BLOCK TOY BLAST SCALA

AMPLIACIÓN DE PROGRAMACIÓN AVANZADA. PECL2

Ángel Martín Segura

Rubén Fernández Lopesino

Memoria completa sobre el desarrollo del juego pedido en la segunda práctica del curso 2017-2018

Contenido

[Descripción general del trabajo 2](#_Toc511941238)

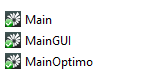
[Métodos principales 2](#_Toc511941239)

[Código de la práctica 4](#_Toc511941240)

# Descripción general del trabajo

El trabajo consiste en implementar el juego “Block Toy Blast” en Scala, ya sea manejado por un jugado o por la propia máquina.

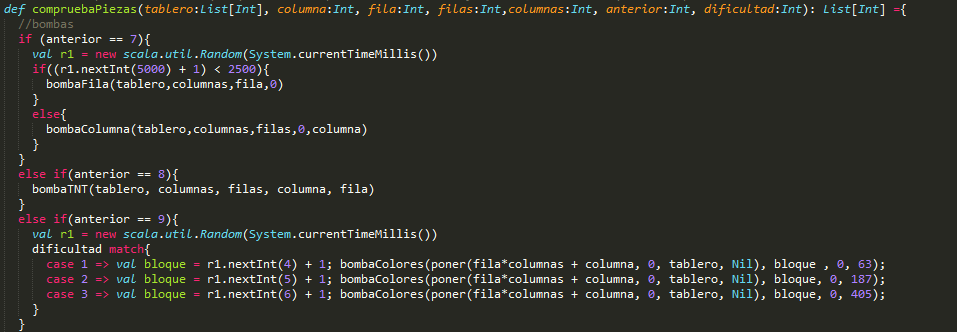
En nuestro caso, presentamos las tres fases de la práctica: el desarrollo obligatorio, el desarrollo de optimización y el desarrollo avanzado.

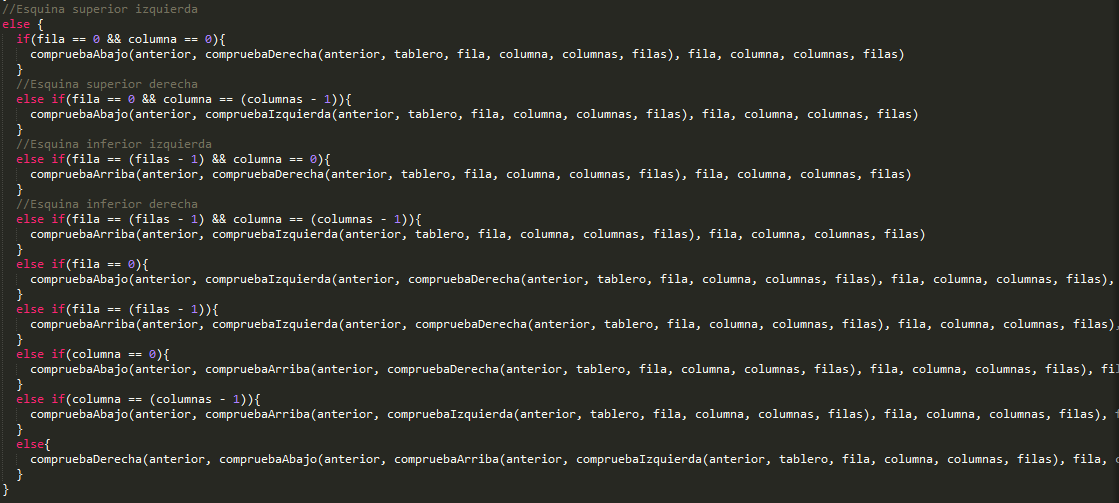


# Métodos principales

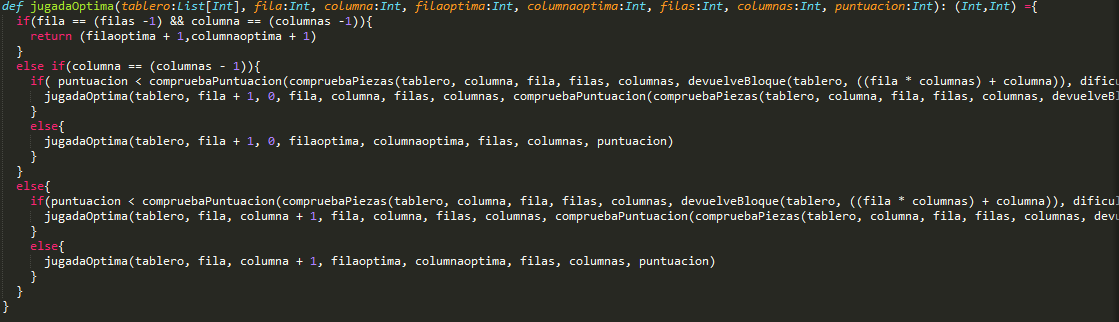
A continuación, vamos a presentar y explicar los métodos principales de la práctica. Dado que el código se incluirá al final de la memoria, sólo adjuntaremos unas capturas de la parte principal de los métodos.

El primero va a ser *“compruebaPiezas”*. Este método se encarga de llamar a los metodos *“compruebaAbajo”*, *“compruebaArriba”*, *“compruebaDerecha”* y *“compruebaIzquierda”* en función de la posición en la que nos encontremos. A su vez comprueba si el bloque que se le ha pasado es una bomba y llama a la función que corresponda en cada caso.

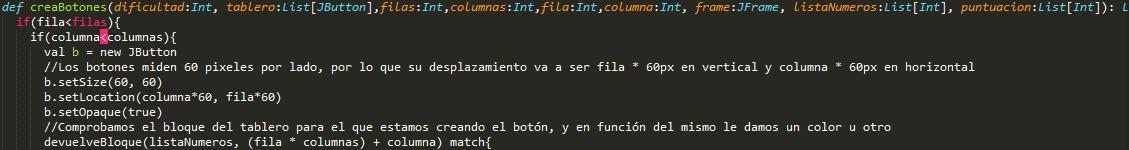




Otro de los métodos más importantes es el de *“jugadaOptima”*. Adquiere importancia en la parte del desarrollo de optimización. Esta jugada recorre todo el tablero comprobando en qué posición tenemos la jugada que más bloques elimina.



Por último, encontramos el método que se encargará de crear los botones para la interfaz, *“creaBotones”*. Esta es una de las partes más importantes de la fase del desarrollo avanzado, dado que afecta a la parte de la interfaz. Se va a encargar de crear el listado de los botones en función de la dificultad.



# Código de la práctica

* Main:

import scala.util.Random

import Console.\_

import scala.sys.process.\_

object Main extends App {

//Este método pone el elemento n en la posicion pos de la lista l

def poner(pos:Int,n:Int,l:List[Int],laux:List[Int]):List[Int]={

if(pos==0) laux:::List[Int](n):::l.tail

else poner(pos-1,n,l.tail,laux:::List[Int](l.head))

}

//Este método se utiliza en subirCerosAux, pone la ficha de la pos1 en la pos2 y viceversa

def intercambiarCasilla(pos1: Int, pos2: Int, lista: List[Int]): List[Int] = {

//Dado que hemos intentado ahorrar todos los val posible, vamos dejar comentado el código de manera más extensa para que se entienda mejor

/\*

val ficha1 = devuelveBloque(lista, pos1)

val ficha2 = devuelveBloque(lista, pos2)

val tableroAux = poner(pos1, ficha2, lista, Nil)

poner(pos2,ficha1,tableroAux,Nil)

Lo que hemos hecho ha sido ir sustituyendo la parte derecha de la igualacion en las llamadas a funciones que las prosiguen, en este caso quedaría así paso a paso:

1) Dado que ficha1 = devuelveBloque(lista,pos1) -> en poner(pos2, ficha1,tableroAux,Nil) sustituimos, quedando así -> poner(pos2,devuelveBloque(lista, pos1),tableroAux,Nil)

2) tableroAux = poner(pos1, ficha2, lista, Nil) -> sustuituimos ficha2 por su igualación -> poner(pos1, devuelveBloque(lista, pos2), lista, Nil)

3) Por ultimo, en poner(pos2,devuelveBloque(lista, pos1),tableroAux,Nil), sustituimos tableroAux por lo de arriba -> poner(pos1, devuelveBloque(lista, pos2), lista, Nil)

-> poner(pos2,devuelveBloque(lista, pos1),poner(pos1, devuelveBloque(lista, pos2), lista, Nil) ,Nil)

\*/

poner(pos2,devuelveBloque(lista,pos1),poner(pos1,devuelveBloque(lista,pos2),lista,Nil),Nil)

}

def subirCeros(pos:Int,col:Int,l:List[Int]):List[Int]={

if (pos==0) l

else if (devuelveBloque(l, pos) == 0) subirCerosAux(pos,pos-col,col,l)

else subirCeros(pos-1,col,l)

}

def subirCerosAux(pos:Int,pos2:Int,col:Int,l:List[Int]):List[Int]={

if(pos<=col || pos2<1)subirCeros(pos-1,col,l)

else if(devuelveBloque(l, pos2) == 0) subirCerosAux(pos,pos2-col,col,l)

else subirCeros(pos-1,col,intercambiarCasilla(pos,pos2,l))

}

//Aquí controlamos que en el menú inicial se seleccione un valor entre 1 y 4 para no tener problemas a la hora de llamar a los niveles

def controlDificultad(): Int ={

println("La opción elegida no es correcta, por favor seleccione un valor del 1 al 4")

println("1 - Fácil \n2 - Medio \n3 - Díficil \n4 - Salir")

val eleccion = Integer.parseInt(Console.in.readLine())

//Si la elección no está entre los valores que queremos, volvemos a llamar al mismo metodo para que vuelva a pedir la dificultad

if(eleccion < 1 || eleccion > 4){

controlDificultad()

}

else{

//Si está dentro de los límites que nos interesan, devolvemos el int

eleccion

}

}

//En este método se elige la fila, y se comprueba que está dentro de unos rangos

def elegirFila(filas:Int): Int = {

val fila = Integer.parseInt(Console.in.readLine())

if(fila < 1 || fila > 7){

//Si se sale del rango que nos interesa, llamamos a controlFilas

controlFilas(filas)

}

else{

//Si no, devolvemos la fila elegida

fila

}

}

def elegirColumna(columnas:Int): Int = {

val columna = Integer.parseInt(Console.in.readLine())

if(columna < 1 || columna > 7){

controlColumnas(columnas)

}

else{

columna

}

}

def controlFilas(filas:Int): Int ={

println("La fila elegida no es válida, introduzca un valor entre 1 y " + filas)

val eleccion = Integer.parseInt(Console.in.readLine())

if(eleccion < 1 || eleccion > filas){

controlFilas(filas)

}

else{

eleccion

}

}

def controlColumnas(columnas:Int): Int ={

println("La columna elegida no es válida, introduzca un valor entre 1 y " + columnas)

val eleccion = Integer.parseInt(Console.in.readLine())

if(eleccion < 1 || eleccion > columnas){

controlFilas(columnas)

}

else{

eleccion

}

}

def imprimirMenu(): Int ={

//Este es el método que imprime el menú inicial para seleccionar la dificultad

println("+-+- BIENVENIDO A TOYBLAST -+-+")

println("Seleccione la dificultad del juego")

println("1 - Fácil \n2 - Medio \n3 - Díficil \n4 - Salir")

val dificultad = Integer.parseInt(Console.in.readLine())

//Si los valores no están entre las opciones que nos interesan, llamamos al metodo explicado arriba para controlar el entero recibido

if(dificultad < 1 || dificultad > 4){

controlDificultad()

}

else{

dificultad

}

}

def terminarJuego(): Unit ={

//Funcion que sale del juego devolviendo error si pulsamos 4 en el menu inicial

println("Saliendo del juego...\n")

System.exit(-1)

}

//A este método se le llama desde imprimir tablero para que sea más fácil localizar la columna en la que queremos jugar

def imprimirColumnas(columnas:Int): Unit = {

if(columnas == 9){

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

println(" | | | | | | | | | ")

}

else if(columnas == 17){

println(" 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17")

println(" | | | | | | | | | | | | | | | | |")

}

else if(columnas == 27){

println(" 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27")

println(" | | | | | | | | | | | | | | | | | | | | | | | | | | |")

}

}

//Metodo para imprimir una letra en funcion del bloque o bomba que se encuentre en el tablero

def imprimirLetra(bloque:Int): String ={

bloque match{

case 1 => return "A"

case 2 => return "R"

case 3 => return "N"

case 4 => return "V"

case 5 => return "P"

case 6 => return "M"

case 7 => return "a"

case 8 => return "b"

case 9 => return "c"

case 0 => return "X"

}

}

def imprimirTablero(columnas:Int, filas:Int, fila:Int, columna:Int, tablero:List[Int]): Unit = {

