Análisis del código

- `Juego(dificultadN: Int)`

  Se pasa como parámetro el nivel de dificultad del Juego.

  0: 3 colores
  1: 5 colores
  2: 7 colores

- `val colores: List[Char]`
  Lista con los 7 colores que podrá utilizar nuestro juego.

- `val dificultad: List[Int]`
  Lista con el número de colores que utilizará el juego dependiendo del nivel de dificultad.

- `val rnd: scala.util.Random`
  Objeto que utilizaremos para generar valores aleatorios en función de la dificultad del juego.

- `val sizeX: Int`
  Número de columnas.

- `val sizeY: Int`
  Número de filas.

- `var puntos: Int`
  Puntos conseguidos durante el juego.

- `def print(tabla: List[List[Char]]): Unit`
  Imprime el contenido de una tabla de juego.

- `def turno(tabla: List[List[Char]]): List[List[Char]]`
  Método que se ejecutará cada turno y llamará a los métodos de juego. Se llama a sí mismo de forma recursiva hasta que, tras la jugada, no hay fichas que eliminar.

  Comprueba que no hay fichas que eliminar comparando el tablero original con un tablero al que se le ha aplicado el proceso de borrado de fichas iguales. Si son iguales los dos tableros es que no hay cambios, con lo cual, acaba el turno.

- `def juegaTurno(tabla: List[List[Char]]: List[List[Char]]`
  Método de juego. Analiza la tabla, marca con 0 las fichas que hay que eliminar, añade puntos al juego y elimina los elementos, rellenando los huecos que queden con colores aleatorios.

  Analiza el tablero en vertical, y repite el proceso con la transpuesta (era más fácil que crear métodos de análisis horizontal.) Como marcamos como '0' las casillas que eliminaremos, podemos obtener rápidamente los puntos obtenidos.

  Después, borra los '0' y rellena los huecos superiores con colores aleatorios.

- `def cuentaElementos(c: Char, t: List[List[Char]])`
  Cuenta el número de ocurrencias de c en el tablero t.

- `def borraCoincidencias(tabla: LIst[List[Char]]): List[List[Char]]`

Marca como 0 los elementos que se repiten contiguamente 3 o más veces en una tabla.

- `def compactaYRellena(tabla: List[List[Char]]): List[List[Char]]`
  Elimina los ceros y rellena los huecos que quedan tras compactar con colores aleatorios.

- `def getMod(lista: List[Char], i: Integer, mod: Integer): List[Char]`
  Método de apoyo a 'transpose'.

- `def transpose(tabla: List[List[Char]]): List[List[Char]]`
  Transpone la tabla 'tabla' (como si fuese una matriz).

- `def creaTabla(n: Integer, y: Integer): List[List[Char]]`
  Crea una tabla aleatoria con 'n' columnas e 'y' filas.

- `def creaLista(n: Integer): List[Char]`
  Crea una lista aleatoria con 'n' elementos.

- `def nuevaLista(c: Char, n: Integer): List[Char]`
  Crea una lista de 'n' elementos y la rellena con caracteres 'c'.

- `def shorter(l1: List[Char], l2: List[Char]): List[Char]`
  Devuelve la lista más corta.

- `def todosIguales(list: List[Char]): Boolean`
  Comprueba si todos los elementos de una lista son iguales. Ponemos como marca que no compruebe si el elemento es 0 porque es el caracter con el que marcaremos los elementos repetidos.

- `def borraCoincidenciasLista(lista: List[Char]): List[Char]`
  Marca con 0 los elementos que hay que eliminar de la lista (con 3 o más coincidencias contiguas).

  Realiza el proceso con la lista y con la lista invertida, y devuelve la que más '0' tiene, es decir, la que más elementos repetidos tiene.

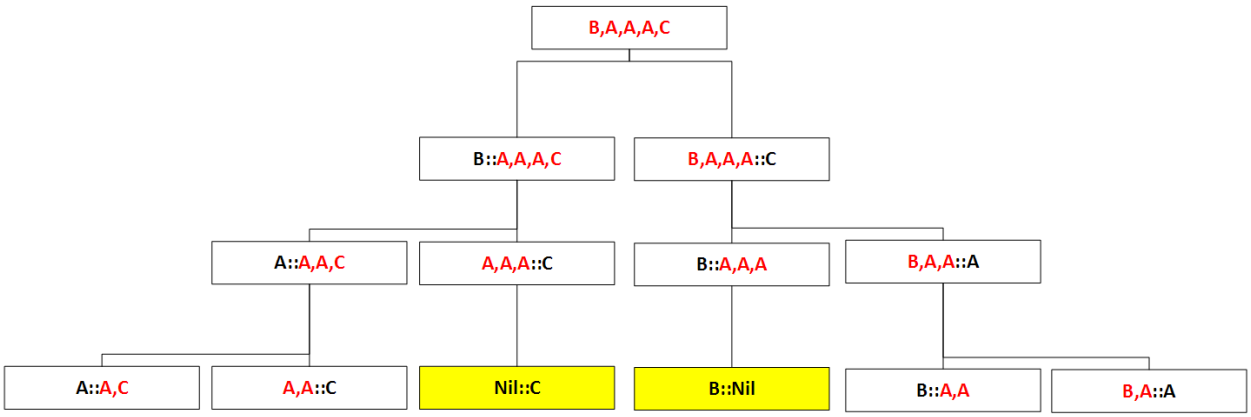
- `def borraCoincidenciasAux(list: List[Char]): List[Char]`
  Método auxiliar para ayudar en la recursividad del método *borraCoincidenciasLista*.

  Si el tamaño es menor que 3, devuelve la lista, porque no podremos eliminar nada con menos de 3 coincidencias.

  Si son todos iguales, devuelve una lista vacía (que representaremos como una lista de n 0's para poder utilizarlo tanto en horizontal como en vertical).

  Si no, devuelve el primer elemento concatenado al resultado de ejecutar el proceso *borraCoincidenciasLista* en la cola.

  Podemos ver el funcionamiento del método en el siguiente esquema.



- `def cuentaElementosLista(c: Char, lista: List[Char]): Integer`
  Cuenta el número de repeticiones de c en la lista l.

- `def compacta(lista: List[Char]): List[Char]`
  Borra los elementos marcados como 0 en una lista.

- `def compactaYRellenaLista(lista: List[Char]): List[Char]`
  Llama al método *compacta* y rellena la cabeza de la lista con colores aleatorios.