Reversi Playing Problem

PROJECT 1 REPORT OF CS303 ARTIFICIAL INTELLIGENCE

高圣迪 Gao Shengdi

SUSTECH

Table of Content

1.Preliminaries

Reversi Playing is the process of selecting the best landing place on the board. The goal is to get more points than your opponent when all the points on the board are occupied. The study of this problem can help people improve his or her reversi skill and have fun when he or she wants to play reversi but lonely.

1.1**Software & Hardware**

This project is written in Python with editor PyCharm Community Edition 2020.2.1 x64. Then main testing platform is Windows 10 家庭中文版 with Inter® Core(TM) i5-9300 CPU @2.40GHz 2.40Ghz.

1.2**Algorithm**

This project mainly involves an AI algorithm called Minmax Tree[i], by predicting where your opponent would go to get best choice.

The bubble sort algorithm to help me to prune my Minmax tree.

Trapezoidal scanning algorithm to help me get stable discs.

2.Methodology

In this part, I will introduce the representation of the project, the structure of my program and some details of algorithms I used.

2.1Representation

2.1.1 Notation:

C/X-squares[ii]**:** C-squares are located at a2, a7, b1, b8, g1, g8, h2 and h7 on the board. X-squares are located at b2, b7, g2, g7 on the board. These positions should be careful to occupy.

Liberty: The count of places where you can go.

Stable discs: The point that will never be flipped.

Potential Liberty: The count of whether the area around the drop is occupied. This value is bad when it is big.

Find_move (board, color): Get the list of all positions this color can go in the board

S (board, color): The count of *Stable discs* with color on the board.

L (board, color): The count of *Liberties* with color on the board.

P (board, color): The count of *Potential Liberties* with color on the board.

E (board, color): The *evaluation* of the board for this color.

2.1.2 Data Structure:

Board: A 8×8 arrays which means the chessboard. When the position is occupied by black chess, it should be -1, 1 means white, 0 means not occupied.

Candidate_list: A list of the all available positions with the most valuable one in the last of the list.

Weighting: A 8×8 arrays which means each value of the position on the board. For example, the corner has the value of 500 because it is important and the C-squares has the value of -50 because it should be pay attention to occupy.