//Si ya hemos imprimido todo hacemos un salto de línea para que no quede todo pegado

if(tablero.isEmpty) println("")

else if(columna == 0){

//Si estamos en la primera fila y en la primera columna, llamamos al método imprimir columnas de más arriba para la guía

if(fila == 1){

imprimirColumnas(columnas)

}

//Para que todas las columnas queden a la misma altura, controlo el número de fila e imprimo un guión menos, ya que si no a partir de la 9 quedan desplazadas a la derecha

if(fila > 9){

//Imprimimos el numero de fila y aumentamos la columna para acceder al elemento siguiente, esto solo pasa si columna es == 0

print(fila + "---- " + imprimirLetra(tablero.head))

imprimirTablero(columnas, filas, fila, columna + 1, tablero.tail)

}

else{

//Imprimimos el numero de fila y aumentamos la columna para acceder al elemento siguiente, esto solo pasa si columna es == 0

print(fila + "----- " + imprimirLetra(tablero.head))

imprimirTablero(columnas, filas, fila, columna + 1, tablero.tail)

}

}

else if(columna < columnas){

//Si columna es distinto de 0 y columna es < que el número de columnas, solo se imprime la cabeza de la lista y se llama de nuevo a imprimir tablero con columna + 1

print(" " + imprimirLetra(tablero.head))

imprimirTablero(columnas, filas, fila, columna + 1, tablero.tail)

}

else{

//Si entramos aquí es porque hemos imprimido la última columna de la fila, por lo que imprimimos salto de línea, aumentamos fila en uno y llamamos al metodo con columna == 0

println("")

imprimirTablero(columnas, filas, fila + 1, 0, tablero)

}

}

//Metodo para génerar el tablero con números aleatorios

def generarTablero(tam:Int,tablero:List[Int],bloques:Int):List[Int]={

if(tam==0) tablero

else{

val x : List[Int]= tablero:::List(Random.nextInt(bloques)+1)

generarTablero(tam-1,x,bloques)

}

}

//Metodo que sustituye los 0 del tablero por bloques en funcion de la dificultad con la que juguemos

def rellenarTablero(tablero:List[Int], tam:Int, pos:Int, bloques:Int):List[Int] ={

//Si ya hemos recorrido todo el tablero, devolvemos el mismo

if(tam == pos) tablero

else{

//Si no, comprobamos si la posicion en la que nos encontramos es igual a 0

if(devuelveBloque(tablero, pos) == 0){

//Si es igual a cero, ponemos el bloque correspondiente en esa posicion y llamamos a rellenar con pos+1, el código extendido quedaría así

/\*

val bloqueNuevo = Random.nextInt(bloques)+1

val tableroAux = poner(pos,bloqueNuevo, tablero, Nil)

rellenarTablero(tableroAux, tam, pos + 1, bloques)

\*/

rellenarTablero(poner(pos, Random.nextInt(bloques)+1, tablero, Nil), tam, (pos + 1), bloques)

}

else{

//Si no es igual a cero, pasamos esa posición sin modificarla

rellenarTablero(tablero, tam, pos+1, bloques)

}

}

}

//Metodo que devuelve una posicion concreta del tablero

def devuelveBloque(tablero:List[Int],pos:Int):Int={

if(pos == 0) tablero.head

else devuelveBloque(tablero.tail,pos-1)

}

//Metodo que comprueba si la ficha de encima de la comprobada es igual

def compruebaArriba(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

//Comprobamos que anterior (es el bloque que estamos comprobando) es igual a su posicion de encima

if(anterior == devuelveBloque(tablero, (fila - 1) \* columnas + columna)){

//Si es igual, ponemos la posicion comprobada más la de arriba a 0 y volvemos a llamar a compruebaPiezas para que vaya siguiendo los distintos bloques y comprobando si

//Hay mas combinaciones

/\*

def compruebaPiezas(tablero:List[Int], columna:Int, fila:Int, filas:Int,columnas:Int, anterior:Int, dificultad:Int): List[Int] ={

tableroAux = poner((fila \* columnas) + columna, 0, tablero, Nil)

tableroAux2 = poner(((fila - 1) \* columnas) + columna, 0, tableroAux, Nil)

compruebaPiezas(tableroAux2, columna, fila - 1, filas, columnas, anterior, 0)

¡IMPORTANTE!

En este caso de arriba, la dificultad la pasamos como 0 ya que no nos va a hacer falta en las comprobaciones, solo es necesaria en el caso de la bomba que elimina todos

los bloques de un color

\*/

compruebaPiezas(poner(((fila -1) \* columnas + columna),0,poner((fila \* columnas) + columna, 0, tablero,Nil),Nil), columna, (fila - 1), filas, columnas, anterior, 1)

}

else{

tablero

}

}

//En el resto de csos la comprobación se realiza como en el método anterior, solo que comprobando debajo, derecha e izquierda respectivamente

def compruebaAbajo(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

if(anterior == devuelveBloque(tablero, (fila + 1) \* columnas + columna)){

compruebaPiezas(poner((fila + 1) \* columnas + columna, 0, poner((fila \* columnas) + columna, 0,tablero,Nil),Nil), columna, (fila + 1), filas, columnas, anterior, 1)

}

else{

tablero

}

}

def compruebaDerecha(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

if(anterior == devuelveBloque(tablero, fila \* columnas + (columna + 1))){

compruebaPiezas(poner(fila \* columnas + (columna + 1), 0,poner((fila \* columnas) + columna, 0,tablero,Nil),Nil), (columna + 1), fila, filas, columnas, anterior, 1)

}

else{

tablero

}

}

def compruebaIzquierda(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

if(anterior == devuelveBloque(tablero, fila \* columnas + (columna - 1))){

compruebaPiezas(poner(fila \* columnas + (columna - 1), 0, poner((fila \* columnas) + columna, 0,tablero,Nil),Nil), (columna - 1), fila, filas, columnas, anterior, 1)

}

else{

tablero

}

}

//Esta bomba elimina la fila en la que se encuentra

def bombaFila(tablero:List[Int], columnas:Int, fila:Int, columna:Int) : List[Int] ={

//Se empieza en columna = 0, cuando lleguemos a columna == columnas, se devuelve el tablero ya que habremos recorrido toda

if(columna == columnas) return tablero

else{

//El código expandido es el siguiente

/\*

tableroAux = (poner((fila \* columnas) + columna), 0, tablero, Nil)

bombaFila(tableroAux, columnas, fila, columna + 1)

\*/

return bombaFila(poner(((fila \* columnas) + columna), 0,tablero,Nil), columnas, fila, columna +1)

}

}

def bombaColumna(tablero:List[Int], columnas:Int, filas:Int, fila:Int, columna:Int) : List[Int] ={

//En este caso se comprueba igual que en el de arriba, pero con el numero de filas como tope, ya que lo que elimina es una columna

if(fila == filas) return tablero

else{

return bombaColumna(poner(((fila \* columnas) + columna), 0,tablero,Nil), columnas, filas, fila + 1, columna)

}

}

def bombaTNT(tablero:List[Int], columnas:Int, filas:Int, columna:Int, fila:Int): List[Int] ={

//comprobamos donde estamos situados para explotar los bloques que correspondan

if(fila == 0 && columna == 0){

/\*

Si estamos en la esquina superior izquierda, solo se puede eliminar a la derecha, abajo, y la diagonal inferior derecha

val tableroAux1:List[Int] = poner(1,0,tablero,Nil)

val tableroAux2:List[Int] = poner(columnas,0,tableroAux1,Nil)

val tableroAux3:List[Int] = poner(columnas + 1, 0 , tableroAux2, Nil)

poner(0, 0 , tableroAux3, Nil)

\*/

poner(0, 0 , poner(columnas + 1, 0 , poner(columnas,0,poner(1,0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == 0 && columna == (columnas - 1)){

//Si estamos en la esquina superior derecha, solo se puede eliminar abajo, izquierda y diagonal inferior izquierda

poner((columnas + columna - 1), 0 , poner((columnas + columna), 0 , poner((columna - 1),0,poner(columna,0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == (filas -1) && columna == 0){

//Si estamos en la esquina inferior izquierda, solo se puede eliminar arriba, derecha y esquina superior derecha

poner((((filas - 1) \* columnas) + 1), 0 , poner(((fila - 1) \* columnas), 0 , poner(((fila \* columnas) + 1),0,poner((fila \* columnas),0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == (filas - 1) && columna == (columnas - 1)){

//Si estamos en la esquina inferior derecha, solo se puede eliminar arriba, izquierda y diagonal superior izquierda

poner((((fila - 1) \* columnas) + (columna - 1)), 0 , poner((((fila - 1) \* columnas) + columna), 0 , poner(((fila \* columnas) + (columna - 1)),0,poner(((fila \* columnas) + columna),0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == 0 && columna != 0 && columna != (columnas -1)){

//Si estamos en la primera fila y no estamos en las esquinas, se puede eliminar a izquierda, derecha, debajo, diagonal inferior izquierda y diagonal inferior derecha

poner((columnas + columna - 1), 0 , poner((columnas + columna + 1), 0 , poner((columnas + columna), 0 , poner((columna + 1), 0 , poner((columna - 1),0,poner(columna,0,tablero,Nil),Nil), Nil), Nil), Nil), Nil)

}

else if(fila == (filas - 1) && columna != 0 && columna != (columnas -1)){

//Si estamos en la ultima fila y no estamos en las esquinas, se puede eliminar a izquierda, derecha, debajo, diagonal superior izquierda y diagonal superior derecha

poner((((fila -1 ) \* columnas) + columna ),0,poner((((fila -1 ) \* columnas) + (columna + 1)),0,poner((((fila -1 ) \* columnas) + (columna - 1)),0,poner(((fila \* columnas) + (columna - 1)),0,poner(((fila \* columnas) + (columna + 1)),0,poner(((fila \* columnas) + columna),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil)

}

else if(columna == 0 && fila != 0 && fila != (filas - 1)){

//Si estamos en la columna izquierda y no estamos en las esquinas, eliminamos arriba, abajo, derecha, diagonal superior derecha y diagonal inferior derecha

poner(((fila - 1) \* (columnas + 1)),0,poner(((fila + 1) \* (columnas + 1)),0,poner(((fila + 1) \* columnas),0,poner(((fila - 1) \* columnas),0,poner(((fila \* columnas) + 1),0,poner((fila \* columnas),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil)

}

else if(columna == (columnas - 1) && fila != 0 && fila != filas -1){

//Si estamos en la columna derecha y no estamos en las esquinas, eliminamos arriba, abajo, izquierda, diagonal superior izquierda y diagonal inferior izquierda

poner((((fila - 1) + columnas) + (columna - 1)),0,poner((((fila + 1) + columnas) + (columna - 1)),0,poner((((fila + 1) + columnas) + columna),0,poner((((fila - 1) + columnas) + columna),0,poner(((fila + columnas) + (columna -1)),0,poner(((fila + columnas) + columna),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil)

}

else{

//Si estamos en cualquier otra posicion, eliminamos la matriz de 3x3

poner((((fila + 1 ) \* columnas) + (columna + 1)),0,poner((((fila + 1 ) \* columnas) + (columna -1)),0,poner((((fila + 1 ) \* columnas) + (columna)),0,poner((((fila - 1 ) \* columnas) + (columna - 1)),0,poner((((fila - 1 ) \* columnas) + (columna + 1)),0,poner((((fila - 1 ) \* columnas) + (columna)),0,poner(((fila \* columnas) + (columna + 1)),0,poner(((fila \* columnas) + (columna + 1)),0,poner(((fila \* columnas) + columna),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil),Nil),Nil),Nil)

}

}

//Eliminamos todos los bloques del mismo color que se haya pasado como parametro

def bombaColores(tablero:List[Int], bloque:Int, pos:Int, tam:Int): List[Int] ={

if(pos == tam) { //Si estamos en el final devolvemos el tablero

return tablero

}

else{

//Si no, comprobamos que la posicion en la que estamos es igual al bloque que queremos eliminar

if(devuelveBloque(tablero, pos) == bloque){

//Si es igual, ponemos un 0 en esa posicion y llamamos al metodo con pos + 1

bombaColores(poner(pos, 0, tablero, Nil), bloque, pos + 1, tam)

}

else{

//Si no es igual, pasamos esa posicion sin modificarla

bombaColores(tablero, bloque, pos + 1, tam)

}

}

}

//Pone la bomba que se le pasa por parametro en el tablero

def ponerBomba(tablero:List[Int], fila:Int, columna:Int, columnas:Int, bomba:Int): List[Int]={

return poner((fila \* columnas) + columna, bomba,tablero,Nil)

}

//Comprueba piezas es el método encargado de llamar a los metodos compruebaAbajo,compruebaArriba, compruebaDerecha y compruebaIzquierda en función de la posición en la que nos

