

# Práctica 2

## Paradigmas Avanzados de la Programación

César Martín Guijarro y Alejandro Raboso Vindel

12 de mayo de 2023

# Índice general

<b>1. Detalles de la implementación</b>	<b>2</b>
<b>2. Implementación de algoritmos</b>	<b>3</b>
2.1. Algoritmo de búsqueda y eliminación . . . . .	3
2.2. Algoritmo de gravedad . . . . .	3
2.3. Eliminación Bomba . . . . .	4
2.4. Eliminación TNT . . . . .	4
2.5. Eliminación Rompecabezas . . . . .	5
2.6. Modo aleatorio . . . . .	5
<b>3. Optimización: Modo Automático</b>	<b>6</b>
<b>4. Extra: Implementación Gráfica</b>	<b>7</b>
4.1. Introducción . . . . .	7
4.2. Reescalado dinámico . . . . .	9
4.2.1. <i>reescalar()</i> . . . . .	9
4.2.2. <i>actualizarLabels()</i> . . . . .	9
4.3. Animación de gravedad: . . . . .	10
4.3.1. Animación de carga: . . . . .	10
<b>5. Extra: Puntuación</b>	<b>11</b>
<b>6. Extra: Integración Cloud con Intefaz Gráfica</b>	<b>13</b>
<b>7. Extra: Tablero dinámico</b>	<b>14</b>
<b>8. Mejora descartada: Tienda</b>	<b>15</b>

# Capítulo 1

## Detalles de la implementación

Para esta práctica de Candy Crosh en Scala, se han utilizado objetos inmutables para hacer las respectivas operaciones. En este caso la estructura de datos que hemos tenido que utilizar principalmente han sido las listas.

Además, para representar el tablero, hemos creado una clase Matrix que tiene como atributos el número de filas y columnas así como la dificultad en la que se jugará y finalmente con la lista que representará la matriz en la que se guardaran los caramelos. Esto lo hemos hecho debido a que gracias a tener estos atributos como variables globales es mucho más fácil acceder a ellos en cada una de las funciones de la matriz, y así también nos hemos podido ahorrar bastante código al no tener que estar pasándole los atributos en cada función que se necesite.

Junto con la clase Matrix, hemos creado un object Matrix con funciones útiles que sirven para hacer operaciones de listas como los generadores de filas y columnas, concatenar listas, y funciones de este estilo para poder hacer las operaciones de listas necesarias cuando las necesitemos.

Para comenzar la ejecución del programa, el usuario elegirá las dimensiones y dificultad de la partida y posteriormente se comenzará con el bucle principal del juego.

Este bucle se ha representado como una función recursiva que se detendrá cuando el jugador se quede sin vidas y que mientras el jugador siga con vidas, se mostrará el tablero y se solicitará la posición a eliminar.

Una vez se ha seleccionado la fila y esté en funcionamiento la partida, se llamará a una serie de funciones que comprobará si se debe eliminar o no la casilla y se ejecutará la gravedad para colocar en las posiciones correspondientes los bloques eliminados y generar nuevos bloques aleatorios en la parte superior de las columnas eliminadas.

Cuando todo el tablero esté ya actualizado, se llamará a las funciones del cálculo de la puntuación para sumar a la puntuación actual, los puntos que se hayan obtenido en esa iteración correspondiente.

Finalmente, cuando se hayan perdido todas las vidas, se llamará a la función controlFinal que será la que se encargue de solicitar al usuario el nombre para guardar su puntuación en un archivo que contenga todos los records ordenados por puntuación.

## Capítulo 2

# Implementación de algoritmos

Los algoritmos más importantes para poder ejecutar el programa son el de búsqueda para ver cuántos caramelos adyacentes respecto del seleccionado hay, y el algoritmo de gravedad para poder colocar todos los caramelos en su sitio después de la eliminación.

### 2.1. Algoritmo de búsqueda y eliminación

A la hora de intentar eliminar un bloque no especial, se debe hacer una búsqueda de todos aquellos elementos adyacentes continuados.

Para poder realizar la búsqueda de los elementos adyacentes al seleccionado, se ha tenido que utilizar una función auxiliar que hará uso recursividad y backtracking en la que se visitarán todas las casillas contiguas y, si son del elemento indicado, se irán eliminando hasta que, para todos los caminos, no quede ningún caramelo coincidente, se haya alcanzado un ciclo o se haya llegado el límite de la matriz.

