PRACTICA PECL2 JESÚS LUQUE MILLÓN

Y

PABLO JARAMILLO DOMINGUEZ

ANTIQUEBLOCKS

**object** AntiqueBlocks {

**class** Bloque (**val** pos:Int,**val** color:Int)

//funcion para pintar indicadores al tablero

**def** pintar\_flechas\_columnas(dificultad:Int) {

**if**(dificultad==1){

println("C O L U M N A S\n")

println("0 1 2 3 4 5 6 7 8 9 10")

println("| | | | | | | | | | |\n")

}

**else**{

**if**(dificultad==2){

println("C O L U M N A S\n")

println("0 1 2 3 4 5 6 7 8 9 101112131415")

println("| | | | | | | | | | | | | | | |\n")

}

**else**{

println("C O L U M N A S\n")

println("0 1 2 3 4 5 6 7 8 9 1011121314")

println("| | | | | | | | | | | | | | |\n")

}

}

}

//funcion parra convertir cada numero a una letra

**def** convertir\_a\_colores(valor:Int):String = {

**if** (valor==1)

**return** "A"

**else** **if** (valor==2)

**return** "R"

**else** **if** (valor==3)

**return** "N"

**else** **if** (valor==4)

**return** "V"

**else** **if** (valor==5)

**return** "P"

**else** **if** (valor==6)

**return** "M"

**else** **if** (valor==7)

**return** "G"

**else** **if** (valor==8)

**return** "\*"

**else** **return** " "

}

//funcion que imprime el tablero

**def** print\_tablero(dificultad:Int, tablero:List[Bloque], dimensiones :(Int,Int)) {

**if** (!tablero.isEmpty) {

**if** (tablero.length==dimensiones.\_1\*dimensiones.\_2)

pintar\_flechas\_columnas(dificultad)

print(convertir\_a\_colores(tablero.head.color) + " ")

**if** ((tablero.head.pos + 1) % dimensiones.\_2 == 0) {

print(" - " + ((tablero.head.pos + 1) / dimensiones.\_2 - 1))

**if** ((tablero.head.pos + 1) / dimensiones.\_2==1) println(" F")

**if** ((tablero.head.pos + 1) / dimensiones.\_2==2) println(" I")

**if** ((tablero.head.pos + 1) / dimensiones.\_2==3) println(" L")

**if** ((tablero.head.pos + 1) / dimensiones.\_2==4) println(" A")

**if** ((tablero.head.pos + 1) / dimensiones.\_2==5) println(" S")

**if** ((tablero.head.pos + 1) / dimensiones.\_2>5) println()

}

print\_tablero(dificultad,tablero.tail, dimensiones)

}

}

//funcion para recorrer el tablero e insertar los bloques a zero que se pasan en la lista como parametro

**def** recorrerLista(tablero:List[Bloque], zeros:List[Bloque]):List[Bloque]={

**if**(!zeros.isEmpty){

**val** hueco = **new** Bloque(zeros.head.pos, 0)

**val** tablero2 = ins\_bloque(hueco, hueco.pos, tablero)

**return** recorrerLista(tablero2, zeros.tail)

}

**return** tablero

}

//funcion que comprueba una lista e inserta un bloque solo en caso de que no se encuentre dentro

**def** ins\_bloque\_no\_repetido(b:Bloque,lista:List[Bloque]):List[Bloque]={

**if**(lista.length==1){

**if**(lista == Nil)

**return** b::lista

**if**(lista.head.pos!=b.pos)**return** b::lista

**else** **return** lista

}

**else**

**if**(b.pos!=lista.head.pos)lista.head::ins\_bloque\_no\_repetido(b,lista.tail)

**else** **return** lista

}

//funcion que crea un tablero dadas unas diimensiones

**def** crear\_tablero(pos:Int, colores:Int, dimensiones:(Int,Int)): List[Bloque] = {

**if** (pos==(dimensiones.\_1\*dimensiones.\_2)-1)

generar\_bloque(pos,colores)::Nil

**else**

generar\_bloque(pos,colores)::crear\_tablero(pos+1,colores, dimensiones)

