#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// board information

#define BOARD\_SIZE 12

#define EMPTY 0

#define BLACK 1

#define WHITE 2

// bool

typedef int BOOL;

#define TRUE 1

#define FALSE 0

// option

typedef int OPTION;

#define UP 0

#define DOWN 1

#define LEFT 2

#define RIGHT 3

#define UP\_LEFT 4

#define UP\_RIGHT 5

#define DOWN\_LEFT 6

#define DOWN\_RIGHT 7

#define MAX\_BYTE 10000

#define START "START"

#define PLACE "PLACE"

#define TURN "TURN"

#define END "END"

//AlphaBeta参数

#define INTINITY 10000

int alpha=-INTINITY,beta=INTINITY;

char copboard1[BOARD\_SIZE][BOARD\_SIZE];

struct Command

{

int x;

int y;

OPTION option;

};

struct Command myboard;

char buffer[MAX\_BYTE] = { 0 };

char board[BOARD\_SIZE][BOARD\_SIZE] = { 0 };

int me\_flag;

int other\_flag;

const int DIR[8][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1}, {-1, -1}, {-1, 1}, {1, -1}, {1, 1} };

const int DI[8][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1}, {-1, -1}, {-1, 1}, {1, -1}, {1, 1} };

int record[20]={0},record1[20]={0},record2[20]={0};//记录每一个字的最佳走法 ,及其子的位置

int newx,newy;//方便值传递

int k;

int card[1000]={0};

void debug(const char \*str) {

printf("DEBUG %s\n", str);

fflush(stdout);

}

void printBoard() {

char visual\_board[BOARD\_SIZE][BOARD\_SIZE] = { 0 };

int i, j;

for (i = 0; i < BOARD\_SIZE; i++) {

for (j = 0; j < BOARD\_SIZE; j++) {

if (board[i][j] == EMPTY) {

visual\_board[i][j] = '.';

}

else if (board[i][j] == BLACK) {

visual\_board[i][j] = 'O';

}

else if (board[i][j] == WHITE) {

visual\_board[i][j] = 'X';

}

}

printf("%s\n", visual\_board[i]);

}

}

BOOL isInBound(int x, int y) {

return x >= 0 && x < BOARD\_SIZE && y >= 0 && y < BOARD\_SIZE;

}

/\*\*

\* YOUR CODE BEGIN

\* 浣犵殑浠ｇ爜寮€濮?

\*/

/\*\*

\* You can define your own struct and variable here

\* 浣犲彲浠ュ湪杩欓噷瀹氫箟浣犺嚜宸辩殑缁撴瀯浣撳拰鍙橀噺

\*/

/\*\*

\* 浣犲彲浠ュ湪杩欓噷鍒濆鍖栦綘鐨凙I

\*/

void initAI(int me) {

}

void MakeNextMove(int x,int y,int flag,int direction,char copboard[BOARD\_SIZE][BOARD\_SIZE])

{

int otherflag=3-flag;

newx = x + DI[direction][0];

newy = y + DI[direction][1];

copboard[x][y] = EMPTY;

copboard[newx][newy] = otherflag;

int intervention\_dir[4][2] = { {1, 0}, {0, 1}, {1, 1}, {1, -1} };

int i;

for (i = 0; i < 4; i++) {

int x1 = newx + intervention\_dir[i][0];

int y1 = newy + intervention\_dir[i][1];

int x2 = newx - intervention\_dir[i][0];

int y2 = newy - intervention\_dir[i][1];

if (isInBound(x1, y1) && isInBound(x2, y2) && copboard[x1][y1] == otherflag && copboard[x2][y2] == otherflag) {

copboard[x1][y1] = flag;

copboard[x2][y2] = flag;

}

}

// 澶?

int custodian\_dir[8][2] = { {1, 0}, {-1, 0}, {0, 1}, {0, -1}, {1, 1}, {1, -1}, {-1, 1}, {-1, -1} };

for (i = 0; i < 8; i++) {

int x1 = newx + custodian\_dir[i][0];

int y1 = newy + custodian\_dir[i][1];

int x2 = newx + custodian\_dir[i][0] \* 2;

int y2 = newy + custodian\_dir[i][1] \* 2;

if (isInBound(x1, y1) && isInBound(x2, y2) && copboard[x2][y2] == flag && copboard[x1][y1] == otherflag) {

copboard[x1][y1] = flag;

