UNIVERSIDADE FEDERAL DE SANTA CATARINA

KEVIN MONDADORI MATTOS
LUCAS MATHEUS DOS SANTOS
VITOR CITELLI CRIVELARO

TRABALHO FINAL DE PROGRAMAÇÃO III

PROJETO JOGO DE XADREZ

JOINVILLE, SC
2018

KEVIN MONDADORI MATTOS
LUCAS MATHEUS DOS SANTOS
VITOR CITELLI CRIVELARO

TRABALHO FINAL DE PROGRAMAÇÃO III

Relatório referente ao trabalho final vinculado à disciplina de Programação III na Universidade Federal de Santa Catarina sob tutela do professor Valdir Pedrinho.

JOINVILLE, SC
2018

RESUMO

O trabalho a seguir discorre sobre duas implementações de um jogo de xadrez, uma utilizando-se do conceito de polimorfismo e outra sem utilizar conceitos de herança e polimorfismo, nas duas pode-se iniciar o jogo a partir de um jogo anterior ou do início do jogo, além disso foram utilizadas diferentes formas de leitura para as jogadas nas duas implementações, ademais é discutido sobre como seria a implementação utilizando os conceitos de herança e comparação entre as três abordagens, incluindo suas vantagens e desvantagens.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Com origem na Índia por volta do século VI, o xadrez é um esporte de estratégia mundialmente conhecido, jogado por duas pessoas, encanta milhares de jogadores, a estimativa é de que 605 milhões de pessoas ao redor do mundo saibam jogar xadrez.

Este trabalho tem como propósito, discorrer sobre três abordagens de implementação de um jogo de xadrez, uma utilizando conceitos de polimorfismo, outra sem utilizar conceitos de metaprogramação e a última utilizando conceitos de herança sem polimorfismo, com foco na primeira e segunda aproximação.

Em um segundo momento, comparar estas implementações destacando suas vantagens e desvantagens e, por último, com base na comparação, decidir qual o melhor método a ser utilizado.

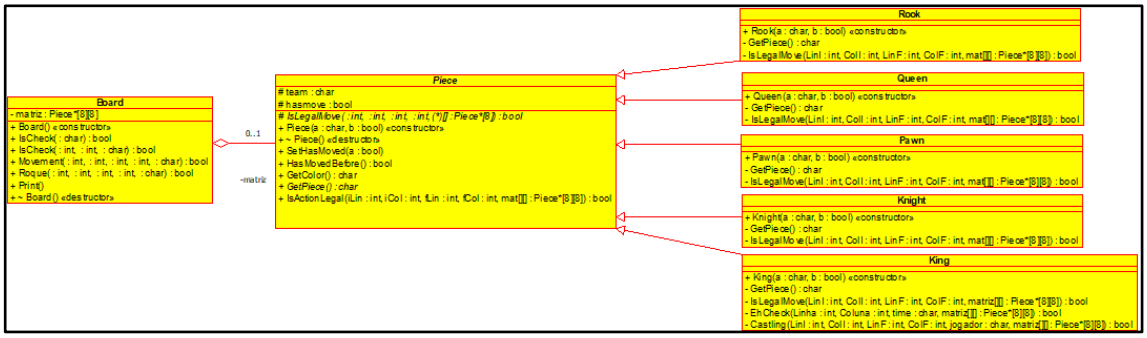
2 IMPLEMENTAÇÃO COM POLIMORFISMO

Nessa implementação, utiliza-se a classe *Board* para modelar o tabuleiro do jogo, esta classe contém os métodos para conferir se o jogo está em xeque, para movimentação e para imprimir o tabuleiro, além dos métodos construtor e destrutor, ademais a classe possui como atributo privado uma matriz 8x8 da classe abstrata *Piece*.

A classe *Piece* é classe base de todas as peças, esta possui métodos para conferir se a jogada pretendida é permitida, para saber de qual cor é a peça e para saber qual o tipo da peça, além dos métodos construtor e destrutor, além disso contém atributos para dizer de qual time é a peça e se a mesma já foi movida alguma vez no jogo. Cada peça possui os métodos construtor, métodos *GetPiece* para indicar qual o tipo da peça e métodos próprios de movimentação.