//encontremos

//También comprueba si el bloque que se le ha pasado es una bomba y llama a la funcíón que corresponda en cada caso

def compruebaPiezas(tablero:List[Int], columna:Int, fila:Int, filas:Int,columnas:Int, anterior:Int, dificultad:Int): List[Int] ={

//bombas

if (anterior == 7){

val r1 = new scala.util.Random(System.currentTimeMillis())

if((r1.nextInt(5000) + 1) < 2500){

bombaFila(tablero,columnas,fila,0)

}

else{

bombaColumna(tablero,columnas,filas,0,columna)

}

}

else if(anterior == 8){

bombaTNT(tablero, columnas, filas, columna, fila)

}

else if(anterior == 9){

val r1 = new scala.util.Random(System.currentTimeMillis())

dificultad match{

case 1 => val bloque = r1.nextInt(4) + 1; bombaColores(poner(fila\*columnas + columna, 0, tablero, Nil), bloque , 0, 63);

case 2 => val bloque = r1.nextInt(5) + 1; bombaColores(poner(fila\*columnas + columna, 0, tablero, Nil), bloque, 0, 187);

case 3 => val bloque = r1.nextInt(6) + 1; bombaColores(poner(fila\*columnas + columna, 0, tablero, Nil), bloque, 0, 405);

}

}

//Esquina superior izquierda

else {

if(fila == 0 && columna == 0){

compruebaAbajo(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

//Esquina superior derecha

else if(fila == 0 && columna == (columnas - 1)){

compruebaAbajo(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

//Esquina inferior izquierda

else if(fila == (filas - 1) && columna == 0){

compruebaArriba(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

//Esquina inferior derecha

else if(fila == (filas - 1) && columna == (columnas - 1)){

compruebaArriba(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(fila == 0){

compruebaAbajo(anterior, compruebaIzquierda(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(fila == (filas - 1)){

compruebaArriba(anterior, compruebaIzquierda(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(columna == 0){

compruebaAbajo(anterior, compruebaArriba(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(columna == (columnas - 1)){

compruebaAbajo(anterior, compruebaArriba(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else{

compruebaDerecha(anterior, compruebaAbajo(anterior, compruebaArriba(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

}

}

//Cuenta los ceros del tablero para ir comprobando la puntuación que tenemos -> Se usa para llevar la puntuación y comprobar la jugada más óptima

def compruebaPuntuacion(tablero:List[Int]): Int = {

if(tablero.isEmpty) return 0

else{

if(tablero.head == 0){

return 1 + compruebaPuntuacion(tablero.tail)

}

else{

return 0 + compruebaPuntuacion(tablero.tail)

}

}

}

//Metodo encargado del comienzo del juego, recibe el tablero en cada jugada, la dificultad para saber a que nivel tiene que acceder, y la puntuacion para comprobar si el juego

//continua o no

def jugarToyBlast(tablero:List[Int], dificultad:Int, puntuacion:List[Int]): Unit ={

if(dificultad == 1){

//Si la dificultad es igual a 1, se comprueba si no se ha sobrepasado la puntuación para ganar

if(devuelveBloque(puntuacion,0) >= 20 && devuelveBloque(puntuacion,1) >= 20 && devuelveBloque(puntuacion,2) >= 20 && devuelveBloque(puntuacion,3) >= 20){

println("Has ganado, ¡FELICIDADES!")

}

else{

imprimirTablero(9, 7, 1, 0, tablero)

println("PUNTUACIÓN")

println("BLOQUES AZULES: " + devuelveBloque(puntuacion,0))

println("BLOQUES ROJOS: " + devuelveBloque(puntuacion,1))

println("BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2))

println("BLOQUES VERDES: " + devuelveBloque(puntuacion,3))

println("Introduzca el n de fila: ")

val fila = elegirFila(7)

println("Introduzca el n de columna: ")

val columna = elegirColumna(9)

val bloque = devuelveBloque(tablero, (fila-1) \* 9 + (columna-1))

if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) == 5 && bloque < 5){

jugarToyBlast(rellenarTablero(subirCeros(62, 9, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad), (fila -1), (columna -1), 9, 7)), 63, 0, 4), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) == 6 && bloque < 5){

jugarToyBlast(rellenarTablero(subirCeros(62, 9, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad), (fila -1), (columna -1), 9, 8)), 63, 0, 4), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) > 7 && bloque < 5){

jugarToyBlast(rellenarTablero(subirCeros(62, 9, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad), (fila -1), (columna -1), 9, 9)), 63, 0, 4), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if (bloque < 5){

jugarToyBlast(rellenarTablero(subirCeros(62, 9, compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)), 63, 0, 4), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else{

jugarToyBlast(rellenarTablero(subirCeros(62, 9, compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)), 63, 0, 4), dificultad,puntuacion)

}

}

}

else if(dificultad == 2){

if(devuelveBloque(puntuacion,0) >= 15 && devuelveBloque(puntuacion,1) >= 15 && devuelveBloque(puntuacion,2) >= 15 && devuelveBloque(puntuacion,3) >= 15 && devuelveBloque(puntuacion,4) >= 15){

println("Has ganado, ¡FELICIDADES!")

}

else{

imprimirTablero(17, 11, 1, 0, tablero)

println("PUNTUACIÓN")

println("BLOQUES AZULES: " + devuelveBloque(puntuacion,0))

println("BLOQUES ROJOS: " + devuelveBloque(puntuacion,1))

println("BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2))

println("BLOQUES VERDES: " + devuelveBloque(puntuacion,3))

println("BLOQUES PLATA: " + devuelveBloque(puntuacion,4))

println("Introduzca el n de fila: ")

val fila = elegirFila(11)

println("Introduzca el n de columna: ")

val columna = elegirColumna(17)

val bloque = devuelveBloque(tablero, (fila-1) \* 17 + (columna-1))

if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) == 5 && bloque < 6){

jugarToyBlast(rellenarTablero(subirCeros(186, 17, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad), (fila -1), (columna -1), 17, 7)), 187, 0, 5), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) == 6 && bloque < 6){

jugarToyBlast(rellenarTablero(subirCeros(186, 17, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad), (fila -1), (columna -1), 17, 8)), 187, 0, 5), dificultad, poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) > 7 && bloque < 6){

jugarToyBlast(rellenarTablero(subirCeros(186, 17, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad), (fila -1), (columna -1), 17, 9)), 187, 0, 5), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if (bloque < 6){

jugarToyBlast(rellenarTablero(subirCeros(186, 17, compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)), 187, 0, 5), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else{

jugarToyBlast(rellenarTablero(subirCeros(186, 17, compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)), 187, 0, 5), dificultad,puntuacion)

}

}

}

else if(dificultad == 3){

if(devuelveBloque(puntuacion,0) >= 10 && devuelveBloque(puntuacion,1) >= 10 && devuelveBloque(puntuacion,2) >= 10 && devuelveBloque(puntuacion,3) >= 10 && devuelveBloque(puntuacion,4) >= 10 && devuelveBloque(puntuacion,5) >= 10){

println("Has ganado, ¡FELICIDADES!")

}

else{

imprimirTablero(27, 15, 1, 0, tablero)

println("PUNTUACIÓN")

println("BLOQUES AZULES: " + devuelveBloque(puntuacion,0))

println("BLOQUES ROJOS: " + devuelveBloque(puntuacion,1))

println("BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2))

println("BLOQUES VERDES: " + devuelveBloque(puntuacion,3))

println("BLOQUES PLATEADOS: " + devuelveBloque(puntuacion,4))

println("BLOQUES MORADOS: " + devuelveBloque(puntuacion,5))

println("Introduzca el n de fila: ")

val fila = elegirFila(15)

println("Introduzca el n de columna: ")

val columna = elegirColumna(27)

val bloque = devuelveBloque(tablero, (fila-1) \* 27 + (columna-1))

if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) == 5 && bloque < 7){

jugarToyBlast(rellenarTablero(subirCeros(404, 27, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad), (fila -1), (columna -1), 27, 7)), 405, 0, 6), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) == 6 && bloque < 7){

jugarToyBlast(rellenarTablero(subirCeros(404, 27, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad), (fila -1), (columna -1), 27, 8)), 405, 0, 6), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) > 7 && bloque < 7){

jugarToyBlast(rellenarTablero(subirCeros(404, 27, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad), (fila -1), (columna -1), 27, 9)), 405, 0, 6), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if (bloque < 7){

jugarToyBlast(rellenarTablero(subirCeros(404, 27, compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)), 405, 0, 6), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else{

jugarToyBlast(rellenarTablero(subirCeros(404, 27, compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)), 405, 0, 6), dificultad,puntuacion)

}

}

}

else{

terminarJuego()

}

}

//AQUÍ TERMINA

/\*

\* DIFICULTAD 1 -> 7 X 9 CON 4 BLOQUES DISTINTOS, LOGRANDO 20 DE CADA COLOR

\* DIFICULTAD 2 -> 11 X 17 CON 5 BLOQUES DISTINTOS, LOGRANDO 15 DE CADA COLOR

\* DIFICULTAD 3 -> 15 X 27 CON 6 BLOQUES DE COLORES, LOGRANDO 10 DE CADA COLOR PARA GANAR

\* \*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* AQUI EMPIEZA LA EJECUCIÓN DEL PROGRAMA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//)

val dificultad = imprimirMenu()

if(dificultad == 1){

jugarToyBlast(generarTablero(63, Nil, 4), dificultad,List(0,0,0,0))

}

if(dificultad == 2){

jugarToyBlast(generarTablero(187, Nil, 5), dificultad,List(0,0,0,0,0))

}

if(dificultad == 3){

jugarToyBlast(generarTablero(405, Nil, 6), dificultad,List(0,0,0,0,0,0))

}

else{

terminarJuego()

}

}

* MainOptimo:

import scala.util.Random

import Console.\_

import scala.sys.process.\_

import java.time.\_

object Main extends App {

//Este método pone el elemento n en la posicion pos de la lista l

def poner(pos:Int,n:Int,l:List[Int],laux:List[Int]):List[Int]={

if(pos==0) laux:::List[Int](n):::l.tail

else poner(pos-1,n,l.tail,laux:::List[Int](l.head))

}

//Este método se utiliza en subirCerosAux, pone la ficha de la pos1 en la pos2 y viceversa

def intercambiarCasilla(pos1: Int, pos2: Int, lista: List[Int]): List[Int] = {

//Dado que hemos intentado ahorrar todos los val posible, vamos dejar comentado el código de manera más extensa para que se entienda mejor

/\*

val ficha1 = devuelveBloque(lista, pos1)

val ficha2 = devuelveBloque(lista, pos2)

val tableroAux = poner(pos1, ficha2, lista, Nil)

poner(pos2,ficha1,tableroAux,Nil)

Lo que hemos hecho ha sido ir sustituyendo la parte derecha de la igualacion en las llamadas a funciones que las prosiguen, en este caso quedaría así paso a paso:

1) Dado que ficha1 = devuelveBloque(lista,pos1) -> en poner(pos2, ficha1,tableroAux,Nil) sustituimos, quedando así -> poner(pos2,devuelveBloque(lista, pos1),tableroAux,Nil)

2) tableroAux = poner(pos1, ficha2, lista, Nil) -> sustuituimos ficha2 por su igualación -> poner(pos1, devuelveBloque(lista, pos2), lista, Nil)

3) Por ultimo, en poner(pos2,devuelveBloque(lista, pos1),tableroAux,Nil), sustituimos tableroAux por lo de arriba -> poner(pos1, devuelveBloque(lista, pos2), lista, Nil)

-> poner(pos2,devuelveBloque(lista, pos1),poner(pos1, devuelveBloque(lista, pos2), lista, Nil) ,Nil)

\*/

poner(pos2,devuelveBloque(lista,pos1),poner(pos1,devuelveBloque(lista,pos2),lista,Nil),Nil)

}

def subirCeros(pos:Int,col:Int,l:List[Int]):List[Int]={

if (pos==0) l

else if (devuelveBloque(l, pos) == 0) subirCerosAux(pos,pos-col,col,l)

else subirCeros(pos-1,col,l)

}

def subirCerosAux(pos:Int,pos2:Int,col:Int,l:List[Int]):List[Int]={

if(pos<=col || pos2<1)subirCeros(pos-1,col,l)

else if(devuelveBloque(l, pos2) == 0) subirCerosAux(pos,pos2-col,col,l)

else subirCeros(pos-1,col,intercambiarCasilla(pos,pos2,l))

}

//Aquí controlamos que en el menú inicial se seleccione un valor entre 1 y 4 para no tener problemas a la hora de llamar a los niveles

def controlDificultad(): Int ={

println("La opción elegida no es correcta, por favor seleccione un valor del 1 al 4")

println("1 - Fácil \n2 - Medio \n3 - Díficil \n4 - Salir")

val eleccion = Integer.parseInt(Console.in.readLine())

