takes a binary function (a function that takes two arguments) and a list as arguments and applies the binary function to the elements of the list in a right-associative manner. In this case, the binary function is the logical **or** function (**(||)**), which returns **True** if either of its arguments is **True**, and **False** otherwise.
* It negates the result of the **foldr1** function using the logical **not** function. If the **foldr1** function returns **True**, **catsGame** returns **False**, and if the **foldr1** function returns **False**, **catsGame** returns **True**.

This function is used to determine if the game is a tie because all spaces on the board have been filled with 'x' and 'o' characters but there is no winner. It is called from the **gameLoop** function to check for a tie after every player's move.

## main.hs

Graphical user interface, text, application

Description automatically generated

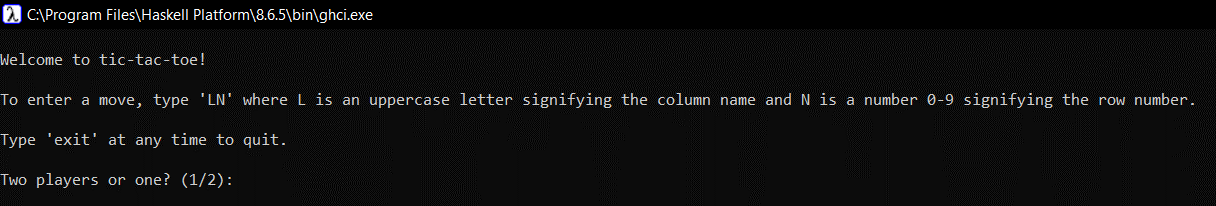
**main** is the main function of the program. It is the entry point of the program and is executed when the program is run. It does the following:

* It prints 20 empty lines on the screen using the **take** and **repeat** functions. **take** takes a number and a list, and returns a new list containing the specified number of elements from the original list. **repeat** takes an element and creates a list containing that element repeated indefinitely. So **take 20 $ repeat '\n'** creates a list containing 20 newline characters.
* It prints a welcome message to the user.
* It prints instructions on how to enter a move.
* It prints a message asking the user if they want to play a two-player game or a one-player game against the computer.
* It reads the user's input using the **getLine** function and stores it in the **input** variable.
* It checks the value of **input**.
  + If **input** is "1", it calls the **gameLoop** function with the empty board, the player character 'x', and the single-player flag **True**. This starts a one-player game against the computer.
  + If **input** is "2", it calls the **gameLoop** function with the empty board and the player character 'x', and the single-player flag **False**. This starts a two-player game.
  + If **input** is "exit", it terminates the program.
  + Otherwise, it prints an error message and calls itself again to ask for input again.

This function is the entry point of the program and sets up the game by asking the user for input and calling the **gameLoop** function with the appropriate arguments.

# Screenshots

## snip of the running program



## TWO PLAYERS

Text

Description automatically generated Text

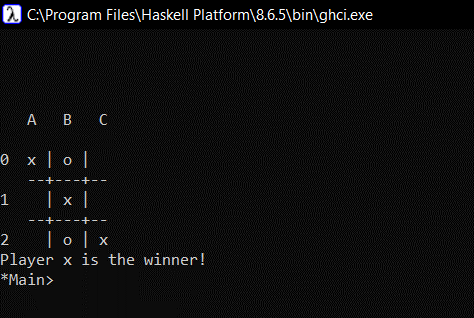
Description automatically generated

## pLAYER 1 PLAYING WITH CPU

Text

Description automatically generated

## Wins left diagonal

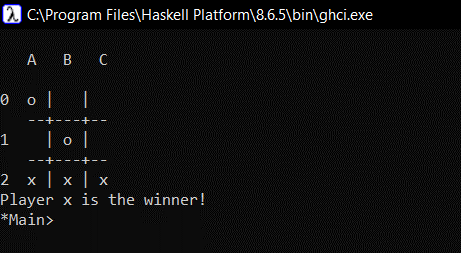


## Wins right diagonal

A picture containing diagram

Description automatically generated

## Win row

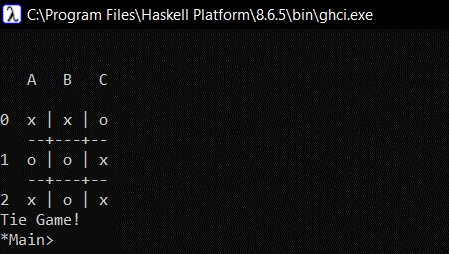


## Win column

A screenshot of a computer

Description automatically generated with medium confidence

## Game Over – Draw



## wrong input

Text

Description automatically generated

## Wrong move

A picture containing graphical user interface

Description automatically generated