Una vez se haya visitado una dirección completa, se mandará una nueva matriz con los elementos eliminados actualizada para así poder visitar la siguiente dirección.

Se hará uso de la función *reemplazarElemento* que cambiará el elemento en una posición concreta por un 0, que representa una posición vacía. Ya que las listas no se pueden mutar, esta función también trabajará recursivamente concatenando la primera parte de la lista hasta la posición, y la concatenará a la lista resultante de unir un cero y el resto de la lista.

Además de eliminar los caramelos correspondientes, en caso de haber eliminado más de cuatro caramelos, se tendrá que mirar cual es el caramelo especial a generar y colocarlo en la posición correspondiente; llamando de nuevo a *reemplazarElemento*.

### 2.2. Algoritmo de gravedad

Este algoritmo se encarga de desplazar todos los valores nulos de la matriz a la parte superior de la misma, haciendo de çaigan.<sup>el</sup> resto de elementos en el proceso.

Al igual que el de búsqueda y eliminación, se ha implementado con una función auxiliar. En este caso, dicha función se ejecutará por cada una de las columnas del tablero.

Para conseguir aplicar este algoritmo, se ha utilizado un contador que indicará cuantos elementos no eliminados (representados como un distinto de 0) hay en cada columna. Recursivamente se irá recorriendo la columna de abajo a arriba y si en esa posición se encuentra un valor diferente a cero, se copiará el valor de dicha celda al del índice del contador (inicializado en 0) en la columna. Además, se incrementará el contador. En el caso de que el elemento sea igual a cero, se continuará sin hacer ninguna copia ni incrementar el contador. Al finalizar la recursión, los elementos se habrán desplazado correctamente y en la parte superior quedarán una serie de elementos "basura" que se deben aleatorizar. En concreto habrá *filas-contador* elementos a reemplazar. Una vez sustituidos estos elementos, la columna tendrá que reemplazar esa columna original de la matriz. Oara ello, se hará uso de la función *reemplazarColumna*.

## 2.3. Eliminación Bomba

Para eliminar una bomba primero se debe ver si se elimina una fila o una columna.

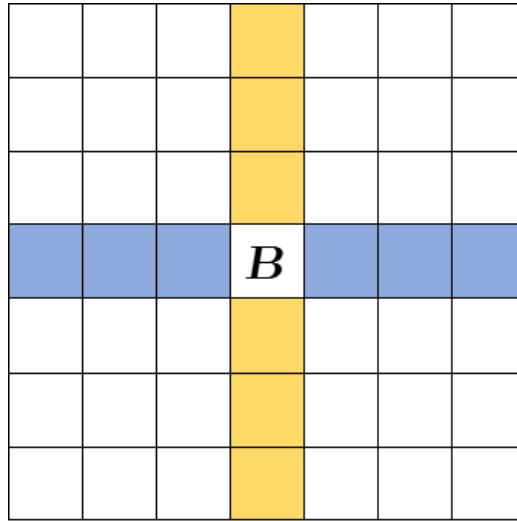


Figura 2.1: Eliminar Bomba

En caso de eliminar una fila, se generará una lista de ceros con el tamaño de la fila (el número de columnas que haya) y se cogerán todas las filas hasta la fila a eliminar y se concatenará con el con la lista de ceros y a eso, se le concatenarán el resto de las filas de la matriz. Esto se ha podido hacer gracias a las funciones de toma y deja que cogen o quitan los primeros  $n$  elementos de una lista.

En caso de eliminar una columna voy reemplazando fila a fila el elemento de la columna por un 0.

## 2.4. Eliminación TNT

Para eliminar un TNT tendremos que mirar cuales son los elementos afectados por él.

