```
testBoard = [[1, 0, 0, 0, 0, 1, 0], [0, 1, 0, 0, 2, 1, 0], [0, 0, 1, 2, 1, 1, 0], [0, 0, 2, 2, 0, 0, 1], [1, 0, 2,
0, 1, 0, 1], [2, 1, 0, 1, 0, 2, 1]]

testBoard' = [[1, 0, 0, 0, 0, 2, 0], [0, 1, 0, 0, 2, 1, 0], [0, 0, 1, 2, 1, 1, 0], [0, 0, 2, 1, 0, 0, 1], [1, 0,
2, 0, 1, 0, 1], [2, 1, 0, 1, 0, 2, 1]]

testBoard1 = [[1, 0, 0, 0, 0, 2, 0], [0, 1, 0, 0, 2, 1, 0], [0, 0, 1, 2, 0, 1, 0], [0, 0, 2, 1, 2, 1, 1], [1, 0,
2, 0, 2, 1, 1], [2, 1, 0, 1, 0, 2, 1]]

testBoard2 = [[1, 0, 0, 0, 0, 2, 0], [0, 1, 0, 0, 2, 1, 0], [0, 0, 1, 1, 1, 1, 0], [0, 0, 2, 1, 0, 0, 1], [1, 0,
2, 0, 1, 0, 1], [2, 1, 0, 1, 0, 2, 1]]

draw_board :: [[Int]] -> [Char]
-- Turn the position values of a connect-four board into an image of the board
draw_board [] = ""
draw_board (x:xs) = (convert x) ++ "\n" ++ draw_board xs

convert :: [Int] -> [Char]
-- Turn one row of position values into an image of one row of the board
convert [] = ""
convert (y:ys)
  | y == 0 = "_" ++ convert ys
  | y == 1 = "X" ++ convert ys
  | y == 2 = "O" ++ convert ys

pull :: [[Int]] -> (Int, Int) -> Int
-- Takes a value out of the board using x and y values
-- Right = x increases, Down = y increases
pull l (x,y) = l !! (y - 1) !! (x - 1)

vacant :: [[Int]] -> (Int, Int) -> Bool
-- Takes a position and sees if a check is not in that space
vacant l (x,y) = (pull l (x,y)) == 0

is_legal_move :: [[Int]] -> Int -> (Int, Int) -> Bool
-- Takes a position and see if that is allowed in the game
is_legal_move b p (x,y)
  | length b < y = False
  | length b == y = (vacant b (x,y))
  | otherwise = (vacant b (x,y)) && (not (vacant b (x,(y+1))))

update_row :: [Int] -> Int -> Int -> [Int]
```

```
-- Takes in a column value and returns the row that has the updated value of the specific
column
update_row (x:xs) a v
  | a > length (x:xs) = (x:xs)
  | a == 1 = v : xs
  | otherwise = x : update_row xs (a-1) v

findc_update :: [[Int]] -> Int -> (Int, Int) -> [[Int]]
-- Takes in the board and position and returns the updated board with the value in the position
changed. Does it by scanning through the rows first then updating the selected row
findc_update (x:xs) a (b,c)
  | c > length (x:xs) = (x:xs)
  | c == 1 = (update_row x b a) : xs
  | otherwise = x : findc_update xs a (b,(c-1))

make_move :: [[Int]] -> Int -> (Int, Int) -> [[Int]]
-- Checks to see if the position is valid then returns the updated board
make_move b p (x,y)
  | is_legal_move b p (x,y) && (p == 1) = findc_update b p (x,y)
  | otherwise = b

scan_u l (a,b) x = pull l (a,(b-x))
scan_l l (a,b) x = pull l ((a-x), b)

check_vertical' :: [[Int]] -> (Int, Int) -> Bool
check_vertical' l (a,b) = (initial == scan_u l (a,b) 1) && (initial == scan_u l (a,b) 2) && (initial ==
scan_u l (a,b) 3) && (initial /= 0)
     where initial = pull l (a,b)

check_vertical :: [[Int]] -> (Int, Int) -> Bool
-- Start at (1,4)
check_vertical l@(x:xs) (a,b)
  | a == (length x) && b == (length l) = check
  | b == (length l) = check || check_vertical l ((a+1), 4)
  | otherwise = check || (check_vertical l (a,b+1))
    where check = check_vertical' l (a,b)

check_horizontal' :: [[Int]] -> (Int, Int) -> Bool
check_horizontal' l (a,b) = (initial == scan_l l (a,b) 1) && (initial == scan_l l (a,b) 2) && (initial ==
scan_l l (a,b) 3) && (initial /= 0)
     where initial = pull l (a,b)

check_horizontal :: [[Int]] -> (Int, Int) -> Bool
```

```haskell
-- Start at (4,1)
check_horizontal l@(x:xs) (a,b)
  | a == (length x) && b == (length l) = check
  | a == (length x) = check || check_horizontal l (4,(b+1))
  | otherwise = check || check_horizontal l ((a+1), b)
    where check = check_horizontal' l (a,b)

scan l (a,b) x = pull l ((a-x), (b-x))
scan' l (a,b) x = pull l ((a-x), (b+x))

check_right_diag' :: [[Int]] -> (Int, Int) -> Bool
check_right_diag' l (a,b) = (initial == scan l (a,b) 1) && (initial == scan l (a,b) 2) && (initial ==
scan l (a,b) 3) && (initial /= 0)
                  where initial = pull l (a,b)

check_right_diag :: [[Int]] -> (Int,Int) -> Bool
-- Must always start at (4,4)
check_right_diag l@(x:xs) (a,b)
  | a == (length x) && b == (length l) = check
  | a == (length x) = check || check_right_diag l (4,(b+1))
  | otherwise = check || check_right_diag l ((a+1), b)
    where check = check_right_diag' l (a,b)

check_left_diag' :: [[Int]] -> (Int,Int) -> Bool
-- This goes top-down
check_left_diag' l (a,b) = (initial == scan' l (a,b) 1) && (initial == scan' l (a,b) 2) && (initial ==
scan' l (a,b) 3) && (initial /= 0)
                where initial = pull l (a,b)

check_left_diag :: [[Int]] -> (Int,Int) -> Bool
-- Must always start at (4,1)
check_left_diag l@(x:xs) (a,b)
  | a == (length x) && b == ((length l) - 3) = check
  | a == (length x) = check || check_left_diag l (4, (b+1))
  | otherwise = check || check_left_diag l ((a+1), b)
    where check = check_left_diag' l (a,b)

check_diag l = (check_left_diag l (4,1)) || (check_right_diag l (4,4))

is_won l = check_diag l || check_horizontal l (4,1) || check_vertical l (1,4)

main = do
  putStrLn $ (draw_board testBoard)
  putStrLn $ show (pull testBoard (3,2))
  putStrLn $ show (make_move testBoard 3 (2,2))
  putStrLn $ show (is_legal_move testBoard 1 (2,1))
  putStrLn $ show (is_legal_move testBoard 1 (1,2))
  putStrLn $ show (make_move testBoard 1 (1,3))
  putStrLn $ show (is_won testBoard)
  putStrLn $ show (is_won testBoard')
  putStrLn $ show (is_won testBoard1)
  putStrLn $ show (is_won testBoard2)
```