

**计算机专业类课程**

**实验报告**

**课程名称：W0801220.01高级程序设计**

**学　　院：计算机科学与工程学院**

**学生姓名：马彦江**

**学　　号：2022080902003**

**指导教师：俸志刚**

**日　　期：　　　　2022年　　12月　26日**

电子科技大学计算机学院实验中心

**电 子 科 技 大 学**

**实 验 报 告**

**实验一**

**一 实验项目名称：五子棋AI**

**二 实验目的：**实现包括以下功能的五子棋AI(执白)

1 无禁手

2 无时间限制

3 显示当前落子

**三：实验器材：**

* 开发环境：VisualStudio2022
* 配置：Debug
* 操作系统：Windows11
* GUI：Easy X

**四：实验步骤：**

**1.问题描述：**使用基于Easy X的GUI实现鼠标下棋（无时限）

**2.算法分析与概要设计**

1. **MINMAX（全局估值）**

**(1)棋型表示**

6维数组表示棋型（活4需6个点表示)->type[4][4][4][4][4][4]//0:空1:玩家2:AI 3:界外

**(2)权重**

OTHER 0;// 其他棋型

WIN 1000000;//AI赢

LOSE -10000000;//玩家赢

FLEX4 50000;//大写：AI

flex4 -100000;//flex 活棋

BLOCK4 400;//block 冲棋

block4 -10000;//小写：玩家

FLEX3 400;

flex3 -8000;

BLOCK3 20;

block3 -50;

FLEX2 20;

flex2 -50;

BLOCK2 1;

block2 -3;

FLEX1 1;

flex1 -3;

1. ****alpha beta剪枝****

****（1）原理概述****

Alpha-Beta只能用递归来实现。这个思想是在搜索中传递两个值，第一个值是Alpha，即搜索到的最好值，任何比它更小的值就没用了，因为策略就是知道Alpha的值，任何小于或等于Alpha的值都不会有所提高。第二个值是Beta，即对于对手来说最坏的值。这是对手所能承受的最坏的结果，因为我们知道在对手看来，他总是会找到一个对策不比Beta更坏的。如果搜索过程中返回Beta或比Beta更好的值，那就够好的了，走棋的一方就没有机会使用这种策略了。在搜索着法时，每个搜索过的着法都返回跟Alpha和Beta有关的值，它们之间的关系非常重要，或许意味着搜索可以停止并返回。如果某个着法的结果小于或等于Alpha，那么它就是很差的着法，因此可以抛弃。因为我前面说过，在这个策略中，局面对走棋的一方来说是以Alpha为评价的。如果某个着法的结果大于或等于Beta，那么整个节点就作废了，因为对手不希望走到这个局面，而它有别的着法可以避免到达这个局面。因此如果我们找到的评价大于或等于Beta，就证明了这个结点是不会发生的，因此剩下的合理着法没有必要再搜索。如果某个着法的结果大于Alpha但小于Beta，那么这个着法就是走棋一方可以考虑走的，除非以后有所变化。因此Alpha会不断增加以反映新的情况。有时候可能一个合理着法也不超过Alpha，这在实战中是经常发生的，此时这种局面是不予考虑的，因此为了避免这样的局面，我们必须在博弈树的上一个层局面选择另外一个着法。

****（2）伪代码：****function alphabeta(node, depth, α, β, maximizingPlayer) // node = 节点，depth = 深度，maximizingPlayer = 大分玩家

**if** depth = 0 or node是终端节点

**return** 节点的启发值

**if** maximizingPlayer v := -∞

**for** 每个子节点

 v := max(v, alphabeta(child, depth - 1, α, β, FALSE)) // child = 子节点

              α := max(α, v)

**if** β ≤ α        **break** // β裁剪

**return** v

**else** v := ∞

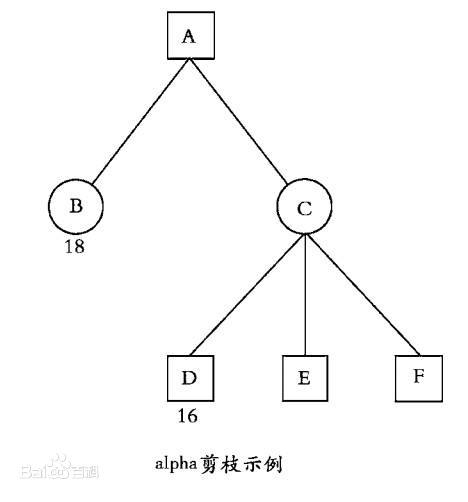
**for** each 每个子节点

              v := min(v, alphabeta(child, depth - 1, α, β, TRUE))

              β := min(β, v)

