



UNIVERSIDADE DE ÉVORA

Trabalho Final de Inteligência Artificial,
Jogo "Ouri"

Yaroslav Kolodiy 139859 Eduardo Medeiros 139873

Junho, Ano Letivo 2019/2020

Inteligência Artificial

Prof. Paulo Quaresma

Índice

1	Introdução	3
2	Algoritmos abordados	4
2.1	Minimax	4
2.2	Corte alpha-beta	4
3	Torneio	5
3.1	Resultados	5
3.2	Justificação	6
4	Execução do Programa	7
4.1	Respostas	7
5	Anexos	8
5.1	Anexo 1 - main.py	8
5.2	Anexo 2 - constants.py	12
5.3	Anexo 3 - utils.py	13
5.4	Anexo 4 - minimax.py	15
5.5	Anexo 5 - alphabeta.py	17

1 Introdução

Neste trabalho propusemo-nos a implementar um jogador inteligente para o Jogo de Ouri.

Para tal recorreremos à linguagem Python, que apesar de não ter sido a linguagem abordada nas aulas, é uma linguagem com a qual o grupo se sente confortável.

No que diz respeito à implementação propriamente dita, foram tidos em conta os algoritmos explorados nas aulas e os respectivos pontos positivos e negativos. Tendo isso em conta, o grupo escolheu os dois que achou mais adequados para a situação dada. Os detalhes das implementações serão referidos mais adiante.

2 Algoritmos abordados

Tendo em conta os algoritmos lecionados ao longo da disciplina, o grupo decidiu escolher dois presente no capítulo "Jogos de 2 jogadores". Tendo sido os eleitos os seguintes:

2.1 Minimax

Usando o resultado de uma função de heurística, i.e. uma função que devolve o valor de um determinado estado, que neste caso o representa o jogador que está a ganhar, o algoritmo vai tentar fazer com que o valor do estado seja o melhor possível para um dado jogador. Faz isto maximizando o valor a para as jogadas de um jogador, e minimizando para as jogadas do outro, pois um número positivo simboliza a vitória do primeiro e um negativo a vitória do segundo. Zero simboliza o empate.

Este algoritmo foi escolhido pelo grupo pela sua implementação acessível e pela sua eficácia demonstrada noutros problemas de decisão previamente abordados ao longo da disciplina.

2.2 Corte alpha-beta

Analogamente ao algoritmo anterior, também este recorre a uma função de heurística para decidir que jogada efectuar. Apesar de, mais uma vez, como o algoritmo anterior, este escolher a melhor jogada para um dado jogador, este possui a propriedade de descartar alguns ramos da árvore de pesquisa. Um nó é descartado caso o pai do mesmo simbolize um pior jogada, relativamente a outra já descoberta, para o dado jogador. Assim o algoritmo, no mesmo espaço de tempo, consegue atingir profundidades superiores e ter mais informação sobre as possíveis jogadas.

De acordo com o entendimento acerca do Jogo de Ouri, o grupo decidiu que ambos os algoritmos irão sempre escolher a primeira melhor jogada encontrada.

3 Torneio

3.1 Resultados

Jogador 0 - Algoritmo minimax

Jogador 1 - Algoritmo corte alpha-beta

No jogos 1, 3 e 5, o jogo foi iniciado pelo Jogador 1 enquanto que nos 2, 4 e 6 foi iniciado pelo Jogador 0.

A cada par de jogos, o tempo permitido para cada resposta foi aumentando. Tendo sido os tempos para cada par de jogos os seguintes:

- Jogos 1 e 2 - 5s
- Jogos 3 e 4 - 15s
- Jogos 5 e 6 - 30s

Resultados dos jogos:

1. Jogador 0: 9
Jogador 1: 26
2. Jogador 0: 9
Jogador 1: 32
3. Jogador 0: 12
Jogador 1: 29
4. Jogador 0: 4
Jogador 1: 27
5. Jogador 0: 0
Jogador 1: 26
6. Jogador 0: 7
Jogador 1: 35

3.2 Justificação

Observando os resultados dos jogos efectuados, concluimos que a profundidade de pesquisa do algoritmo tem influência nos resultados dos jogos. Podemos concluir isto, pois como o corte alpha-beta descarta alguns nós, consegue atingir profundidades superiores às do minimax no mesmo intervalo de tempo. De tal modo este consegue ter uma visão mais abrangente das possíveis jogadas, não apenas das suas como também das do seu oponente. Como possui mais informação para efetuar as escolhas, acaba por fazer jogadas melhores às do seu oponente levando, nestes casos, sempre à vitória.

