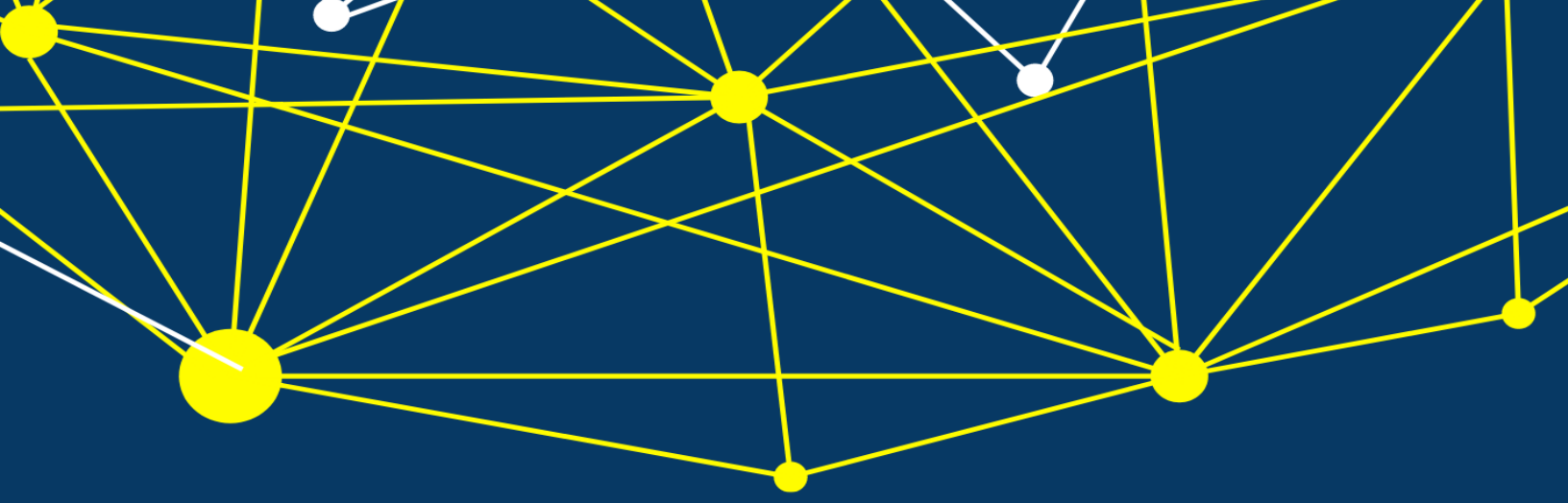

Paradigmas Avanzados de Programación, 3ºGII

Daniel de Heras Zorita

Adrián Borges Cano

PECL2: Scala

ÍNDICE

Contenido

Introducción

Este documento ha sido creado para facilitar la explicación del programa desarrollado en Scala, un lenguaje de programación funcional (basado principalmente en la recursión) y orientado a objetos (permite dividir los elementos partícipes de la lógica del programa en distintas clases).

La práctica trata sobre la realización de un juego llamado 'Cundy Crosh Soga', una adaptación al famoso juego de Smartphone 'Candy Crush Saga'. Este trabajo llevado a cabo por Daniel de Heras Zorita y Adrián Borges Cano, corresponde a la PECL2 de Paradigmas Avanzados de Programación, de 3ºGII de la UAH, cuya entrega se ha realizado en abril de 2023. En este documento explicaremos las distintas versiones que hemos debido generar para cumplir con lo requerido, al igual que una descripción clara sobre el funcionamiento de cada una de las funciones realizadas, en caso de que el lector no sepa cuál es su propósito y quiera conocerlo. Finalmente, haremos una breve pasada por la ejecución del programa en sus distintas versiones. Sin más dilación, comencemos a adentrarnos en el proyecto.

Versiones del Proyecto

Para el desarrollo de la práctica hemos tenido que dividir lo desarrollado en distintas partes, donde cada una de ellas es una evolución de la anterior. A continuación, haremos un breve resumen sobre cada una de ellas, indicando al mismo tiempo qué componentes han sido añadidos con respecto a su versión anterior.

Implementación básica

Esta primera versión de la práctica contiene, en resumidas cuentas, lo necesario para que sea posible la ejecución del famoso juego de caramelos. Todos los requisitos principales comentados en el enunciado están contenidos en este proyecto, y como se puede imaginar, dichos componentes del juego van a ser arrastrados durante el desarrollo de las nuevas versiones. Algunas de las funciones realizadas para abarcar la lógica del programa son: 'activarBomba', 'reemplazarElemento', 'longitudLista', 'eliminarElementosFila'… entre muchas otras. Sin embargo, la explicación completa de estas funciones se puede encontrar en su apartado correspondiente, después de esta explicación.

Además de este conjunto de métodos que permiten ejecutar correctamente el programa, también encontramos una interfaz simple pero intuitiva, donde cada caramelo se muestra con su respectivo color, que permite al usuario poder interactuar fácilmente con el juego. Igualmente, todos estos detalles sobre la impresión de datos para el usuario se mostrarán posteriormente.

Implementación optimizada

En este apartado encontramos (como ya hemos repetido en numerosas ocasiones) todo lo realizado en la implementación básica, con la diferencia de que ahora añadimos un método extra. En caso de que el usuario quiera ejecutar el programa automáticamente, ahora el juego va a ser el encargado de elegir la mejor opción en todo momento; lo que equivale a elegir en cada turno la opción que elimine mayor número de bloques. Dicha ejecución

automática será mostrada en todo momento al usuario paso a paso, de forma que este último sea capaz de comprender las decisiones que va tomando el programa.