**if** β ≤ α **break** // α裁剪 **return** v

 (\* 初始调用\*)alphabeta(origin, depth, -∞, +∞, TRUE) // origin = 初始节点



**（3）原理图示（上图）**

**三、核心算法的详细设计与实现**

****（1）棋型初始化****

**1.直接赋值**

type[2][2][2][2][2][2] = WIN;

type[2][2][2][2][2][0] = WIN;

type[0][2][2][2][2][2] = WIN;

type[2][2][2][2][2][1] = WIN;

type[1][2][2][2][2][2] = WIN;

type[3][2][2][2][2][2] = WIN;

type[2][2][2][2][2][3] = WIN;

type[1][1][1][1][1][1] = LOSE;

type[1][1][1][1][1][0] = LOSE;

type[0][1][1][1][1][1] = LOSE;

type[1][1][1][1][1][2] = LOSE;

type[2][1][1][1][1][1] = LOSE;

type[3][1][1][1][1][1] = LOSE;

type[1][1][1][1][1][3] = LOSE;

type[0][2][2][2][2][0] = FLEX4;

type[0][1][1][1][1][0] = flex4;

type[0][2][2][2][0][0] = FLEX3;

type[0][0][2][2][2][0] = FLEX3;

type[0][2][0][2][2][0] = FLEX3;

type[0][2][2][0][2][0] = FLEX3;

type[0][1][1][1][0][0] = flex3;

type[0][0][1][1][1][0] = flex3;

type[0][1][0][1][1][0] = flex3;

type[0][1][1][0][1][0] = flex3;

type[0][2][2][0][0][0] = FLEX2;

type[0][2][0][2][0][0] = FLEX2;

type[0][2][0][0][2][0] = FLEX2;

type[0][0][2][2][0][0] = FLEX2;

type[0][0][2][0][2][0] = FLEX2;

type[0][0][0][2][2][0] = FLEX2;

type[0][1][1][0][0][0] = flex2;

type[0][1][0][1][0][0] = flex2;

type[0][1][0][0][1][0] = flex2;

type[0][0][1][1][0][0] = flex2;

type[0][0][1][0][1][0] = flex2;

type[0][0][0][1][1][0] = flex2;

type[0][2][0][0][0][0] = FLEX1;

type[0][0][2][0][0][0] = FLEX1;

type[0][0][0][2][0][0] = FLEX1;

type[0][0][0][0][2][0] = FLEX1;

type[0][1][0][0][0][0] = flex1;

type[0][0][1][0][0][0] = flex1;

type[0][0][0][1][0][0] = flex1;

type[0][0][0][0][1][0] = flex1;

**2.间接赋值，穷举各种情况(**关键实现代码)

int p1, p2, p3, p4, p5, p6, x, y, ix, iy;//x:左5中白个数,y:左5黑'个数,ix:右5中白个数,iy:右5中黑个数

for (p1 = 0; p1 <= 3; ++p1) {

for (p2 = 0; p2 <= 2; ++p2) {

for (p3 = 0; p3 <= 2; ++p3) {

for (p4 = 0; p4 <= 2; ++p4) {

for (p5 = 0; p5 <= 2; ++p5) {

for (p6 = 0; p6 <= 3; ++p6) {

x = y = ix = iy = 0;

if (p1 == 1) { ++x; }

else if (p1 == 2) { ++y; }

if (p2 == 1) { ++x; ++ix; }

else if (p2 == 2) { ++y; ++iy; }

if (p3 == 1) { ++x; ++ix; }

else if (p3 == 2) { ++y; ++iy; }

if (p4 == 1) { ++x; ++ix; }

else if (p4 == 2) { ++y; ++iy; }

if (p5 == 1) { ++x; ++ix; }

else if (p5 == 2) { ++y; ++iy; }

if (p6 == 1) { ++ix; }

else if (p6 == 2) { ++iy; }

****（3）全局估值****

//判断竖向棋型

for (i = 1; i <= 15; ++i) {

for (j = 0; j <= 11; ++j) {

Type = type[Board[i][j]][Board[i][j + 1]][Board[i][j + 2]][Board[i][j + 3]][Board[i][j + 4]][Board[i][j + 5]]; ++sample[0][Type];} }

//判断横向棋型

for (j = 1; j <= 15; ++j) {

for (i = 0; i <= 11; ++i) {Type = type[Board[i][j]][Board[i + 1][j]][Board[i + 2][j]][Board[i+3][j]][Board[i + 4][j]][Board[i + 5][j]]; ++sample[1][Type];}}

//判断左上至右下棋型

for (i = 0; i <= 11; ++i) {

for (j = 16; j >= 5; --j) {

Type = type[Board[i][j]][Board[i + 1][j - 1]][Board[i + 2][j - 2]][Board[i + 3][j - 3]][Board[i + 4][j - 4]][Board[i + 5][j - 5]];

