TEC | Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Programación Orientada a Objetos

Número de grupo: 02

Programa II: “Futoshiki”

Estudiantes: Juan Pablo Cambronero y Ximena Molina

Fecha de entrega: 29/11/2024

Semestre II 2024

Profesor: William Mata Rodríguez

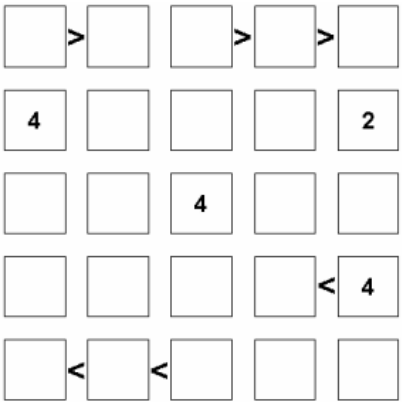
Contenido:

Enunciado del proyecto:

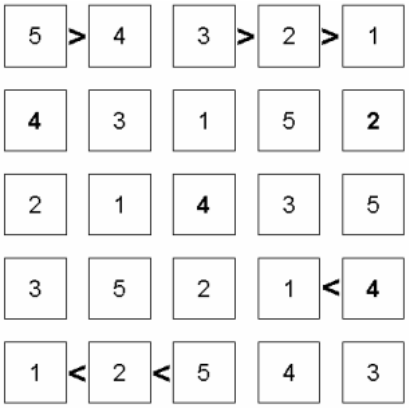
Futoshiki es un rompecabezas de lógica originario de Japón, desarrollado por Tamaki Seto en 2001, su nombre significa “desigualdad”. Hay varias aplicaciones que permiten jugarlo.

En este rompecabezas hay que llenar con dígitos las casillas de una cuadrícula de tal forma que cada dígito no se repita ni en la fila ni en la columna a que pertenece y los dígitos cumplan con restricciones de desigualdad: mayor que (>) o menor que (<), que pueden estar a nivel de filas o a nivel de columnas: 4 X 4, 5 X 5, 6 X 6, etc.

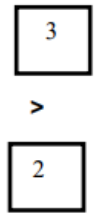
En esta especificación se va a usar una cuadrícula de tamaño 5 x 5, por tanto, los dígitos a colocar estarían entre 1 y 5. Cada juego tiene de forma predeterminada las restricciones de desigualdad y podría tener algunos dígitos constantes como se muestra en la figura 1:



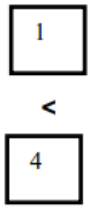
La solución al juego anterior sería:



Además de desigualdades a nivel de fila, el juego puede tener desigualdades a nivel de columna, por ejemplo:



Se lee: 3 > 2



Se lee: 1 < 4

Temas Investigados:

- Uso de archivos XML:

El proyecto **Futoshiki** utiliza un archivo en formato **XML** para almacenar las partidas de juego. Este enfoque facilita la organización y el acceso a datos estructurados y permite que las configuraciones y partidas se guarden de manera eficiente y legible tanto para humanos como para máquinas.

¿Qué es XML?

XML (Extensible Markup Language) es un lenguaje de marcado diseñado para almacenar y transportar datos. Su principal característica es la simplicidad y la capacidad de estructurar información de manera jerárquica a través de elementos y atributos. Por ejemplo:

<partidasFutoshiki>

<partida>

<nivel>Facil</nivel>

<cuadricula>5</cuadricula>

<desigualdades>

<des>maf,0,1</des>

<des>mec,2,3</des>

</desigualdades>

<constantes>

<cons>3,1,1</cons>

<cons>5,2,2</cons>

</constantes>

</partida>

</partidasFutoshiki>

En este proyecto, el XML se usa para definir:

1. **Nivel de dificultad**: Fácil, Intermedio o Difícil.
2. **Tamaño de la cuadrícula**: Dimensión NxN.
3. **Restricciones de desigualdad**: Entre casillas.
4. **Constantes**: Números predefinidos en el tablero.

