Gomoku

Ver 3.0(Final)

09016401

顾婷瑄

Botzone id: Gutingxuan

1. Old versions review

i. Gomoku1.0 is based on a static evaluation function which simply evaluates the value of each available position. It is easy to understand and realize and it seems to be effective at first. But with the time going, its ability is not strong enough to beat my classmates’ programs, so I decided to evolve it with a more complicated algorithm next time.

ii. Gomoku2.0 is simply based on a MaxMin algorithm and although it can foresee future moves, it spends a lot of time on doing calculations so the deepest searching level it can reach is a mere three. So I was thinking about making some improvements to cut down the searching time to enhance the depth of searching.

iii. Gomoku2.1 is based on a MaxMin algorithm together with the alpha-beta pruning. It can make tentative moves and foresee future moves of both sides in a shorter time and the depth of the MaxMin algorithm was enhanced due to the help of alpha-beta pruning. This one is basically my final version of Gomoku.

Last but not least, I made some changes to my evaluation functions every time when working on my new version of Gomoku. I guess this helps too.

2. Literatures I read

[1] 王志水. 基于搜索算法的人工智能在五子棋博弈中的应用研究[D].山东：中国石油大学, 2006

[2] Blog: 五子棋基本棋型@我是老邱

五子棋估值算法@maxuewei2

3. Current idea of design

i. Essence

A function based on the MaxMin algorithm and the alpha-beta pruning.

ii. Origin of the idea

The paper I’ve read.

iii. Expectation

The function can make tentative moves, evaluate the value of each state on both sides and then make the best choice. Furthermore, the time it spends to do the calculations will be decreased.

iv. Implementation

A MaxMin algorithm, alpha-beta pruning, new evaluation method.

v. Advantages of the design

With the alpha-beta pruning, the time spent on calculating can be decreased. In the last version, I sorted the values of the available positions before searching and only visited the first 12 positions and the depth it could go was 6. But in this version, I tried to visit the first 45 possible positions and go with the depth of 4 and the result turns out to be better than the last version on Botzone. However if there is not a time limit, the depth of 6 will definitely beat the 4 by visiting same amount of positions. And I fixed a possible bug of my evaluation function which evaluates the value of the board, I guess this helps a lot.

vi. Weakness

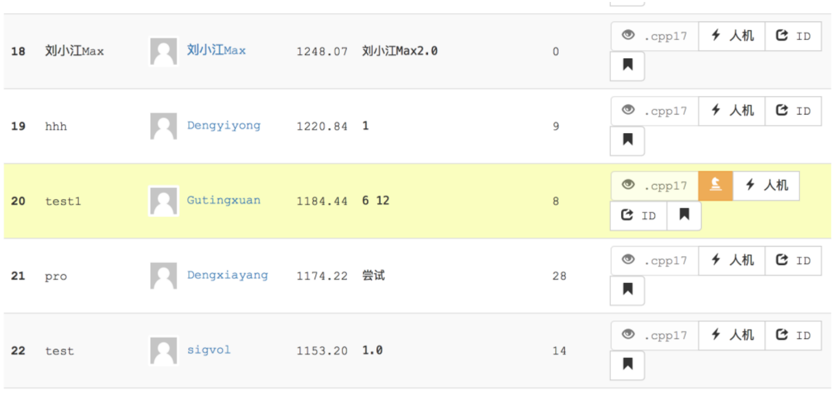
The alpha-beta pruning was not strong enough so the number of positions the program can visit in a pleasant time is still restricted. And the evaluation of the board value was not that accurate compared with the actual value of the board.

vii. Improvement directions

I will try to optimize the evaluation method and try to improve my alpha-beta pruning. But I guess this will only happen if I got some spare time since I’ m pretty satisfied with my current version of Gomoku.

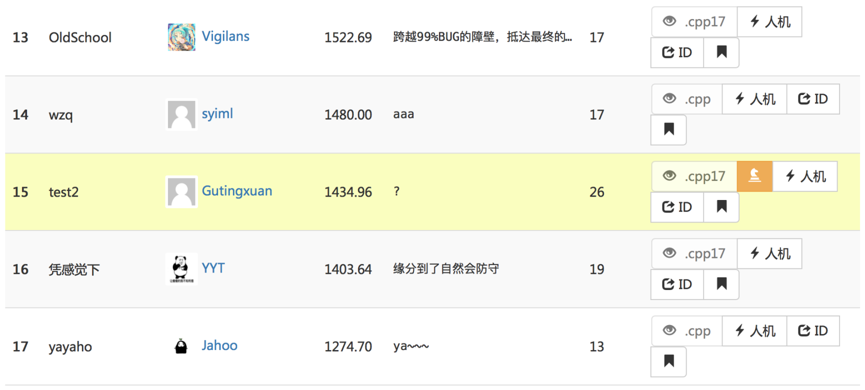
4. Evaluation of the performance

Former ranking on Botzone: 20/37 (2018/5/3 20:00 Ver2.1)



18	刘小江Max	刘小江Max	1248.07	刘小江Max2.0	0	.cpp17 人机 ID
19	hhh	Dengyiyong	1220.84	1	9	.cpp17 人机 ID
20	test1	Gutingxuan	1184.44	6 12	8	.cpp17 人机 ID
21	pro	Dengxiayang	1174.22	尝试	28	.cpp17 人机 ID
22	test	sigvol	1153.20	1.0	14	.cpp17 人机 ID

Current ranking on Botzone: 15/54　　　　　　　(2018/6/1 15:00 Ver3.0)



13	OldSchool	Vigilans	1522.69	跨越99%BUG的障壁，抵达最终的...	17	.cpp17 人机 ID
14	wzq	syiml	1480.00	aaa	17	.cpp 人机 ID
15	test2	Gutingxuan	1434.96	?	26	.cpp17 人机 ID
16	凭感觉下	YYT	1403.64	缘分到了自然会防守	19	.cpp17 人机 ID
17	yayaho	Jahoo	1274.70	ya~~~	13	.cpp 人机 ID

5. Code structure and detailed implementations

Since the MaxMin algorithm and alpha-beta pruning were explained in former reports and they were not changed in this version, this time I will only introduce my enhanced evaluation function.

To evaluate the value of the board, I only collect the alive-threes, dead-fours, alive-fours and five-in-a-rows because these can best represent the value of the current board and these are the moves that have to be dealt with, otherwise you will definitely lose the game. This kind of simplification can save a lot of time spent on the searching of board types.

But there’ s a problem with this evaluation function which is that the same set value may be calculated for a few times since there might be more than one position can detect this set. But since there are huge gaps between different levels of sets, this error won’ t make a big mistake.

```
int Evaluate3(int x, int y, int choice)
{
    int temp=0;
    int count=0;
    for (int i = 1; i <= 8; i++)
    {
        //连五
        if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == choice &
& getLine(x, y, i, 3) == choice
```

```
            && getLine(x, y, i, 4) == choice && getLine(x, y, i, 5) == choice)
        {
            temp += 5500000;
        }

        //活四 01111*
        if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == choice && getLine(x, y, i, 3) == choice
            && getLine(x, y, i, 4) == choice && getLine(x, y, i, 5) == -1)
            temp += 500000;

        //冲四 A 21111*
        if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == choice && getLine(x, y, i, 3) == choice
            && getLine(x, y, i, 4) == choice && (getLine(x, y, i, 5) == 1-choice||getLine(x,y,i,5)==-2))
            temp += 450000;

        //冲四 B   1*111
        if (getLine(x, y, i, -1) == choice && getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == choice
            && getLine(x, y, i, 3) == choice)
            temp += 400000;

        //冲四 C   11*11
        if (getLine(x, y, i, -2) == choice && getLine(x, y, i, -1) == choice && getLine(x, y, i, 1) == choice
            && getLine(x, y, i, 2) == choice)
            temp += 250000;

        //活三 A-1 0111*0
                if (getLine(x, y, i, 1) == -1 && getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == choice
            && getLine(x, y, i, -3) == choice && getLine(x, y, i, -4) == -1)

           {
               temp += 35000;
               count++;
           }

        //活三 B-1  01*110
        if (getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == -1 && getLine(x, y, i, 1) == choice
                && getLine(x, y, i, 2) == choice && getLine(x, y, i, 3) == -1)
                {
                    temp += 35000;
                    count++;
                }

            //活三 B-2  *10110
            if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == -1 && getLine(x, y, i, 3) == choice
                && getLine(x, y, i, 4) == choice && getLine(x, y, i, 5) == -1)
                {
                    temp += 35000;
                    count++;
                }

            //活三 B-3  01011*
            if (getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == choice && getLine(x, y, i, -3) == -1
                && getLine(x, y, i, -4) == choice && getLine(x, y, i, -5) == -1)

                {
                    temp += 35000;
                    count++;
                }
        }

        if(count>=2) temp+=400000;
        return temp;
}
```

Gomoku

Ver 1.0

09016401
顾婷瑄
Botzone id: Gutingxuan

1. Old versions review

This is my very first version of Gomoku, and I am excited to introduce it to you.

2. Literatures I read

[1] 王志水. 基于搜索算法的人工智能在五子棋博弈中的应用研究[D].山东：中国石油大学, 2006

[2] Blog: 五子棋基本棋型@我是老邱

五子棋估值算法@maxuewei2

3. Current idea of design

i. Essence

A function based on the evaluation of each available position.

ii. Origin of the idea

Myself

iii. Expectation

The function can comprehensively take every potential board type into account and then make the best move.

iv. Implementation

① A function that collect all the positions that has neighbors in the distance of 2 so that it can save searching time.

② Two evaluation functions which will separately evaluate the attacking potential and the defense potential of each available position.

③ A function that can evaluate the chess from eight directions.

④ A function which will choose the best position to make the next move.

v. Advantages of the design

It is easy to implement and it is convenient to edit the current known board type so that the computer can give out a more accurate score.

vi. Weakness

The computer can only evaluate the current board and give the score. It can not evaluate the potential moves in future turns and give out a better solution.

vii. Improvement directions

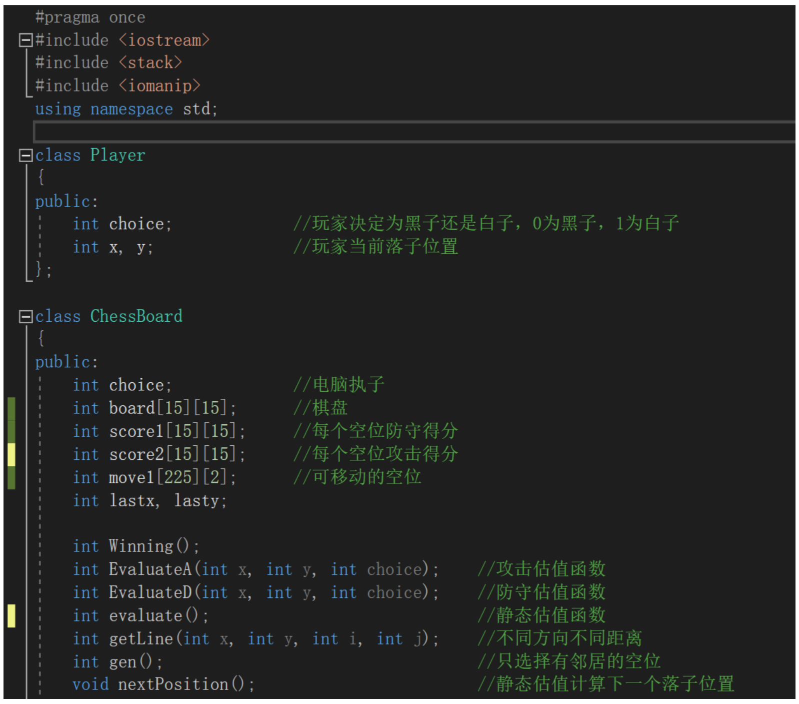
I will implement a search algorithm which can foresee future moves and thus give out a better solution.

4. Evaluation of the performance

It is still pretty weak now since even I can beat it. However, after being posted on the Botzone, sometimes it can actually beat other Gomoku AI players. But since this is just the very first version, I am not ready to put it in the ranking list because the result might hurt feelings so there will not be any rank or score now. I am still glad it can successfully detective the winning move of the player and then prevent in advance and the accident victories really give me a thrill.