++sample[2][Type];}}

//判断右上至左下棋型

for (i = 0; i <= 11; ++i) {

for (j = 11; j >= 0; --j) {

Type = type[Board[i][j]][Board[i + 1][j + 1]][Board[i + 2][j + 2]][Board[i + 3][j + 3]][Board[i + 4][j + 4]][Board[i + 5][j + 5]];

++sample[3][Type];}}

**（4）偏僻剪枝**

for (int i = 1; i <= 15; ++i) {//每个非空点附近8个方向延伸3个深度,若不越界则标记可走

for (int j = 1; j <= 15; ++j) {

if (Board[i][j] != 0) {

for (int k = -num; k <= num; ++k) {

if (i + k >= 1 && i + k <= 15) {

isEmpty[i + k][j] = 1;//非偏僻

if (j + k >= 1 && j + k <= 15) { isEmpty[i + k][j + k] = 1; }

if (j - k >= 1 && j - k <= 15) { isEmpty[i + k][j - k] = 1; }

}

if (j + k >= 1 && j + k <= 15) { isEmpty[i][j + k] = 1; }}}}}

****（5）GUI****

**1.棋盘**

initgraph(480, 480, SHOWCONSOLE);

setbkcolor(BROWN);

cleardevice();

//画线

setlinestyle(PS\_SOLID, 1);//直线宽度

setlinecolor(BLACK);

for (i = 30; i <= 480; i += 30) //横线

{line(30, i, 450, i);}

for (j = 30; j <= 480; j += 30) //竖线

{line(j, 30, j, 450); }

**2.鼠标下棋**

for (int i = 1; i <= 15; ++i)

{

for (int j = 1; j <= 15; ++j)

{

if (abs(Mouse.x - i \* 30) < 10 && abs(Mouse.y - j \* 30) < 10)

//将点击的范围内的值取整{

Mouse.x = i \* 30;

Mouse.y = j \* 30;

flag = 1;}} }

//防止重叠

if (!Board[Mouse.x / 30][Mouse.y / 30]&&flag==1) {

setfillcolor(BLACK);

solidcircle(Mouse.x, Mouse.y, 12); ...}

****(6)判断胜负****

**1.纵横**

for (i=1; i <= 15; ++i) {

for (j=1; j <= 15; ++j) {

if (Board[i][j] == 1)

{ ++flag; }

else

{ flag = 0; }

if (flag == 5)

{ return 1; }}}

for (i=1; i <= 15; ++i) {

for (j=1; j <= 15; ++j) {

if (Board[j][i] == 1)

{ ++flag; }

else

{ flag = 0; }

if (flag == 5)

{ return 1; }} }

**2.右上-左下 右下-左上**

for (int a = 11; a >= 1; --a) {

i = a; j = 15;

while (i <= 15 && j >= 1) {

if (Board[i][j] == 1) { ++flag; }

else { flag = 0; }

++i; --j;//

if (flag == 5)

{ return 1; } }}

for (int b = 14; b >= 5; --b) {

i = 1;

j = b;

while (i <= 15 && j >= 1) {

if (Board[i][j] == 1) { ++flag; }

else { flag = 0; }

++i; --j;//

if (flag == 5)

{ return 1; } }}

for (int a = 11; a >= 1; --a) {

i = a;

j = 1;

while (i <= 15 && j <= 15) {

if (Board[i][j] == 1) { ++flag; }

else { flag = 0; }

++i; ++j;//

if (flag == 5)

{ return 1; }}}

for (int b = 2; b <= 11; ++b) {

i = 1;

j = b;

while (i <= 15 && j <= 15) {

if (Board[i][j] == 1) { ++flag; }

else { flag = 0; }

++i; ++j;

if (flag == 5) { return 1; } }}

****(7)alphabeta剪枝（递归实现）****

int analyse(int depth, int alpha, int beta, int Depth) {//alpha max beta -max

int tempX, tempY, temp;

if (ifWin() != 0) {

return evaluate();//如果模拟落子可以分出输赢，那么直接返回结果

}

else if (depth == 0)

{ //生成最佳的可能落子位置

seekPoints(2);

return best\_points.score[1];

//返回最佳位置对应的最高分

}

else if (depth % 2 == 0) {//max层ai决策

for (int i = 1; i <= 10; ++i) {//取是个最优点

seekPoints(2);

tempX = best\_points.x[i], tempY = best\_points.y[i];

Board[tempX][tempY] = 2;//模拟落子

temp = analyse(depth - 1, alpha, beta, Depth);

