AI Project1 Report

11811620毛尊尧

1.Preliminaries

1.1.Problem Description

Othello is a strategy board game invented in 1883.There are 8*8 pieces called *disks* (often spelled "discs"), which are white on one side and dark on the other. Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color. The object of the game is to hava larger number of discs when no one can move anymore.**The goal of this project, is to build an intelligent expert decision system of Othello.**

Software: Python,Pycharm, Wzebra

Algorithm: Alpha-Beta Pruning, Min-Max Search

1.2.The Applications of Problem.

The two-player board game can be seemed as **complete information and zero-sum game.**

Complete information: the information that both parties have in their decision-making is equal, and there is no information asymmetry.

zero-sum: the sum of the interests of all game parties is zero or a constant, that is, one party has gains and the other party must lose.

Complete information games are applied in many fields.

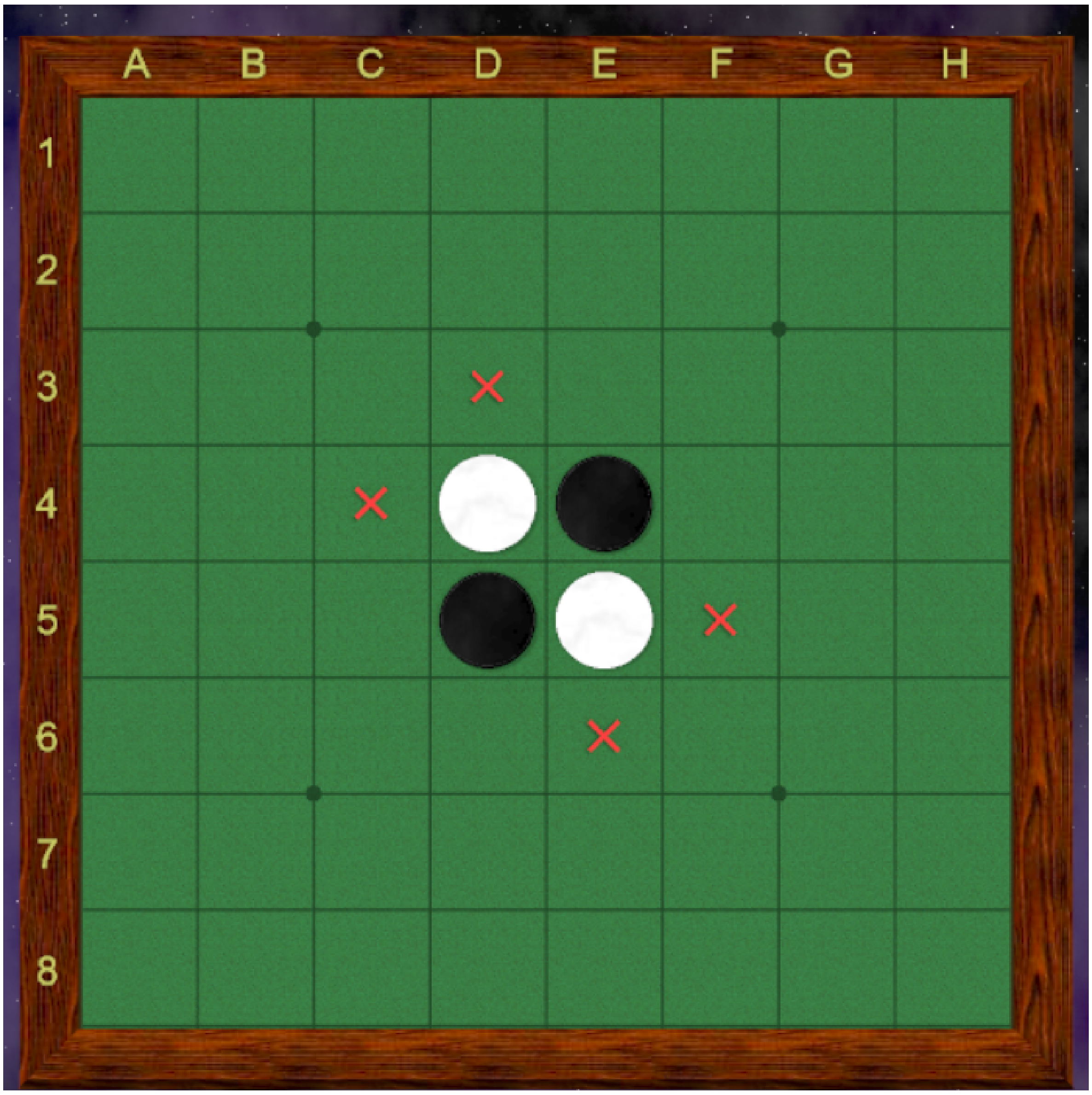
Applications: industrial clusters have unmatched competitive advantages, among which technological innovation advantage is one. However, in the real economic life, it is found that some industrial clusters decline due to the lack of technological innovation ability. In view of this phenomenon, the static game model of complete information can be used to explain it theoretically. Based on the game model and game analysis, it is found that the enterprises in the cluster lack the power of technological innovation in nature, and how to solve the problem of insufficient power of technological innovation in the cluster effectively. This report is help to give a solution model of such problem.

