



DEPARTAMENTO DE ENGENHARIA ELETRÓNICA E DE  
TELECOMUNICAÇÕES E COMPUTADORES

Licenciatura em Engenharia Informática e Multimédia

---

# Trabalho Prático Final

## Modelação e Programação

---

*Realizado por:*

52483 - Bernardo Aguiar

*Docente:*

Pedro Fazendas

01/06/2025

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Desenvolvimento</b>	<b>1</b>
2.1	Classe, Atributo, Método e outros objectos . . . . .	2
2.1.1	package chess.core.board.pieces . . . . .	2
2.1.2	package chess.core.board . . . . .	2
2.1.3	package chess.core . . . . .	3
2.1.4	package chess.userinterface . . . . .	3
2.2	Arquitetura . . . . .	4
2.2.1	Modelo MVC . . . . .	4
2.2.2	Estratégias de programação orientada por objetos . . . . .	4
2.3	Modelos UML . . . . .	5
2.3.1	Diagrama de classes . . . . .	5
2.3.2	Diagramas de sequência . . . . .	6
2.3.3	Padrões usados . . . . .	7
2.3.4	Estrutura do projeto . . . . .	8
2.3.5	Exportação do tabuleiro para XML e o uso de XPATH . . . . .	8
2.3.6	Exportação do tabuleiro usando <i>Serialization</i> . . . . .	9
2.3.7	Utilização do Log4J . . . . .	10
2.3.8	Utilização do JUnit . . . . .	10
2.3.9	Gestão de dependências . . . . .	11
2.3.10	Considerações sobre performance . . . . .	11
2.4	Execução . . . . .	12

## Lista de Figuras

1	Diagrama de classes. . . . .	5
2	Diagrama de sequência da inicialização do tabuleiro. . . . .	6
3	Diagrama de sequência simplificado da jogada. . . . .	6

# 1 Introdução

Neste trabalho implementei um jogo de xadrez em que para além dos movimentos comuns das peças, implementei ainda os movimentos especiais do peão de "*En passant*", a promoção do peão e o movimento de roque - ou *castling* em inglês - da torre e do rei.

No que diz respeito a estrutura geral, segui uma arquitetura MVC. Em que como irei detalhar neste relatório, separei as classes de domínio como as diferentes peças, da classe que implementa todas as regras - lógica - da classe que controla a dinâmica do jogo como a jogada à vez por cada um dos jogadores.

Sempre que possível fiz uso de padrões, como o *Singleton*, *Factory* e ainda a adição do padrão *Facade*.

Nas minhas classes optei por usar *overloading* de algumas classes publicas para criar flexibilidade, clareza e consistência, evitando ambiguidades e garantindo que as versões sobrecarregadas - *overloaded* - implementem a mesma lógica com argumentos diferentes. Mesmo que alguns destes métodos não estejam a ser usados, simulam uma boa pratica em projetos mais complexo e que poderiam ser usados como livrarias. No decurso do desenvolvimento, foram amplamente utilizadas classes abstratas e métodos abstratos para otimizar a estruturação do trabalho, garantindo maior coesão e organização. Inicialmente, ponderou-se a utilização de interfaces nas fases primárias do projeto. Contudo, ao longo da evolução, a sua relevância prática diminuiu, o que levou à decisão de as descontinuar na implementação final. O projeto integra ainda funcionalidades de serialização, permitindo a persistência dos dados do programa para acesso e manipulação posteriores. Adicionalmente, foi implementada a criação de um ficheiro *.xml* para facilitar a visualização e interpretação da informação.

Por fim, criei este relatório em  $\text{\LaTeX}$  fazendo uso do [Overleaf](#), usando uma *template* que me pareceu adequada para esta finalidade, já com muitos exemplos de formatação, colocação de imagens e indexes variados.

## 2 Desenvolvimento

Para o desenvolvimento usei como referencia principal os conceitos aprendidos nas aulas e mencionados nos slides da [1]disciplina.

