



**PECL2: Scala**

Paradigmas Avanzados de Programación, 3ºGII

Daniel de Heras Zorita

Adrián Borges Cano

ÍNDICE

Contenido

[Introducción 3](#_Toc132364734)

[Versiones del Proyecto 3](#_Toc132364735)

[Implementación básica 3](#_Toc132364736)

[Implementación optimizada 3](#_Toc132364737)

[Implementación gráfica 4](#_Toc132364738)

[Métodos Implementados 4](#_Toc132364739)

[Ejecución del programa 4](#_Toc132364740)

[Mejoras realizadas 4](#_Toc132364741)

[Conclusiones 4](#_Toc132364742)

# 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.

* 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, y otra para la columna.

\*CONTINUAR\*

* eliminarElementosFila: texto
* activarBombaColumna: texto
* reemplazarElemento: texto
* CONTINUAR

\*TODO: Lista y explicación de los métodos\*

\*TODO: En algunas funciones realizamos recursividad de cola para no estar sujetos al problema de que se agote la pila de llamadas en memoria\*

# Ejecución del programa

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

# 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\*