Implementación gráfica (extra)

Sumando las características que encontrábamos en los dos puntos anteriores, en esta última versión del trabajo encontramos que dicho proyecto realizado se encuentra expandido por una interfaz gráfica programada con Java Swing, que provoca una interacción con el usuario aún más amigable, no solo haciéndole posible usar la aplicación con comodidad, sino que esta última es tan atractiva que le va a incitar a usarla. Para dicha interfaz, encontramos el uso de 3 clases, una por cada ventana de la interfaz: 'ventanaInicial', 'ventanaDatos' y 'ventanaMatriz'. Estas ventanas van siendo llamadas unas a otras progresivamente hasta llegar a última, que permite jugar al usuario.

Métodos Implementados

En este apartado, enunciaremos todos los métodos realizados durante la práctica que permiten la ejecución del juego. Antes de continuar, debemos aclarar que, en todos aquellos casos en los que lo hemos necesitado, hemos hecho uso de la **recursividad de cola**: una forma de realizar recursividad en la que nos permite hacer cuantas iteraciones queramos sin agotar la pila de llamadas en memoria, puesto que el resultado que queremos obtener (por ejemplo, en obtener la suma de elementos en una lista) va implícito en uno de los parámetros de la llamada recursiva, en vez de encontrarse fuera de la llamada como se suele hacer.

Por otro lado, debemos comentar que, debido a los requerimientos del enunciado de la práctica, en vez de utilizar un array de 2 dimensiones para representar el tablero, hemos hecho uso de una lista cuya longitud es la multiplicación del número de filas por el número de columnas. Consecuentemente, cuando queremos acceder a cualquier posición de dicho tablero, utilizamos la fórmula que usamos en el pasado para acceder a las matrices linealizadas. Para hallar la posición 'posicion', siendo la posición de la fila en el tablero 'posFila', y la posición de la columna 'posColumna', conociendo que el número de columnas totales que tiene la matriz son 'numColumnas', la fórmula queda de la forma:

```
posicion = posFila * numColumnas + posColumna
```

Figura 1: Fórmula para acceder a una posición del tablero en forma de lista

Ahora, sin más preámbulos, realizaremos la explicación de los métodos desarrollados:

- rellenarTablero: recorre el tablero y sustituye los elementos que tengan valor igual a 0 por un número aleatorio equivalente al número de caramelos asignado, de forma que cada vez que se eliminen bloques, se pueda llamar a esta función. En adición, al iniciar el juego se crea una lista de ceros que tenga el número de elementos igual a filas*columnas, para poder llamar a este método y asegurar que la inicialización del tablero en cada ejecución del código sea siempre distinta.
- activarBomba: sustituir todos los elementos de su fila o columna (que en la ejecución del programa elegirá una de estas dos opciones de forma aleatoria) por ceros. Esta función hace uso de otras dos funciones auxiliares: una para la fila (eliminarElementosFila), y otra para la columna (activarBombaColumna).

- eliminarElementosFila: sobrescribir a cero todos los elementos dando el rango de posiciones de la fila indicada del tablero. Este método se usa en 'activarBomba', donde elimina todos los elementos que se encuentren en la fila. Es decir, dada la fila en cuestión, eliminar el número de elementos equivalente al número de columnas, porque es lo que delimita el ancho de la matriz. También se usa en 'activarTNT', aunque será explicado posteriormente.
- activarBombaColumna: sustituir por cero todos los elementos del tablero que se encuentren en la columna indicada. Para ello, reemplaza el elemento que se encuentre en la posición equivalente al número de la columna que se quiera borrar. Después, a dicha posición se le suma el número de columnas totales, lo que nos permite desplazarnos hasta la siguiente fila, y donde borraremos el elemento que se encuentre en la columna que deseamos borrar. Esto se realiza recursivamente hasta que el índice que se está usando para eliminar los elementos sea mayor que el tamaño de la matriz, en cuyo caso hace finalizar la función.

  A continuación, ejemplificaremos lo anterior para facilitar la explicación:
  Supongamos que tenemos una lista de longitud 25, lo que equivale a un tablero de 5x5, y queremos activar la bomba para la columna 3. Como las listas en Scala comienzan por el número 0, entonces en realidad queremos borrar la columna 2. Por lo tanto, en la primera iteración sobrescribimos a 0 la posición 2 de la lista. Acto seguido, como el número de columnas del tablero es 5, y 2+5=7, la siguiente posición que eliminaremos será la 7. Visualmente, esto equivale a eliminar el primer y segundo elemento de la columna. La iteración recursiva continuará de la siguiente forma: 7+5=12 → 12+5=17 → 17+5=22 → 22+5=27.
  Por lo tanto, lo que estaremos haciendo en este método será eliminar las posiciones 2, 7, 12, 17 y 22, que equivalen a las posiciones de la columna 3 de la matriz. Cuando llegamos a 27, no eliminamos nada, sino que la ejecución del programa se detiene porque es mayor que el tamaño de la lista tablero (27>25).
- reemplazarElemento: dado el índice del elemento que se quiere reemplazar, el nuevo elemento que se quiere escribir, y el propio tablero en el que se quiere realizar esta operación, recorrer este último secuencialmente hasta llegar al índice querido, donde se indica el nuevo valor que tomará dicha posición, y se le concatena el resto de la lista sin modificar.
- longitudLista: devolver el número de elementos de la lista introducida como parámetro de entrada.
- activarRompecabezas: sustituir por cero todas las apariciones de un determinado número (color de caramelo) en el tablero. Dicho tablero se recorre secuencialmente. En caso de que la posición evaluada sea igual al caramelo que se quiere eliminar, se sobrescribe a cero, de lo contrario, mantiene el valor original.
- activarTNT: sobrescribir a cero todos bloques que se encuentren en un radio de 4 elementos del bloque original en el que se activa el TNT. Para poder conseguir esto, se utiliza una función auxiliar llamada 'activarTNT_Aux'. 'activarTNT' se encarga de calcular cuales son los índices de comienzo y fin de las filas que se van a ver afectadas por la explosión, y también calcula cual es el rango de elementos de cada una de estas filas que va a sufrir la explosión. Después, dependiendo de los valores

obtenidos en los índices del rango de elementos de cada fila, llamaremos de una forma u otra a la función auxiliar. Esto se hace para no borrar posiciones de la matriz que no correspondan a su fila correspondiente. Por ejemplo, si la posición en la que queremos activar el TNT se encuentra en la 2ª columna de la matriz (columna 1), al después aplicar el rango de índices que deben ser eliminados, obtendremos que la primera posición que se debe eliminar de la fila será 1-4=-3. Si permitimos que elimine la posición -3 con respecto al origen, acabará eliminando elementos de otras filas que no queremos borrar. Por tanto, en ese caso, se llamará a la función auxiliar indicando que la primera posición a eliminar es 0.

