Universidade Federal de São Carlos - Campus Sorocaba

Ciência da Computação

Programação Orientada a Objetos

Jogo de Xadrez

Fase 2 e 3 - Data de entrega: 08/01/2021

Prof. Dra. Katti Faceli

Nome:Felipe Augusto Vale Amorim RA: 743530

Sorocaba / SP

2020

Sumário

1. Introdução

Foi proposto pela Professora Dra. Katti Faceli, docente da disciplina de Programação Orientada a Objetos, o trabalho responsável pelo gerenciamento de uma partida de Xadrez. A proposta seria desenvolver e implementar uma solução computacional em Java que seja responsável pelo controle de todas as atividades e movimentos que acontecem em um jogo de xadrez, de acordo com as regras para cada peça.

PALAVRAS CHAVES: Xadrez, Peça, Aplicação

2. Descrição das Classes

Nesta seção é apresentada a descrição de todas as classes do projeto.

2.1 Classe Gerenciador

A classe *Gerenciador* é responsável pela impressão de um menu para que o usuário possa informar qual será a primeira tarefa a ser executada pela aplicação. As opções são iniciar um novo jogo, carregar um jogo salvo ou sair. Aqui também é criado o objeto de tipo *Jogo* com seus respectivos jogadores.

Possui os seguintes atributos:

- não possui atributos.

Possui os seguintes métodos:

- main(): cria um menu selecionável ao usuário para informar qual será a primeira tarefa a ser executada pela aplicação.

2.2 Classe Jogo

A classe *Jogo* é responsável por iniciar a partida de Xadrez e gerenciar a partida.

Possui os seguintes atributos:

- tabuleiro: indica o tabuleiro da partida;
- j1: objeto jogador;
- j2: objeto jogador;

- estado: indica o estado da partida, podendo ser “Em andamento”, “Xeque” ou “Xeque-Mate”;
- vezJogador: indica de qual jogador é a vez, podendo receber 0 (branco) ou 1 (preto);
- pP: vetor de objeto de peças pretas;
- pB: vetor de objeto de peças brancas;

Possui os seguintes métodos:

- Jogo(): construtores responsáveis por criar o objeto de tipo *Jogo.* Existem dois construtores, um é responsável por iniciar um novo jogo e o outro é responsável por carregar um jogo já salvo;
- getEstado(): retorna estado atual da partida;
- setEstado(): atualiza estado da partida;
- menu(): responsável por exibir um menu ao usuário para que indica qual será o próximo passo a ser seguido pela aplicação. As opções são: realizar um movimento novo, salvar o jogo e sair;
- rodada(): responsável por cuidar das rodadas do Xadrez. Ela recebe os dados informados pelo usuário e envia para as checagens posteriores;
- xequeMate(): responsável por verificar se o estado do jogo está em Xeque Mate;
- salvar(): responsável por salvar todas as informações da partida em um arquivo “.txt”. Recebe do usuário o nome do arquivo a ser salvo.

2.3 Classe Jogador

A classe *Jogador* é responsável por armazenar as informações dos dois jogadores da partida.

Possui os seguintes atributos:

- nome: indica o nome do jogador;
- p: indica o conjunto de peças do jogador;
- cor: indica a cor das peças do jogador.

Possui os seguintes métodos:

- Jogador(): construtor responsável por criar o objeto *Jogador.*
- getCor(): responsável por retornar qual é a cor das peças do jogador;
- getNome(): responsável por retornar o nome do jogador.

2.4 Classe Tabuleiro

A classe *Tabuleiro* é responsável por estabelecer as configurações iniciais do tabuleiro na partida e pela manutenção e checagem das peças. Também salva as novas coordenadas da peça quando o movimento é válido.

Possui os seguintes atributos:

- mesa: indica as 64 posições do tabuleiro;
- linha: indica as linhas que compõem o tabuleiro (1 - 8);
- coluna: indica as colunas que compõem o tabuleiro (A - H).

