

21707 - Programación II: puzzle

Escola Politècnica Superior, 2023.
Dr. Estrany Bonnín, Bartomeu

<https://youtu.be/25eegqrzlaI>

Irene K. Blokker Ramírez
(DNI: 49605837N)

Blanca Atienzar Martínez
(DNI: 45185593T)

ÍNDICE:

1. Introducción	2
2. Puntos importantes	3
3.1 Diseño de clases	3
3.2 Diseño descendente	6
3. Métodos y clases más importantes	7
4. Conclusión	16

1. Introducció

La pràctica a realitzar consisteix en un Puzzle. Este "Puzzle" és un joc de habilitat i paciència on hem de recompondre una imatge seleccionada de manera aleatòria. La imatge serà dividida segons el nombre de divisions horitzontals i verticals introduïdes per l'usuari mitjançant una pestanya emergent (esta pestanya permet un diàleg amb l'usuari, a través de tres dades a introduir, el primer serà un "String" relacionat amb el nom de l'usuari per guardar la informació de la partida, un "int" relacionat amb el nombre de divisions verticals i un "int" relacionat amb el nombre de divisions horitzontals nomenats anteriorment).

En altres paraules, el joc, al igual que un puzzle tradicional, es basa en acertar la posició correcta de les subimatges provinents de les divisions de la imatge original (és a dir, són les divisions de la imatge elegida aleatòriament), de tal forma que si aciertes les posicions correctes de totes les subimatges guanyes la partida. Al finalitzar la partida, guanyaràs un nombre de punts corresponents al nombre de divisions de la partida (esta cifra guanyada es calcularà de la següent forma $[n^{\circ}\text{horiz} \times n^{\circ}\text{verti}]$, on $n^{\circ}\text{horiz}$ és el nombre de divisions horitzontals, i $n^{\circ}\text{verti}$ és el nombre de divisions verticals], se haurà d'acertar en un temps determinat, que també dependrà del nombre de divisions. Si no es fa abans de temps, es perd la partida.

Per jugar, s'ha de seleccionar una peça del puzzle (que serà una subimatge) que està en una posició concreta i posteriorment, seleccionar una altra peça del puzzle (subimatge) en una posició determinada, entre elles es intercanjaran les posicions i les subimatges.

Durant el transcurs de la partida, no és possible crear noves partides, ni canviar directori ni mirar el historial. És a dir, no és possible fer cap altra acció que no sigui moure les peces i jugar.

També caldria destacar que l'usuari pot elegir la carpeta de la qual s'escogen de manera aleatòria les fotos que vol utilitzar per jugar en este "Puzzle" fent ús del 'canvi de directori', el qual es pot accedir tant dels icones a la capçalera com de l'opció del menú desplegable.

Les opcions de Nova partida, veure historial general i selectiu es poden accedir tant mitjançant els botons del menú general de botons negres

del panel de la izquierda, como los iconos de la cabecera como por medio del menú desplegable.

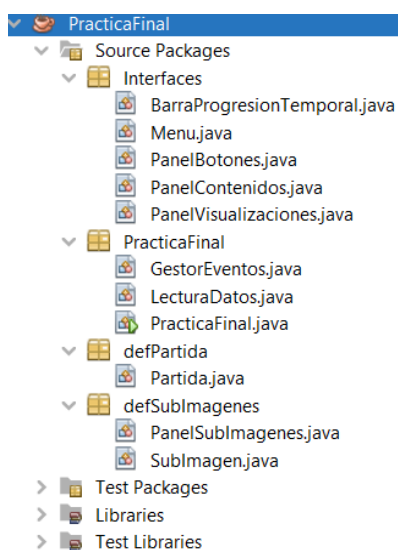
Para poder realizar esta práctica, necesitaremos los conceptos y las herramientas adquiridas en las clases teóricas y prácticas durante todo el curso de Programación II y lo realizaremos con el lenguaje de Programación explicado en clase, Java, y el entorno integrado de desarrollo NetBeans.

Además, hemos añadido un aspecto adicional que no tiene la versión básica, entre ellos: La barra de Menús Iconos, con sus respectivas imágenes y funcionalidades. Hemos introducido este aspecto adicional, puesto que así podemos facilitar la comprensión de las funcionalidades del juego con imágenes, y así poder hacer el juego más divertido y dinámico. Así como también hemos añadido la barra de progresión y las acciones que conllevan como calcular el tiempo de transcurso además de bloquear las acciones que no sea mover piezas en el transcurso de la partida.

