**Universidade da Beira Interior**
Departamento de Informática


# Nº 1 - 2022: *[TP1] Agente inteligente para Xadrez*

Elaborado por:

**Simão Andrade a46852**

Orientador:

**Professor Doutor Hugo Pedro Proença**

17 de novembro de 2022

# *Agradecimentos*

A conclusão deste trabalho, bem como da grande maior parte da minha vida académica, não seria possível sem a ajuda do auxílio e esclarecimento de dúvidas do Professor Doutor Hugo Proença, e de dúvidas retiradas a colegas fora desta universidade, porém experientes no jogo de xadrez, que deram algumas ideias de implementação para o meu agente.

# *Conteúdo*

# *Lista de Excertos de Código*

# *Acrónimos*

**IA** Inteligência Artificial

**UBI** Universidade da Beira Interior

**UC** Unidade Curricular

*Capítulo*

# 1

# *Introdução*

## 1.1 Enquadramento

Este relatório foi feito no contexto da unidade curricular de Inteligência Artificial (IA) da Universidade da Beira Interior (UBI).

## 1.2 Objetivos

Neste trabalho prático tem como finalidade entender a representação do estado do jogo, pesquisa em árvore, algoritmos de busca e o desenvolvimento a melhor heurística para ajudar a determinar qual a melhor jogada a fazer.

Para atingir o desenvolvimento da heurística é importante considerar que, devido à quantidade imensa de jogadas possíveis no xadrez, vamos apenas fazer uma pesquisa de profundidade 3 a cada decisão de jogada feita, ou seja, vamos apenas prever para as 3 jogadas seguintes que irão ser feitas no jogo (incluindo as do adversário) e determinar aquela onde o nosso jogador tem a melhor pontuação possível (maximizador) e o seu adversário tem a pior pontuação possível (minimizador), isto é possível usando o algoritmo "minimax".

Mesmo fazendo a pesquisa de apenas 3 de profundidade, continuam a ser avaliados milhares de casos onde já existe uma melhor jogada possível, para evitar isto vamos usar o algoritmo "alpha-beta prunning"e permitir que as avaliações futuras na heurística sejam de maior complexidade sem comprometer o desempenho do programa.

## 1.3 Organização do Documento

De modo a refletir o trabalho que foi feito, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projeto, a motivação para a sua escolha, o enquadramento para o mesmo, os seus objetivos e a respetiva organização do documento.

2. O segundo capítulo – **1ª Abordagem** – descreve o desenvolvimento de uma implementação de heurística baseada na quantidade e importância das peças no tabuleiro.

3. O terceiro capítulo – **2ª Abordagem** – descreve o desenvolvimento de uma implementação de heurística baseada numa estrutura padrão que as peças podem seguir.

4. O quarto capítulo – **3ª Abordagem** – descreve o desenvolvimento de uma implementação de heurística baseada nas peças que podemos atacar.

5. O quinto capítulo – **Tentativa Abordagem** – descreve uma tentativa de implementação de métodos que permitem uma melhor pesquisa de estados.

6. O sexto capítulo – **Conclusão** – descrevem-se os objetivos conseguidos e não conseguidos, reunidos em tópicos autoexplicativos, e o trabalho futuro relacionado com o projeto.

*Capítulo*

# 2

# 1ª Abordagem

## 2.1 Introdução

Neste capitulo vai ser pormenorizado o desenvolvimento de uma implementação de heurística baseada na quantidade e importância das peças no tabuleiro.

## 2.2 Processo de desenvolvimento

Inicialmente, foi usado como inspiração o código fornecido na página da cadeira denominado *client_a_bit_smart.py*. Nele continha parte das funções para auxiliar o desenvolvimento deste trabalho e permitir um foco maior na função objetivo, sendo ela a maior diferenciação na complexidade dos trabalhos.