Board[tempX][tempY] = 0;//撤销落子

if (depth == Depth) {//记录好棋

decision.x[i] = tempX;

decision.y[i] = tempY;

decision.eval[i] = temp;

}

if (temp > alpha) {

alpha = temp;

}

if (temp<alpha&&depth!=Depth)

{ break; }//剪枝

}

return alpha;

}

else if (depth % 2 == 1) {//min层,敌方决策

for (int i = 1; i <= 10; ++i) {

seekPoints(1);

tempX = best\_points.x[i], tempY = best\_points.y[i];

Board[tempX][tempY] = 1;

temp = analyse(depth - 1, alpha, beta, Depth);

Board[tempX][tempY] = 0;

if (temp < beta) { beta = temp; }

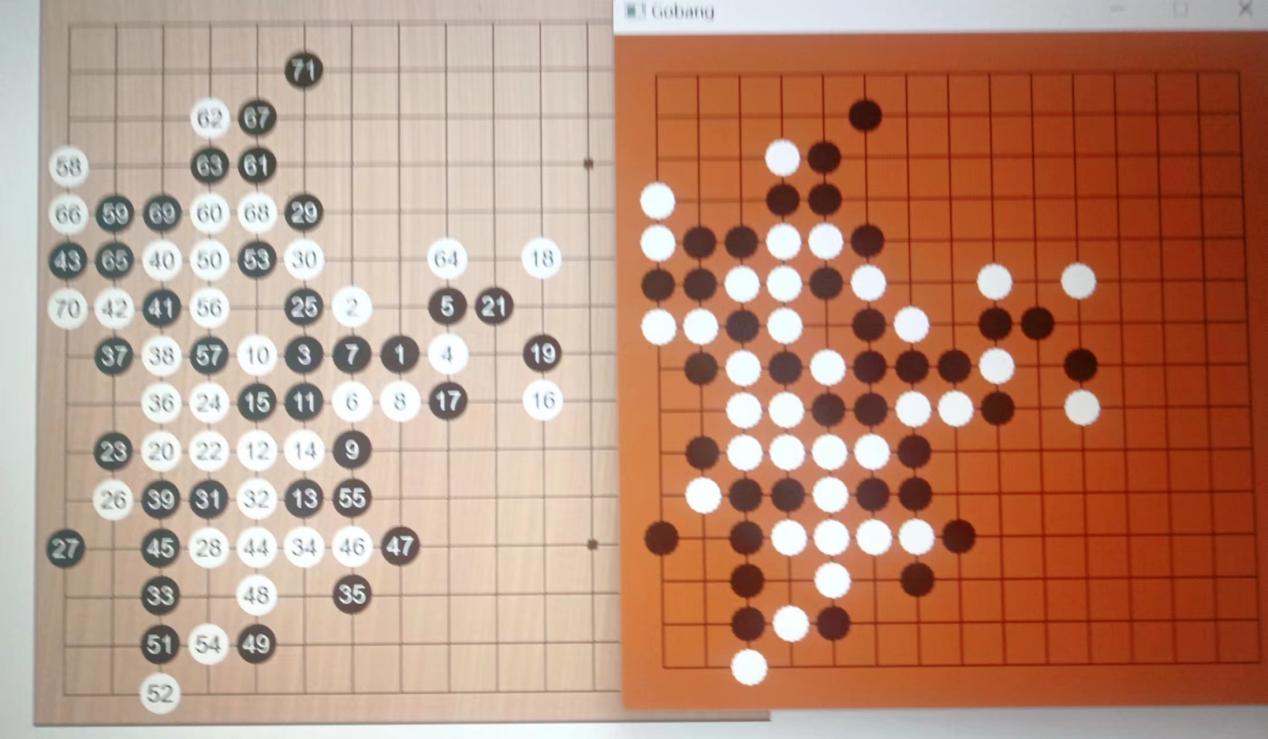
if (beta <temp)

{ break; }

} return beta;}}

# 实验数据及结果分析：

(1)程序运行界面



(2)程序运行速度:2秒内落子，无卡顿（4层）

(3)AI棋力评价：测试中打败了多位同学

**六：参考资料：**

[极小化极大值搜索 · Issue #13 · lihongxun945/myblog (github.com)](https://github.com/lihongxun945/myblog/issues/13)

[Alpha-Beta 剪枝 · Issue #14 · lihongxun945/myblog (github.com)](https://github.com/lihongxun945/myblog/issues/14)

