Избрана тема:

Тема 1. Морски шах (Кръстчета и нулички)
Да се дефинира функция, която получава текуща игрова ситуация и предлага следващия ход на играча.
Да се реализира вход и изход от/на конзолата.

Ако не е проблем документацията на кода съм я написала на английски език (по-лесно мисля на този език като става дума за програмиране), но ако желаете ще изпратя и копие на български.

В самият код има документация във форма на коментари над всяка функция, а тук максимално ще се опитам да я доразвия заедно с визуални примери.

Documentation:

I chose to develop the minimax algorithm so I can generate the best next move given a current game board.

Using zipWith, concatMap and other functions from this library.
import Data.List
Found a very nifty way to change an element in matrix using some fuctions from here.
import Control.Lens
Using comparing.
import Data.Ord
Using random generator of Ints in an interval getStdRandom, randomR.
import System.Random
To convert an IO Int into a "normal" Int.
import System.IO.Unsafe
For the IO operations in main.
import System.IO

The game tree structure.
data Tree a = Nil | Node a [(Tree a)] deriving (Show,Eq)

It's easier to tink of it as a board.
type Board = [String]

-- The first parameter is the number of the row and the secons is the column number.
type Positions = (Int,Int)

Finding the diagonal of the board.
The parameter "x" is the board.
diag :: [[a]] -> [a]

Finding the second diagonal of the board.
The parameter "x" is the board.
secDiag ::[[a]] -> [a]

Checks if the board "b" is empty.

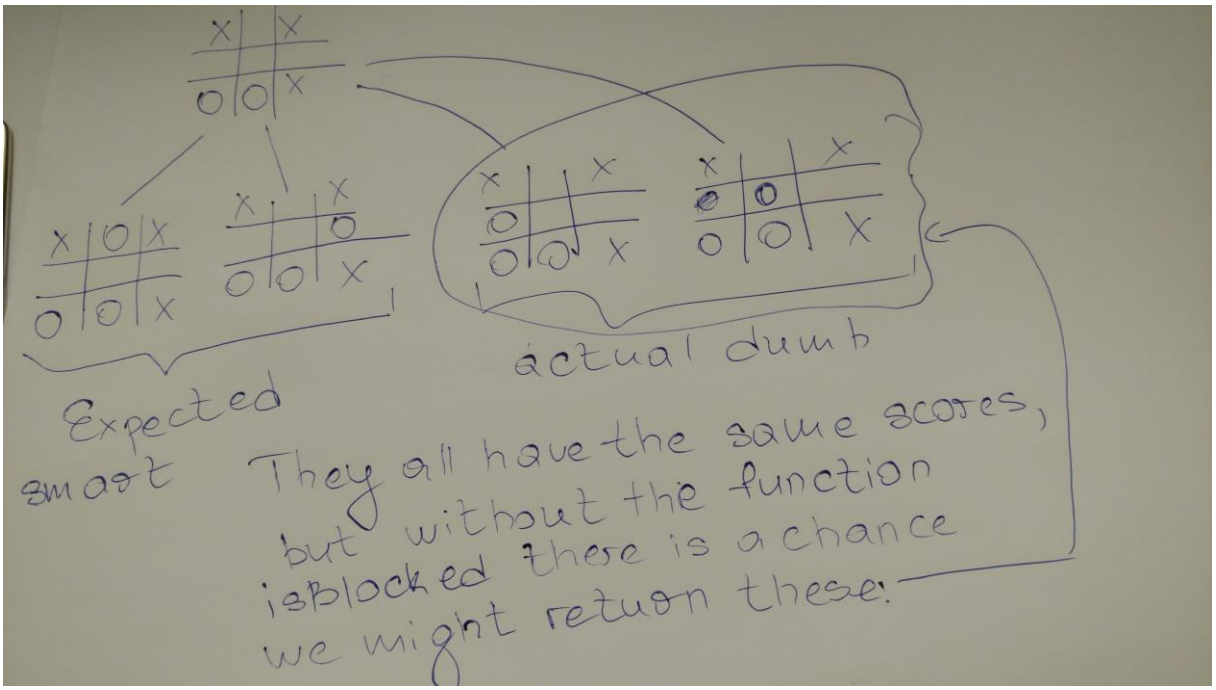
isEmpty :: Board -> Bool

Checks if the board "b" is full.
isFull :: Board -> Bool

We generate all the empty positions. We return a list of them.
emptySpace :: Board -> [Position]

We check if the board is won by the current player, who is the parameter "pl".
isWon :: Char -> [String] -> Bool