5. Code structure and detailed implementations

i. The code structure



```cpp
#pragma once
#include <iostream>
#include <stack>
#include <iomanip>
using namespace std;

class Player
{
public:
    int choice;             //玩家决定为黑子还是白子，0为黑子，1为白子
    int x, y;               //玩家当前落子位置
};

class ChessBoard
{
public:
    int choice;             //电脑执子
    int board[15][15];      //棋盘
    int score1[15][15];     //每个空位防守得分
    int score2[15][15];     //每个空位攻击得分
    int move1[225][2];      //可移动的空位
    int lastx, lasty;

    int Winning();
    int EvaluateA(int x, int y, int choice);    //攻击估值函数
    int EvaluateD(int x, int y, int choice);    //防守估值函数
    int evaluate();                             //静态估值函数
    int getLine(int x, int y, int i, int j);    //不同方向不同距离
    int gen();                                  //只选择有邻居的空位
    void nextPosition();                        //静态估值计算下一个落子位置
```

ii. The score table

Originally, the table goes:

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0
0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0
0, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 0
0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 3, 2, 1, 0
0, 1, 2, 3, 4, 5, 5, 5, 5, 5, 4, 3, 2, 1, 0
0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0
0, 1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1, 0
0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0
0, 1, 2, 3, 4, 5, 5, 5, 5, 5, 4, 3, 2, 1, 0
0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 3, 2, 1, 0
0, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 0
0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Different position is given different initial score so that if the player goes black and makes the first move, then the computer will choose the position that near the center to make the move instead of making the move randomly from the first row.

```
/*
*: 当前空位置
0: 其他空位置
1: 当前player
2: 对手
*/
```

Different board type is given different scores:

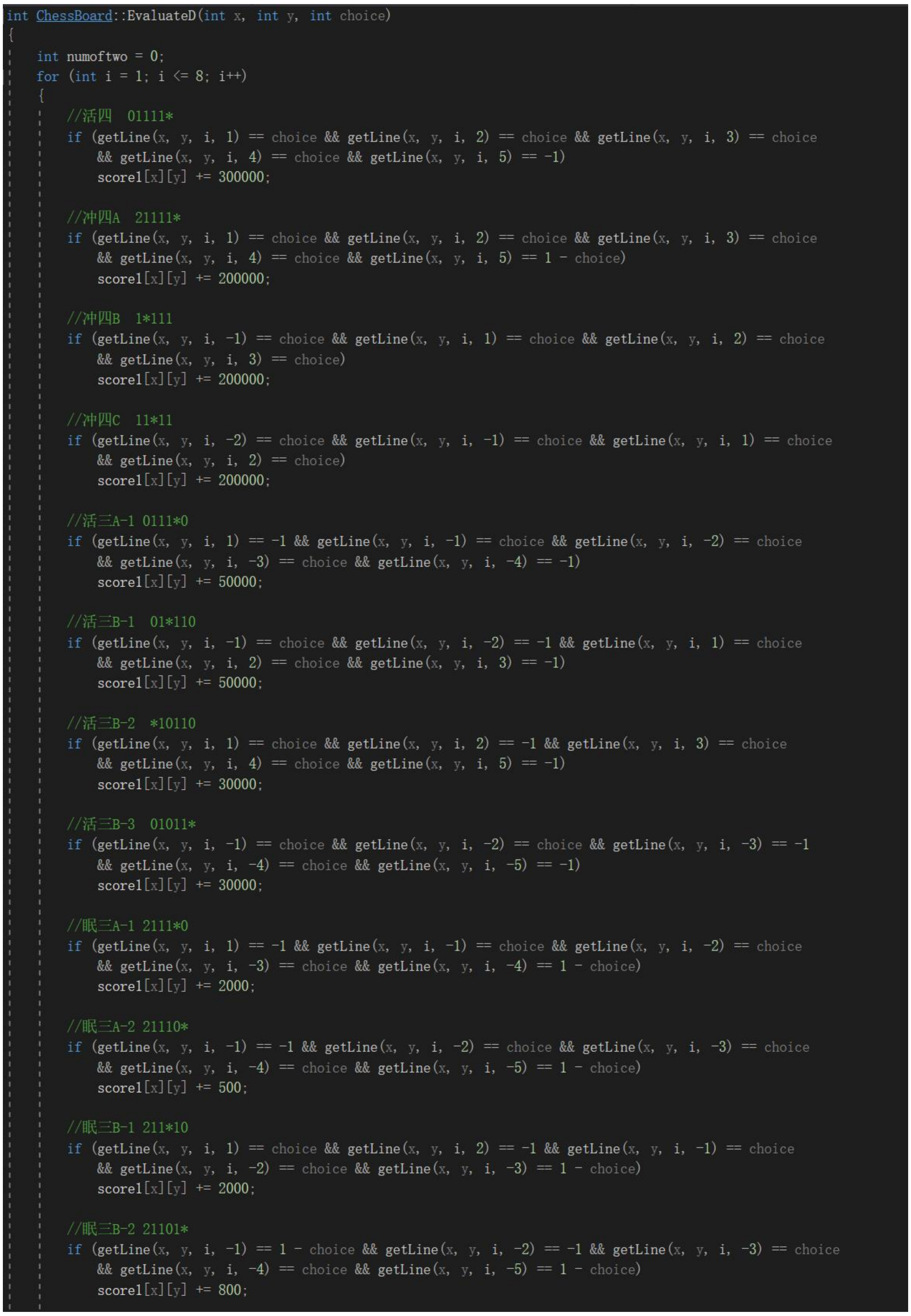
		Attack	Defense
活四	01111*	300000	300000
冲四A	21111*	250000	200000
冲四B	1*111	250000	200000
冲四C	11*11	250000	200000
活三A-1	0111*0	60000	50000
活三B-1	01*110	60000	50000
活三B-2	*10110	40000	30000
活三B-3	01011*	40000	30000
眠三A-1	2111*0	3000	2000
眠三A-2	21110*	1500	500
眠三B-1	211*10	3000	2000
眠三B-2	21101*	1800	800
眠三C-1	21*110	3000	2000
眠三C-2	21011*	1800	800

眠三D-1	211*01	1600	600
眠三D-2	2110*1	1600	600
眠三E	21*101	1550	550
活二A-1	0*0110	1650	650
活二A-2	0*110	4000	3000
活二B-1	0*1010	1650	650
活二B-2	01*10	4000	3000
活二C-1	01*010	1650	650
活二C-2	*10010	1650	650
眠二A-1	211*00	1150	150
眠二A-2	2110*0	1150	150
眠二A-3	21100*	1150	150
眠二B-1	21*100	1250	250
眠二B-2	2101*0	1250	250
眠二B-3	21010*	1250	250
眠二C-1	21*010	1200	200
眠二C-2	210*10	1200	200
眠二C-3	21001*	1200	200
眠二D-1	21*001	1100	100
眠二D-2	210*01	1100	100
Numoftwo>=2		3000	2000

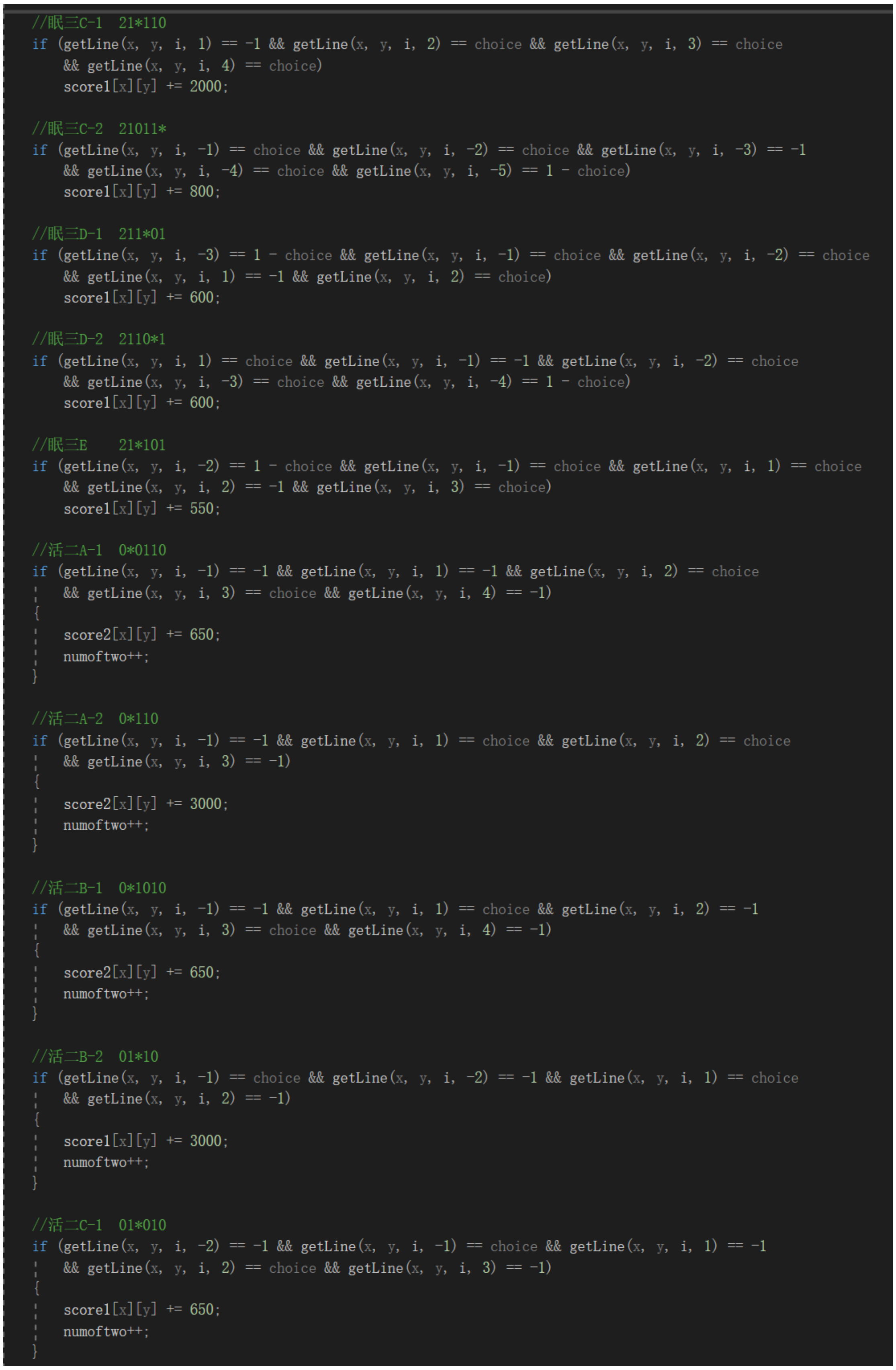
Different board type scores can help computer choose the best position to make the move. Attack score is the one that computer can gain if it places the stone on a position and the defense score is the one that player can gain if he places the stone on a position. Therefore, the computer will have to choose whether to make a move to attack or to defense in case the player will make a move to gain a higher score. And I choose to make the computer a more competitive one so the attack score will be a little bit higher than defense score in the same board type case. And after playing a few games, I found that the number of alive-two board types sometimes means a lot so I give extra points to the position that meets at least two alive-two board types.

