

Universidade da Beira Interior

Departamento de Informática



**Departamento de
Informática**

Nº 1 - 2022: *[Bot de Xadrez]*

Elaborado por:

45842 Diogo Manuel Paiva dos Santos

Orientador:

Professor Doutor Hugo Pedro Proença

20 de novembro de 2022

Agradecimentos

A conclusão deste trabalho, não seria possível sem a ajuda do Professor Doutor Hugo Pedro Proença pois não teríamos o conhecimento adquirido nas aulas para a resolução do mesmo.

Conteúdo

| | |
|--|------------|
| Conteúdo | iii |
| Lista de Figuras | v |
| 1 Introdução | 1 |
| 1.1 Enquadramento | 1 |
| 1.2 Motivação UBI | 1 |
| 1.3 Objetivos | 1 |
| 1.4 Organização do Documento | 2 |
| 2 Cliente Inicial | 3 |
| 2.1 Introdução | 3 |
| 2.2 Conceitos e a sua aplicação | 3 |
| 3 Implementação | 5 |
| 3.1 Introdução | 5 |
| 3.2 Ameaça Ativa | 5 |
| 3.3 Valoração Peça-Posição | 6 |
| 3.4 Xequre-Mate | 8 |
| 4 Conclusões e Trabalho Futuro | 9 |
| 4.1 Introdução | 9 |
| 4.2 Conclusões e Trabalho Futuro | 9 |
| Bibliografia | 11 |

Lista de Figuras

| | | |
|-----|---|---|
| 2.1 | Explicação do algoritmo <i>minimax Alpha-Beta</i> | 4 |
| 3.1 | Valoração de <i>Hans Berliner's system</i> | 6 |
| 3.2 | Valoração posição-peça | 7 |

Lista de Excertos de Código

| | | |
|-----|---|---|
| 2.1 | Função Objetiva Inicial | 4 |
| 3.1 | Excerto de código da ameaça ativa | 6 |
| 3.2 | Função peça-posição | 7 |
| 3.3 | Exerto da Função de Xeque-Mate | 8 |

Acrónimos

IA Inteligência Artificial

Capítulo

1

Introdução

1.1 Enquadramento

Este documento funciona como relatório do primeiro trabalho prático da unidade curricular de Inteligência Artificial (IA).

Este trabalho, cotado para 3 valores, tenciona solidificar os conhecimentos adquiridos tanto nas aulas teóricas como práticas.

1.2 Motivação UBI

Pretende-se através deste projeto, colocar em prática os conhecimentos adquiridos na cadeira de IA, nomeadamente a elaboração de um inteligência artificial para jogar xadrez. Esta implementação propõe desafiar os alunos na melhoria da função objetiva oferecida pelo professor.

1.3 Objetivos

A finalidade do trabalho é perceber e melhorar a IA feita para jogar xadrez. Para isso será necessário enumerar um conjunto de passos, sendo eles:

- Implementação da ameaça ativa, para as peças no tabuleiro.
- Otimização do código para tentar fazer uma pesquisa mais .
- Elaboração deste **Relatório**.

1.4 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 – **Cliente Inicial** – descreve o que recebemos e como procede aplicação fornecida.
3. O terceiro capítulo – **Implementação** – descreve o que feito sobre a função objetiva colocadas na IA.
4. O quarto capítulo – **Conclusões e Trabalho Futuro** – descreve os objetivos atingidos e os não atingidos, reunidos em tópicos auto explicativos.

Capítulo

2

Cliente Inicial

2.1 Introdução

Neste capítulo, o objetivo é, através dos conteúdos lecionados em sala de aula, perceber as funcionalidades presentes no cliente fornecido pelo professor.

2.2 Conceitos e a sua aplicação

O cliente fornecido pelo professor, receberá uma mensagem vinda do servidor, em seguida o cliente irá analisar o tabuleiro e verá todos os movimentos possíveis nesse tabuleiro e para os seguintes, criando uma árvore em 'n' níveis, sendo o 'n' um número que o utilizador escolher.

