Memoria TETRIS

- 21707 - Programació II -

Escola Politècnica Superior (UIB), 2024.

https://youtu.be/8NzUEab8ee0

Blanca Atienzar Martínez, [redacted]

Hai Zi Bibiloni Trobat, [redacted]

ÍNDICE:

1. Introducción

En esta práctica de la asignatura Programación II, curso 2023-2024, se nos encomienda el desarrollo de una aplicación que permita jugar al Tetris UIB. El objetivo principal es implementar una interfaz gráfica de usuario (GUI) y las funcionalidades necesarias utilizando las librerías gráficas de Java, aplicando los principios de la programación orientada a objetos y el manejo de tipos abstractos de datos.

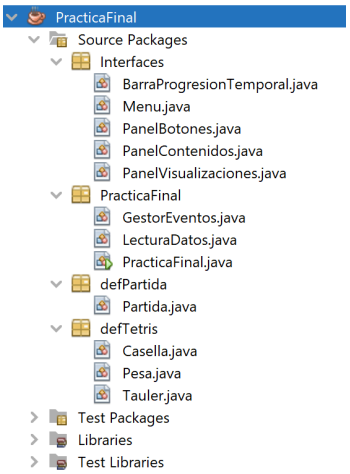
La aplicación debe incluir varias funcionalidades clave:

- Nueva Partida: El usuario puede iniciar una nueva partida mediante un botón, un icono, o una opción del menú desplegable. Se solicitará el nombre del jugador, y al empezar la partida, el tiempo comenzará a contarse, finalizando cuando el cronómetro llegue a su fin. La puntuación se incrementa al llenar filas o columnas del tablero, y se pueden realizar rotaciones o generar nuevas formas, lo cual afecta negativamente la puntuación.
- Configuración: Permite al usuario modificar parámetros del juego como la puntuación por eliminar casillas, rotar formas, generar nuevas formas, y la imagen de las casillas. También se puede ajustar la duración de la partida.
- Historial: Muestra el historial de partidas realizadas, almacenadas en un archivo *partidasTetrisUIB.dat*.
- Información: Proporciona información sobre el juego y la creación de este.
- Salir: Permite cerrar la aplicación.

El desarrollo de esta práctica tiene como objetivos consolidar los conocimientos en la creación de interfaces gráficas y en la implementación de funcionalidades mediante programación orientada a objetos. Además, se busca reforzar la capacidad de diseñar y desarrollar aplicaciones completas, abarcando desde la estructura del código hasta la presentación final del proyecto.

2. Puntos Importantes

2.1. Diseño de clases



Para poder organizar de una forma más clara nuestro programa "PracticaFinal", hemos optado por un diseño descendente, con la creación de 12 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. **defTetris**: que contiene las clases encargadas de crear las piezas, el tablero y las casillas y gestionar las operaciones que realiza cada elemento del juego.
4. **PracticaFinal**: 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) 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 la llamada 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 crean las piezas en la clase Pesa (extends JPanel), las piezas se guardan en una matriz(int [][]). Se selecciona la pieza con la que jugar aleatoriamente (Random) con un switch. Dentro de la clase también encontramos MouseEvent que gestiona las acciones que realizan las piezas durante la partida.

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.

2.2. Diseño ascendente

Como hemos aprendido en clase, la estrategia adecuada para abordar este proyecto implica iniciar con un enfoque general y luego ir refinando a cada nivel, siguiendo lo que llamamos un diseño descendente. Al estudiar cada clase de nuestro programa, notamos que este enfoque se refleja en todos los niveles.

En la clase principal (PracticaFinal), comenzamos con un método que establece la ventana principal donde se desarrollará el juego. A partir de aquí, se definen métodos más específicos, como la creación y organización de los distintos paneles que se integran en la ventana (tales como Menú, PanelBotones, PanelContenidos, PanelVisualizaciones, Barra Progresión Temporal), la lectura y escritura de archivos (Partida), así como la validación de cada variable de entrada nueva. Estos aspectos o métodos se enfocan en niveles más particulares. Además, se organizan en diferentes clases para facilitar la comprensión tanto para otras personas que revisen o modifiquen el código, como para nosotras mismas como programadoras, permitiéndonos localizar y corregir errores más fácilmente.