- activarTNT_Aux: iterar y sobrescribir a 0 en el rango de posiciones de todas las filas indicadas. Como se ha explicado anteriormente, el método principal se encarga de que los rangos de elementos a borrar dentro de la fila sean válidos, y no se salgan de los límites de la matriz. Bien, en este método realizaremos estas mismas comprobaciones, pero en este caso con respecto a las filas que eliminar. Por ejemplo, si el elemento que queremos eliminar se encuentra en la 3ª fila (fila 2), cuando indiquemos qué filas debe borrar, obtendremos que la primera de ellas será la fila 2-4=-2, fila que no existe. Por tanto, en ese caso, avanzaremos el índice inicial de filas a borrar hasta llegar a 0, en cuyo caso ya podremos comenzar a eliminar elementos.
  A continuación, mostraremos cómo quedaría una supuesta matriz en la que se aplica esto anterior. El número en verde indica dónde se ha activado el TNT.

0	0	0	0	0	0	0	5	3
0	0	0	0	0	0	0	2	2
0	0	0	0	0	0	0	1	3
0	0	0	0	0	0	0	3	5
0	0	0	0	0	0	0	5	1
0	0	0	0	0	0	0	6	2
0	0	0	0	0	0	0	2	4
1	5	4	3	2	6	3	4	3
4	3	2	5	6	1	3	2	4

Figura 2: Tablero con TNT activado

Otro caso importante es cuando la primera fila a eliminar es mayor que 0. En ese caso, se hace uso de un rastreador. Este rastreador se encarga de hacer que no se elimine ningún elemento de la matriz hasta que se llegue al índice correspondiente. En la matriz inferior se encuentra un ejemplo de lo siguiente.

5	3	5	1	5	4	3	2	3
2	2	6	2	5	6	1	2	2
1	3	5	0	0	0	0	0	0
3	5	2	0	0	0	0	0	0
5	1	3	0	0	0	0	0	0
6	2	5	0	0	0	0	0	0
5	3	2	0	0	0	0	0	0
1	5	4	0	0	0	0	0	0
4	3	2	0	0	0	0	0	0

Figura 3: Tablero con TNT activado usando 'rastreador'

- comprobarCaramelo: dependiendo del valor del caramelo, activar una funcionalidad u otra para realizar lo que corresponda. Por ejemplo, si su valor es 10, 20 o mayor que 30, entonces activará su power-up correspondiente. En caso contrario, probará a eliminar sus bloques adyacentes llamando a 'eliminarBloques_aux'.
- eliminarBloques: llamará la función de 'comprobarCaramelo' para que realice los cambios en el tablero. Después, se encargará de comprobar cuantos bloques se han eliminado para poner el power-up correspondiente en el tablero en caso de que sea necesario.
- eliminarBloques_aux: recibirá como parámetros, entre otros, un índice del tablero, el tablero, y el caramelo que se selecciona originalmente. Comprobará si la posición está dentro de los límites del tablero, y si en ella coincide con el caramelo de la selección original. Si es el caso, sustituiremos nuestra posición por un 0, y llamaremos recursivamente a la función en las posiciones de izquierda y derecha, y de arriba y debajo respecto a la que nos encontramos.
- contarEliminados: recorrerá el tablero, sumando 1 a un contador cuando la cabeza de la lista sea 0 y continuará recursivamente con la cola. Cuando quede vacía, devolveremos el contador
- flotarCeros: la función principal que llamaremos desde 'main' para ejecutar la 'gravedad' y dejar caer los bloques. De forma recursiva y por cada columna, ejecutaremos 'findCeroIndices' para conseguir las posiciones de los 0, y 'moverCeros' para moverlos a la parte superior de la columna.
- findCeroIndices: recorrerá la columna correspondiente al número que se pase como parámetro, creando una lista con las ejecuciones recursivas que tendrá los índices en el tablero de las posiciones con un 0, con las más superiores en la columna al principio de la lista.
- moverCeros: cuando ya tenemos la lista de las posiciones de los 0, ejecutaremos recursivamente esta función con cada una. La función llamará a 'bajarColumnas' para flotar cada 0 de la lista

- bajarColumnas: el objetivo de esta función será mover el 0 que se encuentre en el índice que recibe como parámetro a la parte superior de la columna. Intercambiaremos el elemento que haya encima de nuestro 0 con él de forma recursiva, hasta que lleguemos a la primera fila del tablero o nos encontremos justo debajo de otro 0.
- elegirBloqueAutomatico: seleccionar un número aleatorio entre 0 y la longitud del tablero menos 1 para el modo automático, ya que este número corresponde a una casilla del tablero.
- imprimir: imprimir la lista del tablero en forma de matriz, con el número de columnas que queramos, aunque lo recomendable es elegir un número 'N' de forma que la longitud de la lista '`longitud`' sea divisible entre '`N`' para que tenga un formato correcto. O lo que es lo mismo: `longitud = 0 (mod N).`
  Además, antes de imprimir cada número, se comprueba su valor, ya que dependiendo del tipo de caramelo que sea se imprimirá en un color distinto, además que los power-ups son impresos con una letra, y los bloques vacíos no son impresos.
- pedirNumero: pedir un número al usuario que se encuentre entre 0 y un valor máximo, y seguir preguntando hasta validarlo.
- bucleJuego: bucle principal que llamará a todas las funciones para modificar el tablero del juego. Al inicio, rellenará los espacios vacíos de la matriz con nuevos caramelos aleatorios. Después, preguntará al usuario que elija una posición del tablero, o elegirá automáticamente la mejor posición, dependiendo del modo de ejecución seleccionado. Tras esto, intentará eliminar los caramelos de la posición dada. Si no se consigue eliminar nada, se lanzará la llamada recursiva al juego con una vida menos. En el caso de acertar, imprimiremos la matriz actualizada con los bloques eliminados, llamaremos a la función para ejecutar la 'gravedad', y volveremos a imprimir la nueva matriz con los caramelos desplazados al fondo del tablero. Finalmente, ejecutamos una llamada de nuevo al bucle con la matriz modificada para continuar con el juego. La función hace uso de la recursividad de cola para evitar acumular llamadas que colapsen la memoria, ya que esta función se llama constantemente para continuar el juego.

