Report for Project Reversi

陈宇
11811203
Department of Computer Science and Engineering
Southern University of Science and Technology

1. Preliminaries

1.1 Problem Description：

Reversi is an interesting board game with the following rules: when your pieces surround your opponent's pieces in a straight line, you can flip the colors of these pieces to make them your color. Traditional chess AI algorithms include deep learning, minimax tree, Monte Carlo algorithm and so on.

Goal:realize the artificial intelligence which can play Reversi intelligently.

Algorithm: alpha–beta pruning, heuristic algorithm, evaluation function

Software:

Python 3.8
IDE: JetBrains PyCharm

Code: implement the pseudo code of paper

1.2 Problem Applications ：

1. Learn AI algorithm deeply
2. Promote the development of black and white chess

2. Methodology

2.1 notation：

Evaluation(board，color）:evaluate the board score of the current color pieces

Negmax(board,color):negamax Algorithm

Find_avilible(color)：calculate the current possible drop points

Sumnum（board,color）:calculate the number of pieces with current color on the current chessboard

Best_move:tuple<int, int>, store position of one piece

Chessboard: list<int, int>, a record of piece positions for the chessboard

Make_move(board,color):the simulation generates the chessboard after the falling point of the current color chessboard

go():according to the judgment, make a decision

timeout()：whether the detection time is out of time

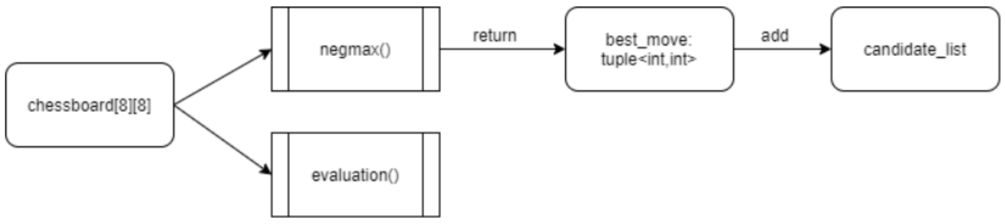
2.2 data structure：

candidate_list: a list of the most valuable positions which have found.

color： The color of the pieces currently playing chess

chessboard： chessboard

Vmap： Assignment table for the current chessboard

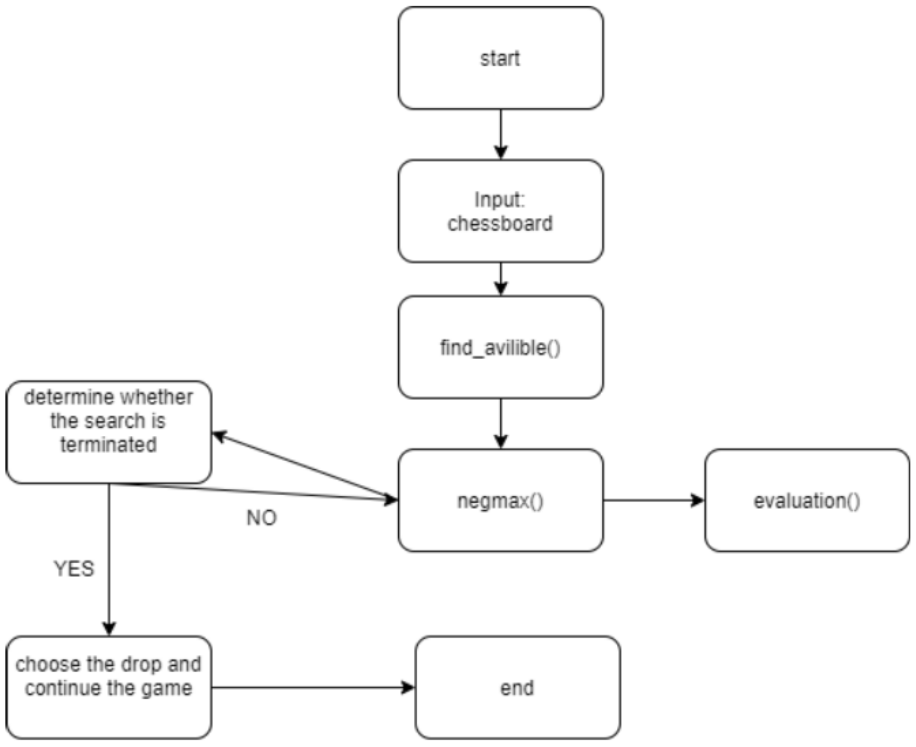


2.3 Model design：

2.3.1 minimax algorithm：

To calculate the drop by AI is actually to select the best drop point by analyzing the chessboard situation and evaluating the function. The most common way is to set different values for different positions of the chessboard, and then find the best place to fall by violent search. However, it is not possible to predict the long-term benefits of the other party. In fact, when calculating the income, we should take into account the possible landing point of the other party. Since both AI and AI will consider the location of the best return in the game, we need to choose the best one from the other's choice, that is, the minimax algorithm. [1] Each time you select a drop point, the score of each possible drop point is calculated by the evaluation function and saved in the tree. Because your own benefit should be positive and the opposite party's profit should be negative, you should choose the maximum value of your own side and the minimum value of the other side each time you choose the path. The general process can be divided into the following :

