极大极小

#include <stdio.h>

#include<conio.h>

#define COM -1

#define MAN 1

#define STEP 9

#define DRAW 0

#define ROW 3

#define COL 3

#define MAX\_NUM 1000;

struct Move

{

int x;

int y;

};

//棋盘

int board[3][3] = { { 0,0,0 },

{ 0,0,0 },

{ 0,0,0 }

};

int tempBoard[3][3] = { { 0,0,0 },

{ 0,0,0 },

{ 0,0,0 }

};

//玩家

int player = MAN;

//最好的一步

Move bestMove;

//当前深度

int currentDepth;

//谁先走

bool MAN\_first = true;

//判断输赢

int isWin() {

int i, j;

for (int i = 0; i < 3; i++)

{

if (board[i][0] + board[i][1] + board[i][2] == 3)

return 1;

else if (board[i][0] + board[i][1] + board[i][2] == -3)

return -1;

}

for (int j = 0; j < 3; j++)

{

if (board[0][j] + board[1][j] + board[2][j] == 3)

return 1;

else if (board[0][j] + board[1][j] + board[2][j] == -3)

return -1;

}

if (board[0][0] + board[1][1] + board[2][2] == 3 || board[0][2] + board[1][1] + board[2][0] == 3)

return 1;

else if (board[0][0] + board[1][1] + board[2][2] == -3 || board[0][2] + board[1][1] + board[2][0] == -3)

return -1;

else return 0;

}

//评估函数

int evaluteMap() {

bool flag = true;

int i, j;

if (isWin() == COM)

return MAX\_NUM;//如果计算机赢了，返回最大值

if (isWin() == MAN)

return -MAX\_NUM;//如果计算机输了，返回最小值

//for (i = 0; i < 3; i++)

// for (j = 0; j < 3; j++)

// if (board[i][j] == 0)

// {

// flag = false;

// break;

// }

//if (flag) //如果Flag为真 ， 棋盘都满了 就退出

// return 0;

int count = 0;//该变量用来表示评估函数的值

//将棋盘中的空格填满自己的棋子，既将棋盘数组中的0变为1

for (i = 0; i < 3; i++)

for (j = 0; j < 3; j++)

{

if (board[i][j] == 0)

tempBoard[i][j] = COM;

else

tempBoard[i][j] = board[i][j];

}

//电脑一方

//计算每一行中有多少行的棋子连成3个的

for (i = 0; i < 3; i++)

count += (tempBoard[i][0] + tempBoard[i][1] + tempBoard[i][2]) / 3;

for (i = 0; i < 3; i++)

count += (tempBoard[0][i] + tempBoard[1][i] + tempBoard[2][i]) / 3;

count += (tempBoard[0][0] + tempBoard[1][1] + tempBoard[2][2]) / 3;

count += (tempBoard[2][0] + tempBoard[1][1] + tempBoard[0][2]) / 3;

//将棋盘中的空格填满对方的棋子，既将棋盘数组中的0变为-1

for (i = 0; i < 3; i++)

for (j = 0; j < 3; j++)

{

if (board[i][j] == 0)

tempBoard[i][j] = MAN;

else tempBoard[i][j] = board[i][j];

}

//对方

//计算每一行中有多少行的棋子连成3个的

for (i = 0; i < 3; i++)

count += (tempBoard[i][0] + tempBoard[i][1] + tempBoard[i][2]) / 3;

for (i = 0; i < 3; i++)

count += (tempBoard[0][i] + tempBoard[1][i] + tempBoard[2][i]) / 3;

count += (tempBoard[0][0] + tempBoard[1][1] + tempBoard[2][2]) / 3;

count += (tempBoard[2][0] + tempBoard[1][1] + tempBoard[0][2]) / 3;

return count;

}

void makeMove(Move curMove)

{

board[curMove.x][curMove.y] = player;

player = (player == COM) ? MAN : COM;

}

void unMakeMove(Move curMove) {

board[curMove.x][curMove.y] = 0;

player = (player == COM) ? MAN : COM;

}

//得到有空位的集合

int getMoveList(Move moveList[]) {

int moveCount = 0;

int i, j;

for (i = 0; i < COL; i++)