4 Execução do Programa

Nota Prévia: Caso o output do programa seja pretendido de uma forma mais agradável, devem ser instaladas as dependências presentes no Pipfile e de seguida correr os comandos dentro do virtual environment.

Para correr o programa de acordo com o pretendido, i.e. para dar a resposta a uma determinada pergunta do enunciado, devem ser tido em contas as seguintes flags:

-h, --help	show this help message and exit
-p, --first	Opponent (PC) plays first
-s, --second	Opponent (PC) plays play second
-t, --board	Displays the board during the game
-r, --answer	Displays a single an answer
-d, --level LEVEL	Choose the level of the game. a for 5s, b for 15s and c for 30s

4.1 Respostas

- Versão normal:
 1. `python3 main.py -d LEVEL -r`
 2. `python3 main.py -d LEVEL` (se o jogador quiser jogar em segundo usar a flag -p)
- Versão extendida:
 1. O algoritmo alternativo implementado foi corte alpha-beta.
 2. (a) Os comandos são os seguintes:
(b) `python3 main.py -d LEVEL -s` ou `-p`
 3. Respondido em 3.
 4. O código enviado já vai pronto para o troneio, no entanto, fica explicito que a versão escolhida é a do algoritmo corte alpha-beta.

5 Anexos

5.1 Anexo 1 - main.py

```
1 import time
2 from random import randint
3 import argparse
4
5 from utils import *
6 from constants import *
7 import copy
8 from minimax import minimax
9 from alphabeta import alphabeta
10
11 from sys import setrecursionlimit
12
13
14 def print_scores(state: dict):
15
16     if state['player_1'] > state['player_0']:
17         print("PLAYER_1_WON!")
18     elif state['player_0'] > state['player_1']:
19         print("PLAYER_0_WON!")
20     else:
21         print("IT'S_A_DRAW!")
22
23     print("SCORES: _P_0:", state['player_0'], "_P_1:",
24           state['player_1'])
25
26 if __name__ == '__main__':
27
28     parser = argparse.ArgumentParser(prog='IA_final',
29                                     description='Final_project,_OUR_game')
30
31     parser.add_argument('-p, --first', action='
32         store_true', dest='first', help='To_play_first',
33                             required=False)
34     parser.add_argument('-s, --second', action='
35         store_true', dest='second', help='To_play_second',
36                             required=False)
```



```

33         required=False)
34 parser.add_argument( '-t, --board', action='
    store_true', dest='board', help='To display the
    board',
35         required=False)
36 parser.add_argument( '-r, --answer', action='
    store_true', dest='singleAnswer', help='To
    display a single an answer',
37         required=False)
38 parser.add_argument( '-d, --level', type=str, dest='
    level',
39         help='The level of the game, a
    -> 5s, b-> 15s and c-> 30s
    ',
40         required=True)
41
42 args = parser.parse_args()
43
44 setrecursionlimit(pow(10, 8))
45
46 # initial_state = {
47 #     'player_1': player1 score,
48 #     'player_0': player0 score,
49 #     'line_1': [1, 2, 3, 4, 5, 6], mirrored view
    of the opponents side of the board
50 #     'line_0': [1, 2, 3, 4, 5, 6],
51 # }
52 initial_state = {
53     'player_1': 0,
54     'player_0': 0,
55     'line_1': [4, 4, 4, 4, 4, 4],
56     'line_0': [4, 4, 4, 4, 4, 4],
57 }
58
59 print_state(initial_state)
60 tab = copy.deepcopy(initial_state)
61 player0_plays = 0
62 player1_plays = 0
63
64 if args.singleAnswer:
65