We check if we can block an opponent move. We use it to sort the boards when all of the scores are equal.
It will become clearer when we go to the genNextMove function.
The idea if that some moves may have equal scores, but we don't want to return the first random board.



isBlocked :: Board -> Int

Changing the current player.
other :: Char -> Char

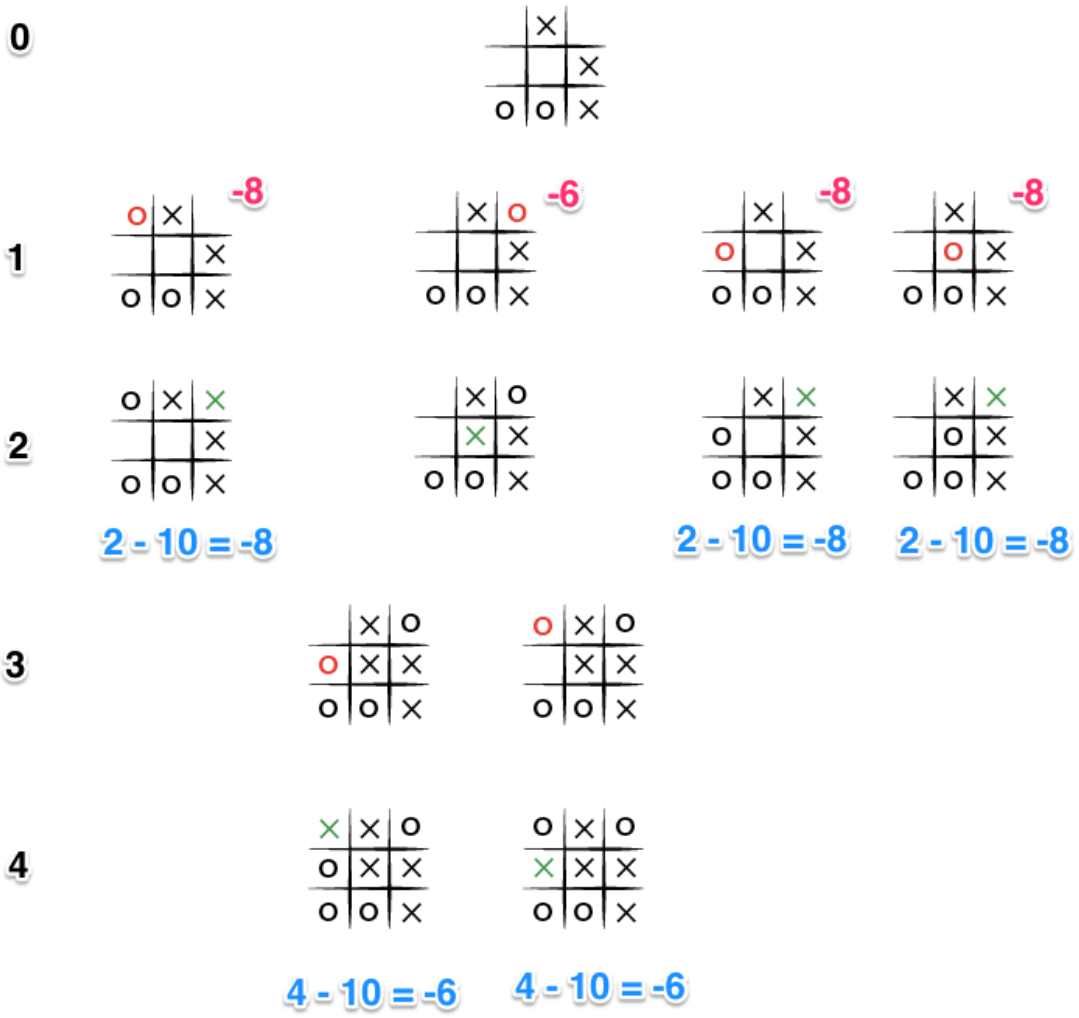
We generate the next possible moves using the positions from emptySpace.
If a winning board(s) is(are) generated then the generated along them do not pose any interest to us.
doMoves :: Char -> Board -> [Board]

Generating the game tree.
The leaves are the won boards and the full boards which have come to a draw.
generate :: Board -> Char -> (Tree Board)

For the minimax algorithm given a current game board, a player and the depth of the recursion on the current step of the scan of the already generated game tree.

makeScore :: Char -> Board -> Int -> Int
If the current player wins we give a positive score.
If the opponent player wins we give a negative score.

