Relatório T2

INF01048 - Inteligência Artificial - UFRGS

João Davi Martins Nunes 00285639 - Turma B
Patrick Dornelles da Silva Vieira 00242320 - Turma B
Tiago Ehlers Binz 00279792 - Turma B

A função de parada usada foi o tempo. Usando a profundidade incremental a cada incremento temos uma melhor jogada, e, se o tempo acaba, interrompemos a busca e retornamos a melhor jogada até o momento.

Para a decisão das jogadas, foram feitas 5 funções de avaliação entretanto usadas só 4, pois uma não deu muito certo na prática. Essas funções são:

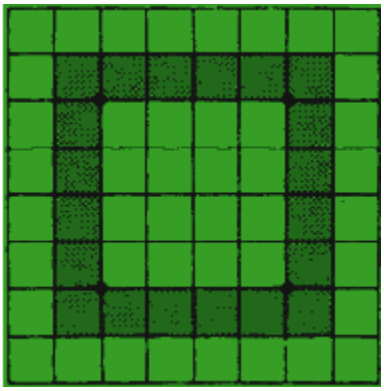
1. Caso o movimento é um dos cantos, recebe prioridade;

```
# se o movimento é uma das bordas recebe prioridade
def is_move_on_corner(self, move):
    if move == self.top_left \
            or move == self.top_right \
            or move == self.bottom_left \
            or move == self.bottom_right:
        return 5
    return 0
```

2. Caso haja a possibilidade numa das bordas, o algoritmo é encorajado a fazer o movimento;

```
def is_move_in_borders(self, move):
    if move in self.borders_zone:
        return 3
    return 0
```

3. Caso a zona de perigo é a zona anterior a borda do tabuleiro, o algoritmo é desencorajado a escolher esse movimento;



```
def is_move_in_danger_zone(self, move):
    if move in self.danger_zone:
        return -2
    return 0
```

4. Avaliando os próximos movimentos que o adversário pode fazer e se o oponente não tem ou tem poucas jogadas, encorajamos que o algoritmo escolha esse movimento. Contudo, se existem momentos bons para o adversário, penalizamos esse movimento;

```python
def check_opponent_next_move(self, a_board, move, color):
    score = 0
    a_board.process_move(move, color)

    opponent_next_moves = a_board.legal_moves(a_board.opponent(color))

    if len(opponent_next_moves) < 3:  #se oponente não tem poucos  movimentos é bom
        score += 5

    if len(opponent_next_moves) < 1: #se oponente não tem mais movimentos é bom
        score += 10

    for opponent_move in opponent_next_moves:
        if self.is_move_on_corner(opponent_move):  #se oponente pode usar canto é ruim
            score -= 2
            return score

        if self.is_move_in_borders(opponent_move):  #se oponente pode usar borda é ruim
            score -= 1
            return score
    return score
```

5. (heurística que não deu certo na prática e que não foi utilizada) Penalização por movimentos que se afastam do centro.

```python
# movimento que se afasta do centro recebe uma penalidade
def off_center_move(self, move):
    center = 3.5
    penalty1 = move[0] - center
    penalty2 = move[1] - center
    return (abs(penalty1) + abs(penalty2)) * -1
```

A execução feita em cada passo é essa descrita abaixo. A cada avaliação adicionamos ou removemos pontos daquele movimento.

```
def move_eval(self, a_board, move, color):
    score = 0
    score += self.is_move_on_corner(move)
    #score += self.off_center_move(move) # não deu muito certo
    score += self.is_move_in_danger_zone(move)
    score += self.is_move_in_borders(move)
    score += self.check_opponent_next_move(a_board, move, color)
```

O resultado da avaliação do movimento é somado ao resultado do min ou do max.

```
def alfa_beta_max(self, a_board, alfa, beta, max_depth):
    if self.cut_test(a_board, alfa, beta, max_depth):
        score = self.__get_board_score(a_board)
        return score

    for move in a_board.legal_moves(self.my_color):
        imaginary_board = board.from_string(str(a_board))
        imaginary_board.process_move(move, self.my_color)

        self.node_expands_counter += 1

        v = self.alfa_beta_min(imaginary_board, alfa, beta, max_depth - 1)
        v += self.move_eval(imaginary_board, move, self.my_color)

        alfa = max(v, alfa)
        if beta < alfa:
            self.pruning_counter += 1
            return alfa

    return alfa
```

```
def alfa_beta_min(self, a_board, alfa, beta, max_depth):
    if self.cut_test(a_board, alfa, beta, max_depth):
        score = self.__get_board_score(a_board)
        return score

    for move in a_board.legal_moves(self.opponent_color):
        imaginary_board = board.from_string(str(a_board))
        imaginary_board.process_move(move, self.opponent_color)

        self.node_expands_counter += 1
        v = self.alfa_beta_max(imaginary_board, alfa, beta, max_depth - 1)
        v += self.move_eval(imaginary_board, move, self.opponent_color)

        beta = min(v, beta)
        if alfa >= beta:
            self.pruning_counter += 1
            return beta
    return beta
```

Referências:

https://www.ultraboardgames.com/othello/tips.php