Instituto Superior de Engenharia de Coimbra

Programação Avançada

Trabalho Prático

Jogo de Xadrez

Licenciatura em Engenharia Informática

2024 / 2025

Nuno Tomás Paiva

a2023137363@isec.pt

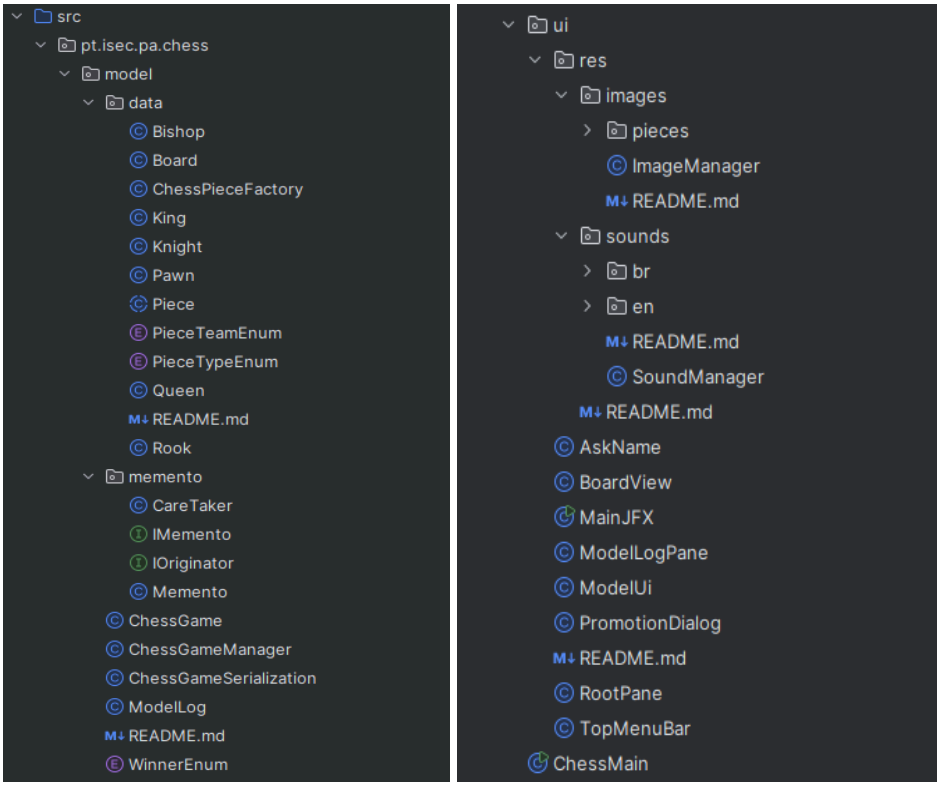
Rui Martins dos Santos

a2023145822@isec.pt

1. Introdução

Este trabalho foi desenvolvido em Java no âmbito da unidade curricular de Programação Avançada. O desenvolvimento deverá tirar partido do paradigma da Programação Orientada aos Objetos, da tecnologia JavaFX e de padrões arquiteturais e funcionais aplicados ao desenvolvimento de software.

O nosso trabalho está organizado da seguinte maneira:



2. Descrição acerca das decisões tomadas

Sendo este trabalho orientado por stage's, sinto que não houveram muitas decisões tomadas diretamente por nós. Mas das que tomámos (entre outras) foram:

- Para guardar as peças do tabuleiro, optámos por utilizar uma **List** em vez de uma estrutura como Map. Esta escolha foi motivada pela sua maior simplicidade de implementação, facilidade na iteração sobre todas as peças (por exemplo, para desenhar ou validar jogadas), e também por ter uma sobrecarga de memória inferior em comparação com o map. Como as operações mais frequentes envolvem percorrer todas as peças ou filtrar por equipa ou tipo, a List revelou-se suficiente e eficaz para as necessidades do jogo.
- Para implementar a funcionalidade de **undo/redo**, foi inicialmente adotado o **padrão Command**, permitindo encapsular cada jogada como um objeto com as operações de execute e undo. No entanto, com o avanço da implementação e a inclusão de regras especiais como en passant, roque (castling) e promoção de peões, verificámos que o **padrão Command** se tornava complexo e difícil de manter, pois exigia guardar múltiplas informações específicas por jogada. Por isso, decidimos substituir o **Command** pelo **padrão Memento**, que se revelou mais adequado para capturar o estado completo do jogo de forma simples e confiável antes de cada jogada. Esta abordagem facilitou a gestão de estados e tornou o mecanismo de **undo e redo** mais robusto e menos propenso a erros.