Uso del XML en el Proyecto:

El proyecto tiene un método clave para leer las partidas almacenadas en XML:

```
//cargar archivo tipo xml que contiene todas las posibles combinaciones de partidas
public List cargarArchivoPartidas(int numNivel, int cuadricula){
```

Lectura y configuración:

- Utiliza el paquete javax.xml.parsers para parsear el archivo XML.
- Filtra las partidas por el nivel de dificultad (<nivel>) y el tamaño de la cuadrícula (<cuadricula>).

Estructura jerárquica:

- Cada <partida> tiene elementos <nivel>, <cuadricula>, <des> (desigualdades) y <cons> (constantes).
- Los elementos <des> y <cons> detallan las desigualdades y constantes para esa partida específica.

Construcción de objetos:

- Las desigualdades y constantes se almacenan en listas para ser procesadas posteriormente y desplegarse en la interfaz gráfica.

Beneficios del XML

1. **Portabilidad**: El archivo XML puede ser utilizado o modificado en cualquier sistema que soporte este formato.
2. **Legibilidad**: Es fácil de leer para humanos, lo que facilita el mantenimiento y la depuración.
3. **Escalabilidad**: La estructura puede adaptarse fácilmente para agregar nuevos niveles o configuraciones.

- Cómo se implementó todo lo relacionado con el reloj:

Implementación y Uso del Reloj en el Proyecto Futoshiki:

El manejo del **reloj** en el proyecto **Futoshiki** juega un papel esencial para realizar un seguimiento del tiempo durante la partida. El reloj está implementado como parte del **modelo** dentro del patrón **MVC**, y su lógica abarca tanto un cronómetro como un temporizador. Este componente garantiza la funcionalidad relacionada con medir el tiempo y permite la interacción con la interfaz gráfica.

Estructura del Reloj

El reloj está implementado en la clase Reloj dentro del paquete modeloFutoshiki. Esta clase contiene atributos y métodos para manejar las unidades de tiempo y diferenciar entre los dos tipos de funcionalidad: cronómetro y temporizador.

Atributos principales:

```
public class Reloj {
    private int tipo; // 0: sin reloj | 1: cronómetro | 2: temporizador
    private int horas;
    private int minutos;
    private int segundos;
```

Horas, Minutos y Segundos: Controlan el tiempo actual.

Tipo: Indica si el reloj se comporta como un cronómetro (cuenta hacia arriba) o como un temporizador (cuenta hacia abajo).

Métodos Clave

1. **Setters y Getters:**
    - Permiten actualizar y consultar las unidades de tiempo desde otras clases.
    - Ejemplo: setHoras, setMinutos, setSegundos.

2. **Incremento y Decremento:**
    - Métodos que ajustan los valores del tiempo según el tipo del reloj.
    - En un cronómetro, los segundos aumentan y, si alcanzan 60, los minutos incrementan, y así sucesivamente.
    - En un temporizador, el tiempo disminuye, y cuando llega a 00:00:00, el juego termina.

3. **Validaciones:**
    - Los métodos manejan los límites para evitar valores negativos en un temporizador o superar los máximos válidos (60 segundos, 60 minutos, etc.).

Uso del Reloj en el Proyecto

El reloj se utiliza principalmente en la clase PantallaJuegoControlador, dentro del controlador del juego. Su integración incluye:

1. **Iniciar y Actualizar el Reloj**:
    - Un temporizador (javax.swing.Timer) invoca métodos que actualizan el reloj cada segundo.
    - Dependiendo del tipo (cronómetro o temporizador), se elige la lógica adecuada.
2. **Vinculación con la Interfaz Gráfica**:
    - La interfaz gráfica (PantallaJuego2) muestra el estado actual del reloj en una tabla (JTable).
    - Cada segundo, los valores se actualizan para reflejar el tiempo actual del juego.
3. **Finalización del Tiempo**:
    - Si el temporizador llega a cero, se detiene el juego y se muestra un mensaje indicando que el tiempo ha terminado.
    - El controlador redirige al jugador al menú principal.

Flujo del Reloj en el Proyecto

1. **Inicialización**:
    - Cuando el jugador inicia una partida (botón "Jugar"), el reloj se configura según las preferencias de la partida.
    - Si se selecciona un temporizador, se establece un límite de tiempo.