2.Methodology

2.1.Notation

$E_p(board)$: the evaluation function. p is the *abbreviation* of "phase", board is the current board (represent as a two dimensional array).

$Postion$: In othello board. the postion can be expressed as one letter and one number.For example. A2 means the disc in line 2 and column 1.



2.2.Data Structure:

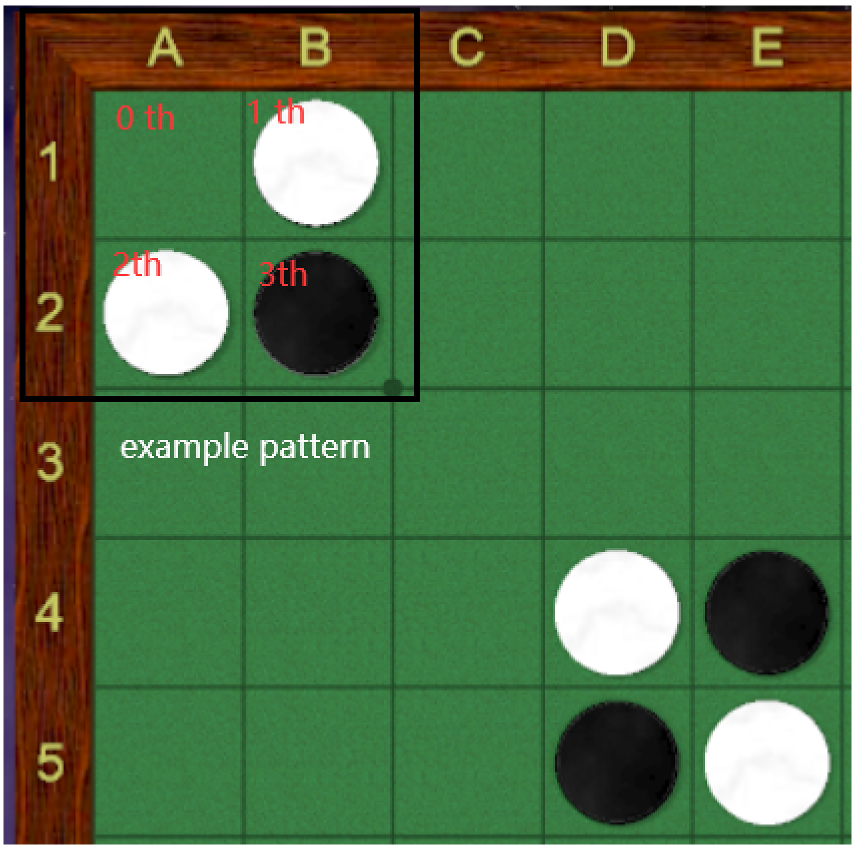
2.2.1.Representation of Board

The chessboard can be represent as 2-dimensional array, and 0 represents black, 1 represents empty, 2 represent white (In judge system, the number is -1,0,1, which can be transformed easily)

2.2.2.Representation of Pattern

In my program, I use **bitmap** to represent the black and white chesses distribution of part of the whole board (**which called pattern**). For example, if we want to enumerate all cases of position (A1,B1,A2,B2), because there are 3 cases for one position: black chess, white chess or no chess on this position. So there are totally $3^4 = 81$ possible situations. In my program, I use 0 for black chess, 1 for empty, and 2 for white, so that we can represent every position uniquely as the coefficient of polynomial in base three. For example, if the chessboard is in the picture below,

The unique index can be calculated as $1(empty) * 3^0 + 2(white) * 3^1 + 2(white) * 3^2 + 0(black) * 3^3$=25 The pattern can be stored in a one-dimensional array of length 81, the value is **the score of black side**. If the local situation has high score, then black side take advantage over white side, otherwise, white side prevails.

van 

Using such data structure, we can represent many local situations of the whole game, which is used to evaluate the game, and help AI to make decision. The evaluation part will be discussed below.