Ejecución del programa

Después de haber hecho un intenso análisis y explicación a todo lo relacionado con la lógica del programa, nos centraremos ahora en el comportamiento de este una vez está puesto en marcha. A pesar de que el lenguaje y el entorno de desarrollo son completamente distintos con respecto a la práctica anterior, el funcionamiento del juego es esencialmente el mismo. Igualmente, volveremos a explicar todo adaptado a este proyecto.

Implementación por consola

Antes de comenzar a jugar, debemos abrir el fichero ejecutable que contiene la práctica. A la hora de ejecutarlo, existen 2 formas de hacerlo:

1.- **Indicando distintos parámetros por consola:** estos parámetros son 4.

- Modo de ejecución. En caso de seleccionar `-m`, entonces el usuario podrá tomar las decisiones sobre qué bloques borrar en Cundy Crosh, de lo contrario, si introduce

-a, el programa se ejecutará de forma automática. El usuario únicamente tendrá que clicar la tecla ENTER cada vez que quiera avanzar hacia la siguiente acción.

- Nivel de dificultad: Para ello, existen dos modos: fácil, introduciendo 1, que tiene bloques del 1 al 4; y difícil, introduciendo 2, que tiene bloques del 1 al 6.
- Tamaño del tablero: En este caso, tenemos 2 parámetros: 'numColumnas', 'numFilas'. Como se puede imaginar, cada uno corresponde al número de columnas y filas que contendrá la matriz, respectivamente.

El orden de cómo deberían ser introducidos dichos parámetros se encuentra en el siguiente ejemplo, en un supuesto caso en el que se ejecuta el programa en modo manual, dificultad fácil, para una matriz 10x10.

```
CandyCrosh -m 1 10 10
```

Figura 4: Ejemplo de ejecución del programa por consola con parámetros

2.- **Ejecutando directamente el programa:** En ese caso, los parámetros mencionados en el punto 1 deberán ser pedidos por consola.

```
    BIENVENIDO A CUNDY CROSH SOGA!
---------------------------------------------------------------
Paradigmas Avanzados de Progamación 3°GII - 21 de Abril de 2023
By: Daniel de Heras Zorita y Adrián Borges Cano

 -> El juego cuenta con 2 modos de ejecución
Elige:
1.-Modo automatico
2.-Modo manual
2
 -> Ahora la dificultad
Elige:
1.-Dificultad facil
2.-Dificultad dificil
2
Introduce el numero de Filas: 12
Introduce el numero de Columnas: 12


-----------------------------------------------
```

Figura 5: Ejemplo de ejecución del programa por consola sin parámetros

No debemos olvidar que, para el **proyecto optimizado** en el que la ejecución automática provoca seleccionar la casilla que mayor número de bloques elimine, llegar al final del juego en el que el usuario se queda sin vidas es una tarea casi imposible, ya que lo que debe ocurrir es que no exista ninguna posible combinación en el tablero. Por tanto, a mayor dimensión de la matriz tablero, mayor dificultad encontraremos en terminar la partida.

Una vez inicializado el programa, se nos presentará la matriz correspondiente al tablero del juego. ¡Ya se ha comenzado a jugar! Para que esto sea llevado a cabo, se genera una matriz llena de ceros con las dimensiones que haya indicado el usuario. Posteriormente, se llama a 'rellenarTablero' para completarlo con bloques de colores. Gracias a este método y esta implementación, llamándole podemos llenar los huecos con ceros siempre que lo necesitemos.

De ahora en adelante, explicaremos cómo funciona la ejecución del programa en caso de que el usuario hubiese seleccionado la opción de juego manual, ya que la automática es igual, solo que más simplificada.

A partir de ahora, el programa le pedirá constantemente al usuario que introduzca las coordenadas del bloque que quiera eliminar.

Figura 6: Primera impresión del juego

Dependiendo del bloque que introduzca el usuario, se pueden dar distintos casos:

1.- **Los bloques eliminados no provocan ningún potenciador**: Los bloques serán eliminados, el número de vidas se mantendrá igual, y se generarán nuevos bloques para ocupar los espacios vacíos.