iii. How to evaluate the score of each position

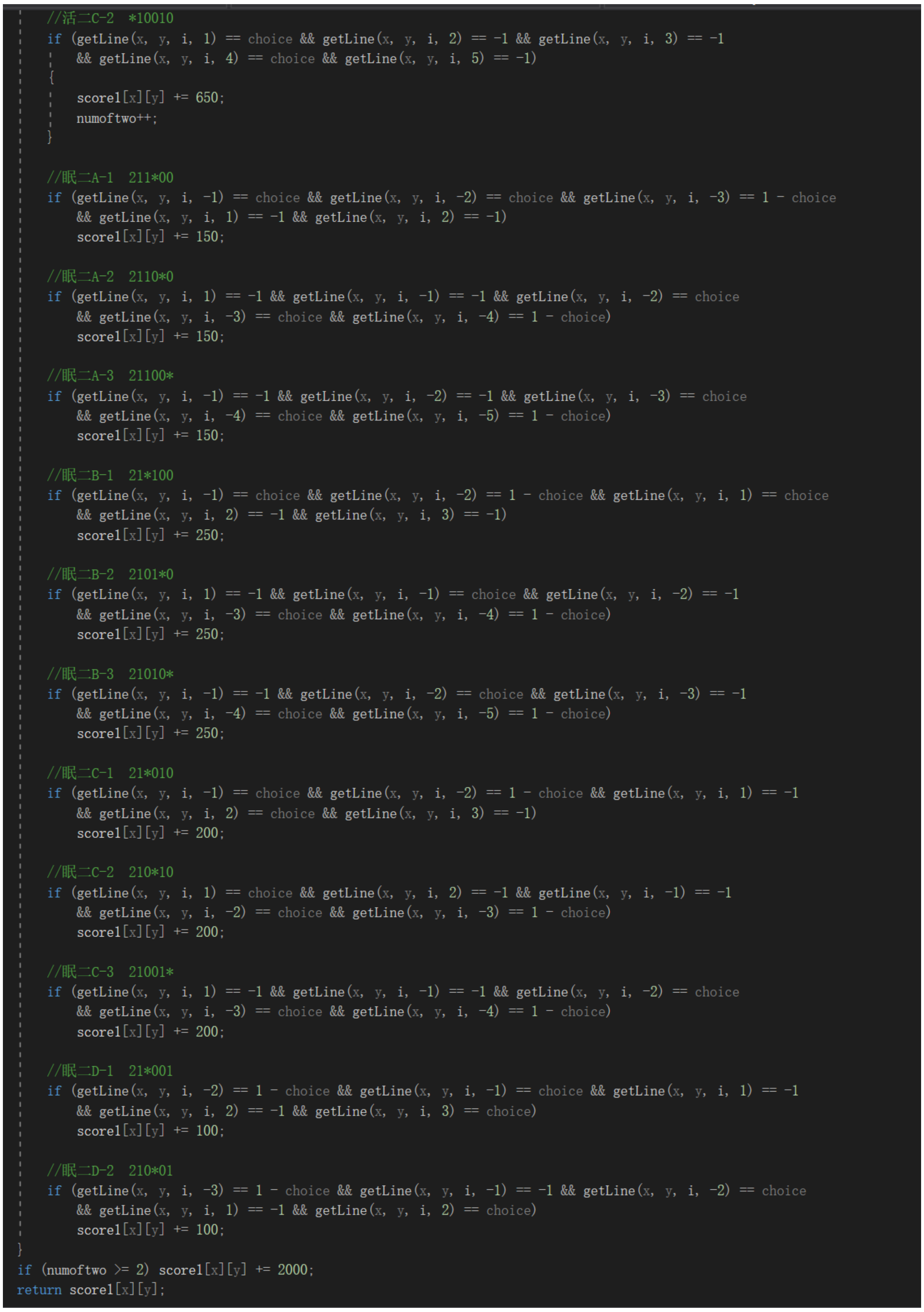
Based on the score table and the search from eight directions that whether this available position is in any valuable board type.



```cpp
int ChessBoard::EvaluateD(int x, int y, int choice)
{
    int numoftwo = 0;
    for (int i = 1; i <= 8; i++)
    {
        //活四  01111*
        if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == choice && getLine(x, y, i, 3) == choice
            && getLine(x, y, i, 4) == choice && getLine(x, y, i, 5) == -1)
            score1[x][y] += 300000;

        //冲四A  21111*
        if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == choice && getLine(x, y, i, 3) == choice
            && getLine(x, y, i, 4) == choice && getLine(x, y, i, 5) == 1 - choice)
            score1[x][y] += 200000;

        //冲四B  1*111
        if (getLine(x, y, i, -1) == choice && getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == choice
            && getLine(x, y, i, 3) == choice)
            score1[x][y] += 200000;

        //冲四C  11*11
        if (getLine(x, y, i, -2) == choice && getLine(x, y, i, -1) == choice && getLine(x, y, i, 1) == choice
            && getLine(x, y, i, 2) == choice)
            score1[x][y] += 200000;

        //活三A-1 0111*0
        if (getLine(x, y, i, 1) == -1 && getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == choice
            && getLine(x, y, i, -3) == choice && getLine(x, y, i, -4) == -1)
            score1[x][y] += 50000;

        //活三B-1  01*110
        if (getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == -1 && getLine(x, y, i, 1) == choice
            && getLine(x, y, i, 2) == choice && getLine(x, y, i, 3) == -1)
            score1[x][y] += 50000;

        //活三B-2  *10110
        if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == -1 && getLine(x, y, i, 3) == choice
            && getLine(x, y, i, 4) == choice && getLine(x, y, i, 5) == -1)
            score1[x][y] += 30000;

        //活三B-3  01011*
        if (getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == choice && getLine(x, y, i, -3) == -1
            && getLine(x, y, i, -4) == choice && getLine(x, y, i, -5) == -1)
            score1[x][y] += 30000;

        //眠三A-1 2111*0
        if (getLine(x, y, i, 1) == -1 && getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == choice
            && getLine(x, y, i, -3) == choice && getLine(x, y, i, -4) == 1 - choice)
            score1[x][y] += 2000;

        //眠三A-2 21110*
        if (getLine(x, y, i, -1) == -1 && getLine(x, y, i, -2) == choice && getLine(x, y, i, -3) == choice
            && getLine(x, y, i, -4) == choice && getLine(x, y, i, -5) == 1 - choice)
            score1[x][y] += 500;

        //眠三B-1 211*10
        if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == -1 && getLine(x, y, i, -1) == choice
            && getLine(x, y, i, -2) == choice && getLine(x, y, i, -3) == 1 - choice)
            score1[x][y] += 2000;

        //眠三B-2 21101*
        if (getLine(x, y, i, -1) == 1 - choice && getLine(x, y, i, -2) == -1 && getLine(x, y, i, -3) == choice
            && getLine(x, y, i, -4) == choice && getLine(x, y, i, -5) == 1 - choice)
            score1[x][y] += 800;
```



```
//眠三C-1  21*110
if (getLine(x, y, i, 1) == -1 && getLine(x, y, i, 2) == choice && getLine(x, y, i, 3) == choice
    && getLine(x, y, i, 4) == choice)
    score1[x][y] += 2000;

//眠三C-2  21011*
if (getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == choice && getLine(x, y, i, -3) == -1
    && getLine(x, y, i, -4) == choice && getLine(x, y, i, -5) == 1 - choice)
    score1[x][y] += 800;

//眠三D-1  211*01
if (getLine(x, y, i, -3) == 1 - choice && getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == choice
    && getLine(x, y, i, 1) == -1 && getLine(x, y, i, 2) == choice)
    score1[x][y] += 600;

//眠三D-2  2110*1
if (getLine(x, y, i, 1) == choice && getLine(x, y, i, -1) == -1 && getLine(x, y, i, -2) == choice
    && getLine(x, y, i, -3) == choice && getLine(x, y, i, -4) == 1 - choice)
    score1[x][y] += 600;

//眠三E   21*101
if (getLine(x, y, i, -2) == 1 - choice && getLine(x, y, i, -1) == choice && getLine(x, y, i, 1) == choice
    && getLine(x, y, i, 2) == -1 && getLine(x, y, i, 3) == choice)
    score1[x][y] += 550;

//活二A-1  0*0110
if (getLine(x, y, i, -1) == -1 && getLine(x, y, i, 1) == -1 && getLine(x, y, i, 2) == choice
    && getLine(x, y, i, 3) == choice && getLine(x, y, i, 4) == -1)
{
    score2[x][y] += 650;
    numoftwo++;
}

//活二A-2  0*110
if (getLine(x, y, i, -1) == -1 && getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == choice
    && getLine(x, y, i, 3) == -1)
{
    score2[x][y] += 3000;
    numoftwo++;
}

//活二B-1  0*1010
if (getLine(x, y, i, -1) == -1 && getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == -1
    && getLine(x, y, i, 3) == choice && getLine(x, y, i, 4) == -1)
{
    score2[x][y] += 650;
    numoftwo++;
}

//活二B-2  01*10
if (getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == -1 && getLine(x, y, i, 1) == choice
    && getLine(x, y, i, 2) == -1)
{
    score1[x][y] += 3000;
    numoftwo++;
}

//活二C-1  01*010
if (getLine(x, y, i, -2) == -1 && getLine(x, y, i, -1) == choice && getLine(x, y, i, 1) == -1
    && getLine(x, y, i, 2) == choice && getLine(x, y, i, 3) == -1)
{
    score1[x][y] += 650;
    numoftwo++;
}
```



```
    //活二C-2  *10010
    if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == -1 && getLine(x, y, i, 3) == -1
        && getLine(x, y, i, 4) == choice && getLine(x, y, i, 5) == -1)
    {
        score1[x][y] += 650;
        numoftwo++;
    }

    //眠二A-1  211*00
    if (getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == choice && getLine(x, y, i, -3) == 1 - choice
        && getLine(x, y, i, 1) == -1 && getLine(x, y, i, 2) == -1)
        score1[x][y] += 150;

    //眠二A-2  2110*0
    if (getLine(x, y, i, 1) == -1 && getLine(x, y, i, -1) == -1 && getLine(x, y, i, -2) == choice
        && getLine(x, y, i, -3) == choice && getLine(x, y, i, -4) == 1 - choice)
        score1[x][y] += 150;

    //眠二A-3  21100*
    if (getLine(x, y, i, -1) == -1 && getLine(x, y, i, -2) == -1 && getLine(x, y, i, -3) == choice
        && getLine(x, y, i, -4) == choice && getLine(x, y, i, -5) == 1 - choice)
        score1[x][y] += 150;

    //眠二B-1  21*100
    if (getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == 1 - choice && getLine(x, y, i, 1) == choice
        && getLine(x, y, i, 2) == -1 && getLine(x, y, i, 3) == -1)
        score1[x][y] += 250;

    //眠二B-2  2101*0
    if (getLine(x, y, i, 1) == -1 && getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == -1
        && getLine(x, y, i, -3) == choice && getLine(x, y, i, -4) == 1 - choice)
        score1[x][y] += 250;

    //眠二B-3  21010*
    if (getLine(x, y, i, -1) == -1 && getLine(x, y, i, -2) == choice && getLine(x, y, i, -3) == -1
        && getLine(x, y, i, -4) == choice && getLine(x, y, i, -5) == 1 - choice)
        score1[x][y] += 250;

    //眠二C-1  21*010
    if (getLine(x, y, i, -1) == choice && getLine(x, y, i, -2) == 1 - choice && getLine(x, y, i, 1) == -1
        && getLine(x, y, i, 2) == choice && getLine(x, y, i, 3) == -1)
        score1[x][y] += 200;

    //眠二C-2  210*10
    if (getLine(x, y, i, 1) == choice && getLine(x, y, i, 2) == -1 && getLine(x, y, i, -1) == -1
        && getLine(x, y, i, -2) == choice && getLine(x, y, i, -3) == 1 - choice)
        score1[x][y] += 200;

    //眠二C-3  21001*
    if (getLine(x, y, i, 1) == -1 && getLine(x, y, i, -1) == -1 && getLine(x, y, i, -2) == choice
        && getLine(x, y, i, -3) == choice && getLine(x, y, i, -4) == 1 - choice)
        score1[x][y] += 200;

    //眠二D-1  21*001
    if (getLine(x, y, i, -2) == 1 - choice && getLine(x, y, i, -1) == choice && getLine(x, y, i, 1) == -1
        && getLine(x, y, i, 2) == -1 && getLine(x, y, i, 3) == choice)
        score1[x][y] += 100;

    //眠二D-2  210*01
    if (getLine(x, y, i, -3) == 1 - choice && getLine(x, y, i, -1) == -1 && getLine(x, y, i, -2) == choice
        && getLine(x, y, i, 1) == -1 && getLine(x, y, i, 2) == choice)
        score1[x][y] += 100;
}
if (numoftwo >= 2) score1[x][y] += 2000;
return score1[x][y];
```

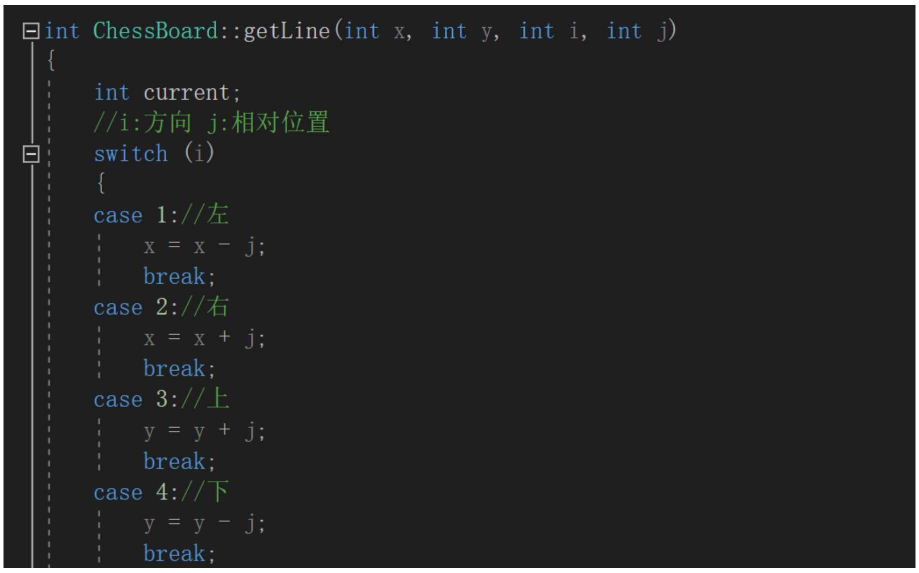
iv. How to evaluate 8 directions

As we all know, there are four kinds of rows in a Gomoku game so if we need to value a position, we need to value it in eight different directions. The getLine() function can help us get the coordinate in eight directions in a easy way and we can choose the distance we want.

