Memoria descriptiva del proceso de desarrollo de la PECL2. Implementación del juego 2048 en Scala utilizando recursividad.

16384 en Scala

Ampliación de Programación Avanzada

**ÍNDICE:**

1. **Trabajo Obligatorio. Descripción del juego**
   1. **Función principal**
   2. **Juego**
   3. **Movimientos y sumas**
   4. **Otros métodos útiles**
2. **Optimización. Juego Automático**
3. **Trabajo Avanzado. Interfaz Gráfica**

**1.- Trabajo Obligatorio. Descripción del juego**

En esta PECL2 de Ampliación de Programación Avanzada se nos pedía que desarrollásemos el juego 2048 pero esta vez en el lenguaje de programación Scala utilizando métodos recursivos.

Para el desarrollo de esta práctica era necesario generar un tablero de determinado tamaño, dependiendo de la dificultad escogida por el usuario al iniciar.

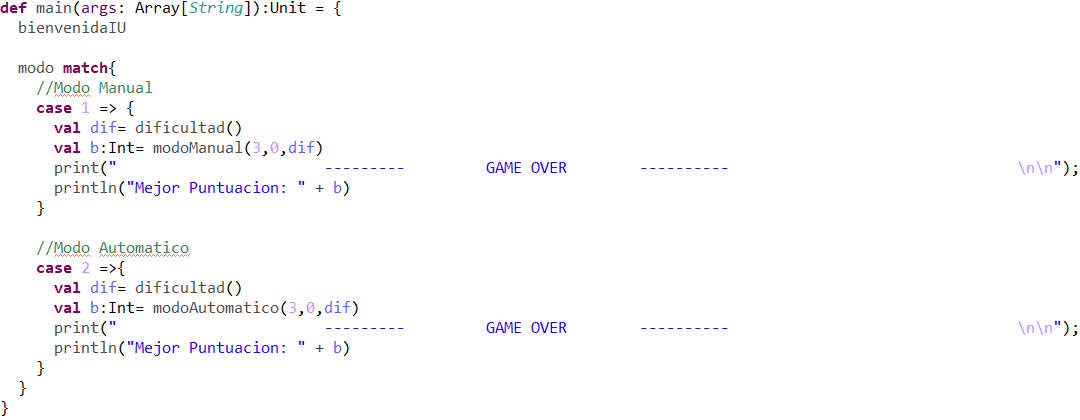
En el tablero aparecerían una serie de números, que a través de movimientos en 4 direcciones (Arriba, Abajo, Izquierda y Derecha) podremos fusionar y sumar, aumentando nuestra puntuación.

Por último, la partida terminará cuando el tablero esté lleno y no se puedan realizar más movimientos.

Para esta práctica en específico, el tablero era necesario que fuese una lista de enteros.

**1.1.- Función principal**

En nuestro programa, lo primero con lo que nos encontramos, como en cualquier otro, es el método main:

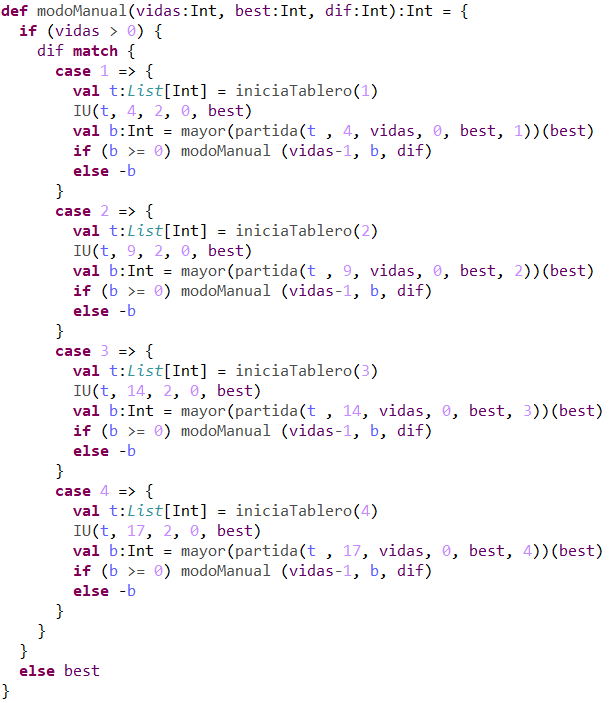


En esta función, aparte de mostrar una breve interfaz de bienvenida a 16384, comprobamos el modo en el que va a jugar el usuario, modo manual o modo automático, el cual explicaremos más adelante.

Cuando elegimos el modo manual pedimos al usuario por teclado la dificultad, especificando valores entre 1 y 4, y comenzaremos la ejecución del modo Manual, iniciando con 3 vidas, una puntuación inicial de 0 y la dificultad seleccionada.

**1.2.- Juego**

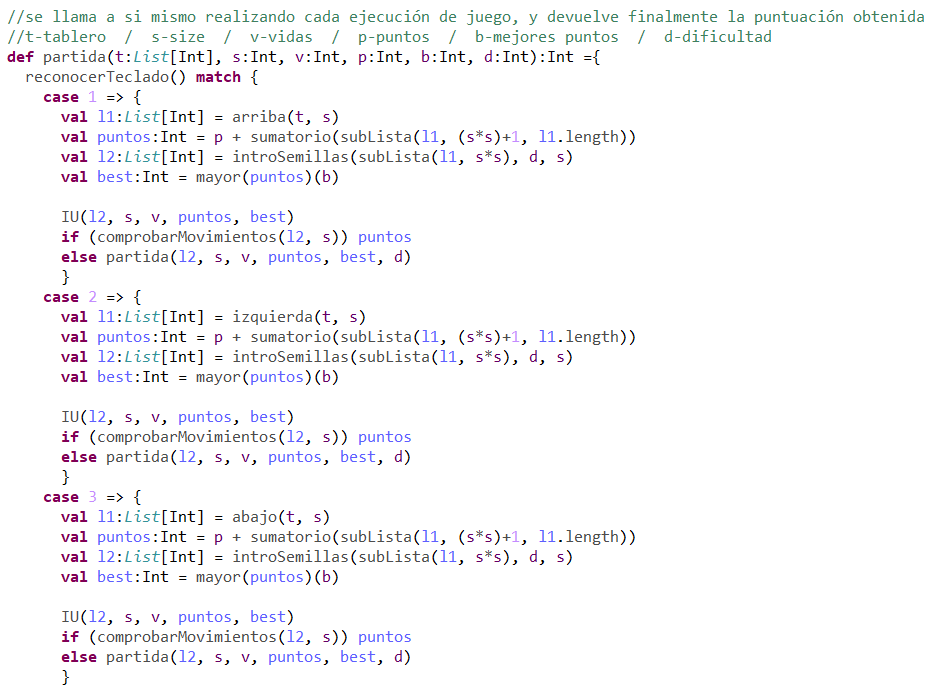
A continuación, vamos a explicar el funcionamiento del bucle principal del juego:

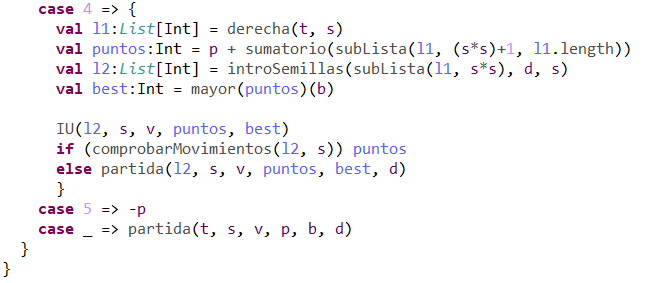


Cada vez que ejecutamos el método modoManual comprobamos la dificultad, y en función de la dificultad escogida **iniciamos un tablero** de un determinado tamaño especificado en el enunciado de la práctica:

|  |  |
| --- | --- |
| **DIFICULTAD** | **TAMAÑO DEL TABLERO** |
| 1 | 4X4 |
| 2 | 9X9 |
| 3 | 14X14 |
| 4 | 17X17 |

Después de iniciar al tablero, mostramos una interfaz de usuario y realizamos la llamada al método partida.