*Aquellas palabras que están en verde son correspondientes a las clases que hemos creado, con un total de 11 clases que son las siguientes: **Menu**, **PanelBotones**, **PanelContenidos**, **PanelVisualizaciones**, **BarraProgresionTemporal**, **Partida**, **PanelSubImagenes**, **SubImagen**, **GestorEventos**, **LecturaDatos** y **PracticaFinal**.

2. Puntos importantes

3.1 Diseño de clases



Para poder organizar de una forma más clara nuestro programa “PracticaFinal”, hemos optado por un diseño descendente, con la creación de 11 clases, en las cuales cada una, está encargada de un fragmento del proyecto; sin embargo, están unidas entre ellas para su máxima contribución entre variables y facilitar el trabajo.

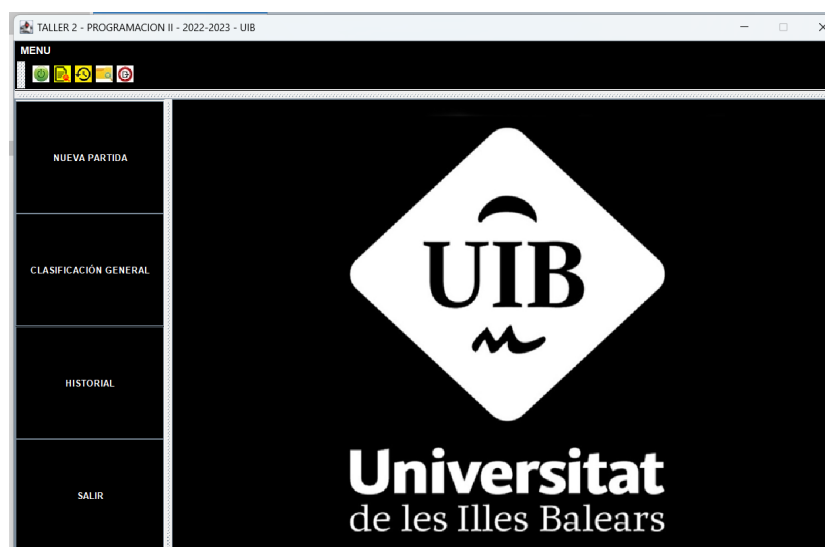
Para poder organizar mejor el proyecto hemos creado diferentes **packages**.

1. **Interfaces**: el cual contiene las clases que se encargan de las interfaces como su nombre indica, aunque en otras clases que no están en esta carpeta también se gestionan objetos relacionados con las interfaces.
2. **defPartida** : que a través de ella se instancian objetos Partida, y sus clases correspondientes a la gestión de la lectura y escritura de objetos Partida desde el fichero.
3. **defSubImágenes** : que contiene las clases encargadas de dividir la imagen y gestionar las subimágenes creadas colocándolas en una tabla de JLabel y permitiendo gestionar las acciones.
4. **nuevoireneyblanca**: que contiene las clases principales para leer los datos introducidos en la carpeta emergente, gestiona los eventos principales para cambiar de contenido y la clase principal que contiene el main para ejecutar el programa.

El proyecto comienza en la clase principal con el main que está en la clase **PracticaFinal** el cual extiende de un JFrame y es la ventana principal y en el cual con la línea:

`"setContentPane(panelContenidos)"`; // Asigna a panel Contenidos el panel de contenidos del JFrame de la clase principal.

En la ventana principal, que está organizada con paneles y separadores mediante la clase **PanelContenidos**, nos permite visualizar la ventana principal de la siguiente manera.



En la ventana puedes elegir la acción que se desea realizar a través del panel de Botones, la barra de menú o la barra de Iconos.

Para poder comenzar el juego creamos la clase **PracticaFinal** (**extends JFrame**), donde se crea la ventana y posteriormente la clase **PanelContenidos** (**extends JPanel**) ; Es la clase que ayuda a contener todos los componentes y separadores que se ven en la ventana, y crea una conexión que hace visible en la ventana y, por lo tanto, es una clase fundamental para la implementación del juego.

Por otro lado, la clase **LecturaDatos** (**extends JDialog**) se encarga de interpretar las respuestas introducidas por el usuario al formar una nueva partida (como lo son el nombre de usuario, el número de divisiones horizontales y divisiones verticales) lecturas de palabras introducidas, lecturas de fichero, manipular ficheros, y entre otros cambios de variables. Por lo que estas cuatro clases han sido bastante recurrentes durante todo el programa.

Para poder gestionar las acciones y la función que queremos lo hacemos a través de **GestorEventos**, la cual se encarga de llamar a cada clase según el botón seleccionada y así poder aplicar a cada objeto un evento. No obstante esta clase también debe contener las plantillas genéricas creadas para mostrar las diferentes funcionalidades que entre ellas también están implementadas en **PanelVisualizaciones** (**extends JPanel**).