Este enfoque de diseño descendente se aplica de manera consistente en el resto de las clases, las cuales también comienzan con un método genérico y luego desarrollan métodos más específicos.

Por ejemplo, contamos con 3 clases para manejar todo lo relacionado con la creación del tablero, las piezas y sus funciones. Pesa crea las distintas piezas con las que se puede jugar, las elige de forma aleatoria y además están definidas las distintas operaciones que puede llevar a cabo.

3. Métodos y clases más importantes

- La clase PracticaFinal es la piedra angular del programa, alberga el método principal y se extiende desde JFrame en Java Swing. Esta clase se encarga de generar una interfaz gráfica para la aplicación. En su constructor, PracticaFinal, se establecen las configuraciones iniciales de la ventana, tales como el título, tamaño, ubicación y el diseño del BorderLayout. Además, se define la acción a realizar al cerrar la ventana. Por medio del método setContentPane, se asigna el panel de contenidos de la ventana, utilizando la instancia panelContenidos. Este último se designa como el panel principal que contiene los elementos visuales de la interfaz.

- La clase Partida se usará para escribir y leer del fichero “partidaTetrisUIB.dat”, el fichero del historial, para ello, utilizaremos la clase RandomAccessFile. Tenemos el método inserirRegistre() que escribe en el fichero los datos de la partida, el método contingutFitxer() que obtiene el contenido completo del fichero y el método filtreNombre() que es el encargado de filtrar los datos del fichero según el nombre del jugador.

- La Clase Tauler está diseñada específicamente para la creación de una tabla y sus correspondientes funcionalidades. Incluye variables estáticas como "DIMENSIO", “MAXIM” y “COSTAT”, que almacenan información crucial sobre la tabla, como el número de cuadros por lado, el tamaño de cada recuadro y el total. Entre los métodos más destacados se encuentra:
    - algunaFilaOColumnaTotalmenteOcupada(): Este método se encarga de verificar si alguna fila o columna en la tabla está completamente ocupada y, en caso afirmativo, actualizar el puntaje y limpiar esa fila o columna.

- La Clase Pesa está destinada a la creación de piezas o formas aleatorias mediante el método pecesAleatories(). Además, contiene las funcionalidades y ActionListener necesarios para el correcto funcionamiento del juego. Entre las variables estáticas se incluye figuraAle, que almacena una matriz representando la forma de la pieza mediante unos y ceros, donde uno indica una casilla ocupada y cero una casilla libre, proporcionando así información esencial para el juego. Entre los métodos más importantes se destacan:
    - El constructor Pesa inicializa una nueva instancia de la clase Pesa. Establece la ubicación inicial de la pieza, configura su color de fondo y crea una matriz de celdas. Además, genera una pieza aleatoria, añade

escuchadores de eventos de ratón y llama a un método para inicializar las casillas.

- mousePressed(): Este método (implementado de MouseListener) se ejecuta cuando se presiona el botón del ratón sobre la pieza. Su procedimiento es: Principalmente, verificación y ajusta la posición del ratón, y por último registra la posición del ratón y calcula de la casilla seleccionada.
- mouseDragged(): Este método (implementado de MouseMotionListener) se llama mientras se arrastra la pieza con el ratón. Su procedimiento es: Principalmente, calcula el desplazamiento y actualiza la posición de la pieza a través del repaint().
- mouseReleased(): Este método (implementado de MouseListener) se ejecuta cuando se suelta el botón del ratón. Su procedimiento es: Principalmente, obtener la posicion del raton relativa al tablero, luego verifica si la posición está dentro de una casilla válida, y por último, coloca la pieza en el tablero y genera una nueva pieza
- pecesAleatories(): El método pecesAleatories() es responsable de generar una pieza de forma aleatoria para el juego, asignando una de varias posibles configuraciones de una matriz de 3x3 que representa la pieza con unos y ceros. Además, este método inicializa las celdas correspondientes a la pieza en el juego, garantizando que la pieza esté correctamente configurada y lista para ser utilizada.
- rotar(): El método rotar(), gira la figura actual 90 grados en sentido horario. Para ello crea una nueva matriz auxiliar y le pasa de forma rotada la figuraAle.
- inicializarCasellas(): Este método inicializa las casillas de acuerdo con la figura actual.