//Si la elección no está entre los valores que queremos, volvemos a llamar al mismo metodo para que vuelva a pedir la dificultad

if(eleccion < 1 || eleccion > 4){

controlDificultad()

}

else{

//Si está dentro de los límites que nos interesan, devolvemos el int

eleccion

}

}

def imprimirMenu(): Int ={

//Este es el método que imprime el menú inicial para seleccionar la dificultad

println("+-+- BIENVENIDO A TOYBLAST -+-+")

println("Seleccione la dificultad del juego")

println("1 - Fácil \n2 - Medio \n3 - Díficil \n4 - Salir")

val dificultad = Integer.parseInt(Console.in.readLine())

if(dificultad < 1 || dificultad > 4){

//Si los valores no están entre las opciones que nos interesan, llamamos al metodo explicado arriba para controlar el entero recibido

controlDificultad()

}

else{

dificultad

}

}

def terminarJuego(): Unit ={

//Funcion que sale del juego devolviendo error si pulsamos 4 en el menu inicial

println("Saliendo del juego...\n")

System.exit(-1)

}

//A este método se le llama desde imprimir tablero para que sea más fácil localizar la columna en la que queremos jugar

def imprimirColumnas(columnas:Int): Unit = {

if(columnas == 9){

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

println(" | | | | | | | | | ")

}

else if(columnas == 17){

println(" 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17")

println(" | | | | | | | | | | | | | | | | |")

}

else if(columnas == 27){

println(" 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27")

println(" | | | | | | | | | | | | | | | | | | | | | | | | | | |")

}

}

//Metodo para imprimir una letra en funcion del bloque o bomba que se encuentre en el tablero

def imprimirLetra(bloque:Int): String ={

bloque match{

case 1 => return "A"

case 2 => return "R"

case 3 => return "N"

case 4 => return "V"

case 5 => return "P"

case 6 => return "M"

case 7 => return "a"

case 8 => return "b"

case 9 => return "c"

case 0 => return "X"

}

}

def imprimirTablero(columnas:Int, filas:Int, fila:Int, columna:Int, tablero:List[Int]): Unit = {

//Si ya hemos imprimido todo hacemos un salto de línea para que no quede todo pegado

if(tablero.isEmpty) println("")

else if(columna == 0){

//Si estamos en la primera fila y en la primera columna, llamamos al método imprimir columnas de más arriba para la guía

if(fila == 1){

imprimirColumnas(columnas)

}

//Para que todas las columnas queden a la misma altura, controlo el número de fila e imprimo un guión menos, ya que si no a partir de la 9 quedan desplazadas a la derecha

if(fila > 9){

//Imprimimos el numero de fila y aumentamos la columna para acceder al elemento siguiente, esto solo pasa si columna es == 0

print(fila + "---- " + imprimirLetra(tablero.head))

imprimirTablero(columnas, filas, fila, columna + 1, tablero.tail)

}

else{

//Imprimimos el numero de fila y aumentamos la columna para acceder al elemento siguiente, esto solo pasa si columna es == 0

print(fila + "----- " + imprimirLetra(tablero.head))

imprimirTablero(columnas, filas, fila, columna + 1, tablero.tail)

}

}

else if(columna < columnas){

//Si columna es distinto de 0 y columna es < que el número de columnas, solo se imprime la cabeza de la lista y se llama de nuevo a imprimir tablero con columna + 1

print(" " + imprimirLetra(tablero.head))

imprimirTablero(columnas, filas, fila, columna + 1, tablero.tail)

}

else{

//Si entramos aquí es porque hemos imprimido la última columna de la fila, por lo que imprimimos salto de línea, aumentamos fila en uno y llamamos al metodo con columna == 0

println("")

imprimirTablero(columnas, filas, fila + 1, 0, tablero)

}

}

//Metodo para génerar el tablero con números aleatorios en función del número de bloques con los que se juegue

def generarTablero(tam:Int,tablero:List[Int],bloques:Int):List[Int]={

if(tam==0) tablero

else{

val x : List[Int]= tablero:::List(Random.nextInt(bloques)+1)

generarTablero(tam-1,x,bloques)

}

}

//Metodo que sustituye los 0 del tablero por bloques en funcion de la dificultad con la que juguemos

def rellenarTablero(tablero:List[Int], tam:Int, pos:Int, bloques:Int):List[Int] ={

//Si ya hemos recorrido todo el tablero, devolvemos el mismo

if(tam == pos) tablero

else{

//Si no, comprobamos si la posicion en la que nos encontramos es igual a 0

if(devuelveBloque(tablero, pos) == 0){

//Si es igual a cero, ponemos el bloque correspondiente en esa posicion y llamamos a rellenar con pos+1, el código extendido quedaría así

/\*

val bloqueNuevo = Random.nextInt(bloques)+1

val tableroAux = poner(pos,bloqueNuevo, tablero, Nil)

rellenarTablero(tableroAux, tam, pos + 1, bloques)

\*/

rellenarTablero(poner(pos, Random.nextInt(bloques)+1, tablero, Nil), tam, (pos + 1), bloques)

}

else{

//Si no es igual a cero, pasamos esa posición sin modificarla

rellenarTablero(tablero, tam, pos+1, bloques)

}

}

}

//Metodo que devuelve una posicion concreta del tablero

def devuelveBloque(tablero:List[Int],pos:Int):Int={

if(pos == 0) tablero.head

else devuelveBloque(tablero.tail,pos-1)

}

//Metodo que comprueba si la ficha de encima de la comprobada es igual

def compruebaArriba(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

//Comprobamos que anterior (es el bloque que estamos comprobando) es igual a su posicion de encima

if(anterior == devuelveBloque(tablero, (fila - 1) \* columnas + columna)){

//Si es igual, ponemos la posicion comprobada más la de arriba a 0 y volvemos a llamar a compruebaPiezas para que vaya siguiendo los distintos bloques y comprobando si

//Hay mas combinaciones

/\*

def compruebaPiezas(tablero:List[Int], columna:Int, fila:Int, filas:Int,columnas:Int, anterior:Int, dificultad:Int): List[Int] ={

tableroAux = poner((fila \* columnas) + columna, 0, tablero, Nil)

tableroAux2 = poner(((fila - 1) \* columnas) + columna, 0, tableroAux, Nil)

compruebaPiezas(tableroAux2, columna, fila - 1, filas, columnas, anterior, 0)

¡IMPORTANTE!

En este caso de arriba, la dificultad la pasamos como 0 ya que no nos va a hacer falta en las comprobaciones, solo es necesaria en el caso de la bomba que elimina todos

los bloques de un color

\*/

compruebaPiezas(poner(((fila -1) \* columnas + columna),0,poner((fila \* columnas) + columna, 0, tablero,Nil),Nil), columna, (fila - 1), filas, columnas, anterior,0)

}

else{

tablero

}

}

//En el resto de csos la comprobación se realiza como en el método anterior, solo que comprobando debajo, derecha e izquierda respectivamente

def compruebaAbajo(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

if(anterior == devuelveBloque(tablero, (fila + 1) \* columnas + columna)){

compruebaPiezas(poner((fila + 1) \* columnas + columna, 0, poner((fila \* columnas) + columna, 0,tablero,Nil),Nil), columna, (fila + 1), filas, columnas, anterior,0)

}

else{

tablero

}

}

def compruebaDerecha(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

if(anterior == devuelveBloque(tablero, fila \* columnas + (columna + 1))){

compruebaPiezas(poner(fila \* columnas + (columna + 1), 0,poner((fila \* columnas) + columna, 0,tablero,Nil),Nil), (columna + 1), fila, filas, columnas, anterior,0)

}

else{

tablero

}

}

def compruebaIzquierda(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

if(anterior == devuelveBloque(tablero, fila \* columnas + (columna - 1))){

compruebaPiezas(poner(fila \* columnas + (columna - 1), 0, poner((fila \* columnas) + columna, 0,tablero,Nil),Nil), (columna - 1), fila, filas, columnas, anterior,0)

}

else{

tablero

}

}

//Esta bomba elimina la fila en la que se encuentra

def bombaFila(tablero:List[Int], columnas:Int, fila:Int, columna:Int) : List[Int] ={

//Se empieza en columna = 0, cuando lleguemos a columna == columnas, se devuelve el tablero ya que habremos recorrido toda

if(columna == columnas) return tablero

else{

//El código expandido es el siguiente

/\*

tableroAux = (poner((fila \* columnas) + columna), 0, tablero, Nil)

bombaFila(tableroAux, columnas, fila, columna + 1)

\*/

return bombaFila(poner(((fila \* columnas) + columna), 0,tablero,Nil), columnas, fila, columna +1)

}

}

def bombaColumna(tablero:List[Int], columnas:Int, filas:Int, fila:Int, columna:Int) : List[Int] ={

//En este caso se comprueba igual que en el de arriba, pero con el numero de filas como tope, ya que lo que elimina es una columna

if(fila == filas) return tablero

else{

return bombaColumna(poner(((fila \* columnas) + columna), 0,tablero,Nil), columnas, filas, fila + 1, columna)

}

}

def bombaTNT(tablero:List[Int], columnas:Int, filas:Int, columna:Int, fila:Int): List[Int] ={

//comprobamos donde estamos situados para explotar los bloques que correspondan, ya que si no podríamos salirnos de los límites del tablero

if(fila == 0 && columna == 0){

/\*

Si estamos en la esquina superior izquierda, solo se puede eliminar a la derecha, abajo, y la diagonal inferior derecha

val tableroAux1:List[Int] = poner(1,0,tablero,Nil)

val tableroAux2:List[Int] = poner(columnas,0,tableroAux1,Nil)

val tableroAux3:List[Int] = poner(columnas + 1, 0 , tableroAux2, Nil)

poner(0, 0 , tableroAux3, Nil)

\*/

poner(0, 0 , poner(columnas + 1, 0 , poner(columnas,0,poner(1,0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == 0 && columna == (columnas - 1)){

//Si estamos en la esquina superior derecha, solo se puede eliminar abajo, izquierda y diagonal inferior izquierda

poner((columnas + columna - 1), 0 , poner((columnas + columna), 0 , poner((columna - 1),0,poner(columna,0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == (filas -1) && columna == 0){

//Si estamos en la esquina inferior izquierda, solo se puede eliminar arriba, derecha y esquina superior derecha

poner((((filas - 1) \* columnas) + 1), 0 , poner(((fila - 1) \* columnas), 0 , poner(((fila \* columnas) + 1),0,poner((fila \* columnas),0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == (filas - 1) && columna == (columnas - 1)){

//Si estamos en la esquina inferior derecha, solo se puede eliminar arriba, izquierda y diagonal superior izquierda

poner((((fila - 1) \* columnas) + (columna - 1)), 0 , poner((((fila - 1) \* columnas) + columna), 0 , poner(((fila \* columnas) + (columna - 1)),0,poner(((fila \* columnas) + columna),0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == 0 && columna != 0 && columna != (columnas -1)){

//Si estamos en la primera fila y no estamos en las esquinas, se puede eliminar a izquierda, derecha, debajo, diagonal inferior izquierda y diagonal inferior derecha

poner((columnas + columna - 1), 0 , poner((columnas + columna + 1), 0 , poner((columnas + columna), 0 , poner((columna + 1), 0 , poner((columna - 1),0,poner(columna,0,tablero,Nil),Nil), Nil), Nil), Nil), Nil)

}

else if(fila == (filas - 1) && columna != 0 && columna != (columnas -1)){

//Si estamos en la ultima fila y no estamos en las esquinas, se puede eliminar a izquierda, derecha, debajo, diagonal superior izquierda y diagonal superior derecha

poner((((fila -1 ) \* columnas) + columna ),0,poner((((fila -1 ) \* columnas) + (columna + 1)),0,poner((((fila -1 ) \* columnas) + (columna - 1)),0,poner(((fila \* columnas) + (columna - 1)),0,poner(((fila \* columnas) + (columna + 1)),0,poner(((fila \* columnas) + columna),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil)

}

else if(columna == 0 && fila != 0 && fila != (filas - 1)){

//Si estamos en la columna izquierda y no estamos en las esquinas, eliminamos arriba, abajo, derecha, diagonal superior derecha y diagonal inferior derecha

poner(((fila - 1) \* (columnas + 1)),0,poner(((fila + 1) \* (columnas + 1)),0,poner(((fila + 1) \* columnas),0,poner(((fila - 1) \* columnas),0,poner(((fila \* columnas) + 1),0,poner((fila \* columnas),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil)

}

else if(columna == (columnas - 1) && fila != 0 && fila != filas -1){

//Si estamos en la columna derecha y no estamos en las esquinas, eliminamos arriba, abajo, izquierda, diagonal superior izquierda y diagonal inferior izquierda

poner((((fila - 1) + columnas) + (columna - 1)),0,poner((((fila + 1) + columnas) + (columna - 1)),0,poner((((fila + 1) + columnas) + columna),0,poner((((fila - 1) + columnas) + columna),0,poner(((fila + columnas) + (columna -1)),0,poner(((fila + columnas) + columna),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil)