1. Through find_ Possible falling points of color (avible)
2. Through the evaluation function evaluation (board, color) to calculate the score of the current color of various falling points
3. Simulate the chessboard after the best placement
4. Continue to calculate the next layer through negmax (board, color), and select the best placement point of the new chessboard
5. Return to the best location after the search


start
Input: chessboard
find_avilible()
determine whether the search is terminated
negmax()
evaluation()
NO
YES
choose the drop and continue the game
end

2.3.2 evaluation function:

In addition, the implementation of evaluation function is also more important. We can't have absolutely accurate static estimation function. It is called heuristic search to construct evaluation function with specific knowledge. [2] General black and white chess in the more important corner, stabilizer, action and so on. By assigning values to the chessboard, AI can pay more attention to corners. Through the calculation of stabilizers, AI can obtain more pieces that will not be flipped in the middle and late stage, and expand the advantage. The driving force is the number of possible landing points each time, which can expand the advantages in the early and middle period. The possible action force is to evaluate the situation around the falling point. If there are pieces of opponent's pieces, it is disadvantageous; if it is blank, it is beneficial. By judging the current situation of the chess game, combining the above considerations according to different proportions, a static evaluation function can be obtained, thus completing the evaluation of the chessboard.

2.4 Detail of algorithms:

negmax(board, depth, alpha, beta, color): negamax Algorithm

```
negmax(board, depth, alpha, beta, color)
   self_move=find_avilible(color)
   opp_move=find_avilible(-color)
   if depth <= 0:
       return None, evo(board, color)
   if (len(self_move) == 0):
       if (len(opp_move) == 0):
           return evo(board, color)
       else:
           score =negmax(board, depth - 1, -beta, -alpha, -color)
           return -score
   for move in self_move:
```

```
            new_board = deepcopy(board)
            makemove(move,color)
            score = self.minmax(newboard, depth - 1, -beta, -alpha, -color)
            score = -score
            if score > alpha:
                alpha = score
                max_move = move
                if (beta <= alpha):
                    return score
        return alpha
```

movibility(board,color): calculating the movibility

```
    movibility(board,color)
        self_move=find_avilible(color)
        opp_move=find_avilible(-color)
       return len(self_move)-len(opp_move)
```

possible_movibility(board,color):calculating potential movibility

```
    possible_movibility(board,color)
        count = 0
            if chesspotision==color
               foreach adjacentpotision ==None
                     count++
             if chesspotision==-color
               foreach adjacentpotision ==None
                     count--
             return count
```

satblility(board,color): calculating satblility

```
    satblility(board,color)
        corner=(0,0),(0,7),(7,0),(7,7)
        if board[corner]==color
            len = 8
            for i ← 0 to 7:
                for j ← 0 to len:
                    if board[i][j] == color:
                        count = count + 1
                    else:
                        len = j
                        break
```

evaluation(board, color):calculated the score by the evaluation function

```
    evaluation( board, color)
        if (sumnum(board,color) <= 20):
            score = 50 * weight(color) + 20 * movibility + 15 *
possible_movibility(board, color)
        if (20 < sumnum(board,color) <= 40):
            score = weight(color) + 35 * movibility + 20 *
possible_movibility(board, color) + 200 * satblility(board, color)
        if (40 < sumnum(board,color)):
            score = weight(color) + 20 * movibility + 10 *
possible_movibility(board, color) + 200 * satblility(board, color)
        return score
```