- La Clase GestorEventos está destinada, como su nombre indica, a la gestión de eventos y acciones, principalmente ActionListeners. Contiene variables estáticas como "nombreJugador", la cual almacena el datos introducido, es decir, el nombre del usuario. Entre los métodos destacados se encuentran: crearPartida(), configuracionJuego(), informacionJuego(), estadisticas() y configuracionJuego().
    - crearPartida(): Este método devuelve un objeto de tipo ActionListener y se utiliza para gestionar 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 necesarios para la partida. En caso de que ya haya una partida en curso, se muestra un mensaje de advertencia.

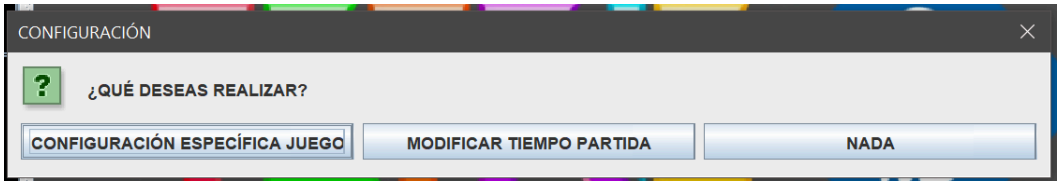
- configuracionJuego(): Este método devuelve un objeto de tipo ActionListener y se encarga de configurar el juego. Si no hay una partida en curso, se crea una nueva instancia de LecturaDatos con la configuración de la partida y se inicializan los componentes del panel de contenidos. Si hay una partida en curso, se muestra un mensaje de advertencia indicando que debe finalizarse la partida actual antes de configurar una nueva.
- informacionJuego(): Este método devuelve un objeto de tipo ActionListener y se encarga de mostrar la información del juego. Si no hay una partida en curso, se llama al método Informacion() del panel de contenidos para mostrar los detalles del juego. Si hay una partida en curso, se muestra un mensaje de advertencia indicando que se debe finalizar la partida actual antes de ver la información del juego.
- estadisticas(): Este método devuelve un objeto de tipo ActionListener y se utiliza para gestionar la visualización de estadísticas. Si no hay una partida en curso, se presenta un cuadro de diálogo con opciones para ver el historial general o el historial específico de un jugador. Según la elección del usuario, se muestra el historial general o se solicita el nombre de un jugador para mostrar su historial específico. Si hay una partida en curso, se muestra un mensaje de advertencia indicando que se debe finalizar la partida actual antes de acceder a las estadísticas.

- 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.

  Tenemos 2 constructores:

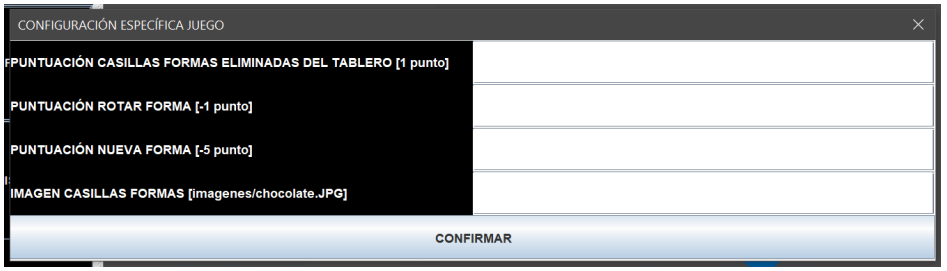
  - LecturaDatos(JFrame frame): crea un cuadro de diálogo modal para que el usuario introduzca el nombre del jugador. Configura el diálogo con un campo de texto y una etiqueta, ambos añadidos a un panel. También incluye un botón de confirmación que valida la entrada del usuario: si el campo está vacío, muestra un mensaje de error y reinicia los componentes; si no, guarda el nombre del jugador y marca que hay una partida en curso. El diálogo se ajusta en tamaño, se centra en la ventana principal y se hace visible.