- Para a **interface gráfica**, decidimos fazer uma **separação** entre os vários componentes através de classes distintas, o que torna o trabalho mais organizado, modular e fácil de manter. Por exemplo, a **RootPane** centraliza a estrutura principal da janela, enquanto componentes como **AskName**, **TopMenuBar** e **BoardView** encapsulam funcionalidades específicas da interface, como a introdução dos nomes dos jogadores, o menu de opções e a visualização do tabuleiro, respetivamente. Esta abordagem promove uma melhor reutilização de código, facilita a leitura e compreensão da lógica da interface e permite uma maior flexibilidade para futuras alterações ou expansões, como a introdução de novos menus ou animações.

- Também implementámos uma classe **ModelLog** que regista **ações relevantes no jogo**, como movimentos inválidos, check, promoção, etc…, útil para debugging e para o histórico.

3. Diagramas de padrões de programação

Para simplificar o processo de criação das peças de xadrez, optámos por implementar dois métodos de fábrica distintos, seguindo o padrão **Factory Method** (**ChessPieceFactory**):

- **Fábrica por Tipo de Peça:** desenvolvemos um método que gera peças com base no seu tipo (definido através de uma enumeração), recebendo como parâmetros o tipo de peça (como rei, rainha, bispo, etc), a equipa correspondente (brancas ou pretas) e a posição no tabuleiro. Esta solução permite-nos criar peças de forma dinâmica durante a inicialização do jogo ou em situações específicas como a promoção de peões, mantendo um código limpo e organizado.
- **Fábrica por Representação Textual:** implementámos um segundo método que interpreta uma representação textual das peças (no formato "Rb2") para criar os objetos correspondentes, em que a primeira letra define o tipo de peça e a sua equipa de cores, preto ou branco, através de ser maisúcula ou não. Este método analisa a string recebida, extrai a informação relevante e cria a peça. Esta abordagem revelou-se particularmente útil para importar jogos parciais, simplificando significativamente o processo de serialização.

A classe **ChessGame** foi concebida para funcionar como o ponto central de interação com o sistema de xadrez, seguindo rigorosamente o **padrão Facade**. Esta abordagem permite gerir toda a complexidade interna do jogo enquanto oferece uma interface simples e segura aos componentes externos. Para tal, não pode retornar ou aceitar instâncias de objetos internos, e então deve ser a favor de retornar dados e receber parâmetros de tipos primitivos ou imutáveis.

Para parte da interface gráfica, seguimos a arquitetura **MVC@PA (**Model-View-Controller com Presenter Adapter**)**, e então, para controlar o acesso e proteger a classe **ChessGame** criamos outra classe que segue rigorosamente o **padrão Facade**, já descrito em cima, a classe **ChessGameManager** que fornece metódos para aceder aos métodos públicos da **ChessGame**, também lida com as operações de open, save, import export.

Para implementar a funcionalidade de **undo/redo**, foi adotado **padrão Memento**, que se revelou adequado para capturar o estado completo do jogo de forma simples e confiável antes de cada jogada. Esta abordagem facilitou a gestão de estados e tornou o mecanismo de **undo e redo** mais robusto e menos propenso a erros.