A função objetivo recebe o estado atual, recebe o estado atual e o qual jogador é que esta a ser jogado com (1 para as brancas, 0 para as pretas) e devolve uma pontuação que irá ser atribuída para esse estado. A função contida no código seguinte tinha a seguinte ordem de ideias:

1. Dar peso às jogadas para avisar que as peças se mantenham na defensiva durante muitas jogadas;
2. Declaração das peças brancas e pretas em modo *string* (a, h = torres; b, g = cavalos; c, f =bispo; d= rainha; e = rei; restantes = peões);
3. Declaração da valoração de cada uma das peças comidas (50 = torres; 30 = cavalos; 35 = bispos; 90 = rainha; 900 = rei; 10 = peões);
4. Percorrer todas as peças contidas na 'string';

5. Procura se o tabuleiro a analisar contêm essa peça;
6. Caso contenha, aumenta a pontuação tendo em conta a valoração da peça dada na lista 'pts.'
7. Aumentar ou diminuir (caso não contenha) a pontuação da posição dentro em conta o peso e a posição dela (eixo) dos x (do lado dos brancos);
8. Repetir o mesmo processo para as peças pretas;
9. Devolver a diferença entre as duas pontuações e devolver um valor positivo ou negativo dependendo da peça com que jogamos;

## 2.3 Escolha da valoração

A escolha dos valores foi feita para atingir os seguintes resultados:

1. Evitar trocas entre uma peça menor (bispo e cavalo) por três peões
2. Encoragar possuir um par de bispos
3. Evitar troca de duas peças menores [1](bispo e cavalo)

Atingimos o ponto um com as seguintes inequações:

- B > 3P
- N > 3P

Ponto dois é atingido com a seguinte inequação:

- B > N

Logo podemos chegar á seguinte conclusão:

- B > N > 3P

Podemos chegar ao ponto 3 através de:

- B + N > R + P
- Q + P = 2R
- B + N = R + 1.5P

Logo considerando o P = 10, temos:

P = 10
B = 35
N = 30
R = 50
Q = 90

## 2.4 Conclusões

Nesta fase do trabalho já foi possível obter resultados bons ao nível de heurística, porém ainda muito básicos, pois ele apenas avalia a preservação das peças e não avalia o que pode ser uma boa jogada.

*Capítulo*

# 3

# 2ª Abordagem

## 3.1 Introdução

A avaliação inicial era bastante simplista, pois apenas contava o quantidade e qualidade das peças no tabuleiro. De modo a melhor a nossa avaliação, vamos adicionar uma avaliação que toma em conta as posições das peças

## 3.2 Processo de seleção de valores

A escolha de valores feita tem que ser de tal modo que aconteçam as seguintes situações:

1. Priorizar estados onde o cavalo esteja no centro do tabuleiro, em constrante a situações onde ele encontra-se na borda.
2. Priorizar boas *Openings*[2].
3. Priorizar uma boa estrutura de peões (*Pawn Struture*[3]).
4. Priorizar estados onde o rei se encontra recuado e protegido pelas restantes peças.
5. Priorizar uma movimentação centralizada de bispos.

Aqui será dado uma estrutura para cada peça representado numa lista com 64 valores (um para cada quadrado no tabuleiro), e a sua posição no tabuleiro terá uma valoração associada ao mesmo, determinando assim a jogada ideal para aquela peça por padrões de jogo.

Estes foram os valores obtidos:

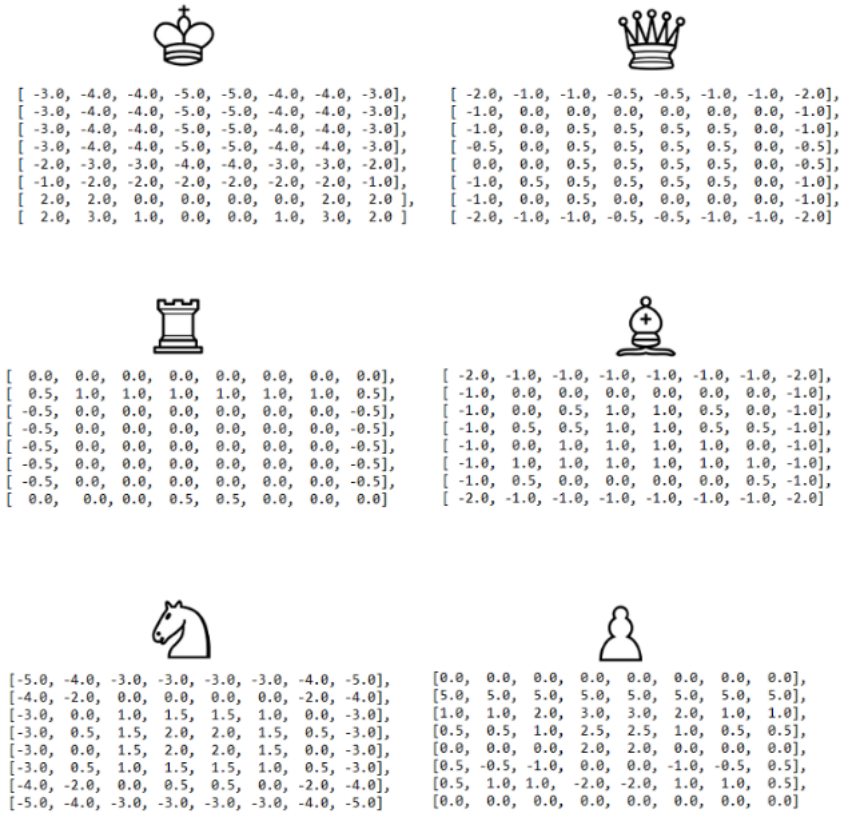



Figura 3.1: Valores das posições de cada peça no tabuleiro.

## 3.3 Conclusões

Através deste método aqui introduzido, foi possível criar um agente que mesmo sendo simples, não comete erros comuns. Porém, o mesmo ainda não tem nenhuma noção estratégica.

*Capítulo*

# 4

# 3ª Abordagem

## 4.1 Introdução

Na seguinte abordagem, foi implementada uma função dado uma peça, irá determinar quais peças ela ameaça (seja acabada). A função irá receber o tabuleiro e a peça a analisar e irá devolver uma lista com as peças ameaçadas em formato *string*.

## 4.2 Processo de desenvolvimento

Contêm o seguinte processo de análise:

1. Obtém a peça a analisar e a sua posição no tabuleiro em 2D;
2. Verifica qual a peça em questão e qual o jogador que a tem;
3. Percorre, num ciclo *for*, todos os quadrados que essa peça pode legalmente mover-se
4. Caso encontre um quadrado ocupado por uma peça adversária, adiciona essa peça à lista de peças ameaçadas;
5. No final retorna a lista de peças ameaçadas;
6. Na função objetivo, irá ser declarado a variavel que guarda as peças ameaçadas e irá verificar qual a peça em questão;
7. Após determinar qual a peça em questão, irá ser dado uma valoração á variável **'score_w_threats'** ou **'score_b_threats'**, dependendo do jogador que a tem, relativamente á importância dessa peça no jogo.

O trecho de código seguinte mostra uma porção da função active_threat e o seu funcionamento:

```
    elif piece == 'b' or piece == 'g':  # Cavalo branco
        # Ciclo que percorre o tabuleiro em L
        for i in range(1, 3):  # Nordeste
            if pos2[0] + i > 7 or pos2[1] + 3 - i > 7:
                break
    o = board[pos2_to_pos1((pos2[0] + i, pos2[1] + 3 - i))]  # Peca em
        ameaca
            if o in w or o == 'z':  # Ascii de 'a' a 'z'
                break
            elif o in b:
                res.append(o)
                break
        for i in range(1, 3):  # Sudeste
            if pos2[0] - i < 0 or pos2[1] + 3 - i > 7:
                break
            o = board[pos2_to_pos1((pos2[0] - i, pos2[1] + 3 - i))]  #
                Peca em ameaca
            if o in w or o == 'z':  # Ascii de 'a' a 'z'
                break
            elif o in b:
                res.append(o)
                break
        for i in range(1, 3):  # Sudoeste
            if pos2[0] - i < 0 or pos2[1] - 3 + i < 0:
                break
            o = board[pos2_to_pos1((pos2[0] - i, pos2[1] - 3 + i))]  #
                Peca em ameaca
            if o in w or o == 'z':  # Ascii de 'a' a 'z'
                break
            elif o in b:
                res.append(o)
                break
        for i in range(1, 3):  # Noroeste
            if pos2[0] + i > 7 or pos2[1] - 3 + i < 0:
                break
            o = board[pos2_to_pos1((pos2[0] + i, pos2[1] - 3 + i))]  #
                Peca em ameaca
            if o in w or o == 'z':  # Ascii de 'a' a 'z'
                break
            elif o in b:
                res.append(o)
                break
        return res
```

Excerto de Código 4.1: Função sobre ameaça ativa.