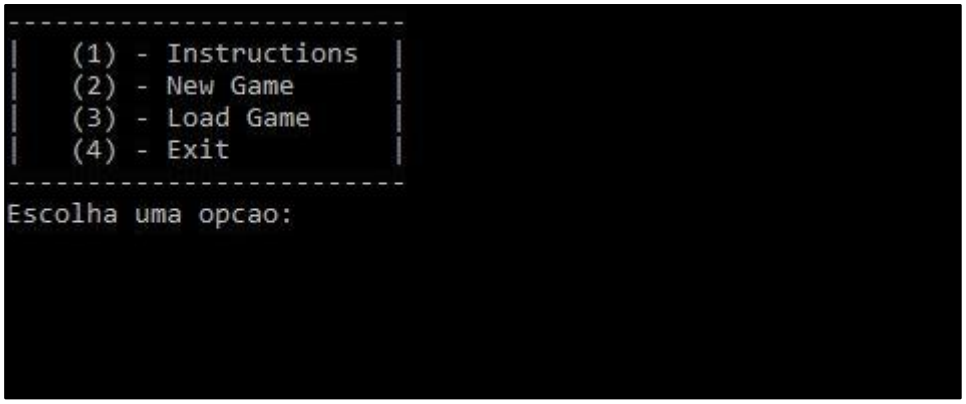
A relação entre tais classes pode ser visualizada na figura a seguir:

Figura 1 – UML de classes da implementação com polimorfismo



No começo de cada jogo, o usuário tem o direito de escolher entre instruções, começar um jogo novo, carregar um jogo ou sair do programa, como mostrado na figura a seguir:

Figura 2 - Menu inicial da implementação com polimorfismo



Em cada rodada, o jogador digita qual a posição de origem e a posição de destino, então o programa analisa se o jogador moveu uma peça que seja do seu time, se isto é confirmado, o programa confere se a posição de destino é válida, a movimentação de cada peça é explicada a seguir, se qualquer um destes dos passos não é obedecido, o jogador deve digitar outra posição inicial e final.

A movimentação para cada uma das peças embora diferente, segue o mesmo padrão, primeiro é verificada a posição de destino no tabuleiro, se é uma posição válida para movimento, ou seja, se está dentro do tabuleiro e não é ocupada por peças do mesmo time, caso esta primeira verificação seja validada, então é chamado o método de movimento para verificar se o movimento é válido, este depende de cada peça.

Para os peões verifica-se se o movimento é correto com o auxílio do atributo *hasmove*, assim, pode-se diferenciar quando o peão pode mover duas casas e quando não pode. Para as torres, o movimento deve ser vertical ou horizontal, isto é, se o parâmetro do método que indica a linha final do movimento é igual ao inicial, o da coluna final deve ser diferente, e vice-versa, além disso todas as posições do tabuleiro entre a posição final e inicial são conferidas para ter a certeza de que a peça não está pulando nenhuma outra peça.

No movimento do bispo, após a certificação de que a posição final não é ocupada pelo time ou se está fora do tabuleiro, calcula-se se o módulo da diferença entre a linha final e linha inicial é igual ao módulo da diferença entre a coluna inicial e coluna final, a fim de que sempre que ele movimente uma linha

também movimente uma coluna, garantindo que seu movimento seja na diagonal, após isso, assim como a torre, é conferido se todas as posições do tabuleiro entre a posição final e inicial são posições livres no tabuleiro.

O movimento da dama pode ser entendido como uma mistura dos movimentos do cavalo e do bispo, em um primeiro momento, ocorre a verificação da posição final, e depois utilizando-se de condições para a posição final e posição inicial, o programa decide qual movimento será feito, na vertical, horizontal ou diagonal, checando sempre se a peça não está pulando outras peças.

Para o movimento do cavalo, como ele pode pular as peças, só é conferido se o destino é válido, isso é feito através do seguinte cálculo, se o módulo da diferença da linha e final e linha final é igual a um, então o módulo da diferença da coluna final e inicial deve ser igual a dois, e vice-versa.

