UNIVERSIDADE FEDERAL DE SÃO CARLOS

Grupo
Bernardo Guimarães Evangelista - 792171
Vinicius Gabriel Nanini da Silva - 795181

Projeto 2

Jogo de Xadrez

Relatório do Projeto - Projeto 2
Programação Orientada a Objetos
Orientação: Katti Faceli

Sorocaba
20/11/2021

Sumário

1 - Introdução

O objetivo deste trabalho é desenvolver a segunda fase de um jogo de xadrez, usando os conhecimentos obtidos na disciplina de Programação Orientada a Objetos.

2 - Classes

- **Peça :**

A classe Peça é uma classe pai para todas peças do Jogo, onde uma peça específica herda atributos e funções da classe. Esta classe possui 4 atributos sendo eles: “preto” uma variável do tipo boolean para determinar a cor da peça, “vivo” também booleana para saber se a peça está viva no jogo, “linha” e “coluna” do tipo int e char, respectivamente para localizar a posição da peça específica no tabuleiro. Também contamos com 4 métodos, sendo dois deles abstratos: “desenho()” que retorna um char que representa a peça e “checaMovimento(int linhaOrigem, char colunaOrigem, int linhaDestino, char colunaDestino)” que verifica se o movimento de uma determinada peça está correta. As outras duas são: isVivo(), que retorna se tal peça ainda está viva no tabuleiro e morrer() que seta a variável vivo para falso. Nessa classe contamos com 6 peças que são:

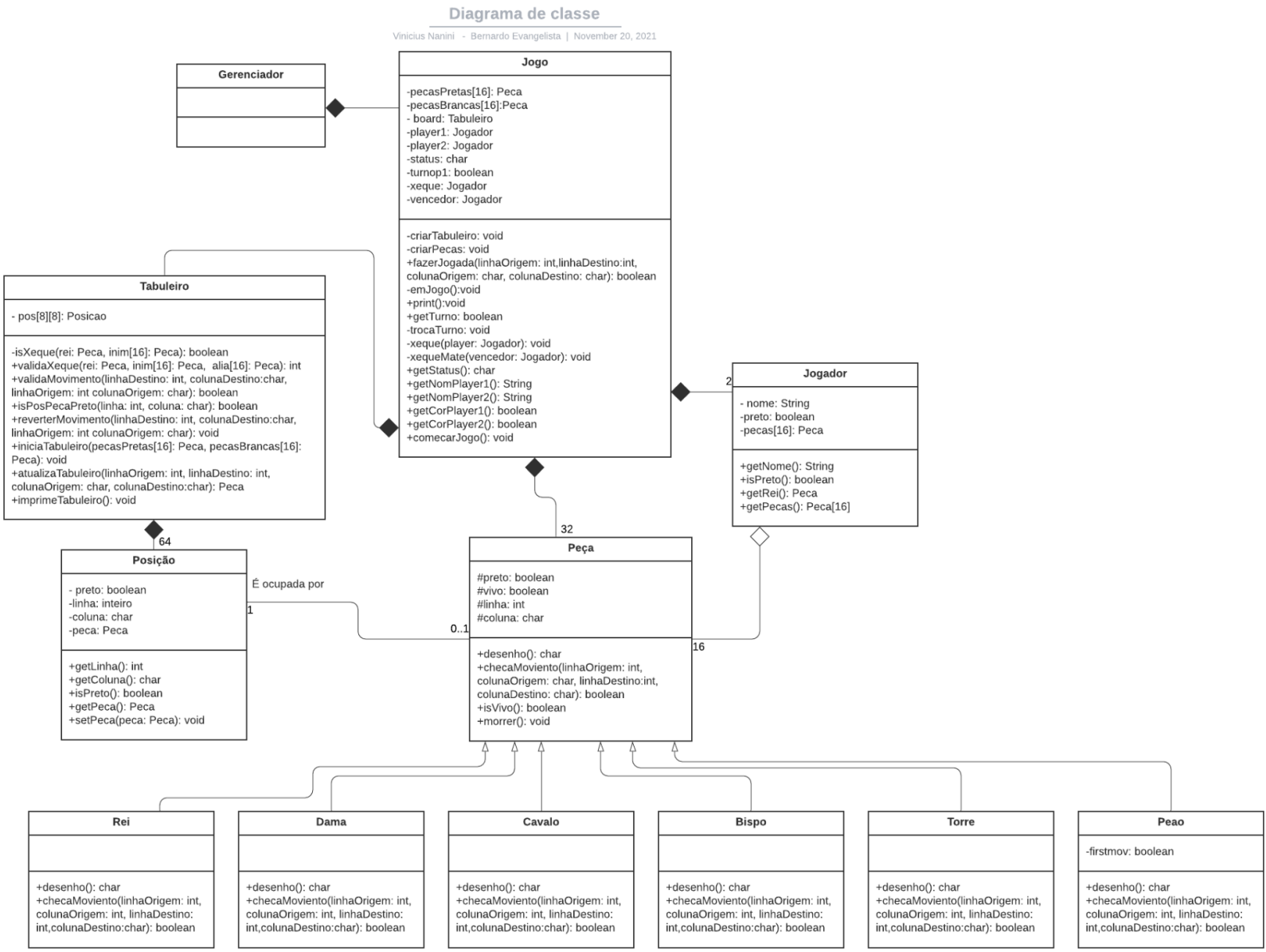
  - **Rei :** o desenho do rei é ‘R’, quando preto e ‘r’ quando branco. Para determinar se o movimento está correto, basta checar se ele se movimentou estritamente uma casa na horizontal ou vertical ou se ele movimentou uma casa na diagonal (para determinar se um movimento é diagonal deve-se checar se o movimento horizontal é igual ao vertical).
  - **Dama :** o desenho da dama é ‘D’, quando preta e ‘d’, quando branca. Para determinar se o movimento está correto, basta checar se ela se movimentou estritamente na horizontal ou vertical (a variação das colunas ou linhas deve ser zero) ou se movimentou na diagonal (para determinar se um movimento é diagonal deve-se checar se o movimento horizontal é igual ao vertical).
  - **Cavalo :** o desenho do cavalo é ‘C’, quando preto e ‘c’, quando branco. Para determinar se o movimento está correto, basta checar se ele movimentou duas casas na horizontal e uma na vertical; alternativamente, uma casa na horizontal e duas na vertical.
  - **Bispo :** o desenho do bispo é ‘B’, quando preto e ‘b’, quando branco. Para determinar se o movimento está correto, basta checar se ele se movimentou na diagonal (para determinar se um movimento é diagonal deve-se checar se o movimento horizontal é igual ao vertical).