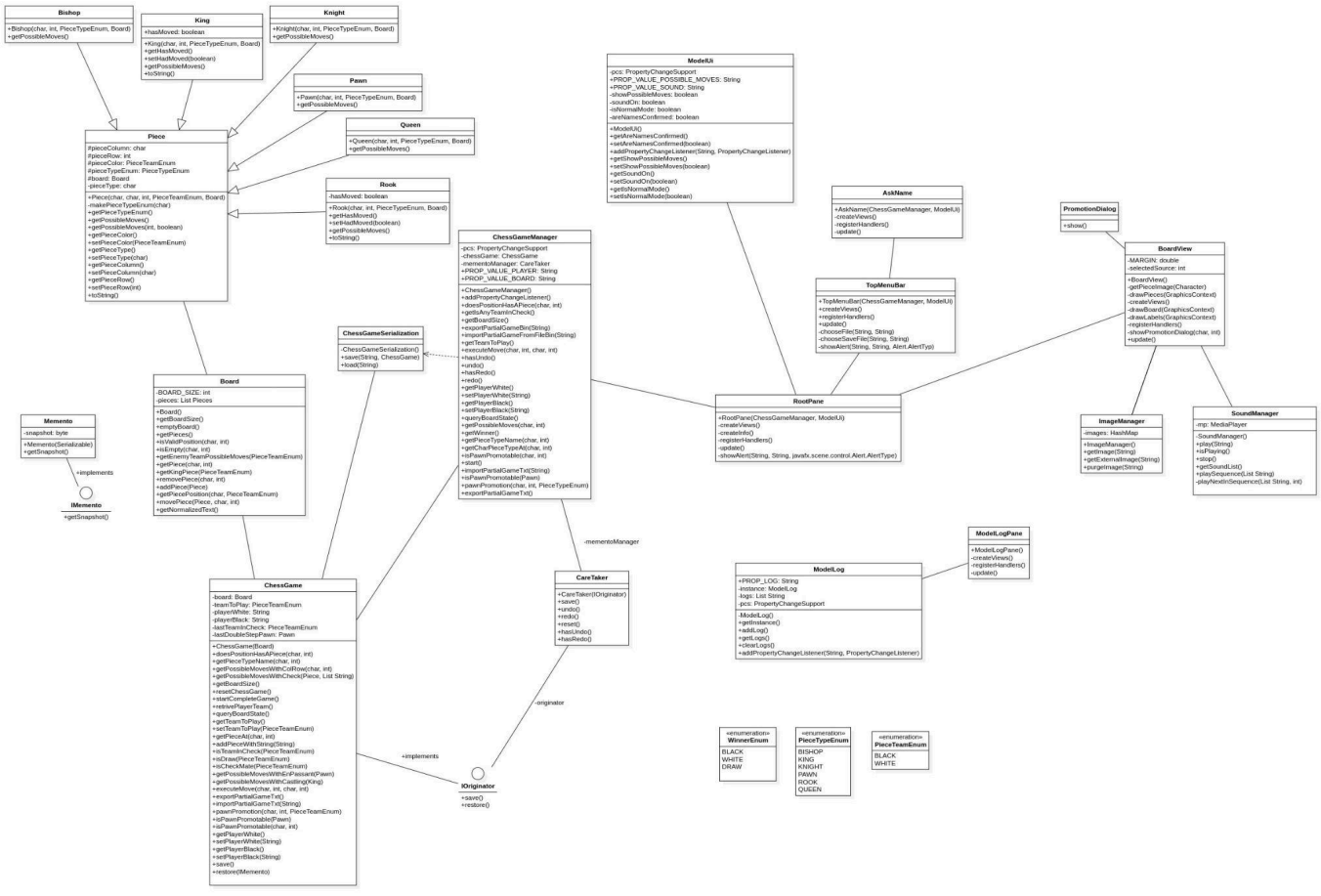
Para reportar os logs foi criada uma classe **ModelLog** que segue o **padrão Singleton**.

4. Tabela com descrição sucinta das classes utilizadas

Piece	classe que representa uma peça de xadrez;
PieceTypeEnum	enumeração relativamente ao tipo de peça (king, pawm, ...)
PieceTeamEnum	enumeração que representa a equipa preta ou branca
Board	classe que representa o tabuleiro e gere as peças
ChessGame	classe que gere as regras do jogo xadrez
IMemento	interface para objetos que armazenam estado
IOriginator	interface para classes que podem salvar/restaurar estado
Memento	classe que guarda snapshots do jogo
Caretalker	classe que gere a stack de estados para undo/redo
ChessGameManager	classe facade entre a UI e ChessGame
ModelLog	classe que gere a lista de logs do jogo
ChessGameSerialization	classe responsável por salvar/carregar o jogo em formatos binário
BoardView	classe que representa graficamente o tabuleiro e peças
ModelLogPane	classe que representa graficamente uma lista de logs
PromotionDialog	classe que representa o diálogo para selecionar uma peça quando um peão é promovido
RootPane	classe que representa graficamente um layout principal que organiza o resto

TopMenuBar	classe que representa graficamente o menu superior.
AskName	classe que representa graficamente o diálogo para inserir os nomes dos jogadores
ModelUI	classe que contém a lógica entre o modelo e a UI.
ImageManager	classe que carrega em mete em cache as imagens das peças
SoundManager	classe que controla os sons do jogo
WinnerEnum	enumeração muito parecida com a PieceTeamEnum, mas inclui também a hipótese de empate

5. Descrição sucinta do relacionamento entre as classes.



6. Funcionalidades implementadas.

Todas as funcionalidades do trabalho foram implementadas à exceção da funcionalidade extra (opcional) - o Board editor.