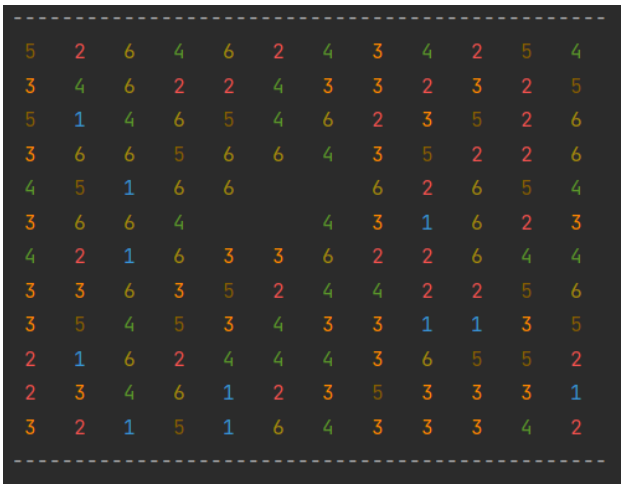

Figura 7: Bloques Vacíos

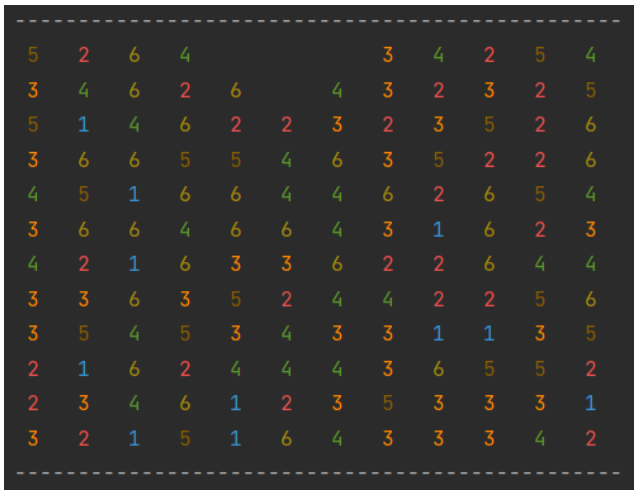

Figura 8: Bloques bajados

2.- **Los bloques eliminados provocan una bomba**: si el jugador la selecciona, se eliminará aleatoriamente su fila o columna correspondiente.

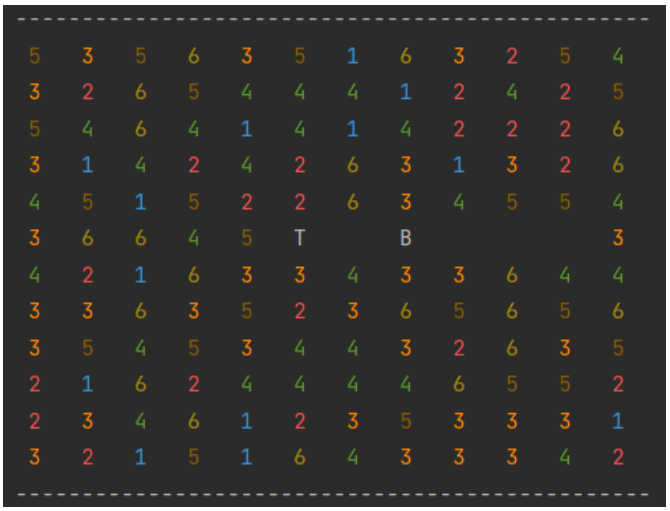

Figura 9: Bloques vacíos con Bomba

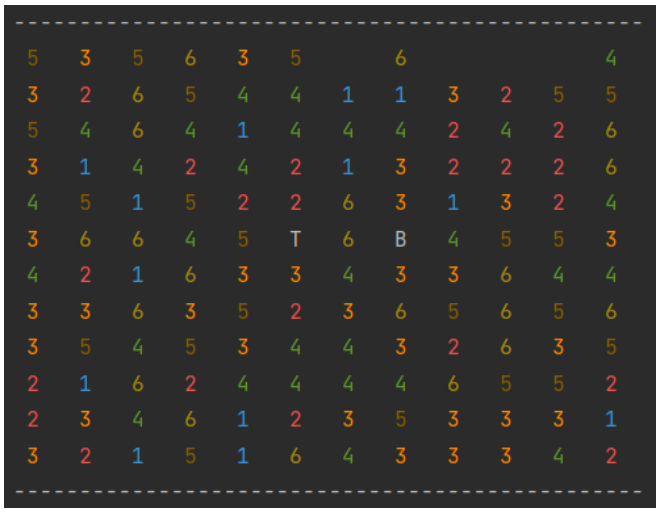

Figura 10: Bloques bajados con Bomba

La ejecución de Figura 9 y Figura 10 es equivalente ante cualquier potenciador. Por ello, a partir de ahora se omitirá.

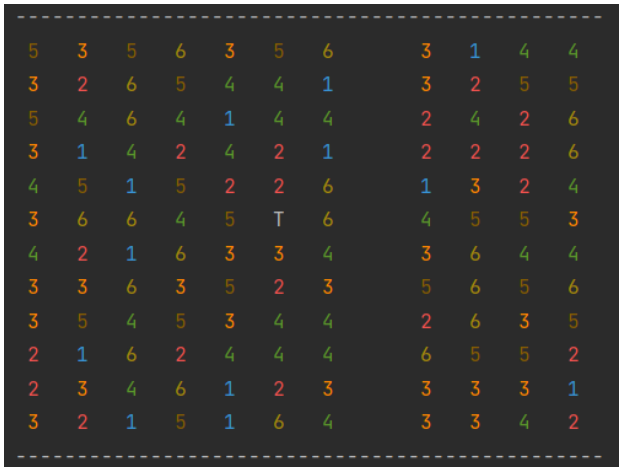

Figura 11: Bloques eliminados con Bomba

En este caso, se muestra que ha sido elegido aleatoriamente la columna en cuestión.

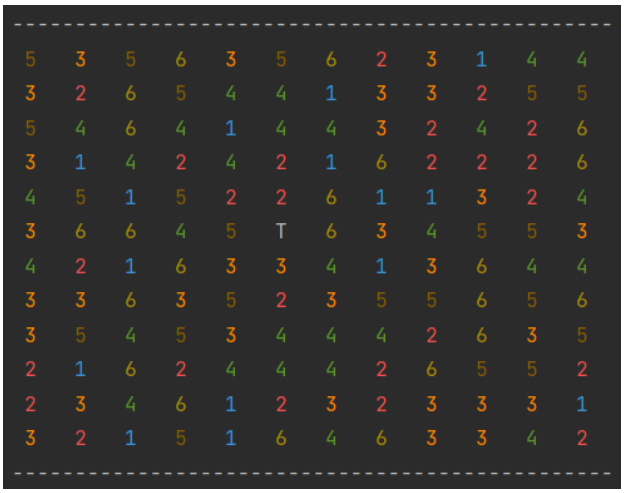

Figura 12: Bloques bajados al activar Bomba

3.- **Los bloques eliminados provocan un bloque de TNT**: si el jugador lo selecciona, se eliminarán todos los bloques en un radio de 4 elementos.