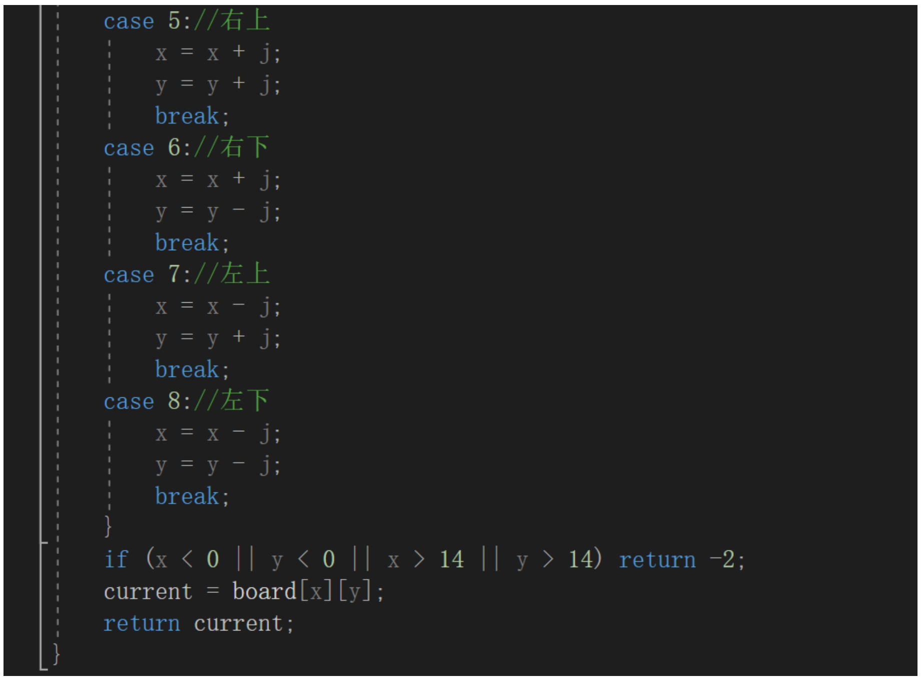
// x , y means the current position

// i is the direction

// j is the relative distance



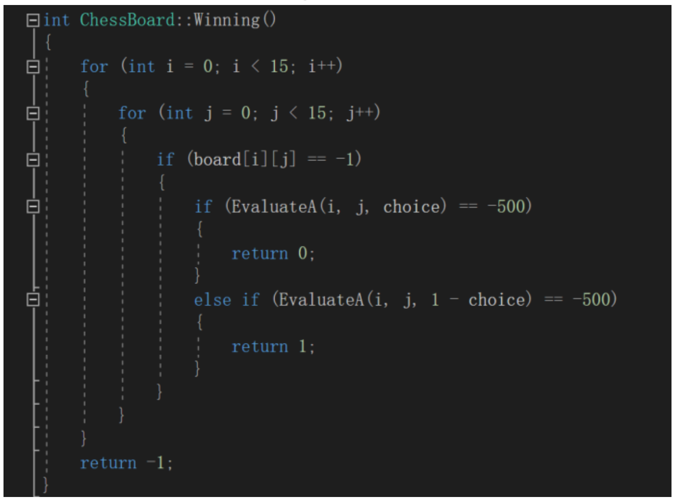
```
int ChessBoard::getLine(int x, int y, int i, int j)
{
    int current;
    //i:方向 j:相对位置
    switch (i)
    {
    case 1://左
        x = x - j;
        break;
    case 2://右
        x = x + j;
        break;
    case 3://上
        y = y + j;
        break;
    case 4://下
        y = y - j;
        break;
```



```
    case 5://右上
        x = x + j;
        y = y + j;
        break;
    case 6://右下
        x = x + j;
        y = y - j;
        break;
    case 7://左上
        x = x - j;
        y = y + j;
        break;
    case 8://左下
        x = x - j;
        y = y - j;
        break;
    }
    if (x < 0 || y < 0 || x > 14 || y > 14) return -2;
    current = board[x][y];
    return current;
}
```

v. The winning function

If the evaluation process detective a five-in-a-row situation, the score of that position will be -500. And if the winning function detective a score of -500, then the game will come to an end. This function will not be used when the program is put on the Botzone.

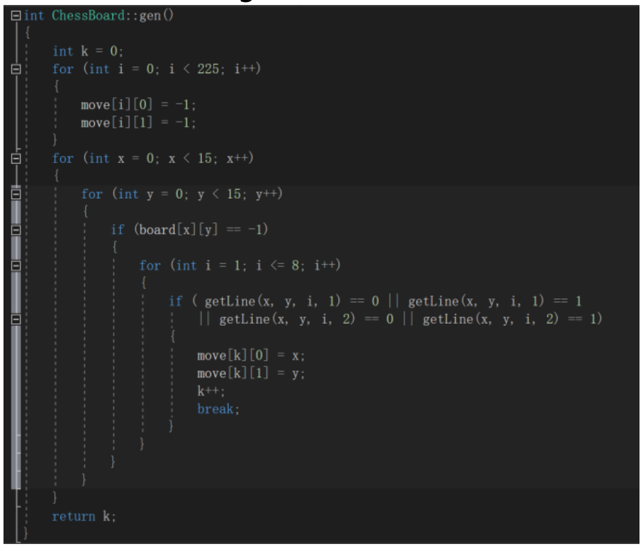


```
int ChessBoard::Winning()
{
    for (int i = 0; i < 15; i++)
    {
        for (int j = 0; j < 15; j++)
        {
            if (board[i][j] == -1)
            {
                if (EvaluateA(i, j, choice) == -500)
                {
                    return 0;
                }
                else if (EvaluateA(i, j, 1 - choice) == -500)
                {
                    return 1;
                }
            }
        }
    }
    return -1;
}
```

vi. The function that collect all the available position that has neighbors in the distance of 2

Based on experience, it is usually a waste of time to evaluate the positions that have no neighbors nearby. Therefore, I decided to evaluate only those positions that has neighbors in the distance of two in order to save the time of searching. Basically it just uses the getLine() function to search whether there is a black or white stone in the distance of two.

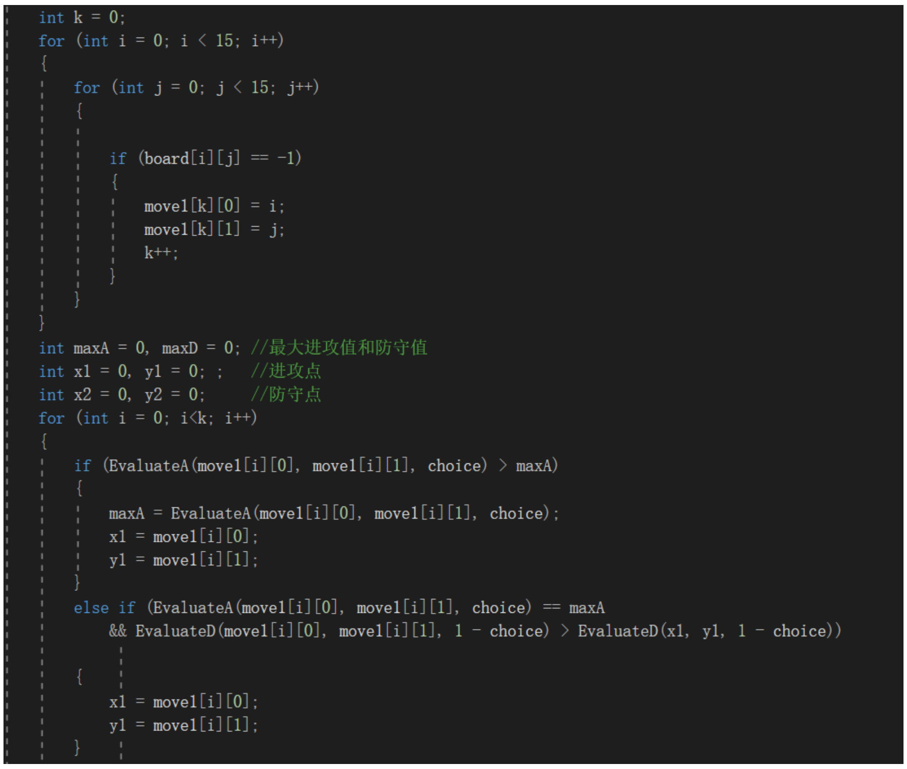
However, when applying this function on the Botzone, there seems to be a problem in the later period of the game, so I decided to use it in the future search algorithm instead of the current version.



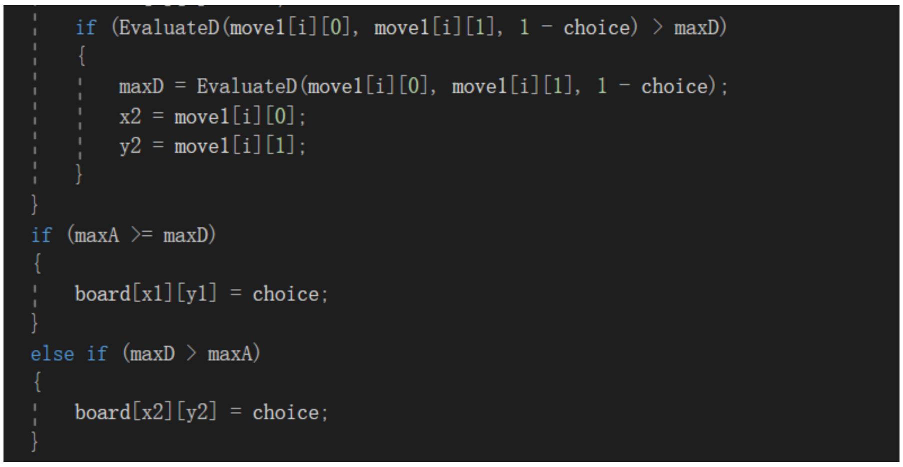
```
int ChessBoard::gen()
{
    int k = 0;
    for (int i = 0; i < 225; i++)
    {
        move[i][0] = -1;
        move[i][1] = -1;
    }
    for (int x = 0; x < 15; x++)
    {
        for (int y = 0; y < 15; y++)
        {
            if (board[x][y] == -1)
            {
                for (int i = 1; i <= 8; i++)
                {
                    if ( getLine(x, y, i, 1) == 0 || getLine(x, y, i, 1) == 1
                        || getLine(x, y, i, 2) == 0 || getLine(x, y, i, 2) == 1)
                    {
                        move[k][0] = x;
                        move[k][1] = y;
                        k++;
                        break;
                    }
                }
            }
        }
    }
    return k;
}
```

vii. The function that makes the next move

This is the main function in this version of design that makes the next move. It evaluates the attack potential and the defense potential of each position and picks out the position which has the max attack score and the position that has the max defense potential. Compare these two scores and choose the bigger one to make the move. If two positions have the same attack score, choose the one that has a bigger defense score in it to make the move.



```
int k = 0;
for (int i = 0; i < 15; i++)
{
    for (int j = 0; j < 15; j++)
    {

        if (board[i][j] == -1)
        {
            move1[k][0] = i;
            move1[k][1] = j;
            k++;
        }
    }
}
int maxA = 0, maxD = 0; //最大进攻值和防守值
int x1 = 0, y1 = 0; ;   //进攻点
int x2 = 0, y2 = 0;     //防守点
for (int i = 0; i<k; i++)
{
    if (EvaluateA(move1[i][0], move1[i][1], choice) > maxA)
    {
        maxA = EvaluateA(move1[i][0], move1[i][1], choice);
        x1 = move1[i][0];
        y1 = move1[i][1];
    }
    else if (EvaluateA(move1[i][0], move1[i][1], choice) == maxA
        && EvaluateD(move1[i][0], move1[i][1], 1 - choice) > EvaluateD(x1, y1, 1 - choice))

    {
        x1 = move1[i][0];
        y1 = move1[i][1];
    }
```



```
    if (EvaluateD(move1[i][0], move1[i][1], 1 - choice) > maxD)
    {
        maxD = EvaluateD(move1[i][0], move1[i][1], 1 - choice);
        x2 = move1[i][0];
        y2 = move1[i][1];
    }
}
if (maxA >= maxD)
{
    board[x1][y1] = choice;
}
else if (maxD > maxA)
{
    board[x2][y2] = choice;
}
```