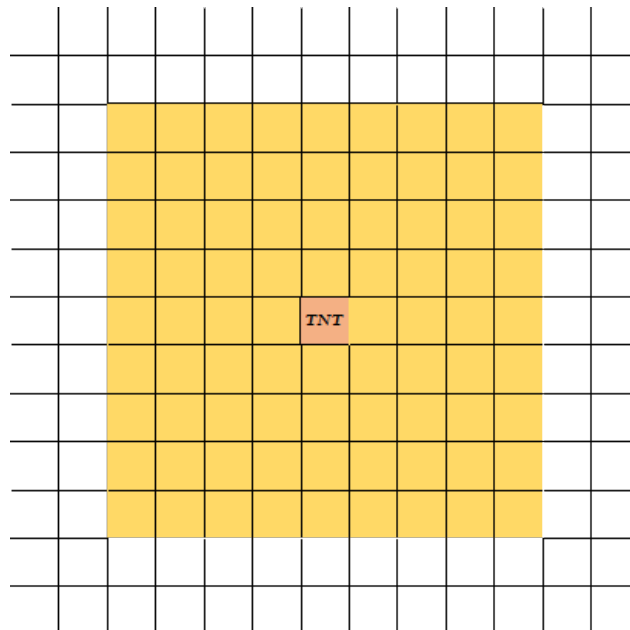


Figura 2.2: Eliminar TNT

Se tendrá que recorrer un cuadrado al rededor del TNT con una distancia máxima de 4 filas y 4

columnas. Mirará todos y cada uno de los elementos de ese cuadrado y se usará pitágoras para saber si su distancia respecto al TNT es 4, en caso afirmativo, se eliminará el caramelo y seguirá con la siguiente casilla hasta haber recorrido todo el cuadrado. Si la distancia al elemento era mayor de cuatro, se pasará al siguiente sin haberlo eliminado.

## 2.5. Eliminación Rompecabezas

Los caramelos Rompecabezas se generarán después de haber eliminado 7 o más caramelos sin haber seleccionado un caramelo especial. Los rompecabezas pueden ser desde R1 hasta R6 y después de seleccionar uno de ellos se llamará a la función `eliminarRompecabezas` que comprobará todo el tablero y reemplazará los caramelos por un 0 en caso de coincidir con el tipo.

## 2.6. Modo aleatorio

Para poder realizar el modo aleatorio, simplemente hemos generado un número aleatorio según el número de filas y otro según el número de columnas que hay en total para después llamar a las mismas funciones que se llaman desde el modo manual pero con esa fila y columna aleatoria.

## Capítulo 3

# Optimización: Modo Automático

En la etapa de optimización hemos realizado el modo aleatorio en la clase principal (Main) hemos creado una función `modoAutomatico` que llamará a otras funciones auxiliares para seleccionar cual es la mejor opción a eliminar.

Para detectar la mejor opción, se recorrerá todas las posiciones del tablero y se hará la búsqueda de los caminos, en el caso de que el número de elementos encontrados por la búsqueda sea mayor, se guardará junto con la fila y la columna de esa posición. Una vez se haya comprobado todo el tablero, la mejor posición será la última guardada, y se procederá a llamar a los mismos métodos de eliminar del modo normal pero con esa posición elegida.

Habrà que tener en cuenta que este modo aleatorio no perderà la partida salvo en el punto en el que no haya ninguna casilla que se pueda eliminar, en cuyo caso, perderà las cinco vidas consecutivas al fallar todas las veces por no tener ninguna opción posible.

Además, en caso de que la partida se haga en dificultad fácil con un tablero relativamente grande, la partida se haría prácticamente eterna ya que con tan pocos bloques posibles (4 a parte de los especiales) es bastante improbable que se quede sin ninguna casilla con la que pueda eliminar algo.

## Capítulo 4

# Extra: Implementación Gráfica

### 4.1. Introducción

Para poder realizar la implementación gráfica de esta práctica, se ha recurrido a *Java Swing*. Se han utilizado una serie de Paneles (*JPanels*) para poder incluir las distintas ventanas de la interfaz. Además se ha incluido una pequeña librería creada por uno de los miembros del grupo para facilitar el manejo de *JavaSwing*. También se ha llegado a incluir música durante la ejecución de la partida para que se disfrute aún más la relajación de jugar un Cundy Crosh.

Al comenzar la ejecución de la parte gráfica, se mostrará una portada



Figura 4.1: Panel Inicial

Y posteriormente un menú inicial





Figura 4.2: Menú

En el apartado de juego se desplegará un menú lateral donde se podrán elegir el modo de juego y las dimensiones del tablero y donde se podrá comenzar a jugar.