    - **Torre :** o desenho da torre é ‘T’, quando preta e ‘t’, quando branca. Para determinar se o movimento está correto, basta checar se ele se movimentou estritamente na horizontal ou vertical (ou seja, a variação das colunas ou linhas deve ser zero).
    - **Peão :** o desenho do peão é ‘P’, quando preto e ‘p’, quando branco. O peão pode se mover duas casas quando é seu primeiro movimento e pode se mover na diagonal se for atacar outra peça , desta forma a classe peão deve ter um atributo que determina se é o seu primeiro movimento. Sabendo disso, para determinar se o movimento está correto, basta checar se ele movimentou estritamente 2 casas na vertical (se for seu primeiro movimento) ou 1 casa na vertical e se for preto este movimento deve ser para baixo, se for branco deve ser para cima.
- **Posição :** cada posição tem um identificador de linha (de 1 à 8) e coluna (de ‘a’ à ‘h’), sua cor e se está ocupada por uma peça (se sim, devo saber qual). Esta classe não possui nenhum método fora dos setters e getters, mas estes são necessários, se uma peça se movimentar deve-se atualizar o atributo que identifica a presença de peças, por exemplo.
    - getLinha: retorna a linha da posição.
    - getColuna: retorna a coluna da posição.
    - isPreto: retorna a cor da peça que ocupa aquela posição.
    - setPeca: Determina a peça que ocupa aquela posição.
    - getPeca: Retorna a peça que ocupa determinada posição.
- **Tabuleiro :** seu único atributo é uma matriz de posições que forma o tabuleiro. O tabuleiro deve verificar se um movimento está dentro de seus limites e, em conjunto com os métodos das peças, determina se um movimento é válido e se há xeque e com base nisso verificar se existe um possível xeque-mate, movendo todas as peças aliadas para verificar se existe uma possibilidade de fuga do rei, sendo esses movimentos revertidos depois de testados pela método reverterMovimento(). Deve também se atualizar (atualizaTabuleiro()) depois que uma jogada é feita e poder imprimir o estado de todas as posições através do método imprimeTabuleiro().
    - iniciaTabuleiro: Método responsável por iniciar Tabuleiro com as 32 peças, colocando nas posições iniciais.
    - isXeque: Esse método faz uma série de movimentos em comparação com a posição do rei sobre todas as peças inimigas, verificando se ela pode ser eliminada, sendo assim declarando um xeque.