Gomoku

Ver 2.0

09016401
顾婷瑄
Botzone id: Gutingxuan

1. Old versions review

Gomoku1.0 is simply based on an evaluation function and can not foresee future moves. Although it can comprehensively consider every available position and make the best move, it is not very competitive.

2. Literatures I read

[1] 王志水. 基于搜索算法的人工智能在五子棋博弈中的应用研究[D].山东：中国石油大学, 2006

[2] Blog: 五子棋基本棋型@我是老邱

五子棋估值算法@maxuewei2

3. Current idea of design

i. Essence

A function based on a new evaluation function and a MaxMin algorithm.

ii. Origin of the idea

The paper I’ve read.

iii. Expectation

The function can make tentative moves, evaluate the value of each state on both sides and then make the best choice.

iv. Implementation

A new evaluation function of the board and a MaxMin algorithm.

v. Advantages of the design

With the attempt to make possible moves, it can take more moves into consideration and thus can make a better choice in comparison with merely evaluating the value of current available positions.

vi. Weakness

With the increase of search depth and with the increase number of made moves, the time spent on calculating will increase exponentially so it’s quite slow when in practice. And since I didn’t use any pruning algorithm, the deepest depth it can go with is 2. A depth of 3 will take a couple of minutes to make the move and I can’t tolerate to test it.

vii. Improvement directions

I will try to optimize the searching algorithm in order to decrease the time spent on searching. And if possible, I will try to increase the searching depth. Perhaps I will use the alpha-beta pruning algorithm.

4. Evaluation of the performance

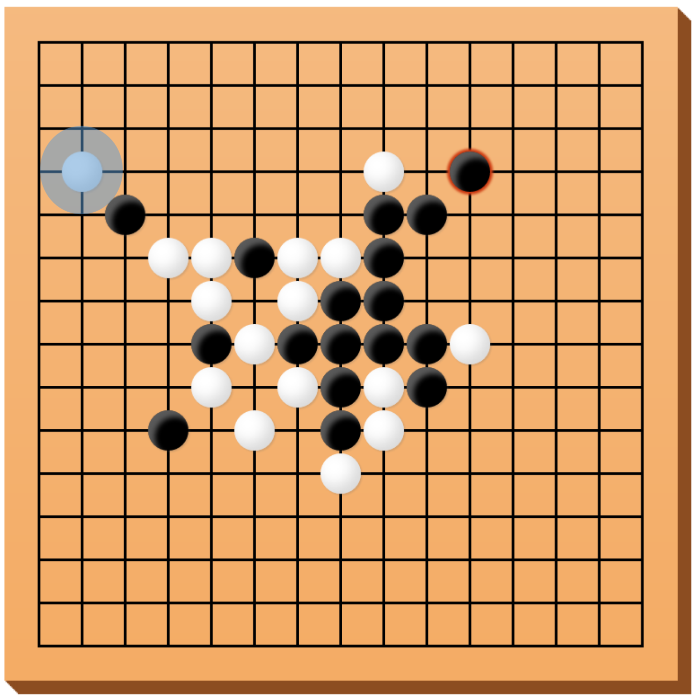
There are still some flaws in this version of Gomoku. Although it uses a more delicate algorithm, it didn’ t perform as well as I expected. For example, it sometimes ignores the obvious winning move of the opponent and then lose the game with no doubt.

Here are some recorded games with my Gomoku program on Botzone.org. As you can see, there still are some flaws in it. But from my perspective, it is better than the previous version.

Gutingxuan: me

Tttttttest: computer

Depth=1

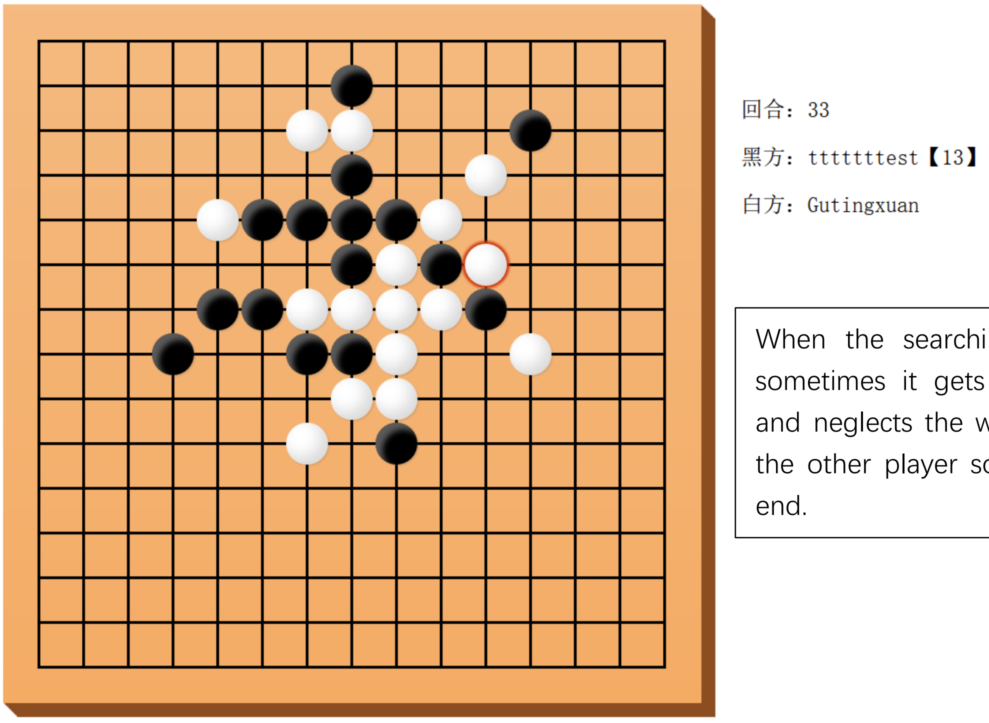


回合：34

黑方：Gutingxuan

白方：tttttttest【13】

I notice that when the computer finds that there is an inevitable failure, it just chooses the first vacant position it traversed to make the move. The stone circled is an example.

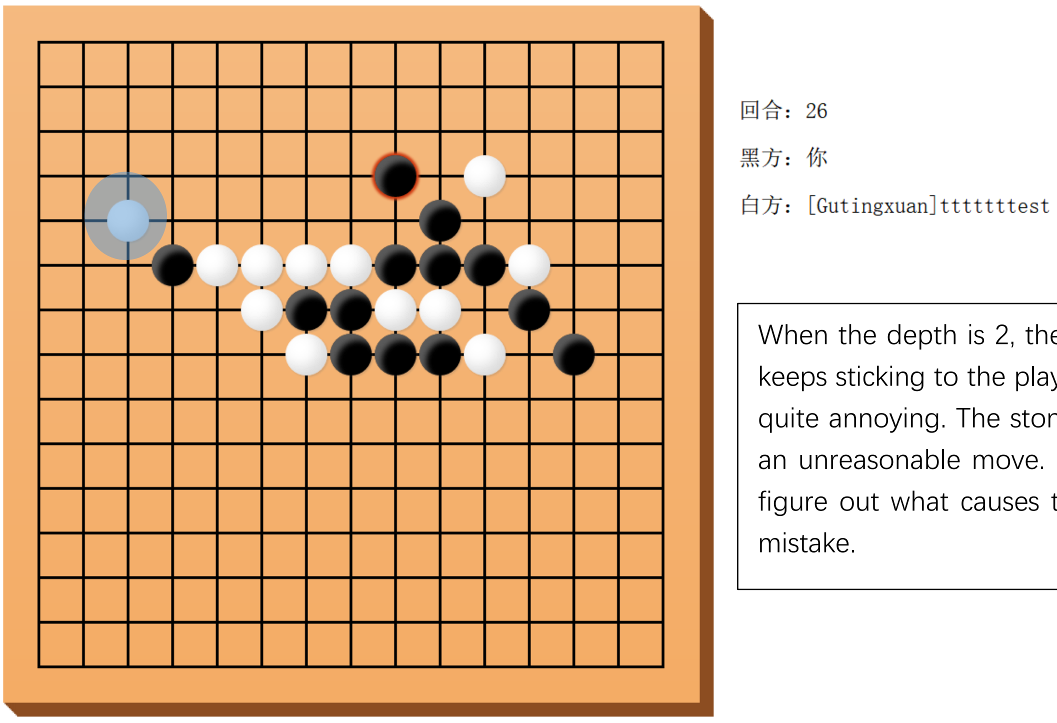

回合：33

黑方：tttttttest【13】

白方：Gutingxuan

When the searching depth is 1, sometimes it gets too aggressive and neglects the winning move of the other player so it loses at the end.

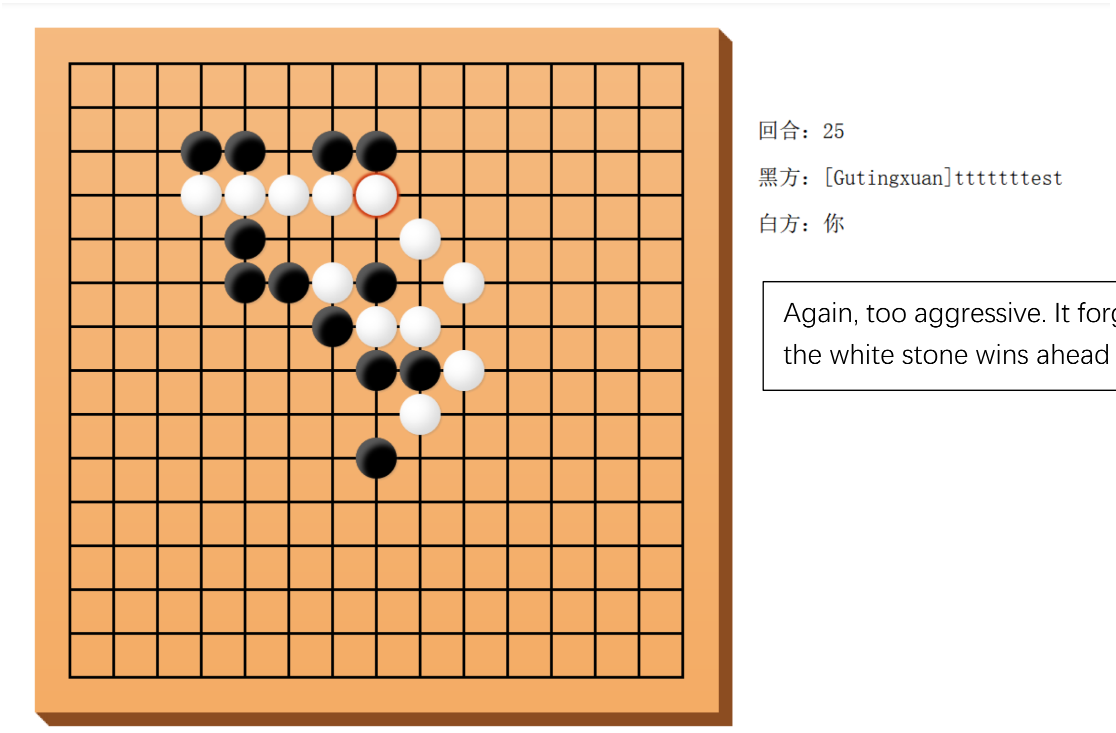
Depth=2


回合：26

黑方：你

白方：[Gutingxuan]tttttttest

When the depth is 2, the computer keeps sticking to the player which is quite annoying. The stone circled is an unreasonable move. I will try to figure out what causes this kind of mistake.