En el método partida llamamos al método **reconocerTeclado()**, el cual devuelve un entero entre 1 y 5 dependiendo de la tecla que se pulse **(WASD para mover, 0 para salir)** o muestra un mensaje de **“Dirección Imposible”** si no es ninguna tecla válida.

Después, dependiendo de la tecla seleccionada se realiza un movimiento en una de las 4 direcciones disponibles, se suman las puntuaciones, se muestra la matriz nueva al usuario, y se realiza una llamada recursiva a si mismo, al método partida, con la nueva puntuación y el nuevo tablero.

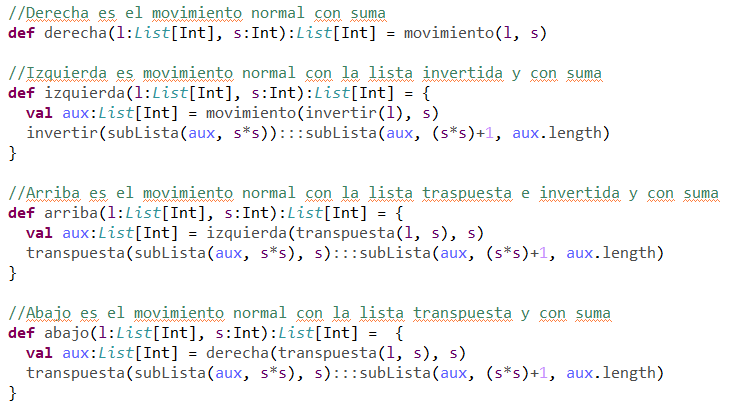
La llamada recursiva finaliza cuando **no quedan movimientos disponibles** en ninguna de las 4 direcciones.

**1.3.- Movimientos y sumas**

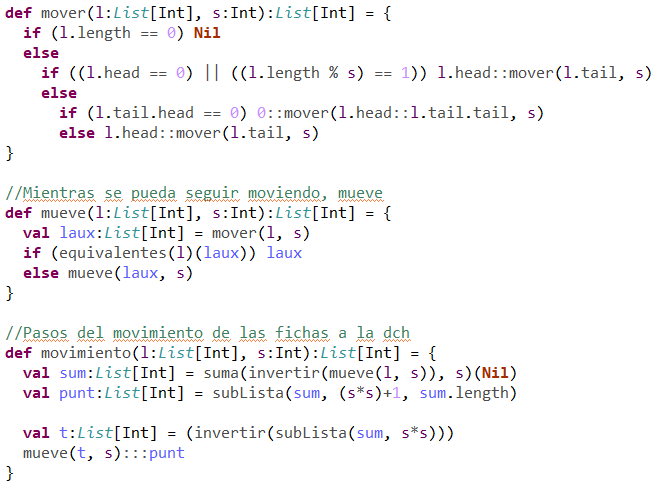
A continuación, vamos a pasar a explicar lo métodos de los movimientos y las sumas entre casillas del tablero, parte esencial de la práctica.

Para realizar los movimientos en las 4 direcciones se nos ocurrió que podríamos simplemente programa un movimiento hacia la derecha, y que el resto de las direcciones se basasen en ese movimiento, pero con la lista ordenada de distinta manera.

