Chetan Hiremath

Dr. Johnson

EECS 368

April 28, 2022

EECS 368 Assignment 7

Haskell Source Code:

--This line imports the Char Data that contains the built-in functions for the Char Data type for the Nim game since those functions are required to print the characters.

import Data.Char

--This line shows the Board's type declaration, which is [Int].

type Board = [Int]

--This line defines the initial function. It considers Board or [Int] as the output, which is enclosed with brackets since you need the rows to represent the game's board.

initial :: Board

--This line prints the list of the values of the rows for the game's board.

initial = [5,4,3,2,1]

--This line defines the nextPlayer function. It considers Int as the input and Int as the output.

nextPlayer :: Int -> Int

--These lines return the next players when those players have their respective turns to play the game.

nextPlayer 1 = 2

nextPlayer 2 = 1

--This line defines the newLine function. It considers IO () as an output that returns no result value.

newLine :: IO ()

--This line shows the putChar action that writes a character, which is a new line, to the screen and returns no result value. Therefore, it creates a new line.

newLine = putChar '\n'

--This line defines the getDigit function. It considers String as the input and IO Int as the output that returns an integer.

getDigit :: String -> IO Int

--This line takes, checks, and returns the input if it's a digit or an integer. If not, then it asks you to write the correct input as an integer to complete the turn.

getDigit message = do putStr message

x <- getChar

newLine

if isDigit x then

do return (digitToInt x)

else

do putStrLn "Invalid input! Try again!"

getDigit message

--This line defines the printBoard function. It considers Board or [Int] as the input and IO () as the output that returns no result value.

printBoard :: Board -> IO ()

--This line prints the board by appending the rows, their respective colons, and their respective stars by the list comprehension.

printBoard board = putStr $ unlines [(show row) ++ " : " ++ replicate stars '\*'| (stars, row) <- zip board [1..length board]]

--This line defines the checkEnd function. It considers Board or [Int] as the input and Bool as the output.

checkEnd :: Board -> Bool

--This line checks the end condition of the game. It returns True if the checking is successful. If not, it returns False.

checkEnd board = all (==0) board

--This line defines the checkValidMove function. It considers Board or [Int], Int, and Int as the inputs and Bool as the output.

checkValidMove :: Board -> Int -> Int -> Bool

--This line checks whether the move is valid for the row and its respective stars. If the move is valid, then it returns True. If not, it returns False.

checkValidMove board row rowStars = if (board !! (row - 1)) >= rowStars then True else False

--This line defines the generateNewBoard function. It considers Board or [Int], Int, and Int as the inputs and Board or [Int] as the output.

generateNewBoard :: Board -> Int -> Int -> Board

--This line refreshes and changes the board when the player completes the move.

generateNewBoard board row rowStars = [if rows == row then stars - rowStars else stars | (stars, rows) <- zip board [1..length board]]

--This line defines the play function. It considers Board or [Int] and Int as the inputs and IO () as the output that returns no result value.

play :: Board -> Int -> IO ()

--This line uses the do notation that runs the game in a loop. If the player clears the board, then it prints a string that shows that this player wins the game.

--If not, it tells the players to continue the game by picking the row and its numbers of stars until the board is empty. If the number of stars is valid,

--then the generateNewBoard function removes those stars and changes the board. If the number of stars is invalid, then it prints a string that shows that

--the move is invalid. That player's turn will be over, and another player will get the turn to play the game.

play board player = do printBoard board

if checkEnd board then

do putStr "Player "

putStr (show (nextPlayer player))

putStrLn " wins!"

else

do putStr "Player "

putStrLn (show player)

row <- getDigit "Enter a row number: "

stars <- getDigit "Stars to remove: "

if checkValidMove board row stars then

play (generateNewBoard board row stars) (nextPlayer player)

else

do putStrLn "Invalid move! Try again!"

play board player

--This line defines the nim function. It considers IO () as the output that returns no result value.

nim :: IO ()

--This line tells the first player to start the game with the initial board.

nim = play initial 1