}

//Crea un bloque con valor aleatorio

**def** generar\_bloque(pos:Int, dificultad:Int):Bloque = {

**val** rnd = scala.util.Random

**return** **new** Bloque (pos,rnd.nextInt(dificultad)+1)

}

//Pone bombas en un tablero dependiendo de la dificultad

**def** ponerBombas(dificultad:Int, tablero:List[Bloque]):List[Bloque]={

**if**(dificultad==1){

**val** rnd1 = scala.util.Random

**val** rnd2 = scala.util.Random

**val** bomba1= **new** Bloque(rnd1.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero1= ins\_bloque(bomba1,bomba1.pos,tablero)

**val** bomba2= **new** Bloque(rnd2.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero2= ins\_bloque(bomba2,bomba2.pos,tablero1)

**return** tablero2

}

**else**{

**if**(dificultad==2){

**val** rnd1 = scala.util.Random

**val** rnd2 = scala.util.Random

**val** rnd3 = scala.util.Random

**val** bomba1= **new** Bloque(rnd1.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero1= ins\_bloque(bomba1,bomba1.pos,tablero)

**val** bomba2= **new** Bloque(rnd2.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero2= ins\_bloque(bomba2,bomba2.pos,tablero1)

**val** bomba3= **new** Bloque(rnd3.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero3= ins\_bloque(bomba3,bomba3.pos,tablero2)

**return** tablero3

}

**else**{

**val** rnd1 = scala.util.Random

**val** rnd2 = scala.util.Random

**val** rnd3 = scala.util.Random

**val** rnd4 = scala.util.Random

**val** rnd5 = scala.util.Random

**val** bomba1= **new** Bloque(rnd1.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero1= ins\_bloque(bomba1,bomba1.pos,tablero)

**val** bomba2= **new** Bloque(rnd2.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero2= ins\_bloque(bomba2,bomba2.pos,tablero1)

**val** bomba3= **new** Bloque(rnd3.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero3= ins\_bloque(bomba3,bomba3.pos,tablero2)

**val** bomba4= **new** Bloque(rnd4.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero4= ins\_bloque(bomba4,bomba4.pos,tablero3)

**val** bomba5= **new** Bloque(rnd5.nextInt(filas\_tablero(dificultad)\*columnas\_tablero(dificultad))+1,8)

**val** tablero5= ins\_bloque(bomba5,bomba5.pos,tablero4)

**return** tablero5

}

}

}

//funcion para obtener la siguiente jugada del jugador

**def** movimiento\_jugador(tablero:List[Bloque], dimensiones :(Int, Int)): Int = {

print("\n\n\n INTRODUCIR POSICION\n Fila: ")

**val** fila=~~readInt~~

print(" Columna: ")

**val** columna=~~readInt~~

println()

**if**(fila>dimensiones.\_1||fila<0||columna>dimensiones.\_2||columna<0){

print(" POSICION INVALIDA ")

**return** movimiento\_jugador(tablero, dimensiones)

}

**return** (fila\*dimensiones.\_2+columna)

}

//funcion que comprueba si una posicion dada es valida

**def** validar\_posicion(tablero:List[Bloque], fila:Int, columna:Int, dimensiones: (Int, Int)): Boolean = {

**if**(fila>=dimensiones.\_1||fila<0||columna>=dimensiones.\_2||columna<0)

**return** **false**;

**return** **true**;

}

//funcion para obtener un bloque del tablero dada su posicion

**def** get\_elem(tablero:List[Bloque], pos:Int):Bloque = {

**if**(tablero.head.pos == pos){

**return** tablero.head

}

get\_elem(tablero.tail, pos)

}

//funcion para comprobar los bloques adyacentes del mismo color

**def** comprobarAdyacentes2(tablero:List[Bloque], posi:(Int, Int), listaBorrar:List[Bloque], color:Int, dimensiones:(Int, Int)):List[Bloque] = {

