Universidade Federal de São Carlos
Bacharelado em Ciência da Computação

Grupo 8
FERNANDO MACHADO AOKI
HENRIQUE DE CAMARGO BRICOLERI
LEONARDO SOUZA DA COSTA
LORENZO LIMA GERMANO
THIAGO KRAIDE DE LIMA FERNANDES

XADREZ

Projeto Final - Programação Orientada À Objetos

Sorocaba - SP
2025

1. Metodologia..... 3
2. Classes Utilizadas..... 4
2.1 Gerenciador..... 4
2.2 Jogo..... 4
2.3 Jogador..... 5
2.4 Jogada..... 5
2.5 Caminho..... 6
2.6 Casa..... 6
2.7 Tabuleiro..... 6
2.8 Peças..... 7
3. Condições de Erro..... 7
3.1 Gerenciador..... 7
3.2 Jogo..... 7
3.3 Jogador..... 8
3.4 Tabuleiro..... 8
3.5 Peças..... 8
4. Testes Realizados..... 9
4.1 Teste da Classe Bispo:..... 9
4.2 Teste da Classe Torre:..... 9
4.3 Teste da Classe Cavalo:..... 9
4.4 Teste da Classe Peão..... 10
4.5 Teste da Classe Dama..... 10
4.6 Teste da Classe Rei..... 10
4.7 Teste da Classe Casa..... 10
4.8 Teste da Classe Jogador..... 11
4.9 Teste da Classe Tabuleiro..... 11
4.10 Teste da Classe Caminho..... 11
4.11 Teste da classe jogada..... 11
5. Método de Utilização..... 12
6. Problemáticas e Dificuldades..... 13
6.1 Tratamento de entrada de usuário..... 13
6.2 Carregamento de arquivos..... 13
6.3 Polimorfismo e hierarquia de peças..... 13
7. Conclusão..... 13

1. Metodologia

Para lidar com a complexidade do projeto, adotamos uma abordagem estruturada, focando na divisão de tarefas, organização e comunicação. Inicialmente, distribuímos as classes principais entre os membros da equipe: um ficou responsável pelo Tabuleiro e Casa, garantindo a correta representação do layout e das posições; outro assumiu Jogador e Jogada, cuidando da interação com os usuários e validação de movimentos; um terceiro desenvolveu a hierarquia de Peça, implementando o polimorfismo para cada tipo específico; e o último trabalhou no Gerenciador e Jogo, integrando todas as partes no fluxo principal.

Utilizamos o **Git** para versionamento, criando branches específicos para cada funcionalidade e realizando merges somente após revisão detalhada. Isso evitou conflitos e garantiu um código estável. Reuniões ajudaram a alinhar o progresso e resolver bloqueios rapidamente, utilizando um canal dedicado no **Discord,** que facilitou a comunicação contínua.

Adotamos uma estratégia de testes em duas fases: primeiro, cada membro testou individualmente suas classes implementadas (Peça, Tabuleiro, Jogador etc.) com casos específicos, verificando comportamentos básicos e tratamento de exceções. Na segunda fase, integramos todas as classes e realizamos testes sistêmicos, simulando partidas completas para validar interações entre componentes.

Essa abordagem parte-todo permitiu identificar erros pontuais antes da integração, garantindo maior estabilidade ao sistema final. Utilizamos o método testes() do Gerenciador para automatizar verificações críticas.

A documentação foi atualizada em paralelo, com comentários no código e um relatório semanal registrando decisões técnicas. Priorizamos as classes base primeiro, como Tabuleiro e Peça, e deixamos funcionalidades avançadas, como xeque-mate, para a fase final.

Essas estratégias permitiram um desenvolvimento eficiente, com baixo retrabalho e entrega dentro do prazo, resultando em um sistema coeso e bem estruturado.

2. Classes Utilizadas

2.1 Gerenciador

A classe Gerenciador atua como o núcleo central do programa, responsável por coordenar todas as interações iniciais e o fluxo principal da aplicação. Sua função principal é gerenciar o menu do jogo, oferecendo opções como iniciar uma nova partida, carregar um jogo salvo ou encerrar a aplicação, garantindo uma comunicação clara com o usuário. Além disso, ela implementa a lógica de persistência de dados, salvando o estado atual do jogo em arquivos de texto e recuperando-o quando necessário, seguindo um formato específico que armazena os nomes dos jogadores e o histórico de jogadas.

Como detentora do método main(), a classe Gerenciador é responsável por inicializar todas as outras classes do sistema, como Jogo, Tabuleiro e Jogador, criando uma estrutura coerente para a execução do xadrez.

