Informe del Proyecto: Arcade de Puzzles Lógicos

Descripción de la arquitectura implementada

En mi proyecto he desarrollado una aplicación de escritorio que implementa tres juegos lógicos clásicos: el problema de las N reinas, el recorrido del caballo en el tablero de ajedrez y las Torres de Hanói. He decidido utilizar JavaFX (en lugar de Swing) para crear una interfaz más moderna y atractiva, manteniendo la estructura solicitada.

La arquitectura que he implementado sigue el patrón Modelo-Vista-Controlador (MVC), lo que me ha permitido separar claramente las responsabilidades:

- **Modelo**: Contiene la lógica de juego y las entidades para la persistencia. Cada juego tiene su propia clase que extiende de una clase base común Juego.
- **Vista**: Implementada mediante archivos FXML de JavaFX y hojas de estilo CSS.
- **Controlador**: Gestiona la interacción entre la vista y el modelo, respondiendo a las acciones del usuario.

La aplicación tiene un menú principal desde el cual se puede acceder a cualquiera de los tres juegos. Al seleccionar un juego, se abre una nueva ventana con la interfaz específica de ese juego.

Para la persistencia de datos he utilizado Hibernate con una base de datos H2 embebida, que almacena los resultados de cada partida (parámetros utilizados, si se resolvió correctamente y el número de movimientos).

Los módulos de cada juego se integran en la aplicación a través de los patrones de diseño implementados, principalmente Factory Method y Facade, que permiten crear y gestionar cada juego de manera uniforme a pesar de sus diferencias.

Patrones de diseño utilizados

Patrones Creacionales

1. Singleton

- **Implementación**: HibernateConfig.java
- **Clases involucradas**: Solo la propia clase HibernateConfig
- **Justificación**: Decidí usar este patrón para garantizar que exista una única instancia de la sesión de Hibernate en toda la aplicación. Esto evita problemas de conexiones múltiples a la base de datos y optimiza el uso de recursos. La configuración de Hibernate es un caso típico donde Singleton es apropiado, ya que necesitamos un punto de acceso global pero controlado.

2. Factory Method

- **Implementación**: JuegoFactory.java (interfaz) y JuegoFactoryImpl.java (implementación)
- **Clases involucradas**: La interfaz Factory, su implementación y las clases de juego (NReinas, RecorridoCaballo, TorresHanoi)

- **Justificación**: Utilicé este patrón para encapsular la creación de los diferentes juegos. Me pareció ideal porque permite cambiar el tipo de juego a crear sin modificar el código que lo utiliza. El cliente simplemente solicita un juego por su tipo (string) y la fábrica se encarga de instanciar la clase concreta apropiada, lo que hace el código más mantenible y extensible.

Patrones Estructurales

1. Facade

- **Implementación**: JuegoFacade.java (interfaz) y sus implementaciones específicas para cada juego
- **Clases involucradas**: JuegoFacade, NReinasFacade, CaballoFacade, HanoiFacade
- **Justificación**: Implementé este patrón para simplificar la interacción entre la interfaz gráfica y la lógica interna de cada juego. Las fachadas proporcionan métodos de alto nivel (como inicializar(), resolver()) que ocultan la complejidad de las operaciones internas de cada juego. Esto facilita el trabajo de los controladores, que no necesitan conocer los detalles de implementación de cada juego.

2. Adapter

- **Implementación**: TableroAdapter.java (interfaz) y adaptadores específicos para cada juego
- **Clases involucradas**: TableroAdapter, TableroNReinasAdapter, TableroCaballoAdapter, TorresHanoiAdapter
- **Justificación**: Decidí usar este patrón para convertir la representación interna de cada juego (matrices, listas) en componentes visuales de JavaFX. El adaptador actúa como un puente entre el modelo y la vista, permitiendo que los componentes de la UI muestren el estado del juego sin tener que modificar el modelo. Esto separa claramente las responsabilidades y hace que el código sea más limpio.

Instrucciones de ejecución y uso

Requisitos previos

- Java 11 o superior
- Maven (opcional, si se quiere compilar desde fuente)

Compilación y ejecución

1. Clonar o descargar el repositorio
2. Abrir el proyecto en IntelliJ IDEA:
    - File > New > Project from Existing Sources
    - Seleccionar el archivo pom.xml

    - Seguir las instrucciones del asistente
3. Ejecutar la clase ArcadeApp.java

Alternativamente, si prefieres usar Maven desde la línea de comandos:

mvn clean compile

mvn javafx:run

Uso de la aplicación

Menú Principal

Al iniciar la aplicación, verás el menú principal con tres botones correspondientes a los tres juegos disponibles.

Problema de las N Reinas

1. Selecciona el número de reinas (N) usando el slider
2. Haz clic en "Resolver" para encontrar una solución
3. El tablero mostrará la disposición de las reinas donde ninguna amenaza a otra
4. Un mensaje indicará si se encontró una solución y el número de iteraciones

Recorrido del Caballo

1. Selecciona el tamaño del tablero con el slider
2. Establece la posición inicial (fila y columna) usando los controles numéricos
3. Haz clic en "Encontrar Recorrido" para resolver el puzzle
4. El tablero mostrará el recorrido completo numerado
5. Un mensaje indicará si se encontró un recorrido completo

Torres de Hanoi

1. Selecciona el número de discos usando el slider
2. Puedes resolver el puzzle de dos formas:
    - Automáticamente: haciendo clic en "Resolver Automáticamente"
    - Manualmente: usando los botones para mover discos entre torres (A→B, A→C, etc.)
3. La visualización mostrará el estado actual de las torres
4. Un contador indicará el número de movimientos realizados

En todos los juegos, los resultados se guardan automáticamente en la base de datos para poder consultar estadísticas sobre partidas anteriores.

Comentarios adicionales y dificultades encontradas

Durante el desarrollo me enfrenté a algunos desafíos interesantes:

1. **Implementación de los algoritmos de backtracking**: Tanto para N Reinas como para el Recorrido del Caballo, tuve que tener cuidado con la lógica recursiva para asegurar que funcionara correctamente y eficientemente.
2. **Migración de Swing a JavaFX**: Aunque en los requisitos se pedía usar Swing, decidí implementar la interfaz con JavaFX por su mayor flexibilidad y mejor apariencia visual. Esto supuso una curva de aprendizaje adicional pero creo que el resultado final valió la pena.
3. **Visualización de Torres de Hanoi**: Representar gráficamente este puzzle fue particularmente desafiante, ya que requiere mostrar discos de diferentes tamaños y permitir su movimiento entre torres.

Como posibles mejoras futuras, considero:

- Añadir un sistema de puntuación basado en el tiempo y número de movimientos
- Implementar animaciones para hacer más visual el recorrido del caballo o los movimientos en Torres de Hanoi
- Permitir guardar y cargar partidas en curso
- Añadir niveles de dificultad adicionales