**val** fila = posi.\_1

**val** columna = posi.\_2

**val** posicion = fila\*dimensiones.\_2 + columna

**if**(!validar\_posicion(tablero, fila, columna, dimensiones))

**return** listaBorrar

**val** centro = get\_elem(tablero, posicion);

**if**(centro.color == 0)

**return** listaBorrar

**if**(centro.color != color)

**return** listaBorrar

**val** milista = ins\_no\_repetidos(centro, listaBorrar)

**if**(milista == listaBorrar)

**return** milista

**else**{

**val** list1 = comprobarAdyacentes2(tablero, (fila+1, columna), milista, color, dimensiones)

**val** list2 = comprobarAdyacentes2(tablero, (fila-1, columna), list1, color, dimensiones)

**val** list3 = comprobarAdyacentes2(tablero, (fila, columna+1), list2, color, dimensiones)

**val** list4 = comprobarAdyacentes2(tablero, (fila, columna-1), list3, color, dimensiones)

**if**(milista != list4){

**return** list4

}

**return** milista

}

}

**def** generar\_listas(b:Bloque):List[Bloque]={

**val** lista=List(b)

**return** lista

}

//Inserta en una lista elementos si no estan repetidos

**def** ins\_no\_repetidos(b:Bloque,lista:List[Bloque]):List[Bloque]={

**if**(lista.length==1 || lista.isEmpty){

**if**(lista.isEmpty) {

**return** generar\_listas(b)

}

**if**(lista.head.pos!=b.pos)**return** b::lista

**else** **return** lista

}

**else**

**if**(b.pos!=lista.head.pos)lista.head::ins\_no\_repetidos(b,lista.tail)

**else** **return** lista

}

//insertar bloque en una posicion

**def** ins\_bloque(b:Bloque,pos:Int,lista:List[Bloque]) : List[Bloque] ={

**if**(pos==0){b::lista.tail}

**else** lista.head::ins\_bloque(b,pos-1,lista.tail)

}

//funcion que obtiene el mejor moovimiento posible para un tablero dado

**def** optimo(tableroEnt:List[Bloque], tablero:List[Bloque], posiPunt:(Int,Int), dimensiones: (Int,Int) ) : (Int,Int) = {

**if**(!tablero.isEmpty){

**val** listaPrueba:List[Bloque] = Nil

**val** fila = tablero.head.pos / dimensiones.\_2;

**val** columna = tablero.head.pos % dimensiones.\_2;

**val** puntos = comprobarAdyacentes2(tableroEnt, (fila, columna), listaPrueba, tablero.head.color, dimensiones).length

**if**(puntos > posiPunt.\_2)

optimo(tableroEnt, tablero.tail, (tablero.head.pos, puntos), dimensiones)

**else**

optimo(tableroEnt, tablero.tail, posiPunt, dimensiones)

}

**else**

**return** posiPunt

}

//decrementa una unidad

**def** dec(n: Int) = n - 1

//Recorre el array en busca de ceros

**def** bucleFor(longTablero:Int,l:List[Bloque],aux:List[Bloque],filas:Int,columnas:Int):List[Bloque]={

**if**(longTablero==0){

**if**(devolver\_bloque\_lista(longTablero,l).color==0){

**val** aux2=subir\_ceros(longTablero,longTablero-columnas,aux,filas,columnas)

**return** aux2

}

**else** **return** aux

}

**else** {

**if**(devolver\_bloque\_lista(longTablero,l).color==0){

**val** aux2=subir\_ceros(longTablero,longTablero-columnas,aux,filas,columnas)

bucleFor(dec(longTablero),l,aux2,filas,columnas)

}

**else** bucleFor(dec(longTablero),aux,aux,filas,columnas)}

}

//Enocontrar0

**def** encontrarCero(posicion:Int,l:List[Bloque],columnas:Int):Int={

**if**(posicion<0 || devolver\_bloque\_lista(posicion,l).color==0){**return** posicion}

**else** encontrarCero(posicion-columnas,l,columnas)

}

//Devuelve un bloque que esta en una posicion (bien)