## 2.1 Classe, Atributo, Método e outros objectos

### 2.1.1 package chess.core.board.pieces

Este pacote agrupa classes relacionadas às peças do jogo. Estas são classes essencialmente de domínio ou entidade. Usando uma peça genérica da classe Piece, e com as restantes peças são herança de uma peça genérica, as classes das peças concretas herdam - estendem - a classe Piece.

- **Classe: Piece**

*Descrição:* classe abstrata que representa um peça genérica. E define atributos comuns como a cor e se já foi movida, com os respetivos *setters* e *getters*. Assim como o *equals* e o *hashCode*.

O método *toString* está definido como abstract, para que tenha de ser implementado como cada peça que herdará esta classe, a respetiva representação usando o carácter unicode.

- **Classe: Bishop**

*Descrição:* classe que representa a peça bispo.

- **Classe: King**

*Descrição:* classe que representa a peça rei.

- **Classe: Knight**

*Descrição:* classe que representa a peça cavalo.

- **Classe: Pawn**

*Descrição:* classe que representa a peça peão.

- **Classe: Queen**

*Descrição:* classe que representa a peça rainha.

- **Classe: Rook**

*Descrição:* classe que representa a peça torre.

### 2.1.2 package chess.core.board

Este pacote agrupa classes relacionadas o tabuleiro. Também contém algumas classes de domínio associadas ao tabuleiro, como "movimento", "quadrado", "posição".

- **Classe: Board**

*Descrição:* representação do tabuleiro usando um array de duas dimensões - array de arrays - de "quadrados".

Usando uma *LinkedList* guarda-se um histórico dos movimento efetuados. Esta classe permite colocar e obter uma peças num dada posição, reiniciar o tabuleiro, obter o tabuleiro, e realizar movimentos simples das respetivas peças, assim como movimentos especiais.

- **Classe: Square**

*Descrição:* classe que representa um quadrado do respetivo tabuleiro.

Esta classe permite colocar e obter uma peça, e verificar se o quadrado está vazio.

- **Classe: Position**

*Descrição:* classe que representa uma posição no tabuleiro - linha e coluna. A linha e coluna pode ser referenciada tanto usando inteiros de 1 a 8, ou usando uma referencia mais usual como A-H para colunas e inteiro 1-8 para linhas. Isto fazendo um *overloading* do construtor.

- **Classe: Move**

*Descrição:* classe que representa o movimento de uma peça de uma posição inicial, para uma posição final.

- **Classe: PieceFactory**

*Descrição:* classe *static* para criação de um dado tipo de peça.

- **Classe: RuleMaster**

*Descrição:* classe que representa todas as regras do xadrez.

Esta é uma das classes principais, que validada um dado movimento, dentro do tabuleiro, e que segue as regras dos movimentos de cada peça, se o jogo está terminado - verificando a existência no tabuleiro dos dois reis -, e quem ganhou caso já tenha terminado.

Esta classe representa toda a lógica do jogo.

- **Enum: Type**

*Descrição:* Enum com os tipos de peças.

### 2.1.3 package chess.core

Este pacote contem a classe expostas que representam o jogo do xadrez.

- **Classe: GameManager**

*Descrição:* esta classe encapsula todas as outras classes e que em conjunto representam o jogo de xadrez.

Esta classe expõe todos os métodos que são necessário para jogar, alternando dois jogadores depois de um movimento válido.

Implementa ainda uma classe para fazer o *print* do estado do board.

- **Enum: PieceColor**

*Descrição:* Enum com as duas cores das peças.

### 2.1.4 package chess.userinterface

Este pacote contem a classe que implementa a interface com os jogadores.

- **Classe: ConsoleInterface**

*Descrição:* esta classe implementa uma interface simples para ser usada em modo de carácter usando Unicode.

Esta classe só depende da classe `GameManager` para obter o estado do tabuleiro e realizar as jogadas.