Possui os seguintes métodos:

- Tabuleiro(): construtor responsável por criar o objeto de tipo *Tabuleiro*. Aqui é feito as configurações iniciais do tabuleiro, antes da partida começar;
- getPosicao(): responsável por retornar o objeto de tipo *Posição* dentro do tabuleiro, de acordo com as coordenadas informadas;
- desenho(): responsável por imprimir em tela as configurações atuais do tabuleiro;
- checaMovimento(): responsável por verificar a movimentação da peça dentro do tabuleiro e informar se a movimentação é válida ao usuário;
- caminhoPeca(): responsável por verificar se o movimento do usuário é inválido por possuir peças no meio do caminho;
- novaPosicao(): responsável por remover a peça do jogo e estabelecer as novas coordenadas de origem e de destino.

2.5 Classe Posição

A classe *Posição* é responsável por armazenar as informações de uma posição dentro do tabuleiro.

Possui os seguintes atributos:

- cor: indica se a posição é branca ou branca;
- linha: indica a coordenada linha da posição;
- coluna: indica a coordenada coluna da posição;
- ocupante: indica qual peça está ocupando a posição, podendo ser vazia.

Possui os seguintes métodos:

- Posicao(): construtor responsável por criar o objeto *Posição.*
- isCor(): responsável por retornar qual é a cor da posição no tabuleiro;
- getOcupante(): retorna a peça que está na posição;
- setOcupante(): responsável por definir a peça que ocupa a posição no tabuleiro.

2.6 Classe Peça

A classe *Peça* é abstrata. Ela é responsável pela interface com as classes das peças específicas.

Possui os seguintes atributos:

- cor: indica se a peça é branca ou preta;
- inGame: indica se a peça está em jogo ou não.

Possui os seguintes métodos:

- Peca(): construtor da classe Mãe das peças.
- getCor(): responsável por retornar qual é a cor da peça;
- isInGame(): retorna se a peça está em jogo;
- setInGame(): responsável por atualizar o estado da peça na partida;
- desenho(): método abstrato. Serve de interface para as peças específicas. Desenha a peça no tabuleiro de acordo com sua cor.
- checaMovimento(): método abstrato. Serve de interface para as peças específicas. Realiza as verificações das peças de acordo com as regras do Xadrez.

2.7 Classe Peao

A classe *Peao* é responsável por construir a peça, fazer suas verificações para movimentação e retornar o desenho da peça que será inserida dentro do tabuleiro.

Possui os seguintes atributos:

- cor: indica se a peça é branca ou branca. Herda da classe mãe;
- inGame: indica se a peça está em jogo ou não. Herda da classe mãe;
- primeiraJogada: indica se é a primeira movimentação da peça ou não. Isso é necessário, pois o peão, em caso de primeira jogada, tem uma movimentação diferente da padrão.

Possui os seguintes métodos:

- Peao(): responsável por construir o objeto de tipo *Peao.* Usa também o construtor da classe mãe;
- desenho(): responsável por retornar o “desenho” (peça preta: letra maiúscula, peça branca: letra minúscula) da peça no tabuleiro ;
- checaMovimento(): responsável por verificar se o movimento da peça é válido de acordo com as regras do jogo de xadrez.

2.8 Classe Bispo

A classe *Bispo* é responsável por construir a peça, fazer suas verificações para movimentação e retornar o desenho da peça que será inserida dentro do tabuleiro.

Possui os seguintes atributos:

- cor: indica se a peça é branca ou branca. Herdam da classe mãe;
- inGame: indica se a peça está em jogo ou não. Herdam da classe mãe.

Possui os seguintes métodos:

- Bispo(): responsável por construir o objeto de tipo *Bispo.* Usa também o construtor da classe mãe;
- desenho(): responsável por retornar o “desenho” (peça preta: letra maiúscula, peça branca: letra minúscula) da peça no tabuleiro ;
- checaMovimento(): responsável por verificar se o movimento da peça é válido de acordo com as regras do jogo de xadrez.