    - validaXeque: Faz uma varredura de possíveis movimentos das peças aliadas e do próprio rei para verificar se é possível sair do estado xeque, caso não, quer dizer que é um xeque-mate.
    - atualizaTabuleiro: Responsável por atualizar a posição de uma peça após um movimento válido.
    - reverterMovimento: Utilizado para reverter o movimento após a verificação de possíveis movimentos para sair ou colocar o rei em xeque, uma vez que o movimento é feito para teste.
    - isPosPecaPreto: Esse método pega as informações de linha e coluna passada para verificar se aquela posição possui alguma peça, se sim retorna a cor dela em boolean (true para preta e false para branca).
    - validaMovimento: Responsável por validar o movimento de uma peça, verificando se tal movimento é possível para determinada peça.
    - imprimeTabuleiro: Esse método imprime todo o tabuleiro sendo “O” representando a cor preta e “-” a cor branca e as peças são representadas pelo seu desenho.
- **Jogador :** esta classe deve guardar como atributo o nome do jogador e a cor que o jogador controla. A classe deve ter acesso às 16 peças que estão sob o seu controle, através desses atributos existem métodos para retornar o nome do jogador, a cor de suas peças e seu rei.
    - getNome: Retorna o nome do jogador.
    - isPreto: Retorna qual é a cor da peça do jogador.
    - getRei: Esse método retorna a peça Rei do jogador.
    - getPecas: Responsável por retornar um vetor de Peças pertencentes ao jogador.
- **Jogo :** é esta classe que cria o tabuleiro, as peças, os jogadores e controla todo sobre o andamento do jogo, portanto ela tem atributos que determinam de quem é o turno e qual é o estado do jogo (em andamento, xeque, xeque-mate, etc) e decide quando o turno e o estado são trocados. Esta classe também recebe a entrada do jogador através do método fazerJogada() e deve decidir se é uma jogada válida ou não, caso seja correta, deverá atualizar o tabuleiro e as peças aplicando a jogada.
    - criarTabuleiro: Responsável por criar e popular o tabuleiro.
    - criarPecas: Esse método cria as 16 peças de ambas as cores.
    - fazerJogada: Responsável por realizar a jogada feita pelo jogador, verificando se é o turno dele, se a peça movida pertence a ele, verifica o movimento, se coloca ele em um xeque e também é responsável por fazer a troca do turno.

    - EmJogo: Um método simples para tirar o status de xeque e mudar para ‘j’ que significa “Em Jogo”.
    - print: Esse método imprime o tabuleiro através do método imprimeTabuleiro.
    - getTurno: Responsável por retornar de quem é a vez de fazer a jogada.
    - trocaTurno: Realiza a mudança do turno.
    - xeque: Coloca um jogador em xeque e muda o status do jogo para “x” determinando o xeque.
    - xequeMate: Coloca um jogador em Xeque mate, colocando o status em “X” e determina o vencedor.
    - getStatus: Apenas retorna o status da partida.
    - getNomPlayer1 ou getNomPlayer2: Retorna o nome do jogador.
    - getCorPlayer1 ou getCorPlayer2: Retorna a cor pertencente ao jogador.
    - comecarJogo: Inicia um jogo e apenas finaliza quando o status for mudado para “X” (xeque-mate), esse método faz as perguntas de jogadas e informa a situação da partida a cada turno.
- **Gerenciador:** É a Classe main, responsável por iniciar o programa e pedir as informações para os jogadores (Nome do jogo, cor da peça) e perguntar a jogada que ele deseja realizar.

Figura 1 - Diagrama de classe atualizado

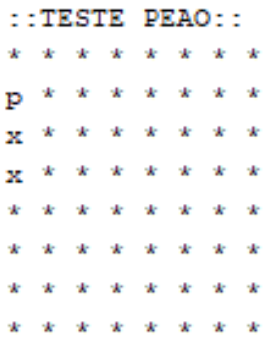


3 - Testes

Teste 1: Peão

- Objetivo do teste: checar os possíveis movimentos do peão e verificar sua cor
- Objeto: Peao.java
- Método testado: Peao.checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)
    - Valores dos parâmetros: checaMovimento(2, ‘a’, i, j)
    - Valor de retorno: true caso movimento seja válido
    - Letra da peça minúscula significa que sua cor é branca.

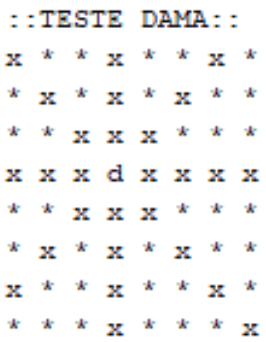
Figura 2 - Tela da classe Peão (sendo “*” movimentos inválidos, “x” movimentos possíveis e “p” posição atual da peça com sua inicial)



Teste 2: Dama

- Objetivo do teste: checar os possíveis movimentos do dama e verificar sua cor
- Objeto: Dama.java
- Método testado: Dama.checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)
    - Valores dos parâmetros: checaMovimento(4, ‘d’, i, j)
    - Valor de retorno: true caso movimento seja válido
    - Letra da peça minúscula significa que sua cor é branca.

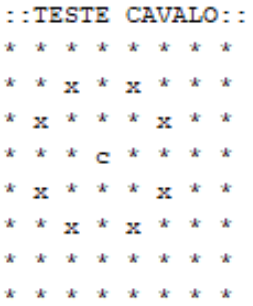
Figura 3 - Tela da classe Dama (sendo "*" movimentos inválidos, "x" movimentos possíveis e "d" posição atual da peça com sua inicial)



Teste 3: Cavalo

- Objetivo do teste: checar os possíveis movimentos do cavalo e verificar sua cor
- Objeto: Cavalo.java
- Método testado: Cavalo.checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)
    - Valores dos parâmetros: checaMovimento(4, 'd', i, j)
    - Valor de retorno: true caso movimento seja válido
    - Letra da peça minúscula significa que sua cor é branca.

