UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

JOÃO VICTOR DA SILVA FRANCISCO - DRE: 121055346

GABRIEL SANTOS SCHUINA - DRE: 121056295

RELATÓRIO FINAL - COMPUTAÇÃO II

Rio de Janeiro

2021

Neste relatório será abordado a conclusão do Trabalho da Final da Disciplina de Computação 2. O objetivo do trabalho era construir um jogo de xadrez utilizando conceitos de objetos e classes, além de seguir algumas diretrizes propostas pelo professor, tais como: salvamento do jogo, tratamento de exceções, geração de estatísticas e validação dos movimentos das peças.

Em relação à entrega anterior, foram acrescentadas duas classes, **Classe Player** e **Classe Jogo.**

A seguir serão explicadas todas as classes utilizadas no projetos, as quais também estão listadas logo abaixo:

Classes:

- Classe Peca()
    - Classe Peao()
    - Classe Torre()
    - Classe Cavalo()
    - Classe Bispo ()
    - Classe Dama()
    - Classe Rei()
- Classe Tabuleiro()
- Classe Player()
- Classe Jogo()

Neste relatório parcial vamos falar sobre o desenvolvimento das ideias propostas no relatório inicial, comentando as novas características do código, nosso tratamento de exceções e a explicação mais elaborada dos métodos antigos e novos.

- **Classe Peça → Class Peca():**

A classe Peca() teve uma alteração nos itens que ela qualificava cada peça do tabuleiro. Antes ela só informava o tipo de peça e a cor dela. Agora além de informar esses dados também temos a informação da posição inicial.

- **Classes das peças:**

Criamos classes específicas para cada peça do jogo. Todas elas são herdeiras da classe “Peca()”, porém cada uma possui um método extra chamado “movimentacao()” e casas “percorridas()”. O primeiro tem o objetivo de avaliar o formato esperado do movimento de cada peça. Cada peça do xadrez possui movimentos específicos, e a função deste método é dado duas coordenadas avaliar se a partida da coordenada inicial para a final é válida tendo em mente como a peça tal se movimenta. Já o segundo, tem como objetivo único contabilizar o número de casas percorridas por uma peça em uma jogada, para fins estatísticos. Além disso a classe Rei, possui um método a mais que se chama “atacavel()” que dada a coordenada do rei retorna se o rei está na mira de algum ataque ou não.

- **Classe Tabuleiro → Class Tabuleiro():**

A classe Tabuleiro() foi incrementada de novos métodos e os já existentes foram mais desenvolvidos. A seguir vão ser listados os métodos e as funcionalidades que eles representam.

  - **Método arrumar_tabuleiro:**

    Uma mudança do relatório inicial para esse foi o método **arrumar_tabuleiro**. Antes sua definição dizia que ele apenas arrumaria o tabuleiro com as peças na posição inicial, agora, no entanto ele além de organizar as peças na posição inicial, também é capaz de organizar as peças nas posições em que estavam em um arquivo salvo. O método possui um argumento de entrada **inicio** que possui um default **True**. Esse argumento é o responsável por decidir se o método organizará o tabuleiro no formato inicial ou no formato do arquivo.

  - **Método verificar_posicao:**

    Método que verificará se a coordenada inserida possui uma peça. Esse método será de grande importância para o gerenciamento da

movimentação das peças do tabuleiro. Esse método não sofreu alteração. Seu objetivo ainda é o mesmo.

- **Método verificar_xeque, informar_xeque e verificar_xequemate:**

  O método **verificar_xeque** possui o objetivo de limitar as jogadas possíveis para que o jogador não faça um movimento que deixe seu rei em xeque. A função só permite opções de movimento do jogador quando a opção for um movimento de defesa para o rei.

  O método **informar_xeque** possui objetivo de exibir uma mensagem na tela do jogador que está com seu rei em xeque, informando-o sobre a necessidade de proteção do mesmo.

  O método **verificar_xequemate** tem relação ao seguimento do jogo. Ele que avaliará se algum rei está em xeque mate e diante disso ditará o fim do jogo.

- **Método verificar_posicao_rei:**

  O método tem como finalidade, dada uma coordenada de uma peça qualquer, retornar o mapeamento da posição de um rei de mesma cor no tabuleiro.

- **Método caminho_livre:**

  Esse foi um novo método criado o qual tem como objetivo verificar o caminho de uma coordenada a outra. Ele faz uso do conhecimento de que os movimentos do xadrez são no geral 3: movimento diagonal, movimento reto, vertical ou horizontal, e o movimento em “L” do cavalo. Dessa forma quando fornecidos duas coordenadas, o método já sabe qual tipo de movimento é, e dessa maneira verifica cada casa no trajeto da coordenada inicial até a final, através da função **verificar_posicao**, e retorna um valor booleano para indicar se o caminho está livre (**True**) ou se há peças nele (**False**).

- Método moviementacao_valida:

  Método que realizará as devidas operações de movimento e ataque das peças do tabuleiro. Neste método tem que haver uma validação das escolhas das coordenadas inseridas pelo usuário tanto para não permitir a colocação do rei em xeque, quanto para protegê-lo caso ele já esteja como tal.

Novamente este foi um método que não teve alteração na sua definição. Na questão do código em si esse método foi bastante desenvolvido e nele estão a junção de outros métodos como o **caminho_livre** da própria classe Tabuleiro() e o método **movimentacao** das classes de peças. Através desses métodos é possível verificar se o movimento proposto pelo usuário é válido e assim retornar **True** ou inválido e retornar **False**.