En **GestorEventos**, logramos implementar todas las acciones de los botones generales en una clase, de forma que no lo hacemos repetidamente en sus diversas clases. En otro sentido, esta clase se utiliza para seleccionar las diferentes funciones y plantillas genéricas comentadas en el **PanelContenidos** , entre ellas está llama a **LecturaDatos** el cual, como hemos mencionado, implementa un **JDialog** para la interacción con el usuario, o acceder a los diferentes historiales.

Una vez elegida la opción de jugar la partida, se recorta la imagen en la clase **SubImagen** (**extends JPanel**), las subimágenes que se han creado en la clase **SubImagen** se guardan en un array de **JLabel** y se colocan aleatoriamente (**Random**) en **PanelSubImagenes** (**extends JPanel**), la cual también se encarga tanto de crear la tabla de **JLabel** sobre el cual se pondrá en cada casilla una subimagen, y también se encarga de la gestión de eventos (**Mouse Event**).

Las demás clases como **Menu**(**extends JPanel**) y **PanelBotones** (**extends JPanel**). Las hemos utilizado para simplificar la creación de paneles y subclases dentro de la clase principal y así garantizar una mejor organización. En ellas, se organizan los paneles tal y como dice su nombre para los botones de la izquierda en negro y el la barra de menú y los iconos. La clase **BarraProgresionTemporal** (**extends JProgressBar**) se usa como indica su nombre para gestionar las acciones y el aspecto de la barra de progresión.

3.2 Diseño descendente

Como hemos aprendido en clase la forma correcta para afrontar este proyecto, es crear un programa empezando por lo general, y debemos ir minimizando por cada nivel (lo que definimos como diseño descendente). Si entramos y estudiamos cada clase de nuestro programa, en todas podemos observar esta proporción de niveles.

En la clase principal (**PracticaFinal**), comenzamos con un método que construye una ventana donde se llevará a cabo todo el juego, a partir de este se confeccionan métodos más particulares como crear y clasificar los diferentes paneles introducidos a la ventana (en este caso **Menu**, **PanelBotones**, **PanelContenidos**, **PanelVisualizaciones**, **BarraProgresionTemporal**), leer un fichero y escribir sobre él (**Partida**), o la validación de cada variable de entrada nueva (estos aspectos o métodos van dirigidos niveles más particulares), entre otras clases y metodos. Están separados en diferentes clases para hacer más fácil la comprensión de otras personas que lean o modifiquen el código así como para nosotras mismas como programadoras para poder localizar y detectar más fácilmente los posibles errores y su origen.

Este ejemplo de diseño de descendente de lo podemos visualizar de igual forma por el resto de clases que también se caracterizan por empezar con un método genérico hasta métodos particulares.

Tenemos 2 clases para manejar todo lo relacionado a la Imagen con la que vamos a jugar. **Subimagen** recorta y divide la imagen en diferentes trozos, mientras que la clase **PanelSubimagenes** se encarga de gestionar de manera aleatoria su distribución y los eventos para poder jugar con la imagen.

3. Métodos y clases más importantes

- Clase **PracticaFinal**: es la clase principal que contiene el main, el cual extiende la clase "JFrame" en Java Swing. Esta clase se utiliza para crear una ventana de aplicación con una interfaz gráfica.
El constructor "PracticaFinal" se utiliza para inicializar la ventana de la aplicación. Aquí se configuran propiedades como el título de la ventana, el tamaño, la ubicación, el diseño del BorderLayout y la operación de cierre al hacer clic en el botón de cierre de la ventana. También se establece el panel de contenidos de la ventana utilizando el método "setContentPane" y se le asigna la instancia de "panelContenidos" que se asigna como el panel de contenidos de la ventana.
- Clase **GestorEventos**: en esta clase como menciona el nombre, se usará para gestionar eventos y acciones principalmente ActionListeners. Tiene declaradas variables estáticas: "nombreJugador", "divisionesHorizontales", "divisionesVerticales" y "RutaDirectorio" que se utilizarán para almacenar datos relacionados con la configuración de la partida y el directorio de las imágenes. Algunos métodos a destacar son: crearPartida(), cambiarDirectorio(), introducirDatos() y elegirFotorandom().
 1. crearPartida(): El método devuelve un objeto de tipo ActionListener. Este método se utiliza para manejar el evento de crear una nueva partida. Si no hay una partida en curso, se llama al método "introducirDatos()" que permite al usuario ingresar los datos de la partida. Si hay una partida en curso, se muestra un mensaje de advertencia.
 2. cambiarDirectorio: devuelve un objeto de tipo ActionListener. Este método se utiliza para manejar el evento de cambiar el directorio de las imágenes. Si no hay una partida en curso, se llama al método "obtenerDirectorio(clase)" que muestra un cuadro de diálogo para que el usuario seleccione un nuevo directorio. Si hay una partida en curso, se muestra un mensaje de advertencia.
 3. introducirDatos(): El método se utiliza para obtener los datos ingresados por el usuario para iniciar una partida. Utiliza la clase "LecturaDatos" para mostrar un cuadro de diálogo donde el usuario puede ingresar el nombre del jugador, el número de subdivisiones horizontales y el número de subdivisiones verticales. Si se ingresan datos válidos, se asignan a las variables correspondientes y se llama al método "Partida(divisionesHorizontales, divisionesVerticales)" en el