}

else{

//Si estamos en cualquier otra posicion, eliminamos la matriz de 3x3

poner((((fila + 1 ) \* columnas) + (columna + 1)),0,poner((((fila + 1 ) \* columnas) + (columna -1)),0,poner((((fila + 1 ) \* columnas) + (columna)),0,poner((((fila - 1 ) \* columnas) + (columna - 1)),0,poner((((fila - 1 ) \* columnas) + (columna + 1)),0,poner((((fila - 1 ) \* columnas) + (columna)),0,poner(((fila \* columnas) + (columna + 1)),0,poner(((fila \* columnas) + (columna + 1)),0,poner(((fila \* columnas) + columna),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil),Nil),Nil),Nil)

}

}

//Eliminamos todos los bloques del mismo color que se haya pasado como parametro

def bombaColores(tablero:List[Int], bloque:Int, pos:Int, tam:Int): List[Int] ={

if(pos == tam) { //Si estamos en el final devolvemos el tablero

return tablero

}

else{

//Si no, comprobamos que la posicion en la que estamos es igual al bloque que queremos eliminar

if(devuelveBloque(tablero, pos) == bloque){

//Si es igual, ponemos un 0 en esa posicion y llamamos al metodo con pos + 1

bombaColores(poner(pos, 0, tablero, Nil), bloque, pos + 1, tam)

}

else{

//Si no es igual, pasamos esa posicion sin modificarla

bombaColores(tablero, bloque, pos + 1, tam)

}

}

}

//Pone la bomba que se le pasa por parametro en el tablero

def ponerBomba(tablero:List[Int], fila:Int, columna:Int, columnas:Int, bomba:Int): List[Int]={

return poner((fila \* columnas) + columna, bomba,tablero,Nil)

}

//Comprueba piezas es el método encargado de llamar a los metodos compruebaAbajo,compruebaArriba, compruebaDerecha y compruebaIzquierda en función de la posición en la que nos

//encontremos

//También comprueba si el bloque que se le ha pasado es una bomba y llama a la funcíón que corresponda en cada caso

def compruebaPiezas(tablero:List[Int], columna:Int, fila:Int, filas:Int,columnas:Int, anterior:Int, dificultad:Int): List[Int] ={

//bombas

if (anterior == 7){

val r1 = new scala.util.Random(System.currentTimeMillis())

if((r1.nextInt(5000) + 1) < 2500){

bombaFila(tablero,columnas,fila,0)

}

else{

bombaColumna(tablero,columnas,filas,0,columna)

}

}

else if(anterior == 8){

bombaTNT(tablero, columnas, filas, columna, fila)

}

else if(anterior == 9){

val r1 = new scala.util.Random(System.currentTimeMillis())

dificultad match{

case 1 => val bloque = r1.nextInt(4) + 1; bombaColores(poner(fila\*columnas + columna, 0, tablero, Nil), bloque , 0, 63);

case 2 => val bloque = r1.nextInt(5) + 1; bombaColores(poner(fila\*columnas + columna, 0, tablero, Nil), bloque, 0, 187);

case 3 => val bloque = r1.nextInt(6) + 1; bombaColores(poner(fila\*columnas + columna, 0, tablero, Nil), bloque, 0, 405);

}

}

//Esquina superior izquierda

else {

if(fila == 0 && columna == 0){

compruebaAbajo(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

//Esquina superior derecha

else if(fila == 0 && columna == (columnas - 1)){

compruebaAbajo(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

//Esquina inferior izquierda

else if(fila == (filas - 1) && columna == 0){

compruebaArriba(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

//Esquina inferior derecha

else if(fila == (filas - 1) && columna == (columnas - 1)){

compruebaArriba(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(fila == 0){

compruebaAbajo(anterior, compruebaIzquierda(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(fila == (filas - 1)){

compruebaArriba(anterior, compruebaIzquierda(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(columna == 0){

compruebaAbajo(anterior, compruebaArriba(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(columna == (columnas - 1)){

compruebaAbajo(anterior, compruebaArriba(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else{

compruebaDerecha(anterior, compruebaAbajo(anterior, compruebaArriba(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

}

}

//Cuenta los ceros del tablero para ir comprobando la puntuación que tenemos -> Se usa para llevar la puntuación y comprobar la jugada más óptima

def compruebaPuntuacion(tablero:List[Int]): Int = {

if(tablero.isEmpty) return 0

else{

if(tablero.head == 0){

return 1 + compruebaPuntuacion(tablero.tail)

}

else{

return 0 + compruebaPuntuacion(tablero.tail)

}

}

}

//Esta jugada recorre todo el tablero comprobando en qué posición tenemos la jugada que más bloques elimina

def jugadaOptima(tablero:List[Int], fila:Int, columna:Int, filaoptima:Int, columnaoptima:Int, filas:Int, columnas:Int, puntuacion:Int): (Int,Int) ={

if(fila == (filas -1) && columna == (columnas -1)){

return (filaoptima + 1,columnaoptima + 1)

}

else if(columna == (columnas - 1)){

if( puntuacion < compruebaPuntuacion(compruebaPiezas(tablero, columna, fila, filas, columnas, devuelveBloque(tablero, ((fila \* columnas) + columna)), dificultad))){

jugadaOptima(tablero, fila + 1, 0, fila, columna, filas, columnas, compruebaPuntuacion(compruebaPiezas(tablero, columna, fila, filas, columnas, devuelveBloque(tablero, ((fila \* columnas) + columna)), dificultad)))

}

else{

jugadaOptima(tablero, fila + 1, 0, filaoptima, columnaoptima, filas, columnas, puntuacion)

}

}

else{

if(puntuacion < compruebaPuntuacion(compruebaPiezas(tablero, columna, fila, filas, columnas, devuelveBloque(tablero, ((fila \* columnas) + columna)), dificultad))){

jugadaOptima(tablero, fila, columna + 1, fila, columna, filas, columnas, compruebaPuntuacion(compruebaPiezas(tablero, columna, fila, filas, columnas, devuelveBloque(tablero, ((fila \* columnas) + columna)), dificultad)))

}

else{

jugadaOptima(tablero, fila, columna + 1, filaoptima, columnaoptima, filas, columnas, puntuacion)

}

}

}

//Metodo encargado del comienzo del juego, recibe el tablero en cada jugada, la dificultad para saber a que nivel tiene que acceder, y la puntuacion para comprobar si el juego

//continua o no

def jugarToyBlast(tablero:List[Int], dificultad:Int, puntuacion:List[Int]): Unit ={

if(dificultad == 1){

//Si la dificultad es igual a 1, se comprueba si no se ha sobrepasado la puntuación para ganar

if(devuelveBloque(puntuacion,0) >= 20 && devuelveBloque(puntuacion,1) >= 20 && devuelveBloque(puntuacion,2) >= 20 && devuelveBloque(puntuacion,3) >= 20){

println("Has ganado, ¡FELICIDADES!")

println("PUNTUACIÓN")

println("BLOQUES AZULES: " + devuelveBloque(puntuacion,0))

println("BLOQUES ROJOS: " + devuelveBloque(puntuacion,1))

println("BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2))

println("BLOQUES VERDES: " + devuelveBloque(puntuacion,3))

}

else{

//Si no hemos logrado la puntuación, imprimimos el tablero y la puntuacion acumulada hasta el momento

imprimirTablero(9, 7, 1, 0, tablero)

println("PUNTUACIÓN")

println("BLOQUES AZULES: " + devuelveBloque(puntuacion,0))

println("BLOQUES ROJOS: " + devuelveBloque(puntuacion,1))

println("BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2))

println("BLOQUES VERDES: " + devuelveBloque(puntuacion,3))

//Calculamos la fila y la columna con el mejor movimiento

val posOptimas = jugadaOptima(tablero, 0, 0, 0, 0, 7, 9, 0)

//Mostramos la eleccion realizada y la almacenamos en los val que vamos a usar despues

println("La fila seleccionada es : " + posOptimas.\_1)

val fila = posOptimas.\_1

println("La columna seleccionada es : " + posOptimas.\_2)

val columna = posOptimas.\_2

Thread.sleep(1000)

val bloque = devuelveBloque(tablero, (fila-1) \* 9 + (columna-1))

//Si eliminamos 5 bloques, ponemos la bomba de fila

if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) == 5 && bloque < 5){

jugarToyBlast(rellenarTablero(subirCeros(62, 9, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad), (fila -1), (columna -1), 9, 7)), 63, 0, 4), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

//Si eliminamos 6, ponemos el TNT

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) == 6 && bloque < 5){

jugarToyBlast(rellenarTablero(subirCeros(62, 9, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad), (fila -1), (columna -1), 9, 8)), 63, 0, 4), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

//Si eliminamos mas de 7, ponemos la bomba de colores

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) > 7 && bloque < 5){

jugarToyBlast(rellenarTablero(subirCeros(62, 9, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad), (fila -1), (columna -1), 9, 9)), 63, 0, 4), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if (bloque < 5){

jugarToyBlast(rellenarTablero(subirCeros(62, 9, compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)), 63, 0, 4), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else{

jugarToyBlast(rellenarTablero(subirCeros(62, 9, compruebaPiezas(tablero, (columna-1), (fila-1), 7, 9, bloque, dificultad)), 63, 0, 4), dificultad,puntuacion)

}

}

}

else if(dificultad == 2){

if(devuelveBloque(puntuacion,0) >= 15 && devuelveBloque(puntuacion,1) >= 15 && devuelveBloque(puntuacion,2) >= 15 && devuelveBloque(puntuacion,3) >= 15 && devuelveBloque(puntuacion,4) >= 15){

println("Has ganado, ¡FELICIDADES!")

println("PUNTUACIÓN")

println("BLOQUES AZULES: " + devuelveBloque(puntuacion,0))

println("BLOQUES ROJOS: " + devuelveBloque(puntuacion,1))

println("BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2))

println("BLOQUES VERDES: " + devuelveBloque(puntuacion,3))

println("BLOQUES PLATA: " + devuelveBloque(puntuacion,4))

}

else{

imprimirTablero(17, 11, 1, 0, tablero)

println("PUNTUACIÓN")

println("BLOQUES AZULES: " + devuelveBloque(puntuacion,0))

println("BLOQUES ROJOS: " + devuelveBloque(puntuacion,1))

println("BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2))

println("BLOQUES VERDES: " + devuelveBloque(puntuacion,3))

println("BLOQUES PLATA: " + devuelveBloque(puntuacion,4))

val posOptimas = jugadaOptima(tablero, 0, 0, 0, 0, 11, 17, 0)

println("La fila seleccionada es : " + posOptimas.\_1)

val fila = posOptimas.\_1

println("La columna seleccionada es : " + posOptimas.\_2)

val columna = posOptimas.\_2

Thread.sleep(1000)

val bloque = devuelveBloque(tablero, (fila-1) \* 17 + (columna-1))

if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) == 5 && bloque < 6){

jugarToyBlast(rellenarTablero(subirCeros(186, 17, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad), (fila -1), (columna -1), 17, 7)), 187, 0, 5), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) == 6 && bloque < 6){

jugarToyBlast(rellenarTablero(subirCeros(186, 17, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad), (fila -1), (columna -1), 17, 8)), 187, 0, 5), dificultad, poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) > 7 && bloque < 6){

jugarToyBlast(rellenarTablero(subirCeros(186, 17, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad), (fila -1), (columna -1), 17, 9)), 187, 0, 5), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if (bloque < 6){

jugarToyBlast(rellenarTablero(subirCeros(186, 17, compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)), 187, 0, 5), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else{

jugarToyBlast(rellenarTablero(subirCeros(186, 17, compruebaPiezas(tablero, (columna-1), (fila-1), 11, 17, bloque, dificultad)), 187, 0, 5), dificultad,puntuacion)

}

}

}

else if(dificultad == 3){

if(devuelveBloque(puntuacion,0) >= 10 && devuelveBloque(puntuacion,1) >= 10 && devuelveBloque(puntuacion,2) >= 10 && devuelveBloque(puntuacion,3) >= 10 && devuelveBloque(puntuacion,4) >= 10 && devuelveBloque(puntuacion,5) >= 10){

println("Has ganado, ¡FELICIDADES!")

println("PUNTUACIÓN")

println("BLOQUES AZULES: " + devuelveBloque(puntuacion,0))

println("BLOQUES ROJOS: " + devuelveBloque(puntuacion,1))

println("BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2))

println("BLOQUES VERDES: " + devuelveBloque(puntuacion,3))

println("BLOQUES PLATEADOS: " + devuelveBloque(puntuacion,4))

println("BLOQUES MORADOS: " + devuelveBloque(puntuacion,5))