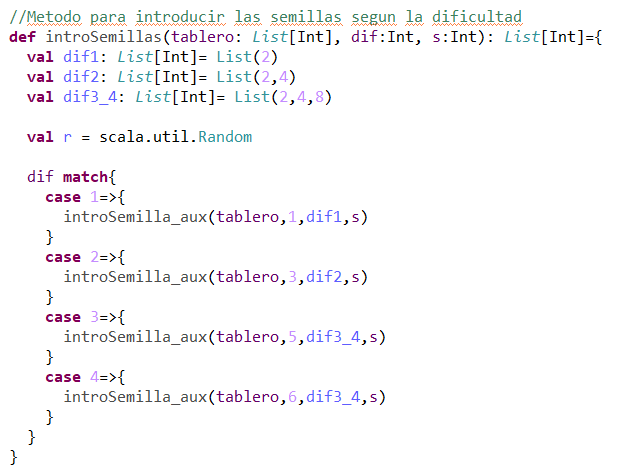
Por ejemplo, el **movimiento hacia la izquierda** es igual que el movimiento hacia la derecha, pero con la lista invertida. Así, el **movimiento hacia abajo** es igual que hacia la derecha, pero con la lista como si fuese una matriz traspuesta. Por último, el **movimiento hacia arriba** es el movimiento hacia la derecha, pero con la lista invertida y traspuesta.

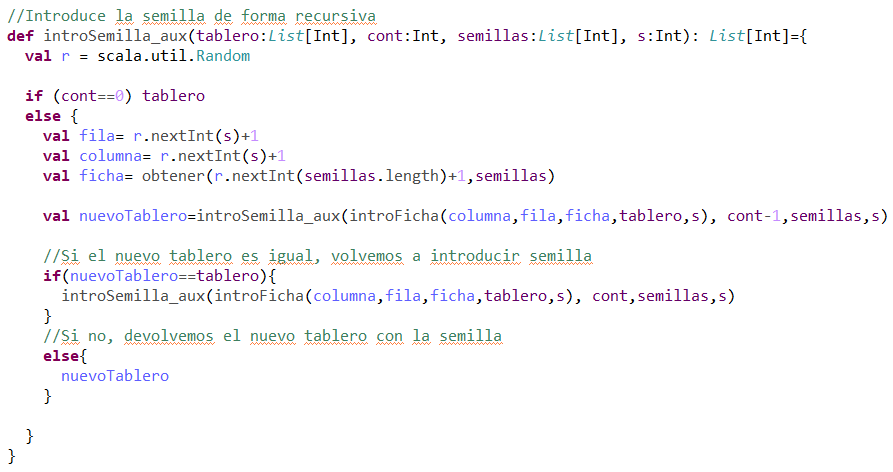


Por lo tanto, el único método de movimiento programado es el propio movimiento(l:*List*[Int], s:Int) el cual realiza la suma de las casillas y almacena las puntuaciones al final de la propia lista de tablero, y se apoya en los métodos mueve y mover para desplazar las casillas hacia la derecha.

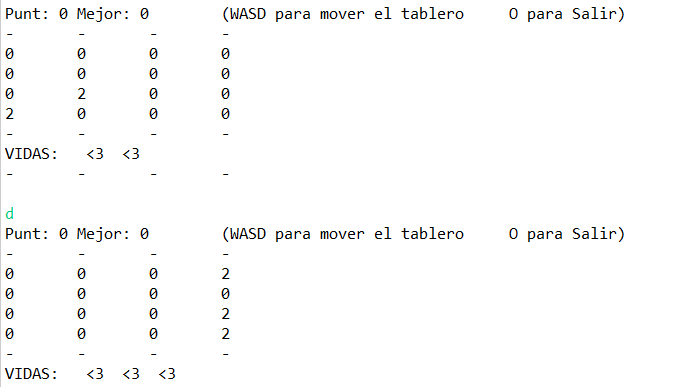


Después de cada movimiento llega la hora de introducir nuevas semillas al tablero utilizando el método introSemillas el cual dependiendo de la dificultad realiza una llamada a introSemillas\_aux que introducirá un número concreto de semillas nuevas en el tablero hasta que el contador sea 0 realizando llamadas recursivas a sí mismo.