panel de contenidos para iniciar la partida. Si se ingresan datos incorrectos o se cancela el cuadro de diálogo, se muestra un mensaje de error.

4. `elegirfotorandom()`: El método se utiliza para seleccionar una imagen aleatoria del directorio de imágenes. Lee los archivos en el directorio, elige uno al azar y valida su extensión. Si se encuentra una imagen válida, se devuelve la ruta completa de la imagen. Si no se encuentra ninguna imagen válida, se muestra un mensaje de advertencia y se restablece el directorio al predeterminado.
- Clase **LecturaDatos**: en esta clase que extiende de `JDialog` se leerá los datos que introduzca el usuario. Muestra un cuadro de diálogo donde el usuario puede introducir datos para iniciar una partida en el juego. Algunos métodos relevantes a comentar es su constructor y `getDatosTexto()`.
1. Constructor: `LecturaDatos(JFrame frame, String[] campos)`: recibe como parámetros un `JFrame` y un array de `String` "campos". El `JFrame` se utiliza como el contenedor padre del cuadro de diálogo. El array de `String` "campos" representa los conceptos o enunciados de los datos que se deben introducir. El constructor crea un cuadro de diálogo modal con un título y un contenedor de contenidos. Luego, se crean y configuran los componentes del cuadro de diálogo, como etiquetas `JLabel` y campos de texto `TextField`, y se añaden al contenedor de contenidos. Finalmente, se muestra el cuadro de diálogo.
 2. `getDatosTexto()`: se utiliza para obtener los datos introducidos por el usuario en forma de un array de `String`. Itera sobre los campos de texto `TextField` y guarda el texto introducido en el array "datosIntroducidos". Si algún campo está vacío, muestra un mensaje de error y devuelve `null`. También realiza validaciones adicionales para el número de divisiones horizontales y verticales. Si hay algún error, muestra un mensaje de error y devuelve `null`. En caso contrario, devuelve el array de datos introducidos.

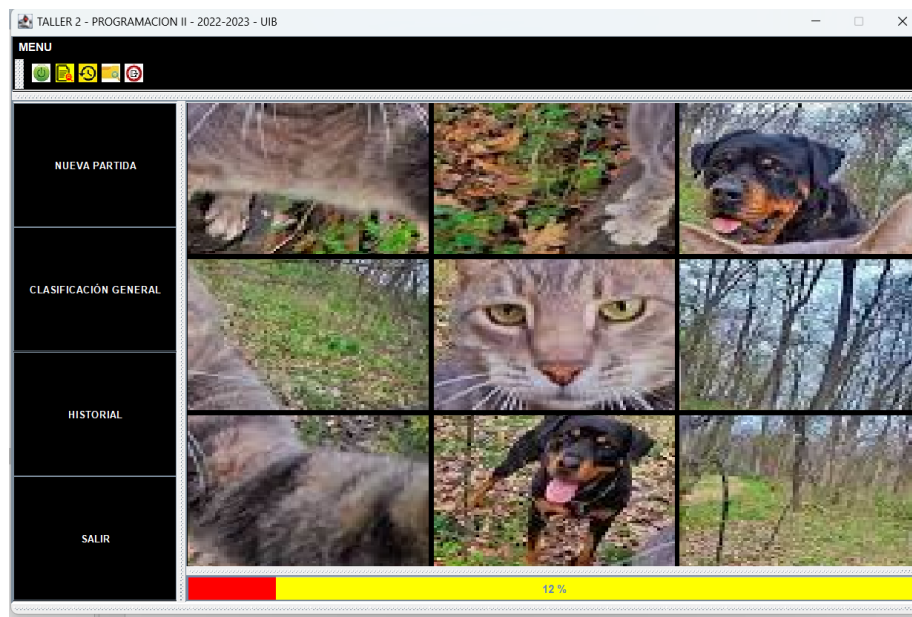


LecturaDatos

- Clase **SubImagen**: extensión de `JPanel`. Esta clase se utiliza para dividir una imagen en subimágenes más pequeñas y mostrarlas en forma de etiquetas `JLabel`.