2.2Architecture

2.2.1 Flow Chart of Whole Program

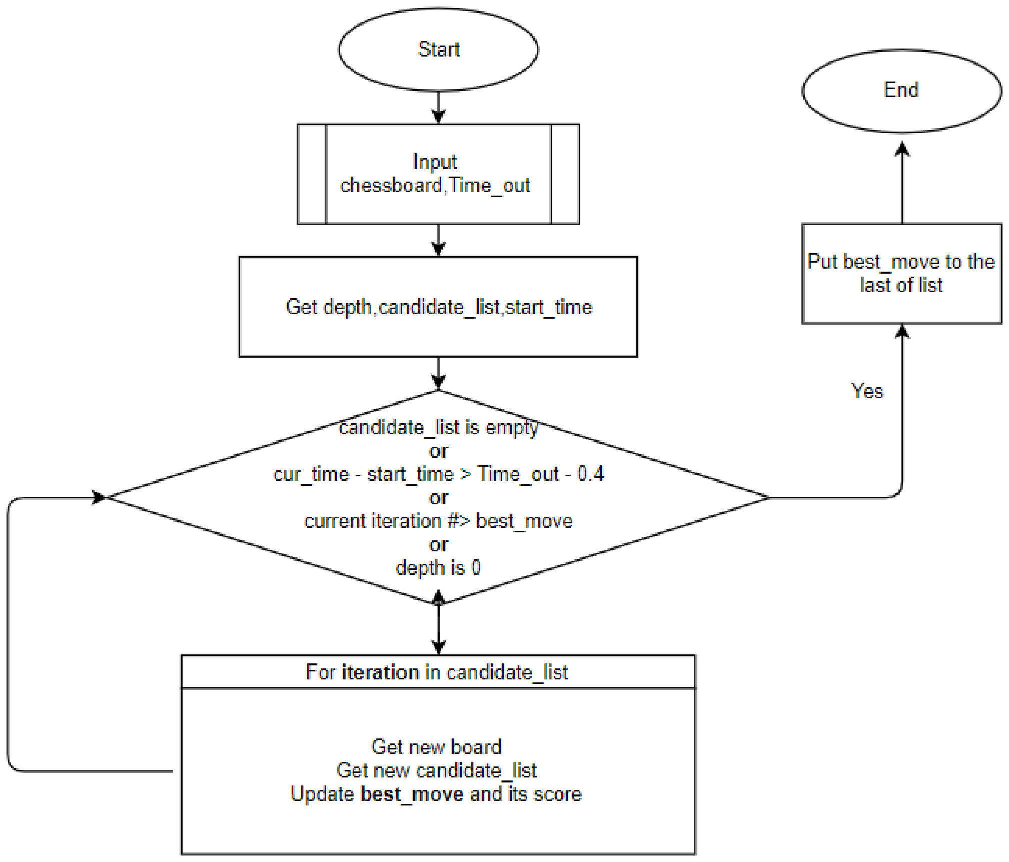
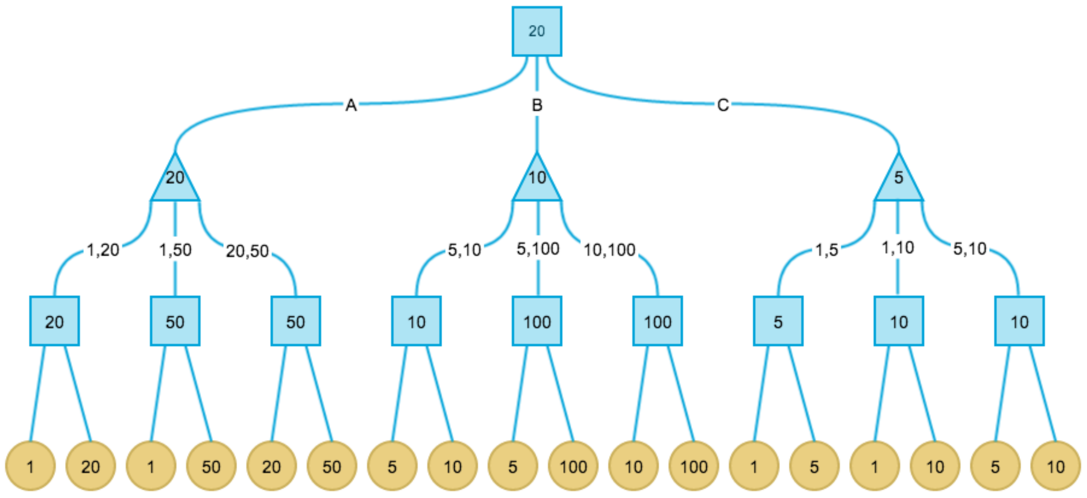

Start
Input chessboard,Time_out
Get depth,candidate_list,start_time
candidate_list is empty or cur_time - start_time > Time_out - 0.4 or current iteration #> best_move or depth is 0
Yes
Put best_move to the last of list
End
For iteration in candidate_list
Get new board
Get new candidate_list
Update best_move and its score

Figure 1 Flow Chart of Whole Program

2.2.2 Search Strategy

The victory condition of Reversi is when all positions are occupied, the side with more pieces will win. The search strategy is to win the game at last, so *greedy algorithm* is not a good algorithm. In order to get the final victory, AI should think about the chessboard more times.

So, I use Minmax Search with Alpha-Beta pruning and a certain depth. Actually, Minmax Search is a DFS application. There is a figure to show how it works.



iii

Figure 2 Schematic Diagram of Minmax

At first, you have three choices and you want to achieve the max score. By DFS, after spanning the first node, it's your opponent's turn, he/she wants to make your score as min as possible, so he/she will choose min choice in these three nodes. And, let's see the last node, it's your turn again, you want the best node, so you choose 20 in (1,20), 50 in (1,50) and 50 in (20,50). And your opponent will choose the min(20,50,50) which is 20. In the same way, actually, you will get 3 nodes which has 20,10,5 respectively and the AI will choose the first node as the best_move!

Then I will introduce the Alpha-Beta pruning. There is also a figure to show how Alpha-Beta pruning works.