- **Classe: `ChessBoardUI`**

*Descrição:* esta classe implementa uma interface gráfica de um tabuleiro de Xadrez.

Esta classe só depende da classe `GameManager` para obter o estado do tabuleiro e realizar as jogadas.

## 2.2 Arquitetura

### 2.2.1 Modelo MVC

Esta implementação segue um modelo MVC.

- **Model:** As classes e objetos - Enum - pertencentes ao pacote `chess.core.board.pieces` e `chess.core.board` gerem o estado da aplicação, realizam as validações e interagem com as estruturas de dados usadas para guardar os estados no decorrer da execução.
- **Controller:** As classes e objetos - Enum - no pacote `chess.core` atuam como um intermediário entre o Model e a View ao receber os *input* dos jogadores (ações da View) e traduzem as ações para o Model e/ou para a View. Faz ainda a gestão da alternância entre jogadas valida dos dois jogadores.
- **View:** As classes no pacote `chess.userinterface` são as interfaces com os jogadores, implementando uma representação do tabuleiro. Implementando o mínimo da lógica do jogo de xadrez e dependendo unicamente da classe `GameManager` que pertence ao model. Estão implementadas duas interfaces, uma para ser executada em modo carácter UTF-8, e outra para ser executada em modo gráfico usando o `javax.swing` e `java.awt`.

### 2.2.2 Estratégias de programação orientada por objetos

As seguintes estratégias foram usadas:

- **Encapsulamento:** A classe `GameManager` expõe todos os métodos necessário para jogar o jogo. Escondendo toda a lógica, regras e ações. A classe `Board` também esconde a estrutura de dados usada para guardar o estado do board. Apesar de neste caso usar um array de arrays, seria possível alterar para outra estrutura de dados, fazendo alterações mínimas nesta classe.
- **[1]Herança:** As classes que implementam cada tipo de peça, estendem uma classe abstrata de uma peça genérica. A escolha desta estratégia usou o princípio, por exemplo, *"bishop is a piece"*.
- **Injeção de dependências:** A classe `RuleMaster` implementa uma estratégia de injeção de dependência ao obrigar que seja instanciada com o `Board` como atributo. Em vez de ser ela a instanciar internamente. Só o código relevante:

```

public RulesMaster(Board board) {
    this.board = board;
}

```

Instanciação:

```

public class GameManager {
    ...
    private final RulesMaster ruleMaster;

    public GameManager() {
        this.board = Board.getInstance();
        this.ruleMaster = new RulesMaster(board);
    }
    ...
}