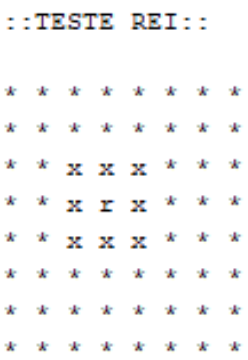
Figura 4 - Tela da classe Cavalo (sendo "*" movimentos inválidos, "x" movimentos possíveis e "c" posição atual da peça com sua inicial)



Teste 4: Rei

- Objetivo do teste: checar os possíveis movimentos do rei e verificar sua cor
- Objeto: Rei.java
- Método testado: Rei.checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)
    - Valores dos parâmetros: checaMovimento(4, 'd', i, j)
    - Valor de retorno: true caso movimento seja válido
    - Letra da peça minúscula significa que sua cor é branca.

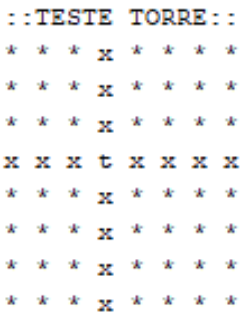
Figura 5 - Tela da classe Rei (sendo "*" movimentos inválidos, "x" movimentos possíveis e "r" posição atual da peça com sua inicial)



Teste 5: Torre

- Objetivo do teste: checar os possíveis movimentos da torre e verificar sua cor
- Objeto: Torre.java
- Método testado: Torre.checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)
    - Valores dos parâmetros: checaMovimento(4, 'd', i, j)
    - Valor de retorno: true caso movimento seja válido
    - Letra da peça minúscula significa que sua cor é branca.

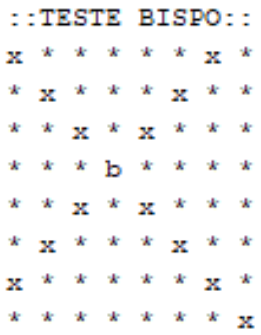
Figura 6 - Tela da classe Torre (sendo "*" movimentos inválidos, "x" movimentos possíveis e "t" posição atual da peça com sua inicial)



Teste 6: Bispo

- Objetivo do teste: checar os possíveis movimentos do bispo e verificar sua cor
- Objeto: Bispo.java
- Método testado: Bispo.checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)
    - Valores dos parâmetros: checaMovimento(4, 'd', i, j)
    - Valor de retorno: true caso movimento seja válido
    - Letra da peça minúscula significa que sua cor é branca.

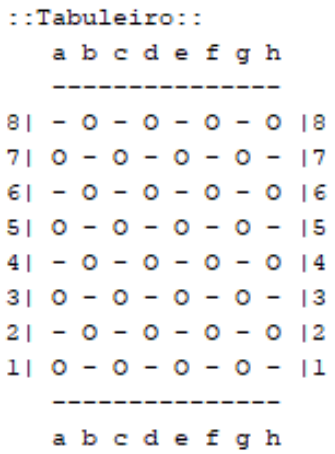
Figura 7 - Tela da classe Bispo (sendo "*" movimentos inválidos, "x" movimentos possíveis e "b" posição atual da peça com sua inicial)



Teste 7: Tabuleiro

- Objetivo do teste: checar se o Tabuleiro foi gerado corretamente
- Objeto: Tabuleiro.java
- Método testado: Tabuleiro.imprimeTabuleiro()
    - Valores dos parâmetros: não recebe parâmetros!
    - Valor de retorno: retorna o desenho do tabuleiro
    - Sendo "O" a casa preta e "-" a casa branca.

Figura 8 - Tela da classe Tabuleiro



Teste 8: Tabuleiro

- Objetivo do teste: checar o atributo validaMovimento()
- Objeto: Tabuleiro.java
- Método testado: Tabuleiro.validaMovimento(int linhaDestino, char colunaDestino, int linhaOrigem, char colunaOrigem)
    - Valores dos parâmetros: validaMovimento(10, 'z', 8, 's')
    - Valor de retorno: retorna falso se for um movimento fora do tabuleiro!

Figura 9 - Tela da classe Tabuleiro (método validaMovimento)

```
TESTE VALIDAR MOVIMENTO (Movimento fora do tabuleiro)
false
------------
```

Teste 9: Jogo e Jogador

- Objetivo do teste: checar os atributos pertencentes a classe jogo e jogador
- Objeto: Jogo.java e Jogador.java
- Métodos testados: Jogo.getPlayer1().getNome()
    - Valores dos parâmetros: não recebe parâmetros!
    - Valor de retorno: retorna nome do jogador
- Métodos testados: Jogo.getPlayer1().isPreto()
    - Valores dos parâmetros: não recebe parâmetros!
    - Valor de retorno: caso true, a peça do jogador 1 é preta!
- Métodos testados: Jogo.getTurno()
    - Valores dos parâmetros: não recebe parâmetros!
    - Valor de retorno: caso true, é a vez do jogador 1!
- Métodos testados: Jogo.getStatus()
    - Valores dos parâmetros: não recebe parâmetros!
    - Valor de retorno: d = jogo ocorrendo // e = erro!