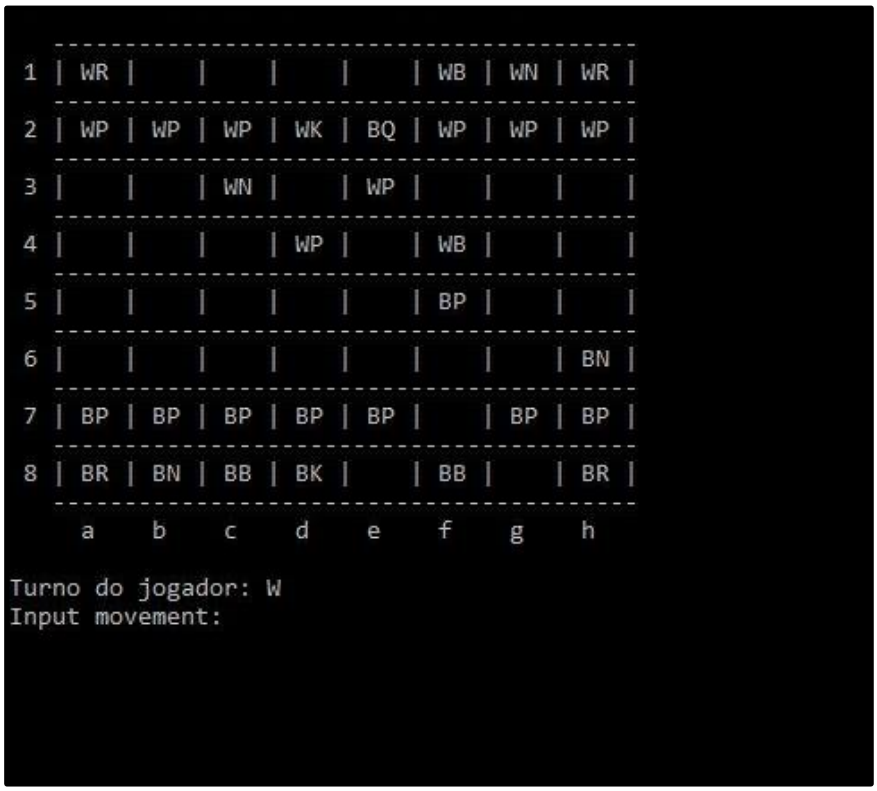
Por fim, para a movimentação do rei, ele só pode se mover para posições ao redor do mesmo e que não deixariam o mesmo em xeque.

Em um último momento, em situações de xeque, o próprio jogador é responsável por decidir se continua o jogo ou termina, ficando em cargo de determinar quando a jogada é xeque mate e assim encerrar o jogo.

Ilustrações do começo de jogo e de um jogo em andamento podem ser visualizadas nas figuras a seguir:

Figura 3 - Começo do jogo na implementação com polimorfismo

Figura 4 - Xadrez na implementação com polimorfismo em andamento



3 IMPLEMENTAÇÃO SEM METAPROGRAMAÇÃO

Nesta implementação, utiliza-se uma matriz de inteiros 8x8 como tabuleiro, cada posição da matriz recebe um número referente as peças, a casa da dezena do número é referente ao tipo da peça e a casa das unidades ao time, enquanto que uma posição vazia recebe somente o número zero, diferentemente da implementação anterior, as entradas desta implementação são as entradas oficiais da notação algébrica do xadrez, ao imprimir o tabuleiro na tela, a primeira letra é referente a cor do time e a segunda ao tipo da peça.

No começo do jogo, o jogador decide se vai começar o mesmo do início ou se vai carregar um jogo a partir de um arquivo, similarmente a implementação com polimorfismo, quando um jogador tenta fazer um movimento inválido, este não é feito e o jogador deve fazer o movimento novamente.

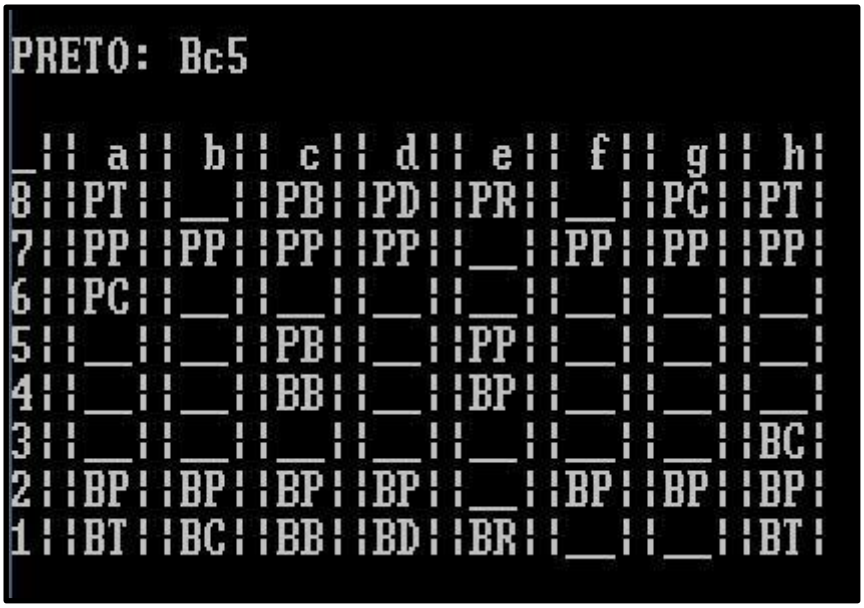
Para o movimento das peças, o programa identifica qual o tipo da peça através da dezena do número na matriz, depois, utilizando dos mesmos padrões da implementação anterior verifica-se se o movimento é válido para a posição final, se somente uma peça puder se mover, a movimentação ocorre, se mais de uma peça puder se mover e o jogador não tiver especificado qual era a peça a ser movida, o programa move a primeira encontrada.