1. El constructor de la clase "SubImagen" recibe como parámetro el nombre del archivo de imagen. Inicializa las variables y llama al método "dividirimag()" para dividir la imagen en subimágenes.
 2. dividirimag(): se utiliza para dividir la imagen en subimágenes más pequeñas. Primero, escala la imagen llamando al método "escalarimagen()". Luego, calcula el número total de subimágenes (chunks) y el ancho y alto de cada subimagen. A continuación, crea un array de BufferedImage para almacenar las subimágenes y utiliza un bucle anidado para recorrer todas las filas y columnas de la imagen original. Dentro del bucle, se crea una subimagen del tamaño adecuado y se dibuja utilizando Graphics2D. Cada subimagen se guarda en el array de BufferedImage.
 3. escalarimagen(): se utiliza para cargar la imagen original, escalarla utilizando el método "scaleImage()" y guardarla en la variable "scaledImage".
- Clase **PanelSubImágenes**: en esta clase que extiende de JPanel. Esta clase se utiliza para mostrar una cuadrícula de subimágenes sobre una cuadrícula de JLabel de JLabel llamada imag y permitir al usuario interactuar con ellas para resolver un rompecabezas. Un objeto panelSubImágenes representará el contenedor donde aparecen las diferentes subdivisiones en las que ha sido subdividida la imagen a solucionar.

La clase **PanelSubImágenes** tiene varios atributos, como imag (un array de JLabel que representa las subimágenes), subimagenes (un array de JLabel para las subimágenes desordenadas), contenedorNumerolimagenes y contenedorNumerolimagenesOrdenada (arrays para almacenar los índices de las subimágenes desordenadas y ordenadas respectivamente), subimagen1 y subimagen2 (las subimágenes seleccionadas para intercambiar), progresoThread (un hilo para controlar la barra de progreso temporal) y barraTemporal (una instancia de **BarraProgresionTemporal** que representa una barra de progreso visual).



panelSubImágenes

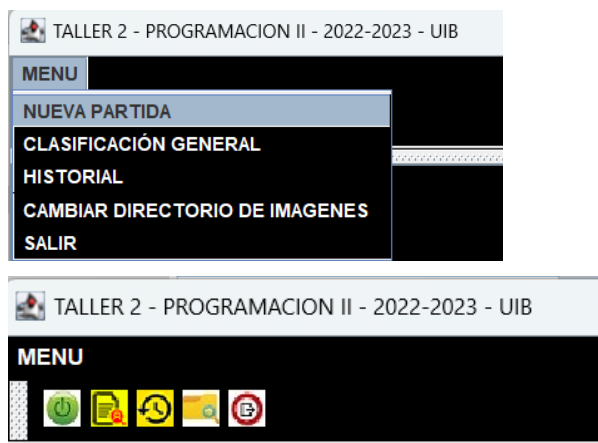
1. El constructor de la clase recibe una imagen y la cantidad de divisiones horizontales y verticales para crear la cuadrícula de subimágenes. Luego, se llama al método `setup()` para configurar el diseño del panel y al método `initComponents()` para inicializar y desordenar las subimágenes en la cuadrícula.
2. En el método `initComponents()`, se utiliza un objeto `Random` para generar números aleatorios y desordenar las subimágenes. Se recorren las subimágenes y se agregan al panel en posiciones aleatorias utilizando el método `add()`. También se agrega un `MouseListener` a cada subimagen para permitir la interacción del usuario. Después de desordenar las subimágenes, se verifica si la última subimagen no pudo ser colocada en su posición correcta debido al desorden. En ese caso, se agrega al panel.
3. A continuación, se crea una instancia de `BarraProgresionTemporal` y un hilo `progresoThread` que actualiza la barra de progreso en incrementos. Este hilo se ejecuta durante un tiempo determinado y, si alcanza el 100%, se muestra un mensaje indicando que el tiempo ha terminado.
4. El `MouseListener` permite al usuario seleccionar dos subimágenes para intercambiar. Cuando se selecciona una subimagen, se resalta con un borde rojo. Cuando se selecciona la segunda subimagen, se intercambian las posiciones y las imágenes de las subimágenes seleccionadas utilizando el método `intercambiarPosiciones()`. Luego, se reinician las variables de selección.
5. El método `intercambiarPosiciones()` recibe las dos subimágenes seleccionadas y las intercambia en el array `subimagenes`. También actualiza

las variables subimagen1 y subimagen2. Además, se actualiza la apariencia de las subimágenes intercambiadas.