Figura 10 - Tela da classe Jogo (métodos getPlayer1().getNome(), getPlayer1().isPreto(), getTurno() e getStatus() **)**

```
Nome jogador 1: Bernas
Cor Preto: false
Vez do jogador 1: true
Status do jogo: d
------------
Nome jogador 2: Motorista
Cor Preto: true
```

Teste 10: Xeque

- Objetivo do teste: checar jogadas de estado “Xeque”
- Jogo.java
- Métodos testados: Jogo.fazerJogada(), Jogo.xeque(). Jogo.xequeMate()

Figura 11 - Tela de Teste: Xeque 1

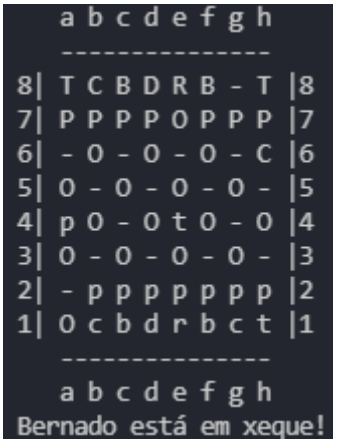


Figura 12 - Tela de Teste: Xeque 2

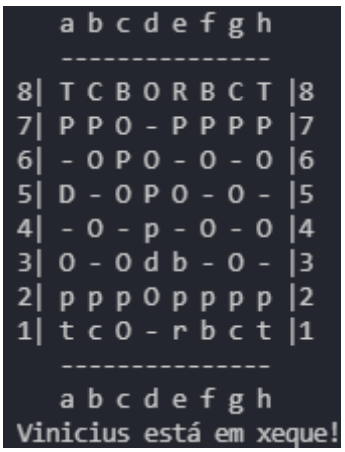
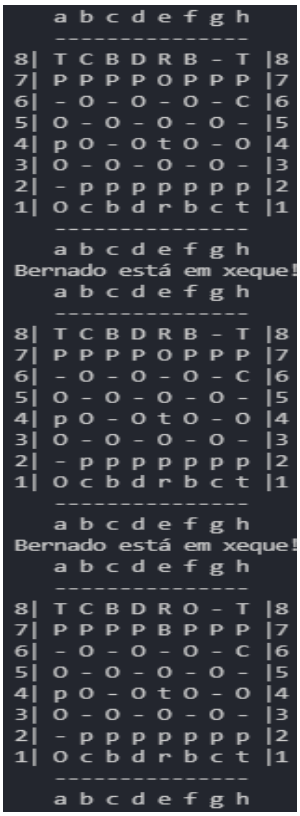


Figura 13 - Tela de Teste: Fuga do Xeque

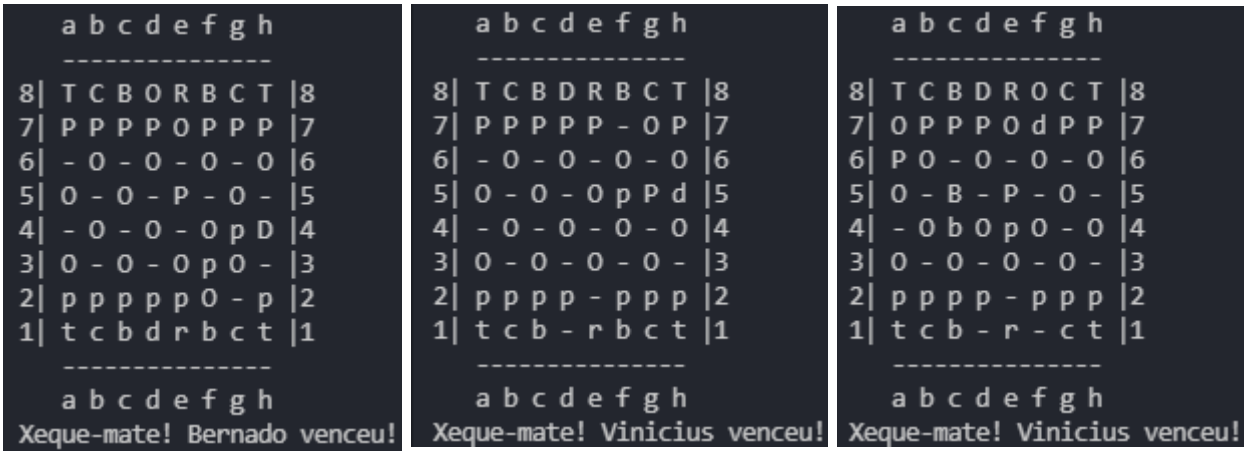


Na figura 11 e 12 vemos duas jogadas em que são colocadas o jogador “Bernado” em estado de Xeque. Já na figura 13 vemos movimentos que são válidos porém que ainda deixa o jogador em estado de xeque, sendo assim invalidando o movimento, na última parte da figura vemos um movimento válido em que o jogador sai do estado de xeque

Teste 11: Xeque-Mate

- Objetivo do teste: checar jogadas de estado “Xeque-Mate”
- Jogo.java
- Métodos testados: Jogo.fazerJogada(), Jogo.xeque(). Jogo.xequeMate()

Figura 14 - Tela de Teste: Xeque-Mate



Na primeira parte da figura 14 vemos uma jogada chamada “Mate do Louco” é o xeque-mate mais rápido possível no xadrez, com o segundo lance das peças pretas.

