Práctica Haskell: 4 en raya

Aarón Portales Rodrigo

Funciones:

Atoi

atoi::String->Int

atoi=read

Declaración de la función atoi que pasa a int un char

Tablero

tablero n m l t = if length t < m then tablero n m l (inser_fila n l t) else t

Tablero imprime una lista de listas, va añadiendo recursivamente listas de ceros llamando a la función: inser_fila hasta que la longitud es m

inser_fila n l t = t++[columnas n l]

inser_fila llama a columnas que inserta tantos ceros como el numero de columnas se le haya pasado

columnas n l = if length l < n then columnas n (inser_cero l) else l

inser_cero [] = [0]

inser_cero [x] = [x,0]

inser_cero (x:xs) = (x:xs)++[0]

Pintar

pintar [x] = show x

pintar (x:xs) = show x ++"\n"++ pintar xs

Pinta una lista dando un salto de linea entre cada elemento

Lista_colum

lista_colum t n = if last t == (n)

then t

else t++(lista_colum [q+1] n)

where q = primero t

Transforma a una lista la columna n del tablero (lista de listas)

Insertar

insertar x t c m = inserlist (inser x c (t!!f)) f t

where f = pos_columna t m c

Inserta x en el tablero t de m filas en la posicion de la columna c que primero este libre.

Pos_columna

pos_columna t m c = if t!!m!!c /= 0 then pos_columna t (m-1) c else m

Devuelve el numero de la primera fila vacia de la columna c

Va comprobando, si la posicion es distinto de cero (esta ocupada) se incrementa para comprobar la de más ariba.

inser

inser x n [] = [x]

inser x n (y:ys) = q++[x]++i

where q = take n (y:ys)

i = resto n (y:ys)

Inserta x en la posicion n de la lista l. la inserta antes de los primeros n elementos y despues cogemos el resto llamando a la funcion que hemos definido como resto.

inserlist

inserlist i c [[]] = [i]

inserlist i c (y:ys) = q++[i]++h

where q = take c (y:ys)

h = resto c (y:ys)

Inserta lista i en lista de listas l en la pos c(que es columna). Utiliza el mismo método que la función anterior pero en este caso para lista de listas.

resto

resto n [] = []

resto 0 (x:xs) = xs

resto n (x:xs) = if n/=0 then resto (n-1) (tail (x:xs)) else xs

Devuelve la sublista sobrante que deja el take n xs. Se comprueba que n es distinto de cero, si es así se llama recursivamente a resto eliminando el primer elemento de xs. Si es 0 devolvemos xs.

es_posible

es_posible t m n x = if limit n x == True

then llena t m x

else False

Esta función comprueba si es posible insertar una ficha donde le dice el jugador

limit

limit n x = if x>n then False else if x<0 then False else True

Comprueba que x se sale de los limites, para comprobar que el jugador no introduce una ficha fuera del tablero.

llena

llena t m x = if t!!0!!x == 0 then True else False

Devuelve verdadero si la columna está llena, comprueba si la ultima ficha esta ocupada.

comph

comph t m = if m<0 then False else comph1 t m

comph1 t m = if horizontal q == True then True else comph t (m-1)

where q = t!!m

Le pasamos el tablero y las filas que hay, comprueba si hay cuatro en linea en todas las listas horizontales. Devuelve un boleano.

horizontal

horizontal l = if igual12 q

then if length q >= 4 then True else False

else False

where q = mayoritario l

Comprueba que hay linea en una lista horizontal

igual12

igual12 l = if (primero l) == 1 || (primero l) == 2 then True else False

Comprueba si el primer elemento de una lista es una ficha. Para comprobar que las columnas no están llenas y para comprobar que hay 4 en lina, ya que se pasa la lista mayor que 4 y si es ficha hay linea.

mayoritario

mayoritario [] = []

```
mayoritario [x] = [x]

mayoritario (x:xs) = if length j >= length q then j else q
  where j = lista_seguida (x:xs)
        q = mayoritario (resto_seguida (x:xs))
```

Devuelve la lista seguida más larga.

lista_seguida

```
lista_seguida [] = []

lista_seguida [x] = [x]

lista_seguida (x:xs) = if x==q then [x]++(lista_seguida xs) else [x]
  where q = primero xs
```

Devuelve la lista de los primeros x seguidos.

resto_seguida

```
resto_seguida [] = []

resto_seguida [x] = []

resto_seguida (x:xs) = if x==q then resto_seguida xs else xs
  where q = primero xs
```