```

## 2.3 Modelos UML

### 2.3.1 Diagrama de classes

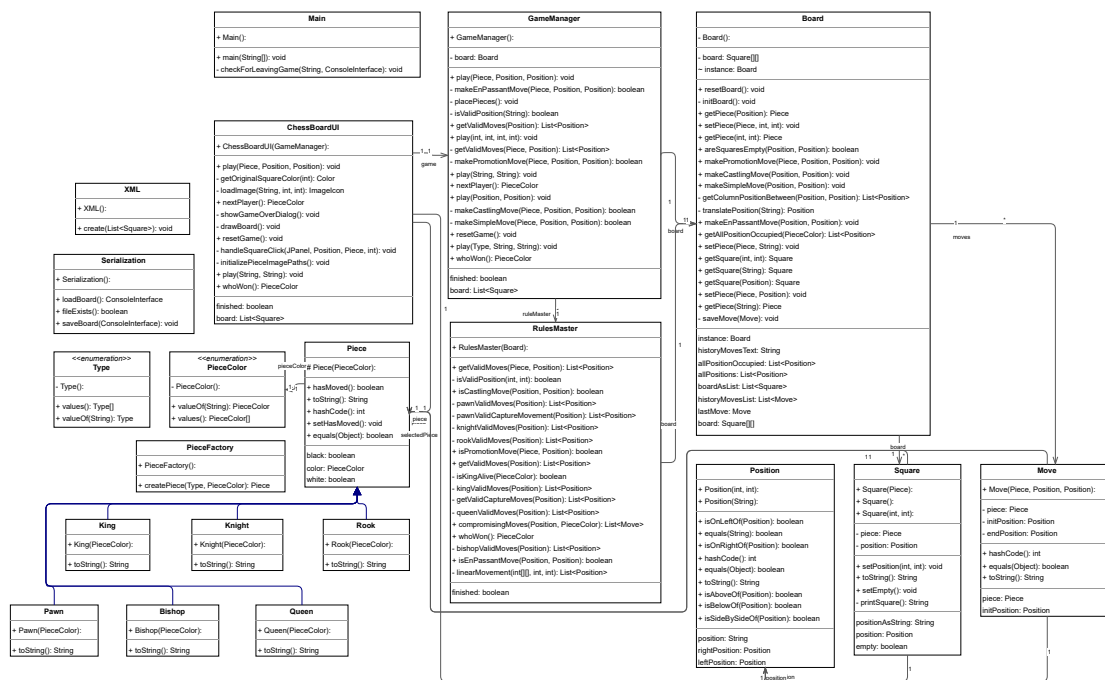


Figura 1: Diagrama de classes.



### 2.3.2 Diagramas de sequência

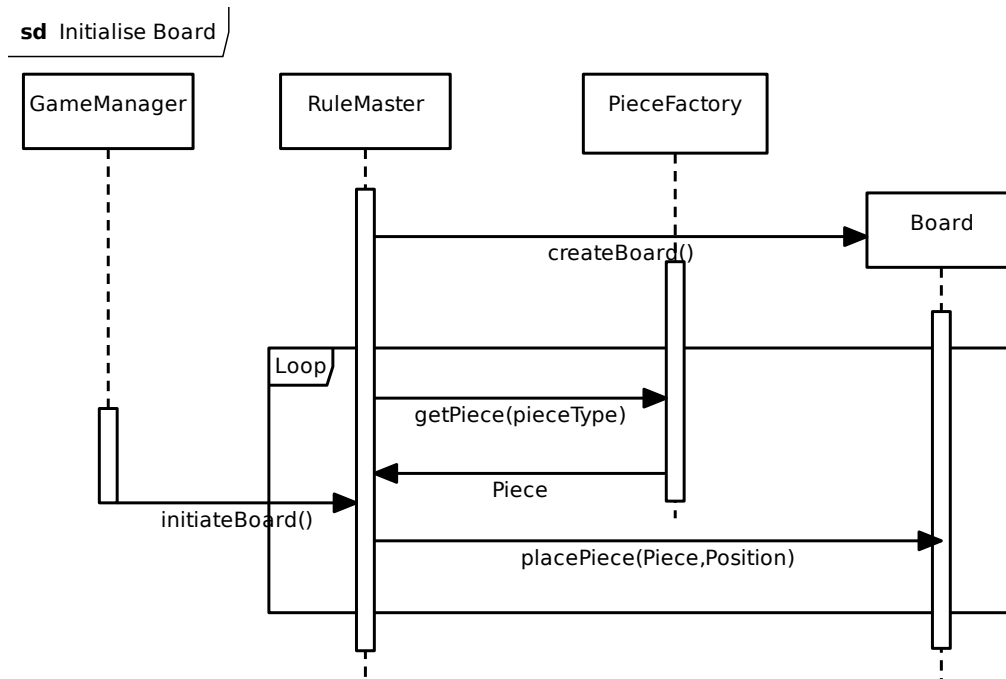


Figura 2: Diagrama de sequência da inicialização do tabuleiro.