**def** devolver\_bloque\_lista(posicion:Int,l:List[Bloque]):Bloque={

**if**(l.head.pos==posicion){**return** l.head}

**else** devolver\_bloque\_lista(posicion,l.tail)

}

//Devuelve el bloque que esta encima de una posicion

**def** encontrar\_arriba(l:List[Bloque],pos:Int,columnas:Int):Bloque={

**if** (devolver\_bloque\_lista((pos-columnas),l).color!=0){**return** devolver\_bloque\_lista(pos-columnas,l)}

**else** {

**if**(pos-columnas>=0){encontrar\_arriba(l,pos-columnas,columnas) }

**else** **return** devolver\_bloque\_lista(pos,l)

}

}

//Mueve los ceros hacia arriba

**def** subir\_ceros(pos0:Int,posIntercambio:Int,l:List[Bloque],filas:Int,columnas:Int):List[Bloque]={

**if**((posIntercambio+columnas)-columnas<0) **return** l

**else** {

**if**(devolver\_bloque\_lista(posIntercambio,l).color!=0){

**val** bloqAux1= devolver\_bloque\_lista(posIntercambio,l)

**val** bloqAux2= **new** Bloque(encontrarCero(pos0,l,columnas),0)

**val** bloqAux12=**new** Bloque(bloqAux1.pos,bloqAux2.color)

**val** bloqAux21=**new** Bloque(bloqAux2.pos,bloqAux1.color)

//bajar el bloque k no es 0

**val** aux=ins\_bloque(bloqAux12,bloqAux1.pos,l)

//poner el 0

**val** x=ins\_bloque(bloqAux21,bloqAux2.pos,aux)

subir\_ceros(pos0-columnas,posIntercambio,x,filas,columnas)//-columnas

}

**else** {

subir\_ceros(pos0,posIntercambio-columnas,l,filas,columnas)//pos0

}

}

}

//Encuentra columnas 0 distintas de cero de una 'matriz'

**def** encontrar\_columna\_distinta\_cero(pos:Int,limite:Int,l:List[Bloque],columnas:Int):Int={

**if**(pos<l.length-columnas){**return** 0}

**else**{

**if** (pos!=l.length && devolver\_bloque\_lista(pos,l).color!=0 && devolver\_bloque\_lista(pos-1,l).color==0 && pos>limite)//&& devolver\_bloque\_lista(pos,l).color!=0

**return** pos

**else**

encontrar\_columna\_distinta\_cero(pos-1,limite,l,columnas)

}

}

//Cambia 2 columnas de una 'matriz'

**def** bucle\_cambiar\_columnas(pos0:Int,posIntercambio:Int,l:List[Bloque],columnas:Int):List[Bloque]={

**if**(pos0<0) **return** l

**else**{

**val** bloq1=devolver\_bloque\_lista(pos0,l)

**val** bloq2=devolver\_bloque\_lista(posIntercambio,l)

**val** bloq12= **new** Bloque(bloq1.pos,bloq2.color)

**val** bloq21= **new** Bloque(bloq2.pos,bloq1.color)

**val** aux=ins\_bloque(bloq12,bloq12.pos,l)

**val** x=ins\_bloque(bloq21,bloq21.pos,aux)

bucle\_cambiar\_columnas(pos0-columnas,posIntercambio-columnas,x,columnas)

}

}

//Funcion para cambiar columnas

**def** desplazar\_derecha(pos:Int,l:List[Bloque],columnas:Int):List[Bloque]={

**if**(pos<(l.length-columnas)) **return** l

**else** {

**if** (pos<l.length-1){

**if**(devolver\_bloque\_lista(pos,l).color==0 && devolver\_bloque\_lista(pos+1,l).color!=0){

**val** pos0=devolver\_bloque\_lista(pos,l)

**val** posIntercambio=encontrar\_columna\_distinta\_cero(l.length-1,pos,l,columnas)

**if**(posIntercambio==0){**return** l}

**val** aux= bucle\_cambiar\_columnas(pos,posIntercambio,l,columnas)

desplazar\_derecha(pos-1,aux,columnas)

