**五子棋AI实验报告**

2023080909018 于弋涵

1. 设计要求

执后手白子的五子棋AI，并由玩家执先手黑子，无禁手。程序具有五子棋游戏的基本功能，无重开棋局功能。

1. 主要算法
2. 极小化极大值搜索（Minimax算法）
3. Alpha\_beta剪枝算法
4. 设计思路

1.初始化：

首先将棋盘画出，接着画出棋盘的图片，即将背景的颜色处理为棋盘的颜色，并在上面画出黑色的线。

同时设置了棋盘的尺寸，线与线之间的间距，棋子的大小等。

还对数据进行了处理。

2.胜负判定

void Gobang::isGameOver(int x, int y)

{

int offset[4] = { 1,size - 1,size,size + 1 };

int checkValue = currentState ? -1 : 1;

int cnt;

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

int offsetX[4] = { 0,1,1,1 }, offsetY[4] = { 1 , 0 ,1,-1 };

cnt = 1;

//检查左边的连子数

int k = 4;

int x1 = x, y1 = y;

while (k--){

x1 -= offsetX[i];

y1 -= offsetY[i];

if (x1 > -1 && x1 < size && y1 > -1 && y1 < size && board[x1][y1] == checkValue) ++cnt;

else break;

}

k = 4;

x1 = x;

y1 = y;

//检查右边的连子数

while (k--){

x1 += offsetX[i];

y1 += offsetY[i];

if (x1 > -1 && x1 < size && y1 > -1 && y1 < size && board[x1][y1] == checkValue) ++cnt;

else break;

}

if (cnt >= 5){

gameOver = currentState ? 2 : 1;

break;

}

else if (cnt <5 && piecesCnt == size\*size){

gameOver = 3;

break;

}

}

}

3.棋型判定

七种基本棋型：连五、活四、冲四、活三、眠三、活二、眠二

活四：有两个连五点（即有两个点可以形成五）。活四出现的时候，如果对方单纯过来防守的话，是已经无法阻止自己连五了。

冲四：有一个连五点。相对比活四来说，冲四的威胁性就小了很多，因为这个时候，对方只要跟着防守在那个唯一的连五点上，冲四就没法形成连五。

活三：可以形成活四的三颗棋子，有两种最基本的活三棋型。

活三棋型是进攻中最常见的一种，活三之后，如果对方不以理会，将可以下一手将活三变成活四，而活四是已经无法单纯防守住的。所以，当我们面对活三的时候，需要非常谨慎对待。在自己没有更好的进攻手段的情况下，需要对其进行防守，以防止其形成活四棋型。

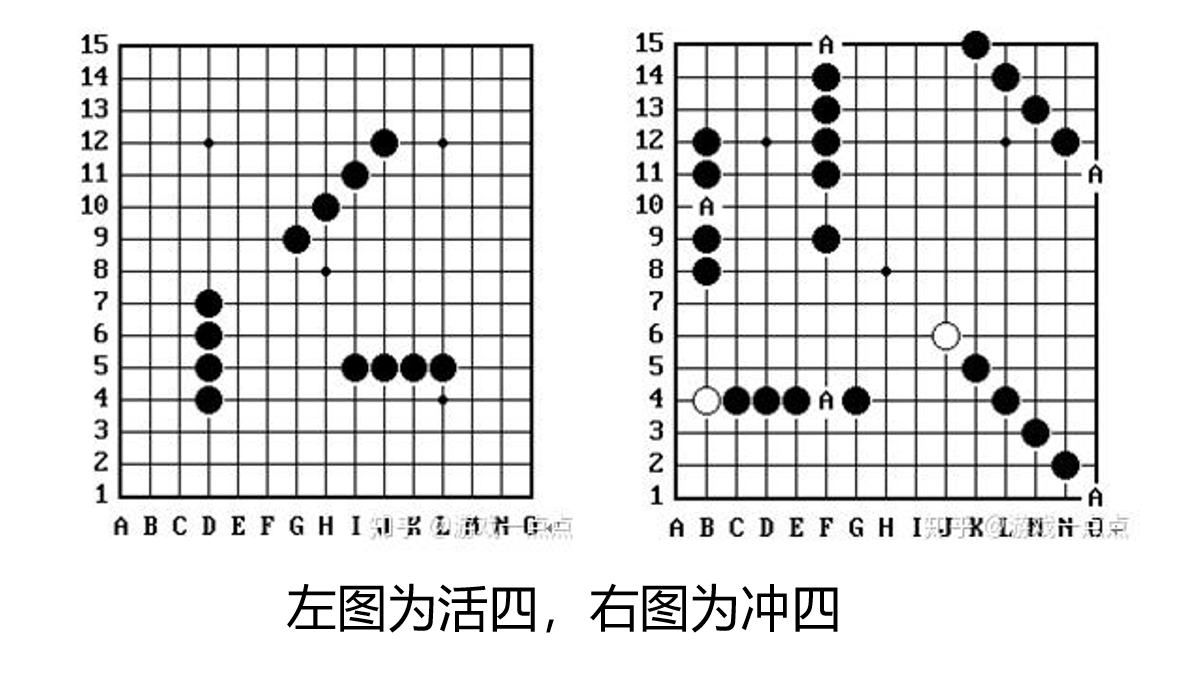
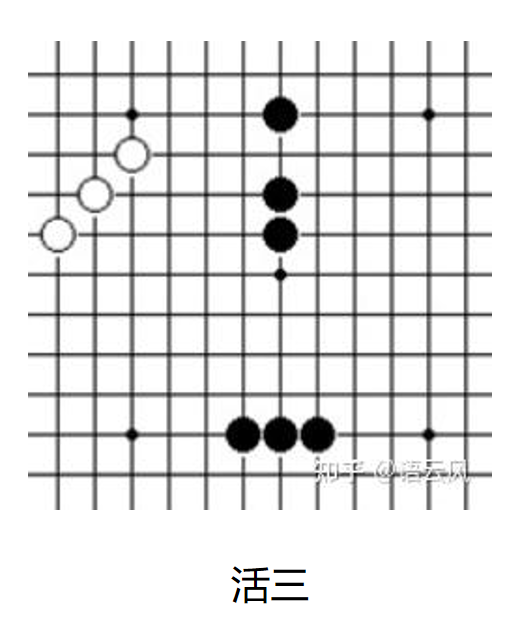
眠三：只能够形成冲四的三，共有六种最基础的眠三棋形。眠三的棋型与活三的棋型相比，危险系数下降不少，即使不去防守，下一手它也只能形成冲四，单纯的冲四棋型，是可以轻易防守住的。

