POO Projeto Final

Professor: Jose F Rodrigues Jr, SSC0204

Rafael Souza Nº 13696370, Luis Vogel Nº 13730051

1 Introdução

Jogo de xadrez implementado em Python por meio de um ambiente virtual Anaconda, usando a IDE PyCharm, com as seguintes pendências para funcionamento ideal:

- Python, versão 2.7.18
- Pygame, versão 1.9.2a0 (cogsci)

2 Detalhes Implementação

O jogo foi desenvolvido para funcionar com a versão 1.9.2a0 do Pygame, o que exigiu o uso da versão 2.7 do Python. Como resultado, a sintaxe utilizada é um pouco mais antiga em comparação com as versões mais recentes. Por exemplo, a forma de instanciar a classe **Peão**, que é derivada da superclasse **Peça**, segue uma abordagem diferente. Para chamar o construtor da superclasse, utiliza-se o seguinte formato:

```
class Pawn(Piece):
    def __init__(self, Atributos):
        Piece.__init__(self, Atributos)
```

Nas versões anteriores à 3.0 do Python, não é possível usar o pacote **abc (Abstract Base Class)**. No entanto, o jogo foi desenvolvido de acordo com as instruções, tratando a classe **Peça** como uma classe abstrata que não é instanciada. Essa classe possui apenas a função abstrata **def moves_list()**, responsável por retornar uma lista de movimentos possíveis para a peça quando clicada, dinâmica essa essencial para o funcionamento do jogo. Lembrando que a superclasse abstrata **Peça** também contém outros métodos base implementados.

Uma observação importante é que, no Python, os métodos get e set são opcionais. Apesar de existirem modificadores de acesso, é possível acessar atributos ou métodos não públicos. Portanto, o uso desses modificadores serve apenas como um aviso para que os usuários ou programadores terceiros não os utilizem.

No entanto, alguns atributos e métodos foram declarados com modificadores de acesso protegidos com o intuito de ilustrar o funcionamento do encapsulamento na programação orientada a objetos (POO), como por exemplo:

- _atributo ou _metodo protegido
- __atributo ou __metodo privado

2.0.1 Atributos Protegidos

Dentro da classe Game, foram criados dois atributos protegidos, _white_pieces e _black_pieces, e em seguida o método de acesso **get_pieces_color()** para obter acesso a uma das listas de acordo com a cor desejada.

```
class Game:
    def __init__(self, screen):
        self._white_pieces = []
        self._black_pieces = []

    def get_pieces_color(self, white):
        return self._white_pieces if white else self._black_pieces
```

2.0.2 Metodos Protegidos

Dentro da classe Game, existem métodos protegidos que são utilizados pelo método público **draw_piece_and_effects()** no arquivo Main.

```
class Game:
  def _drawing_pieces(self):
    #Codigo para desenhar as peças
  def _draw_check(self):
    #Codigo para desenhar o rei em perigo
  def _draw_moves_list(self, piece):
    #Codigo para os movimentos possiveis da peça

  def draw_pieces_and_effects(self, current_piece):
      self._draw_moves_list(current_piece)
      self._draw_check()
      self._drawing_pieces()
```

3 Regras do Jogo

Como exigido na documentação do projeto o jogo apresenta algumas regras e criterios para empate e derrota, apresentados a baixo:

Casos de Derrota:

I Algum rei seja capturado

II Rei em check após o movimento do próprio rei

Casos de Empate:

I Caso algum dos jogares **peça empate apertando a tecla "t"** em qualquer momento do jogo, e o outro jogador aceite.

II Caso tenha apenas reis em campo.

III Caso exista apenas o rei, um cavalo e o rei inimigo em campo.

IV Caso exista apenas o rei, um bispo e o rei inimigo em campo.

V Caso aconteça 50 movimentos consecultivos sem nenhuma peça capturada, informação exibida no tabuleiro.

Promoção:

Quando um peão consegue atravessar todo o tabuleiro, uma tela de Promoção e exibida, onde e possível escolher as seguintes opções:

I Bispo - "b"

II Cavalo - "k"

III Torre - "r"

IV Rainha - "q"

Reiniciar o jogo:

I Solicitar empate com a tecla "t".

II Algum jogador vença e seja apertado a tecla "Enter".

No jogo, existem placares para cada jogador, portanto é necessário haver um vencedor ou um empate, a fim de contabilizar os pontos sempre que houver um vencedor, e manter os placares inalterados quando houver um empate.

4 Considerações Finais

O jogo foi implementado de forma que a classe Game, tenha em sua composição as classes Peças e também a classe Board, onde cada uma desempenhando sua função.

Classe Peça: Responsável por armazenar os atributos de cada peça, como imagem, posição. Além de calcular os movimentos, mover e alertar que a peça foi capturada.

Classe Board: Responsável praticamente por desenhar todo o tabuleiro, exibindo informações sobre o estado atual do jogo, telas de vencedor, empate, promoção, mensagens e por fim imagens.

Classe Game: Responsável pela dinamica do jogo, por exemplo checagem e aplicação das regras, monitoramento das batalhas e status da partida como game over ou pause.

É importante ressaltar que este é apenas um resumo, para complementar as informações segue a baixo o UML elaborado.