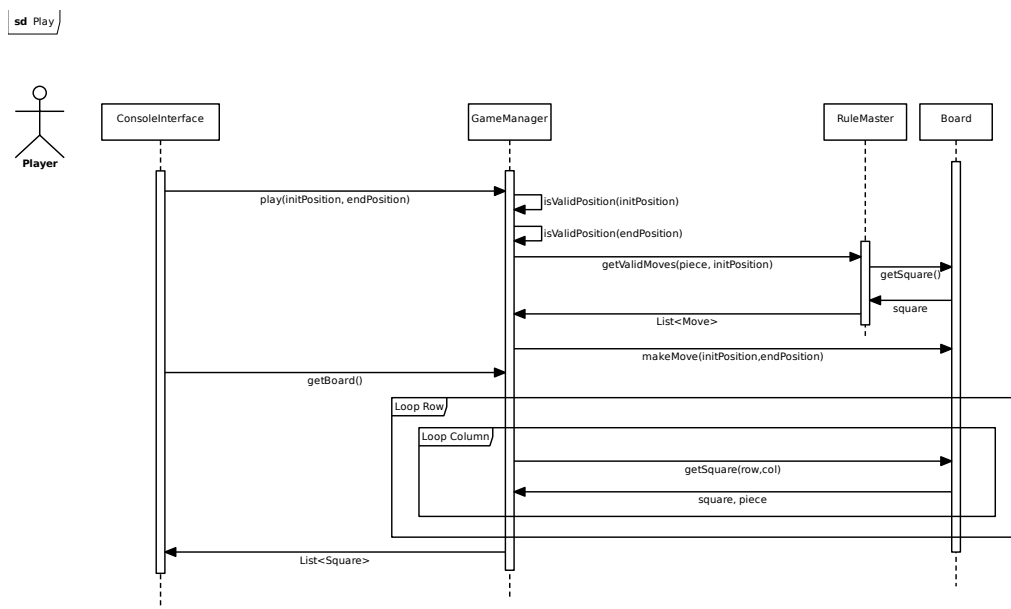


Figura 3: Diagrama de sequência simplificado da jogada.

Os modelos anteriores são uma representação simples e esboça da sequência com que os objetos interagem.

### 2.3.3 Padrões usados

Neste trabalho, sempre que adequado foram usados os [2]padrões. Os seguintes padrões foram usados:

- **Singleton**: A classe Board implementa este padrão para obrigar à existência de um único tabuleiro.

```
public class Board {
    static Board instance = null;
    ...

    private Board() {
        ...
    }

    public static Board getInstance() {
        if (instance == null) instance = new Board();
        return instance;
    }
}
```

- **Factory**: A classe PieceFactory implementa este padrão para encapsular o processo de criação de objetos das respectivas peças.

```
public class PieceFactory {
    public static Piece createPiece(Type type, Color color) {
        return switch (type) {
            case KING -> new King(color);
            case PAWN -> new Pawn(color);
            case ROOK -> new Rook(color);
            case QUEEN -> new Queen(color);
            case KNIGHT -> new Knight(color);
            case BISHOP -> new Bishop(color);
        };
    }
}
```

- **Facade**: A classe GameManager implementa este padrão para uma interface simplificada para jogo, escondendo toda a complexidade. Exemplo da implementação da classe e utilização dos métodos play() e isFinished(), é o suficiente para jogar. Os métodos printBoardUnicode() e nextPlayer permitem obter uma representação do tabuleiro - útil na ausência de uma interface gráfica mais sofisticada - e obter quem o próximo jogador - sem que isto seja necessário para realizar uma jogada.

```
GameManager gameManager = new GameManager();
Scanner play = new Scanner(System.in);
String initPosition, endPosition;

while (!gameManager.isFinished()) {
    System.out.println(gameManager.printBoardUnicode());
    System.out.print("Piece to move for ");
    System.out.print(gameManager.nextPlayer() == Color.WHITE
```

```

        ? "White:" : "Black:");
    initPosition = play.nextLine();
    System.out.print("Move_ to:");
    endPosition = play.nextLine();
    gameManager.play(initPosition, endPosition);
}