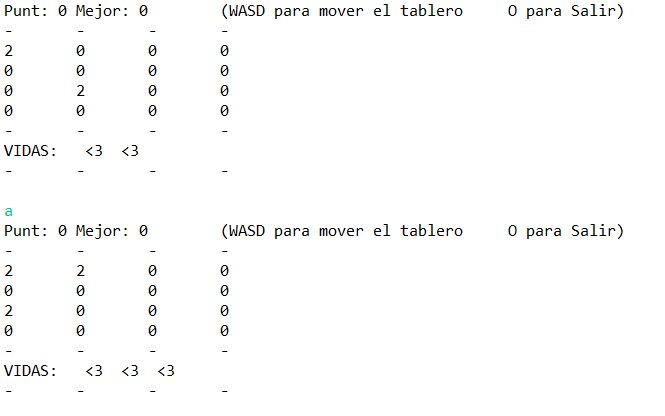




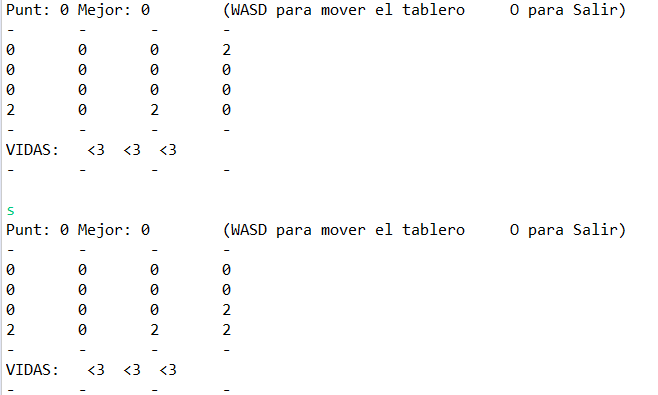
**EJEMPLO DE MOVIMIENTO HACIA LA DERECHA:**

****

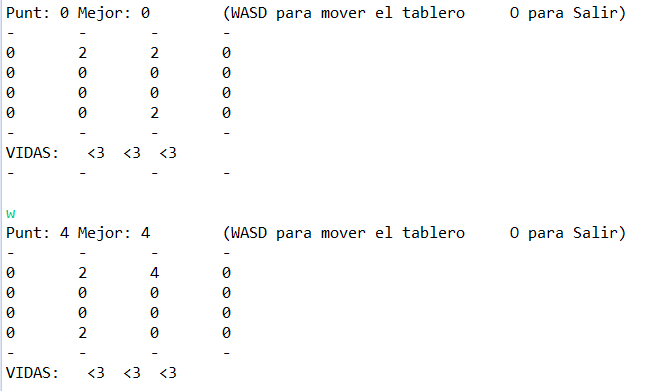
**EJEMPLO DE MOVIMIENTO HACIA LA IZQUIERDA:**

****

**EJEMPLO DE MOVIMIENTO HACIA ABAJO:**

****

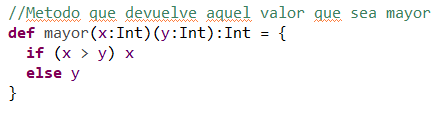
**EJEMPLO DE MOVIMIENTO HACIA ARRIBA:**

****

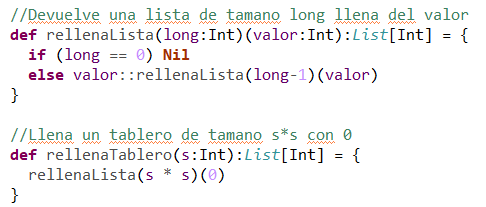
**1.4.- Otros métodos útiles**

Aparte de los métodos principales para la ejecución del programa que han quedado descritos en los apartados anteriores hemos utilizado una serie de métodos que nos han servido de apoyo para facilitar la programación.