A forma como o cliente avalia os tabuleiros é verificando se as peças estão no mesmo e caso se verifique é-lhe atribuído pontos por cada peça, fazendo o mesmo para as do oponente subtraindo pontos por cada peça ao valor do tabuleiro. Cada peça tem um valor, como por exemplo, a rainha vale 100 pontos. Também é fornecido mais pontos conforme as peças avancem no tabuleiro, para evitar jogadas sem senso e forçando mais o ataque.

```

def f_obj(board, play):
    weight_positions = 1e-1
    w = 'abcdedghijklmnop'
    b = 'ABCDEFGHGIJKLMNOP'
    pts = [10, 7, 6, 100, 9999, 6, 7, 10, 1, 1, 1, 1, 1, 1, 1, 1]
    score_w = 0
    score_w_positions = 0
    for i, p in enumerate(w):
        ex = board.find(p)
        if ex >= 0:
            score_w += pts[i]
            p2 = pos1_to_pos2(ex)
            score_w_positions += weight_positions * p2[0]
    score_b = 0
    score_b_positions = 0
    for i, p in enumerate(b):
        ex = board.find(p)
        if ex >= 0:
            score_b += pts[i]
            p2 = pos1_to_pos2(ex)
            score_b_positions += weight_positions * (7- p2[0])

    return (score_w + score_w_positions - score_b -
            score_b_positions) * pow(-1, play)

```

Excerto de Código 2.1: Função Objetiva Inicial

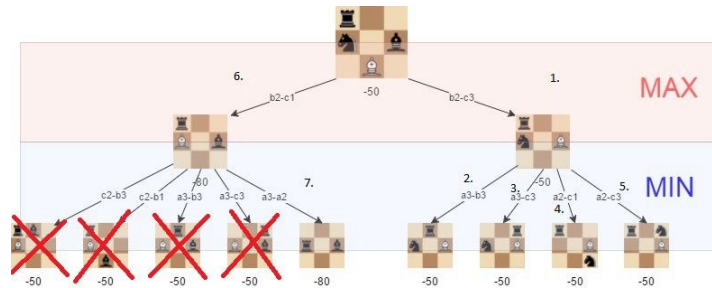


Figura 2.1: Explicação do algoritmo *minimax Alpha-Beta*

O cliente vai avaliar e fazer uma busca em árvore em *minimax Alpha-Beta*, sendo a nossa jogada maximizador e a jogada do oponente minimizador. Em suma, O algoritmo *minimax Alpha-Beta* avalia o tabuleiro com um determinado valor através da função objetiva, e se esse valor for pior que um tabuleiro previamente examinado, os tabuleiros posteriores não necessitam ser avaliados, como se verifica na figura 2.1. No final desta busca, já vai ser possível ver o melhor caminho devolvendo a jogada maximizadora para o servidor.

Capítulo

3

Implementação

3.1 Introdução

Este capítulo resume-se as implementações. A estas foi adicionado três funcionalidades, de modo a melhorar o cliente inicial e a tornar o processo mais rápido. Este capítulo divide-se em três secções:

- Ameaça Ativa;
- Valoração Peça-Posição;
- Xeque-Mate.

3.2 Ameaça Ativa

A ameaça ativa é um função que vai receber um tabuleiro e uma peça. Esta vai devolver todas as peças oponentes que estão a ser ameaçadas (podem ser "comidas"). Na função objetiva é dada uma valoração a cada peça ameaçada.

A função funciona da seguinte forma, se a peça for uma torre branca (a ou h), como no código referido em baixo, esta vai percorrer as posições possíveis para onde a torre se pode movimentar. Se tiver um peão branco à frente, a função devolve uma lista vazia, se por acaso no sentido norte tiver uma rainha preta vai devolver uma lista com um 'D' lá dentro, fazendo isto para as quatro direcções da torre se estas tiverem dentro dos limites do tabuleiro.

```

def ameaca_ativa(board, piece):
    res = []
    pos1 = board.find(piece)
    pos2 = pos1_to_pos2(pos1)
    if piece == 'a' or piece == 'h': # TORRE BRANCA
        for i in range(1, pos2[0] + 8): # north
            if i > 8 or pos2[0]+i > 7:
                break
            o = board[pos2_to_pos1([pos2[0]+i, pos2[1]])] # pe a
                ameaca
            if ord(o) >= 97 and ord(o) <= 112: # ascii de a ate p
                break
            if ord(o) >= 65 and ord(o) <= 80: # ascii de A a P
                res.append(o)
                break
#
pts_ameaca = [1, 0.75, 0.67, 5, 100, 0.67, 0.75, 1, 0, 0, 0, 0, 0, 0, 0,
0]
#torre cavalo bispo rainha rei bispo cavalo torre

```

Excerto de Código 3.1: Excerto de código da ameaça ativa

A segunda parte do código corresponde a valoração que atribui a cada peça quando esta é ameaçada, a rainha e o rei são aqueles que têm mais pontos, pois são as mais importantes, sendo aquela com mais importância o rei. O resto foi uma adaptação da valoração das peças do *Hans Berliner's system*. Uma peça ameaçada não pode ter o mesmo valor que uma peça não ameaçada então foi reduzido a escala.





| Symbol |  |  |  |  |  |
|--------|---|---|---|---|--|
| Piece | pawn | knight | bishop | rook | queen |
| Value | 1 | 3.2 | 3.33 | 5.1 | 8.8 |