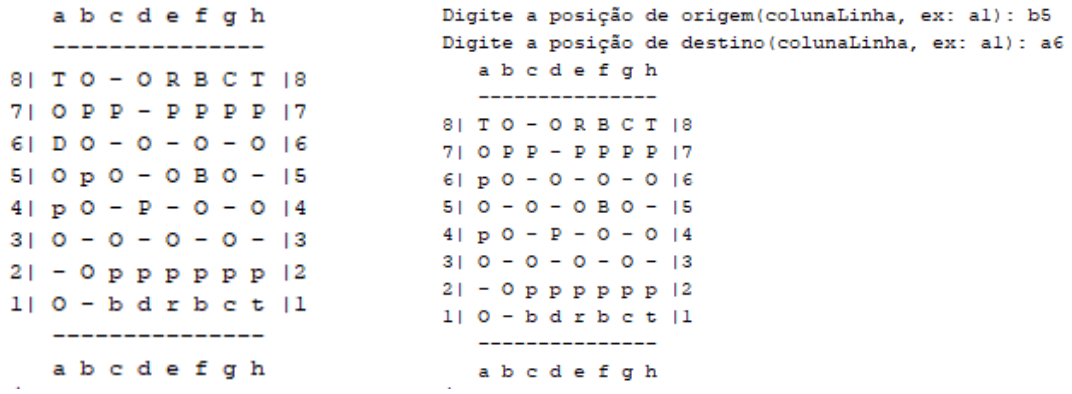
Na segunda parte da figura vemos um xeque-mate com as peças brancas, essa jogada é muito difícil de ocorrer.

E por último temos um dos xeque-mates mais famosos, o "pastorzinho" feito por uma série de movimentos.

Teste 12: Movimentação e eliminação de peças

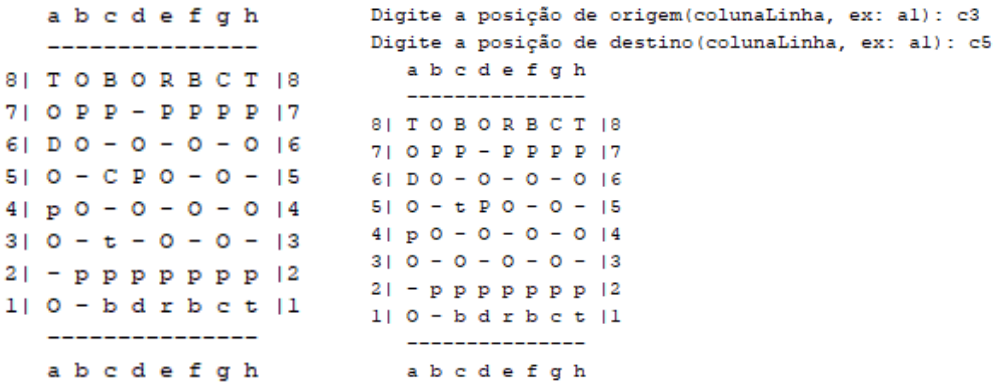
- Objetivo do teste: Testar a movimentação e eliminação das peças
- Jogo.java
- Métodos testados: Jogo.fazerJogada()

Figura 15 - Tela de Teste: Peão elimina Dama



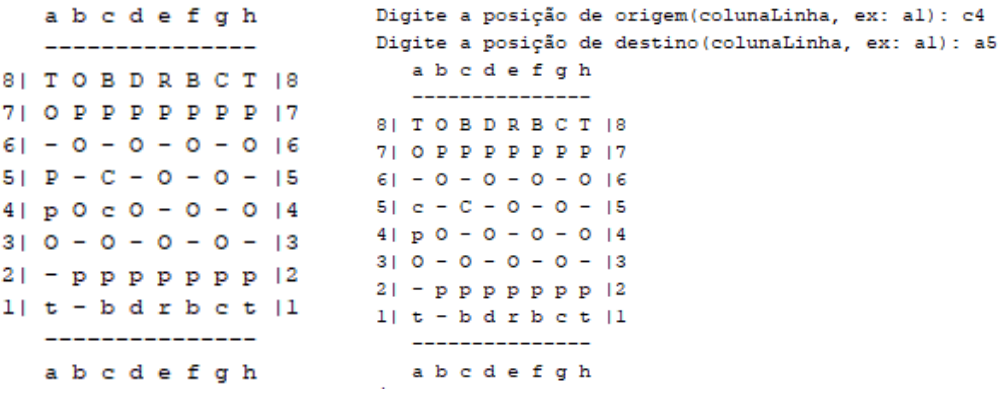
Na figura 15 vemos o peão de b5 eliminando a dama em a6 através do seu movimento na diagonal.