眠三的棋型是很丰富的。眠三只能形成冲四，而活三既可以形成眠三，又能够形成活四。因为活四棋型比冲四棋型具有更大的优势，所以，我们在既能够形成活四又能够形成冲四时，一定要选择形成活四。

判断是眠三还是活三非常关键，而且判定起来十分复杂。

活二：能够形成活三的二。活二棋型看起来似乎很无害，但其实是非常重要的，尤其是在开局阶段，如果我们形成较多的活二棋型的话，当我们将活二变成活三时，才能够令自己的活三不停出现，让对手无暇进攻，只能一味防守。

眠二：能够形成眠三的二。危险系数不太大，相对而言形成活二才是获胜的最佳策略。



这些基本棋型存储在字符串数组里，需要判断时直接调用提前写好的棋型来判断。

string pattern[39] = {

"aaaaa",///0:连五

"oaaaao",///1:活四

"baaaao","oaaaab",///2~3眠四

"aoaaab","baaaoa","aaoaa","boaaaoa","aoaaaob",///4~8低级死四

"oaaaoo","ooaaao",///9~10活三

"oaaoao","oaoaao",///11~12跳活三 "baaaoo","ooaaab","baaoao","oaoaab","oaaoab","baoaao","aooaa","aaooa","aoaoa","boaaaob",///13~22眠三

"ooaaoo","oooaao","oaaooo",///23~25活二

"ooaoao","oaoaoo","oaooao",///26~28低级活二 "aaooo","oooaa","baoaoo","ooaoab","oaooab","baooao","aoooa","boaoaob","boaaoob","booaaob"///29~38眠二

};

3.评估分数

（1）打分规则

1.我们对整个棋盘形势进行打分，存在两个二维数组上，其中一个为我方的形势分数，而另一个为敌方的形势分数。

2.然后找出我方形势分数的最大值及其对应的位置，找出敌方形势的最大值及其对应的位置。

3.如何判断是进攻还是防守：

如果我方分数最大值>=对方最大值，则进攻，下我方形势最大值对应的位置；如果有多个我方分数均为最大值，则下这几个对应位置上对方分数最大的位置。否则防守，下敌方分数最大值对应的位置。如果有多个敌方分数均为最大值，则下这几个对应位置上我方分数最大的位置。