No trecho de código acima, é mostrado a analise feita caso a peça a analisar chegar um cavalo branco, representado pelas letras **'b'** e **'g'** no tabuleiro.

## 4.3 Conclusões

Esta abordagem permitiu a nossa avaliação chegar a resultados mais promissores, diminuindo considerávelmente o número de estados a analisar e chegando a melhores resultados.

*Capítulo*

# 5

# *Tentativa de abordagem*

## 5.1 Introdução

Nesta abordagem, temos como objetivo diminuir o número de estados analisados a poder assim eventualmente aumentar a profundidade do nosso programa.

## 5.2 Ordem de Ideias

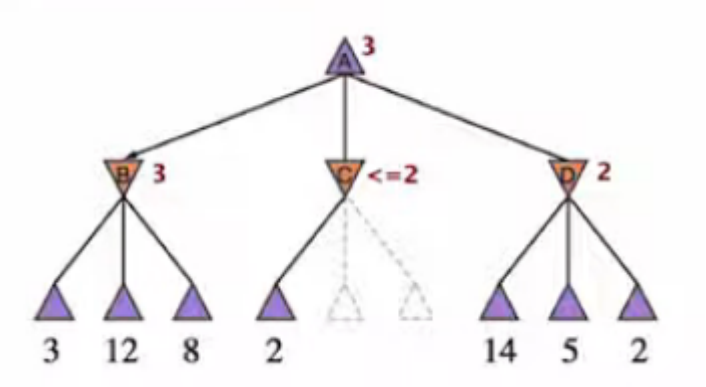
Pretendemos melhorar o algoritmo "minimax"através do "move ordering", onde o algoritmo irá analisar os filhos do estado atual e reordena-los pela sua heurística, assim evitando que o algoritmo análise estados que não são bons para o jogador.

O *'Move ordering'* pode ser implementado de duas maneiras:

1. Dar sugestões sobre o que é melhor fazer. Por exemplo, na promoção de um peão no xadrez, capturar peças de alto valor com peças de menor valor são, em média, bons movimentos.

2. A função de geração de jogadas (*'sucessor_states'*) devolve melhores jogadas antes, obtendo a heurística de quão bom é o movimento a avaliar na posição ao nível 1 de profundidade menor (a sua busca rasa / aprofundamento iterativo). Calcula a avaliação na profundidade **N-1**, ordenou os movimentos e depois avaliou na profundidade **N**.

**Exemplo:**




- Não foi possivel cortar nenhum nodo de 'D', pois o pior estado já foi analisado primeiro.
- Se o terceiro estado (folha 2), tivesse sido analisado primeiro, o algoritmo iria cortar os outros nodos (14 e 5).

## 5.3 Conclusão

O código foi implementado e testado, mas não foi possível obter resultados positivos, pois o algoritmo chegava a cortar nodos com boas heurísticas, o que pode ser devido a uma má implementação do mesmo.

*Capítulo*

# 6

# *Conclusões e Trabalho Futuro*

## 6.1 Conclusões Principais

Em suma, neste trabalho foram melhorados os conceitos de pesquisa, otimização e a diferenciação entre qualidades de heurísticas no jogo de xadrez. Também foi possível solidificar conhecimentos de *Python* e do seu funcionamento.

## 6.2 Trabalho Futuro

O próximo trabalho desta Unidade Curricular (UC) envolve *machine learning* e redes neuronais, e acredito que com este trabalho obtive boas bases para melhor começar o mesmo.

# Bibliografia

[1] Minor pieces chess. [Online]https://www.chess.com/forum/view/general/minor-and-major-pieces Último acesso a 10 de Novembro de 2022.

[2] Openings, 2022. [Online] https://www.chess.com/openings. Último acesso a 10 de Novembro de 2022.

[3] Pawn structure, 2022. [Online] https://www.chess.com/terms/pawn-structure. Último acesso a 10 de Novembro de 2022.