- **Método imprimir_tabuleiro:**

  Método que imprime a máscara de um tabuleiro no console, a partir da matriz 8x8 criada no método construtor e organizada no método arrumar tabuleiro.

- **Método salvar_tabuleiro:**

  O método salvar tabuleiro tem por objetivo, armazenar os dados do tabuleiro em um arquivo .txt para uma futura continuação de um jogo já iniciado, não possui valor de entrada, porém está diretamente ligada, com o método **arrumar_tabuleiro**

- **Classe Player:**

A Classe Player é uma novidade que foi criada neste último versionamento do programa, com finalidades estatísticas e alteração da dinâmica do jogo.

- **Método registrar_estatisticas:**

  Registra o número de casas percorridas e o número de peças capturadas em uma dada jogada, registrando-as em uma lista.

- **Método mudar_vez:**

  O método mudar vez, foi criado com o intuito de trocar de turnos durante uma partida, alternando assim a vez de jogada entre os jogadores 1 e 2

- **Classe Jogo:**

A classe Jogo também é uma novidade nesta última versão do jogo, a classe foi criada pensando na interação do usuário com o jogo. Portanto, seus métodos são 100% voltados à interação do usuário.

- **Método transformar_letra e verificar_numero:**

  Os métodos possuem finalidades extremamente semelhantes, diferenciando-se apenas no eixo em que agem. O método **transformar_letra** converte as letras das colunas (A-H) pelos valores correspondentes na lista do tabuleiro (0-7), já o **verificar_numero**, converte os números das linhas (1-8) pelos valores correspondentes na lista do tabuleiro (0-7)

- **Método verificar_salvamento:**

  Método que verifica se a resposta inserida no console é um pedido de salvamento do jogo. O método foi criado, a partir da necessidade de salvamento em qualquer momento do jogo.

- **Método verificar_estatisticas:**

  Método semelhante ao anterior, mas que verifica se a resposta inserida no console é um pedido de visualização das estatísticas. As estatísticas de jogo são feitas a partir da biblioteca matplotlib, gerando, então, um gráfico com peças capturadas por turno e quantidade de casas percorridas também em um turno.

- **Método solicitar_nome_arquivo:**

  Método que pergunta ao usuário qual nome o mesmo deseja dar ao arquivo de salvamento. Retorna o nome do arquivo, que será utilizado para o salvamento do tabuleiro (classe Tabuleiro) e das estatísticas (classe Jogo).

- **Método salvar_estatisticas:**

  Este método salva as estatísticas, em um arquivo nomeado a partir do “solicitar_nome_arquivo()”. O método possui duas entradas, uma delas é o nome do arquivo como dito anteriormente e a outra entrada indicará de qual player serão salvas as estatísticas.

- **Método mostrar_casas_percorridas e mostrar_historico_capturas:**

  Estes métodos possuem finalidades semelhantes, uma vez que, neles, as estatísticas são plotadas em um gráfico a partir da biblioteca matplotlib. Cada método cria seu determinado gráfico, ou seja, um gráfico contendo o histórico de casas percorridas durante a partida e no outro o histórico de capturas.

- **Método mostrar_estatisticas:**

  O método “mostrar_estatisticas()” é responsável por juntar ambos os métodos anteriores que criam os gráficos. Este método foi criado pensando na organização do jogo, já que ambos os métodos sempre são “chamados” juntos, logo, o método mostrar_estatisticas junta ambos em um método só.

- **Método verificar_existencia_arquivo:**

  Sua finalidade é verificar a existência de um arquivo,dando um nome de entrada, tentando assim, abrir o arquivo pedido. Caso o arquivo não exista a função retorna uma expressão booleana negativa, caso exista, a função retorna True.

- **Métodos solicitar_primeira_coordenada e solicitar_segunda_coordenada:**

  Dois métodos semelhantes e de extrema importância para a continuidade do jogo, pois com eles as ações de movimentação, salvamento e visualização das estatísticas, são possíveis. Recebe como entrada apenas um objeto da classe tabuleiro, com finalidade de movimentar as peças do no tabuleiro a partir das coordenadas inseridas na função principal “main” do programa. Entretanto eles também podem verificar se a resposta obtida no console é um pedido de salvamento ou um pedido de visualização das estatísticas, com isso, conseguem realizar os desejos dos usuários.

- **Método verifica_escolha_peca_inimigo:**

  Este, tem como finalidade bloquear o manuseio das peças do jogador oponente, portanto, limitando assim, o jogador 1 a mexer somente nas peças brancas e o jogador 2 a mexer somente nas peças pretas.

Com isso, já conseguimos estabelecer parâmetros dentro do jogo, fazendo com que o mesmo tivesse as regras de um jogo de xadrez real. A mecânica das movimentações de todas as peças estão correspondentes ao pedido, ademais, outros requisitos foram cumpridos no código.

Além disso, concluímos também com sucesso a implementação de um sistema de cores e símbolos no programa para que melhorasse a visualização do jogo para os usuários.

É válido ressaltar que, o processo de criação do programa foi de extremo aprendizado, desafios e conquistas. Desenvolvemos habilidades ímpares, como o trabalho em equipe, além do incentivo no autodidatismo em pesquisar fontes de conhecimento fora das aulas de computação, melhorando assim, nosso repertório na linguagem de programação.

Por fim, todos os requisitos foram cumpridos gradualmente durante as atualizações e entregas do código. Agradecemos veementemente pelo incentivo e nos sentimos realizados por essa conquista. Acreditamos que ainda há muito o que melhorar no código, mas, no momento, demos nosso melhor. Aguardamos o feedback.