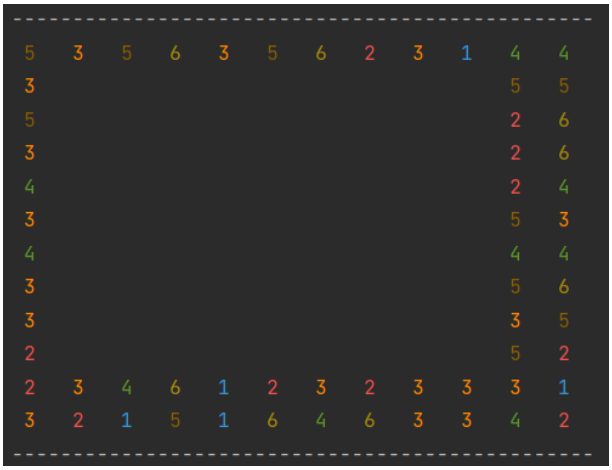

Figura 13: Bloques eliminados con TNT

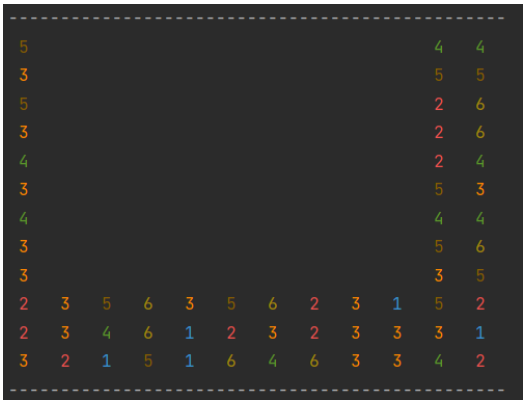

Figura 14: Bloques bajados al activar TNT

4.- **Los bloques eliminados provocan un rompecabezas**: si el jugador la selecciona, eliminará todos los bloques que tengan el mismo color que dicho rompecabezas.

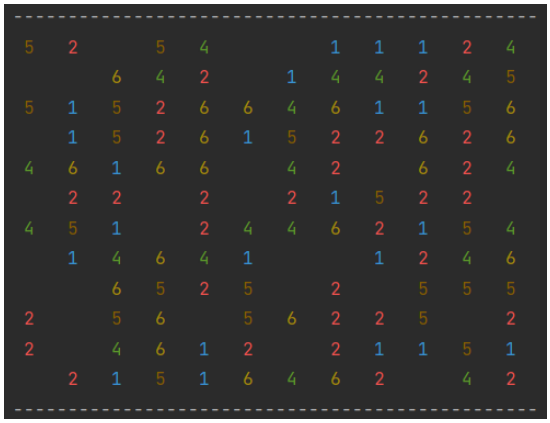

Figura 15: Bloques eliminados con Rompecabezas

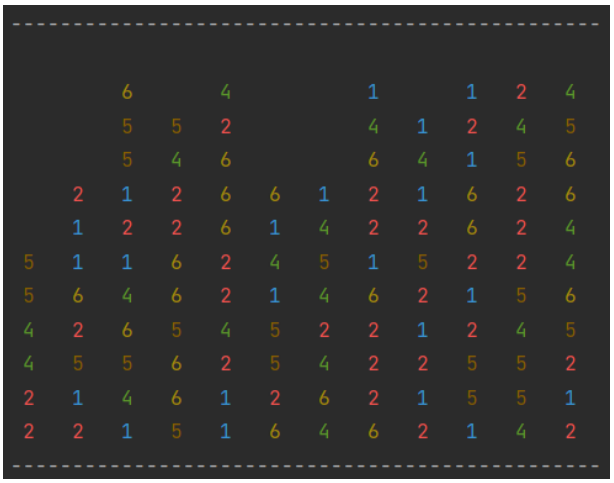

Figura 16: Bloques bajados al activar Rompecabezas

5.- **No se pueden eliminar bloques en la posición indicada**: se mostrará por pantalla que no se ha podido hacer ninguna combinación posible, y se restará una vida.

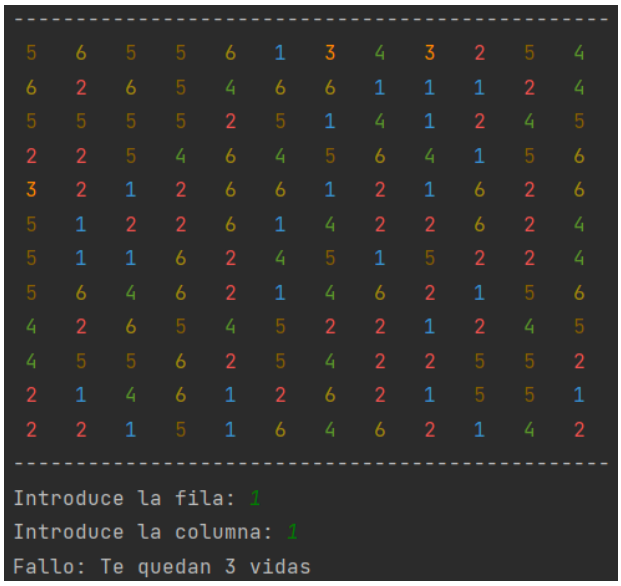

Figura 17: No hay combinación

Combinando todos los puntos anteriormente comentados, el usuario podría permitirse jugar durante horas y, si es habilidoso, mantenerse con suficientes vidas como para seguir jugando.