2. **Actualización Periódica**:
    - Un temporizador (Timer) ejecuta una acción cada segundo:
        - Incrementa el tiempo para un cronómetro.
        - Decrementa el tiempo para un temporizador.
    - La tabla JTable muestra los valores actualizados en la pantalla.
3. **Persistencia**:
    - Cuando se guarda una partida, los valores del reloj (horas, minutos, segundos) también se almacenan en el archivo de guardado.
    - Al cargar una partida, el estado del reloj se recupera para que el jugador continúe desde el mismo punto.

Iniciar el Reloj

```
// Inicia el temporizador si el juego está configurado como temporizador
private void iniciarTemporizadorSiEsNecesario() {
```

Este método verifica si ya hay un temporizador corriendo y lo detiene.

Se inicializa un nuevo temporizador que actualiza el reloj cada segundo.

Actualizar el Cronómetro

```
private void actualizarCronometro() {
```

Incrementa los valores del reloj y actualiza la interfaz gráfica.

Actualizar el Temporizador

```
// Actualiza el temporizador en el JTable y detiene el temporizador cuando llega a cero
private void actualizarTemporizador() {
```

Resta valores del reloj y finaliza el juego cuando el tiempo llega a cero.

Ventajas de la Implementación

1. **Flexibilidad**: Al soportar tanto cronómetro como temporizador, se adapta a diferentes estilos de juego.
2. **Persistencia**: Al guardar y cargar partidas, el reloj retoma el estado exacto en que se dejó.
3. **Separación de Responsabilidades**:
    - La clase Reloj encapsula toda la lógica del tiempo.
    - El controlador gestiona su integración con la vista

- Abrir archivos pdfs:

Implementación de la Función para Abrir un PDF en el Proyecto:

Descripción General

En este proyecto, se implementó una funcionalidad para abrir el manual de usuario en formato PDF desde el menú principal. Esta funcionalidad está diseñada para proporcionar a los jugadores una guía clara sobre cómo interactuar con el sistema. Se utiliza el componente Desktop de Java, que permite ejecutar aplicaciones predeterminadas del sistema operativo para abrir documentos, como lectores de PDF.

La función se invoca cuando el usuario selecciona la opción de ayuda desde el menú principal, abriendo automáticamente el archivo programa2_futoshiki_manual_de_usuario.pdf . Este archivo debe estar ubicado en el directorio del proyecto o en una ruta específica que sea accesible para el programa.

Cómo Funciona la Apertura del PDF

El mecanismo principal para abrir el PDF utiliza la clase Desktop de Java, específicamente el método Desktop.open(File). Esta clase interactúa con el sistema operativo para abrir un archivo utilizando la aplicación predeterminada configurada para manejar ese tipo de archivo. El flujo de trabajo es el siguiente:

1. **Verificar la Existencia del Archivo**:
    - Antes de intentar abrir el PDF, se verifica si el archivo existe en la ruta esperada.
    - Si el archivo no se encuentra, se muestra un mensaje de error al usuario utilizando JOptionPane.
2. **Compatibilidad del Sistema**:
    - Se verifica si el sistema soporta la clase Desktop mediante Desktop.isDesktopSupported().
    - Si no es compatible, se muestra otro mensaje de error indicando que la funcionalidad no está disponible.
3. **Apertura del Archivo**:
    - Si el archivo existe y el sistema soporta Desktop, el archivo se abre con el lector de PDF predeterminado del sistema operativo.
4. **Manejo de Excepciones**:
    - Si ocurre un error durante la apertura del archivo, como un problema de permisos o un archivo corrupto, el sistema lo informa al usuario de manera adecuada.