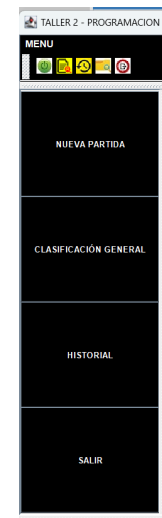
6. El método `actualizartabla()` actualiza el panel con los cambios realizados en las subimágenes. Si las subimágenes desordenadas y las subimágenes originales son iguales, significa que se ha completado el rompecabezas y se muestra un mensaje de felicitación. Si no, las subimágenes desordenadas se vuelven a pintar en el panel.
7. El método `sonArreglosIguales()` verifica si dos arrays de subimágenes son iguales. Se utiliza para comprobar si se ha llegado a la solución del rompecabezas.

- Clase **Menú, PanelBotones, PanelVisualizaciones**: Como hemos explicado anteriormente en estas clases lo más esencial es la creación de JPanel. Son clases en los que contienen los botones de la ventana tanto los del panel de la izquierda de Botones (PanelBotones), la barra de menú que es un menú desplegable con las opciones de Nueva Partida, Historial General, Historial Específico, Cambiar Directorio y Salir (Menu) o la barra de Iconos que tiene las mismas opciones que el Menu.

Menu:



PanelBotones:



Por otro lado, PanelVisualizaciones que también extiende de JPanel se utiliza para mostrar diferentes visualizaciones en una interfaz de usuario. La clase **PanelVisualizaciones** tiene varios métodos para mostrar diferentes paneles de visualización:

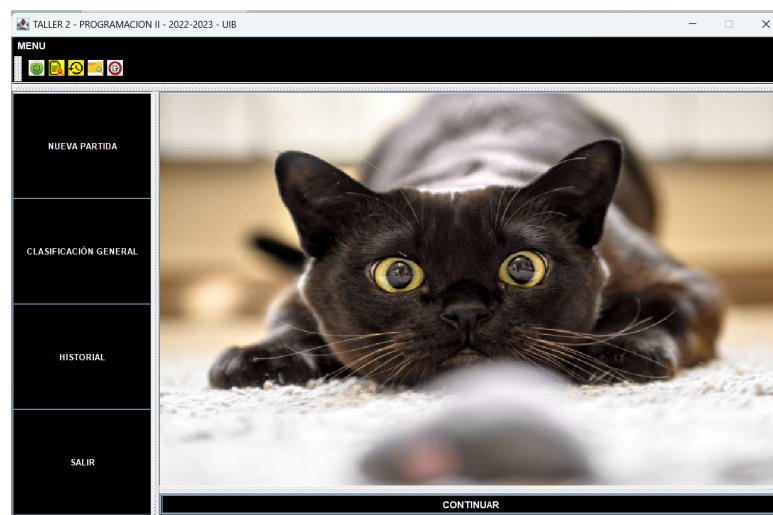
1. El método `panelHistorial()` crea un panel que muestra un historial de resultados. Es el encargado de seleccionar el tipo JTextArea que muestra "resultados.dat" que según si el nombre de usuario es null, significa que

mostrará el historial general, si el nombre de usuario es diferentes de null significado era que debe mostrar el historial específico del usuario, en ambos casos si no se encuentra el fichero o de otra forma si el fichero está vacío, saldrá un texto mostrando “No se ha encontrado”.

El contenido del historial se obtiene de un objeto Partida y se muestra en un JTextArea. El panel utiliza un JScrollPane para permitir el desplazamiento vertical si el contenido del historial es demasiado largo.