回合：25

黑方：[Gutingxuan]tttttttest

白方：你

Again, too aggressive. It forgets that the white stone wins ahead of it.

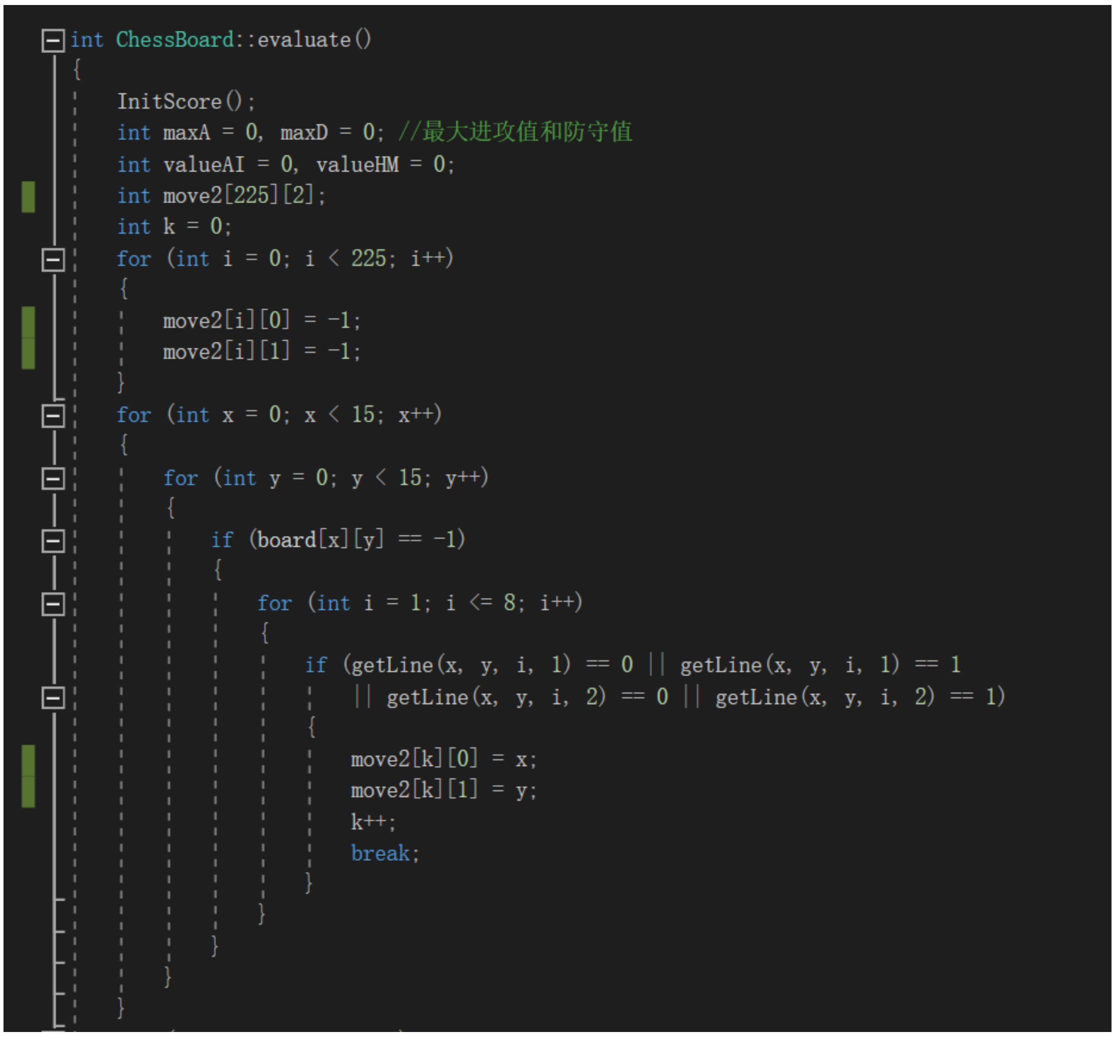
5. Code structure and detailed implementations

i. The new evaluation function

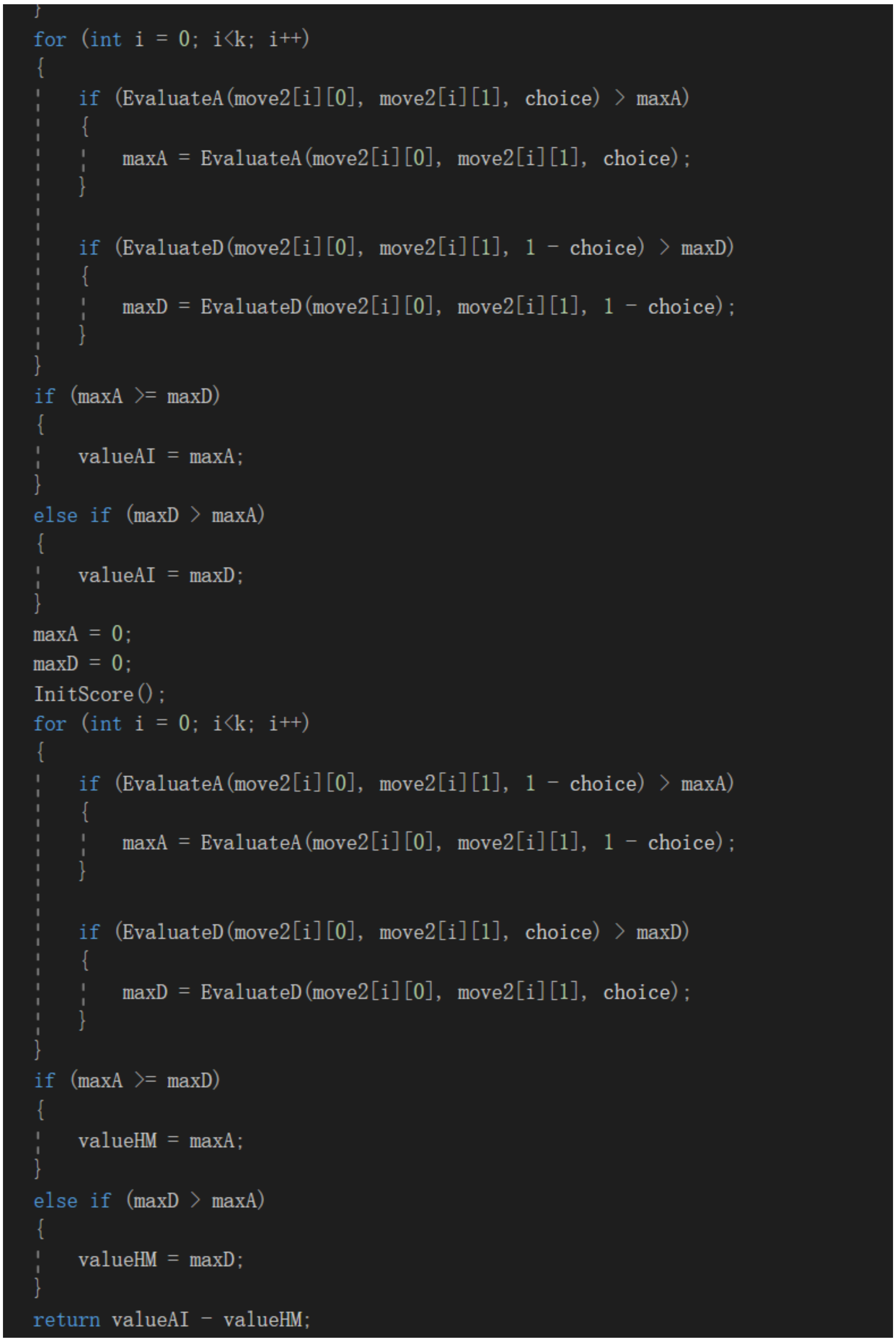
This function returns the current value of the board. It calculates the max position value of the player and the max position value of the computer and then makes a subtraction. The calculation of position value is based on former defense and attack evaluation functions.

Board value= computer value – player value

This function serves for the MaxMin algorithm.



```
int ChessBoard::evaluate()
{
    InitScore();
    int maxA = 0, maxD = 0; //最大进攻值和防守值
    int valueAI = 0, valueHM = 0;
    int move2[225][2];
    int k = 0;
    for (int i = 0; i < 225; i++)
    {
        move2[i][0] = -1;
        move2[i][1] = -1;
    }
    for (int x = 0; x < 15; x++)
    {
        for (int y = 0; y < 15; y++)
        {
            if (board[x][y] == -1)
            {
                for (int i = 1; i <= 8; i++)
                {
                    if (getLine(x, y, i, 1) == 0 || getLine(x, y, i, 1) == 1
                        || getLine(x, y, i, 2) == 0 || getLine(x, y, i, 2) == 1)
                    {
                        move2[k][0] = x;
                        move2[k][1] = y;
                        k++;
                        break;
                    }
                }
            }
        }
    }
```



```
}
for (int i = 0; i<k; i++)
{
    if (EvaluateA(move2[i][0], move2[i][1], choice) > maxA)
    {
        maxA = EvaluateA(move2[i][0], move2[i][1], choice);
    }

    if (EvaluateD(move2[i][0], move2[i][1], 1 - choice) > maxD)
    {
        maxD = EvaluateD(move2[i][0], move2[i][1], 1 - choice);
    }
}
if (maxA >= maxD)
{
    valueAI = maxA;
}
else if (maxD > maxA)
{
    valueAI = maxD;
}
maxA = 0;
maxD = 0;
InitScore();
for (int i = 0; i<k; i++)
{
    if (EvaluateA(move2[i][0], move2[i][1], 1 - choice) > maxA)
    {
        maxA = EvaluateA(move2[i][0], move2[i][1], 1 - choice);
    }

    if (EvaluateD(move2[i][0], move2[i][1], choice) > maxD)
    {
        maxD = EvaluateD(move2[i][0], move2[i][1], choice);
    }
}
if (maxA >= maxD)
{
    valueHM = maxA;
}
else if (maxD > maxA)
{
    valueHM = maxD;
}
return valueAI - valueHM;
```

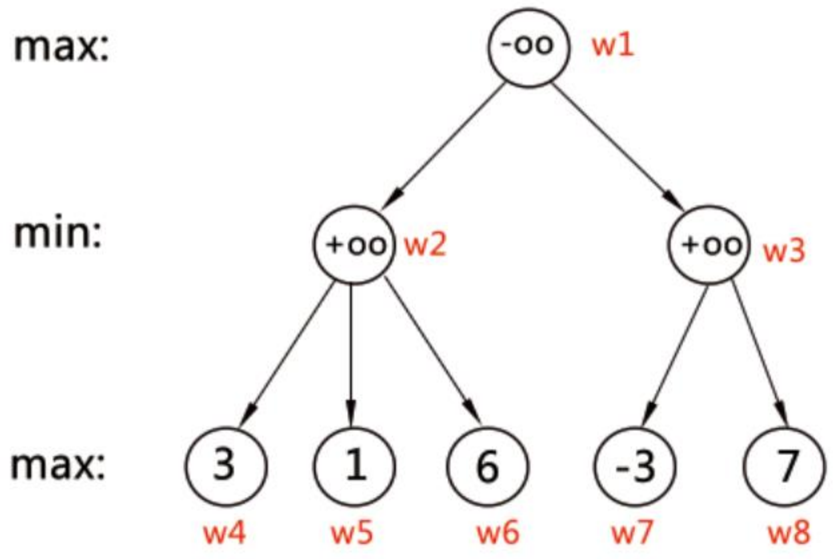
ii. The MaxMin algorithm

The evaluation function: We use this to evaluate the value of each state.