}

else{

imprimirTablero(27, 15, 1, 0, tablero)

println("PUNTUACIÓN")

println("BLOQUES AZULES: " + devuelveBloque(puntuacion,0))

println("BLOQUES ROJOS: " + devuelveBloque(puntuacion,1))

println("BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2))

println("BLOQUES VERDES: " + devuelveBloque(puntuacion,3))

println("BLOQUES PLATEADOS: " + devuelveBloque(puntuacion,4))

println("BLOQUES MORADOS: " + devuelveBloque(puntuacion,5))

val posOptimas = jugadaOptima(tablero, 0, 0, 0, 0, 15, 27, 0)

println("La fila seleccionada es : " + posOptimas.\_1)

val fila = posOptimas.\_1

println("La columna seleccionada es : " + posOptimas.\_2)

val columna = posOptimas.\_2

Thread.sleep(1000)

val bloque = devuelveBloque(tablero, (fila-1) \* 27 + (columna-1))

if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) == 5 && bloque < 7){

jugarToyBlast(rellenarTablero(subirCeros(404, 27, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad), (fila -1), (columna -1), 27, 7)), 405, 0, 6), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) == 6 && bloque < 7){

jugarToyBlast(rellenarTablero(subirCeros(404, 27, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad), (fila -1), (columna -1), 27, 8)), 405, 0, 6), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if(compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) > 7 && bloque < 7){

jugarToyBlast(rellenarTablero(subirCeros(404, 27, ponerBomba(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad), (fila -1), (columna -1), 27, 9)), 405, 0, 6), dificultad,poner((bloque-1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else if (bloque < 7){

jugarToyBlast(rellenarTablero(subirCeros(404, 27, compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)), 405, 0, 6), dificultad,poner((bloque - 1),compruebaPuntuacion(compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)) + devuelveBloque(puntuacion,(bloque-1)),puntuacion,Nil))

}

else{

jugarToyBlast(rellenarTablero(subirCeros(404, 27, compruebaPiezas(tablero, (columna-1), (fila-1), 15, 27, bloque, dificultad)), 405, 0, 6), dificultad,puntuacion)

}

}

}

else{

terminarJuego()

}

}

//AQUÍ TERMINA

/\*

\* DIFICULTAD 1 -> 7 X 9 CON 4 BLOQUES DISTINTOS, LOGRANDO 20 DE CADA COLOR

\* DIFICULTAD 2 -> 11 X 17 CON 5 BLOQUES DISTINTOS, LOGRANDO 15 DE CADA COLOR

\* DIFICULTAD 3 -> 15 X 27 CON 6 BLOQUES DE COLORES, LOGRANDO 10 DE CADA COLOR PARA GANAR

\* \*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* AQUI EMPIEZA LA EJECUCIÓN DEL PROGRAMA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//)

val dificultad = imprimirMenu()

//Llamamos a jugarToyBlast en funcion de la dificultad, generando el tablero con las posiciones que correspondan y con una Lista inicializada a 0 para las puntuaciones

if(dificultad == 1){

jugarToyBlast(generarTablero(63, Nil, 4), dificultad,List(0,0,0,0))

}

if(dificultad == 2){

jugarToyBlast(generarTablero(187, Nil, 5), dificultad,List(0,0,0,0,0))

}

if(dificultad == 3){

jugarToyBlast(generarTablero(405, Nil, 6), dificultad,List(0,0,0,0,0,0))

}

else{

terminarJuego()

}

}

* MainGUI:

import scala.util.Random

import Console.\_

import javax.swing.\_

import javax.imageio.ImageIO

import java.\_

import java.lang.Short

object Main extends App{

def compruebaArriba(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

//Comprobamos que anterior (es el bloque que estamos comprobando) es igual a su posicion de encima

if(anterior == devuelveBloque(tablero, (fila - 1) \* columnas + columna)){

//Si es igual, ponemos la posicion comprobada más la de arriba a 0 y volvemos a llamar a compruebaPiezas para que vaya siguiendo los distintos bloques y comprobando si

//Hay mas combinaciones

/\*

def compruebaPiezas(tablero:List[Int], columna:Int, fila:Int, filas:Int,columnas:Int, anterior:Int, dificultad:Int): List[Int] ={

tableroAux = poner((fila \* columnas) + columna, 0, tablero, Nil)

tableroAux2 = poner(((fila - 1) \* columnas) + columna, 0, tableroAux, Nil)

compruebaPiezas(tableroAux2, columna, fila - 1, filas, columnas, anterior, 0)

¡IMPORTANTE!

En este caso de arriba, la dificultad la pasamos como 0 ya que no nos va a hacer falta en las comprobaciones, solo es necesaria en el caso de la bomba que elimina todos

los bloques de un color

\*/

compruebaPiezas(poner(((fila -1) \* columnas + columna),0,poner((fila \* columnas) + columna, 0, tablero,Nil),Nil), columna, (fila - 1), filas, columnas, anterior, 1)

}

else{

tablero

}

}

//En el resto de csos la comprobación se realiza como en el método anterior, solo que comprobando debajo, derecha e izquierda respectivamente

def compruebaAbajo(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

if(anterior == devuelveBloque(tablero, (fila + 1) \* columnas + columna)){

compruebaPiezas(poner((fila + 1) \* columnas + columna, 0, poner((fila \* columnas) + columna, 0,tablero,Nil),Nil), columna, (fila + 1), filas, columnas, anterior, 1)

}

else{

tablero

}

}

def compruebaDerecha(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

if(anterior == devuelveBloque(tablero, fila \* columnas + (columna + 1))){

compruebaPiezas(poner(fila \* columnas + (columna + 1), 0,poner((fila \* columnas) + columna, 0,tablero,Nil),Nil), (columna + 1), fila, filas, columnas, anterior, 1)

}

else{

tablero

}

}

def compruebaIzquierda(anterior:Int, tablero:List[Int], fila:Int, columna:Int, columnas:Int, filas:Int): List[Int] ={

if(anterior == devuelveBloque(tablero, fila \* columnas + (columna - 1))){

compruebaPiezas(poner(fila \* columnas + (columna - 1), 0, poner((fila \* columnas) + columna, 0,tablero,Nil),Nil), (columna - 1), fila, filas, columnas, anterior, 1)

}

else{

tablero

}

}

//Esta bomba elimina la fila en la que se encuentra

def bombaFila(tablero:List[Int], columnas:Int, fila:Int, columna:Int) : List[Int] ={

//Se empieza en columna = 0, cuando lleguemos a columna == columnas, se devuelve el tablero ya que habremos recorrido toda

if(columna == columnas) return tablero

else{

//El código expandido es el siguiente

/\*

tableroAux = (poner((fila \* columnas) + columna), 0, tablero, Nil)

bombaFila(tableroAux, columnas, fila, columna + 1)

\*/

return bombaFila(poner(((fila \* columnas) + columna), 0,tablero,Nil), columnas, fila, columna +1)

}

}

def bombaColumna(tablero:List[Int], columnas:Int, filas:Int, fila:Int, columna:Int) : List[Int] ={

//En este caso se comprueba igual que en el de arriba, pero con el numero de filas como tope, ya que lo que elimina es una columna

if(fila == filas) return tablero

else{

return bombaColumna(poner(((fila \* columnas) + columna), 0,tablero,Nil), columnas, filas, fila + 1, columna)

}

}

def bombaTNT(tablero:List[Int], columnas:Int, filas:Int, columna:Int, fila:Int): List[Int] ={

//comprobamos donde estamos situados para explotar los bloques que correspondan

if(fila == 0 && columna == 0){

/\*

Si estamos en la esquina superior izquierda, solo se puede eliminar a la derecha, abajo, y la diagonal inferior derecha

val tableroAux1:List[Int] = poner(1,0,tablero,Nil)

val tableroAux2:List[Int] = poner(columnas,0,tableroAux1,Nil)

val tableroAux3:List[Int] = poner(columnas + 1, 0 , tableroAux2, Nil)

poner(0, 0 , tableroAux3, Nil)

\*/

poner(0, 0 , poner(columnas + 1, 0 , poner(columnas,0,poner(1,0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == 0 && columna == (columnas - 1)){

//Si estamos en la esquina superior derecha, solo se puede eliminar abajo, izquierda y diagonal inferior izquierda

poner((columnas + columna - 1), 0 , poner((columnas + columna), 0 , poner((columna - 1),0,poner(columna,0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == (filas -1) && columna == 0){

//Si estamos en la esquina inferior izquierda, solo se puede eliminar arriba, derecha y esquina superior derecha

poner((((filas - 1) \* columnas) + 1), 0 , poner(((fila - 1) \* columnas), 0 , poner(((fila \* columnas) + 1),0,poner((fila \* columnas),0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == (filas - 1) && columna == (columnas - 1)){

//Si estamos en la esquina inferior derecha, solo se puede eliminar arriba, izquierda y diagonal superior izquierda

poner((((fila - 1) \* columnas) + (columna - 1)), 0 , poner((((fila - 1) \* columnas) + columna), 0 , poner(((fila \* columnas) + (columna - 1)),0,poner(((fila \* columnas) + columna),0,tablero,Nil),Nil), Nil), Nil)

}

else if(fila == 0 && columna != 0 && columna != (columnas -1)){

//Si estamos en la primera fila y no estamos en las esquinas, se puede eliminar a izquierda, derecha, debajo, diagonal inferior izquierda y diagonal inferior derecha

poner((columnas + columna - 1), 0 , poner((columnas + columna + 1), 0 , poner((columnas + columna), 0 , poner((columna + 1), 0 , poner((columna - 1),0,poner(columna,0,tablero,Nil),Nil), Nil), Nil), Nil), Nil)

}

else if(fila == (filas - 1) && columna != 0 && columna != (columnas -1)){

//Si estamos en la ultima fila y no estamos en las esquinas, se puede eliminar a izquierda, derecha, debajo, diagonal superior izquierda y diagonal superior derecha

poner((((fila -1 ) \* columnas) + columna ),0,poner((((fila -1 ) \* columnas) + (columna + 1)),0,poner((((fila -1 ) \* columnas) + (columna - 1)),0,poner(((fila \* columnas) + (columna - 1)),0,poner(((fila \* columnas) + (columna + 1)),0,poner(((fila \* columnas) + columna),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil)

}

else if(columna == 0 && fila != 0 && fila != (filas - 1)){

//Si estamos en la columna izquierda y no estamos en las esquinas, eliminamos arriba, abajo, derecha, diagonal superior derecha y diagonal inferior derecha

poner(((fila - 1) \* (columnas + 1)),0,poner(((fila + 1) \* (columnas + 1)),0,poner(((fila + 1) \* columnas),0,poner(((fila - 1) \* columnas),0,poner(((fila \* columnas) + 1),0,poner((fila \* columnas),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil)

}

else if(columna == (columnas - 1) && fila != 0 && fila != filas -1){

//Si estamos en la columna derecha y no estamos en las esquinas, eliminamos arriba, abajo, izquierda, diagonal superior izquierda y diagonal inferior izquierda

poner((((fila - 1) + columnas) + (columna - 1)),0,poner((((fila + 1) + columnas) + (columna - 1)),0,poner((((fila + 1) + columnas) + columna),0,poner((((fila - 1) + columnas) + columna),0,poner(((fila + columnas) + (columna -1)),0,poner(((fila + columnas) + columna),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil)

}

else{

//Si estamos en cualquier otra posicion, eliminamos la matriz de 3x3

poner((((fila + 1 ) \* columnas) + (columna + 1)),0,poner((((fila + 1 ) \* columnas) + (columna -1)),0,poner((((fila + 1 ) \* columnas) + (columna)),0,poner((((fila - 1 ) \* columnas) + (columna - 1)),0,poner((((fila - 1 ) \* columnas) + (columna + 1)),0,poner((((fila - 1 ) \* columnas) + (columna)),0,poner(((fila \* columnas) + (columna + 1)),0,poner(((fila \* columnas) + (columna + 1)),0,poner(((fila \* columnas) + columna),0,tablero,Nil),Nil),Nil),Nil),Nil),Nil),Nil),Nil),Nil)

}

}

//Eliminamos todos los bloques del mismo color que se haya pasado como parametro

def bombaColores(tablero:List[Int], bloque:Int, pos:Int, tam:Int): List[Int] ={

if(pos == tam) { //Si estamos en el final devolvemos el tablero

return tablero

}

else{

//Si no, comprobamos que la posicion en la que estamos es igual al bloque que queremos eliminar

if(devuelveBloque(tablero, pos) == bloque){

//Si es igual, ponemos un 0 en esa posicion y llamamos al metodo con pos + 1

bombaColores(poner(pos, 0, tablero, Nil), bloque, pos + 1, tam)

}

else{

//Si no es igual, pasamos esa posicion sin modificarla

bombaColores(tablero, bloque, pos + 1, tam)