2. El método `panelStandby()` crea un panel que muestra una imagen de la UIB. La imagen se carga utilizando la clase `ImageIO` y se muestra en un `JLabel`.
 3. El método `panelPartida(int hori, int verti)` crea un panel que muestra la imagen para jugar dividida en subimágenes, es decir, la partida del juego del puzzle en el que se puede intercambiar las piezas y ordenarlas. El panel utiliza la clase `PanelSubImagenes` para mostrar las subimágenes en un diseño de cuadrícula.
 4. El método `panelImagenSolucion(String imag)` crea un panel que muestra la imagen solucionada en 2 situaciones distintas: al finalizar el tiempo de la partida o cuando se ha solucionado el juego. La imagen se carga utilizando la clase `ImageIO` y se muestra en un `JLabel`.
- Clase **PanelContenidos**: extiende de `JPanel`. Clase en el que se hace la organización para los paneles. Los diferentes paneles que puede enseñar son la imagen de la UIB, imagen de la solución y paneles historiales.
1. El método `setup()` configura el diseño del panel principal estableciendo un `BorderLayout` y un fondo negro.
 2. El método `initComponents()` inicializa los componentes de la interfaz principal. Se utiliza un `JSplitPane` para dividir la ventana verticalmente en dos secciones. En la parte superior se muestra un panel de menú (`Menu`) y en la parte inferior se muestra otro `JSplitPane` llamado `separador3`. Dentro de `separador3`, se coloca un panel de botones (`PanelBotones`) en la parte izquierda y un panel de visualización (`PanelVisualizaciones`) con la imagen de standby (UIB) en la parte derecha.
 3. El método `panelImagenSolucion()` se utiliza para mostrar el panel principal cuando se ha resuelto la imagen o el usuario ha abandonado la partida. Se crea un nuevo `JSplitPane` llamado `separador4` para dividir la ventana verticalmente en dos secciones. En la parte izquierda se muestra un panel de visualización (`PanelVisualizaciones`) con la imagen solucionada (`nombreFoto`). En la parte derecha se coloca un botón "CONTINUAR"

- (JButton) y se configura un ActionListener para que realice una acción cuando se presiona el botón.
4. El método Partida(int hori, int verti) se utiliza para mostrar el panel principal cuando se está jugando una partida. Se crea un nuevo JSplitPane llamado separador4 para dividir la ventana verticalmente en dos secciones. En la parte izquierda se muestra un panel de visualización (PanelVisualizaciones) con la imagen dividida en subimágenes para jugar (hori y verti). En la parte derecha se coloca una barra temporal (barraTemporal).
 5. El método Historial(String nombreUsuario) se utiliza para mostrar el panel principal cuando se visualiza el historial. Se coloca un panel de botones (PanelBotones) en la parte izquierda y un panel de visualización (PanelVisualizaciones) con el historial correspondiente al nombreUsuario en la parte derecha.



panellImagenSolucion

- Clase **Partida**: se usará para registrar y manipular los datos de una partida que posteriormente se escribirá en un fichero "resultados.dat" y también para poder leer el fichero, para enseñarlas en el historial , tanto general como específico. Para poder acceder al fichero hacemos uso de la clase RandomAccessFile y realizar operaciones de lectura y escritura en él.

1.Atributos:

- **FORMAT**: Es una cadena de formato que define la estructura en la que se guardarán los datos en el archivo.
- **file**: Es una cadena que representa el nombre del archivo en el que se almacenarán los datos.
- **nombreJugador**: Es una cadena que contiene el nombre del jugador.
- **fecha**: Es una cadena que representa la fecha y hora en que se registró la partida.

- puntos: Es un entero que almacena la puntuación obtenida en la partida.
 - longExacta: Es un entero que representa la longitud exacta (en caracteres) para almacenar el nombre del jugador en el archivo.
 - midaReg: Es un número entero que representa la longitud total de un registro en el archivo, teniendo en cuenta la longitud del nombre del jugador, la fecha y los puntos.
2. Constructor:
- El constructor `Partida(String fichero, String nombre, int puntuación)` recibe el nombre del archivo, el nombre del jugador y la puntuación de la partida. Asigna los valores correspondientes a los atributos `file`, `nombreJugador`, `fecha` y `puntos`.
3. Método `insertirRegistre()`:
- Este método se utiliza para insertar un registro en el archivo de resultados. Abre el archivo en modo de lectura y escritura (`rw`), comprueba su longitud y busca el final del archivo (`f.length()`) utilizando `seek()`. Luego, se asegura de que el nombre del jugador tenga la longitud exacta (`longExacta`) agregando espacios en blanco adicionales si es necesario. A continuación, se escriben el nombre del jugador, la fecha y los puntos en el archivo utilizando los métodos `writeChars()` y `writeInt()` de `RandomAccessFile`.
4. Método `contingutFitxer()`:
- Este método se utiliza para obtener el contenido completo del archivo de resultados. Abre el archivo en modo de lectura (`r`) y recorre todos los registros en el archivo. Para cada registro, lee el nombre del jugador, la fecha y los puntos utilizando los métodos `readChar()` y `readInt()` de `RandomAccessFile`. Luego, utiliza `String.format()` para formatear los datos del registro según el formato definido en `FORMAT` y los agrega a la cadena texto. Al final, retorna la cadena texto que contiene el contenido completo del archivo.
5. Método `filtreNombre(String name)`:
- Este método se utiliza para filtrar los registros del archivo según el nombre del jugador. Recibe el nombre como parámetro y devuelve los registros que coinciden con el nombre. Abre el archivo en modo de lectura (`r`) y recorre todos los registros en el archivo. Para cada registro, lee el nombre del jugador, la fecha y los puntos utilizando los métodos `readChar()` y `readInt()` de `RandomAccessFile`. Compara el nombre del jugador leído con el nombre proporcionado, ignorando los espacios en blanco adicionales al final. Si el nombre coincide, agrega los datos del registro formateados en la cadena texto. Al final, retorna la cadena texto que contiene los registros filtrados según el nombre.

