Artificial Intelligence and Knowledge Engineering Laboratory

Assignment 3

Game Playing Algorithm

Reversi

Each **reversi piece** has a black side and a white side. On your turn, you place one piece on the board with your color facing up. You must place the piece so that an opponent's piece, or a row of opponent's pieces, is flanked by your pieces. All of the opponent's pieces between your pieces are then turned over to become your color.

The object of the game is to own more pieces than your opponent when the game is over. The game is over when neither player has a move. Usually, this means the board is full.

The game is started in the position shown below on a reversi board consisting of 64 squares in an 8x8 grid.

A move consists of placing one piece on an empty square.

The game ends when:

- One player wins, by making his color dominant on the board.
- The players agree to finish the game (as a resignation, or a draw).

Pseudocode of minimax algorithm

```
function minimax(position, depth, maximizingPlayer)
    if depth == 0 or game over in position
        return static evaluation of position
    if maximizingPlayer
        maxEval = -infinity
        for each child of position
            eval = minimax(child, depth - 1, false)
            maxEval = max(maxEval, eval)
        return maxEval
    else
        minEval = +infinity
        for each child of position
            eval = minimax(child, depth - 1, true)
            minEval = min(minEval, eval)
        return minEval
```

Pseudocode of alpha-beta algorithm

```
function ALPHA-BETA-SEARCH($state$) returns an action
    $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
    return the $action$ in ACTIONS($state$) with value $v$
```

```
function MAX-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS($state$) do
        $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha, \beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ MAX($\alpha$, $v$)
    return $v$
```

```
function MIN-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS($state$) do
        $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$) , $\alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ MIN($\beta$, $v$)
    return $v$
```

Evaluation function

Returns a positive value when the player wins.

Returns zero when there is a draw.

Returns a negative value when the opponent wins.

Evaluation checks whether the player has more pieces than its opponent, how many player pieces are located on the edges (edge bricks are important in order to control flipping of the opponent's pieces), how many player pieces are in the corners which are also very important strategically.

Also it evaluates the corner pieces, adds if you select them, otherwise subtracts. Moreover, check if someone has selected 2 neighbour corners.

Finally, evaluation score for 1st player = outcome of pieces * 2 + corners * 1.5 + edges * 1.2

for 2nd player = outcome of pieces * 3 + corners * 1.5 + edges * 1.2

Move generation function

The game is played by giving input in the shape of x,y coordinates for the valid moves, using the common format of x∈(a-h), y∈(1,8), for example: d3, h1, a2, f8, h7 etc.

That’s why, first of all, we verify if we are not going outside of board.

The possible moves are those which are located near the pieces.

As evaluation f-n calculates stands for the player with the most pieces next turn, then this move is

the most optimal for current state.

After each step we check the action : a list of tuples as coordinates for the valid moves for the current player and select the best one for ai-player and for real-player the piece which he wants.

The game is over when there are no more moves left, and the winner will be determined by a count of pieces, who have received more than opponent then he is winner.

Experiments

M = 50

A vs B, where white pieces starts first.

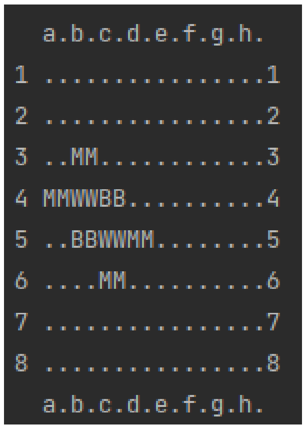
Regarding 1st part by comparing game where we have 2 ai players, white has win 39/50.

Also there is an option to play games as ai vs random (random selects the moves), ai has won 50/50

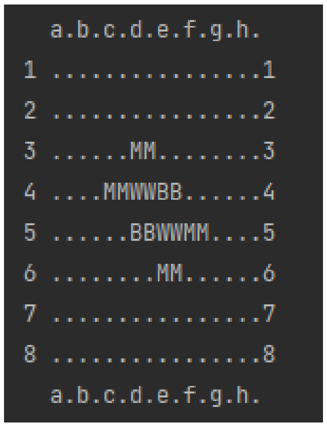
If we take a look the game with real player vs ai player, where the human starts the game he has 6 wins out of 50. (of course it depends where the start pieces are located, i.e if pieces stand at the any corner or edge of the board, human has already some advantages over an opponent, as well depends on the depth, as bigger it is as smaller probability to win)

Examples of different starting positions :