**七：总结与心得体会：**

对多分支的递归有了更深入的理解，大体习得制作图形界面，基本掌握多文件编程

# 八、对本实验过程及方法、手段的改进建议及展望：

（1）动态改变递归层数

（2）增加随机下法

（3）算法可以继续改进：检索时跳过空白

（4）剪枝可以分为深入与返回多个函数

**报告评分：**

**指导教师签字：**

#ifndef \_GOBANG\_GUI\_H\_//GUI.h

#define \_GOBANG\_GUI\_H\_

#include<graphics.h>

#include<stdio.h>

void playChess(void);//鼠标下棋

#endif//GUI.h

#ifndef \_GOBANG\_AI\_H\_//AI.h

#define \_GOBANG\_AI\_H\_

#include<string.h>

#include<limits.h>

const int OTHER = 0;// 其他棋型

const int WIN = 1;

const int LOSE = 2;

const int FLEX4 = 3;

const int flex4 = 4;

const int BLOCK4 = 5;

const int block4 = 6;

const int FLEX3 = 7;

const int flex3 = 8;

const int BLOCK3 = 9;

const int block3 = 10;

const int FLEX2 = 11;

const int flex2 = 12;

const int BLOCK2 = 13;

const int block2 = 14;

const int FLEX1 = 15;

const int flex1 = 16;

//权重

const int weight[18] =

{ 0,100000000,-10000000,50000,-100000,400,-8000,400,-8000,20,-50,20,-50,1,-3,1,-3 };

typedef struct {//前十个最高分

int x[11];

int y[11];

int score[11];//此处落子的局势分数

}Points;

typedef struct {//位置

int x[11];

int y[11];

int eval[11];//分数评估

}Decision;

void init6type(void);

int evaluate(void);

void seekPoints(int blackORwhite);

int ifWin(void);

int analyse(int deep, int alpha, int beta, int Depth);

#endif//AI.h

#include"gobang\_gui.h"//main.c

#include"gobang\_AI.h"

int main() {

init6type();

playChess();

return 0;

}//main.c

#include"gobang\_gui.h"GUI.c

extern struct Decision {

int x[11];

int y[11];

int eval[11];

};

extern Decision decision;

extern int analyse(int depth, int alpha, int beta, int Depth);

extern int Board[17][17];

extern int ifWin(void);

void playChess(void)

{

int i, j;

initgraph(480, 480, SHOWCONSOLE);

setbkcolor(BROWN);

cleardevice();

setlinestyle(PS\_SOLID, 1);//直线宽度

setlinecolor(BLACK);

for (i = 30; i <= 480; i += 30) //横线

{

line(30, i, 450, i);

}

for (j = 30; j <= 480; j += 30) //竖线

{

line(j, 30, j, 450);

}

MOUSEMSG Mouse;

int flag;

while (1)

{

Mouse = GetMouseMsg();

flag = 0;

for (int i = 1; i <= 15; ++i)

{

for (int j = 1; j <= 15; ++j)

{

if (abs(Mouse.x - i \* 30) < 10 && abs(Mouse.y - j \* 30) < 10)

//将点击的范围内的值取整

{

Mouse.x = i \* 30;

Mouse.y = j \* 30;

flag = 1;

}

}

}

if (Mouse.uMsg == WM\_LBUTTONDOWN) //判断当前鼠标左键是否按下

{

if (!Board[Mouse.x / 30][Mouse.y / 30]&&flag==1) {

setfillcolor(BLACK);

solidcircle(Mouse.x, Mouse.y, 12);

Board[Mouse.x / 30][Mouse.y / 30] = 1;

int val = -INT\_MAX, X, Y;

analyse(4, -INT\_MAX, INT\_MAX, 4);

for (int i = 1; i <= 10; ++i) {

if (val < decision.eval[i]) {

X = decision.x[i];

Y = decision.y[i];

val = decision.eval[i];

}

}

Board[X][Y] = 2;

setfillcolor(WHITE);

solidcircle(X \* 30, Y \* 30, 12);

if (ifWin() != 0)

{

if (ifWin() == 1) { printf("you win\n"); }

else if (ifWin() == 2) { printf("you lose\n"); }

system("pause"); break;

}

}

}

}

}//GUI.c

#include"gobang\_AI.h"//AI.c

//flex4 needs 6 point

int type[4][4][4][4][4][4];//0 '\_',1 '0',2 'X',3 '#'

int Board[17][17] = { 0 };

Points best\_points;

Decision decision;

int sample[4][17];//统计4个方向上每种棋型的个数

int isEmpty[16][16];

int worth[16][16];

int ifWin(void) {

int flag = 0, i = 1, j = 1;

for (i=1; i <= 15; ++i) {

for (j=1; j <= 15; ++j) {

if (Board[i][j] == 1) { ++flag; }

else { flag = 0; }

if (flag == 5)

{ return 1; }

}

}

for (i=1; i <= 15; ++i) {

for (j=1; j <= 15; ++j) {

if (Board[i][j] == 2) { ++flag; }

else { flag = 0; }

if (flag == 5)