Finalmente, si el usuario agota las vidas, el juego terminará.

Implementación gráfica (extra)

A diferencia de la implementación por consola, en este caso solo tenemos una forma de ejecutar el programa, y es introduciendo nosotros mismos los parámetros haciendo uso de la GUI (Graphical User Interface) facilitada y desarrollada por nosotros mismo como parte extra del trabajo.

Debemos mencionar que todas las ventanas han sido desarrolladas con la idea de que la modificación de su tamaño no fuese posible, ya que eso rompería completamente con la estética del programa. Al mismo tiempo, se han estudiado los distintos flujos de ejecución que podría tomar el usuario (por ejemplo, que inicie el juego habiéndosele olvidado de insertar una opción del tablero) de forma que el programa quedase sólidamente estructurado.

Todo el desarrollo de esta parte gráfica ha sido desarrollado por completo mediante código ya que, aunque IntelliJ cuente con un editor que permita diseñar de cierta manera el aspecto de las ventanas, las posibilidades de este eran muy limitadas y bastante engorrosas de utilizar. Por lo tanto, todo lo que se muestra en las pantallas ha sido posible mediante la realización constante de pruebas y recalibraciones para poder ajustar correctamente todos los elementos en las ventanas. La ejecución del juego haciendo uso de este apartado gráfico es la siguiente.

Figura 18: Ventana inicial de la interfaz

Nada más iniciarlo, podemos observar como se muestra una pantalla simple y cuidada, donde se presenta el trabajo realizado. Se muestra el nombre de los autores de forma abreviada, para que la portada quede más limpia. Además, para aportar mayor realismo al juego, tanto en la barra de la ventana como en la barra de tareas del equipo, se aprecia un logo personalizado con la seña de identidad del famoso juego de móviles.

Figura 19: Icono personalizado en la barra de la ventana

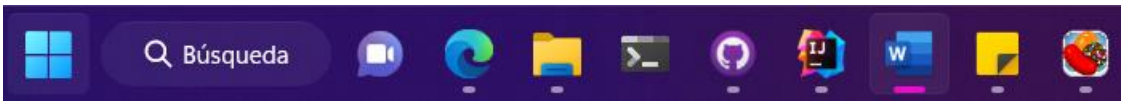

Figura 20: Icono personalizado en la barra de tareas

Si el usuario pulsa en ‘Adelante’, le llevará a la siguiente ventana.

Figura 21: Selector de opciones del juego

En este apartado encontramos todas las opciones que debe rellenar el usuario para poder comenzar a jugar a 'Cundy Crosh'. Una vez el usuario rellene con sus propios gustos cómo quiera ejecutar el juego, se le redirigirá a la próxima ventana. No obstante, antes de continuar, si el usuario no ha seleccionado ningún valor ya sea en el modo de juego, en la dificultad, o en ambas, se mostrará un mensaje de aviso indicándole que no se puede ejecutar el juego sin todos los valores necesarios.

Figura 22: Ventana de aviso cuando faltan datos por rellenar

Además, si el lector ha sido observador, podrá apreciar que tanto el número de columnas como el número de filas está inicializado a 1. Esto se hace para poder asegurar que, en caso de que el usuario no modifique dichos valores, no le aparezca un error como el

mostrado en la Figura 22. También se ha puesto como límite de número de filas y columnas un total de 50 para cada caso, permitiendo como máximo una matriz de 50x50. Aunque la lógica del programa es perfectamente capaz de soportar cualquier dimensionalidad, creemos que este tamaño es más que suficiente como para ejecutar nuestro juego.

Dependiendo de la opción que seleccionemos (ya sea en modo manual o automático), la ventana a mostrar se presentará con pequeñas modificaciones. Para poder mostrar esto, ejemplificaremos con una matriz cuadrada de dimensión 10 (no es necesario que sea cuadrada), en modo difícil, para presentar todos los caramelos posibles. En el caso del **modo manual**, se muestra lo siguiente.

Figura 23: Ventana de juego en modo manual con rompecabezas y bomba

El diseño es minimalista pero completo, donde incluye todo lo necesario para poder jugar. También hemos decidido incluir el logo del propio juego para imitar lo máximo posible este último. Por si esto no fuese suficiente, es fácilmente observable que, a diferencia de las

anteriores ventanas, el estilo de esta es vertical, haciendo una vez más un guiño al juego de móvil, ya que este se acostumbra a jugar en modo vertical.

Para conseguir implementar este diseño, hemos tenido que crear un nuevo panel, contenido en el panel principal ya existente, que es el que incluye el fondo, los textos, imágenes, etc. Este nuevo panel contiene únicamente el tablero, representado por un JTable, donde cada celda corresponde a un número. Para asegurar que el panel y el JTable se mantengan constantes sin importar la dimensión de la matriz, se asigna un tamaño específico, proporcional y uniforme a todas las celdas, de forma que cuando se imprima dicha matriz, se ajuste perfectamente al panel que contiene la tabla. Finalmente, la impresión de los caramelos en forma de imágenes ha sido la parte más sencilla en comparación con la tabla. Lo único que se hace para esto es, dependiendo del color al que corresponda el número de cada una de las celdas, se le asignará una imagen u otra (de igual forma se hace con los potenciadores).

Cabe mencionar que los **bloques marrones** son representados por los caramelos morados, ya que son propios del juego original, y creemos que de esta forma el estilo es más bonito y uniforme, en vez de colocar un caramelo marrón como se proponía.

Después de un par de movimientos, hemos sido capaces de llegar a lo que se muestra en la figura 23. El bloque marrón con puntitos de colores corresponde al bloque del **rompecabezas**, y hemos reutilizado el diseño del juego original para aportarle mayor realismo. También observamos que en la última columna de la tercera fila se encuentra una **bomba**. Como el juego original no contiene este diseño, hemos decidido obtenerlo del banco de emojis que usa iOS en sus distintos dispositivos.