}

}

}

void UnmakeMove(int x,int y,int flag,char copboard[BOARD\_SIZE][BOARD\_SIZE],char copboard2[BOARD\_SIZE][BOARD\_SIZE])

{

for (x = 0; x < BOARD\_SIZE; x++) {

for (y = 0; y < BOARD\_SIZE; y++) {

copboard[x][y] = copboard2[x][y];

}

}

}

int Evaluate(int newx,int newy,int flag,int direction)

{

int x=newx,y=newy;

int count=0;

int number=0;

int otherflag=3-flag;

for(direction=0;direction<8;direction++)

{

if(board[x+2\*DI[direction][0]][y+2\*DI[direction][1]]=flag){count++;}

}

for(direction=0;direction<8;direction+2)

{

if((board[x+DI[direction][0]][y+DI[direction][1]]=otherflag)||(board[x+DI[direction+1][0]][y+DI[direction+1][1]]=otherflag))

number=number+2;

}

number=number>count?number:count;

return number;

}

//两者同体

int canmove(int i,int j,int flag,char copboard[BOARD\_SIZE][BOARD\_SIZE],int direction)

{

int otherflag=3-flag;

while(direction<8){

newx = i + DI[direction][0];

newy = j + DI[direction][1];

if ((copboard[newx][newy] == flag) || (copboard[newx][newy] == otherflag)){continue;}

else {return direction;}

return 0;

}

int AlphaBeta(int depth,int alpha,int beta,int x,int y,int flag)

{

int m,n;

int direction=0;

int val=0;

if (depth == 0) {

return Evaluate(newx,newy,flag,direction);

}

char copboard[BOARD\_SIZE][BOARD\_SIZE];

for (m = 0; m < BOARD\_SIZE; m++) {

for (n = 0; n < BOARD\_SIZE; n++) {

copboard[m][n] = board[m][n];

}

}

for(direction=0;canmove(y,u,flag,copboard,direction)>0;direction++) {

char copboard[BOARD\_SIZE][BOARD\_SIZE];

char copboard2[BOARD\_SIZE][BOARD\_SIZE];

for (m = 0; m < BOARD\_SIZE; m++) {

for (n = 0; n < BOARD\_SIZE; n++) {

copboard2[m][n] = board[m][n];

}

}

direction--;

MakeNextMove(x,y,flag,direction,copboard);

val = -AlphaBeta(depth - 1, -beta, -alpha,x,y,flag);

UnmakeMove(x,y,flag,copboard,copboard2);

if (val >= beta) {

return beta;

}

if (val > alpha) {

alpha = val;

if(depth==2){record[k]=direction;record1[k]=x;record2[k]=y;}

}

}

return alpha;

}

struct Command findValidPos(char board[BOARD\_SIZE][BOARD\_SIZE], int flag) {

struct Command myboard = { 0,0,0 };

int i = 0, j = 0;

int max,max1,max2;//价值最大的棋的最佳走法

int x, y,m=0;

int direction = 0;

int depth=2;

for (i = 0; i < 12; i++)

{

for (j = 0; j < 12; j++)

{

if (board[i][j] == flag)//找到我的棋。

{

x = i; y = j;

card[m]=AlphaBeta(depth,alpha,beta,x,y,flag);

m++;

}

}

}

for(k=0;k<20&&(!(record[k]==0&&record1[k]==0));k++){

max=max>record[k]?max:record[k];

max1=max1>record1[k]?max1:record1[k];

max2=max2>record2[k]?max2:record2[k];

}

myboard.x=max1;

myboard.y=max2;

myboard.option =max;

return myboard;

}

/\*\*

\* 杞埌浣犺惤瀛愩€?

\* 妫嬬洏涓?琛ㄧず绌虹櫧锛?琛ㄧず榛戞锛?琛ㄧず鐧芥棗

\* me琛ㄧず浣犳墍浠ｈ〃鐨勬瀛?1鎴?)

\* 浣犻渶瑕佽繑鍥炰竴涓粨鏋勪綋Command锛屽湪x灞炴€у拰y灞炴€у～涓婁綘鎯宠绉诲姩鐨勬瀛愮殑浣嶇疆锛宱ption濉笂浣犳兂瑕佺Щ鍔ㄧ殑鏂瑰悜銆?