3.Model Design

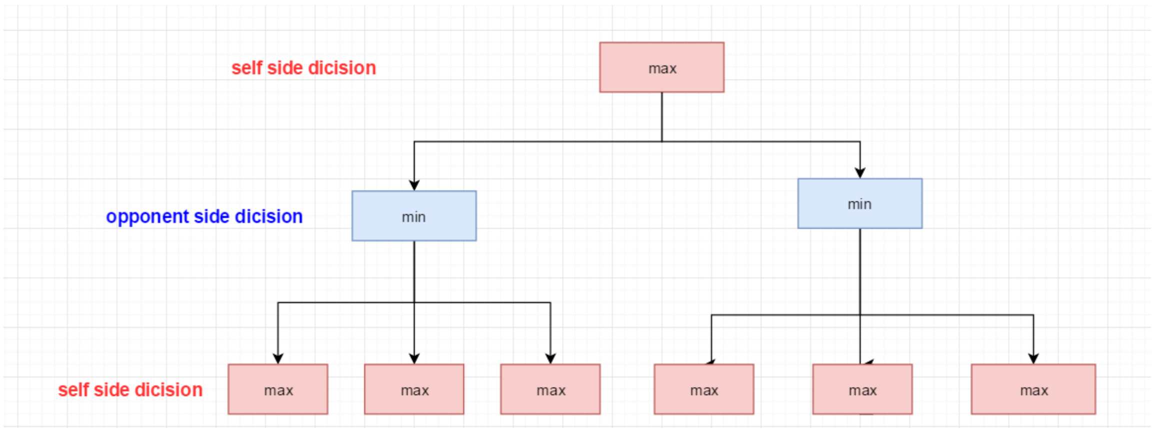
3.1.Problem Formulation

Given a chess board (two-dimensional-array) and current player to move(an integer), please find out all the feasible chess points of the current chess player, and give the best chess point, so that your chess player has the largest winning rate.

3.2.Minmax Algorithm and Game Tree

Two-player chess is a typical model of complete information, zero-sum game, so the minmax algorithm is very suitable. In Othello, the two players alternately move, and each person usually has more than one chess position to choose from. The minmax algorithm assumes that the decision-making process of the computer AI is as follows: when it is our turn to play chess, we will measure the "value" of all the positions that can be played and choose the position with the greatest profit; and if the enemy is playing chess, we will choose the area where the square's profit is the smallest; in this way, alternate decision-making forms a game tree. If the search space is small, we can use the final match result as the evaluation function to get an accurate minmax game tree, but because the search complexity increases exponentially with the increase in the number of search layers, we can usually only search 3-4 layers , And use some evaluation functions that estimate the phases of the situation as the evaluation function. The schematic diagram of the minmax game search tree is as follows:

▪



3.3.Alpha-beta Pruning

Alpha beta pruning is an optimization based on the minimax algorithm.The idea of the algorithm, is that in a min-max game tree search, for choice that "too bad" , we can end search early because we already know the max (min) node will not be chosen.

3.4.Evaluation function

Under the limitation of search depth, optimizing search algorithm may only optimize performance, and the accuracy of evaluation function plays a vital role in the strength of AI's chess power. Therefore, it is very necessary to design a set of valuations that can more accurately estimate the situation on the field.