- LecturaDatos(JFrame frame, String nada): crea un cuadro de diálogo modal para configuraciones específicas del juego o modificación del tiempo de partida. Presenta un cuadro de diálogo con tres opciones: "CONFIGURACIÓN ESPECÍFICA JUEGO", "MODIFICAR TIEMPO PARTIDA" y "NADA". Según la elección del usuario, se llama al método correspondiente para la configuración específica del juego o la modificación del tiempo. Finalmente, el cuadro de diálogo se hace invisible.



Además también encontramos estos métodos destacados:

- getNombreJugador(): devuelve el nombre del jugador ingresado como una cadena de texto (String) obtenida del campo de texto.
- configuracionEspecifica(): abre un cuadro de diálogo para ajustar configuraciones específicas del juego. El diálogo, que no es redimensionable, contiene etiquetas y campos de texto que permiten al usuario ingresar valores para diferentes parámetros del juego, como la puntuación por eliminar formas, rotar formas, y la ruta de la imagen de las casillas. Las etiquetas muestran los valores actuales y los campos de texto permiten la entrada de nuevos valores. Un botón de confirmación valida y guarda estos valores; si son correctos, actualiza las configuraciones del juego. También verifica que la ruta de la imagen proporcionada sea válida y no un directorio, mostrando un mensaje de error si es necesario. Finalmente, el cuadro de diálogo se ajusta en tamaño, se centra en la ventana principal y se hace visible.