\*/

struct Command aiTurn(const char board[BOARD\_SIZE][BOARD\_SIZE], int me) {

struct Command preferenPos;

int x, y;//stand for the position

for (x = 0; x < BOARD\_SIZE; x++) {

for (y = 0; y < BOARD\_SIZE; y++) {

copboard1[x][y] = board[x][y];

}

}

preferenPos = findValidPos(copboard1, me);

return preferenPos;

}

/\*\*

\* 浣犵殑浠ｇ爜缁撴潫

\*/

BOOL place(int x, int y, OPTION option, int cur\_flag) {

// 绉诲姩涔嬪墠鐨勪綅缃病鏈夋垜鏂规瀛?

if (board[x][y] != cur\_flag) {

return FALSE;

}

int new\_x = x + DIR[option][0];

int new\_y = y + DIR[option][1];

// 绉诲姩涔嬪悗鐨勪綅缃秴鍑鸿竟鐣? 鎴栬€呬笉鏄┖鍦?

if (!isInBound(new\_x, new\_y) || board[new\_x][new\_y] != EMPTY) {

return FALSE;

}

board[x][y] = EMPTY;

board[new\_x][new\_y] = cur\_flag;

int cur\_other\_flag = 3 - cur\_flag;

// 鎸?

int intervention\_dir[4][2] = { {1, 0}, {0, 1}, {1, 1}, {1, -1} };

int i;

for (i = 0; i < 4; i++) {

int x1 = new\_x + intervention\_dir[i][0];

int y1 = new\_y + intervention\_dir[i][1];

int x2 = new\_x - intervention\_dir[i][0];

int y2 = new\_y - intervention\_dir[i][1];

if (isInBound(x1, y1) && isInBound(x2, y2) && board[x1][y1] == cur\_other\_flag && board[x2][y2] == cur\_other\_flag) {

board[x1][y1] = cur\_flag;

board[x2][y2] = cur\_flag;

}

}

// 澶?

int custodian\_dir[8][2] = { {1, 0}, {-1, 0}, {0, 1}, {0, -1}, {1, 1}, {1, -1}, {-1, 1}, {-1, -1} };

for (i = 0; i < 8; i++) {

int x1 = new\_x + custodian\_dir[i][0];

int y1 = new\_y + custodian\_dir[i][1];

int x2 = new\_x + custodian\_dir[i][0] \* 2;

int y2 = new\_y + custodian\_dir[i][1] \* 2;

if (isInBound(x1, y1) && isInBound(x2, y2) && board[x2][y2] == cur\_flag && board[x1][y1] == cur\_other\_flag) {

board[x1][y1] = cur\_flag;

}

}

return TRUE;

}

//........

//........

//..XXXX..

//XXXX....

//....OOOO

//..OOOO..

//........

//........

void start(int flag) {

memset(board, 0, sizeof(board));

int i;

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

board[2][2 + i] = WHITE;

board[6][6 + i] = WHITE;

board[5][3 + i] = BLACK;

board[9][7 + i] = BLACK;

}

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

board[8 + i][2] = WHITE;

board[2 + i][9] = BLACK;

}

initAI(flag);

}

void turn() {

// AI

struct Command command = aiTurn((const char(\*)[BOARD\_SIZE])board, me\_flag);

place(command.x, command.y, command.option, me\_flag);

printf("%d %d %d\n", command.x, command.y, command.option);

fflush(stdout);

}

void end(int x) {

}

void loop() {

// freopen("../input", "r", stdin);

while (TRUE)

{

memset(buffer, 0, sizeof(buffer));

gets(buffer);

if (strstr(buffer, START))

{

char tmp[MAX\_BYTE] = { 0 };

sscanf(buffer, "%s %d", tmp, &me\_flag);

other\_flag = 3 - me\_flag;

start(me\_flag);

printf("OK\n");

fflush(stdout);

}

else if (strstr(buffer, PLACE))

{

char tmp[MAX\_BYTE] = { 0 };

int x, y;

OPTION option;

sscanf(buffer, "%s %d %d %d", tmp, &x, &y, &option);

place(x, y, option, other\_flag);

}

else if (strstr(buffer, TURN))

{

turn();

}

else if (strstr(buffer, END))

{

char tmp[MAX\_BYTE] = { 0 };

int x;

sscanf(buffer, "%s %d", tmp, &x);

end(x);

}

// printBoard();

}

}

int main(int argc, char \*argv[]) {

loop();

return 0;

}