Figura 3.1: Valoração de *Hans Berliner's system*

3.3 Valoração Peça-Posição

Esta secção, corresponde aos valores que cada peça têm num determinado sitio do tabuleiro. Esta função recebe uma posição em 2D e uma peça e a mesma vai devolver o valor da peça nessa determinada posição.

```

# Recebe uma peça e a sua posição e devolve os pontos que essa
# peça tem naquela posição
def points_position(piece, pos):
    ## PEAS BRANCAS ##
    # Torre
    if piece == 'a' or piece == 'h':
        area = [[0.0, 0.0, 0.0, 0.1, 0.1, 0.0, 0.0, 0.0],
                [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                [0.1, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.1],
                [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
        return area[pos[0]][pos[1]]
    # Cavalo

```

Excerto de Código 3.2: Função peça-posição

Para os valores em cada posição, houve a necessidade de adaptar os valores visto que estes se encontravam muito altos para valoração por peça, que eu pretendia. Para tal otimizei o 3.2 para se encontrar dentro dos valores. Para uma posição não valer mais que uma peça dividi todos os valores por cinco.

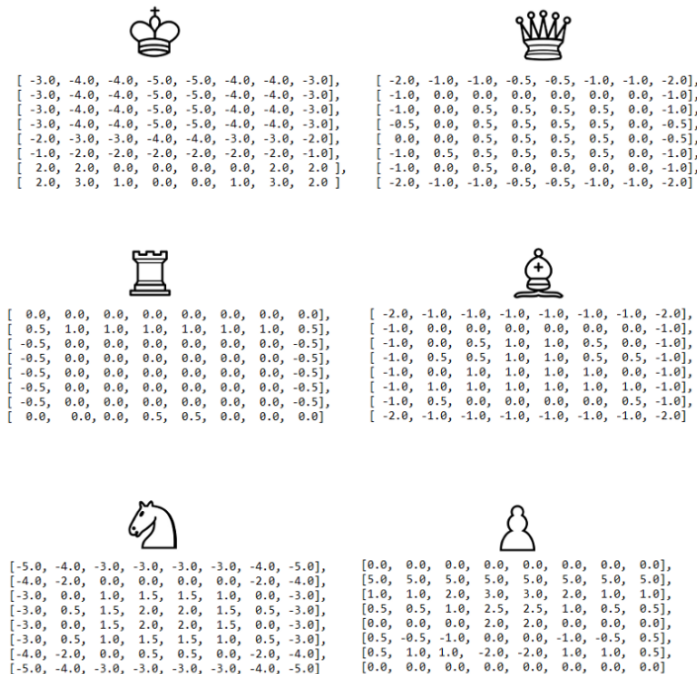


Figura 3.2: Valoração posição-peça

3.4 Xeque-Mate

Esta função localiza-se na função main. Mal o cliente receba o tabuleiro atualizado vindo do servidor, a função verifica a possibilidade de haver alguma peça que possa fazer xeque-mate. Caso tal se verifique a função pega nessa peça coloca-a na posição do rei ficando posição anterior como 'z' forçando assim o final do jogo, impossibilitando a pesquisa por árvore.

```
w = 'abcdefghijklnop'
b = 'ABCDEFGHGIJKLMNOP'
bool = False

if player == 0:
    for i, p in enumerate(w):
        lista = ameaca_ativa(message, p)
        if 'E' in lista:
            list_message = list(message)
            if p in list_message:
                bool = True
                list_message[message.index('E')] = p
                list_message[message.index(p)] = 'z'
                message = ''.join(list_message)
```

Excerto de Código 3.3: Exerto da Função de Xeque-Mate

Capítulo

4

Conclusões e Trabalho Futuro

4.1 Introdução

Esta secção contém uma breve análise relativa aos objetivos cumpridos, ao que poderia ser feito para um trabalho futuro.

4.2 Conclusões e Trabalho Futuro

Neste trabalho foi possível consolidar os conhecimentos adquiridos em contexto de aula, nomeadamente:

- Pesquisa em árvores, em contexto de criação de bots;
- Função Objetiva para ajudar a pesquisa de árvores;
- Aprimoramento da linguagem python.

A melhoria para este *bot* era criar uma função de defesa, uma função que caso o rei estivesse em perigo, esta usaria outras peças para o proteger obrigatoriamente, ao invés de ser feita através da avaliação de árvores.

Bibliografia

- [1] Wikipedia
https://en.wikipedia.org/wiki/Chess_piece_relative_value#Hans_Berliner's_system
- [2] FreeCodeCamp
<https://www.freecodecamp.org/portuguese/news/um-guia-passo-a-passo-para-criar-uma-ia-de-xadrez-simples/>