Outro aspecto crucial do Gerenciador é sua capacidade de lidar com entradas inválidas, como nomes de arquivos inexistentes ou comandos incorretos, utilizando tratamento de exceções para evitar falhas inesperadas. Essa robustez garante que o programa seja tolerante a erros e ofereça feedback adequado ao usuário em situações problemáticas. Em resumo, a classe serve como a ponte entre o usuário e a lógica interna do jogo, assegurando uma experiência fluída e bem estruturada desde o início até o encerramento da partida.

2.2 Jogo

A classe Jogo é responsável por coordenar toda a dinâmica e lógica do jogo. Ela atua como o controlador principal que gerencia o fluxo da partida, desde a inicialização até o final da partida. Esta classe mantém e atualiza constantemente o estado do jogo, incluindo a posição de todas as peças no tabuleiro, o turno atual dos jogadores e as condições especiais como xeque e xeque-mate.

Entre suas principais responsabilidades estão: inicializar e configurar o tabuleiro com as peças nas posições corretas; alternar os turnos entre os jogadores de forma ordenada; validar cada movimento de acordo com as regras específicas do xadrez; detectar situações de xeque e xeque-mate; e gerenciar o histórico completo de todas as jogadas realizadas. A classe também é encarregada de atualizar a interface do usuário após cada movimento, garantindo que os jogadores visualizem sempre o estado atualizado do tabuleiro.

2.3 Jogador

A classe Jogador representa cada participante do jogo de xadrez, sendo responsável por armazenar e gerenciar todas as informações específicas de um competidor. Ela guarda o nome do jogador e a cor das peças que controla (brancas ou pretas), mantendo também um registro das peças capturadas durante a partida. Durante o jogo, esta classe atua como intermediária entre a interface do usuário e a lógica do jogo, solicitando e processando as jogadas do participante.

Entre suas principais funcionalidades estão: validar se é o turno do jogador atual, receber e interpretar os comandos de movimento (como "e2e4" ou "parar"), e gerenciar o conjunto de peças sob seu controle, tanto as ativas no tabuleiro quanto as já capturadas. A classe também é responsável por fornecer feedback visual ao usuário, exibindo suas peças capturadas durante a partida. Com um design intuitivo, ela facilita a interação humano-computador, transformando entradas simples em ações complexas no tabuleiro, sempre em conformidade com as regras do xadrez.

2.4 Jogada

A classe Jogada encapsula toda a lógica e informações relacionadas a um movimento específico no jogo de xadrez, atuando como a representação concreta de uma ação realizada

por um jogador. Ela armazena as coordenadas de origem (linhaO, colunaO) e destino (linhaD, colunaD), além de manter referência à peça movimentada e à peça adversária capturada, quando aplicável. Esta classe é responsável por validar a legalidade básica do movimento, verificando se as posições estão dentro dos limites do tabuleiro e se a peça de origem pertence ao jogador atual.

Além da validação inicial, a classe Jogada implementa a lógica para identificar situações especiais como captura, xeque e xeque-mate, trabalhando em conjunto com as classes Tabuleiro e Caminho para analisar o contexto completo do movimento. Ela também gera a representação textual da jogada no formato padrão para registro no histórico de partidas. Durante a execução, a classe coordena com a classe Jogo para atualizar o estado do tabuleiro e verificar as consequências estratégicas do movimento, garantindo que todas as regras do xadrez sejam respeitadas.

A implementação da Jogada segue princípios de encapsulamento rigoroso, expondo apenas métodos necessários para sua validação e execução, enquanto mantém internamente os detalhes de como as verificações são realizadas. Esta abordagem permite que a lógica complexa de movimentação no xadrez seja tratada de forma modular e coesa, facilitando a manutenção e extensão do código para implementar regras avançadas no futuro.

2.5 Caminho

A classe Caminho é responsável por modelar e analisar o trajeto percorrido por uma peça durante uma jogada no xadrez. Ela determina todas as casas intermediárias entre a posição inicial (casaInicial) e final (casaFinal) do movimento, verificando se o percurso está livre para a peça se deslocar. Esta classe diferencia os tipos de trajetória (retilíneo, diagonal ou em L) conforme as regras específicas de cada peça, sendo crucial para validar movimentos como os da torre (que exige caminho reto livre) ou do bispo (que requer diagonal desobstruída).