La funcionalidad para abrir el PDF está implementada dentro de la clase MenuPrincipalControlador y se vincula al botón de ayuda (btnAyuda).

```
// Método para abrir el manual de usuario en formato PDF
private void abrirManualUsuario() {
```

Cómo se Usa en el Proyecto

1. **Ubicación del Código**:
    - El método abrirManualUsuario se encuentra en la clase MenuPrincipalControlador, que gestiona la interacción entre la vista (MenuPrincipal) y el modelo.
2. **Vinculación al Botón de Ayuda**:
    - En el constructor de MenuPrincipalControlador, el método se asocia al evento de clic del botón de ayuda (btnAyuda) utilizando un ActionListener.
3. **Interacción del Usuario**:
    - Cuando el usuario selecciona la opción de ayuda, el sistema intenta abrir el archivo manual_de_usuario.pdf.
    - Si el archivo está disponible y el sistema soporta la funcionalidad, el manual se abre automáticamente en el lector de PDF predeterminado.
4. **Manejo de Errores**:
    - Si el archivo no existe o si el sistema no puede abrirlo, el usuario recibe un mensaje claro sobre la naturaleza del problema.

Ventajas de Esta Implementación

- **Simplicidad**: Utiliza clases nativas de Java (Desktop y File) que son fáciles de implementar y mantener.
- **Compatibilidad**: Se apoya en las configuraciones del sistema operativo, garantizando que se use el lector de PDF predeterminado.
- **Robustez**: Incluye validaciones para manejar errores comunes, como la falta del archivo o la incompatibilidad del sistema.

- **Interactividad**: Mejora la experiencia del usuario al proporcionar acceso directo a un recurso importante (el manual de usuario) sin salir de la aplicación.

Solución:

- Modelo del sistema con un diagrama de casos de uso UML: incluye todos los casos de uso a un nivel superior.
- Modelo del sistema con un diagrama de clases UML que incluya:
- Aplicación del modelo MVC en este proyecto.

El patrón de diseño Modelo-Vista-Controlador (MVC) se utiliza en este proyecto para organizar y estructurar el código, separando las responsabilidades en tres componentes principales: el **Modelo**, la **Vista** y el **Controlador**. A continuación, se detalla qué partes del proyecto corresponden a cada componente y cómo interactúan entre sí:

1. Modelo

El **Modelo** es responsable de manejar la lógica del negocio, los datos del juego y las operaciones internas del sistema. En este proyecto, el modelo está representado por el paquete modeloFutoshiki e incluye las siguientes clases:

- **Archivo.java**: Gestiona la lectura y escritura de datos relacionados con las partidas (guardar y cargar juegos).
- **Correo.java**: Gestiona funcionalidades relacionadas con el envío de correos, si aplica en el contexto del juego.
- **Juego.java**: Contiene la lógica principal del juego, incluyendo el estado actual del tablero, el nivel, y el temporizador.
- **MatrizJuego.java**: Representa la estructura interna del tablero de Futoshiki, con las casillas, desigualdades y sus relaciones.

- **Movimiento.java**: Almacena los movimientos realizados por el jugador, permitiendo implementar funciones como deshacer y rehacer jugadas.
- **Reloj.java**: Maneja el cronómetro o temporizador del juego.
- **Jugador.java**: Representa al jugador, incluyendo su nombre y otros datos relevantes.

El modelo no interactúa directamente con la interfaz gráfica; su objetivo es realizar cálculos y almacenar datos que el controlador solicita.

2. Vista

La **Vista** es la capa responsable de mostrar la interfaz gráfica al usuario. En este proyecto, el paquete vistaFutoshiki contiene las clases que generan las ventanas y elementos visuales del juego:

- **MenuConfiguracion.java**: Proporciona una interfaz para configurar el juego antes de iniciarlo.
- **MenuPrincipal.java**: Es la ventana principal del juego, desde donde el usuario puede navegar a otras funcionalidades.
- **PantallaJuego2.java**: Es la interfaz principal del tablero de Futoshiki, donde se muestran las casillas, desigualdades, temporizador y botones de control.
- **PantallaOlvidoContraseña.java**: Interfaz para recuperar contraseñas o gestionar datos del usuario.
- **PantallaTop10.java**: Muestra una tabla con los mejores puntajes o tiempos de los jugadores.

Estas clases contienen los elementos visuales, como botones, etiquetas y tablas, pero no incluyen lógica de negocio ni procesamiento de datos.