2.9 Classe Torre

A classe *Torre* é responsável por construir a peça, fazer suas verificações para movimentação e retornar o desenho da peça que será inserida dentro do tabuleiro.

Possui os seguintes atributos:

- cor: indica se a peça é branca ou branca. Herdam da classe mãe;
- inGame: indica se a peça está em jogo ou não.

Possui os seguintes métodos:

- Torre(): responsável por construir o objeto de tipo *Torre.* Usa também o construtor da classe mãe;
- desenho(): responsável por retornar o “desenho” (peça preta: letra maiúscula, peça branca: letra minúscula) da peça no tabuleiro ;

- checaMovimento(): responsável por verificar se o movimento da peça é válido de acordo com as regras do jogo de xadrez.

2.10 Classe Cavalo

A classe *Cavalo* é responsável por construir a peça, fazer suas verificações para movimentação e retornar o desenho da peça que será inserida dentro do tabuleiro.

Possui os seguintes atributos:

- cor: indica se a peça é branca ou branca. Herdam da classe mãe;
- inGame: indica se a peça está em jogo ou não. Herdam da classe.

Possui os seguintes métodos:

- Cavalo(): responsável por construir o objeto de tipo *Cavalo.* Usa também o construtor da classe mãe;
- desenho(): responsável por retornar o “desenho” (peça preta: letra maiúscula, peça branca: letra minúscula) da peça no tabuleiro ;
- checaMovimento(): responsável por verificar se o movimento da peça é válido de acordo com as regras do jogo de xadrez.

2.11 Classe Dama

A classe *Dama* é responsável por construir a peça, fazer suas verificações para movimentação e retornar o desenho da peça que será inserida dentro do tabuleiro.

Possui os seguintes atributos:

- cor: indica se a peça é branca ou branca. Herdam da classe mãe;
- inGame: indica se a peça está em jogo ou não. Herdam da classe mãe.

Possui os seguintes métodos:

- Dama(): responsável por construir o objeto de tipo *Dama.* Usa também o construtor da classe mãe;
- desenho(): responsável por retornar o “desenho” (peça preta: letra maiúscula, peça branca: letra minúscula) da peça no tabuleiro;
- checaMovimento(): responsável por verificar se o movimento da peça é válido de acordo com as regras do jogo de xadrez.

2.12 Classe Rei

A classe *Rei* é responsável por construir a peça, fazer suas verificações para movimentação e retornar o desenho da peça que será inserida dentro do tabuleiro.

Possui os seguintes atributos:

- cor: indica se a peça é branca ou branca. Herdam da classe mãe;
- inGame: indica se a peça está em jogo ou não. Herdam da classe mãe;

Possui os seguintes métodos:

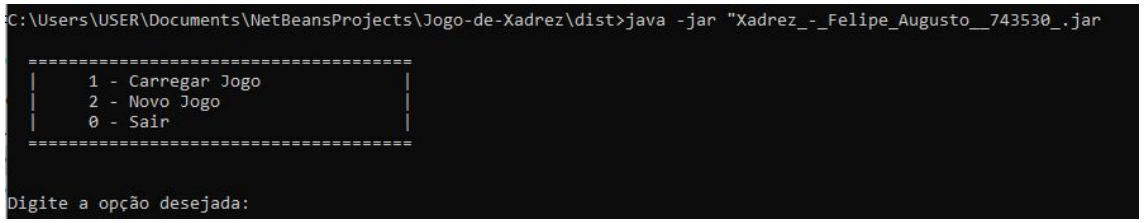
- Rei(): responsável por construir o objeto de tipo *Rei.* Usa também o construtor da classe mãe;
- getCor(): responsável por retornar a cor da peça;
- desenho(): responsável por retornar o “desenho” (peça preta: letra maiúscula, peça branca: letra minúscula) da peça no tabuleiro ;
- checaMovimento(): responsável por verificar se o movimento da peça é válido de acordo com as regras do jogo de xadrez.