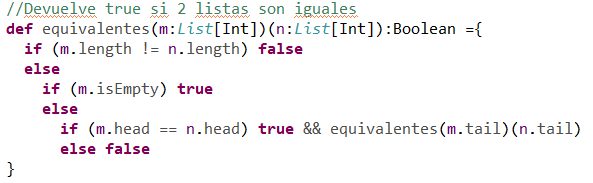
**Mayor:** Mayor es una función auxiliar que se encarga de devolver el número mayor entre dos.



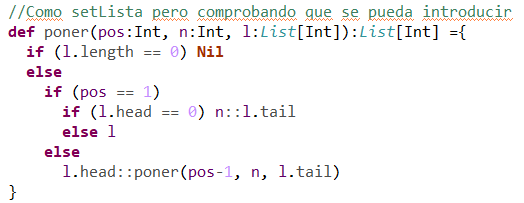
**Rellena Lista y Rellena Tablero:** Estos métodos están encargados de generar una lista de tamaño s\*s y llenarla de 0



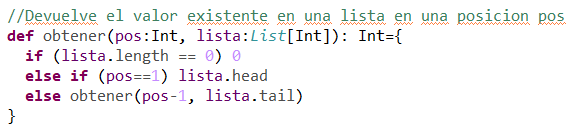
**Equivalentes:** Este método se encargar de comprobar si dos listas son de la misma longitud y contienen exactamente los mismos valores.



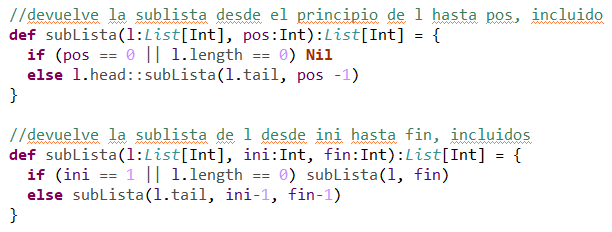
**Poner:** Este método se encarga de cambiar un valor en una posición de la lista por otro valor dado.



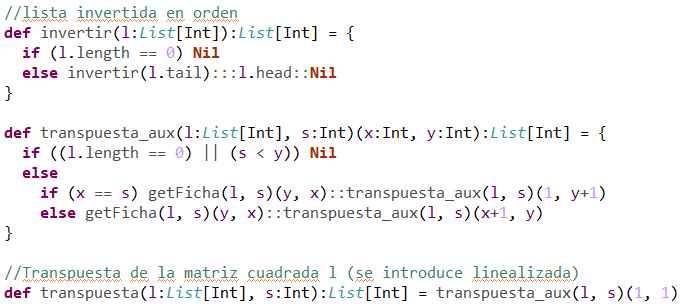
**Obtener:** Este método devuelve el valor existente en una determinada posición de la lista.



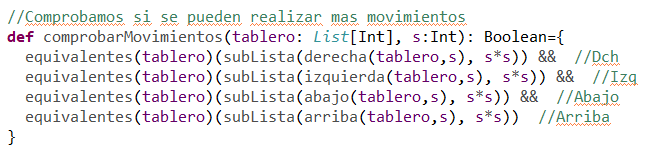
**SubLista:** Este método devuelve una sublista desde el incio hasta una determinada posición o desde una determinada posición hasta el final de una lista.



**Invertir y traspuesta:** Como ya hemos mencionado anteriormente estos métodos los utilizamos para devolver una lista al revés o como si fuera una matriz traspuesta.



