



**PECL2: Scala**

Paradigmas Avanzados de Programación, 3ºGII

Daniel de Heras Zorita

Adrián Borges Cano

ÍNDICE

Contenido

[Introducción 3](#_Toc132555724)

[Versiones del Proyecto 3](#_Toc132555725)

[Implementación básica 3](#_Toc132555726)

[Implementación optimizada 3](#_Toc132555727)

[Implementación gráfica 4](#_Toc132555728)

[Métodos Implementados 4](#_Toc132555729)

[Ejecución del programa 8](#_Toc132555730)

[Implementación por consola 8](#_Toc132555731)

[Implementación gráfica 14](#_Toc132555732)

[Mejoras realizadas 15](#_Toc132555733)

[Conclusiones 15](#_Toc132555734)

# 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 mayo 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. 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, el desarrollo completo de estas funciones se puede encontrar en su [apartado correspondiente](#_Métodos_Implementados), 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](#_Ejecución_del_programa).

## 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

\*TODO: Necesitamos la parte de interfaz gráfica\*

# 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 ej, en obtener la suma de elementos en una lista) va implícito en uno de los parámetros de la llamada recursiva, en vez de encontrarse “externamente” 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 querida ‘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. Consecuentemente, 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 columa 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 la 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, yel 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 la suma de todos los 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 utilizan 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, de forma que si elimina la posición -3 con respecto al origen, acabará eliminando elementos de otras fila 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 misma 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: llamara a 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.
* eliminarBloques\_aux: recibirá como parámetros, entre otros, un índice del tablero, el tablero, y el caramelo que se selecciona originalmente. Comprobara si la posición esta dentro de los limites 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 los lados y de arriba y debajo respecto a en 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, devolvemos 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 numero 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 mas 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 llamara a bajarColumnas para flotar cada 0 de la lista
* bajarColumnas: el objetivo de esta función es mover el 0 que se encuentra en el índice que recibe como parámetro a la parte superior de la columna. Intercambiamos el elemento que haya encima de nuestro 0 con el 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 numero al usuario que se encuentre entre 0 y un valor máximo, y seguir preguntando hasta validarlo.
* bucleJuego: bucle principal que llamara a todas las funciones para modificar el tablero del juego. Al inicio, rellenara 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, intentara 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 del mismo 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.

En la siguiente figura se muestra un ejemplo de cómo se puede ejecutar el programa.

**\*Se ejecuta así el compilado?\***

CandyCrosh.exe -m 2 20 20

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.

Texto

Descripción generada automáticamente

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 pedirá́ constantemente al usuario que introduzca las coordenadas del bloque que quiera eliminar.

Imagen que contiene Calendario

Descripción generada automáticamente

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 mantendrá́ igual, y se generarán nuevos bloques para ocupar los espacios vacíos.

Imagen que contiene Calendario

Descripción generada automáticamente

Figura 7: Bloques Vacíos

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 8: Bloques bajados

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

Imagen que contiene Calendario

Descripción generada automáticamente

Figura 9: Bloques vacíos con Bomba

Imagen que contiene Calendario

Descripción generada automáticamente

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 omitirá́.

Imagen que contiene Calendario

Descripción generada automáticamente

Figura 11: Bloques eliminados con Bomba

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

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

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.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Figura 13: Bloques eliminados con TNT

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

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.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 15: Bloques eliminados con Rompecabezas

Imagen de la pantalla de un videojuego

Descripción generada automáticamente con confianza media

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.

Imagen que contiene Calendario

Descripción generada automáticamente

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

\*TODO: Distintas formas de ejecutar el código, adjuntar capturas de cada una de las formas de hacerlo (mirar memoria de la parte 1)\*

\*TODO: Necesitamos la parte de interfaz gráfica\*

# Mejoras realizadas

\*TODO: EXPLICAR TODO LO REALIZADO EN LA INTERFAZ\*

\*TODO: Necesitamos la parte de interfaz gráfica\*

# Conclusiones

\*TODO: Conclusiones finales para cerrar el trabajo\*