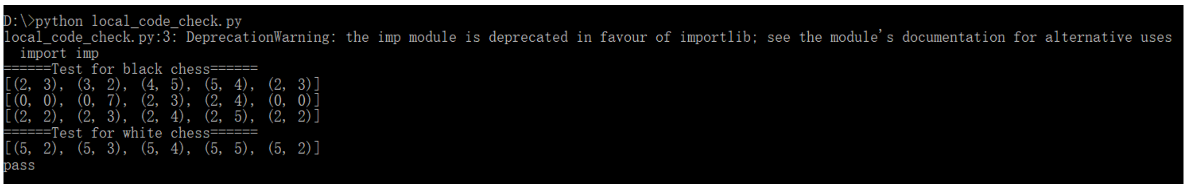
weight(board,color): calculate the score according to the board weight

```
    weight(board,color)
         for i ← 0 to 7:
             for j ← 0 to 7:
                if board[x][y] == color:
                   score += Vmap[x][y]
                if board[x][y] == -color:
                   score -= Vmap[x][y]
         return score
```

3. Empirical Verification

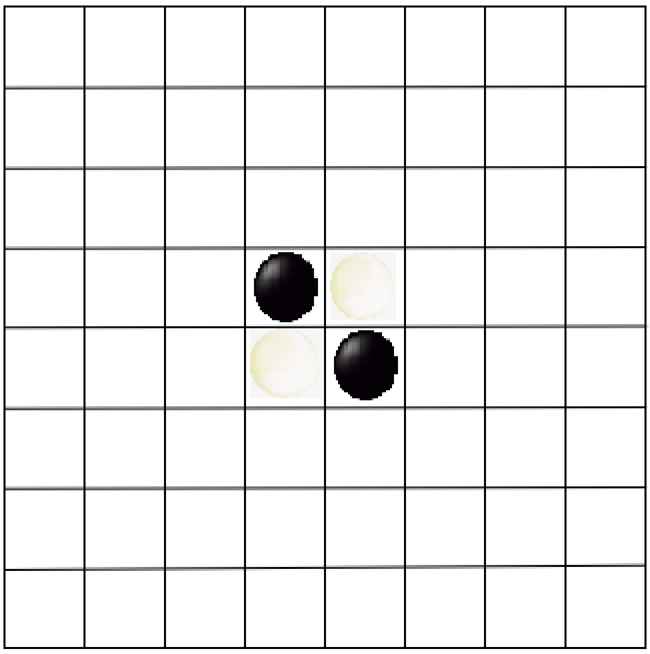
3.1 Dataset:

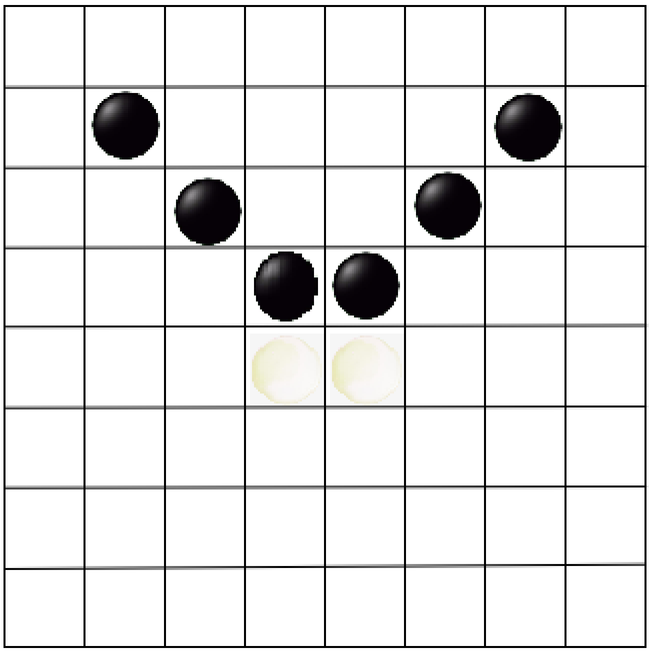
In the usability test, I used the four test cases given by the teacher and passed the sample test in the first stage. I will output a list of points that can be dropped to check if there is a problem with the drop points.