Figura 4.3: Selección juego

Una vez se inicie la partida se iniciará una pantalla con un fondo de creación propia. Además cargarán unos gráficos para indicar las vidas y la puntuación del usuario. Finalmente, iniciará la animación de carga del tablero. El tablero además se podrá ajustar dinámicamente, de esta manera, independientemente de las dimensiones especificadas por el usuario, éste siempre estará centrado y con un tamaño adecuado a la ventana.

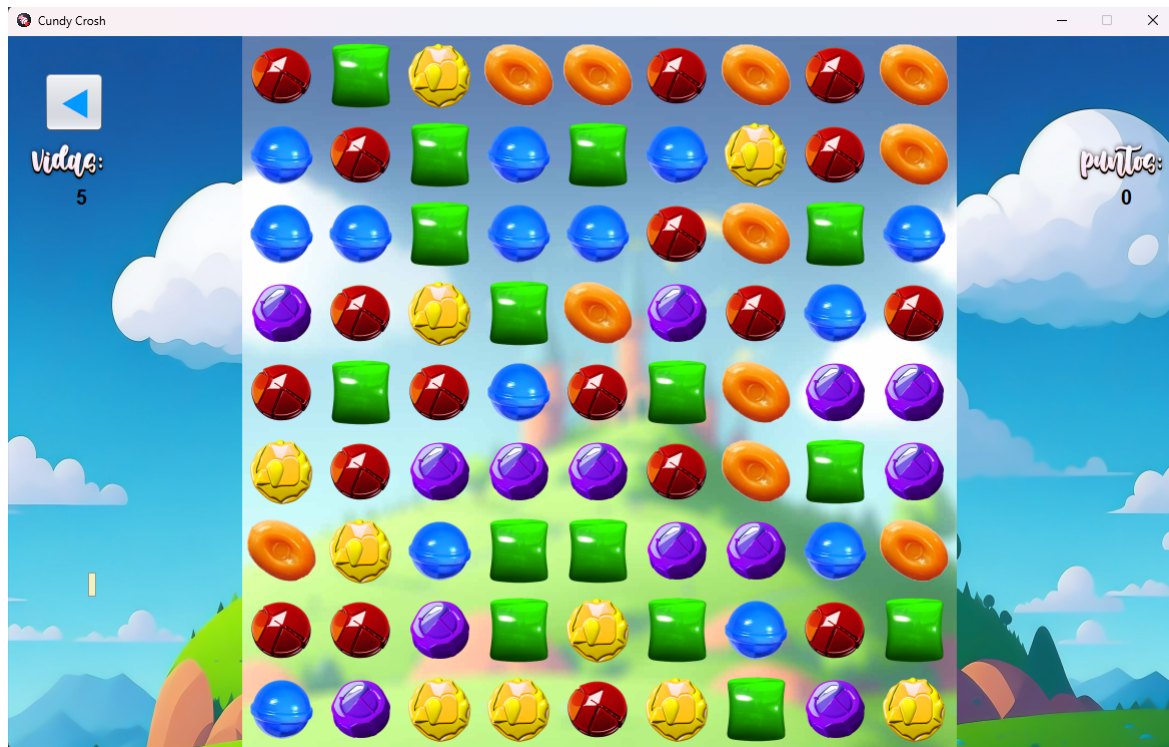


Figura 4.4: Partida

## 4.2. Reescalado dinámico

### 4.2.1. *reescalar()*

El método `reescalar()` se encarga de reajustar el tamaño de los botones que se utilizan en el tablero de juego dentro del `JPanel`.

Primero, se obtiene el ancho y alto actual del `JPanel` utilizando los métodos `getWidth()` y `getHeight()`.

Luego, se define la variable `tamBoton` como el tamaño mínimo entre el ancho dividido por el número de columnas de botones y el alto dividido por el número de filas de botones. Esto asegura que los botones sean lo suficientemente grandes como para caber en el `JPanel` sin salirse de su borde.