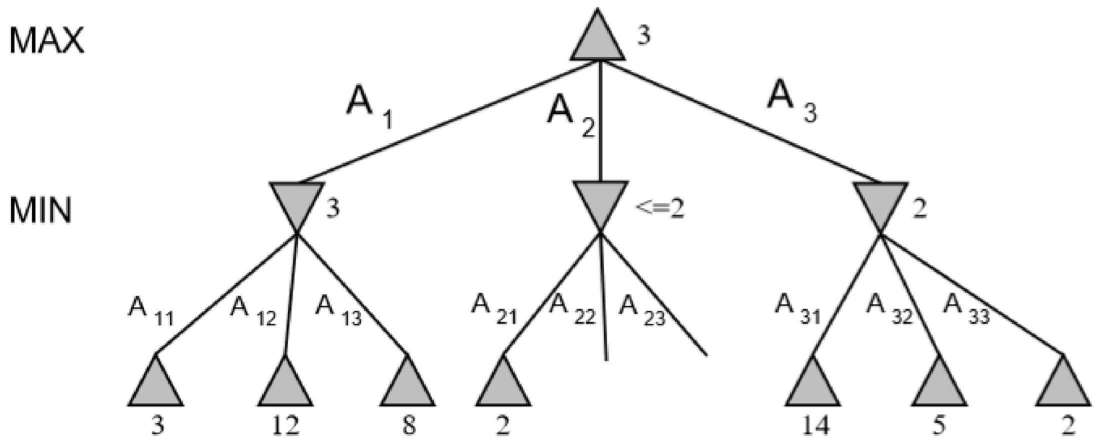


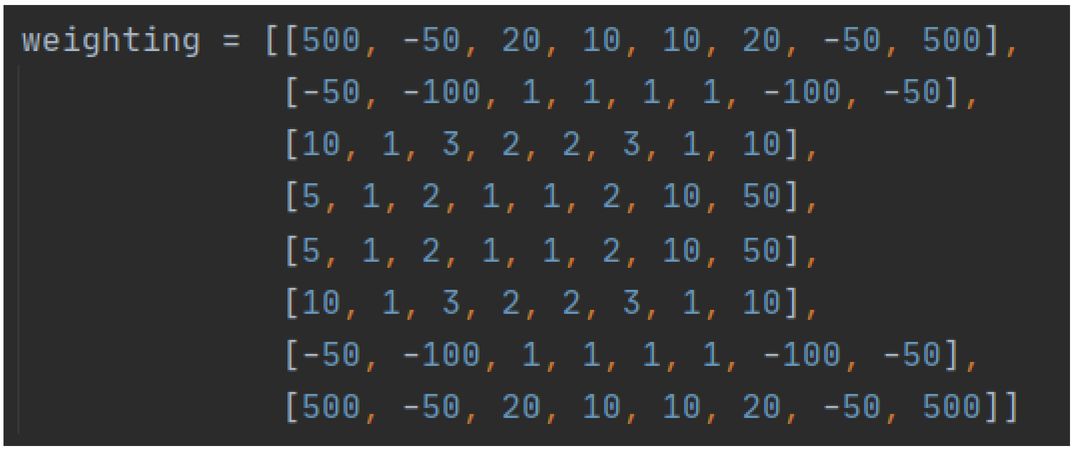
Figure 3 Schematic Diagram of Alpha-Beta

Alpha represents the lower limit, which is the minimum evaluation value that a node can reach; Beta represents the upper limit, which is the maximum situation estimate that a node can reach. For a MIN node, it will make its Beta value as small as possible. For a MAX node, it will make its Alpha as large as possible. Take layer 1 as an example. Because we have a node in layer 2 with the value 3 and a node spanned from the second node in layer 3 with the value 2. In order to get the min value so although we span A_{22} and A_{23}, the value of A_2 is <=2, A_1=3 > A_2, so there is no need to span A_{22} and A_{23}.

2.2.3 Evaluation Function

Evaluation Function is to evaluate the vale the drop brings to the situation. For a good AI, the evaluation function should be diversified. In this section, I will only introduce one—“weighting”. For more detail, please read **2.3.5**

The use of weighting has been introduced in **2.1.1**. Now I want to show the whole weighting map in my AI.



```
weighting = [[500, -50, 20, 10, 10, 20, -50, 500],
             [-50, -100, 1, 1, 1, 1, -100, -50],
             [10, 1, 3, 2, 2, 3, 1, 10],
             [5, 1, 2, 1, 1, 2, 10, 50],
             [5, 1, 2, 1, 1, 2, 10, 50],
             [10, 1, 3, 2, 2, 3, 1, 10],
             [-50, -100, 1, 1, 1, 1, -100, -50],
             [500, -50, 20, 10, 10, 20, -50, 500]]
```

Figure 4 Weighting of Chessboard

2.3Detail of Algorithm

2.3.1 Find_move

find_move is used to get all available move of the color on the board. It contains two sub functions “check (board, i, j, color)” and “nextCheck (board, i, j, color)”.

