game AI design of Gomoku

Design and analysis of algorithms

陈诚 09016429

徐真杰 09016430

Version 1.0

最初的算法思路

假定五子棋棋盘大小为 15*15，可以视作一个 15*15 的二维数组。AI 算法的核心思想是：**遍历棋盘上的每一个点，给每个点根据棋局当前形势评估一个分数（即走这一步的优先级），最后 AI 选择分数最高，即优先级最高的点落子。**

在开局时，如果玩家先落子，那么 AI 在落子的任意一个方向随机落子；如果 AI 先手，那么就规定落在天元的位置上。

遍历前需要初始化一个分数数组，让它们全部为 0。遍历过程中每一个点本身有三种状态：空白，黑子，白子。如果有子那么分数为 0（优先级最低，即不能走这一步）。如果该处空白，那么向四周八个方向（上、下、左、右、左上、右下、左下、右上）进行搜索，用一个数组记录该方向的落子情况（判断遇到边界则修改）并判断棋局类型，随后根据形势评估分数，最后对分数数组搜索最大值，最大值坐标即为落子位置。

部分源代码：

```
int judgeType(const int chess[9]);//判断当前方向棋局类型
void getChess(int chess[9], const int state[15][15], Position position, int color, int direction);//获得当前方向棋局数组 chess
int getType(const int state[15][15], Position position, int color, int direction);//获得当前方向棋局类型
int judgeChessSituation(const int state[15][15], Position position, int color);//综合 4 个方向评判当前位置棋局形势的分数
int giveScore(Situation situation);//根据形势，给分
Position maxScore(const int myscore[15][15], const int hisscore[15][15]);//根据我的分数和对手的分数，选取最大利益的位置
Position getPosition(const int chesspadstate[15][15], int color);//根据当前形势，计算下步棋的位置
```

ArtificialIntelligence.h

```
Position ArtificialIntelligence::getPosition(const int chesspadstate[15][15], int color) {//计算下一步棋子的位置
//旗手和棋子的标志 + 1 = 棋盘的标志
int mychesspadcolor = color + 1;//棋盘标志
int hischesspadcolor;

int myscore[15][15] = { 0 };//我的分数
```

```
    int hisscore[15][15] = { 0 };//对手的分数
    int tempstate[15][15] = { 0 };//临时标志

    //判断是否第一次下棋
    int flag = 0;
    int k = 0, h = 0;
    for (k = 0;k < 15;k++) {
        for (h = 0;h < 15;h++) {
            if (chesspadstate[k][h]>0) {
                flag = 1;
                break;
            }
        }
        if (flag)
            break;
    }
    if (k ==15 && h ==15) {//第一次下棋
        Position position = { 7,7 };//默认最中间
        return position;
    }

    //把最后一步标志还原
    for (int i = 0;i < 15;i++) {
        for (int j = 0;j < 15;j++) {
            if (chesspadstate[i][j]>2)
                tempstate[i][j] = chesspadstate[i][j] - 2;
            else
                tempstate[i][j] = chesspadstate[i][j];
        }
    }


    //打分
    for (int i = 0;i < 15;i++)
        for (int j = 0;j < 15;j++) {
            Position position;
            int score;

            position.x = i;
            position.y = j;
            //我的分数
            score = judgeChessSituation(tempstate, position, mychesspadcolor);//返回当
前形势分数
            myscore[i][j] = score;
```

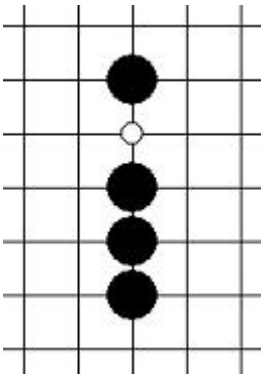
```
            if (mychesspadcolor == BLACKFLAG)
                hischesspadcolor = WHITEFLAG;
            else
                hischesspadcolor = BLACKFLAG;
            //对手分数
            score = judgeChessSituation(tempstate, position, hischesspadcolor);//返回当
前形势分数
            hisscore[i][j] = score;
        }

    //根据分数，给出位置
    return maxScore(myscore, hisscore);

    }
```