}

**else** desplazar\_derecha(pos-1,l,columnas)

}

**else** desplazar\_derecha(pos-1,l,columnas)

}

}

//Sirve para llamar a la funcion desplazar derecha tantas veces como columnas hay

**def** repetir\_mov\_ceros (num:Int,l:Int,lista:List[Bloque],columnas:Int,cantidad\_ceros:Int):List[Bloque]={

**if**(num==columnas)**return** lista

**else** {

**val** x=desplazar\_derecha(lista.length,lista,columnas)

repetir\_mov\_ceros(num+1,x.length,x,columnas,cantidad\_ceros)

}

}

//Devuelve la cantidad de ceros que tiene la ultima fila de una matriz

**def** encontrar\_ceros\_ultima\_fila(pos:Int,lista:List[Bloque],columnas:Int,cantidad:Int):Int={

**if**(pos<(lista.length-columnas)){

**return** cantidad

}

**else**

**if**(devolver\_bloque\_lista(pos,lista).color==0)

encontrar\_ceros\_ultima\_fila(pos-1,lista,columnas,cantidad+1)

**else**

encontrar\_ceros\_ultima\_fila(pos-1,lista,columnas,cantidad)

}

//Explosion de bloques o bombas

**def** explotar(pos:Int,l:List[Bloque],columnas:Int,filas:Int):List[Bloque]={

**if**(devolver\_bloque\_lista(pos,l).color==8){

**val** listaAux=explotarBomba(pos,l,columnas,filas)

**return** listaAux

}

**else**{

**val** bloqueAux=**new** Bloque(pos,0)

**val** listaAux=ins\_bloque(bloqueAux,pos,l)

**return** listaAux

}

}

//Funcion que hace explotar una bomba teniendo en cuenta su posicion

**def** explotarBomba(pos:Int,l:List[Bloque], columnas:Int, filas:Int):List[Bloque]={

**val** bloqueAux=**new** Bloque(pos,0)

**val** listaAux=ins\_bloque(bloqueAux,pos,l)

**val** filaBomba=pos/columnas

**val** columnaBomba=pos%columnas

**if**(filaBomba==0){

**if**(columnaBomba==0){//derecha abajo derechaAbajo

**val** derecha = devolver\_bloque\_lista(pos+1,l)

**val** abajo = devolver\_bloque\_lista(pos+columnas,l)

**val** abajoDerecha = devolver\_bloque\_lista((pos+columnas)+1,l)

**val** lista1=explotar(derecha.pos,listaAux,columnas,filas)

**val** lista2=explotar(abajo.pos,lista1,columnas,filas)

**val** lista3=explotar(abajoDerecha.pos,lista2,columnas,filas)

**return** lista3

}

**else**{

**if**(columnaBomba==columnas-1){//izq abajo abajoIzquierda

**val** izquierda = devolver\_bloque\_lista(pos-1,l)

**val** abajo = devolver\_bloque\_lista(pos+columnas,l)

**val** abajoIzquierda =devolver\_bloque\_lista((pos+columnas)-1,l)

**val** lista1=explotar(izquierda.pos,listaAux,columnas,filas)

**val** lista2=explotar(abajo.pos,lista1,columnas,filas)

**val** lista3=explotar(abajoIzquierda.pos,lista2,columnas,filas)

**return** lista3

}

**else**{//derecha izq abajo abajoDer abajoIzq

**val** derecha = devolver\_bloque\_lista(pos+1,l)

**val** izquierda = devolver\_bloque\_lista(pos-1,l)

**val** abajo = devolver\_bloque\_lista(pos+columnas,l)

**val** abajoDerecha = devolver\_bloque\_lista((pos+columnas)+1,l)

**val** abajoIzquierda =devolver\_bloque\_lista((pos+columnas)-1,l)

**val** lista1=explotar(derecha.pos,listaAux,columnas,filas)

**val** lista2=explotar(izquierda.pos,lista1,columnas,filas)

**val** lista3=explotar(abajo.pos,lista2,columnas,filas)

**val** lista4=explotar(abajoDerecha.pos,lista3,columnas,filas)

**val** lista5=explotar(abajoIzquierda.pos,lista4,columnas,filas)

**return** lista5

}