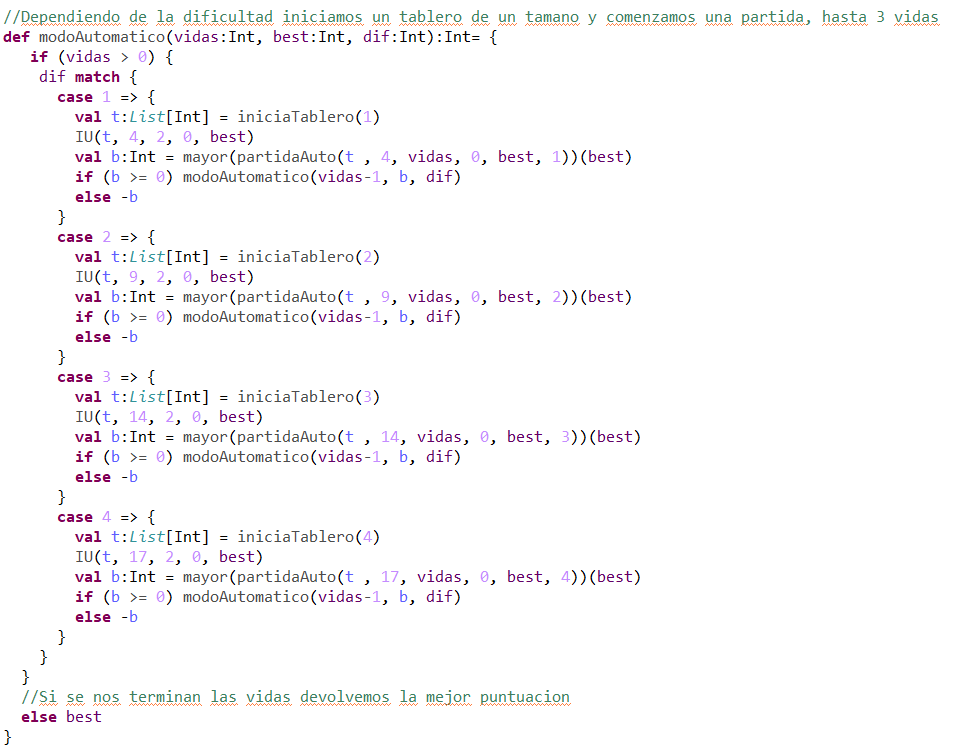
**Comprobar Movimientos:** Este método se apoya en la función mencionada anteriormente “equivalentes” para ver si se pueden seguir haciendo movimientos hacia cualquiera de las 4 direcciones.