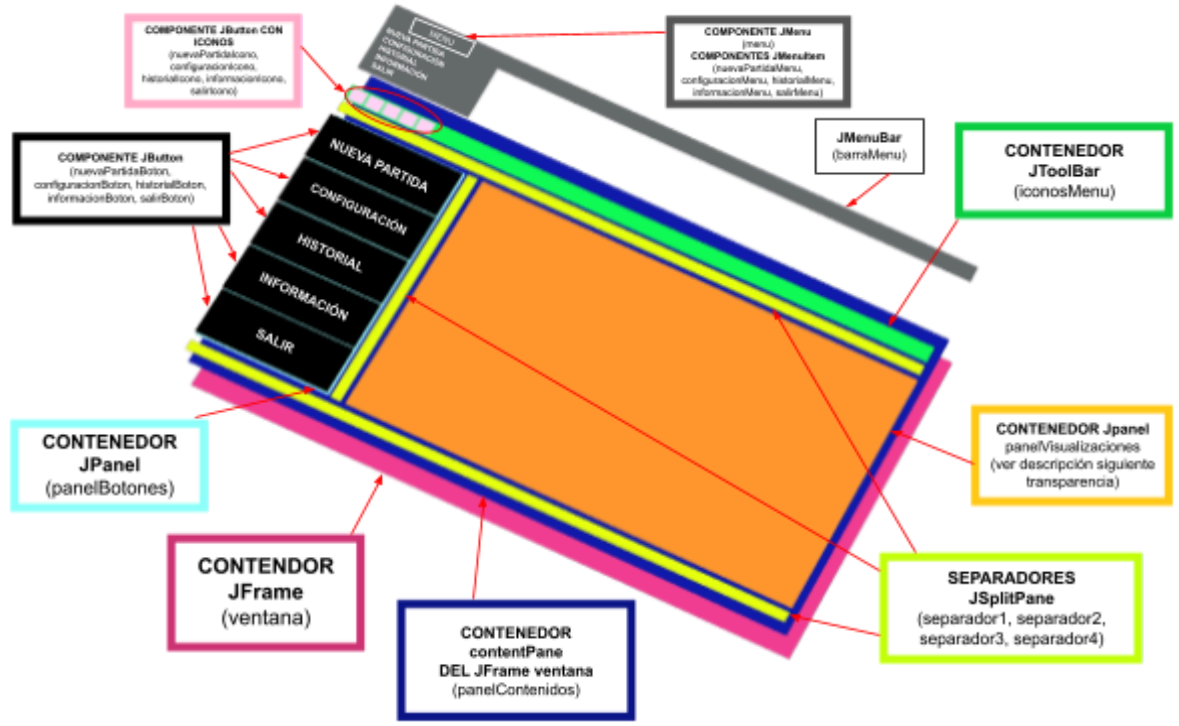


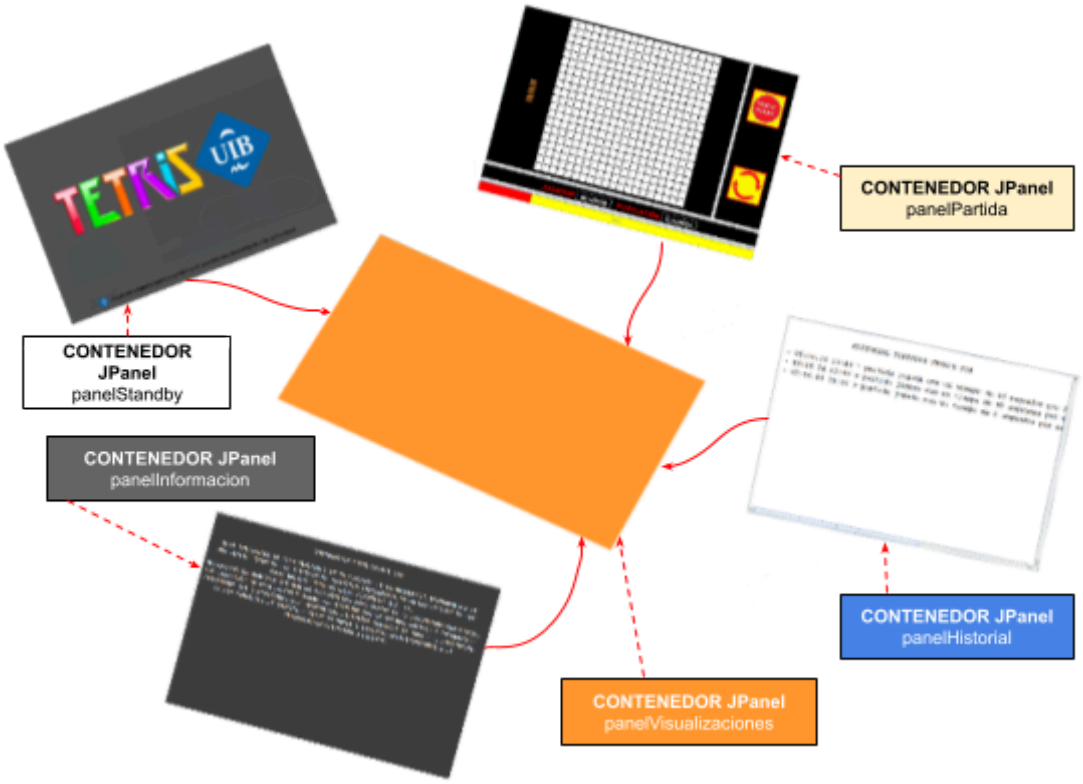
- configuracionTiempo(): establece un cuadro de diálogo para modificar el tiempo de la partida. El diálogo, sin posibilidad de redimensionar, muestra el tiempo actual de la partida y un campo de texto para