Figura 16 - Tela de Teste: Torre elimina Cavalo



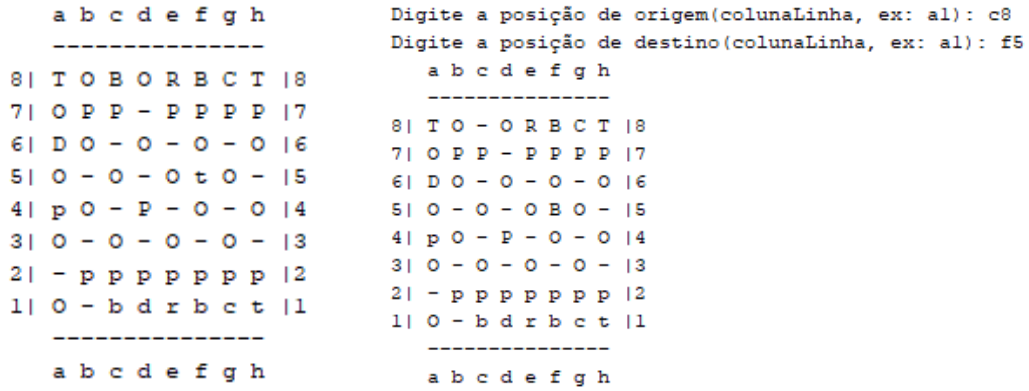
A figura 16 mostra que a torre de c3 elimina o cavalo em c5 através do seu movimento na vertical.

Figura 17 - Tela de Teste: Cavalo elimina Peão



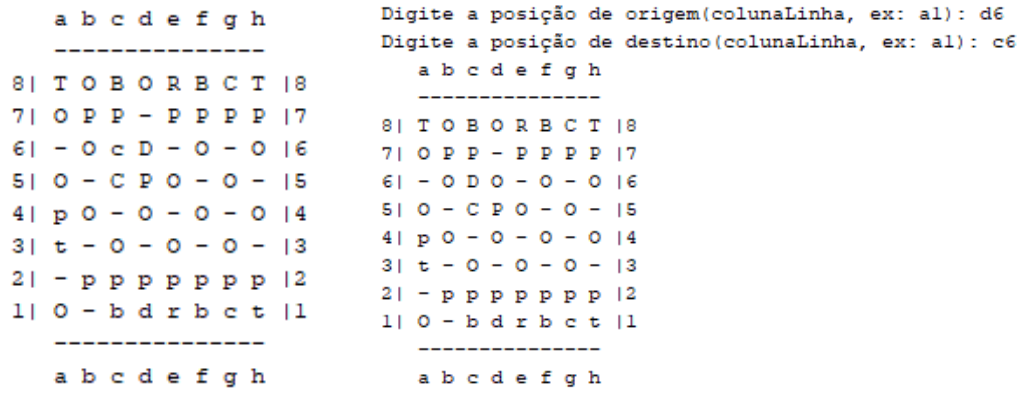
Já na figura 17 o cavalo de c4 elimina o peão de a5 com seus movimentos em “L” pulando qualquer tipo de obstáculo.

Figura 18 - Tela de Teste: Bispo elimina Torre



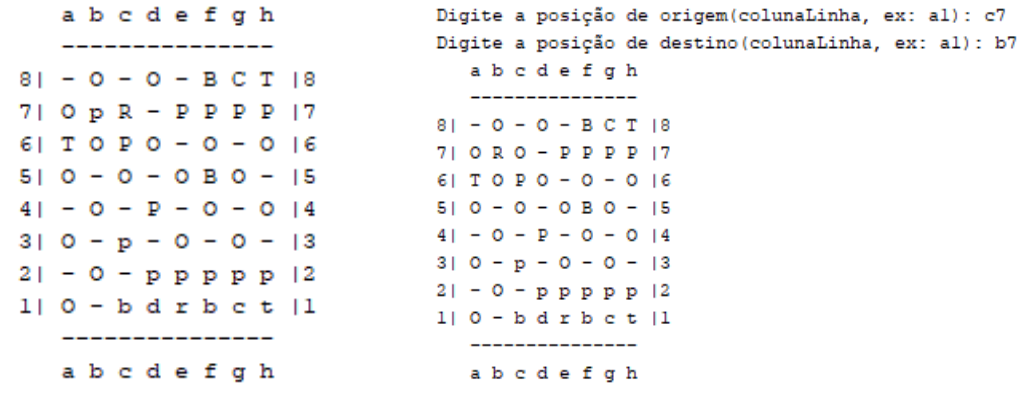
A figura 18 o bispo de c8 eliminando a torre em f5 com seu movimento em diagonal.

Figura 19 - Tela de Teste: Dama elimina Cavalo



Na figura 19 temos a dama eliminando o cavalo de d6 para c6.

Figura 20 - Tela de Teste: Dama elimina Cavalo



Na figura 20 temos o rei eliminando um peão de c7 para b7 sem colocar ele em xeque.

Teste 13: Gerenciador

- Objetivo do teste: Testar a main Gerenciador
- Gerenciador.java
- Métodos testados: Jogo.comecarJogo, Jogo.Jogo()

Figura 21 - Tela de Teste: Gerenciador

```
::: XADREZ :::
Selecione uma das opções abaixo:
1 - Iniciar novo jogo
2 - Carregar jogo
3 - Sair
1
Digite o nome do jogo (sem espaços):
Exemplo: 'Jogo1'
Jogo1
Digite o nome do Jogador 1:
Vinicius
Escolha a cor das Peças
         1- Brancas
         2- Pretas
1
Digite o nome do Jogador 2:
Bernardo
- - - - - - - - - - -
```

Na figura acima vemos as informações que o sistema pede, e como ele é passado através do usuário, sendo guardado nas variáveis em que vai ser usada para criar o construtor de Jogo.