Devuelve la lista quitando los primeros x seguidos que sean iguales. El resto de la anterior función.

columnalista

```
columnalista t m n = if m>0 then [t!!m!!n]++columnalista t (m-1) n else [t!!0!!n]
```

Devuelve una lista de la columna n.

compv

compv t m n = if n>0

then if compv1 t m n == False then compv t m (n-1) else True

else compv1 t m 0

Comprueba todas las linas que puede haber en vertical.

compv1

compv1 t m n = vertical q

where q = columnalista t m n

Comprueba si hay 4 en lina en una sola columna, para hacer el recorrido de todas la función anterior.

vertical

vertical l = if igual12 q

then if length q >= 4 then True else False

else False

where q = mayoritario l

Comprueba que hay 4 en linea en una lista.

list_diagonal1

list_diagonal1 t m n j = if length j < 4

then list_diagonal1 t (m-1) (n-1) (j++[t!!m!!n])

else j

Crea la lista de la diagonal con la siguiente forma:

{-Diagonal_1

*

*

*

* -}

list_diagonal1Com

list_diagonal1Com l = if igual12 q

then if length q >= 4 then True else False

else False

where q = mayoritario l

Comprueba que en esa lista de diagonal hay 4 en linea.

diagonal1

diagonal1 t m n j = if m>(m-(m-3))

then if rec_horizontal t m n j == True

then True

else diagonal1 t (m-1) n j

else rec_horizontal t m 3 j

Hace la comprobación de la diagonal incluyendo todas las posibles soluciones

rec_horizontal

```
rec_horizontal t m n j = if n>(n-(n-3))

                then if list_diagonal1Com q == True

                    then True

                    else rec_horizontal t m (n-1) j

                else list_diagonal1Com h

  where q = list_diagonal1 t m n j

      h = list_diagonal1 t m 3 j
```

Recorre horizontalmente el tablero y devuelve True si ha habido algún 4 en linea.

list_diagonal2

```
list_diagonal2 t m n j = a++b++c++d

  where a = [t!!(m-3)!!(n)]

      b = [t!!(m-2)!!(n-1)]

      c = [t!!(m-1)!!(n-2)]

      d = [t!!(m)!!(n-3)]
```

Devuelve una lista a partir del segundo tipo de diagonales:

{- Diagonal_2

*

*

*

* -}

list_diagonal2Com

list_diagonal2Com l = if igual12 q

then if length q >= 4 then True else False

else False

where q = mayoritario l

Comprueba si en esa diagonal hay linea.

rec_horizontal2

rec_horizontal2 t m n j = if n>(n-(n-3))

then if list_diagonal2Com q == True

then True

else rec_horizontal2 t m (n-1) j

else list_diagonal2Com h

where q = list_diagonal2 t m n j

h = list_diagonal2 t m 3 j

Recorre horizontalmente el tablero buscando 4 en linea según la segunda forma de diagonal.

diagonal2

```
diagonal2 t m n j = if m>(m-(m-3))

                    then if rec_horizontal2 t m n j == True

                         then True

                         else diagonal2 t (m-1) n j

                    else rec_horizontal2 t m 3 j
```

Hace la comprobación de la segunda diagonal en todas las posiciones posibles del tablero.

comprob

```
comprob t m n j = if q==True || p==True || h==True || f==True then True else False

  where q = compv t m n

        p = comph t m

        h = diagonal1 t m n j

        f = diagonal2 t m n j
```

Comprueba si hay cuatro en linea, de cualquiera de los jugadores y de cualquiera de las formas, ya sea tanto horizontal, vertical o diagonalmente.

empate

```
empate t 0 = fila_ocupada (t!!0)

empate t m = if fila_ocupada (t!!m) == True then empate t (m-1) else False
```

Devuelve True en caso de empate o False en caso de que no lo haya, lo hace comprobando si todas

las últimas posiciones están llenas.

fila_ocupada

fila_ocupada [] = True

fila_ocupada [x] = if x==0 then False else True

fila_ocupada (x:xs) = if x==0 then False else fila_ocupada xs

Devuelve verdadero si todas las posiciones de (x:xs) son distintas de 0.

primero

primero [] = error "lista vacia1"

primero [x] = x

primero (x:xs) = x

Devuelve el primer elemento de una lista.