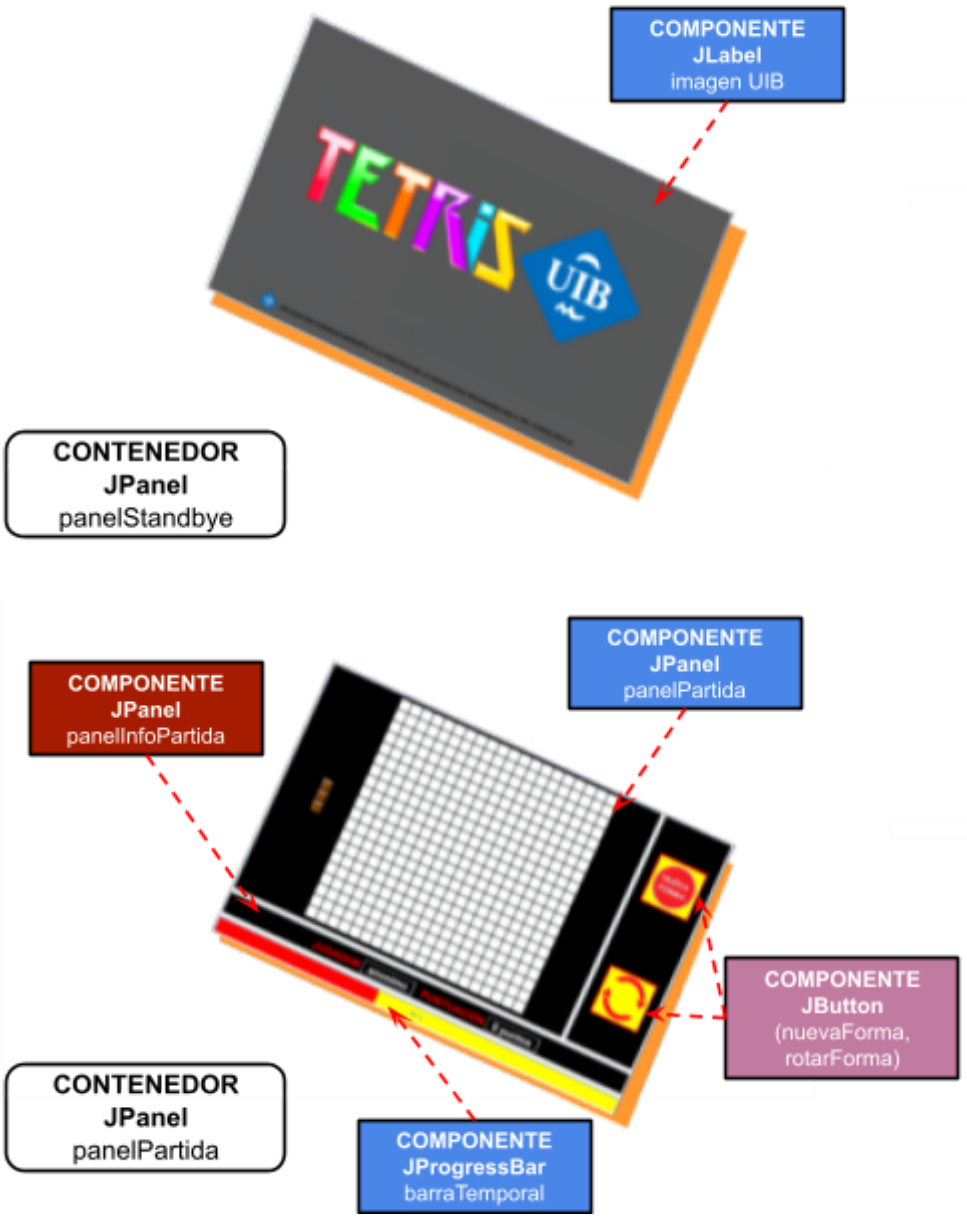
ingresar un nuevo valor. Cuando se hace clic en el botón "CONFIRMAR", se valida que el valor ingresado sea un número entero válido y no negativo. Si es válido, se actualiza el tiempo de la partida. El cuadro de diálogo se ajusta en tamaño, se centra en la ventana principal y se hace visible para que el usuario pueda realizar la modificación.