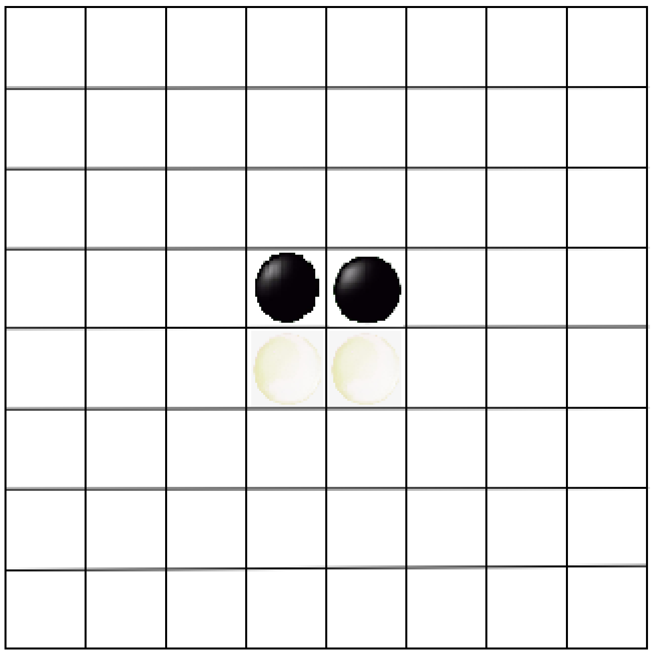


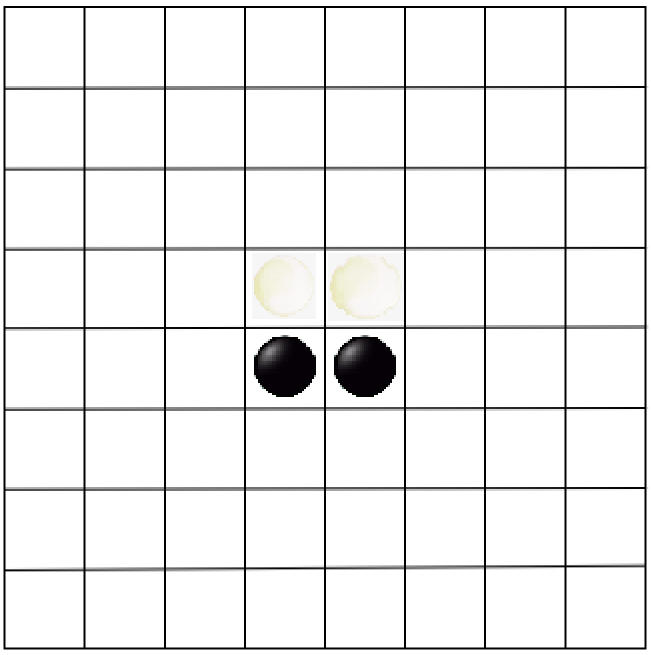
```
D:\>python local_code_check.py
local_code_check.py:3: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
======Test for black chess======
[(2, 3), (3, 2), (4, 5), (5, 4), (2, 3)]
[(0, 0), (0, 7), (2, 3), (2, 4), (0, 0)]
[(2, 2), (2, 3), (2, 4), (2, 5), (2, 2)]
======Test for white chess======
[(5, 2), (5, 3), (5, 4), (5, 5), (5, 2)]
pass
```

Here are four examples from the teacher:

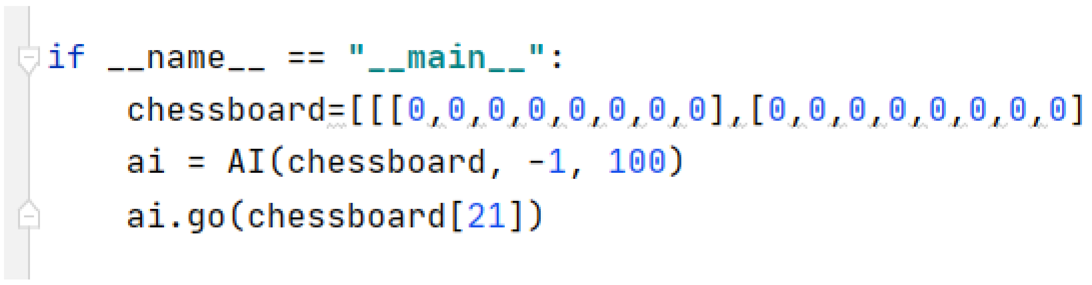








In the points race and round robin, I use 10.20.26.45:8080 / whitelog (blacklog) and download chessdata .Then I selectively used main method to test some steps in the dataset.If there is an unexpected drop, then I will check my function.



```
if __name__ == "__main__":
    chessboard=[[[0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0]
    ai = AI(chessboard, -1, 100)
    ai.go(chessboard[21])
```

After playto function is disabled, sometimes I use autoplay to see if I can increase the ranking, and sometimes I play on the botzone.

3.2.Performance measure:

There are three parts to test expressiveness.

First, you need to pass the usability test. If this step is wrong, then all the later algorithms can not work normally.