```

```

66         if args.level == 'a':
67             level = FIVE_SECONDS_ALPHABETA_SA
68         elif args.level == 'b':
69             level = FIFTEEN_SECONDS_ALPHABETA_SA
70         elif args.level == 'c':
71             level = THIRTY_SECONDS_ALPHABETA_SA
72
73         start = time.time()
74
75         pos = alphabeta(tab, level)
76
77         end = time.time()
78         print('Evaluation time: {}s'.format(round(end -
79             start, 7)))
80
81         tab = play(tab, pos, PLAYER_1)
82
83         print_state(tab)
84         print("P_1_PLAYED: ", pos + 1)
85     else:
86
87         if args.level == 'a':
88
89             level = FIVE_SECONDS_ALPHABETA
90             m_level = FIVE_SECONDS_MINIMAX
91
92         elif args.level == 'b':
93
94             level = FIFTEEN_SECONDS_ALPHABETA
95             m_level = FIFTEEN_SECONDS_MINIMAX
96
97         elif args.level == 'c':
98
99             level = THIRTY_SECONDS_ALPHABETA
100             m_level = THIRTY_SECONDS_MINIMAX
101
102         if args.first:
103             p = 1
104         elif args.second:
105             p = 0

```

```

106         else:
107             p = 0
108
109         while not final_state(tab):
110
111             if p % 2 == 0:
112                 pos = int(input(">")) - 1
113
114                 print("P_0_PLAYED:", pos + 1)
115
116             else:
117
118                 start = time.time()
119
120                 if player1_plays == 0:
121                     pos = randint(0, 5)
122                     player1_plays += 1
123                 else:
124                     pos = alphabeta(tab, level)
125
126                 end = time.time()
127                 print('Evaluation_time: {}s'.format(
128                     round(end - start, 7)))
129
130                 print("P_1_PLAYED:", pos + 1)
131
132             tab = play(tab, pos, p % 2)
133
134             if args.board:
135                 print_state(tab)
136             p += 1
137
138         print_state(tab)
139         print_scores(tab)

```

5.2 Anexo 2 - constants.py

```
1 MAX_CAPTURE = 3
2 MIN_CAPTURE = 2
3 LINE_SIZE = 6
4 MAX_SCORE = 25
5 PLAYER_0 = 0
6 PLAYER_1 = 1
7 FIVE_SECONDS_MINIMAX = 7
8 FIFTEEN_SECONDS_MINIMAX = 8
9 THIRTY_SECONDS_MINIMAX = 8
10 FIVE_SECONDS_ALPHABETA_SA = 9
11 FIFTEEN_SECONDS_ALPHABETA_SA = 10
12 THIRTY_SECONDS_ALPHABETA_SA = 11
13 FIVE_SECONDS_ALPHABETA = 10
14 FIFTEEN_SECONDS_ALPHABETA = 11
15 THIRTY_SECONDS_ALPHABETA = 12
```

5.3 Anexo 3 - utils.py

```
1 from math import inf
2
3 from constants import LINE_SIZE, PLAYER_1, PLAYER_0
4 from utils import final_state as is_final, play,
   pos_is_playable
5
6
7 def minimax(state: dict, depth: int):
8     to_play = -1
9     value = -inf
10
11     for x in range(0, LINE_SIZE):
12
13         if pos_is_playable(state, x, PLAYER_1):
14             played = play(state, x, PLAYER_1)
15             game_value = minimizer(played, depth)
16             if game_value > value:
17                 value = game_value
18                 to_play = x
19
20     return to_play
21
22
23 def maximizer(state: dict, depth: int):
24     if is_final(state) or depth == 0:
25         return heur(state)
26
27     value = -inf
28
29     for x in range(0, LINE_SIZE):
30         if pos_is_playable(state, x, PLAYER_1):
31             played = play(state, x, PLAYER_1)
32             value = max(value, minimizer(played, depth
33                                     - 1))
34
35     return value
36
37 def minimizer(state: dict, depth: int):
```

```

38     if is_final(state) or depth == 0:
39         return heur(state)
40
41     value = inf
42
43     for x in range(0, LINE_SIZE):
44         if pos_is_playable(state, x, PLAYER_0):
45             played = play(state, x, PLAYER_0)
46             value = min(value, maximizer(played, depth
47                         - 1))
48
49     return value
50
51 def heur(state: dict):
52     return state['player_1'] - state['player_0']