}

}

}

//Pone la bomba que se le pasa por parametro en el tablero

def ponerBomba(tablero:List[Int], fila:Int, columna:Int, columnas:Int, bomba:Int): List[Int]={

return poner((fila \* columnas) + columna, bomba,tablero,Nil)

}

def imprimirLetra(bloque:Int): String ={

bloque match{

case 1 => return "azules"

case 2 => return "rojos"

case 3 => return "naranjas"

case 4 => return "verdes"

case 5 => return "plateados"

case 6 => return "morados"

case 7 => return "a"

case 8 => return "b"

case 9 => return "c"

}

}

//Comprueba piezas es el método encargado de llamar a los metodos compruebaAbajo,compruebaArriba, compruebaDerecha y compruebaIzquierda en función de la posición en la que nos

//encontremos

//También comprueba si el bloque que se le ha pasado es una bomba y llama a la funcíón que corresponda en cada caso

def compruebaPiezas(tablero:List[Int], columna:Int, fila:Int, filas:Int,columnas:Int, anterior:Int, dificultad:Int): List[Int] ={

//bombas

if (anterior == 7){

val r1 = new scala.util.Random(System.currentTimeMillis())

if((r1.nextInt(5000) + 1) < 2500){

bombaFila(tablero,columnas,fila,0)

}

else{

bombaColumna(tablero,columnas,filas,0,columna)

}

}

else if(anterior == 8){

bombaTNT(tablero, columnas, filas, columna, fila)

}

else if(anterior == 9){

val r1 = new scala.util.Random(System.currentTimeMillis())

dificultad match{

case 1 => val bloque = r1.nextInt(4) + 1; bombaColores(poner(fila\*columnas + columna, 0, tablero, Nil), bloque , 0, 63);

case 2 => val bloque = r1.nextInt(5) + 1; bombaColores(poner(fila\*columnas + columna, 0, tablero, Nil), bloque, 0, 187);

case 3 => val bloque = r1.nextInt(6) + 1; bombaColores(poner(fila\*columnas + columna, 0, tablero, Nil), bloque, 0, 405);

}

}

//Esquina superior izquierda

else {

if(fila == 0 && columna == 0){

compruebaAbajo(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

//Esquina superior derecha

else if(fila == 0 && columna == (columnas - 1)){

compruebaAbajo(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

//Esquina inferior izquierda

else if(fila == (filas - 1) && columna == 0){

compruebaArriba(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

//Esquina inferior derecha

else if(fila == (filas - 1) && columna == (columnas - 1)){

compruebaArriba(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(fila == 0){

compruebaAbajo(anterior, compruebaIzquierda(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(fila == (filas - 1)){

compruebaArriba(anterior, compruebaIzquierda(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(columna == 0){

compruebaAbajo(anterior, compruebaArriba(anterior, compruebaDerecha(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else if(columna == (columnas - 1)){

compruebaAbajo(anterior, compruebaArriba(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

else{

compruebaDerecha(anterior, compruebaAbajo(anterior, compruebaArriba(anterior, compruebaIzquierda(anterior, tablero, fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas), fila, columna, columnas, filas)

}

}

}

//Creación ventana Dificultad

def ventanaDificultad(){

val VentanaDificultad = new JFrame

VentanaDificultad.setSize(500,300) //Anchura,Altura

//Hacemos visible el jframe y lo centramos en la pantalla

VentanaDificultad.setVisible(true)

VentanaDificultad.setLocationRelativeTo(null)

//Todo este código hace que el titulo quede centrado en la ventana

VentanaDificultad.setTitle("\*-\*-\*- BIENVENIDO A TOY BLAST -\*-\*-\*")

val currentTitle = VentanaDificultad.getTitle().trim()

val font = VentanaDificultad.getFont()

val fm = VentanaDificultad.getFontMetrics(font)

val frameWidth = VentanaDificultad.getWidth()

val titleWidth = fm.stringWidth(currentTitle)

val spaceWidth = fm.stringWidth(" ");

val centerPos = (frameWidth / 2) - (titleWidth / 2)

val spaceCount = centerPos / spaceWidth

val pad = String.format("%" + (spaceCount - 14) + "s", "")

VentanaDificultad.setTitle(pad + currentTitle)

VentanaDificultad.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)

//Añadimos los tres botones para que en función del que pulsemos, se pasen unas variables de filas, columnas y dificultad

val Nivel1 = new JButton

Nivel1.setText("Nivel 1");

Nivel1.addActionListener(ActionListener => {

VentanaDificultad.setVisible(false); //Escondemos el frame

VentanaDificultad.dispose(); //Elimina el frame

creacionTableroInicial(7, 9, 1) //Creamos un tablero de 7 filas y 9 columnas

})

val Nivel2 = new JButton

Nivel2.setText("Nivel 2");

Nivel2.addActionListener(ActionListener => {

VentanaDificultad.setVisible(false); //you can't see me!

VentanaDificultad.dispose(); //Elimina el frame

creacionTableroInicial(11, 17, 2) //Creamos un tablero de 11 filas y 17 columnas

})

val Nivel3 = new JButton

Nivel3.setText("Nivel 3");

Nivel3.addActionListener(ActionListener => {

VentanaDificultad.setVisible(false); //you can't see me!

VentanaDificultad.dispose(); //Elimina el frame

creacionTableroInicial(15, 27,3) //Creamos un tablero de 15 filas y 27 columnas

})

val Label = new javax.swing.JLabel();

//Código para créacion de etiqueta y layout donde se van a colocar los botones, no tiene ninguna funcionalidad final, meramente estético

Label.setText("Seleccione el nivel de dificultad:");

val layout = new javax.swing.GroupLayout(VentanaDificultad.getContentPane());

VentanaDificultad.getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addComponent(Nivel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 115, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 72, Short.MAX\_VALUE)

.addComponent(Nivel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 115, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(63, 63, 63)

.addComponent(Nivel3, javax.swing.GroupLayout.PREFERRED\_SIZE, 115, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addComponent(Label, javax.swing.GroupLayout.PREFERRED\_SIZE, 277, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(0, 0, Short.MAX\_VALUE)))

.addContainerGap())

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(30, 30, 30)

.addComponent(Label, javax.swing.GroupLayout.PREFERRED\_SIZE, 32, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(59, 59, 59)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(Nivel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 45, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Nivel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 45, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Nivel3, javax.swing.GroupLayout.PREFERRED\_SIZE, 45, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addContainerGap(62, Short.MAX\_VALUE))

);

VentanaDificultad.pack()

}

//Generamos un tablero de enteros para ir haciendo nuestras comparaciones más adelante

def generarTablero(tam:Int,tablero:List[Int],bloques:Int):List[Int]={

if(tam==0) tablero

else{

val x : List[Int]= tablero:::List(Random.nextInt(bloques)+1)

generarTablero(tam-1,x,bloques)

}

}

//Devuelve el bloque en la posicion pasada por parametro

def devuelveBloque(tablero:List[Int],pos:Int):Int={

if(pos == 0) tablero.head

else devuelveBloque(tablero.tail,pos-1)

}

//Este método recorre la lista de botones que vamos a generar más adelante y los añade al frame que pasamos por parametro

def anadeBotones(tablero:List[JButton], frame:JFrame){

if(!tablero.isEmpty){

frame.add(tablero.head)

anadeBotones(tablero.tail, frame)

}

}

//Pone el item n en la posicion pos de la lista l

def poner(pos:Int,n:Int,l:List[Int],laux:List[Int]):List[Int]={

if(pos==0) laux:::List[Int](n):::l.tail

else poner(pos-1,n,l.tail,laux:::List[Int](l.head))

}

def intercambiarCasilla(pos1: Int, pos2: Int, lista: List[Int]): List[Int] = {

//Dado que hemos intentado ahorrar todos los val posible, vamos dejar comentado el código de manera más extensa para que se entienda mejor

/\*

val ficha1 = devuelveBloque(lista, pos1)

val ficha2 = devuelveBloque(lista, pos2)

val tableroAux = poner(pos1, ficha2, lista, Nil)

poner(pos2,ficha1,tableroAux,Nil)

Lo que hemos hecho ha sido ir sustituyendo la parte derecha de la igualacion en las llamadas a funciones que las prosiguen, en este caso quedaría así paso a paso:

1) Dado que ficha1 = devuelveBloque(lista,pos1) -> en poner(pos2, ficha1,tableroAux,Nil) sustituimos, quedando así -> poner(pos2,devuelveBloque(lista, pos1),tableroAux,Nil)

2) tableroAux = poner(pos1, ficha2, lista, Nil) -> sustuituimos ficha2 por su igualación -> poner(pos1, devuelveBloque(lista, pos2), lista, Nil)

3) Por ultimo, en poner(pos2,devuelveBloque(lista, pos1),tableroAux,Nil), sustituimos tableroAux por lo de arriba -> poner(pos1, devuelveBloque(lista, pos2), lista, Nil)

-> poner(pos2,devuelveBloque(lista, pos1),poner(pos1, devuelveBloque(lista, pos2), lista, Nil) ,Nil)

\*/

poner(pos2,devuelveBloque(lista,pos1),poner(pos1,devuelveBloque(lista,pos2),lista,Nil),Nil)

}

def subirCeros(pos:Int,col:Int,l:List[Int]):List[Int]={

if (pos==0) l

else if (devuelveBloque(l, pos) == 0) subirCerosAux(pos,pos-col,col,l)

else subirCeros(pos-1,col,l)

}

def subirCerosAux(pos:Int,pos2:Int,col:Int,l:List[Int]):List[Int]={

if(pos<=col || pos2<1)subirCeros(pos-1,col,l)

else if(devuelveBloque(l, pos2) == 0) subirCerosAux(pos,pos2-col,col,l)

else subirCeros(pos-1,col,intercambiarCasilla(pos,pos2,l))

}

//Sustituye los ceros por piezas en funcion de la dificultad

def rellenarTablero(tablero:List[Int], tam:Int, pos:Int, bloques:Int):List[Int] ={

//Si ya hemos recorrido todo el tablero, devolvemos el mismo

if(tam == pos) tablero

else{

//Si no, comprobamos si la posicion en la que nos encontramos es igual a 0

if(devuelveBloque(tablero, pos) == 0){

//Si es igual a cero, ponemos el bloque correspondiente en esa posicion y llamamos a rellenar con pos+1, el código extendido quedaría así

/\*

val bloqueNuevo = Random.nextInt(bloques)+1

val tableroAux = poner(pos,bloqueNuevo, tablero, Nil)

rellenarTablero(tableroAux, tam, pos + 1, bloques)

\*/

rellenarTablero(poner(pos, Random.nextInt(bloques)+1, tablero, Nil), tam, (pos + 1), bloques)

}

else{

//Si no es igual a cero, pasamos esa posición sin modificarla

rellenarTablero(tablero, tam, pos+1, bloques)

}

}

}

//Cuenta los ceros del tablero

def compruebaPuntuacion(tablero:List[Int]): Int = {

if(tablero.isEmpty) return 0

else{

if(tablero.head == 0){

return 1 + compruebaPuntuacion(tablero.tail)

}

else{

return 0 + compruebaPuntuacion(tablero.tail)

}

}

}

//Crea la lista de botones en funcion de la dificultad

def creaBotones(dificultad:Int, tablero:List[JButton],filas:Int,columnas:Int,fila:Int,columna:Int, frame:JFrame, listaNumeros:List[Int], puntuacion:List[Int]): List[JButton]={

if(fila<filas){

if(columna<columnas){

val b = new JButton

//Los botones miden 60 pixeles por lado, por lo que su desplazamiento va a ser fila \* 60px en vertical y columna \* 60px en horizontal

b.setSize(60, 60)

b.setLocation(columna\*60, fila\*60)

b.setOpaque(true)

//Comprobamos el bloque del tablero para el que estamos creando el botón, y en función del mismo le damos un color u otro

devuelveBloque(listaNumeros, (fila \* columnas) + columna) match{

case 1 => {

b.setBackground(new java.awt.Color(0, 51, 255));

b.setForeground(new java.awt.Color(255, 255, 255));

}

case 2 => {

b.setBackground(new java.awt.Color(255, 0, 0));

b.setForeground(new java.awt.Color(255, 255, 255));

}

case 3 => {

b.setBackground(new java.awt.Color(255, 153, 51));

b.setForeground(new java.awt.Color(255, 255, 255));

}

case 4 => {

b.setBackground(new java.awt.Color(102, 255, 0));

b.setForeground(new java.awt.Color(255, 255, 255));

}

case 5 => {

b.setBackground(new java.awt.Color(204, 218, 222));

b.setForeground(new java.awt.Color(0, 0, 0));

}

case 6 => {

b.setBackground(new java.awt.Color(153, 30, 225));

b.setForeground(new java.awt.Color(255, 255, 255));