Figura 24: Ventana de juego en modo manual con TNT

Antes de continuar aprovechamos para hacer mención del último potenciador que faltaba, que es el bloque de **TNT**. Después de unos movimientos, nos encontramos en la figura 24, donde vemos el power-up en la posición correspondiente a la 4ª fila y 7ª columna. Al igual que en el caso de la bomba, lo hemos obtenido del emoji correspondiente que viene por defecto en iOS.

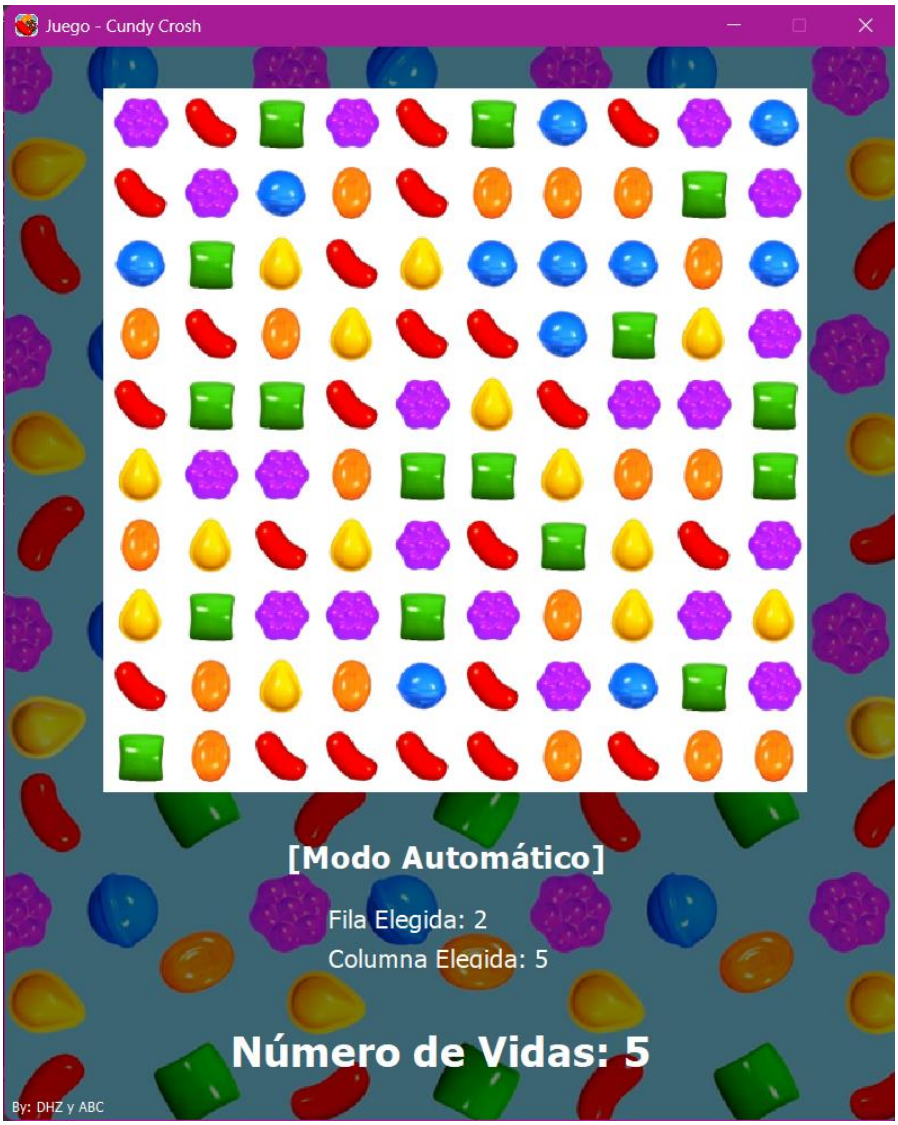

Figura 25: Ventana de juego en modo automático

En este caso, el logo con el nombre del juego es sustituido por información complementaría que incluye el **modo automático**, donde el programa nos indica cual es la posición que ha elegido eliminar en ese momento de la ejecución. Cabe mencionar que dichos números de fila y columna están representados por el lenguaje natural. Es decir, la primera fila corresponde a la fila 1, y lo mismo ocurre con la columna. Hemos querido hacer esta aclaración ya que en el mundo de la programación es común comenzar por 0. Sin embargo, como este juego está orientado a todos los públicos, hemos creído que implementar esto sería confuso para el usuario. Para avanzar hacia el próximo movimiento del juego en este modo, es suficiente con pulsar en cualquier posición del tablero.

Al igual que en la segunda versión del proyecto, hemos incluido el modo automático **optimizado**, el que escoge la mejor opción del tablero. Por lo tanto, a no ser que elijamos un tablero sumamente pequeño o que tengamos muy mala suerte, será prácticamente imposible que el programa pierda alguna vida.

En caso de que, durante la ejecución del juego, el número de vidas llegue a 0 (cosa que es mucho más probable en el modo manual), aparecerá un mensaje informativo, y cuando el usuario clique en este, se cerrará la aplicación.

Figura 26: Mensaje de 'Game Over' y cierre de la aplicación

Conclusiones

Gracias a la primera práctica realizada en CUDA, la forma de llevar este proyecto ha sido mucho más sencilla, puesto que ya teníamos ideado con anterioridad todo lo referente a la lógica del proyecto. A pesar de haber sido dificultoso y tedioso desarrollar la parte práctica únicamente mediante código, y ahora viendo el resultado final, consideramos que ha merecido totalmente la pena pasar por ello para poder terminar presentando semejante programa, el cual estamos muy orgulloso de presentar. Esperamos que el lector haya disfrutado conociendo este trabajo realizado en Scala.