{

for (j = 0; j < ROW; j++)

{

if (board[i][j] == 0)

{

moveList[moveCount].x = i;

moveList[moveCount].y = j;

moveCount++;

}

}

}

return moveCount; //返回一共多少个空的位置

}

int miniMaxsearch(int depth)

{

int value; //估值

int bestValue = 0; //最好的估值

int moveCount = 0;

int i; int m, n;

Move moveList[9];//保存可以下子的位置

if (isWin() == COM || isWin() == MAN)

{

return evaluteMap(); //一般是返回极大极小值

}

//如果搜索深度耗尽 ， 返回估值

if (depth == 0)

{

return evaluteMap();

}

//根据不同的玩家 进行赋值

if (COM == player) {

bestValue = -MAX\_NUM;

}

else if (MAN == player)

{

bestValue = MAX\_NUM;

}

//一共多少步

moveCount = getMoveList(moveList);

for (i = 0; i < moveCount; i++)

{

Move curMove = moveList[i];

makeMove(curMove);

value = miniMaxsearch(depth - 1);

unMakeMove(curMove);

if (player == COM)

{

if (value > bestValue)

{

bestValue = value;

if (depth == currentDepth)

{

bestMove = curMove;

}

}

}

else if (player == MAN)

{

if (value < bestValue)

{

bestValue = value;

if (depth == currentDepth)

{

bestMove = curMove;

}

}

}

}

return bestValue;

}

//打印棋盘 电脑X ，玩家O

void printBoard() {

int i, j;

for (i = 0; i < COL; i++)

{

printf("-------------\n");

for (j = 0; j < ROW; j++)

{

if (board[i][j] == COM)

{

printf("| X ");

}

else if (board[i][j] == MAN)

{

printf("| O ");

}

else

{

printf("| ");

}

}

printf("|\n");

}

printf("-------------\n");

}

void com\_play() {

miniMaxsearch(currentDepth);

board[bestMove.x][bestMove.y] = COM;

}

void man\_play() {

int x, y;

printf("请输入位置坐标 e.g, 0 0为左上角 2 2为右下角 \n");

scanf("%d", &x);

scanf("%d", &y);

while (x < 0 || x > 2 || y < 0 || y > 2)

{

printf("您输入的坐标错误，请重新输入:x:(0~2) , y:(0~2)\n");

scanf("%d", &x);

scanf("%d", &y);

}

while (board[x][y] != 0)

{

printf("该位置已有棋，请重新输入:\n");

scanf("%d", &x);

scanf("%d", &y);

}

board[x][y] = MAN;

}

void setFirst() {

char c;

printf("\nDo you want to play first? y -你先走 , n-电脑先走");

for (c = getche(); c != 'Y'&&c != 'y'&&c != 'N'&&c != 'n'; c = getche());

if (c == 'n' || c == 'N') {

MAN\_first = false;

}

printf("\n");

}

int main()

{

currentDepth = 9;

int step = 1;

setFirst();

printBoard();

if (MAN\_first) {

player = MAN;

for (step = 1; step <= STEP; )

{

man\_play();

printBoard();

if (player == isWin()) {

printf("您获胜了！！");

break;

}

step++;

currentDepth--;

if (step == 10) {

printf("平局 ~~~");

break;

}

player = (player == COM) ? MAN : COM;

com\_play();

printBoard();

if (player == isWin()) {

printf("很遗憾，电脑赢啦！！！");

break;

}

step++;

currentDepth--;

player = (player == COM) ? MAN : COM;

}

}

else

{

player = COM;

for (step = 1; step <= STEP; )

{

com\_play();

printBoard();

if (player == isWin()) {

printf("很遗憾，电脑赢啦！！！");

break;

}

step++;

currentDepth--;

if (step == 10) {

printf("平局 ");

break;

}

player = (player == COM) ? MAN : COM;

man\_play();

printBoard();

if (player == isWin()) {

printf("您获胜了！！");

break;

}

step++;

currentDepth--;

player = (player == COM) ? MAN : COM;

}

}

getch();

}

结果：