{ return 2; }

}

}

for (i=1; i <= 15; ++i) {

for (j=1; j <= 15; ++j) {

if (Board[j][i] == 1) { ++flag; }

else { flag = 0; }

if (flag == 5)

{ return 1; }

}

}

for (i=1; i <= 15; ++i) {

for (j=1; j <= 15; ++j) {

if (Board[j][i] == 2) { ++flag; }

else { flag = 0; }

if (flag == 5)

{ return 2; }

}

}

for (int a = 11; a >= 1; --a) {

i = a; j = 15;

while (i <= 15 && j >= 1) {

if (Board[i][j] == 1) { ++flag; }

else { flag = 0; }

++i; --j;//

if (flag == 5)

{ return 1; }

}

}

for (int b = 14; b >= 5; --b) {

i = 1;

j = b;

while (i <= 15 && j >= 1) {

if (Board[i][j] == 1) { ++flag; }

else { flag = 0; }

++i; --j;//

if (flag == 5)

{ return 1; }

}

}

for (int a = 11; a >= 1; --a) {

i = a; j = 15;

while (i <= 15 && j >= 1) {

if (Board[i][j] == 2) { ++flag; }

else { flag = 0; }

++i; --j;//

if (flag == 5)

{ return 2; }

}

}

for (int b = 14; b >= 5; --b) {

i = 1;

j = b;

while (i <= 15 && j >= 1) {

if (Board[i][j] == 2) { ++flag; }

else { flag = 0; }

++i; --j;//

if (flag == 5)

{ return 2; }

}

}

for (int a = 11; a >= 1; --a) {

i = a;

j = 1;

while (i <= 15 && j <= 15) {

if (Board[i][j] == 1) { ++flag; }

else { flag = 0; }

++i; ++j;//

if (flag == 5)

{ return 1; }

}

}

for (int b = 2; b <= 11; ++b) {

i = 1;

j = b;

while (i <= 15 && j <= 15) {

if (Board[i][j] == 1) { ++flag; }

else { flag = 0; }

++i; ++j;

if (flag == 5)

{ return 1; }

}

}

for (int a = 11; a >= 1; --a) {

i = a;

j = 1;

while (i <= 15 && j <= 15) {

if (Board[i][j] == 2) { ++flag; }

else { flag = 0; }

++i; ++j;//

if (flag == 5)

{ return 2; }

}

}

for (int b = 2; b <= 11; ++b) {

i = 1;

j = b;

while (i <= 15 && j <= 15) {

if (Board[i][j] == 2) { ++flag; }

else { flag = 0; }

++i; ++j;

if (flag == 5)

{ return 2; }

}

}

return 0;

}

void init6type(void) {

for (int i = 0; i <= 16; ++i) { Board[i][0] = 3; Board[i][16] = 3;}

for (int i = 1; i <= 15; ++i) { Board[0][i] = 3; Board[16][i] = 3;}

memset(type, 0, sizeof(type));

type[2][2][2][2][2][2] = WIN;

type[2][2][2][2][2][0] = WIN;

type[0][2][2][2][2][2] = WIN;

type[2][2][2][2][2][1] = WIN;

type[1][2][2][2][2][2] = WIN;

type[3][2][2][2][2][2] = WIN;

type[2][2][2][2][2][3] = WIN;

type[1][1][1][1][1][1] = LOSE;

type[1][1][1][1][1][0] = LOSE;

type[0][1][1][1][1][1] = LOSE;

type[1][1][1][1][1][2] = LOSE;

type[2][1][1][1][1][1] = LOSE;

type[3][1][1][1][1][1] = LOSE;

type[1][1][1][1][1][3] = LOSE;

type[0][2][2][2][2][0] = FLEX4;

type[0][1][1][1][1][0] = flex4;

type[0][2][2][2][0][0] = FLEX3;

type[0][0][2][2][2][0] = FLEX3;

type[0][2][0][2][2][0] = FLEX3;

type[0][2][2][0][2][0] = FLEX3;

type[0][1][1][1][0][0] = flex3;

type[0][0][1][1][1][0] = flex3;

type[0][1][0][1][1][0] = flex3;

type[0][1][1][0][1][0] = flex3;

type[0][2][2][0][0][0] = FLEX2;

type[0][2][0][2][0][0] = FLEX2;

type[0][2][0][0][2][0] = FLEX2;

type[0][0][2][2][0][0] = FLEX2;

type[0][0][2][0][2][0] = FLEX2;

type[0][0][0][2][2][0] = FLEX2;

type[0][1][1][0][0][0] = flex2;

type[0][1][0][1][0][0] = flex2;

type[0][1][0][0][1][0] = flex2;

type[0][0][1][1][0][0] = flex2;

type[0][0][1][0][1][0] = flex2;

type[0][0][0][1][1][0] = flex2;

type[0][2][0][0][0][0] = FLEX1;

type[0][0][2][0][0][0] = FLEX1;

type[0][0][0][2][0][0] = FLEX1;

type[0][0][0][0][2][0] = FLEX1;

type[0][1][0][0][0][0] = flex1;

type[0][0][1][0][0][0] = flex1;

type[0][0][0][1][0][0] = flex1;

type[0][0][0][0][1][0] = flex1;

int p1, p2, p3, p4, p5, p6, x, y, ix, iy;