Después, se comprueba si la animación ha terminado, ya que esto afecta cómo se posicionan los botones. Si la animación ha terminado, se utiliza un bucle `for` para recorrer todos los botones en el tablero de juego. Dentro del bucle, se establece la posición y tamaño de cada botón utilizando el método `setBounds()`. La posición de cada botón se establece en función de su fila y columna en el tablero de juego y el tamaño de cada botón se establece en función de la variable `tamBoton`.

Si la animación no ha terminado, los botones se colocan fuera de la pantalla. Esto se hace para que los botones no se muestren en su posición normal hasta que la animación haya terminado.

Finalmente, se actualizan las etiquetas y el fondo del `JPanel` utilizando los métodos `actualizarLabels()` y `actualizarFondo()`, respectivamente.

Es decir, este método coloca todos los botones en su posición adecuada con su tamaño adecuado. Por ende, se puede ver que es utilizado después de funciones como la animación de la gravedad.

### 4.2.2. *actualizarLabels()*

El método `actualizarLabels()` se encarga de actualizar los iconos de los botones en el tablero de juego utilizando las imágenes reescaladas de los caramelos correspondientes.

Primero, se obtiene el ancho y alto actual del `JPanel` utilizando los métodos `getWidth()` y `getHeight()`.

Luego, se define la variable `tamBoton` como el tamaño mínimo entre el ancho dividido por el número de columnas de botones y el alto dividido por el número de filas de botones. Esto asegura que los botones sean lo suficientemente grandes como para caber en el `JPanel` sin salirse de su borde.

A continuación, se utiliza un bucle `for` para recorrer todas las imágenes de caramelos y se utiliza el método `reescalarImagen()` de la clase `MetodosGUI` para reescalar cada imagen a la dimensión adecuada.

(tamaño de botón).

Luego, se utiliza otro bucle for anidado para recorrer todos los botones en el tablero de juego. Dentro del bucle, se calcula el índice correspondiente en la lista de imágenes de caramelos utilizando la fórmula  $\text{indice} = i * \text{botonesColumnas} + j$ . Luego, se crea un nuevo hilo de ejecución utilizando la clase Thread y el método start(). Dentro del hilo, se establece el icono de cada botón utilizando el método setIcon() y la imagen de caramelo correspondiente de la lista de imágenes reescaladas.

Es importante destacar que se utilizan hilos de ejecución para actualizar los iconos de los botones en el tablero de juego, lo que significa que la actualización se realiza en segundo plano y no interrumpe el hilo principal de ejecución. Esto ayuda a evitar que la interfaz de usuario se congele o se vuelva no interactiva mientras se actualizan los iconos de los botones.

### 4.3. Animación de gravedad:

El método *gravedad()* se encarga de aplicar la gravedad a las celdas del tablero que contienen ceros, lo que indica que no hay caramelos en esas celdas. La idea es mover hacia abajo los caramelos que están por encima de cada celda vacía, de manera que las celdas vacías queden en la parte superior del tablero y los caramelos se acomoden debajo de ellas.

El método comienza recorriendo la lista de caramelos, de manera similar al método anterior. Por cada celda que contenga un caramelo, se busca hacia abajo la primera celda vacía y se calcula cuántas celdas hay que mover hacia abajo el caramelo. Se guarda esta información en una matriz de desplazamientos.

Luego se crea un hilo para cada caramelo que deba ser movido hacia abajo. Dentro de cada hilo, se llama al método "moverBoton", que anima la caída del caramelo hacia abajo. Mientras los hilos están en ejecución, se decrementa un contador, que indica cuántos hilos todavía están activos.

Finalmente, se crea un último hilo que se ejecutará cuando todos los hilos de movimiento hayan terminado. Este hilo se encarga de actualizar la lista de caramelos con los ceros en la posición correcta, actualizar las etiquetas y reescalar el panel para ajustarlo al tamaño actual. También se habilitan los botones del tablero para que puedan ser usados de nuevo.

En resumen, este método se encarga de aplicar la gravedad a las celdas vacías del tablero, moviendo los caramelos hacia abajo para acomodarlos debajo de las celdas vacías.

#### 4.3.1. Animación de carga:

Este método es responsable de la animación que se muestra cuando se carga el tablero por primera vez. La idea es que cada celda caiga desde arriba, uno por uno, fila por fila. Para lograr esto, el método utiliza dos hilos, uno para la animación en sí y otro para esperar a que la animación termine.