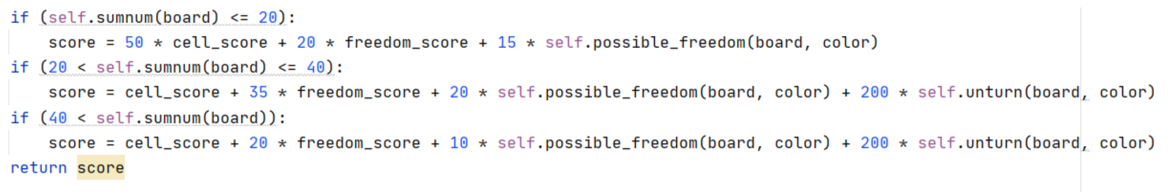
Secondly, through the use of playto against fixed opponents, observe whether their algorithm is wrong or whether it has better performance after changing the parameters. Because when the opponent is fixed, it can indirectly reflect the strength of their AI.

Finally, you can use autoplay for rank competition. If one win and one loss often occurs in the course of the game, then I need to adjust the parameters and let the new version of AI compete with the old version to see the effect of parameter adjustment. Adding stabilizers and potential actions to the evaluation function can make my AI better.

3.3 Hyperparameters:

In the usability test, the problem of parameters is not considered because it only needs to judge whether the falling point is correct or not.

In the points race, at the beginning, my evaluation function was very simple. I only considered the board weight and action power. In order to quickly occupy the angle, I gave a high proportion of the board weight. Through this evaluation function, I can play more than 40 in the early stage, but it has become weak in the middle and late stages of the game, so I added stabilizers and potential action. Because the action force is more important in the early and middle period of the chessboard, and the stabilizer is more important in the middle and late stage, so I changed the evaluation function according to the situation of the chessboard. As shown in the figure below:



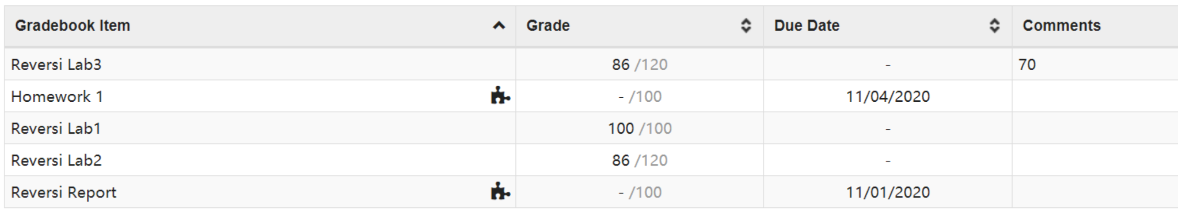
```
if (self.sumnum(board) <= 20):
    score = 50 * cell_score + 20 * freedom_score + 15 * self.possible_freedom(board, color)
if (20 < self.sumnum(board) <= 40):
    score = cell_score + 35 * freedom_score + 20 * self.possible_freedom(board, color) + 200 * self.unturn(board, color)
if (40 < self.sumnum(board)):
    score = cell_score + 20 * freedom_score + 10 * self.possible_freedom(board, color) + 200 * self.unturn(board, color)
return score
```

In addition, in the points race, it is easy to time out due to insufficient resource allocation due to too many competitors. I use the timeout() function to judge and terminate the search when it is about to time out. So towards the end of the game, I changed to search only three layers.

In the round robin, I didn't make too many changes, just changed the search depth to 4 levels

3.4 Experimental results:

I pass all the test case in usability test, get 86 in the points race and in round robin, my rank is 70st, score is 86.



Gradebook Item	Grade	Due Date	Comments
Reversi Lab3	86 /120	-	70
Homework 1	- /100	11/04/2020	
Reversi Lab1	100 /100	-	
Reversi Lab2	86 /120	-	
Reversi Report	- /100	11/01/2020	

3.5 Conclusion:

Through this project, I gained a lot and learned AI related algorithms. At the beginning, I wrote AI at the bottom of the list. Later, I found that the algorithm was wrong. Even though I added the potential chess board to correct the situation, I was able to improve the potential strength of the game quickly. I use the most basic minimax algorithm, which is easy to implement, but the problem is that the number of search layers is not enough, and it depends on the quality of the evaluation function.

Possible improvement directions: on the premise of not changing the algorithm, we can increase AI IQ by adding hash table and opening library, or adjust parameters to obtain better results. We can also learn the Monte Carlo algorithm to increase the search depth and get better results.

4. References

[1] https://blog.csdn.net/wskywskywsky/article/details/93191500?utm_medium=distribute.pc_relevant.none-task-blog-title-2&spm=1001.2101.3001.4242

[2] https://baike.baidu.com/item/%E6%9E%81%E5%A4%A7%E6%9E%81%E5%B0%8F%E5%80%BC%E7%AE%97%E6%B3%95/19473878?fr=aladdin