

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

JOGO DE XADREZ
PROJETO FINAL DA DISCIPLINA

Membro do grupo:
Madson Antonio de Jesus Santana

Curitiba,
Maio de 2021
MADSON ANTONIO DE JESUS SANTANA

JOGO DE XADREZ
PROJETO FINAL DA DISCIPLINA

Trabalho para a disciplina de Técnicas de programação, do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná, *campus* Curitiba.

1 INTRODUÇÃO

O projeto desenvolvido foi um jogo de xadrez, que usa o terminal como interface de interação com o usuário.

Feito de forma individual pelo aluno *Madson A J Santana (RA-2128039)*.

2 REGRAS DO JOGO

Como dito anteriormente, é um jogo de xadrez. Ao executar, é permitido iniciar uma nova partida ou continuar um já iniciada.

Dois jogadores jogam a partida, e as regras do jogo são as tradicionais regras do jogo de xadrez, o primeiro jogador movimenta sua peça, muda o turno, e assim o segundo jogador faz sua jogada. Cada tipo de peça movimentada tem suas regras de movimentação particulares.

Durante a troca de turno, é possível sair da partida e salvá-la para continuar a jogar posteriormente.

O jogo também tem a regra de xeque, que é quando um jogador coloca em risco o *Rei* do adversário, onde o jogador que está com o *Rei* em xeque é obrigado a tirá-lo dessa condição. A partir dessa regra chegamos ao ponto em que se encerra o jogo, o Xeque Mate.

2.1 Movimento das peças*

1. **Rei:** move-se para todas as direções pela vertical, horizontal ou diagonal, mas apenas uma casa por lance;
2. **Rainha:** é a peça mais poderosa do jogo, uma vez que seu movimento combina o da torre e o do bispo, ou seja, pode mover-se pelas colunas, fileiras e diagonal;
3. **Bispo:** move-se pela diagonal, sendo que nunca poderá mudar a cor das casas em que se encontra, uma vez que movendo-se em diagonal, não lhe é permitido passar para uma diagonal de outra cor;
4. **Cavalo:** movimenta-se sempre em "L", ou seja, duas casas para frente, para o lado ou para trás e uma para a esquerda ou direita. O cavalo é a única peça que pode pular sobre as outras, tanto as suas quanto as adversárias, como indo, por exemplo, desde a casa g1 para a casa f3 nos primeiro lances. Comumente se diz que o cavalo move-se "uma casa como torre e uma casa como bispo";
5. **Torre:** movimenta-se em direção reta pelas colunas ou fileiras;
6. **Peão:** movimenta-se apenas uma casa para frente e captura outros peões e peças na primeira casa diagonal superior. Caso uma peça ou peão fique na frente do peão, será impossível movê-lo. Somente se alguma peça adversária fique na sua diagonal acima, ele poderá capturá-la e mudar de coluna. No primeiro movimento de qualquer peão, ele poderá mover-se uma ou duas casas, a critério do enxadrista. Ao contrário das outras peças, o peão não pode mover-se para trás;

* Fonte: [Wikipedia](#)

2.2 Xeque mate - definição

1. O *Rei* que está em xeque não tem nenhuma possibilidade de se movimentar sem permanecer em risco;
2. Nenhuma de suas peças podem ser movimentadas para protegê-lo(o *Rei*)
3. Nenhuma de suas peças podem ser movimentadas para capturar alguma das peças adversárias que o ameaçam(o *Rei*).

3 DESCRIÇÃO DA MODELAGEM UML

1. Classes que herdam de **class Peca**:
 - a. Essas classes têm a missão de implementar as especificidades de cada peça, como regras de movimentações, verificação se o seu movimento não coloca o seu *Rei*, *Rei* esse que está no mesmo pacote de peças, peças do mesmo tipo(preto/branco). Também fazem o trabalho de armazenar estados de uma *Peca* durante o jogo, como por exemplo, seu símbolo, cor, para qual direção a peça se movimenta (no caso de um *Peão*, pois só anda para frente), se já executou a primeira jogada, entre outros dados.
2. Classes que herdam de **class Partida**:
 - a. Essas classes tem como missão especificar as regras da partida, assim podendo ter vários tipos de partida(com e sem tempo por turno, tempo máximo de jogo por jogador, partida de turno simples, etc...). Também orchestra o turno para decidir quem é o jogador a jogar, afinal, a informação sobre quem joga é uma informação da partida, ela quem precisa guardar esse dado, e os jogadores também são associados à partida.
3. Classe **class PecasPack**
 - a. Responsável por armazenar um conjunto de peças de uma mesma cor/estilo, que será “entregue” a um Jogador para que ele as use durante o jogo. Essa classe também é responsável, por exemplo, por alocar as peças em suas posições iniciais no tabuleiro, pois existe uma regra específica no Xadrez para que as peças brancas iniciam em determinado lado do tabuleiro e as pretas em outro lado, logo, responsabilidade desse conjunto de peça.
 - b. Essa classe também interliga as peças, já que elas se movimentam com algumas regras específicas de cada uma, mas também dependem, por exemplo, da posição do *Rei*, essa classe passa algumas infos para que uma peça que vá calcular os movimentos calcule de maneira correta.
4. Class **class Tabuleiro**
 - a. Responsável por armazenar todas as posições do possível para que o jogo ocorra de forma correta. Também gerencia os pacotes de peças, brancas e pretas, afinal todo tabuleiro acompanha 2 pacotes de peça e disponibiliza isso à 2 jogadores quando se inicia uma partida.
5. Class **class Posicao**
 - a. Entidade que armazena coordenadas relacionadas à matriz de posições. A Posição conhece a Peça que está atualmente na posição assim como a peça sabe onde ela se encontra no tabuleiro.