（2）打分方法

在棋盘空位置上暂时添加要判断放的棋子。

取出以空位置为中心的4个方向（上，下，左，右），每个方向以该位置为中心两边各取4个格子信息。

四个方向都判断其棋型，是否连五，活四，冲四，活三，眠三，活二，眠二等中的一种。

最后综合四个方向的棋型，对该位置进行打分。

综合四个方向后判断：

是否能成5,如果是我方的话给予100000分，如果是敌方的话给予100000 分；

是否能成活4或者是双死4或者是死4活3，如果是我方的话给予10000分，如果是敌方的话给予10000分；

是否已成双活3，如果是我方的话给予5000分，如果是敌方的话给予5000 分；

是否成死3活3（高级），如果是我方的话给予1000分，如果是敌方的话给予1000 分；

是否能成死4，如果是我方的话给予500分，如果是敌方的话给予500分；

是否能成低级死4，如果是我方的话给予400分，如果是敌方的话给予400分；

是否能成单活3，如果是我方的话给予100分，如果是敌方的话给予100分；

是否能成跳活3，如果是我方的话给予90分，如果是敌方的话给予90分；

是否能成双活2，如果是我方的话给予50分，如果是敌方的话给予50分；

是否能成活2，如果是我方的话给予10分，如果是敌方的话给予10分；

是否能成低级活2，如果是我方的话给予9分，如果是敌方的话给予9分；

是否能成死3，如果是我方的话给予5分，如果是敌方的话给予5分；

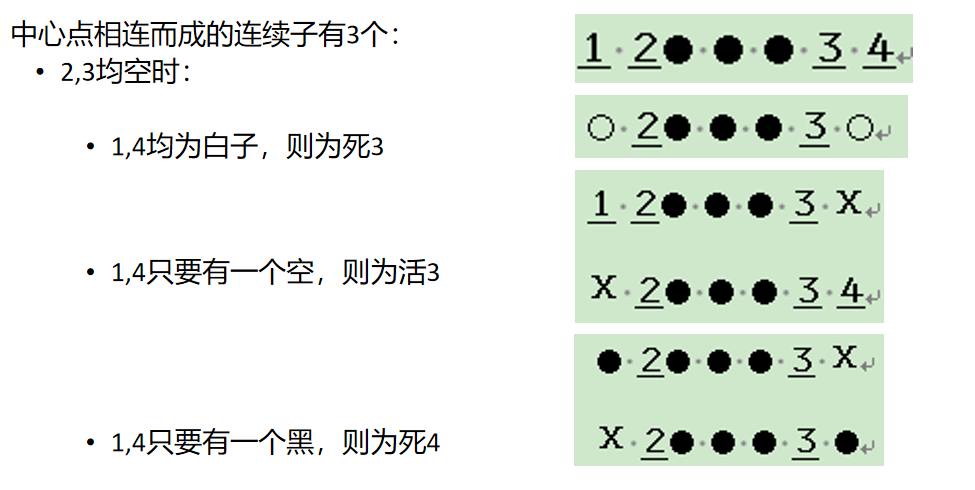
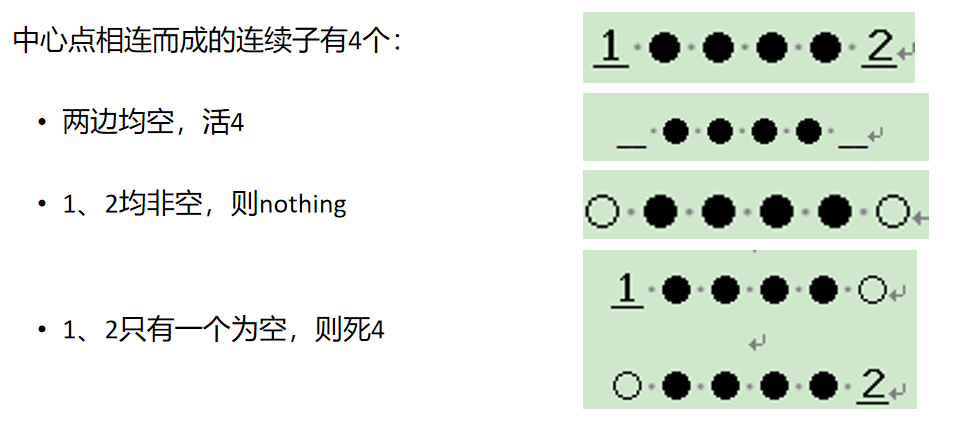
是否能成死2，如果是我方的话给予2分，如果是敌方的话给予2分。

是否其他情况，如果是我方的话给予1分，如果是敌方的话给予1分。

有棋子，则直接0分。

分数可提前存入score数组中。

1. 棋型判断



1. 剪枝算法

int Gobang::negaMax(int depth, int &x, int &y, int alpha, int beta)

{

int best = -99999999, val;

//如果对方已连五，就直接返回

isGameOver(steps[piecesCnt - 1].x, steps[piecesCnt - 1].y);

if (gameOver)

return -1000000;

//如果之前计算出这个局面的值就直接返回

val = searchHashTable(depth);

if (val != unknown){

++hit;

return val;

}

if (depth <= 0)

return evaluate();

point \* move;

move = new point[40];

int moveCnt;

//动态生成可行点,根据局面判断要考虑的节点数

if (piecesCnt<6)

moveCnt = generateBestMoves(8, move);

else if (piecesCnt<12)

moveCnt = generateBestMoves(10, move);

else

moveCnt = generateBestMoves(searchLimit, move);

int x1, y1;

for (int i = 0; i<moveCnt; ++i){

placePiece(move[i].x, move[i].y);

x1 = move[i].x;

y1 = move[i].y;

val = -negaMax(depth - 1, x1, y1, -beta, -alpha);

removePiece(move[i].x, move[i].y);

if (val >= beta) {

writeToHashTable(beta, depth);

return beta;

}

if (val > best) {

best = val;

if (depth == searchDepth){

x = x1;

y = y1;

}

if (val > alpha) alpha = val;

}

}

delete[] move;

writeToHashTable(best, depth);

return best;

}

1. 成果展示