Primero, se crea un objeto AtomicInteger llamado *contador* con un valor inicial igual al número total de celdas en el tablero. El contador se utiliza para realizar un seguimiento de cuántas celdas se han animado hasta el momento. Luego, se crea un hilo que se encarga de la animación. Este hilo se usa para agregar un retraso entre cada animación, lo que da la impresión de que las celdas están cayendo una por una.

El hilo de animación recorre cada fila del tablero, comenzando desde la última fila y trabajando hacia arriba. Dentro de cada fila, el hilo recorre cada celda de izquierda a derecha o de derecha a izquierda, alternando el orden de las filas para dar la impresión de que las celdas caen en una secuencia aleatoria. Para cada celda, se crea un nuevo objeto de hilo *HiloAnimacion* y se lo inicia con la posición final de la celda y un factor de escala de 1.1 para dar la impresión de que la celda se agranda ligeramente al caer.

Después de iniciar cada hilo de animación, el contador se decrementa para indicar que una celda ha sido animada. Se agrega un retraso de tiempo a cada animación para que parezca que las celdas caen una por una.

Una vez que se inicia el hilo de animación, se crea otro hilo que espera a que todos los hilos de animación hayan terminado. Este hilo utiliza un bucle while para comprobar continuamente el valor del contador. Si el contador es mayor que cero, el hilo se duerme durante 150 milisegundos y reproduce un sonido de deslizamiento. Cuando el contador finalmente llega a cero, se activan los botones y se establece el valor de *animacionTerminada* en true para indicar que la animación ha terminado por completo.

En resumen, el método *animacionCarga* utiliza dos hilos para crear una animación de carga de tablero que hace que las celdas caigan una por una desde la parte superior del tablero. El primer hilo se encarga de la animación en sí, mientras que el segundo hilo espera a que la animación termine antes de activar los botones.

## Capítulo 5

# Extra: Puntuación

Como apartado extra quisimos añadir puntuaciones en la partida para así darle un toque de rivalidad entre jugadores. En la práctica 1, esta mejora quedó descartada por falta de tiempo. Por ende, quisimos implementarlo en esta práctica.

Para nuestra sorpresa, mientras estábamos implementando este sistema de puntuación, se publicó el enunciado de la PL3, en el que también se menciona un sistema de puntos. Por ello, hemos decidido adaptar nuestro sistema de puntos al propuesto en esa práctica.

Para implementar las puntuaciones hemos creado una función que después de cada movimiento, suma los puntos dependiendo de cuantos bloques se hayan eliminado. Se suma 1 punto por cada bloque eliminado a parte de otro punto por cada 10 bloques y luego por bloques especiales se suman 5 puntos si es bomba, 10 si es TNT y 15 si es rompecabezas, además de que se duplicará la puntuación en caso de estar en modo difícil.

```
Has perdido 🍌  
Tú puntuacion: 0  
Has durado: 8 segundos jugando  
Fin de la partida: 2023-05-11T19:13:16  
Introduce tu nombre para que figure en los records  
El pato
```

Figura 5.1: Derrota

Después de haber calculado las puntuaciones, se solicitará un nombre para guardarlo todo a una BBDD y poder cargarlo para poder tener las puntuaciones actualizadas antes de imprimirlas. Y una vez esté todo guardado, se mostrará una tabla con las puntuaciones actualizadas para ver todos los records.

Nombre	Puntuación	Fecha	Duración
NPC	32568798	2023-05-1112:27:37	47
AutoGod	98190	2023-05-1112:32:18	23
PedroSanchez	1879	2023-05-1113:50:22	8
McLovin	420	2023-05-1019:47:05	69
Raboso	355	2023-04-1019:47:05	434

Cesar	2	2023-05-1114:01:17	19
AnaBlanco	1	2023-05-1113:50:42	9
MarIPaz	0	2023-05-1114:02:19	7
Elpato	0	2023-05-1119:13:16	8