2.6 Casa

A classe Casa gerencia o estado de ocupação de cada posição no tabuleiro, armazenando a peça que a ocupa (ou null se vazia). Oferece métodos básicos como getPeca(), setPeca() e estaOcupada() para controle da peça associada. Implementa o núcleo essencial para verificar disponibilidade de casas durante o jogo. Apesar da simplicidade atual, serve como base para expansões como adição de coordenadas e validações de movimento. Integra-se diretamente com as classes Peca e Tabuleiro na lógica do xadrez.

2.7 Tabuleiro

A classe Tabuleiro é o componente central que estrutura e gerencia todo o layout do jogo de xadrez. Ela cria e mantém uma matriz 8x8 de objetos Casa, onde cada posição pode

armazenar uma peça ou permanecer vazia. Durante a inicialização, o método posicionarPecasIniciais() configura todas as peças em suas posições tradicionais, separando peças brancas (linhas 1-2) e pretas (linhas 7-8).

A classe oferece métodos essenciais para navegação e manipulação do tabuleiro, incluindo getCasa() para acessar posições específicas usando notação de xadrez (como "e4") e métodos de conversão entre índices numéricos e coordenadas alfabéticas. A função noLimite() valida se movimentos propostos estão dentro dos limites do tabuleiro.

O método desenho() gera uma representação visual completa do estado atual, exibindo peças (usando seus símbolos específicos), coordenadas e um layout alternado de casas claras e escuras.

2.8 Peças

O sistema implementa as seis peças tradicionais do xadrez através de classes especializadas que herdam da classe base Peça. O Rei move-se uma casa em qualquer direção e é central para verificações de xeque. A Dama, a peça mais versátil, combina os movimentos da Torre (horizontal/vertical) e do Bispo (diagonal). As Torres se movem em linhas retas horizontais ou verticais, enquanto os Bispos operam exclusivamente nas diagonais, cada um permanecendo em casas de uma única cor. O Cavalo possui o movimento único em "L", sendo a única peça que pode saltar sobre outras. Já o Peão tem comportamento especial - avança frontalmente mas captura diagonalmente, com regras específicas para seu primeiro movimento. Todas as classes implementam os métodos movimentoValido(), desenho() e caminho(), permitindo tratamento polimórfico enquanto mantêm suas regras individuais. Esta estrutura garante a correta simulação das regras tradicionais do xadrez, desde movimentos básicos até situações complexas como roque e xeque-mate, com total consistência lógica.

3. Condições de Erro

3.1 Gerenciador

A classe *gerenciador* é responsável por carregar e salvar o jogo de xadrez, um erro pode ocorrer quando o usuário tenta carregar o jogo a partir de um arquivo que não existe no sistema, isso pode acontecer devido a:

- **Arquivo inexistente**:
    - Usuário digitou nome incorreto do arquivo;
    - Arquivo foi movido ou excluído do sistema;
    - Caminho relativo/absoluto está errado.
- **Formato inválido**:
    - Arquivo corrompido ou com estrutura diferente do esperado;
    - Dados faltantes (como nome de jogadores);
    - Jogadas em formato incorreto (ex: "9a" em vez de "2a4a").

- **Permissões**:
    - Sem permissão de leitura/escrita no diretório.
    - Arquivo aberto por outro processo.

3.2 Jogo

A classe *jogo* é o núcleo do sistema, responsável por orquestrar toda partida de xadrez, ela controla o fluxo do jogo, valida as jogadas, gerencia o tabuleiro e os jogadores, e verifica as condições de vitória/derrota, apresenta erros como:

- **Xeque não resolvido:**
    - Jogador move peça deixando rei em xeque;
    - Falha ao detectar xeque-mate.
- **Estado inválido:**
    - Tabuleiro em configuração impossível;
    - Peças duplicadas ou faltantes;
    - Turno incorreto (jogador jogando fora de vez);
- **Movimentação:**
    - Tentativa de mover peça inexistente;
    - Movimento interrompido (ex: peça removida durante jogada);
    - Conflito de peças (duas peças na mesma casa).

3.3 Jogador

A classe jogador é responsável por gerenciar as interações do usuário, como inserção de nomes e de jogadas. Abaixo estão as principais condições de erro:

- **Jogada mal formada:**
    - Coordenadas fora do tabuleiro ("i5", "a0");
    - Formato incorreto ("e2-e4" em vez de "e2e4");
    - Caracteres inválidos ("%2", "b?").
- **Movimento inválido:**
    - Peça que não pode mover-se da forma solicitada;
    - Caminho bloqueado por outras peças;
    - Tentativa de captura ilegal.