6. Classes que herdam de **class SqlConnection**:
 - a. A classe **SqlConnection** é responsável por abrir e fechar a conexão com o banco de dados para persistência dos dados. As classes derivadas desta especificam as conexões com as tabelas no banco SQL respectivas entidades. Exemplo: *PecaSql*: especifica e estabelece conexão para a tabela responsável por armazenar os dados de uma peça. E assim, respectivamente, outras entidades se conectam com suas tabelas.
7. Classes **DAO**
 - a. Responsável por fazer a intermediação, como um *Proxy*, com o método de persistência escolhido, podendo ser extensível ao infinito para poder persistir em qualquer lugar esses dados. Exemplo: um banco de dados local, arquivo JSON ou então algum armazenamento na nuvem, etc.

4 EXECUTANDO O CÓDIGO FONTE

Essa explicação também está contida no arquivo *README.md* que acompanha o código fonte.

4.1 Dependências

Execute num ambiente Linux, e garanta que tenha instalado, além do compilador *g++*, o lib *sqlite3*. Caso não tenha instalado o *sqlite3*, execute o seguinte comando*:

```
sudo apt install sqlite3 libsqlite3-dev
```

** Comando para instalar as dependências numa distro baseada no Ubuntu*

4.2 Compilar

Execute o seguinte comando:

```
g++ ./**/*.cpp -lsqlite3 -o ./main
```

4.3 Executar

Execute o seguinte comando:

```
./main
```

5 TABELA DE REQUISITOS

Número	Descrição	% de implementação
1	O jogo deverá poder ser jogado por dois jogadores simultaneamente, cada um identificado por pelo menos o seu nome.	100%
2	O jogo deverá possuir pelo menos 5 elementos e/ou personagens que possam ser definidos e operados polimorficamente a partir de uma definição genérica.	100%
3	O jogo deverá definir pelo menos 3 ambientes (fases, campos de jogo ou similares) diferentes.	33%
4	O jogo deverá definir um esquema de pontuação para os jogadores.	0% *
5	A relação estrutural entre os elementos/personagens e os ambientes deverá se dar por meio de pelo menos um dos relacionamentos entre classes/objetos definidos na OO, respeitando-se o seu significado conceitual.	100%
6	O jogo deverá permitir salvamento e recuperação posterior da sua situação atual por meio da escolha entre pelo menos 2 formatos de arquivo diferentes, implementados segundo o padrão de projeto DAO.	50%
7	O jogo deverá permitir salvamento e recuperação do ranking de melhores pontuações por meio da escolha entre pelo menos 2 formatos de arquivo diferentes, implementados segundo o padrão de projeto DAO.	0% *

** Não implementado pois não parecia ser aplicável no contexto do jogo de xadrez.*

*** Nota ao professor:**

Não sei se há a possibilidade de transferir o peso desses critérios de aceite para algum dos outros, caso não haja, seguimos como o planejado.

6 CONCLUSÕES FINAIS

Como descrito no início deste relatório, o trabalho foi feito de forma individual, então no que se refere à divisão de tarefas (modelagem, desenvolvimento e relatório), todas foram atribuídas a mim mesmo.

O primeiro passo que dei foi modelar a estrutura inicial que imaginei, e no decorrer do desenvolvimento foram aparecendo algumas alterações. A modelagem eu considero de extrema importância para o projeto, talvez por nunca ter feito uma assim, fiquei impressionado em como facilita o desenvolvimento, servindo como guia. Gostei bastante de fazer e vi de fato a aplicabilidade e importância.

Ao chegarmos no ponto da disciplina onde vimos padrões de projeto, acabei identificando que já estava aplicando um ou outro padrão. Vale ressaltar a implementação do *Factory Method* para o instanciamento das *Peças*. Inclusive foi um ponto muito legal da matéria. Junto com a modelagem, os padrões de projetos me deram uma visão muito maior sobre desenvolvimento de software, e que com certeza vai ser aprofundado durante o curso.

Em relação ao paradigma orientado a objetos, eu que trabalho programando mais alinhado com o paradigma funcional, achei extremamente interessante traçar os paralelos.

Acho que o trabalho num todo foi bastante interessante de se fazer, mesmo sozinho, creio que consegui gerenciar bem todos os aspectos do desenvolvimento, envolvendo a aplicação de forma coesa e desacoplada das classes, padrões de projetos e modelagem, gerenciamento do tempo, levando em conta todas as outras disciplinas e o meu trabalho profissional.