}

}

**else**{

**if**(filaBomba==filas-1){

**if**(columnaBomba==0){//derecha arriba arribaDerecha

**val** derecha = devolver\_bloque\_lista(pos+1,l)

**val** arriba = devolver\_bloque\_lista(pos-columnas,l)

**val** arribaDerecha = devolver\_bloque\_lista((pos-columnas)+1,l)

**val** lista1=explotar(derecha.pos,listaAux,columnas,filas)

**val** lista2=explotar(arriba.pos,lista1,columnas,filas)

**val** lista3=explotar(arribaDerecha.pos,lista2,columnas,filas)

**return** lista3

}

**else**{//izquierda arriba arribaIzquierda

**if**(columnaBomba==columnas-1){

**val** izquierda = devolver\_bloque\_lista(pos-1,l)

**val** arriba = devolver\_bloque\_lista(pos-columnas,l)

**val** arribaIzquierda = devolver\_bloque\_lista((pos-columnas)-1,l)

**val** lista1=explotar(izquierda.pos,listaAux,columnas,filas)

**val** lista2=explotar(arriba.pos,lista1,columnas,filas)

**val** lista3=explotar(arribaIzquierda.pos,lista2,columnas,filas)

**return** lista3

}

**else**{//derecha izq arriba arribaDerecha aribaIzquierda

**val** derecha = devolver\_bloque\_lista(pos+1,l)

**val** izquierda = devolver\_bloque\_lista(pos-1,l)

**val** arriba = devolver\_bloque\_lista(pos-columnas,l)

**val** arribaDerecha = devolver\_bloque\_lista((pos-columnas)+1,l)

**val** arribaIzquierda = devolver\_bloque\_lista((pos-columnas)-1,l)

**val** lista1=explotar(derecha.pos,listaAux,columnas,filas)

**val** lista2=explotar(izquierda.pos,lista1,columnas,filas)

**val** lista3=explotar(arriba.pos,lista2,columnas,filas)

**val** lista4=explotar(arribaIzquierda.pos,lista3,columnas,filas)

**val** lista5=explotar(arribaDerecha.pos,lista4,columnas,filas)

**return** lista5

}

}

}

**else**{

**if**(columnaBomba==0 || columnaBomba==columnas-1){

**if**(columnaBomba==0){//derecha abajo arriba arribaDerecha abajoDerecha

**val** derecha = devolver\_bloque\_lista(pos+1,l)

**val** abajo = devolver\_bloque\_lista(pos+columnas,l)

**val** arriba = devolver\_bloque\_lista(pos-columnas,l)

**val** arribaDerecha = devolver\_bloque\_lista((pos-columnas)+1,l)

**val** abajoDerecha = devolver\_bloque\_lista((pos+columnas)+1,l)

**val** lista1=explotar(derecha.pos,listaAux,columnas,filas)

**val** lista2=explotar(abajo.pos,lista1,columnas,filas)

**val** lista3=explotar(arriba.pos,lista2,columnas,filas)

**val** lista4=explotar(arribaDerecha.pos,lista3,columnas,filas)

**val** lista5=explotar(abajoDerecha.pos,lista4,columnas,filas)

**return** lista5

}

**else**{//izquierda abajo arriba abajoIzquierda arribaIzquierda

**val** izquierda = devolver\_bloque\_lista(pos-1,l)

**val** abajo = devolver\_bloque\_lista(pos+columnas,l)

**val** arriba = devolver\_bloque\_lista(pos-columnas,l)

**val** arribaIzquierda = devolver\_bloque\_lista((pos-columnas)-1,l)

**val** abajoIzquierda = devolver\_bloque\_lista((pos+columnas)-1,l)

**val** lista1=explotar(izquierda.pos,listaAux,columnas,filas)

**val** lista2=explotar(abajo.pos,lista1,columnas,filas)

**val** lista3=explotar(arriba.pos,lista2,columnas,filas)

**val** lista4=explotar(arribaIzquierda.pos,lista3,columnas,filas)

**val** lista5=explotar(abajoIzquierda.pos,lista4,columnas,filas)

**return** lista5

}