3.4 Tabuleiro

A classe Tabuleiro gerencia a estrutura do jogo. Seus principais possíveis erros são:

- **Posições inválidas:**
    - Acesso a casa inexistente (linha 0, coluna 'i');
    - Referência nula à peça.
- **Configuração:**
    - Peças em posições iniciais erradas;
    - Duas peças na mesma casa;

- Peças faltantes no início do jogo.

3.5 Peças

De forma generalizada as classes referentes às peças podem apresentar os seguintes erros:

- **Movimento ilegal**;
- **Estado inconsistente**:
  - Peça capturada ainda aparecendo no tabuleiro.

4. Testes Realizados

4.1 Teste da Classe Bispo:

- **Objetivo**: Verificar se a peça Bispo está corretamente representada e se seus movimentos são válidos conforme as regras do xadrez.

  Verificou-se o método desenho() para garantir que retorna a letra “B” (maiúscula para peças brancas).

  Testou-se o método movimentoValido(2, ‘c’, 5, ‘f’), que corresponde ao movimento diagonal válido do bispo da casa 2c para 5f.

4.2 Teste da Classe Torre:

- **Objetivo**: Verificar o movimento da torre e seu comportamento.

  Instanciou-se uma torre preta.

  Checou-se se o desenho retorna “t” (minúsculo para peças pretas).

  Testou-se o movimento horizontal válido de 1a para 1h.

  Verificou-se se o caminho inclui todas as casas na linha 1 entre as colunas ‘a’ até ‘h’.

4.3 Teste da Classe Cavalo:

- **Objetivo:** Confirmar que o cavalo executa movimento em “L” corretamente.

  Verificou-se que o método desenho() retorna “C”.

  Testou-se o movimento de 2b para 4c, que corresponde a um movimento válido do cavalo.

Confirmou-se que o método caminho () retorna apenas origem e o destino pois o cavalo “pula” casas intermediárias.

4.4 Teste da Classe Peão

- **Objetivo**: Verificar o comportamento do peão para movimentação simples.

  Checou-se o método desenho() retornando “P”.

  Testou-se o movimento válido de avanço simples, de 2a para 3a.

  Verificou-se que o caminho inclui apenas a origem e o destino.

4.5 Teste da Classe Dama

- **Objetivo**: Avaliar a capacidade da dama se mover tanto na diagonal quanto em linha reta.

  Criou-se uma dama preta.

  Confirmou que o desenho retorna “d”.

  Testou-se um movimento diagonal válido de 4d para 7g.

  Verificou-se se o caminho inclui todas as casas intermediárias entre origem e destino (4d, 5e, 6f, 7g).

4.6 Teste da Classe Rei

- **Objetivo**: Garantir que o rei se mova apenas uma casa em qualquer direção.

Instanciar-se um rei branco.

Checou-se se o método desenho() retorna “R”.

Testou-se um movimento simples de 1e para 2e.

Verificou-se que o caminho inclui apenas a casa de origem e a casa destino.

4.7 Teste da Classe Casa

- **Objetivo:** Verificar o estado inicial de uma casa no tabuleiro.

  Criou-se uma casa sem peça.

  Verificou-se o método estaOcupada().

  O retorno obtido foi false, identificando que ela está livre

4.8 Teste da Classe Jogador

- **Objetivo:** Validar a criação de um jogador com nome e cor associada.

  Criou-se um jogador chamado “Alice” com cor branca.

  Verificou-se se os métodos getNome() e getCor() retornaram valores corretos.

4.9 Teste da Classe Tabuleiro

- **Objetivo:** Testar a configuração inicial do tabuleiro e seu método de desenho.

  Instanciou-se um novo tabuleiro.

  Chamou-se o método desenho() para exibir o estado inicial das peças.

4.10 Teste da Classe Caminho

- **Objetivo:** Verificar se um caminho entre casas está livre para movimentação.

Criou-se um objeto Caminho.

Adicionaram-se duas casas (vazias) ao caminho.

Consultou-se o método estaLivre().

4.11 Teste da classe jogada

- **Objetivo**: Avaliar se uma jogada realizada pelo jogador é válida no estado atual do tabuleiro.