First, I have defined two arrays.

```
row = [0, -1, -1, -1, 0, 1, 1, 1]
col = [-1, -1, 0, 1, 1, 1, 0, -1]
```

Then the find_move pseudo code is:

```
Function find_move(board, color)
Available_move = [ ]
For i = from 1 to 8
    For j = from 1 to 8
        If board(i, j) is occupied Then
            Continue
        If  check (board, i, j color) Then
            Add (i, j) to Avaiable_move
```

Check and nextcheck pseudo code is:

```
Function check(board, i, j, color)
For k = from 1 to 8
    If 0 <= i + row[k] < 8 and 0 <=col[k] <8 Then
        x, y= i + row[k], j + col[k]
        If board[x][y] is opponent color Then
            return nextCheck (board, x, y, k, color)
return False

Function nextCheck(board, i, j, color)
If 0 <= i + row[k] < 8 and 0 <=col[k] <8 Then
    x, y= i + row[k], j + col[k]
    If board[x][y] is opponent color Then
        return nextCheck (board, x, y, k, color)
    If board[x][y] is color Then
        return True
return False
```

2.3.2 GetDepth

When I was playing reversi by myself, I found that when at the beginning and the end, *Liberty* is not as big as the intermediate stage of the game. Therefore, I set a relatively large value at the beginning and the end. I judge the stage by how many points are occupied.

Then the getDepth pseudo code is:

```
Function getDepth(board)
cnt = 0
For i = from 1 to 8
    For j = from 1 to 8
        If board[i][j] is occupied Then
            cnt += 1
if cnt < 10 Then return 5
else if 10<= cnt < 20 Then return 4
else if 20<= cnt < 45 Then return 3
else if 45<= cnt < 50 Then return 3
else if 50<= cnt < 53 Then return 4
else if 53<= cnt < 58 Then return 5
return 6
```

2.3.3 Sort

“sort” is just a simple bubble sort. It put larger value point in the front of the list, so it can help the minmax tree to prune.

Then the sort pseudo code is:

```
Function sort(avb_move)
For i in avb_move
    For j from i+1 to avb_move
        If weighting[i] < weighting[j] Then
            Exchange i,j
```

2.3.4 Minmax

“minmax” is the most important function in this project. It helps to predict where your opponent would go, and then help the program to make a more sensitive choice. The main idea is you want to get max score while your opponent wants to make you to get min score.

Then the minmax pseudo code is:

```
Function minmax(board,color,alpha,beta,depth)
avb_move = find_move(board,color)
If avb_move is empty Then
  If find_move(board, -color) is empty    #which means end
    Count who has more pieces, return ±INF, best_move
  return -minmax(board,-color,-beta,-alpha,depth)[0],   best_move
If depth is 0, reach the deepest point or time is up
    return E(board,color), best_move
For move in avb_move
    Apply this move and get new_board
    val = -minmax(new_board,-color,-beta,-alpha,depth-1)[0]
    If val > alpha
        update alpha = val
        update best_move = move
        if beta < alpha
            return val, best_move
return alpha,best_move
```

2.3.5 GetWei

"getWei" functions is E(board,color) which I has introduced before. It's the evaluation function. Which is consist of four parts: **weighting**, **Liberty**, **Stable discs** and **Potential Liberty.**

Each part is easy except **Stable discs**[iv]. First, the four corners are obviously stable discs. And from one corner, we can infer more stable discs like a triangle.

○	○	○	○	○	×	○	○
○	○	○	○	×			
○	○	×	×				
○	×						
×							

From the left-top corner to the fifth one in the first line, they are stable discs.

And from the second line, if pieces are the same color, they are stable discs because the top one and the left one is their friends to protect them. So, it can scan to the (5-1) pieces. Based on this method, we can find all stable discs extended from one corner. Repeat this method four times, we find all stable discs. Although It is a rough count, it is enough to act as an important part in our evaluation!