The board value = computer value – player value

Max state: This is the state that computer is the one to make the move. There are some possible moves and each of them will lead to a sub-state. Since computer is going to make the move, it will try to maximize the board value so this is called the Max state.

Min state: This is the state that opponent is going to make the move. The move it makes will also lead to different sub-states. From it’ s perspective, it will try to minimize the board value so this is called the Min state.



Take the game tree above as an example.

Assume the first player is A, the second player is B, then A is a max state and B is a min state. A has 2 choices at first, w2 and w3. Whether it goes with w2 or w3 depends on the evaluation function. Since it is a max state, so it will choose the bigger one between f(w2) and f(w3).

Here are the searching paths:

w1->w2->w4: w2 is now 3

w1->w2->w5: w5 is less than w4 and w2 is a min state so it will go with w5. w2 is now 1.

w1->w2->w6: w6 is 6 so we don’ t take it.

Finally, w2 is 1.

w1->w3->w7: w3 is now -3

w1->w3->w8: w8 is 7 so we don’ t take it.

Finally, w3 is -3.

Since w1 is a max state, so its next move will be w2.

Pseudocode from Wikipedia:

```
function minimax(node, depth)
    if node is a terminal node or depth = 0
        return the heuristic value of node
    if the adversary is to play at node
        let α := +∞
        foreach child of node
            α := min(α, minimax(child, depth-1))
    else {we are to play at node}
        let α := -∞
        foreach child of node
            α := max(α, minimax(child, depth-1))
    return α
```

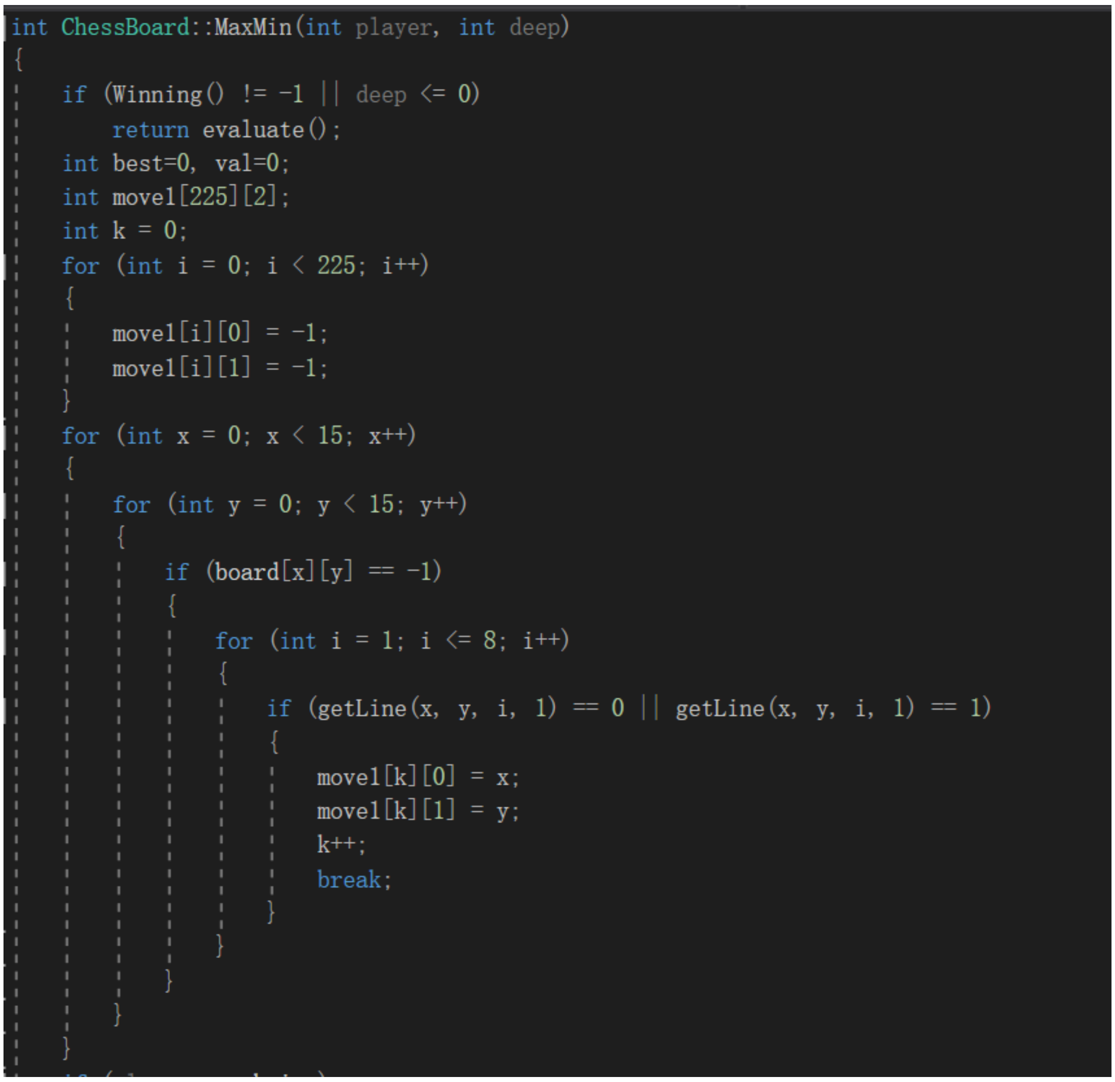
Pseudocode from a blog：This helps me better understand the algorithm

```
// player = 1 表示轮到己方， player = 0 表示轮到对方
// cur_node 表示当前局面(结点)
maxmin(player, cur_node)
{
    if 达到终结局面
        return 该局面结点的估价值 f
    end
    if player == 1 // 轮到己方走
        best = -oo // 己方估价值初始化为 -oo
        for 每一种走法 do
            new_node = get_next(cur_node) // 遍历当前局面 cur_node 的所有子局面
            val = maxmin(player^1, new_node); // 把新产生的局面交给对方，对方返回一个该局面的估价值
            if val > best
                best = val;
            end
        end
        return best;
    else // 轮到对方走
        best = +oo // 对方估价值初始化为 +oo
        for 每一种走法 do
            new_node = get_next(cur_node) // 遍历当前局面 cur_node 的所有子局面
            val = maxmin(player^1, new_node); // 把新产生的局面交给对方，对方返回一个该局面的估价值
            if val < best
                best = val;
            end
        end
        return best;
    end
}
```

My MaxMin function:

The generator function part:

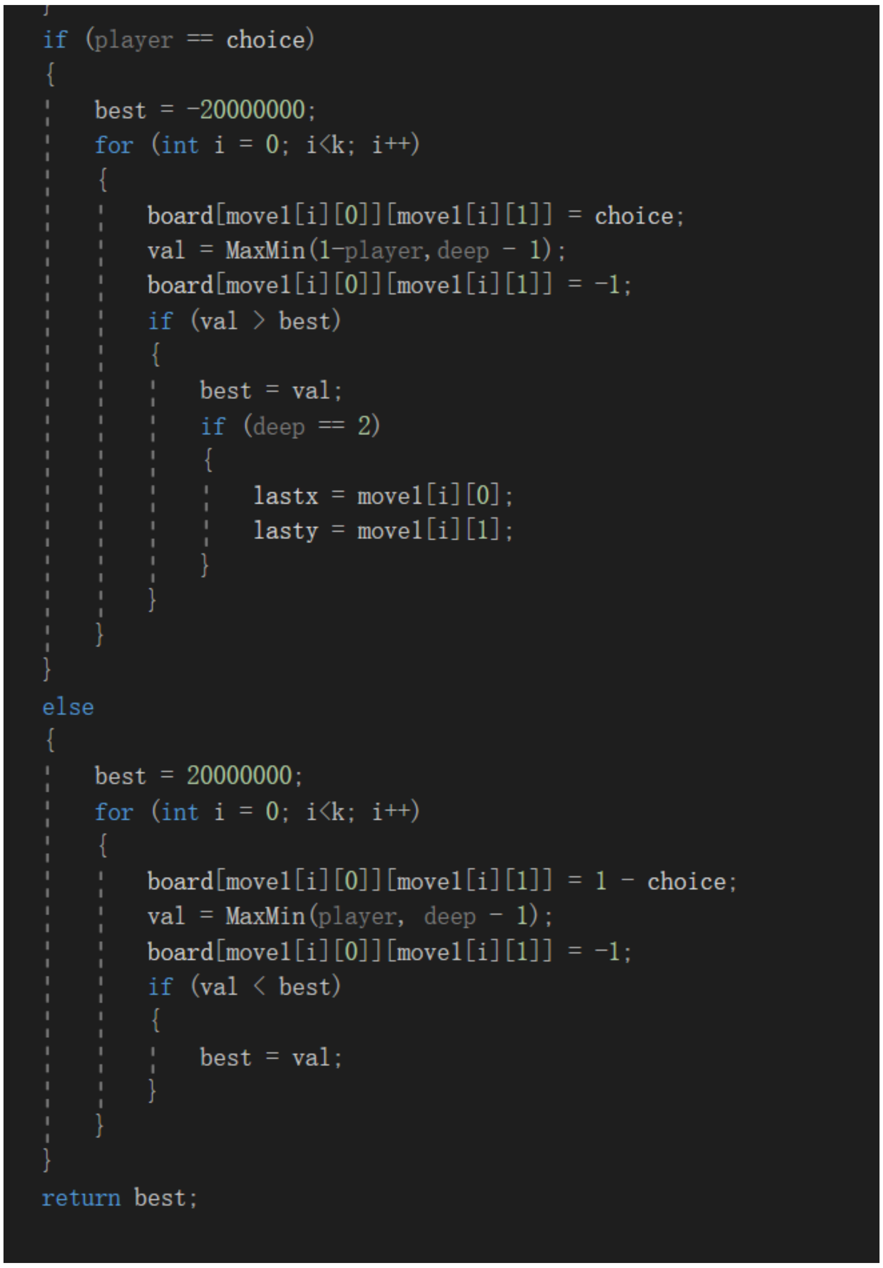
These lines basically form a generator function which collects available positions that have neighbors, which means there is a previous stone nearby. Otherwise there will be too many available positions that need to be traversed and this will increase the searching time.



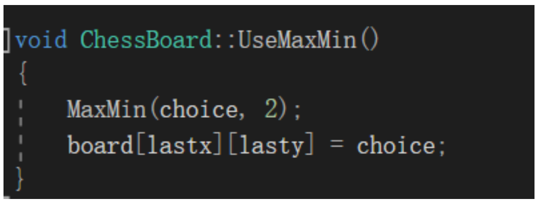
```
int ChessBoard::MaxMin(int player, int deep)
{
    if (Winning() != -1 || deep <= 0)
        return evaluate();
    int best=0, val=0;
    int move1[225][2];
    int k = 0;
    for (int i = 0; i < 225; i++)
    {
        move1[i][0] = -1;
        move1[i][1] = -1;
    }
    for (int x = 0; x < 15; x++)
    {
        for (int y = 0; y < 15; y++)
        {
            if (board[x][y] == -1)
            {
                for (int i = 1; i <= 8; i++)
                {
                    if (getLine(x, y, i, 1) == 0 || getLine(x, y, i, 1) == 1)
                    {
                        move1[k][0] = x;
                        move1[k][1] = y;
                        k++;
                        break;
                    }
                }
            }
        }
    }
```

The body part of the algorithm:

The computer will make a move so as to simulate a sub-state when traversing sub-states. After traversing this sub-state, it will withdraw this move.



```
if (player == choice)
{
    best = -20000000;
    for (int i = 0; i<k; i++)
    {
        board[move1[i][0]][move1[i][1]] = choice;
        val = MaxMin(1-player,deep - 1);
        board[move1[i][0]][move1[i][1]] = -1;
        if (val > best)
        {
            best = val;
            if (deep == 2)
            {
                lastx = move1[i][0];
                lasty = move1[i][1];
            }
        }
    }
}
else
{
    best = 20000000;
    for (int i = 0; i<k; i++)
    {
        board[move1[i][0]][move1[i][1]] = 1 - choice;
        val = MaxMin(player, deep - 1);
        board[move1[i][0]][move1[i][1]] = -1;
        if (val < best)
        {
            best = val;
        }
    }
}
return best;
```



```
void ChessBoard::UseMaxMin()
{
    MaxMin(choice, 2);
    board[lastx][lasty] = choice;
}
```

Complexity analysis:

If there are (n) sub-states for each move, considering the depth as (d), the worst case would be O(n^b).

