Informe final: Fallos y posibles mejoras

A medida que el proyecto fue creciendo, la complejidad del código aumentó debido a la cantidad de reglas del ajedrez que se deben implementar y a la interacción entre las diferentes clases.

A pesar de las dificultades experimentadas debido a la complejidad creciente del proyecto y mi inexperiencia, el código tiene fallos de estructuración, bajo mi punto de vista. Es normal encontrarse con desafíos en proyectos de desarrollo de software, especialmente en proyectos de juegos con reglas complejas como el ajedrez. Lo más importante para mí ha sido aprender de las dificultades y seguir mejorando el código y mis habilidades como desarrollador.

Tras mi análisis final del proyecto llegue a las siguientes conclusiones:

1. **Complejidad del método moverPieza:** Este método contiene una cantidad significativa de lógica que podría dividirse en funciones más pequeñas y modulares para mejorar la legibilidad y facilitar el mantenimiento. Dividir este método en submétodos más pequeños y específicos, cada uno encargado de una tarea particular, hará que sea más fácil de entender y depurar.
2. **Clase muy extensa:** El tamaño del código en esta clase puede ser un desafío para su mantenimiento. Dividir la funcionalidad en clases más pequeñas y cohesivas, siguiendo principios de diseño como el principio de responsabilidad única, puede facilitar la comprensión y el mantenimiento del código.
3. **Manejo de excepciones:** En algunos lugares, las excepciones se manejan imprimiendo el rastreo de la pila (**e.printStackTrace()**). En una aplicación real, manejar las excepciones de manera más informativa y contextual (como mostrar un mensaje de error al usuario) sería más útil para el usuario final y facilitaría la depuración.
4. **Métodos largos:** Algunos métodos, como **esMovimientoValido**, tienen una lógica compleja. Dividir estos métodos en partes más pequeñas y manejables podría haber mejorado la claridad y la facilidad de prueba.
5. **Duplicación de lógica:** Hay fragmentos de código similares que podrían consolidarse en métodos comunes o clases auxiliares para evitar la duplicación y hacer el código más mantenible.
6. **Interfaz de usuario y lógica entrelazadas:** La lógica del juego y la interfaz gráfica están muy entrelazadas en esta implementación. A veces, separar la lógica del juego de la interfaz de usuario puede mejorar la flexibilidad y facilitar la portabilidad a diferentes plataformas o interfaces.

* **Refactorización:** Dividir los métodos largos en fragmentos más pequeños y manejables y separar la lógica del juego de la interfaz gráfica puede facilitar la comprensión y el mantenimiento.