Por fim, quando algum dos times está em xeque, o usuário é responsável por decidir se continua o jogo ou se o mesmo terminar, assim como na implementação anterior.

Ilustrações do início do jogo e final do jogo a seguir:

Figura 5 - Começo do jogo na implementação sem metaprogramação

Figura 6 - Xadrez na implementação sem metaprogramação em andamento



4 IMPLEMENTAÇÃO COM USO DE HERANÇA

Nesta implementação, a classe *Board* não teria referência à classe base *Piece* através de seus atributos, pois não é possível gerenciar classes derivadas através de um ponteiro da classe base sem polimorfismo.

Desta forma, para executar movimentos chamados através de métodos na classe *Board*, é necessário ter conhecimento de qual tipo de classe derivada deve ser acessada para tentar validar o movimento. Assim a classe *Board* deve poder visualizar os métodos das classes derivadas.

Deste modo, ocorreria uma mudança significativa nas implementações de movimento, como o método movimento da classe *Board* não pode ser chamado através de um ponteiro para qualquer derivada, o processo seria, descobrir qual o tipo da peça que vai se mover através de atributos *char* da classe da peça e depois chamar o método movimento correspondente ao tipo da peça.

5 VANTAGENS E DESVANTAGENS NAS IMPLEMENTAÇÕES

Nesta seção serão expostos e comparados diferentes abordagens para implementar o projeto da matéria, nela serão destacadas vantagens e desvantagens entre implementações que utilizam, ou não, conceitos de meta programação.

5.1 IMPLEMENTAÇÃO COM CONCEITOS DE POLIMORFISMO

O uso do Polimorfismo traz várias vantagens ao programador, esse tipo de implementação oferece grande reusabilidade e permite ‘programar no geral’ ao invés de ‘programar no específico’, tal fato ocorre devido ao uso do upcasting, no caso em questão, a classe base abstrata *Piece* pode referenciar qualquer uma de suas classes derivadas e utilizando funções virtuais, faz com que o programa escolha dinamicamente as funções corretas a serem utilizadas por cada classe, assim o programador não precisa se preocupar em decidir qual método de movimentação usar dependendo da peça em questão.

Ao utilizar polimorfismo o programa também oferece grande manutenção do código, pois para fazer mudanças nos métodos das classes derivadas só é preciso que estes sejam alterados, sem se preocupar com a classe base. Além disso, essa aproximação proporciona mais segurança ao programa, controlando o acesso aos atributos e métodos das classes derivadas, devido ao encapsulamento.

5.2 IMPLEMENTAÇÃO UTILIZANDO CONCEITOS DE HERANÇA

A herança também proporciona ao programa mais segurança por causa do encapsulamento, contudo não oferece tanta reusabilidade quanto o polimorfismo, pois mesmo que as classes derivadas possuam os métodos da classe base, o programador é quem decide qual método será utilizado.

Neste tipo de implementação, não há uma ‘programação geral’, isso ocorre pois o tabuleiro não pode referenciar todas as peças a partir da mesma classe, fazendo assim com que o programa precise identificar o tipo da peça a fim de utilizar o método correto para cada uma das classes.

5.3 IMPLEMENTAÇÃO SEM UTILIZAR ORIENTAÇÃO À OBJETO

Esta implementação oferece riscos à segurança do código, a falta de encapsulamento faz com que todas as partes do programa tenham acesso a tudo, e assim como na abordagem anterior, o programador fica em cargo de decidir qual o tipo da peça a ser movimentada e chamar a função correspondente para tal.

6 CONCLUSÃO

Através dos pontos apresentados anteriormente, chega-se à conclusão de que embora a implementação dos métodos e funções de movimentação nas diferentes aproximações seja parecida, o uso de polimorfia faz com que o programa fique mais robusto e intuitivo, devido a reusabilidade de código, e este método também faz com que o programa fique mais enxuto facilitando sua manutenção, além disso, esta é abordagem que traz maior segurança.

Por fim, levando em conta as vantagens do uso de polimorfismo, conclui-se que este é o melhor dos três métodos para implementação do jogo de xadrez em questão.