```

5.4 Anexo 4 - minimax.py

```
1 from math import inf
2
3 from constants import LINE_SIZE, PLAYER_1, PLAYER_0
4 from utils import final_state as is_final, play,
   pos_is_playable
5
6
7 def alphabeta(state: dict, depth: int):
8     to_play = -1
9     value = -inf
10    alpha = -inf
11    beta = inf
12
13    for x in range(0, LINE_SIZE):
14
15        if pos_is_playable(state, x, PLAYER_1):
16            played = play(state, x, PLAYER_1)
17            game_value = minimizer(played, depth, alpha
18                                   , beta)
19            if game_value > value:
20                value = game_value
21                to_play = x
22
23    return to_play
24
25 def maximizer(state: dict, depth: int, alpha: float,
26               beta: float):
27     if is_final(state) or depth == 0:
28         return heur(state)
29
30     value = -inf
31
32     for x in range(0, LINE_SIZE):
33         if pos_is_playable(state, x, PLAYER_1):
34             played = play(state, x, PLAYER_1)
35             value = max(value, minimizer(played, depth
36                                           - 1, alpha, beta))
```

```

36         if value >= beta:
37             return value
38
39         alpha = max(alpha, value)
40
41     return value
42
43
44 def minimizer(state: dict, depth: int, alpha: float,
45              beta: float):
46     if is_final(state) or depth == 0:
47         return heur(state)
48
49     value = inf
50
51     for x in range(0, LINE_SIZE):
52         if pos_is_playable(state, x, PLAYER_0):
53             played = play(state, x, PLAYER_0)
54             value = min(value, maximizer(played, depth
55                                         - 1, alpha, beta))
56
57         if value <= alpha:
58             return value
59
60         beta = min(beta, value)
61
62     return value
63
64 def heur(state: dict):
65     return state['player_1'] - state['player_0']

```


5.5 Anexo 5 - alphabeta.py

```
1 from math import inf
2
3 from constants import LINE_SIZE, PLAYER_1, PLAYER_0
4 from utils import final_state as is_final, play,
   pos_is_playable
5
6
7 def alphabeta(state: dict, depth: int):
8     to_play = -1
9     value = -inf
10    alpha = -inf
11    beta = inf
12
13    for x in range(0, LINE_SIZE):
14
15        if pos_is_playable(state, x, PLAYER_1):
16            played = play(state, x, PLAYER_1)
17            game_value = minimizer(played, depth, alpha
18                                   , beta)
19            if game_value > value:
20                value = game_value
21                to_play = x
22
23    return to_play
24
25 def maximizer(state: dict, depth: int, alpha: float,
26               beta: float):
27     if is_final(state) or depth == 0:
28         return heur(state)
29
30     value = -inf
31
32     for x in range(0, LINE_SIZE):
33         if pos_is_playable(state, x, PLAYER_1):
34             played = play(state, x, PLAYER_1)
35             value = max(value, minimizer(played, depth
36                                          - 1, alpha, beta))
```

```

36         if value >= beta:
37             return value
38
39         alpha = max(alpha, value)
40
41     return value
42
43
44 def minimizer(state: dict, depth: int, alpha: float,
45             beta: float):
46     if is_final(state) or depth == 0:
47         return heur(state)
48
49     value = inf
50
51     for x in range(0, LINE_SIZE):
52         if pos_is_playable(state, x, PLAYER_0):
53             played = play(state, x, PLAYER_0)
54             value = min(value, maximizer(played, depth
55                                     - 1, alpha, beta))
56
57     if value <= alpha:
58         return value
59
60     beta = min(beta, value)
61
62     return value
63
64 def heur(state: dict):
65     return state['player_1'] - state['player_0']

```