Then the getWei pseudo code is:

```
Function getWei(board,color)
board_weight = 0
For i = from 1 to 8
    For j = from 1 to 8
        If board[i][j] is color Then board_weight += weighting[i][j]
        If board[i][j] is oppo color Then board_weight -= weighting[i][j]
choose_weight = len (find_move of color – find_move of oppo color)
stable_weight = getStable(board,color)
potential_weight = getpotential(board,color)
If cnt < 20 Then
    return 100 * board_weight + choose_weight + 10 * potential_weight
If 20<=cnt<40 Then
    return 0.8 * board_weight + 40 * choose_weight + 80 * stable_weight + 15 *
          potential_weight
Else
    return board_weight + 30 * choose_weight + 100 * stable_weight + 15 *
          potential_weight
```

3.Empirical Verification

3.1Dataset

At first, I use only 10 cases given by our SA to pass the Lab1.

Then in the points race, I begin to search some composition on the web. There are some examples.

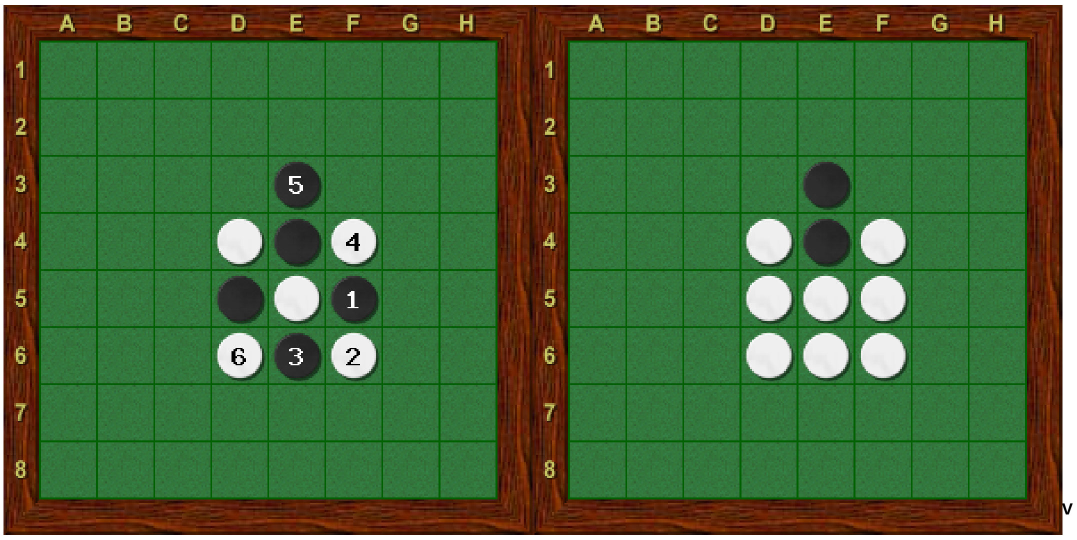
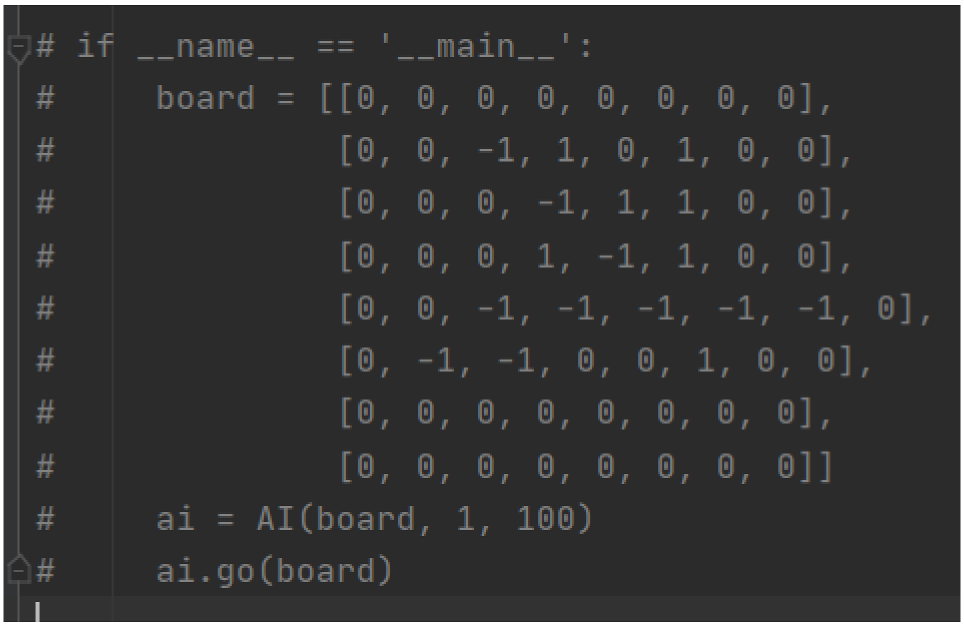