}

**else**{//arriba abajo der izq

**val** arriba = devolver\_bloque\_lista(pos-columnas,l)

**val** abajo = devolver\_bloque\_lista(pos+columnas,l)

**val** derecha = devolver\_bloque\_lista(pos+1,l)

**val** izquierda = devolver\_bloque\_lista(pos-1,l)

**val** arribaIzquierda = devolver\_bloque\_lista((pos-columnas)-1,l)

**val** abajoIzquierda = devolver\_bloque\_lista((pos+columnas)-1,l)

**val** arribaDerecha = devolver\_bloque\_lista((pos-columnas)+1,l)

**val** abajoDerecha = devolver\_bloque\_lista((pos+columnas)+1,l)

**val** lista1=explotar(derecha.pos,listaAux,columnas,filas)

**val** lista2=explotar(izquierda.pos,lista1,columnas,filas)

**val** lista3=explotar(abajo.pos,lista2,columnas,filas)

**val** lista4=explotar(arriba.pos,lista3,columnas,filas)

**val** lista5=explotar(arribaIzquierda.pos,lista4,columnas,filas)

**val** lista6=explotar(arribaDerecha.pos,lista5,columnas,filas)

**val** lista7=explotar(abajoDerecha.pos,lista6,columnas,filas)

**val** lista8=explotar(abajoIzquierda.pos,lista7,columnas,filas)

**return** lista8

}

}

}

}

//Calcula la cantida de ceros en un array

**def** calcular\_ceros(lista:List[Bloque],pos:Int,cantidad:Int):Int={

**if**(pos==lista.length-1){**return** cantidad}

**else**{

**if**(devolver\_bloque\_lista(pos,lista).color==0) calcular\_ceros(lista,pos+1,cantidad+1)

**else** calcular\_ceros(lista,pos+1,cantidad)

}

}

//Calcula la puntuacion conseguida en un movimiento

**def** calcular\_puntuacion(lista1:List[Bloque], lista2:List[Bloque]):Int={

**return** (calcular\_ceros(lista1,0,0)-calcular\_ceros(lista2,0,0))\*10

}

//funcion que implementa la dinamica del juego

**def** bucle\_juego(vidas:Int,puntuacion:Int,dificultad:Int,tablero:List[Bloque],filas:Int,columnas:Int):Int={

println(" ")

println("-------------------")

println(" - VIDAS - "+vidas)

println("- PUNTUACION - "+puntuacion)

println("-------------------")

println(" ")

**val** aux=puntuacion

**val** auxVidas=vidas

print\_tablero(dificultad, tablero, (filas,columnas))

**if**(auxVidas == 0){

println(" ")

println("-------------------")

println(" - VIDAS - "+vidas)

println("- PUNTUACION - "+puntuacion)

println("-------------------")

println(" ")

**return** -99

}

**if**(calcular\_ceros(tablero,0,0) == tablero.length-1){

println(" ")

println("-------------------")

println(" - VIDAS - "+vidas)

println("- PUNTUACION - "+puntuacion)

println("-------------------")

println(" ")

**return** 0

}

**val** mejorjugada = optimo(tablero, tablero, (0,0), (filas, columnas))

println("")

println("Mejor posicion:")

println(" Fila: " + (mejorjugada.\_1/ columnas))

println(" Columna: " + (mejorjugada.\_1 % columnas))

println(" Puntuacion: " + mejorjugada.\_2\*10)