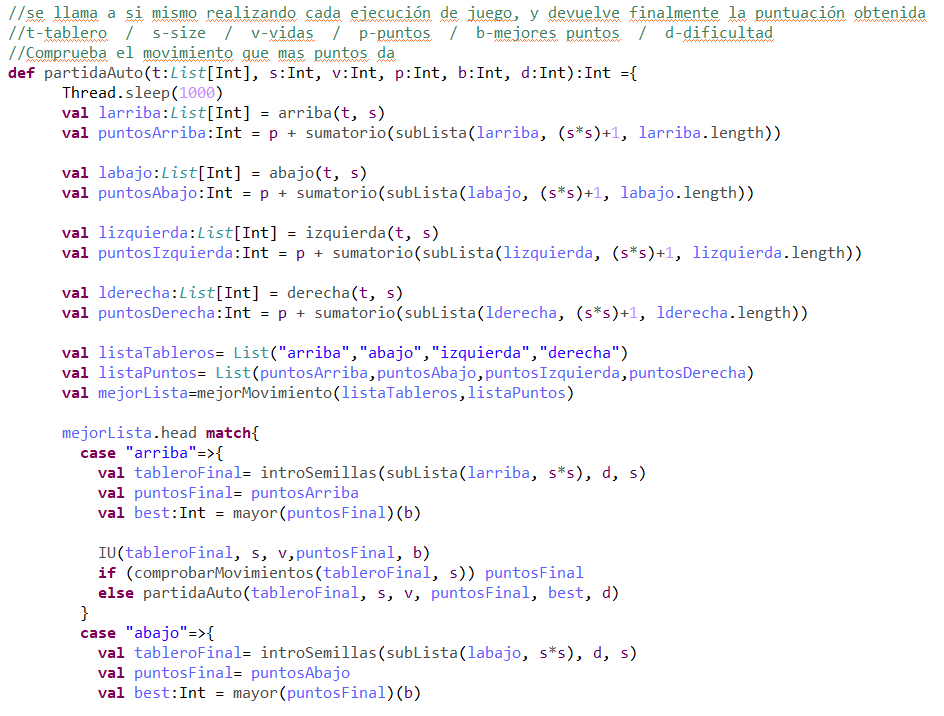
**2.- Optimización. Juego Automático.**

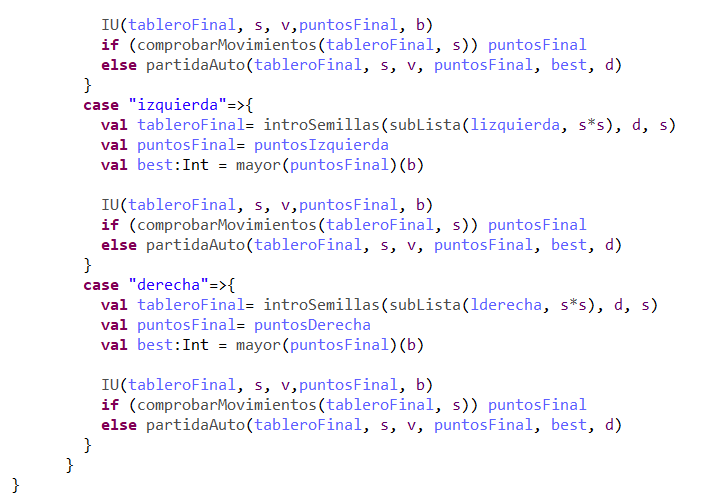
En la segunda parte de esta PECL2 se nos pedía que implementásemos un algoritmo que facilitase la opción de juego más optima al computador, por lo que realizamos la programación del modo Automático.

Como hemos mencionado al principio de esta memoria, el usuario puede seleccionar el modo al iniciar el juego, modos disponibles entre m (Manual) y a (Automático). Pues bien, cuando el usuario introduce a, comienza la ejecución del **modo Automático**.



En este método dependiendo de la dificultad iniciamos un tablero y realizamos la llamada al **método partidaAuto**.

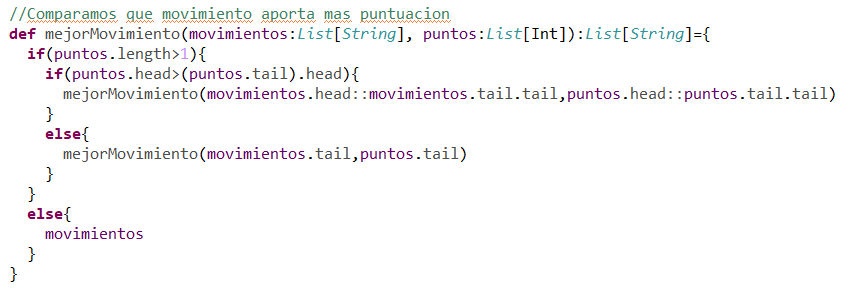




En el modo Automático, no juega la persona, si no la máquina. Además, para elegir la opción óptima en cada momento simulamos los movimientos en las 4 direcciones y realizamos aquel que otorgue más puntos a la máquina.

Para ello creamos los tableros y las puntuaciones correspondientes a los 4 movimientos y con el apoyo del método **mejorMovimiento** decidimos cual es mejor.

Dependiendo de cual sea mejor, realizamos una **llamada recursiva a partidaAuto** con el nuevo tablero escogido e introduciendo las semillas.



La llamada recursiva terminará cuando no se puedan realizar movimientos en ninguna dirección, al igual que en el modo Manual.

**3.- Trabajo Avanzado. Interfaz Gráfica**