//x:左5中'0'个数,y:左5中'X'个数,ix:右5中'0'个数,iy:右5中'X'个数

for (p1 = 0; p1 <= 3; ++p1) {

for (p2 = 0; p2 <= 2; ++p2) {

for (p3 = 0; p3 <= 2; ++p3) {

for (p4 = 0; p4 <= 2; ++p4) {

for (p5 = 0; p5 <= 2; ++p5) {

for (p6 = 0; p6 <= 3; ++p6) {

x = y = ix = iy = 0;

if (p1 == 1) { ++x; }

else if (p1 == 2) { ++y; }

if (p2 == 1) { ++x; ++ix; }

else if (p2 == 2) { ++y; ++iy; }

if (p3 == 1) { ++x; ++ix; }

else if (p3 == 2) { ++y; ++iy; }

if (p4 == 1) { ++x; ++ix; }

else if (p4 == 2) { ++y; ++iy; }

if (p5 == 1) { ++x; ++ix; }

else if (p5 == 2) { ++y; ++iy; }

if (p6 == 1) { ++ix; }

else if (p6 == 2) { ++iy; }

if ((p1 == 3 && p6 != 3) || (p1 != 3 && p6 == 3)) {//有边界

if (ix == 0 && iy == 4) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

{

type[p1][p2][p3][p4][p5][p6] = BLOCK4;

}

}

if (ix == 4 && iy == 0) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

{

type[p1][p2][p3][p4][p5][p6] = block4;

}

}

if (ix == 0 && iy == 3) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

{

type[p1][p2][p3][p4][p5][p6] = BLOCK3;

}

}

if (ix == 3 && iy == 0) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

{

type[p1][p2][p3][p4][p5][p6] = block3;

}

}

if (ix == 0 && iy == 2) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

{

type[p1][p2][p3][p4][p5][p6] = BLOCK2;

}

}

if (ix == 2 && iy == 0) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

{

type[p1][p2][p3][p4][p5][p6] = block2;

}

}

}

else if(p1!=3&&p6!=3) {//无边界

if ((x == 0 && y == 4) || (ix == 0 && iy == 4)) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

type[p1][p2][p3][p4][p5][p6] = BLOCK4;

}

if ((x == 4 && y == 0) || (ix == 4 && iy == 0)) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

type[p1][p2][p3][p4][p5][p6] = block4;

}

if ((x == 0 && y == 3) || (ix == 0 && iy == 3)) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

type[p1][p2][p3][p4][p5][p6] = BLOCK3;

}

if ((x == 3 && y == 0) || (ix == 3 && iy == 0)) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

type[p1][p2][p3][p4][p5][p6] = block3;

}

if ((x == 0 && y == 2) || (ix == 0 && iy == 2)) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

type[p1][p2][p3][p4][p5][p6] = BLOCK2;

}

if ((x == 2 && y == 0) || (ix == 2 && iy == 0)) {

if (type[p1][p2][p3][p4][p5][p6] == 0)

type[p1][p2][p3][p4][p5][p6] = block2;

}

}

}

}

}

}

}

}

}

int evaluate(void) {

//全局估值

memset(sample, 0, sizeof(sample));

int i, j, Type;

//判断竖向棋型

for (i = 1; i <= 15; ++i) {

for (j = 0; j <= 11; ++j) {

Type = type[Board[i][j]][Board[i][j + 1]][Board[i][j + 2]][Board[i][j + 3]][Board[i][j + 4]][Board[i][j + 5]];

++sample[0][Type];

}

}

//判断横向棋型

for (j = 1; j <= 15; ++j) {

for (i = 0; i <= 11; ++i) {

Type = type[Board[i][j]][Board[i + 1][j]][Board[i + 2][j]][Board[i + 3][j]][Board[i + 4][j]][Board[i + 5][j]];

++sample[1][Type];

}

}

//判断左上至右下棋型

for (i = 0; i <= 11; ++i) {

for (j = 16; j >= 5; --j) {

Type = type[Board[i][j]][Board[i + 1][j - 1]][Board[i + 2][j - 2]][Board[i + 3][j - 3]][Board[i + 4][j - 4]][Board[i + 5][j - 5]];