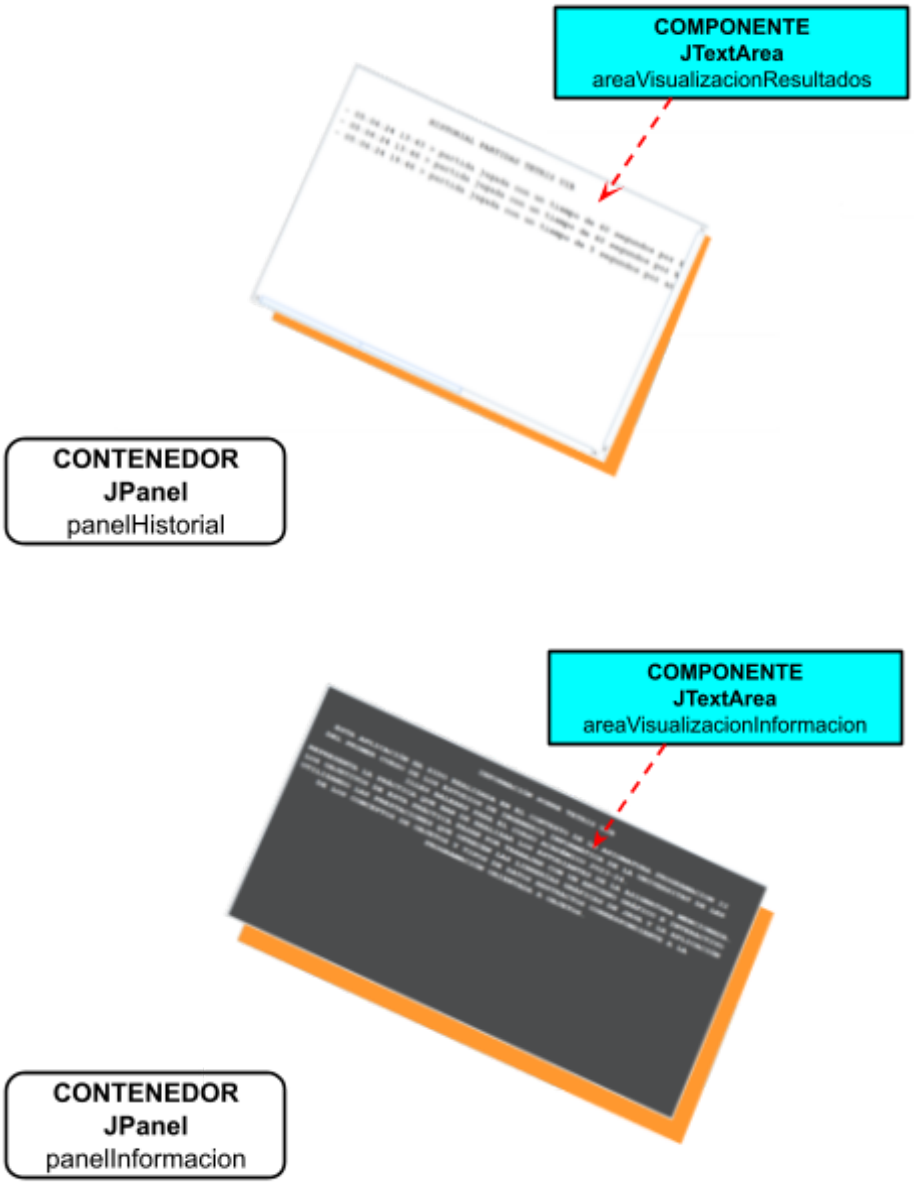
- Clase PanelVisualizaciones: que también extiende de JPanel se utiliza para mostrar diferentes visualizaciones en una interfaz de usuario. La clase PanelVisualizaciones tiene varios métodos para mostrar diferentes paneles de visualización:
    - El método panelPartida() crea un panel para el juego de Tetris. Configura la información de la partida, como el nombre del jugador y la puntuación, junto con los elementos del juego, como la figura y el tablero. También añade botones para acciones como generar una nueva forma y rotar la forma actual. Todos los componentes se organizan dentro del panel principal y se devuelve para su uso en la interfaz gráfica del juego.
    - 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.
    - El método panelPartida() crea un panel para el juego de Tetris. Configura la información de la partida, la interfaz del juego con la figura de Tetris y el tablero, y añade botones para acciones como generar una nueva forma y rotarla. Todos los componentes se organizan en un panel principal usando divisiones JSplitPane.
    - El método panelHistorial() crea un panel que muestra un historial de resultados. Se encarga de seleccionar el tipo de JTextArea que muestra los datos del archivo "resultados.dat". Si el nombre de usuario es null, muestra el historial general; si no es null, muestra el historial específico del usuario. En ambos casos, si el archivo no se encuentra o está vacío, se muestra el texto "No se ha encontrado". El contenido del historial se obtiene de un objeto Partida y se muestra en un JTextArea. Además, el panel utiliza un JScrollPane para permitir el desplazamiento vertical si el contenido del historial es largo.

