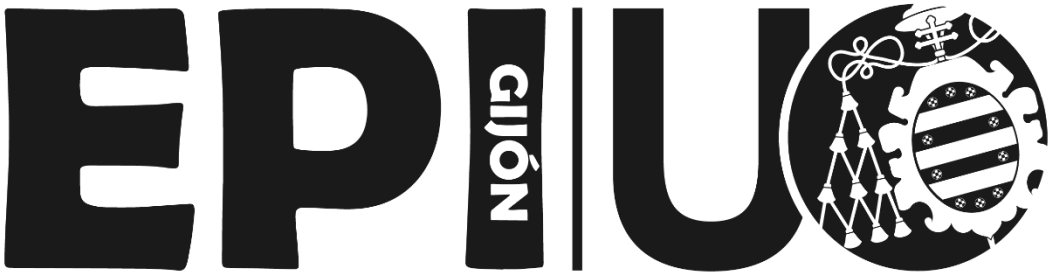

Universidad de Oviedo

SumaUno

Coronel Jaramilla, David Marcelo

Méndez Camino, Carmen

Escuela Politécnica de Ingeniería de Gijón, Universidad de Oviedo

Entregado el 15 de enero de 2024

En este trabajo se utilizan únicamente dos clases: la clase *Main* y la clase *Tablero*, donde la clase *Main* funciona para la inicialización, llamando a la clase *Tablero* y la representación de la interfaz gráfica llamando a la clase *JFrame* que abre la ventana principal con unos valores específicos para asignar el tamaño y su colocación, con el título **SumaUno**. También se utiliza la clase *JOptionPane* que se utiliza para interactuar con el usuario pidiéndole el número de filas y mostrándole la cantidad de filas seleccionadas.

Por otro lado, la clase *Tablero* es una subclase de *JPanel* que es una super clase predefinida y aquí se encuentran todos los métodos que componen a la aplicación. Al principio se definen los **atributos** que se van a utilizar:

- *filas*, que alberga el valor entero de las filas
- *valores*, es la matriz con números enteros de la que partimos
- *puntuacion*, será puntuación del juego
- *r*, se utiliza para generar valores aleatorios especialmente al llenar inicialmente la matriz del tablero
- *hayAlgunoIgual*, tipo boolean para comprobar si hay alguna casilla igual contigua a la que se ha pulsado para más tarde hacer los movimientos necesarios del juego
- *vida*, vida del jugador, parte del 3 (valor máximo)
- *estadoPartida*, sirve para la ampliación de **Detección de Fin de Partida**, donde comprueba el estado de la Partida y si es true la partida sigue funcionando y si es false significa que la partida ha terminado
- *ventana*, almacena la referencia a la ventana principal lo que permite cerrar la aplicación al finalizar la partida
- *éxito*, valor para comprobar los éxitos/combos para obtener una vida más en el juego, utilizada en la ampliación de recuperar vida.

Luego están los **métodos y constructores** que utilizan los atributos adecuados, aquí se encuentran:

- El constructor *Tablero*, inicializa el tablero con el número de filas especificado y llena la matriz con valores aleatorio.
- El método *llenarMatriz*(): Rellena la matriz del tablero con valores aleatorios entre 1 y 6 al inicio del juego

- El método *valorAleatorio* (int j, int i): Cambia los valores de las fichas y realiza ajustes en el tablero cuando se eliminan ciertas fichas, donde primero hacemos un bucle donde se marcan las casillas nulas (el número 0) ya que no nos interesan, y luego hacemos otro bucle para ir bajando casillas hasta que la casilla de número de fila más alto tenga un valor.
- El método *valorAleatorioAbajo* (int j, int i): Similar a *valorAleatorio*, pero diseñado para quitar de la forma más óptima los valores SUR.
- El método *color* (int valor): Devuelve un objeto *Color* correspondiente al valor de la ficha. Define colores basados en los valores de la matriz, se hace de forma gradiente de colores fríos a cálidos, aquí hacemos uso de la biblioteca *java.awt* esta importación permite el uso de la clase *Color* del paquete *java.awt* donde proporciona constantes que representan colores predefinidos, así como la capacidad de crear colores personalizados mediante combinaciones de valores RGB (Rojo, Verde, Azul).
- El método *setApp* (JFrame ventana): Asigna la ventana principal a la clase para permitir el cierre de la aplicación al finalizar la partida.
- El método *dibujarMatriz* (Graphics g): Dibuja visualmente el estado actual del tablero, coloreando y numerando las celdas, aquí usemos uso de la biblioteca *awt.Graphics* esta importación permite el uso de la interfaz *Graphics* del paquete *java.aw,t* la cual proporciona métodos para dibujar formas, texto y otros elementos gráficos en un componente visual.
- El método *puntuacionVidas* (Graphics g): Muestra la puntuación y la cantidad de vidas en la interfaz gráfica, que hace uso de la biblioteca awt.*Graphics* para dibujar el mensaje.
- El método *paintComponent* (Graphics g): Método estándar para dibujar cosas en java con *JPanel graphics* que es una clase que tiene distintas funciones para dibujar cosas en la "screen", la denominaremos como g, es como el pincel que usas para pintar *super.paintComponent*(g).
- El método *esIgual* (int j, int i): Comprueba si hay fichas iguales en direcciones específicas y realiza ajustes en el tablero, donde se crean nuevas variables como contador = 0 para que este contador multiplique luego el *valorBase*(aquí guardamos el valor base de la casilla de la que

partimos para no perderlo) para calcular la puntuación, también se crean valores fila y columna para guardar valores.

- El método *imprimirMatriz* (): Método utilizado para imprimir la matriz por motivos de prueba, como se indica en el enunciado.
- El método *MouseAdapter*: Clase interna que extiende *MouseAdapter*. Maneja los eventos del ratón, lo que permite al usuario interactuar con el juego mediante clics en las celdas del tablero ya que ratón al final va a desencadenar todo el juego, también se mira si el usuario ha ce correctamente los clics y si falla se le resta una vida en cambio sí acierta llamamos al método *jugadaValida*() para que lo contabilice en caso de que tenga menos de tres vidas.

Se hicieron **dos ampliaciones**:

- El método *jugadaValida*: Gestión de jugadas válidas y recuperación de vidas. Aquí cuando una jugada es válida y no tenemos el máximo de vidas, incrementamos el valor (*exito*) mirando primero si la variable vida es inferior a 3, luego r*esetearJugadasValidas* () pone a cero el valor (*exito*) cuando hemos recuperado una vida o cuando hemos cometido un fallo, *recuperarVidas* () hace que recuperamos una vida cuando hacemos 5 jugadas sin tener ningún fallo.
- El método *movimientosPosibles* (): Comprueba si aún existen movimientos posibles en el tablero para determinar el estado de la partida, y por tema de no salirnos de la matriz tendremos que mirar las direcciones una por una para saber si sigue habiendo movimientos posibles el cual devuelve un boolean que dice si sigue habiendo movimientos posibles en la partida.

Conforme a los **errores**, solamente hay una cosa que no funciona. Cuando se acumulan una gran de cantidad contiguas para hacer nulas no siempre se borran todas. Las más alejadas de la casilla principal (la que se le suma uno) no siempre se borran de forma correctamente.

A pesar de este error, el juego es perfectamente jugable, ya que sólo se da en casos relativamente aislados.