3. Como compilar e executar?

Um documento “README.txt” no diretório do projeto possui instruções de execução.

Para que a aplicação seja executada, o usuário deve possuir o Java JRE instalado na máquina. A aplicação foi criada utilizando a IDE NetBeans IDE 8.2.

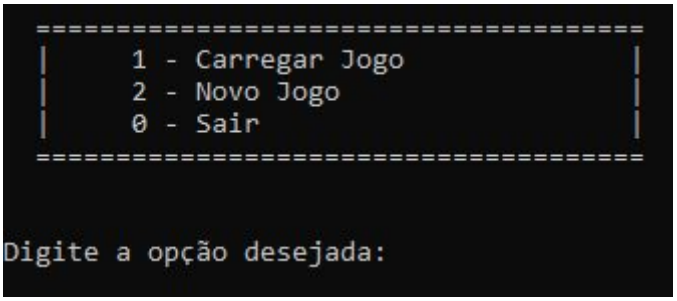
Para executá-la basta ir na pasta “dist” do projeto e executar o comando visto na imagem abaixo:



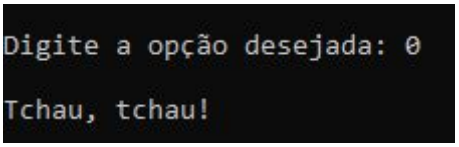
java -jar "Xadrez_-_Felipe_Augusto__743530_.jar"

4. Interface do Jogo (Tutorial de como jogar)

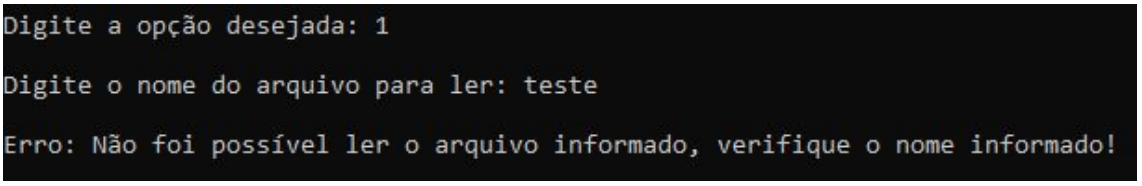
Ao executar a aplicação, é exibido um menu ao usuário para que ele possa informar qual será a primeira tarefa a ser efetuada. Este menu é visto na imagem abaixo:



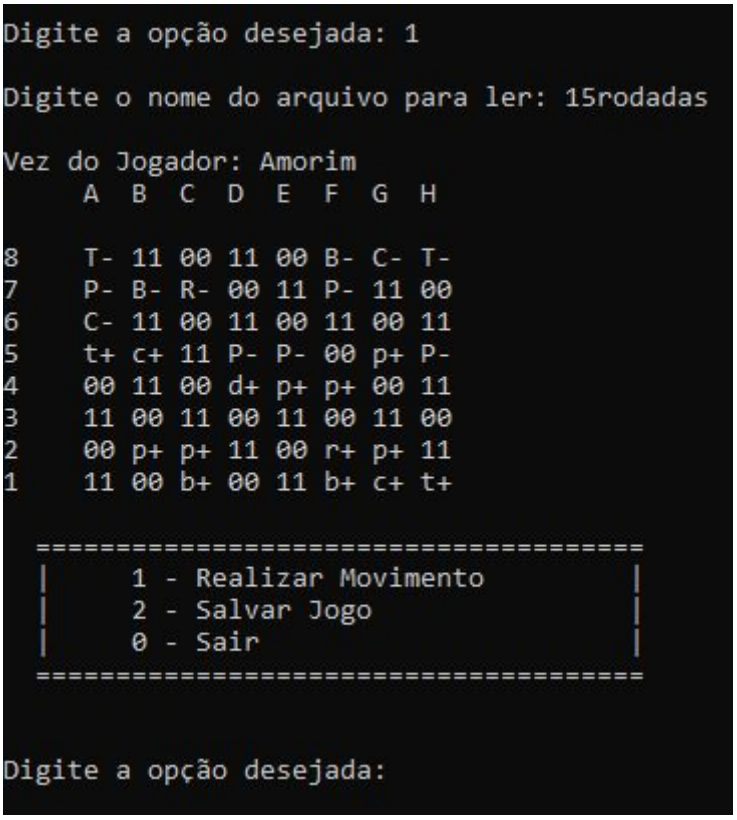
Ao inserir 0, a aplicação é finalizada e a aplicação retorna uma mensagem:



Ao inserir 1, a aplicação pede que seja inserido um nome de arquivo “.txt” para ser lido. Este arquivo deve ter sido criado anteriormente em alguma partida não finalizada. Caso seja um arquivo que não está na pasta do projeto, a seguinte mensagem é exibida:

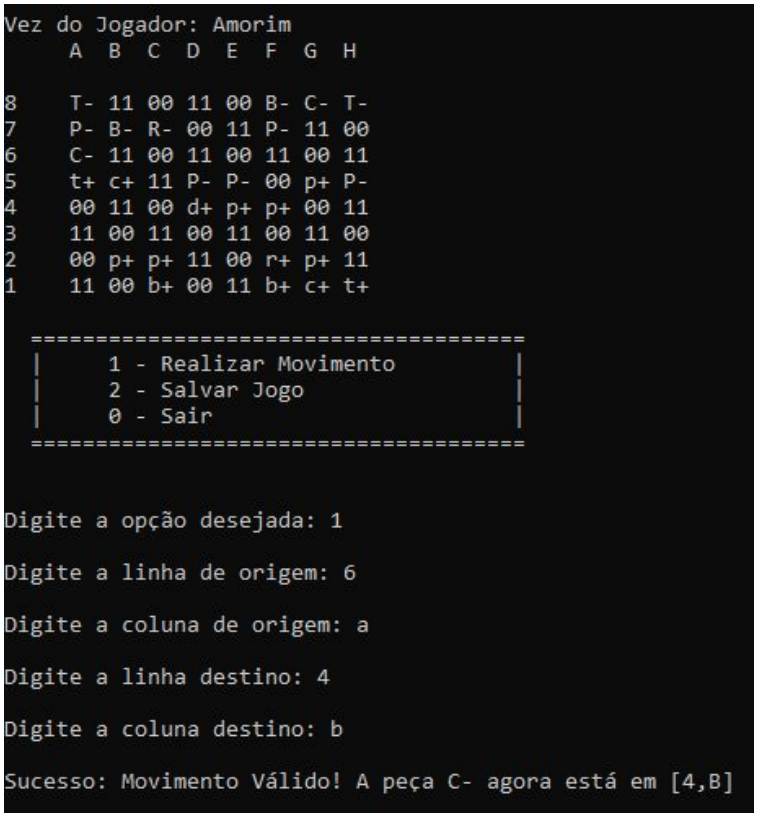


Caso o usuário digite um nome existente de arquivo, o estado salvo pelo jogo é carregado, com todas as peças que estavam na partida. Os nomes e o estado do jogo também são carregados. A figura abaixo mostra isso:

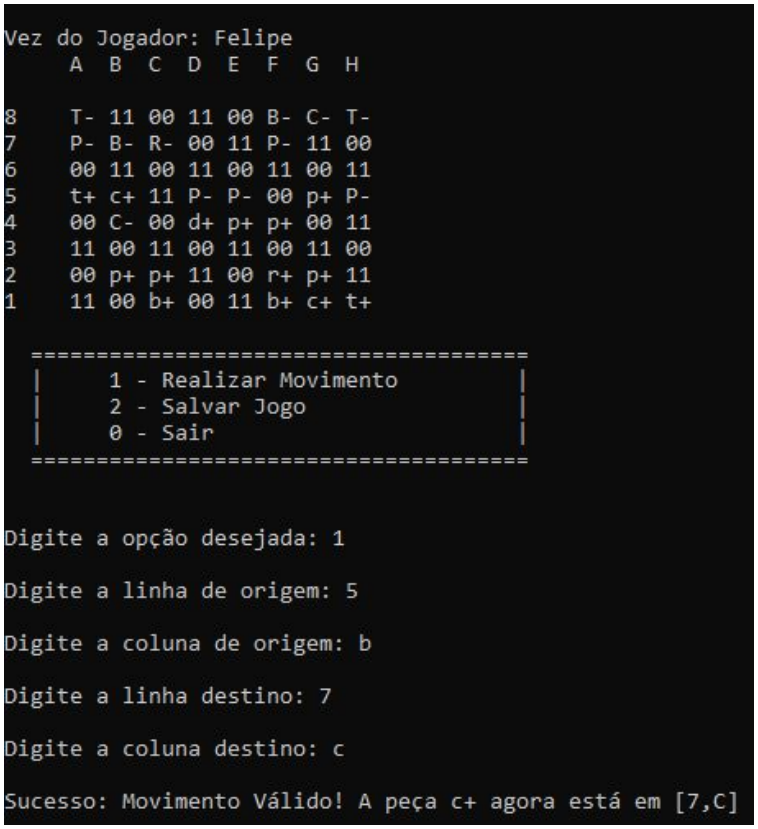


A partir desse momento, o jogo é retomado e pode seguir da última rodada salva.

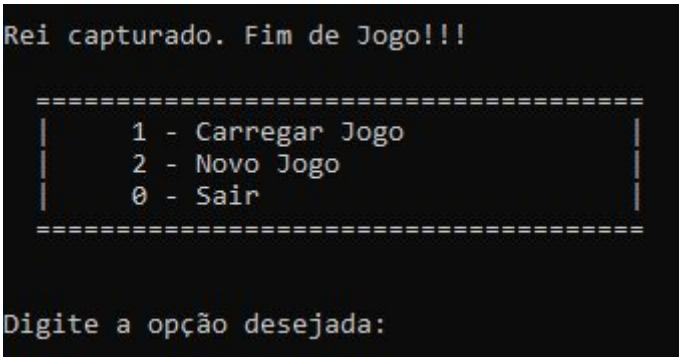
Ao clicar na opção 1 de realizar movimento, a aplicação espera que o usuário informe as coordenadas de origem e destino da peça. As coordenadas de linha aceitam apenas entradas de números, enquanto as coordenadas de coluna esperam um texto de entrada.