```

#### 2.3.4 Estrutura do projeto

Este projeto está estruturado da seguinte forma:

- `src\main\java`: local do código fonte.
- `src\main\resources`: local do ficheiro de configuração do Log4J.
- `src\test\java`: local das unidade de teste JUnit.
- `target\`: local do ficheiro `.jar` depois do *build*.
- `target\classes`: local do código depois do *build*.
- `target\classes`: local do código depois do *build*.
- `target\test-classes`: local do código das unidades de teste JUnit depois do *build*.
- `logs`: local onde é guardado os logs com as jogadas realizadas - usando Log4J.
- `savedFiles`: local onde é guardado o estado do tabuleiro - XML, DTD e serialização.

#### 2.3.5 Exportação do tabuleiro para XML e o uso de XPATH

A exportação do tabuleiro para XML consiste em transformar o estado atual do jogo de xadrez, representado por objetos Java, num documento XML estruturado. Este documento, salvo como `xadrez.xml`, descreve cada peça, a sua cor e posição, permitindo guardar e carregar facilmente o estado do jogo. O processo envolve a criação de elementos XML que correspondem à hierarquia do tabuleiro e das peças, garantindo que o DTD `xadrez.dtd` seja seguido para validação.

```

Element pieceElement = documentTabuleiro.createElement("Peca");
pieceElement.setAttribute("id", String.valueOf(i++));
pieceElement.setAttribute("tipo",
square.getPiece().getClass().getSimpleName());

Element posElement = documentTabuleiro.createElement("posicao");

posElement.appendChild(documentTabuleiro.createTextNode(
square.getPositionAsString()));
pieceElement.appendChild(posElement);

```

```

Element pieceColor = pathFinder(square.getPiece().getColor(),
documentTabuleiro);
pieceColor.appendChild(pieceElement);

```

No código fornecido, o XPATH é usado para navegar e encontrar elementos específicos dentro da estrutura XML, como o caso da função `pathFinder`. Esta função utiliza expressões XPath para localizar e selecionar os elementos `PecasPretas` ou `PecasBrancas` no documento XML.

```

private static Element pathFinder(PieceColor color,
Document documentTabuleiro) throws ParserConfigurationException,
IOException, SAXException, XPathExpressionException {
    XPathFactory xpathFactory = XPathFactory.newInstance();
    XPath xpath = xpathFactory.newXPath();

    String expression;
    if (color == PieceColor.BLACK) expression = "/Tabuleiro/PecasPretas";
    else expression = "/Tabuleiro/PecasBrancas";

    return (Element) ((NodeList) xpath.evaluate(expression,
documentTabuleiro, XPathConstants.NODESET)).item(0);
}

```

### 2.3.6 Exportação do tabuleiro usando *Serialization*.

A função `saveBoardGUI` guarda o objeto *GameManager* (que tem toda a informação do jogo) numa sequência de bytes e guarda-o no ficheiro *latestChessGameGUI.bin*. Isto permite gravar o estado atual do jogo no disco.

```

public static void saveBoardGUI(GameManager game) {
    try (FileOutputStream streamFile = new FileOutputStream("savedFiles/latestChessGameGUI.bin")) {
        ObjectOutputStream objectFile = new ObjectOutputStream(streamFile);
        objectFile.writeObject(game);
        objectFile.flush();
        objectFile.close();
        System.out.println("Tabuleiro salvo com sucesso!");
    } catch (IOException e) {
        System.out.println("Erro ao salvar o tabuleiro!");
    }
}

```

Por outro lado, a função `loadBoardGUI` carrega esse ficheiro para uma instância do *GameManager*. Assim sendo possível voltar a jogar o jogo desde última vez que o programa foi fechado.

```

public static GameManager loadBoardGUI() {
    try (FileInputStream streamFile = new
        FileInputStream("savedFiles/latestChessGameGUI.bin");
        ObjectInputStream objectFile = new ObjectInputStream(streamFile)) {
        return (GameManager) objectFile.readObject();
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Erro ao carregar o tabuleiro");
        return new GameManager();
    }
}

```

### 2.3.7 Utilização do Log4J

Com a utilização do [3]Log4J, todas as jogadas são registadas no ficheiro logs\log4j2.log.