Gomoku

Ver 2.1

09016401
顾婷瑄
Botzone id: Gutingxuan

1. Old versions review

Gomoku2.0 is simply based on a MaxMin algorithm and although it can foresee future moves, it spends a lot of time on doing calculations so the deepest searching level it can reach is a mere three.

2. Literatures I read

[1] 王志水. 基于搜索算法的人工智能在五子棋博弈中的应用研究[D].山东：中国石油大学, 2006

[2] Blog: 五子棋基本棋型@我是老邱

五子棋估值算法@maxuewei2

3. Current idea of design

i. Essence

A function based on the MaxMin algorithm and the alpha-beta pruning.

ii. Origin of the idea

The paper I’ve read.

iii. Expectation

The function can make tentative moves, evaluate the value of each state on both sides and then make the best choice. Furthermore, the time it spends to do the calculations will be decreased.

iv. Implementation

A MaxMin algorithm, alpha-beta pruning, new evaluation method.

v. Advantages of the design

With the alpha-beta pruning, the time spent on calculating can be decreased. I sort the values of the available positions before searching and only visit the first 12 positions, so the same number of searching nodes can lead to a deeper searching level. Since there is a time limit on botzone, so my current searching level is 6. And with the new evaluation method, the current board value will be calculated more accurately.

vi. Weakness

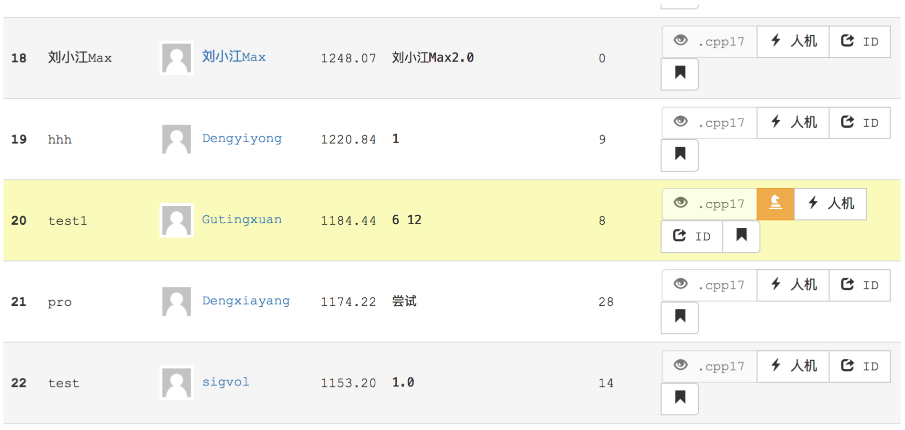
The alpha-beta pruning relies on the order of the positions it visits. If it goes to the worst position first, then the beta-pruning won’t happen. So, I have done the sorting before making attempted moves, which is to say the function will visit those most likely moves first. And to save some searching time, I choose to only visit the first twelve positions. So if there’ s any flaw in my evaluation method of each position, it will affect the alpha-beta pruning.

vii. Improvement directions

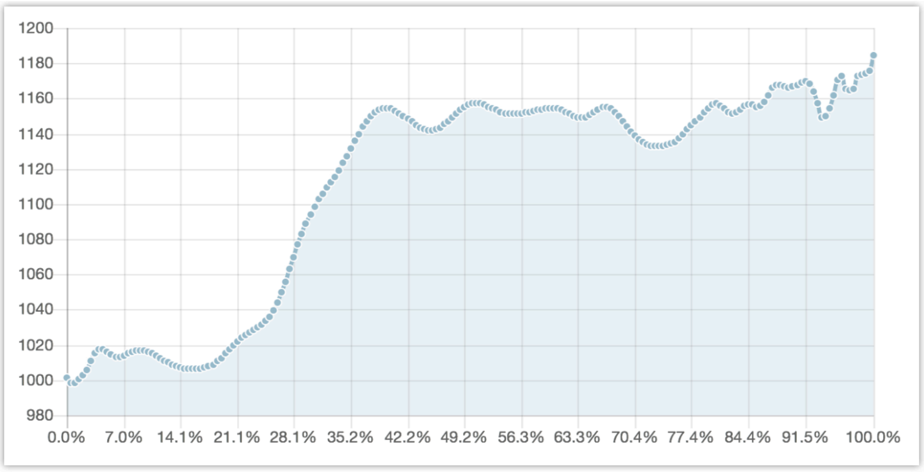
I will try to optimize the evaluation method and try to improve my algorithm.

4. Evaluation of the performance

Current ranking on Botzone: 20/37 (2018/5/3 20:00)



18	刘小江Max	刘小江Max	1248.07	刘小江Max2.0	0	.cpp17 人机 ID
19	hhh	Dengyiyong	1220.84	1	9	.cpp17 人机 ID
20	test1	Gutingxuan	1184.44	6 12	8	.cpp17 人机 ID
21	pro	Dengxiayang	1174.22	尝试	28	.cpp17 人机 ID
22	test	sigvol	1153.20	1.0	14	.cpp17 人机 ID



总趋势

The score increased a lot after I limit the number of the searching nodes. Because there is a time limit on botzone so if the time my gomoku program takes exceeds the time limit, it just loses the game.

5. Code structure and detailed implementations

i. Alpha-beta pruning

Explanation from wikipedia:

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Pseudocode:

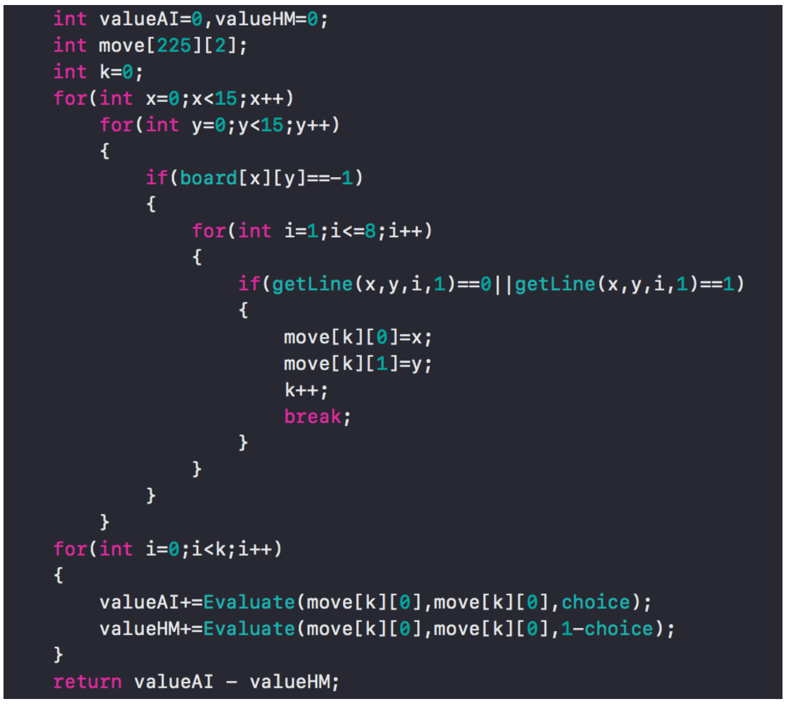
```
int AlphaBeta(int depth, int alpha, int beta)
{
    if (depth == 0)
    {
        return Evaluate();
    }
    GenerateLegalMoves();
    while (MovesLeft())
    {
        MakeNextMove();
        val = -AlphaBeta(depth - 1, -beta, -alpha);
        UnmakeMove();
        if (val >= beta)
        {
            return beta;
        }
        if (val > alpha)
        {
            alpha = val;
        }
    }
    return alpha;
}
```

My alpha-beta pruning part:

```
int count = 0;
for (int i = 0; i < k; i++)
{
    board[move1[list[i]][0]][move1[list[i]][1]] = choice;
    val = -AlphaBeta(deep - 1, -beta, -alpha);
    board[move1[list[i]][0]][move1[list[i]][1]] = -1;
    if (val >= beta)
    {
        return beta;
    }
    if (val > alpha)
    {
        alpha = val;
        if(deep == 8)
        {
            lastx = move1[list[i]][0];
            lasty = move1[list[i]][1];
        }
    }
    if(count++ >= 12)   break;
}
return alpha;
```

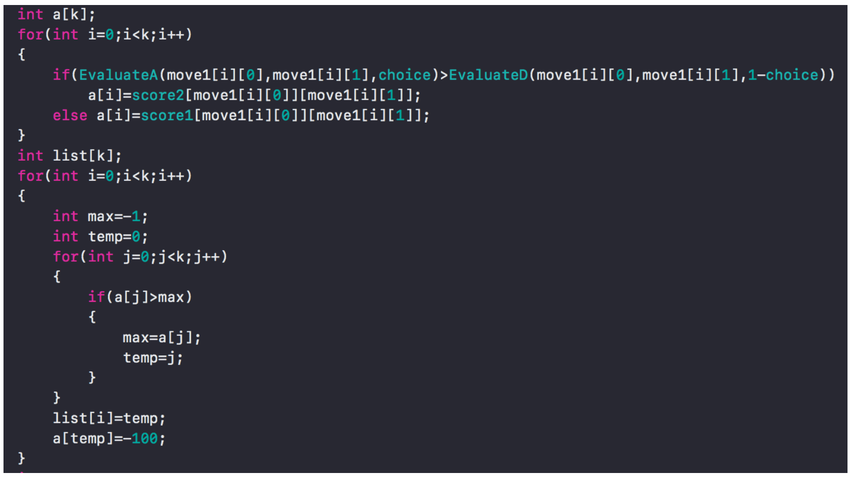
ii. New Evaluation method

In the former version, I choose the most valuable position score as current board value. Actually it's not very wise. In this version, I choose to collect all those positions that can detect a three-in-a-row or a four-in-a-row and add up all those scores as the total board value.



```
    int valueAI=0,valueHM=0;
    int move[225][2];
    int k=0;
    for(int x=0;x<15;x++)
        for(int y=0;y<15;y++)
        {
            if(board[x][y]==-1)
            {
                for(int i=1;i<=8;i++)
                {
                    if(getLine(x,y,i,1)==0||getLine(x,y,i,1)==1)
                    {
                        move[k][0]=x;
                        move[k][1]=y;
                        k++;
                        break;
                    }
                }
            }
        }
    for(int i=0;i<k;i++)
    {
        valueAI+=Evaluate(move[k][0],move[k][0],choice);
        valueHM+=Evaluate(move[k][0],move[k][0],1-choice);
    }
    return valueAI - valueHM;
```

iii. Extra tricks

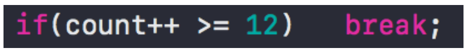


```
int a[k];
for(int i=0;i<k;i++)
{
    if(EvaluateA(move1[i][0],move1[i][1],choice)>EvaluateD(move1[i][0],move1[i][1],1-choice))
        a[i]=score2[move1[i][0]][move1[i][1]];
    else a[i]=score1[move1[i][0]][move1[i][1]];
}
int list[k];
for(int i=0;i<k;i++)
{
    int max=-1;
    int temp=0;
    for(int j=0;j<k;j++)
    {
        if(a[j]>max)
        {
            max=a[j];
            temp=j;
        }
    }
    list[i]=temp;
    a[temp]=-100;
}
```

①

Obviously, the position with a higher value is more likely to be chosen. So these lines are to sort the values of the available positions before the actual searching to enhance the ability of alpha-beta pruning.

②



```
if(count++ >= 12)   break;
```

This is to choose the first 12 valuable positions to visit in alpha-beta pruning in order to save time on visiting nodes on the same level. So the searching level can be deeper.