Figure 5 Chimney

I copy these board by myself and execute it on my computer to see if the drop my AI gives is the same as it gives on the website.



```
# if __name__ == '__main__':
#     board = [[0, 0, 0, 0, 0, 0, 0, 0],
#              [0, 0, -1, 1, 0, 1, 0, 0],
#              [0, 0, 0, -1, 1, 1, 0, 0],
#              [0, 0, 0, 1, -1, 1, 0, 0],
#              [0, 0, -1, -1, -1, -1, -1, 0],
#              [0, -1, -1, 0, 0, 1, 0, 0],
#              [0, 0, 0, 0, 0, 0, 0, 0],
#              [0, 0, 0, 0, 0, 0, 0, 0]]
#     ai = AI(board, 1, 100)
#     ai.go(board)
```

Figure 6 Local Test

3.2 **Performance**

The simplest way to evaluate my AI's performance is "rank talks". In the worst case, my AI will timeout and return a answer without any operations on *candidate_list*. So I use *getDepth* to improve this situation. For not sure getDepth will works well all the time, I get the current time and start time, if the time is not enough, AI will return a relatively good answer.

I use 10.20.26.45:8080/whitelog (blacklog) to download the log of the game in order to check timeout and whether the choice on the web and local are the same.

I'm proud of that my friends and I can't defeat my AI.

3.3 **Hyperparameters**

There are 4 parameters in my evaluation function: **weighting**, **Liberty**, **Stable discs** and **Potential Liberty.**

At first, I only use weighting but it doesn't work well. Sometimes, although my AI has occupied all the four corners but lose the game, I introduced liberty and stable discs and regard weighting as important only at the beginning of the game. In the middle of the game, I set the weight of Liberty large, because in this way, my AI has more choices. In fact, it works very well. In the end of the game, I set the weight of stable disc large in order to get more pieces when game over. I also write a function that if the game is over, return ±INF by counting pieces.

Potential Liberty is a magical parameter because actually I don't know how it works. What I know is that it actually works!

3.4 **Result**

I have passed the first check, passing all 10 situations given by SA.

Early in the points race, I was at the bottom of the list because I wrote a wrong minmax tree. Then after I have fixed my minmax tree, my rank raised fast, up to about 20. At the end of the points race, my rank tends to be stable and finally ranked 58th.

In the round robin, I didn't change my code and finally ranked 45th.

3.5 **Analysis**

On the whole, I'm quite satisfied with my project. I think 45th is a fine rank. I think my evaluation function is such a good one because I add potential liberty which has not been known by a lot of people. I got this from book, which I'm proud of.

The disadvantages of my project are I think my code is too "rude". For example, my sort is just a bubble sort and the way to get all available move is not so elegant as well. I should improve these functions to get a better performance.

What I leaned is not only knowledge of algorithms such as minmax tree but also how to deal with a problem without a right answer.

Acknowledgments

I would like to thank TA Zhao Yao for her labs about minmax tree and her book called ”黑白棋指南”.　Also thanks a lot to my SA Wang Zhiyuan for her patient answer to my questions!

References

[i] Flying Machine Studies (2012). An Exhaustive Explanation of Minimax, a Staple AI Algorithm

[ii]节操 uXS(2016) Term in Reversi https://zhidao.baidu.com/question/266480440112195405.html

[iii] 雪山上的小草（2019）Minimax 算法及实例分析 https://blog.csdn.net/m0_37154839/article/details/86767636

[iv] Othella sky(2005) 第二章 角和稳定子 http://soongsky.com/othello/strategy2/ch2.php

[v] Othella sky (2005) WThor Library http://www.soongsky.com/othello/database/