Para usar o Log4J usasse esta configuração:

```

<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="OFF">
    <Appenders>
        <File name="LOGFILE" fileName="logs/log4j2.log">
            <PatternLayout pattern=
                "[%5level] [%d{yyyy-MM-dd HH:mm:ss.SSS}] [%t] [%c{1}] - %msg%n"
            />
            <ThresholdFilter level="info"
                onMatch="ACCEPT" onMismatch="DENY"/>
        </File>
    </Appenders>

    <Loggers>
        <Root level="info">
            <AppenderRef ref="LOGFILE"/>
        </Root>
    </Loggers>
</Configuration>

```

### 2.3.8 Utilização do JUnit

Como exemplo de utilização de JUnit, foi implementada uma unidade de teste para a classe Position, com o nome de PositionTest.

No projeto estas unidades de test, estarão dentro da pasta src/test/java/.

### 2.3.9 Gestão de dependências

A utilização das livrarias ou módulos externos Log4J e JUnit, obriga a uma configuração do IDE usado e à gestão das dependências deste livrarias - isto é outra livrarias auxiliares para as livrarias acima mencionadas. Para ajudar nisto usei o maven [4] diretamente no IDE. Parte relevante do ficheiro de configuração usado - pom.xml:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.junit</groupId>
      <artifactId>junit-bom</artifactId>
      <version>5.10.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.24.3</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.24.3</version>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

### 2.3.10 Considerações sobre performance

Neste trabalho os métodos que tem um impacto maior na performance é a iteração do array de arrays usado na classe Board, em que é percorrido todas as colunas dentro de cada linha - isto é um ciclo for dentro de um outro ciclo for.

No entanto como o tamanho do tabuleiro é no máximo 64 casas -  $8 \times 8$  - este é um valor muito pequeno para ter um qualquer impacto significativo.

Num caso hipotético de um tabuleiro gigantesco - o que não seria o jogo de xadrez - a iteração poderia ter impacto, porque este processo tem um

complexidade quadrática.

Neste caso teria ser necessário uma outra estrutura de dados para armazenar o estado do tabuleiro. Uma possibilidade seria usar uma `HashMap`. No que toca à performance, por o projeto ser em Java, que não é a linguagem mais rápida, a movimentação das peças na GUI pode ter pequenos atrasos de poucos segundos. Embora breves, estes podem ser notados pelo utilizador.

## 2.4 Execução

Para executar é necessário que a janela de terminal reconheça caracteres Unicode. As livrarias do Log4J estaram que estar na pasta `libs`. Executar o comando abaixo:

```
java -cp .:libs Main
```

Com a utilização do maven, obtém-se a possibilidade de gerar um ficheiro JAR - arquivo que junta um ou mais ficheiros e/ou pastas num único ficheiro - incluindo livrarias dependentes como o Log4J -, ao mesmo tempo que os comprime. O que permite executar o programa da seguinte forma:

```
java -jar chess-1.0-jar-with-dependenciesUTFMode.jar
```

ou

```
java -jar chess-1.0-jar-with-dependenciesGUIMode.jar
```

## Referências

- [1] ISEL. Modelação e programação. Slides da disciplina, 2025.
- [2] Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John, and Grady Booch. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1994.
- [3] Apache Logging Services. Getting started :: Apache log4j. <https://logging.apache.org/log4j/2.x/manual/getting-started.html>, 2025. Accessed: 2025-05-31.
- [4] Helen Scott. Working with maven. <https://www.jetbrains.com/guide/java/tutorials/working-with-maven/>, 2024. Accessed: 2025-05-31.