3. Controlador

El **Controlador** es el intermediario entre la Vista y el Modelo. Se encarga de gestionar los eventos generados en la interfaz de usuario y solicitar al modelo las operaciones necesarias. En este proyecto, el paquete controladorFutoshiki contiene las siguientes clases controladoras:

- **MenuConfiguracionControlador.java**: Gestiona los eventos en la pantalla de configuración, como la selección del nivel o la configuración del tablero.
- **MenuPrincipalControlador.java**: Maneja las interacciones del usuario en el menú principal, como navegar a otras pantallas.
- **PantallaJuegoControlador.java**: Es el núcleo de la lógica del juego. Se encarga de:
    - Manejar los eventos en el tablero (movimientos del jugador, validaciones, temporizador).
    - Coordinar la interacción entre la Vista (PantallaJuego2.java) y el Modelo (Juego.java, MatrizJuego.java).
    - Implementar las funcionalidades de guardar, cargar, borrar y reiniciar partidas.
- **PantallaOlvidoContraseñaControlador.java**: Controla los eventos relacionados con la recuperación de contraseñas.
- **PantallaTop10Controlador.java**: Gestiona la lógica para mostrar los mejores puntajes o tiempos en la pantalla de "Top 10".

El controlador actúa como un puente entre el modelo y la vista, garantizando que la interfaz gráfica muestre datos consistentes y responda correctamente a las acciones del usuario.

Relación entre los componentes

- **Vista a Controlador**: Los eventos generados en la Vista (como un clic en un botón o la selección de una casilla) son manejados por los controladores correspondientes. Por ejemplo, al seleccionar una casilla en PantallaJuego2.java, el evento se envía a PantallaJuegoControlador.java para procesarlo.
- **Controlador a Modelo**: El controlador solicita al modelo realizar operaciones específicas, como validar un movimiento, actualizar el temporizador o guardar una partida. Por ejemplo, PantallaJuegoControlador.java interactúa con Juego.java y MatrizJuego.java para gestionar el estado del juego.
- **Modelo a Vista**: El modelo no interactúa directamente con la vista. En su lugar, el controlador actualiza la vista con los datos proporcionados por el modelo. Por ejemplo, el controlador actualiza las casillas del tablero en PantallaJuego2.java según los valores almacenados en MatrizJuego.java.

- Esquema de manejo de nombre de jugadores (en configuración el dato 6).

Esquema de manejo de jugadores:

En nuestro programa, cada jugador es identificado por un nombre único dentro del sistema. El nombre del jugador debe ser una cadena de caracteres (String) con una longitud que puede variar entre 0 y 30 caracteres. Esta restricción garantiza que los nombres sean lo suficientemente cortos para su almacenamiento eficiente, pero también lo suficientemente largos como para ofrecer una identificación significativa.

Además, cada jugador debe tener un control de acceso único para garantizar la seguridad de su cuenta. Este control de acceso se implementa mediante el uso de una contraseña. La contraseña es un dato confidencial que el jugador debe ingresar para acceder a su cuenta dentro del juego.

Con el fin de proporcionar una capa adicional de seguridad y comodidad al jugador, nuestro programa incluye una opción de recuperación de contraseña. En caso de olvido de la contraseña, el jugador podrá recuperarla de manera sencilla. Para ello, el sistema envía un correo electrónico al jugador con un **PIN de recuperación**, que le permite acceder a una interfaz para cambiar su contraseña a una nueva. Este mecanismo de recuperación asegura que el acceso a las cuentas de los jugadores sea seguro y que los jugadores puedan recuperar fácilmente su acceso en caso de olvido de sus credenciales.

Este enfoque de gestión de nombres y contraseñas asegura tanto la privacidad como la seguridad de cada jugador en el sistema, permitiendo una experiencia de usuario confiable y sin inconvenientes.

Basta con ingresar el código enviado a su correo y podrá modificar su contraseña.

Conclusiones del Trabajo:

- Problemas encontrados y soluciones a los mismos.