Figura 5.2: Tabla Puntuaciones

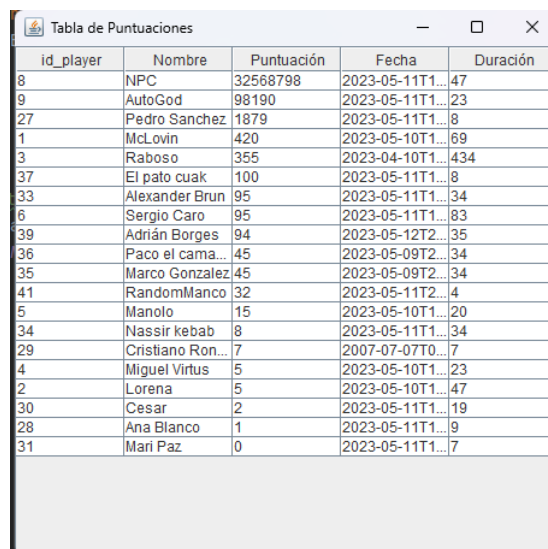
Como se puede ver, se han ordenado por puntuación ya que le da un toque retro típico de las máquinas arcade.

## Capítulo 6

# Extra: Integración Cloud con Intefaz Gráfica

Como es lógico, la parte de Cloud se pide en la tercera práctica, por lo que no hablaremos ahora profundamente sobre ello. En dicha práctica se pide integrar la nube junto con el programa de scala y el visor web, pero en ningún momento se pide integrar esto también con la interfaz gráfica. Como la interfaz gráfica forma parte de la PL2, hemos implementado como mejora una tabla que muestra todas las puntuaciones de la nube pero estando esta tabla en la interfaz.

Para acceder a la tabla, bastará con pulsar el botón de *Ver Puntuaciones* o con obtener un *Game Over*.



id_player	Nombre	Puntuación	Fecha	Duración
8	NPC	32568798	2023-05-11T1...	47
9	AutoGod	98190	2023-05-11T1...	23
27	Pedro Sanchez	1879	2023-05-11T1...	8
1	McLovin	420	2023-05-10T1...	69
3	Raboso	355	2023-04-10T1...	434
37	El pato cuak	100	2023-05-11T1...	8
33	Alexander Brun	95	2023-05-11T1...	34
6	Sergio Caro	95	2023-05-11T1...	83
39	Adrián Borges	94	2023-05-12T2...	35
36	Paco el cama...	45	2023-05-09T2...	34
35	Marco Gonzalez	45	2023-05-09T2...	34
41	RandomManco	32	2023-05-11T2...	4
5	Manolo	15	2023-05-10T1...	20
34	Nassir kebab	8	2023-05-11T1...	34
29	Cristiano Ron...	7	2007-07-07T0...	7
4	Miguel Virtus	5	2023-05-10T1...	23
2	Lorena	5	2023-05-10T1...	47
30	Cesar	2	2023-05-11T1...	19
28	Ana Blanco	1	2023-05-11T1...	9
31	Mari Paz	0	2023-05-11T1...	7

Figura 6.1: Puntuaciones

Además de haber implementado la tabla, si se selecciona una de sus casillas, nos llevará directamente al menú de puntuaciones detallado de la web para que así se pueda ver el elemento seleccionado de manera más cómoda, junto al resto de datos no mostrados en la tabla.

## Capítulo 7

# Extra: Tablero dinámico

Gracias a la creación de la clase/objeto Matrix en Scala, hemos sido capaces de implementar tableros de dimensiones dinámicas. En cualquier momento de la ejecución, se podría sustituir la matriz por una matriz de cualquier otra dimensión. Esto también se ha implementado de la misma manera en la interfaz gráfica, que junto a las funciones de reescalado, se puede adaptar sin problema.

Finalmente, en ningún momento de la ejecución normal se cambian las dimensiones del tablero, ya que por falta de tiempo, no hemos podido llevarlo a cabo. Pero, como idea se propone que según la cantidad de puntos que lleve un jugador, se agregue una nueva fila o columna. Con poco más de un simple if, se podría implementar gracias al sistema que hemos creado.

## Capítulo 8

# Mejora descartada: Tienda

Junto con el sistema de puntos que hemos implementado, nos hubiese gustado crear una tienda donde el jugador puede gastar puntos para obtener beneficios como vidas adicionales, reemplazar un elemento del tablero por una TNT o rompecabezas...

También se planteó haberlo implementado en la interfaz gráfica.