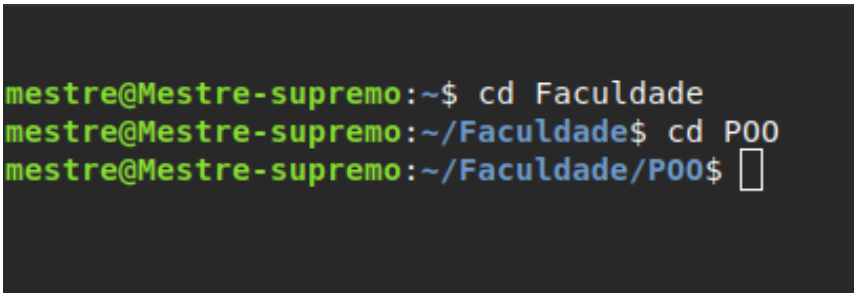
  Criou-se uma jogada da jogadora "Alice" movimentando de 2c para 4c com caminho "2c3c4c".

  Verificou-se o método ehValida() passando o tabuleiro atual.

5. Método de Utilização

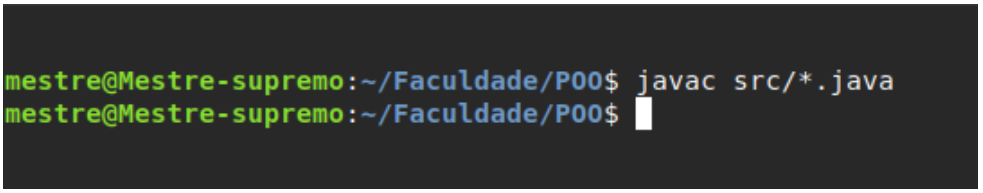
Segue abaixo um passo a passo de como rodar o código:

1. Baixar o arquivo .zip enviado em conjunto com o relatório.
2. Descompactar o arquivo em um diretório com caminho de fácil acesso.
3. Acessar o local em que a pasta do Xadrez está:



(Exemplo de acesso, o caminho depende da pasta que for salvo o arquivo)

4. Executar o comando **javac *nomedapasta*/*.java**.



5. Após o arquivo ser compilado, executar o arquivo java em que está contido a main, nesse caso, gerenciador.java, por meio do comando **java**

***nomedoarquivo*/Gerenciador.java**

```
mestre@Mestre-supremo:~/Faculdade/POO$ javac src/*.java
mestre@Mestre-supremo:~/Faculdade/POO$ java src/Xadrez2.java
Bem-vindo ao Jogo de Xadrez!
1 - Novo Jogo
2 - Carregar Jogo
3 - Sair
Escolha uma opção:
```

6. Agora, só seguir as instruções que aparecem no terminal.

6. Problemáticas e Dificuldades

6.1 Tratamento de entrada de usuário

Uma das dificuldades foi garantir que o programa não encerrasse abruptamente quando o usuário digitava uma entrada inesperada (por exemplo, texto ao invés de número no menu principal). A solução foi adotar o Scanner.nextLine() e validar manualmente as entradas, permitindo mensagens amigáveis e evitando falhas inesperadas.

6.2 Carregamento de arquivos

A implementação do carregamento de partidas salvas exigiu atenção ao formato do arquivo e à restauração correta do estado do jogo. Houve dúvidas sobre como garantir que todas as informações pertinentes (como peças capturadas) fossem restauradas. A solução adotada foi aplicar todas as jogadas do histórico , o que atualiza o tabuleiro e as peças capturadas indiretamente.

6.3 Polimorfismo e hierarquia de peças

A manipulação polimórfica das peças, especialmente para métodos como caminhoCaminho, exigiu ajustes na hierarquia de classes e na declaração de métodos abstratos. Isso foi fundamental para garantir que todas as peças pudessem ser tratadas de forma uniforme, mas gerou dúvidas e erros de compilação até o ajuste correto.

7. Conclusão

O desenvolvimento do projeto de xadrez demonstrou a aplicação eficaz dos princípios de Programação Orientada a Objetos na resolução de um problema complexo. A metodologia estruturada, que abrangeu a divisão de tarefas, o controle de versão e testes sistemáticos, foi fundamental para a construção de um sistema coeso e funcional. A arquitetura, organizada em

classes com responsabilidades bem definidas como Gerenciador, Jogo e a hierarquia polimórfica de Peças, provou-se adequada para gerenciar a lógica do jogo.

A superação de desafios, como o tratamento de entradas de usuário e a persistência de dados, consolidou o aprendizado técnico da equipe. Os testes realizados validaram a correta implementação das regras e a estabilidade dos componentes, assegurando a confiabilidade do software.

Conclui-se, portanto, que o projeto atingiu com sucesso seus objetivos, resultando em um sistema que não apenas implementa as regras do xadrez, mas também representa uma aplicação prática bem-sucedida de conceitos essenciais da engenharia de software. O resultado final é um produto robusto que reflete a importância do planejamento, da organização e da colaboração no ciclo de desenvolvimento.