Problemas relacionados con el reloj:

Uno de los principales desafíos al desarrollar este juego fue la integración y el manejo del reloj. Inicialmente, se intentó utilizar un reloj interno del juego, pero resultó ser complicado adaptarlo adecuadamente a la lógica del programa. El sistema debía permitir dos modos de tiempo distintos: un cronómetro que contara el tiempo desde el inicio del juego, y un temporizador que contara hacia atrás desde un tiempo predeterminado. Sin embargo, no estaba claro cuál era la mejor opción para implementarlo de manera eficiente y sin generar complicaciones adicionales.

El proceso de diseño requería que ambos tipos de reloj interactuaran de forma fluida, lo cual presentaba varias dificultades, entre ellas, mantener ambos relojes sincronizados y ajustados a la lógica de los diferentes niveles del juego sin afectar el rendimiento del programa.

Finalmente, después de investigar diversas soluciones, encontramos que los Java Timers eran una herramienta adecuada para manejar este tipo de tareas. Los Java Timers permiten ejecutar tareas a intervalos regulares o después de un cierto retraso, lo que facilita la implementación de un cronómetro o temporizador con la precisión necesaria.

Para mejorar la visualización y control de los tiempos, decidimos integrarlo en un JTable, lo que nos permitió gestionar de manera eficiente los valores de los relojes y visualizar el tiempo restante de cada jugador o nivel. Este enfoque nos dio la flexibilidad necesaria para cambiar entre cronómetro y temporizador según el estado del juego y los requerimientos específicos de cada nivel.

El uso de los Java Timers y la integración con el JTable resolvió los problemas de sincronización y visualización, permitiendo que los jugadores pudieran ver

claramente el tiempo transcurrido o restante, con la lógica deseada para cada modo de juego.

Generar botones de acuerdo con la configuración:

Otro desafío importante durante el desarrollo del juego fue la configuración del tamaño del tablero o matriz de juego. En el programa, el tamaño del tablero debía ser dinámico, es decir, adaptarse según la selección del jugador. Esto implicaba que el número de casillas (botones) en la pantalla debía variar dependiendo del tamaño elegido, lo cual afectaba no solo la cantidad de botones, sino también su posición y tamaño para que todo encajara correctamente en la interfaz de usuario.

Inicialmente, se intentó utilizar la funcionalidad de drag and drop de Swing para manejar los componentes de manera visual y ajustar la interfaz según el tamaño del tablero. Sin embargo, el sistema no permitió que los botones se ajustaran correctamente a la pantalla, ya que la cantidad de botones variaba según la selección del jugador y esto generaba un problema de distribución. Además, Swing no tenía una forma sencilla de manejar la reubicación y el redimensionamiento de los botones de manera automática para que se ajustaran adecuadamente al tamaño de la matriz.

La solución a este problema fue crear un array en el cual se almacenaban todos los botones que iban a formar parte de la matriz, con la cantidad exacta según la configuración seleccionada por el jugador. A través de este array, se podía acceder a cada botón de manera organizada y dinámica. Posteriormente, se utilizó un ciclo para ubicar y ajustar las posiciones de los botones dentro de la ventana, dependiendo de las dimensiones del array. Este ciclo permitió distribuir los botones en filas y columnas, ajustando sus posiciones y tamaños de manera correcta en la interfaz.

Toda la lógica del acomodo de los botones estaba basada en ese array, lo que hizo posible que el tablero se adaptara al tamaño seleccionado sin problemas. Aunque esta solución fue laboriosa y requirió un esfuerzo considerable, al final resultó ser eficaz, ya que permitió un control total sobre la disposición de los botones en la

interfaz, garantizando que el juego funcionara correctamente independientemente del tamaño de la matriz.

Gracias a esta solución, se logró una experiencia de juego fluida y bien organizada, sin problemas de visualización ni distribución de los botones, lo que cumplió con las expectativas del diseño del juego.

- Aprendizajes obtenidos.

Gestión de Datos de Usuario y Seguridad

- **Autenticación y manejo de contraseñas:** La implementación de un sistema de contraseñas seguras, que permite a los jugadores crear una cuenta con un nombre único y acceder a su perfil mediante una contraseña, ha sido un ejercicio importante en la gestión de datos sensibles. Se ha aprendido a implementar mecanismos de control de acceso y a asegurar la integridad de la información personal.
- **Recuperación de contraseñas:** El sistema de recuperación de contraseñas mediante un PIN enviado al correo electrónico del jugador ha enseñado cómo integrar procesos de recuperación de acceso en aplicaciones. Este tipo de funcionalidad es esencial para mantener la seguridad y mejorar la experiencia del usuario al ofrecer soluciones cuando se pierden las credenciales.