Goal of evaluation function

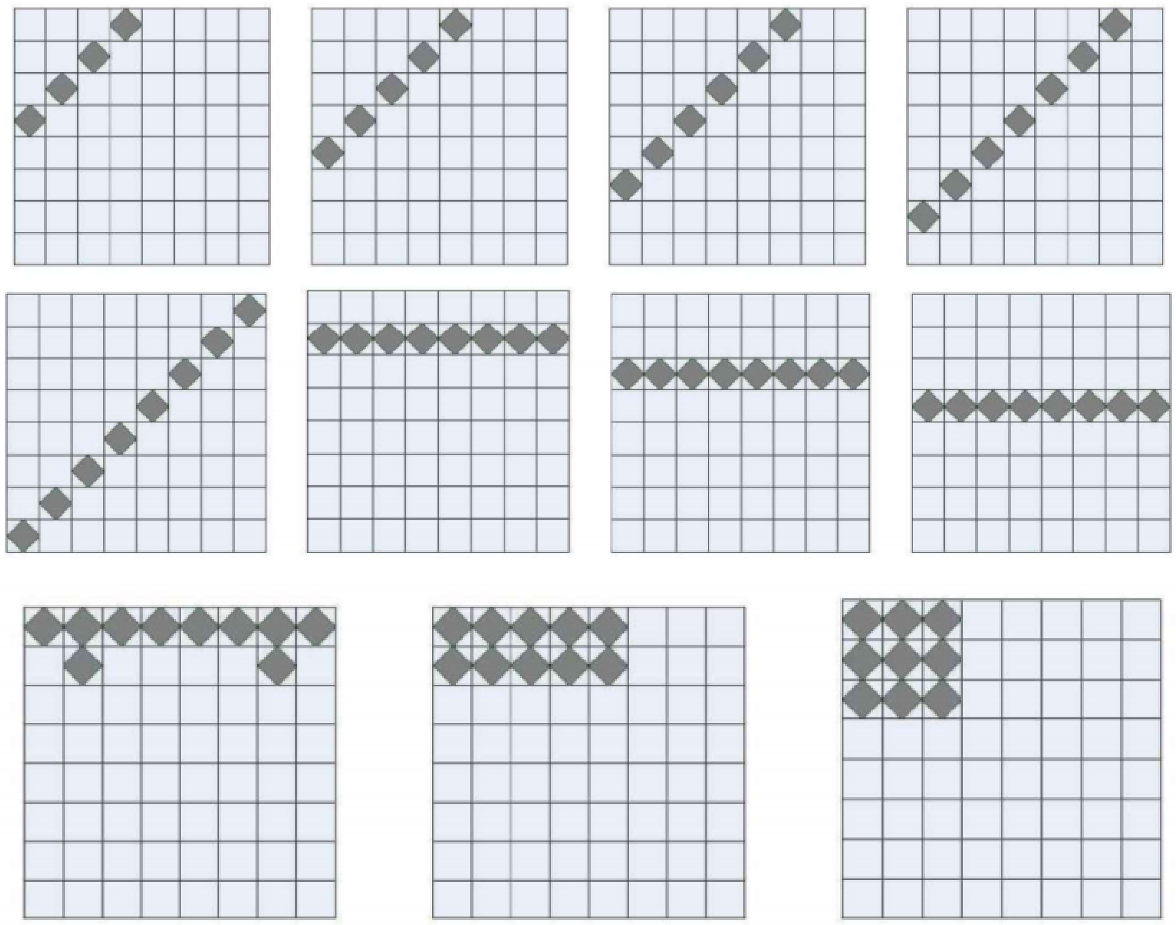
Given a certain situation of board, we can output the "score" of one side.

Assume current side is black: The score illustrate the winning rate of black side. To make the AI more like a expert, we can use statistical method: find sufficient number of othello game record, and calculate the frequency of occurrence of certain "situations".However, the possible "situation" of a othello game can be approximately at most 3^{64} , which is too large to count all of them. So instead, we can choose some local features to estimate score of current game.

Motivation of patterns

I use a more precise way to evaluate the board, it is called "pattern evaluate".Pattern enumerates permutations and combinations of several specific locations.” *Othello* ”[1] provides a lot of useful experience. Angle of attack, occupying stable sides, occupying diagonals, "building walls"...
The "pattern" is to enumerate all permutations and combinations of chess pieces in a local area for these positions, and assign a score for each combination. For example, if a side except corner is completely occupied by the white side, then the situation is favorable for the white side. But if the diagonal is inserted with a black piece, then the situation is favorable for the black player. Because this is an "unbalanced edge" and it is easy to be attacked. Pattern evaluation effectively takes into account the impact of the correlation of local chess pieces on the situation, and provides a more effective evaluation function than static weight evaluation.In 1997, Buro gives 11 classic $patterns$[3], as picture below shows.But because of the symmetry of patterns, totally there 46 pattern instance.SI the score of a board can be estimate as:

$Score = X_1 + X_2 + ... + X_{46}$ where X_i is the score of pattern i.For each pattern, we can use bitmap array to store the evaluation score of each situation in the pattern. For the smallest pattern, there are 4 position, so the size of bitmap is $3^4 = 81$. For the largest pattern, there are 10 positions, so the size of bit map is $3^{10} = 59049$.



Evaluation Patterns

Calculation of Pattern Score

Beacause the evaluation bitmap is too large to give scores manully, we can use statistical method to give scores for each pattern situation.According to *Michael Buro*[3], the value of certain situation of a pattern can be calculated as:

$$V(s) = \frac{Y(s) + 0.5}{N(s) + 1.0} - 0.5$$

where

N(s) = number of cases containing situation s

Y (s) = black win cases+0.5 draw cases containing s

The additive constants 0.5 and 1.0 assure a neutral evaluation (0) of pattern instances that do not occur.