**val** pos = movimiento\_jugador(tablero, (filas, columnas))

**val** tama = tablero.length

**val** listaPrueba:List[Bloque] = Nil

**val** bloqueprueba = get\_elem(tablero, pos)

**val** fila = pos / columnas;

**val** columna = pos % columnas;

**if**(devolver\_bloque\_lista(pos,tablero).color==8){

**val** tablero2= explotarBomba(pos,tablero,columnas,filas)

**val** tablero3 = bucleFor((filas\*columnas)-1,tablero2,tablero2,filas,columnas)

**val** cantidad\_ceros=encontrar\_ceros\_ultima\_fila(tablero3.length-1,tablero3,columnas,0)

**val** tablero4 = repetir\_mov\_ceros(0,tablero3.length-1,tablero3,columnas,cantidad\_ceros)

**val** puntuacion\_jugador=aux + calcular\_puntuacion(tablero4,tablero)

**val** puntuacion2 = puntuacion + bucle\_juego(vidas,puntuacion\_jugador,dificultad,tablero4,filas,columnas)

**return** puntuacion2

}

**else**{

**val** aBorrar = comprobarAdyacentes2(tablero, (fila, columna), listaPrueba, bloqueprueba.color, (filas, columnas))

**val** puntuacion = aBorrar.length

**val** tablero2 = recorrerLista(tablero, aBorrar)

**val** tablero3 = bucleFor((filas\*columnas)-1,tablero2,tablero2,filas,columnas)

**val** cantidad\_ceros=encontrar\_ceros\_ultima\_fila(tablero3.length-1,tablero3,columnas,0)

**val** tablero4 = repetir\_mov\_ceros(0,tablero3.length-1,tablero3,columnas,cantidad\_ceros)

**val** puntuacion\_jugador=aux + calcular\_puntuacion(tablero4,tablero)

**if**(puntuacion<3)

**return** puntuacion + bucle\_juego(vidas-1,puntuacion\_jugador,dificultad,tablero4,filas,columnas)

**else**

**return** puntuacion + bucle\_juego(vidas,puntuacion\_jugador,dificultad,tablero4,filas,columnas)

}

}

//Devuelve el numero de filas dependiendo d la dificultad

**def** filas\_tablero(dificultad:Int):Int={

**if**(dificultad==1) **return** 9

**else**{

**if**(dificultad==2) **return** 12

**else** **return** 25

}

}

//Devulve el numero de columnas dependiendo d la dificultad

**def** columnas\_tablero(dificultad:Int):Int={

**if**(dificultad==1) **return** 11

**else**{

**if**(dificultad==2) **return** 16

**else** **return** 15

}

}

//Devuelve el numero de vidas dependiendo de la dificultad

**def** vidas\_jugador(dificultad:Int):Int={

**if**(dificultad==1) **return** 8

**else**{

**if**(dificultad==2) **return** 10

**else** **return** 15

}

}

//Devuelve el numero de colores dependiendo de la dificultad

**def** colores\_tablero(dificultad:Int):Int={

**if**(dificultad==1) **return** 3

**else**{

**if**(dificultad==2) **return** 5

**else** **return** 7

}

}

**def** main(args: Array[String]) {

println("Introduzca la dificultad: ")

**val** dificultad=~~readInt~~

**val** filas=filas\_tablero(dificultad)

**val** columnas=columnas\_tablero(dificultad)

**val** colores=colores\_tablero(dificultad)

**val** vidas=vidas\_jugador(dificultad)

**val** puntuacion=0

**val** tablero0 = crear\_tablero(0,colores, (filas,columnas))

**val** tablero1 = ponerBombas(dificultad,tablero0)

**val** finalJuego = bucle\_juego(vidas,puntuacion,dificultad,tablero1,filas,columnas)

**if**(finalJuego == -98){

println(" ")

println("-------------------")

println("- FIN DEL JUEGO -")

println("- HAS PERDIDO -")

println("-------------------")

println(" ")

}

**else**{

println(" ")

println("-------------------")

println("- FIN DEL JUEGO -")

println("- HAS GANADO -")

println("-------------------")

println(" ")

}

}

}