Interacción de la IA

insertarIA t m 0 = if q==0

then insertar 2 t 0 m

else insreverse t m 0

where q = t!!0!!0

insertarIA t m n = if q==0

then insIA t m n --si q=0 entonces fila vacia

else insertarIA t m (n-1)

where q = t!!0!!n -- la más alta

insIA t m n = if comprob q m n [] == True

then insertar 2 t n m

else insertarIA t m (n-1)

where q = insertar 1 t n m

insreverse t m n = if q==0 then insertar 2 t n m else insreverse t m (n+1)

where q = t!!0!!n

A la hora de insertar ficha la ia llama a la función insIA que primero comprueba si el jugador hace 4 en linea en la siguiente jugada, si es así mete la ficha en esa columna, sino busca otra columna hasta llegar al final. Si hay columnas llenas, pasa a la siguiente, y en el caso de que se llegue al final y haya columna llena, hace el recorrido hacia atrás buscando una columna que esté vacía para insertar la ficha.

Función Main

main :: IO ()

main = do

args <- getArgs

let n = atoi(args!!0)

let m = atoi(args!!1)

```
if n<7 || m<6

  then ayuda

  else do

    putStrLn "Cuatro en linea"

    let t = tablero n m [] []

    if length args == 3 then jugador1' t (n-1) (m-1) else jugador1 t (n-1) (m-1)
```

Se recogen los argumentos, en el caso de que se le hayan pasado tres se llama al jugador1' que después va a llamar a la función ordenador. En el caso de que solo se le hayan pasado dos argumentos se llamará a la función jugado1, que después llamará a jugador2 para que así jueguen de manera alterna.

Jugador1

```
jjugador1 t n m = do

  print "Numero de columna:"

  let j = lista_colum [1] (n+1)

  print j

  print "------------------"

  putStrLn (pintar t)

  print "------------------"
```

```
print j

if comprob t m n [] == True

  then print "Gana jugador2"

  else do

    if empate t m == True

      then print "Empate"

      else do

        putStrLn "Jugador 1: introduce columna:"

        c <- getLine

        let d = ((atoi c)-1)

        if es_posible t m n d == True

          then do

            let z = (insertar 1 t d m)

            jugador2 z n m    --ordenador z n m

          else do

            print "Columna incorrecta"
```

```
        jugador1 t n m
```

En este modulo, primero se comprueba si ha ganado el otro jugador, después si hay empate y por ultimo ya introduce la ficha pidiendosela por teclado y le pasa el tablero al otro jugador. Si ha introducido mal la ficha se vuelve a llamar otra vez al mismo modulo.

jugador2

```
jugador2 t n m = do

  print "Numero de columna:"

  let j = lista_colum [1] (n+1)

  print j

  print "------------------"

  putStrLn (pintar t)

  print "------------------"

  print j

  if comprob t m n [] == True

       then print "Gana jugador1"

       else do

         if empate t m == True
```

```
then print "Empate"

else do

  putStrLn "Jugador 2: introduce columna:"

  c <- getLine

  let d = ((atoi c)-1)

  if es_posible t m n d == True

    then do

      let z = (insertar 2 t d m)

      jugador1 z n m

    else do

      print "Columna incorrecta"

      jugador2 t n m
```

Como el modulo de jugador 1 pero para jugador 2

ayuda

ayuda = print "Se deben pasar como argumento: < numero de columnas >= 7 > < numero de filas >= 6 >"

Imprime la ayuda de como ejecutar el programa

jugador1'

```
jugador1' t n m = do

  print "Numero de columna:"

  let j = lista_colum [1] (n+1)

  print j

  print "-----------------"

  putStrLn (pintar t)

  print "-----------------"

  print j

  if comprob t m n [] == True

    then print "Gana el Ordenador"

    else do

      if empate t m == True

        then print "Empate"

        else do

            putStrLn "Jugador 1: introduce columna:"
```

```
    c <- getLine

    let d = ((atoi c)-1)

    if es_posible t m n d == True

      then do

        let z = (insertar 1 t d m)

        ordenador z n m

      else do

        print "Columna incorrecta"

        jugador1' t n m
```

Como el modulo de jugador1 pero a este modulo lo va a llamar y va a llamar luego a ordenador, alternadamente.

ordenador

```
ordenador t n m = do

  print "Numero de columna:"

  let j = lista_colum [1] (n+1)

  print j

  print "------------------"
```

```
  putStrLn (pintar t)

  print "-----------------"

  print j

  if comprob t m n [] == True

    then print "Gana jugador1"

    else do

        if empate t m == True

          then print "Empate"

          else do

            putStrLn "Ordenador: introduce columna:"

            let z = insertarIA t m n

            jugador1' z n m

            jugador1' z n m
```

Modulo del ordenador