The game log file can be found at http://www.skatgame.net/mburo/logbook.gam.gz

3.5.Detail of algorithm

Alpha-beta pruning

```
algorithm alphabetaSearch(board, color,remain depth):
max = negative infinity
if remain depth less or equal than 0
return evaluation(board,color)
if currentSide can not move:
    if oppenent side can not move
        return evluation(board,color)
    else
        return -alphabetaSearch(chessboard,-color,remain depth)
foreach available move do
    makemove on current board
    score= -alphabetasearch(chessboard,color,remain depth -1)
    unmake the move
    max= max(score,max)
return max
```

Evaluation Function

```
algorithm evaluation(board,color):
    sum=0
    if(color is black):
        foreach pattern:
            sum=sum+score of pattern(board)
    if(color is white):
        foreach pattern:
            sum=sum+score of pattern(board with all chess flipped)
return sum
```

3.6.Solution Model Summery

The algorithm flow of my program is as follows:

- Firstly, get the information of current board, and calculate all the positions that can be played.
- Secondly, for all the current positions that can be played, search for 3-4 layers, according to the evaluation function of the situation after 3-4 layers, get the score of lead node reached then that should be played, and return the result.

4.Empirical Verification

4.1.Dataset:

I uses a dataset containg 150,000 rounds of chess player game records for the learning of evaluating function.The records can be found at:https://www.skatgame.net/mburo/logbook.gam.gz.

For correctness, I uses a python program which controls the whole game, so my AI can play games offline with other AI program.