4 - Instruções para compilação.

Com a pasta do projeto aberta (Projeto 2), abra o terminal no local da pasta e compile o programa utilizando o comando:

javac -encoding UTF-8 .\Xadrez*.java

Com o programa compilado, basta executá-lo utilizando o seguinte comando:

java Xadrez.Gerenciador

5 - Interface do Jogo e uso do programa

Após ser executado, o programa pergunta se o usuário gostaria de iniciar um novo jogo, carregar um jogo da memória ou sair do programa (Figura 22).

Figura 22 - Gerenciador

```
::: XADREZ :::
Selecione uma das opções abaixo:
1 - Iniciar novo jogo
2 - Carregar jogo
3 - Sair
```

Ao se escolher a primeira opção, que inicia um novo jogo, o programa pedirá o nome do jogo (que é o nome do arquivo em que o jogo será salvo), o nome do primeiro jogador, sua respectiva cor e o nome do segundo jogador (Figura 23).

Figura 23 - Informações para iniciar o jogo

```
Digite o nome do jogo (sem espaços):
Exemplo: 'Jogo1'
Jogo1
Digite o nome do Jogador 1:
Vinicius
Escolha a cor das Peças
         1- Brancas
         2- Pretas
1
Digite o nome do Jogador 2:
Bernardo
- - - - - - - - - - -
```

. Logo em seguida, o programa começará o jogo imprimindo o tabuleiro na tela e pedindo para que o jogador de cor branca escolha seu movimento (primeiro a posição de origem e em seguida a posição de destino, ambos no formato “a1”).

Figura 24 - Primeira rodada

```
O jogo começou!
   a b c d e f g h
   ---------------
8| T C B D R B C T |8
7| P P P P P P P P |7
6| - O - O - O - O |6
5| O - O - O - O - |5
4| - O - O - O - O |4
3| O - O - O - O - |3
2| p p p p p p p p |2
1| t c b d r b c t |1
   ---------------
   a b c d e f g h
É o turno do jogador1 (Vinicius)
Digite a posição de origem(colunaLinha, ex: a1):
```

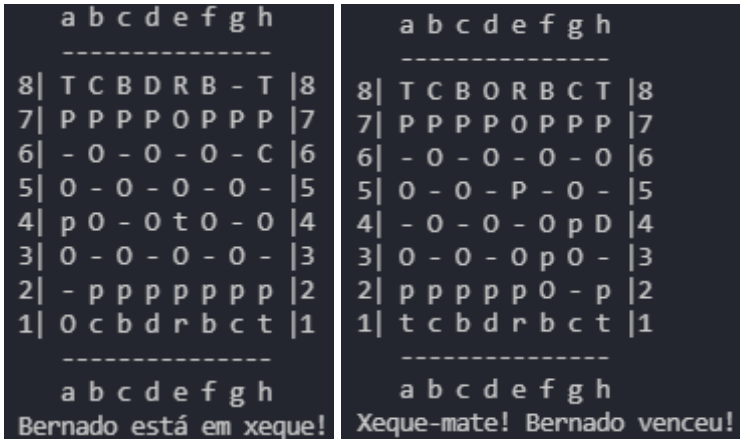
Caso o movimento seja inválido o programa pedirá novamente.

Figura 25 - Movimento Invalído

```
Digite a posição de origem(colunaLinha, ex: a1): a2
Digite a posição de destino(colunaLinha, ex: a1): a5
Movimento inválido!
Digite a posição de origem(colunaLinha, ex: a1):
```

O programa informa quando um jogador entrar em estado de xeque e finaliza quando um jogador der xeque-mate em outro imprimindo na tela uma mensagem com o nome do jogador vencedor.

Figura 26 - Xeque e Xeque-Mate



A função de carregar o jogo, por fazer parte do projeto 3, ainda não foi implementada.

6 - Condições que podem causar erro

Como nessa etapa não foi implementada nenhuma tratativa de erro com os try catch, no Gerenciador ao colocar tipos não correspondentes no scanner o programa irá apresentar falha de execução impossibilitando a continuação, como na imagem abaixo:

Figura 27 - Erro de execução

```
::: XADREZ :::
Selecione uma das opções abaixo:
1 - Iniciar novo jogo
2 - Carregar jogo
3 - Sair
Teste
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:864)
        at java.util.Scanner.next(Scanner.java:1485)
        at java.util.Scanner.nextInt(Scanner.java:2117)
        at java.util.Scanner.nextInt(Scanner.java:2076)
        at projeto.Gerenciador.main(Gerenciador.java:22)
C:\Users\Vinicius\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
FALHA NA CONSTRUÇÃO (tempo total: 4 segundos)
```

Esse erro ocorreu porque o scanner esperava receber um valor do tipo int, porém o valor passado foi uma String. Esses problemas serão tratados no projeto 3.