}

case 7 => {

b.setBackground(new java.awt.Color(255, 255, 102));

b.setForeground(new java.awt.Color(0, 0, 0));

b.setText("ven")

}

case 8 => {

b.setBackground(new java.awt.Color(255, 255, 102));

b.setForeground(new java.awt.Color(0, 0, 0));

b.setText("tnt")

}

case 9 => {

b.setBackground(new java.awt.Color(255, 255, 102));

b.setForeground(new java.awt.Color(0, 0, 0));

b.setText("rub")

}

}

//Añadimos la acción del botón en función de la dificultad para trabajar con los rangos que haya en cada lista

b.addActionListener(ActionListener => {

val tableroAux = compruebaPiezas(listaNumeros, columna, fila, filas, columnas, devuelveBloque(listaNumeros, (fila \* columnas) + columna), dificultad)

dificultad match{

case 1 =>{

if(compruebaPuntuacion(tableroAux) == 5 && devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7) {

val tableroAux2 = ponerBomba(tableroAux, fila, columna, columnas, 7)

val tableroAux3 = subirCeros(62, 9, tableroAux2) //N piezas, columnas, tablero que se modifica

val tableroAux4 = rellenarTablero(tableroAux3, 63, 0, 4)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else if(compruebaPuntuacion(tableroAux) == 6 && devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = ponerBomba(tableroAux, fila, columna, columnas, 8)

val tableroAux3 = subirCeros(62, 9, tableroAux2) //N piezas, columnas, tablero que se modifica

val tableroAux4 = rellenarTablero(tableroAux3, 63, 0, 4)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else if(compruebaPuntuacion(tableroAux) > 7 &&devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = ponerBomba(tableroAux, fila, columna, columnas, 9)

val tableroAux3 = subirCeros(62, 9, tableroAux2) //N piezas, columnas, tablero que se modifica

val tableroAux4 = rellenarTablero(tableroAux3, 63, 0, 4)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else if(devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7) {

val tableroAux3 = subirCeros(62, 9, tableroAux) //N piezas, columnas, tablero que se modifica

val tableroAux4 = rellenarTablero(tableroAux3, 63, 0, 4)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else{

val tableroAux3 = subirCeros(62, 9, tableroAux) //N piezas, columnas, tablero que se modifica

val tableroAux4 = rellenarTablero(tableroAux3, 63, 0, 4)

frame.setVisible(false)

frame.dispose()

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacion)

}

}

case 2 =>

if(compruebaPuntuacion(tableroAux) == 5 && devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = ponerBomba(tableroAux, fila, columna, columnas, 7)

val tableroAux3 = subirCeros(186, columnas, tableroAux2)

val tableroAux4 = rellenarTablero(tableroAux3, 187, 0, 5)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else if(compruebaPuntuacion(tableroAux) == 6 && devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = ponerBomba(tableroAux, fila, columna, columnas, 8)

val tableroAux3 = subirCeros(186, columnas, tableroAux2)

val tableroAux4 = rellenarTablero(tableroAux3, 187, 0, 5)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else if(compruebaPuntuacion(tableroAux) > 7 && devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = ponerBomba(tableroAux, fila, columna, columnas, 9)

val tableroAux3 = subirCeros(186, columnas, tableroAux2)

val tableroAux4 = rellenarTablero(tableroAux3, 187, 0, 5)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else if(devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = subirCeros(186, columnas, tableroAux)

val tableroAux3 = rellenarTablero(tableroAux2, 187, 0, 5)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux3, puntuacionNueva)

}

else{

val tableroAux2 = subirCeros(186, columnas, tableroAux)

val tableroAux3 = rellenarTablero(tableroAux2, 187, 0, 5)

frame.setVisible(false)

frame.dispose()

creacionTableroJugada(filas, columnas, dificultad, tableroAux3, puntuacion)

}

case 3 =>{

if(compruebaPuntuacion(tableroAux) == 5 && devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = ponerBomba(tableroAux, fila, columna, columnas, 7)

val tableroAux3 = subirCeros(404, columnas, tableroAux2)

val tableroAux4 = rellenarTablero(tableroAux3, 405, 0, 6)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else if(compruebaPuntuacion(tableroAux) == 6 && devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = ponerBomba(tableroAux, fila, columna, columnas, 8)

val tableroAux3 = subirCeros(404, columnas, tableroAux2)

val tableroAux4 = rellenarTablero(tableroAux3, 405, 0, 6)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else if(compruebaPuntuacion(tableroAux) > 7 && devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = ponerBomba(tableroAux, fila, columna, columnas, 9)

val tableroAux3 = subirCeros(404, columnas, tableroAux2)

val tableroAux4 = rellenarTablero(tableroAux3, 405, 0, 6)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux4, puntuacionNueva)

}

else if(devuelveBloque(listaNumeros,(fila \* columnas) + columna) < 7){

val tableroAux2 = subirCeros(404, columnas, tableroAux)

val tableroAux3 = rellenarTablero(tableroAux2, 405, 0, 6)

frame.setVisible(false)

frame.dispose()

val puntosJugada = compruebaPuntuacion(tableroAux)

val puntosAnteriores = devuelveBloque(puntuacion,devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1)

val puntuacionNueva = poner(devuelveBloque(listaNumeros,(fila \* columnas) + columna) - 1,puntosJugada + puntosAnteriores,puntuacion,Nil)

creacionTableroJugada(filas, columnas, dificultad, tableroAux3, puntuacionNueva)

}

else{

val tableroAux2 = subirCeros(404, columnas, tableroAux)

val tableroAux3 = rellenarTablero(tableroAux2, 405, 0, 6)

frame.setVisible(false)

frame.dispose()

creacionTableroJugada(filas, columnas, dificultad, tableroAux3, puntuacion)

}

}

}

})

creaBotones(dificultad,b::tablero,filas,columnas,fila,columna+1, frame,listaNumeros,puntuacion)

}

else creaBotones(dificultad,tablero,filas,columnas,fila+1,0,frame,listaNumeros,puntuacion)

}

else{

tablero

}

}

//Primera ventana que se muestra tras seleccionar el nivel de dificultad, hemos decidido realizarlo en dos ventanas ya que en el segundo metodo necesitamos comprobar puntuacion etc

def creacionTableroInicial(fil:Int, col:Int, dificultad:Int){

//creamos la ventana que contendrá el tablero

val frame = new JFrame

//Le damos tamaño y lo centramos en pantalla

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)

frame.setSize((col+1)\*60,(fil+1)\*60)

frame.setVisible(true)

frame.setLayout(null);

frame.setLocationRelativeTo(null)

frame.setTitle("NIVEL " + dificultad.toString())

//Añadimos los botones en función del nivel de dificultad

if(dificultad == 1){

val botones = creaBotones(dificultad,Nil,fil,col,0,0,frame, generarTablero(63, Nil, 4), List(0,0,0,0))

anadeBotones(botones, frame)

}

else if(dificultad == 2){

val botones = creaBotones(dificultad,Nil,fil,col,0,0,frame, generarTablero(187, Nil, 5),List(0,0,0,0,0))

anadeBotones(botones, frame)

}

else{

val botones = creaBotones(dificultad,Nil,fil,col,0,0,frame, generarTablero(405, Nil, 6),List(0,0,0,0,0,0))

anadeBotones(botones, frame)

}

}

//Cuando se pulsa cualquier botón del tablero, se llama a este método, en el que se comprueba si se ha ganado o no la partida, si se gana, muestra un aviso y cierra la aplicacion

//Si no, vuelve a crear otro tablero con la nueva disposicion de las piezas y muestra la puntuacion que llevas hasta el momento

def creacionTableroJugada(fil:Int, col:Int, dificultad:Int, tablero:List[Int],puntuacion:List[Int]){

if(dificultad == 1){

if(devuelveBloque(puntuacion,0) >= 20 && devuelveBloque(puntuacion,1) >= 20 && devuelveBloque(puntuacion,2) >= 20 && devuelveBloque(puntuacion,3) >= 20){

JOptionPane.showMessageDialog(null,"¡HAS GANADO!\n" +

"BLOQUES AZULES: " + devuelveBloque(puntuacion,0) + "\n" +

"BLOQUES ROJOS: " + devuelveBloque(puntuacion,1) + "\n" +

"BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2) + "\n" +

"BLOQUES VERDES: " + devuelveBloque(puntuacion,3)+ "\n" )

System.exit(0)

}

else{

val frame = new JFrame

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)

frame.setSize((col+1)\*60,(fil+1)\*60)

frame.setVisible(true)

frame.setTitle("NIVEL " + dificultad.toString())

frame.setLayout(null);

frame.setLocationRelativeTo(null)

val botones = creaBotones(dificultad,Nil,fil,col,0,0,frame, tablero, puntuacion)

anadeBotones(botones, frame)

JOptionPane.showMessageDialog(null,"PUNTUACIÓN\n" +

"BLOQUES AZULES: " + devuelveBloque(puntuacion,0) + "\n" +

"BLOQUES ROJOS: " + devuelveBloque(puntuacion,1) + "\n" +

"BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2) + "\n" +

"BLOQUES VERDES: " + devuelveBloque(puntuacion,3)+ "\n" )

}

}

else if(dificultad == 2){

if(devuelveBloque(puntuacion,0) >= 15 && devuelveBloque(puntuacion,1) >= 15 && devuelveBloque(puntuacion,2) >= 15 && devuelveBloque(puntuacion,3) >= 15 && devuelveBloque(puntuacion,4) >= 15){

JOptionPane.showMessageDialog(null,"¡HAS GANADO!\n" +

"BLOQUES AZULES: " + devuelveBloque(puntuacion,0) + "\n" +

"BLOQUES ROJOS: " + devuelveBloque(puntuacion,1) + "\n" +

"BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2) + "\n" +

"BLOQUES VERDES: " + devuelveBloque(puntuacion,3)+ "\n" +

"BLOQUES PLATEADOS: " + devuelveBloque(puntuacion,4) )

System.exit(0)

}

else{

val frame = new JFrame

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)

frame.setSize((col+1)\*60,(fil+1)\*60)

frame.setVisible(true)

frame.setTitle("NIVEL " + dificultad.toString())

frame.setLayout(null);

frame.setLocationRelativeTo(null)

val botones = creaBotones(dificultad,Nil,fil,col,0,0,frame, tablero, puntuacion)

anadeBotones(botones, frame)

JOptionPane.showMessageDialog(null,"PUNTUACIÓN\n" +

"BLOQUES AZULES: " + devuelveBloque(puntuacion,0) + "\n" +

"BLOQUES ROJOS: " + devuelveBloque(puntuacion,1) + "\n" +

"BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2) + "\n" +

"BLOQUES VERDES: " + devuelveBloque(puntuacion,3)+ "\n" +

"BLOQUES PLATEADOS: " + devuelveBloque(puntuacion,4) )

}

}

else{

if(devuelveBloque(puntuacion,0) >= 10 && devuelveBloque(puntuacion,1) >= 10 && devuelveBloque(puntuacion,2) >= 10 && devuelveBloque(puntuacion,3) >= 10 && devuelveBloque(puntuacion,4) >= 10 && devuelveBloque(puntuacion,5) >= 10){

JOptionPane.showMessageDialog(null,"¡HAS GANADO!\n" +

"BLOQUES AZULES: " + devuelveBloque(puntuacion,0) + "\n" +

"BLOQUES ROJOS: " + devuelveBloque(puntuacion,1) + "\n" +

"BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2) + "\n" +

"BLOQUES VERDES: " + devuelveBloque(puntuacion,3)+ "\n" +

"BLOQUES PLATEADOS: " + devuelveBloque(puntuacion,4) + "\n" +

"BLOQUES MORADOS: " + devuelveBloque(puntuacion,5))

System.exit(0)

}

else{

val frame = new JFrame

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)

frame.setSize((col+1)\*60,(fil+1)\*60)

frame.setVisible(true)

frame.setTitle("NIVEL " + dificultad.toString())

frame.setLayout(null);

frame.setLocationRelativeTo(null)

val botones = creaBotones(dificultad,Nil,fil,col,0,0,frame, tablero, puntuacion)

anadeBotones(botones, frame)

JOptionPane.showMessageDialog(null,"PUNTUACIÓN\n" +

"BLOQUES AZULES: " + devuelveBloque(puntuacion,0) + "\n" +

"BLOQUES ROJOS: " + devuelveBloque(puntuacion,1) + "\n" +

"BLOQUES NARANJAS: " + devuelveBloque(puntuacion,2) + "\n" +

"BLOQUES VERDES: " + devuelveBloque(puntuacion,3)+ "\n" +

"BLOQUES PLATEADOS: " + devuelveBloque(puntuacion,4) + "\n" +

"BLOQUES MORADOS: " + devuelveBloque(puntuacion,5))

}

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* AQUÍ EMPIEZA LA EJECUCÓN DEL PROGRAMA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ventanaDificultad()

}