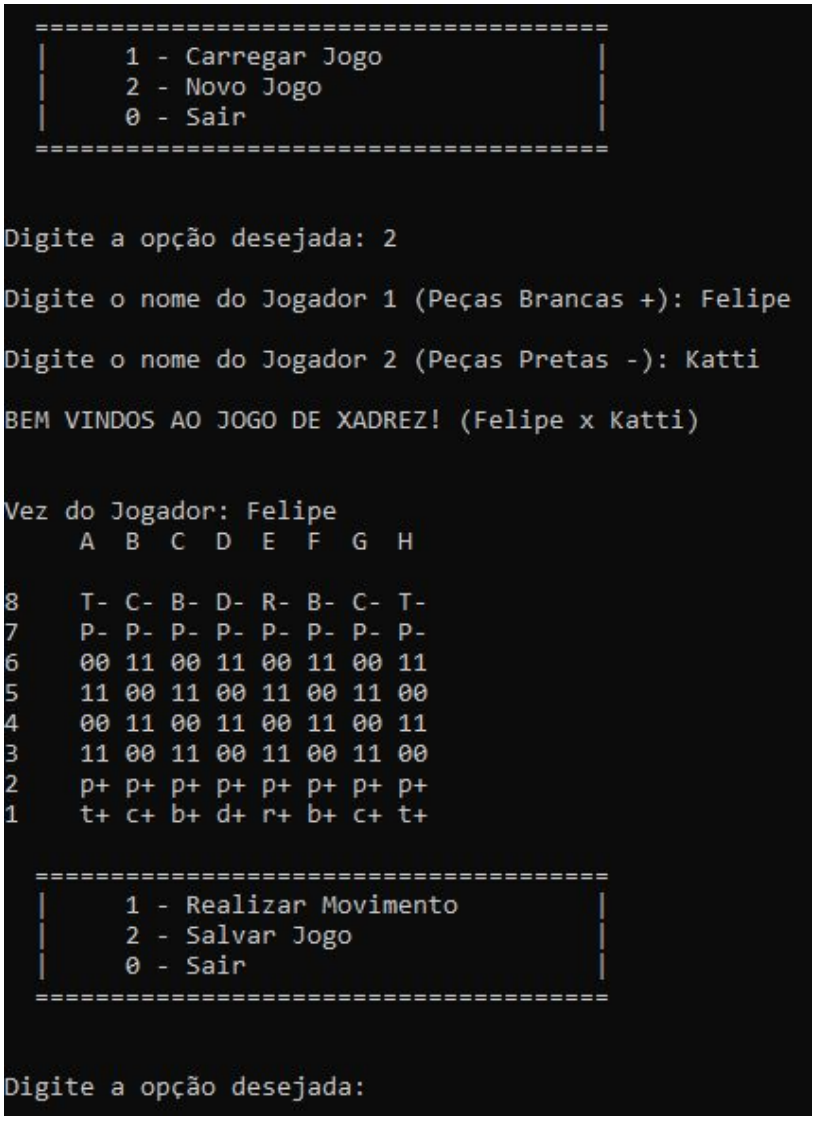
Ao realizar esta jogada, a aplicação encerra o turno do jogador e a vez é do outro jogador.



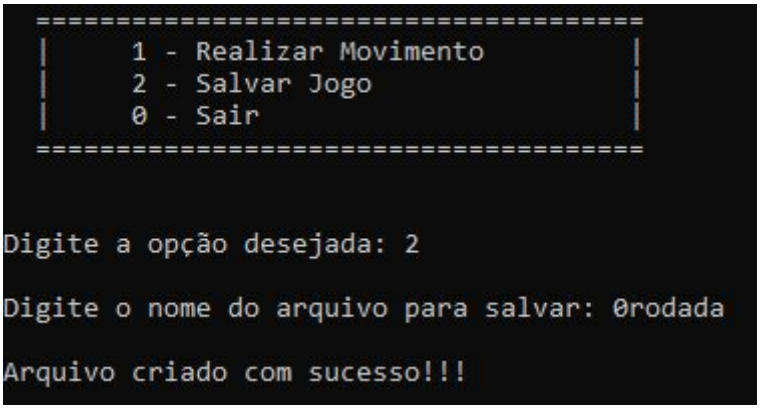
Na figura apresentada acima, aconteceu uma rodada onde o rei foi capturado e dessa maneira, a partida acabou. Ao término de uma partida, o menu inicial é apresentado:

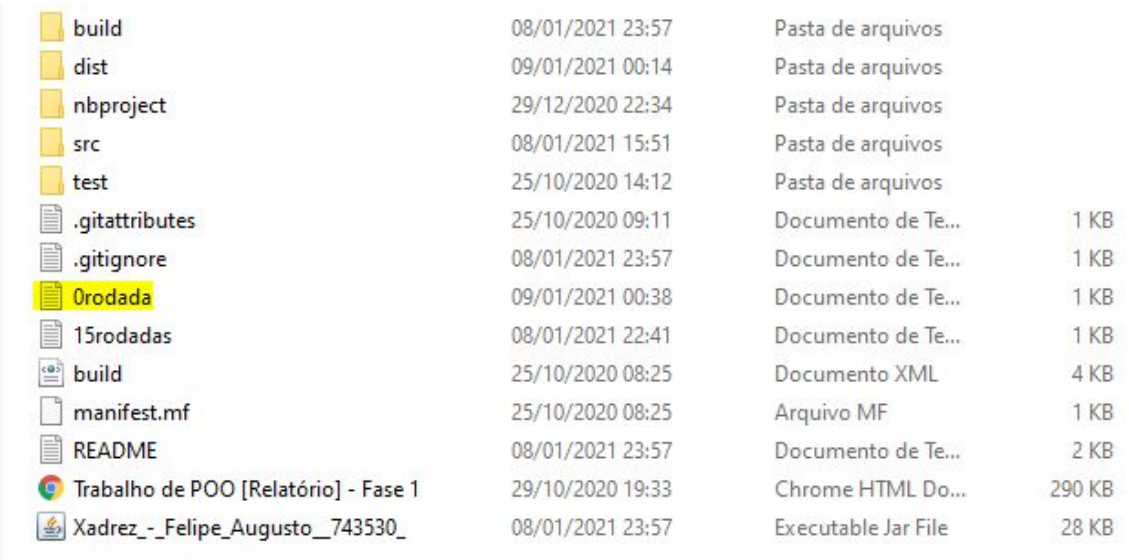


Caso a opção de novo jogo for selecionada, as entradas abaixo são exibidas e uma nova partida é iniciada:



Ao clicar na opção de salvar, uma entrada de texto é esperada para que o nome do arquivo seja informado. Caso o usuário informe um nome de arquivo que já existe na pasta do projeto, este arquivo é substituído pelo novo.





Por fim, é importante citar que aconteceram tratamentos de erro tanto no momento que o usuário informa uma entrada de dado, até o momento de leitura e escrita de arquivo. Isso foi feito para que mesmo que um erro acontecesse na aplicação, ela não deixasse de ser executada, tendo o erro previsto e tratado.

Caso haja algum problema ao executar a aplicação da maneira informada, o projeto também pode ser executado direto do NetBeans.

5. Limitações da Aplicação

Durante o desenvolvimento do projeto, pude notar algumas coisas que poderiam ser acrescentadas para deixar a partida de Xadrez mais parecida como é de verdade.

Alguns funcionalidades que notamos que poderiam ser adicionadas seriam:

- Promoção do Peão: neste momento, caso o Peão atinja o limite do tabuleiro, ele praticamente é descartado do jogo, pois o retorno dele não é possível;
- Roque;
- Xeque: a verificação de estado “Xeque” da partida não foi realizada;
- Xeque Mate: a correta verificação não foi realizada. Foi implementado o estado de “Xeque Mate” no momento que o Rei é capturado e a verificação não é feita ao término da jogada de um dos jogadores.

6. Conclusão

Neste trabalho abordei o Jogo de Xadrez e o desenvolvi, apresentando diversas características e conceitos da programação orientada a objetos e da linguagem Java.

Foi possível usar diversos conhecimentos obtidos com a disciplina, assim, notei algumas mudanças que eu poderia fazer (informadas no tópico anterior).

Apesar de não ter realizado o “Xeque” e o “Xeque Mate” da maneira 100% correta, acredito que o resultado final foi muito bom e acredito que tenha sido um trabalho muito importante para fixar os conceitos de POO.

Por fim acredito que o projeto foi excelente para meu conhecimento e aprofundamento no desenvolvimento da programação, uma vez que me permitiu aperfeiçoar competências de investigação, organização e criação e acredito que tenha sido possível cumprir com todos os objetivos que foram propostos, ganhando experiência e aprendizagem.