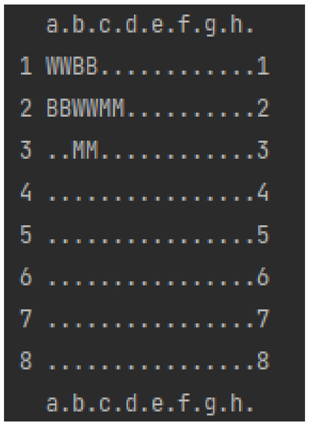
WW BB means simply white, black



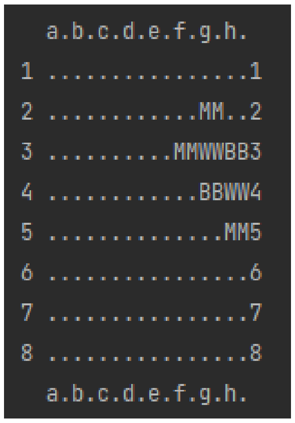
```
  a.b.c.d.e.f.g.h.
1 ................1
2 ................2
3 ..MM............3
4 MMWWBB..........4
5 ..BBWWMM........5
6 ....MM..........6
7 ................7
8 ................8
  a.b.c.d.e.f.g.h.
```



```
  a.b.c.d.e.f.g.h.
1 ................1
2 ................2
3 ......MM........3
4 ....MMWWBB......4
5 ......BBWWMM....5
6 ........MM......6
7 ................7
8 ................8
  a.b.c.d.e.f.g.h.
```

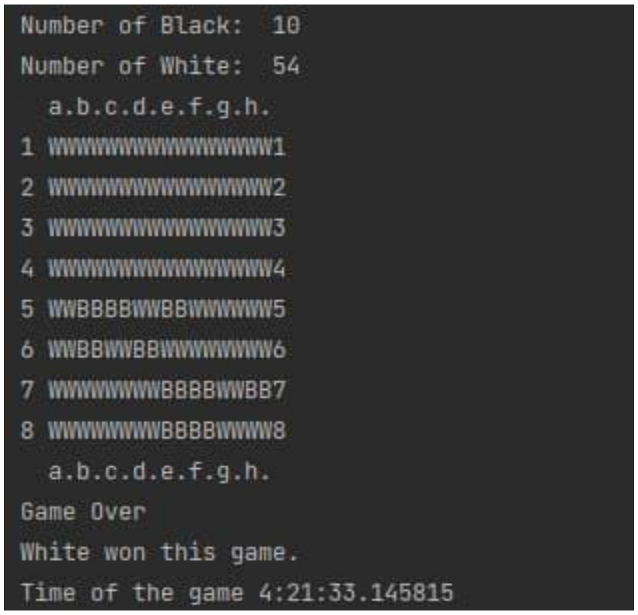


```
  a.b.c.d.e.f.g.h.
1 WWBB............1
2 BBWWMM..........2
3 ..MM............3
4 ................4
5 ................5
6 ................6
7 ................7
8 ................8
  a.b.c.d.e.f.g.h.
```



```
  a.b.c.d.e.f.g.h.
1 ................1
2 ............MM..2
3 ..........MMWWBB3
4 ............BBWW4
5 ..............MM5
6 ................6
7 ................7
8 ................8
  a.b.c.d.e.f.g.h.
```

Time for depth = 5 is 4 hours 21 minutes 33 seconds



```
Number of Black:  10
Number of White:  54
  a.b.c.d.e.f.g.h.
1 WWWWWWWWWWWWWWWW1
2 WWWWWWWWWWWWWWWW2
3 WWWWWWWWWWWWWWWW3
4 WWWWWWWWWWWWWWWW4
5 WWBBBBWWBBWWWWWW5
6 WWBBWWBBWWWWWWWW6
7 WWWWWWWWBBBBWWBB7
8 WWWWWWWWBBBBWWWW8
  a.b.c.d.e.f.g.h.
Game Over
White won this game.
Time of the game 4:21:33.145815
```

For d > 2 program was ran only once, to see how many time it requires, the algorithm was used minimax with alpha-beta pruning on, cause with switched off it would be taken much more time.

For d < = 2 program was executed 10 times.

d	t_avg(d) (alpha-beta turn on)	T_avg(d) (alpha-beta turn on)	t_avg(d) (alpha-beta turn off)	T_avg(d) (alpha-beta turn off)
1	1.08 sec	1 min 5 secs	4.7 sec	4 min 46 sec
2	4.2 sec	4 min 18 sec	13.1 sec	13 min 21 sec
3	49.32 sec	49 min 32 sec		
4	2 min	1h 59 min 42 sec		
5	4 min 21 sec	4h 21 min 33 sec		

Q(A(d_n)/A(d_{n-1}) and Q(A(d_max)/A(d_n))for n in (1,2,...d_max -1)

Minimax with alpha-beta	Minimax
Q(A(5)/A(4)) = 2.1	Q(A(2)/A(1)) = 2,8
Q(A(4)/A(3)) = 2.43	
Q(A(3)/A(2)) = 11,74	
Q(A(2)/A(1)) = 3,88	

Branch factor strongly depends on the current state of game.

The highest value of branch factor for d : 1 , 2

d	Branch factor
1	145
2	1056

From the results above, we can summary up and make conclusion.

The alpha-beta algorithm is much better than minimax, it was obvious as it is modified version of minimax algorithm.

The difference is obtained in time complexity, the reason of that is simply, as is a procedure to reduce the amount of computation and searching during minimax.

With best wishes,
Vladyslav Gavryliuk.