We map the makeScore function to each board and that is how we generate the score tree.
Here the parameters are the game tree, the first player and the depth of the recursive scan.
mapTree :: (Tree Board) -> Char -> Int -> (Tree Int)
We keep track of the recursive scan so we can give a more realistic score.
For example if we have a board in which we are bound to win whatever we do, we



would like to prolong the game for as long as we can so for example:

We gather the leaves of the tree (the Tree of scores) and return them as a list.
getLeaves :: Eq a => Tree a -> [a]

We get the maximum element of the leaves of the tree of scores.
getMax :: (Tree Int) -> Int

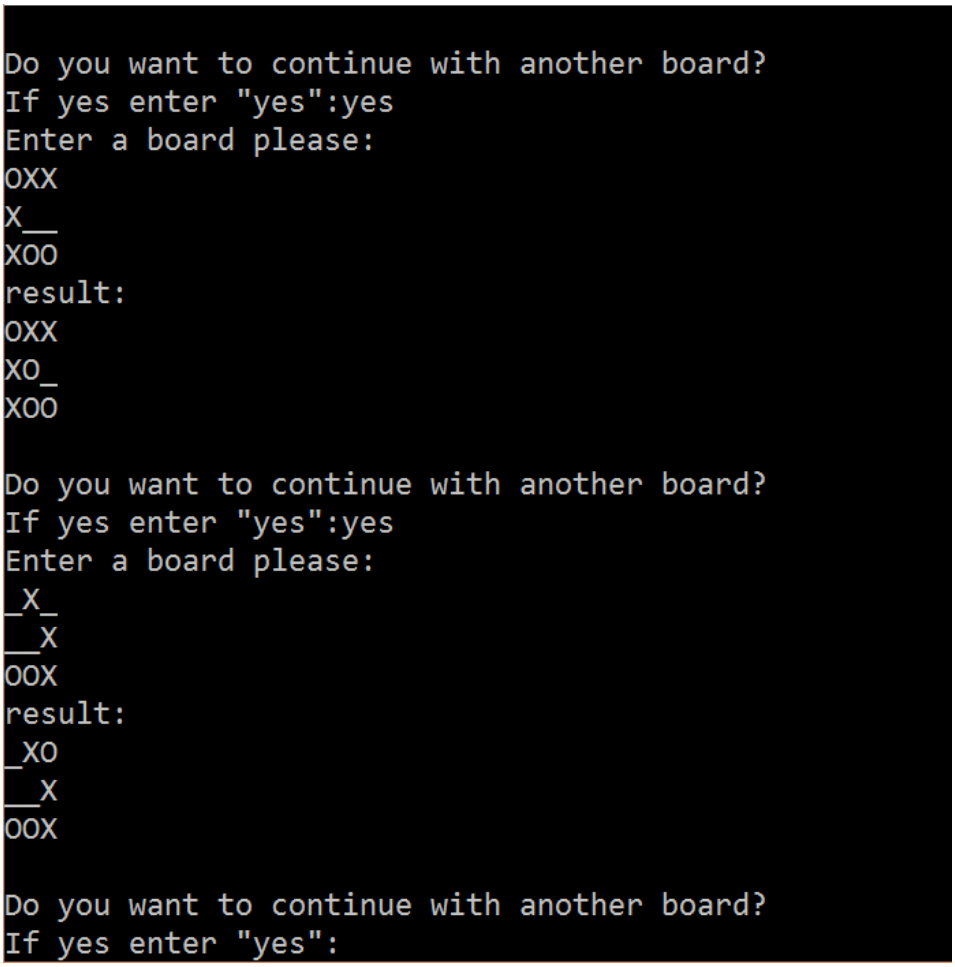
The main function which takes the initial player and the board.
We handle the times when the board is full or already won by any of the players.
If the board is empty we generate the next board with the X put at the 4 random corners of the old empty one.
Otherwise we take the board with the best score generated by minimax algorithm and the best blocking score.
genNextMove :: Char -> Board -> Board

We see which players turn it is.
If the board is invalid we return a 'N' which is an error message.
X always starts first.
initPlayer :: Board -> Char

The function I use to mask that genNextMove takes a second parameter - the initial player
generateMove :: Board -> Board

Helps IO printing of the board in main
makePresentable :: Board -> String

main :: IO()
The only problem is that a valid board looks like this:
You can easily change the "_" symbol. Go to isEmpty, isFull and emptySpace functions.



Used literature:
http://neverstopbuilding.com/minimax