++sample[2][Type];

}

}

//判断右上至左下棋型

for (i = 0; i <= 11; ++i) {

for (j = 11; j >= 0; --j) {

Type = type[Board[i][j]][Board[i + 1][j + 1]][Board[i + 2][j + 2]][Board[i + 3][j + 3]][Board[i + 4][j + 4]][Board[i + 5][j + 5]];

++sample[3][Type];

}

}

int score = 0;

for (i = 1; i <= 16; ++i) {

score += (sample[0][i] + sample[1][i] + sample[2][i] + sample[3][i]) \* weight[i];//初步计分

}

return score;

}

void seekPoints(int blackORwhite) {

// 寻找落子点

memset(isEmpty, 0, sizeof(isEmpty));

memset(best\_points.x, 0, sizeof(best\_points.x));

memset(best\_points.y, 0, sizeof(best\_points.y));

memset(best\_points.score, 0, sizeof(best\_points.score));

int num = 3;

for (int i = 1; i <= 15; ++i) {

//每个非空点附近8个方向延伸3个深度,若不越界则标记为可走

for (int j = 1; j <= 15; ++j) {

if (Board[i][j] != 0) {

for (int k = -num; k <= num; ++k) {

if (i + k >= 1 && i + k <= 15) {

isEmpty[i + k][j] = 1;//非偏僻

if (j + k >= 1 && j + k <= 15) { isEmpty[i + k][j + k] = 1; }

if (j - k >= 1 && j - k <= 15) { isEmpty[i + k][j - k] = 1; }

}

if (j + k >= 1 && j + k <= 15) { isEmpty[i][j + k] = 1; }

}

}

}

}

for (int i = 1; i <= 15; ++i) {

for (int j = 1; j <= 15; ++j) {

if (blackORwhite == 2){worth[i][j] = -INT\_MAX;}

else if(blackORwhite==1) { worth[i][j] = INT\_MAX; }

if (Board[i][j] == 0&& isEmpty[i][j] == 1) {

Board[i][j] = blackORwhite;

worth[i][j] = evaluate();

Board[i][j] = 0;

}

}

}

int goal;

for (int k = 1; k <= 10; ++k) {

if(blackORwhite==2)

{ goal = -INT\_MAX; }

else if (blackORwhite == 1) {goal=INT\_MAX;}

for (int i = 1; i <= 15; ++i) {

for (int j = 1; j <= 15; ++j) {

if (blackORwhite == 2) {

if (worth[i][j] > goal) {

goal = worth[i][j];

best\_points.x[k] = i;

best\_points.y[k] = j;

}

}

else if(blackORwhite==1) {

if (worth[i][j] < goal) {

goal = worth[i][j];

best\_points.x[k] = i;

best\_points.y[k] = j;

}

}

}

}

best\_points.score[k] = goal;

if (blackORwhite == 2) {

worth[best\_points.x[k]][best\_points.y[k]] = -INT\_MAX;

}

else if(blackORwhite==1)

{ worth[best\_points.x[k]][best\_points.y[k]] = INT\_MAX; }

//清除掉上一点,计算下一点的位置和分数

}

}

int analyse(int depth, int alpha, int beta, int Depth) {//alpha max beta -max

int tempX, tempY, temp;

if (ifWin() != 0) {

return evaluate();//如果模拟落子可以分出输赢，那么直接返回结果

}

else if (depth == 0)

{ //生成最佳的可能落子位置

seekPoints(2);

return best\_points.score[1];

//返回最佳位置对应的最高分

}

else if (depth % 2 == 0) {//max层ai决策

for (int i = 1; i <= 10; ++i) {

seekPoints(2);

tempX = best\_points.x[i], tempY = best\_points.y[i];

Board[tempX][tempY] = 2;

temp = analyse(depth - 1, alpha, beta, Depth);

Board[tempX][tempY] = 0;

if (depth == Depth) {

decision.x[i] = tempX;

decision.y[i] = tempY;

decision.eval[i] = temp;

}

if (temp > alpha) {

alpha = temp;

}

if (temp<alpha&&depth!=Depth)

{ break; }

}

return alpha;

}

else if (depth % 2 == 1) {//min层,敌方决策

for (int i = 1; i <= 10; ++i) {

seekPoints(1);

tempX = best\_points.x[i], tempY = best\_points.y[i];

Board[tempX][tempY] = 1;

temp = analyse(depth - 1, alpha, beta, Depth);

Board[tempX][tempY] = 0;

if (temp < beta) { beta = temp; }

if (beta <temp)

{ break; }

}

return beta;}}//AI.c