4. Diseño y esquema de la interfaz gráfica






COMPONENTE
JLabel
imagen UIB
TETRIS
UIB
CONTENEDOR
JPanel
panelStandbye
COMPONENTE
JPanel
panelPartida
COMPONENTE
JPanel
panelInfoPartida
COMPONENTE
JButton
(nuevaForma,
rotarForma)
CONTENEDOR
JPanel
panelPartida
COMPONENTE
JProgressBar
barraTemporal


COMPONENTE
JTextArea
areaVisualizacionResultados
CONTENEDOR
JPanel
panelHistorial
COMPONENTE
JTextArea
areaVisualizacionInformacion
CONTENEDOR
JPanel
panelInformacion

5. Conclusión

La práctica ha sido bastante extensa y este año no nos ha costado tanto para nosotras, ya que ambas ya hemos programado bastante en Java en este último año, pero aún así nos ha obligado a dedicarle muchas horas. A medida que íbamos avanzando, nos encontrábamos con nuevos problemas que teníamos que resolver, lo cual nos permitió mejorar aún más nuestras habilidades para solucionarlos poco a poco.

A pesar de las dificultades, esta práctica nos ha ayudado mucho a condensar nuestros conocimientos en programación en Java. Ha sido un trabajo extenso que abarca varios temas vistos en clase, lo cual ha sido muy útil.

Durante esta práctica, aprendimos a manejar la clase "BarraProgresionTemporal" y a trabajar con la estructura de ficheros usando "RandomAccessFile". También trabajamos con "Tauler", "Casella" y "Pesa", así como con "LecturaDatos" junto con "GestorEventos" para manejar las respuestas del usuario y la división y colocación de imágenes. Al principio, todo esto nos resultó complicado la forma de organizarlo, pero una vez bien ordenado, logramos entenderlo y manejarlo bien. Desde el principio pensamos que era buena idea distinguir varias clases, porque ambas hemos aprendido a organizarnos y programar de esta forma, lo que también nos ayuda a detectar errores con más facilidad.

En resumen, este proyecto nos ha permitido seguir consolidando los conocimientos que sabíamos y aprender nuevos en programación, especialmente en la gestión de ventanas, paneles y ficheros. Sabemos que nuestro Tetris no es el mejor, pero creemos que le hemos dedicado esfuerzo y sabiduría. Consideramos que este tipo de trabajos finales son muy importantes para asimilar bien los conceptos enseñados durante el curso y nos ha sido de ayuda realizarlo.