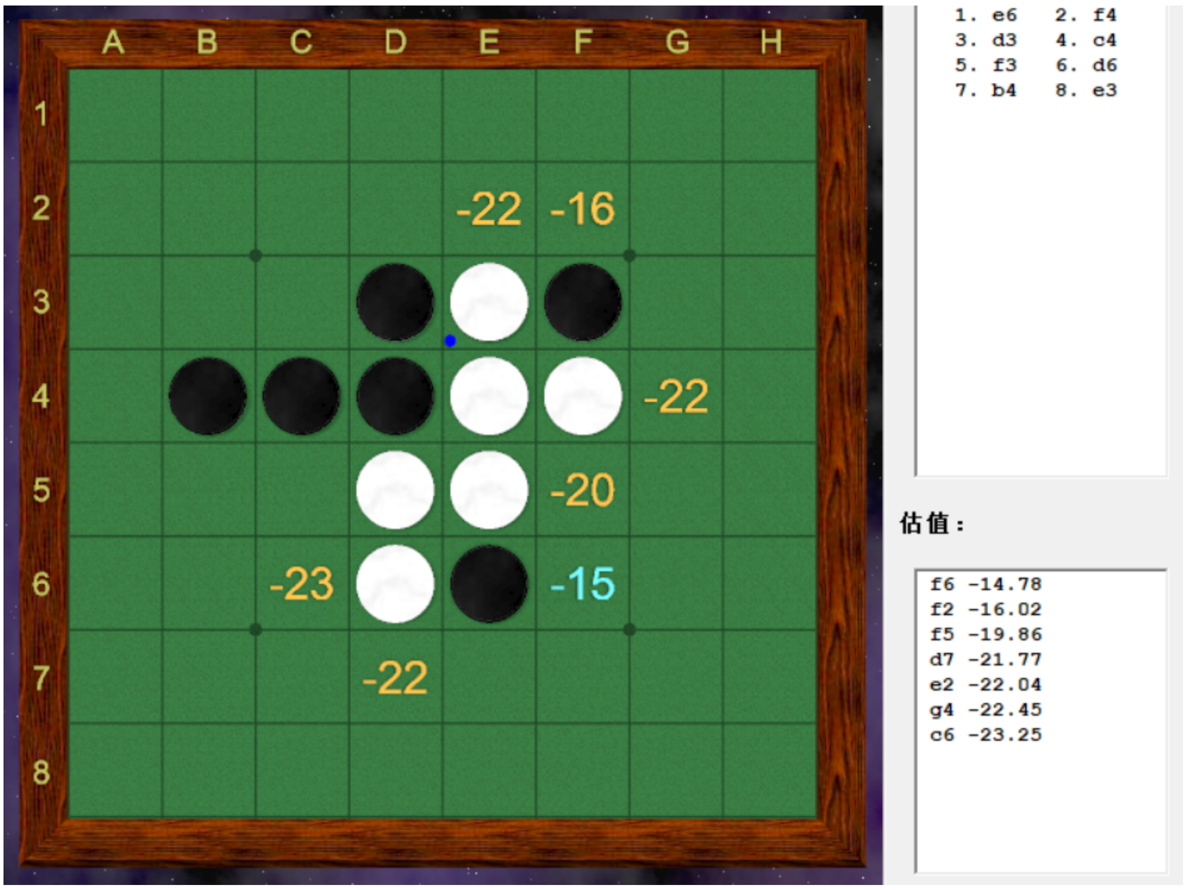
4.2.Correctness Test

The minimum requirement is that the program should not have fatal bug during execution. To ensure that, I uses a nAI platform, letting my code to play with a random AI player.In this way, the correctness of the program is effectively tested in most situations.

4.3.Evaluation Function Validity Test

It is very important to measure the accuracy of the evaluation function. In order to do this, we can randomly generate a situation and compare it with the evaluation function of Wzebra's "spectrum mode". If in most cases, the ranking result of our evaluation function Similar to $Wzebra$[2], it can be considered that the result of our evaluation function is valid.

The picture below is Wzebra's evaluation for each postion,I campare it with mine:



```
1. e6    2. f4
3. d3    4. c4
5. f3    6. d6
7. b4    8. e3
```

估值：

```
f6 -14.78
f2 -16.02
f5 -19.86
d7 -21.77
e2 -22.04
g4 -22.45
c6 -23.25
```

	Wzebra	myAI
f6	-14.78	-43
f2	-16.02	-48
f5	-19.86	-49
d7	-21.77	-60
e2	-22.04	-56

	Wzebra	**myAI**
g4	-22.45	-65
c6	-23.25	-62

As we can see, the evaluation function has similar relative value for each postion, and both choose f2 for best postion.

4.4.Performance Measure

I randomly conducted 100 rounds of tests, and counted the time required for the search depth when the number of feasible steps is the largest, and the average time for searching for layers 2, 3 and 4.

4.5.Parameters Used

In practice, I used 2 parameters in game:

α : the remain step to start end game search. When there are only 10-20 empty position left in the chessboard, it becomes possible to search deep into the end of game.Then we can use the exact game result as evaluation function (disc of black disc -white).When there are only α empty positions, I will begin the end game search.

β : the random factor.When in the rank phase, if the AI does exact the same move in the game when there are some other good options, it may be heuristically or miss the best option to try.So, for example, if the best step has score 48, and random factor β is 0.95， then the AI will choose a move in the range [0.95 * 48, 48].

4.6.Experimental Results

I did a performance test for search depth.

	average	**max**
depth 2	0.03s	0.11s
depth 3	0.42s	1.08s
depth 4	5.96s	21.12s

It can be seen that when the search depth is less than or equal to 3, the search time is very abundant, but when the search depth reaches 4 levels, there is a very large possibility of timeout.

For parameters, I find 15 is a good configuration for remain step end game search.If the number is too large, then it will result in timeout.For random factor, I just use it in rank game, and it helps me to reach 4th ranking.

Points race ranking: 4th

Round robin ranking:4th

4.7.Conclusion

Advantage of my algorithm:

1.Knowledge from expert game

In the lectime time of CS303A, Prof. Tang mentioned that, what makes AI strong is the correct representation of expert knowledge, which inspires me. So my evaluation function is the statistical results of 150,000 games, which gives the AI program 'knowledge'.

2. The design of end game search

It makes the evaluation more accurate. Unlike Go, the final steps of Othello can often determine the outcome, so end game search plays an important role.

Disadvantage of my algorithm:

Lack of search depth. When the search depth is 4, the time cost will exceeds 5 seconds when there are too many alternative moves. It can be further improve by using smarter pruning algorithm.

Difficulties

The difficulties comes from two parts: Debugging of evaluation algorithms and processing of game record data. Ther are both very time consuming but inevitable. Making a good summary of similar work can improve this kind of process.

5.References

[1] B. Rose, *Othello and A Minute to Learn...A lifetime to Master*. Anjar Co., 2004.
[2] G. Andersson, “Wzebra,” *Download WZebra*, 2006. [Online]. Available: http://www.radagast.se/othello/download.html. [Accessed: 01-Nov-2020].
[3] M. Buro, *An Evaluation Function for Othello Based on Statistics* , NECI Technical Report #31 (1997)