Implementación de Niveles y Progresión en Juegos

- **Diseño de niveles de dificultad:** Implementar un sistema multinivel, donde el jugador avanza entre niveles de dificultad (fácil, intermedio, difícil), ha permitido aprender a manejar lógicas de progresión en juegos. Esto implica no solo cambiar la dificultad del juego, sino también almacenar y gestionar el estado del juego a través de diferentes etapas.
- **Transición de niveles y guardado de progreso:** La transición entre niveles y el cálculo de tiempo acumulado para cada uno enseña cómo estructurar el flujo

de un juego de manera que los cambios entre etapas se realicen de manera fluida y sin problemas de integridad de datos.

Manejo del Tiempo y Temporizadores

- **Uso de objetos de tiempo con java.time:** La implementación del cálculo de tiempo usando la API de java.time nos ha proporcionado conocimiento en el manejo de tiempos en un entorno de programación. Esto incluye la gestión de cronómetros, temporizadores y la utilización de la hora del sistema para calcular el tiempo de juego sin depender de un reloj interno del juego.
- **Cálculo de tiempo acumulado:** Aprender a calcular el tiempo transcurrido y acumularlo a través de diferentes niveles es fundamental tanto para la lógica del juego como para la interacción con el jugador. Este aprendizaje es valioso no solo para juegos, sino también para cualquier aplicación que requiera registrar el paso del tiempo entre eventos.

Lista de Revisión del Proyecto:

Concepto	Puntos originales	Avance 100/%/0	Puntos obtenidos	Análisis de resultados
Opción Jugar (despliegue del juego según configuración)	12	100		
Botón Iniciar Juego	10	100		
Crear Top 10	8	100		
Botón Borrar Jugada	5	100		
Botón Rehacer Jugada	5	100		
Botón Borrar Juego	3	100		
Botón Terminar Juego	2	100		
Botón Guardar Juego	5	100		
Botón Cargar Juego (incluye el despliegue del mismo)	9	100		
Opción Top 10	5	100		
Opción Configurar Datos del 1 al 5 Dato 6 (jugador)	 5 6	100		
Ayuda (manual de usuario)	5	100		
Cronómetro tiempo real	5	100		
Temporizador tiempo real	5	100		
Implementación juego multinivel	10	100		
TOTAL	**100**	100		
Partes desarrolladas adicionalmente				Filtro en top 10

Bibliografía:

Bibliografía relacionada con el manejo de archivos PDF:

- Oracle. (n.d.). *Desktop (Java Platform SE 7)*. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/awt/Desktop.html
- Oracle. (n.d.). *File (Java Platform SE 7)*. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/io/File.html

Bibliografía relacionada con el modelo MVC y organización de proyectos Java:

- Oracle. (n.d.). *How to Use Models (The Java™ Tutorials > Creating a GUI With Swing > Using Other Swing Features)*. Retrieved from https://docs.oracle.com/javase/tutorial/uiswing/models/index.html

Bibliografía sobre el manejo de archivos y el uso de FileReader y BufferedReader:

- Oracle. (n.d.). *FileReader (Java Platform SE 7)*. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html
- Oracle. (n.d.). *BufferedReader (Java Platform SE 7)*. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html

Bibliografía relacionada con XML en Java:

- Oracle. (n.d.). *Parsing an XML File Using SAX* (Java Platform SE 8). Retrieved from https://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html
- Oracle. (n.d.). *DocumentBuilder (Java Platform SE 7)*. Retrieved from https://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/DocumentBuilder.html

Bibliografía relacionada con la gestión de relojes y temporizadores:

- Oracle. (n.d.). *Timer (Java Platform SE 7)*. Retrieved from https://docs.oracle.com/javase/7/docs/api/javax/swing/Timer.html