TALLER 2 - PROGRAMACION II - 2022-2023 - UIB

MENU			
NUEVA PARTIDA	Nombre: RENE	Fecha: 10:18 27/06/2023	Puntos: 2
	Nombre: RENE	Fecha: 10:19 27/06/2023	Puntos: 2
	Nombre: PEPE	Fecha: 10:20 27/06/2023	Puntos: 2
	Nombre: RENE	Fecha: 10:20 27/06/2023	Puntos: 2
	Nombre: RENE	Fecha: 10:20 27/06/2023	Puntos: 2
CLASIFICACIÓN GENERAL	Nombre: RENE	Fecha: 10:20 27/06/2023	Puntos: 2
	Nombre: RENE	Fecha: 10:20 27/06/2023	Puntos: 3
	Nombre: ele	Fecha: 17:28 27/06/2023	Puntos: 2
	Nombre: ele	Fecha: 17:28 27/06/2023	Puntos: 0
	Nombre: er12	Fecha: 17:34 27/06/2023	Puntos: 0
HISTORIAL	Nombre: 2	Fecha: 17:34 27/06/2023	Puntos: 0
	Nombre: niquel	Fecha: 19:24 27/06/2023	Puntos: 0
	Nombre: irene	Fecha: 19:25 27/06/2023	Puntos: 4
	Nombre: blanca	Fecha: 11:45 28/06/2023	Puntos: 0
	Nombre: iii	Fecha: 11:46 28/06/2023	Puntos: 2
SALIR	Nombre: k3nefsck	Fecha: 11:46 28/06/2023	Puntos: 0
	Nombre: irene	Fecha: 02:22 30/06/2023	Puntos: 0
	Nombre: irene	Fecha: 02:23 30/06/2023	Puntos: 9

panelHistorial

4. Conclusión

Personalmente, nos ha resultado un poco difícil esta práctica, por lo que nos ha obligado a dedicarle muchas horas, debido a que las dos participantes que integramos el grupo hemos empezado este año a programar.

Nos hemos encontrado con distintas dificultades, a medida que intentábamos encontrar soluciones; surgían nuevos problemas que hemos resuelto de distintas formas, por lo que con este proceso hemos aprendido a resolverlas mejor poco a poco.

En cambio, es cierto que gracias a esta práctica nos ha ayudado a incrementar nuestra sabiduría a la hora de programar en Java, puesto que es un trabajo extenso que abarca algunos temas que hemos dado en clase.

En esta práctica, hemos aprendido a manejar la clase "Partida", donde se implementa la estructura de ficheros con el RandomAccessFile con gran fluidez, así como PanelSubImágenes y SubImagen junto a LecturaDatos con los cuales manejamos la interpretación de las respuestas dadas por el usuario, división de imágenes, y entre otras su colocación, los cuales al principio nos resultaron difíciles de entender y trabajar. Al principio tampoco sabíamos si era buena idea el número de clases que habíamos creado para organizarlo pero nos facilitó la creación de clases para poder detectar errores así como para organizarnos las integrantes del grupo.

En la primera entrega, al profesor no le funcionaba correctamente el juego, puesto que no podía mover correctamente las piezas, y supusimos que el problema residía en el método que se encuentra en PanelSubimágenes en donde hemos cambiado MouseClicked por MousePressed y algunos cambios más sobre los métodos. Además, teníamos fallos que daban error cuando se hacía click dos veces sobre la misma pieza o cuando se cambiaba el directorio. Creemos que hemos arreglado los fallos.

Finalmente, gracias a este proyecto hemos podido mejorar en el ámbito de la programación sobretodo hemos ganado control sobre la gestión de ventanas y paneles. Sabemos que nuestro Puzzle es bastante similar al original, pero en su sencillez abunda su perfección. Por lo que, opinamos que se trata de una herramienta de aprendizaje, nos ha sido de gran ayuda hacerlo y, pensamos que este tipo de trabajos finales de Programación son necesarios para asimilar bien los conceptos impartidos durante el curso.