ArtificialIntelligence.cpp

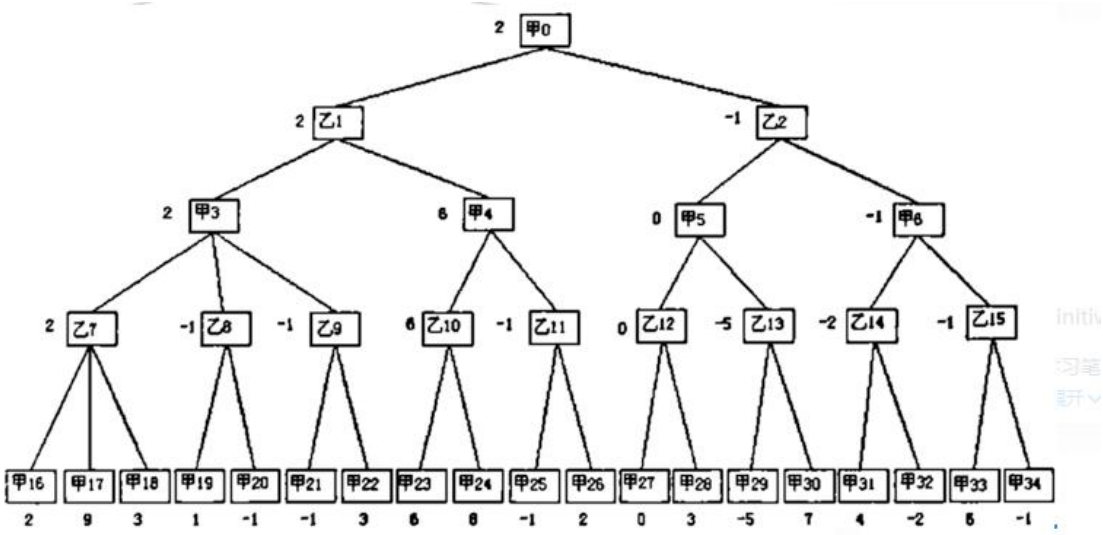
算法评估

我们的算法实际上就是对可选位置的一个评估。在对战中，每一次落子都会使得分数数组得到一些变化，每一次都会导致 AI 判断的变化。在这个基础上，每一次落子还要进行一次对自己本身棋子颜色的一个遍历，判断自己的情况，同时加分加在分数数组之中，这样一来，电脑就会根据自己的棋子的情况以及玩家的落子情况进行判断，哪一个地方更加适合落子。**在我们的算法中，AI 的棋力强弱是通过不同棋局类型分数的划定来决定的。**

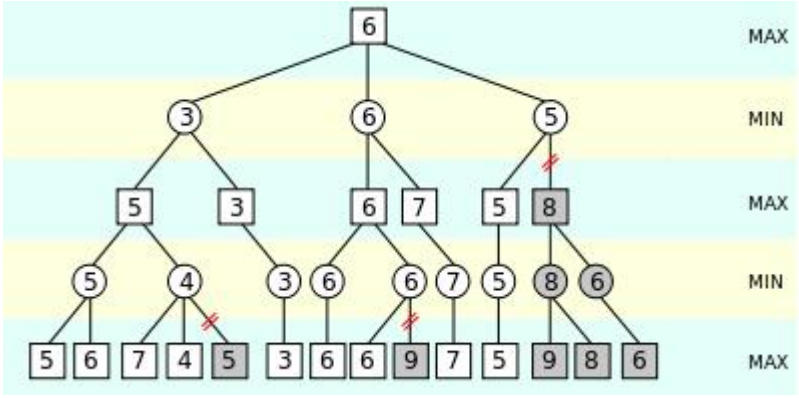
在讨论出最初的简单算法思路后，我们发现了这样几个问题：1.向八个方向不断搜索直到边界的做法没有必要，会影响到 AI 计算效率；2.棋局类型如何准确判断，以及不同类型的优先级分数如何准确划分；3.搜索和判断能否同时进行，如果只沿八个方向中的某个方向进行判断，显然会遗漏某些棋局类型，例如下图中冲四的空白点。



同时，我们也察觉到整个 AI 缺乏一定的远见性，算法仍然只是非常浅显的。通过查阅资料我们了解到与五子棋 AI 相关的一些算法例如极大极小值搜索、Alpha-Beta 剪枝算法、启发式搜索函数等，我们考虑在有余力的情况下尽可能地学习这些算法并应用到我们的 AI 中。



极大极小值搜索算法

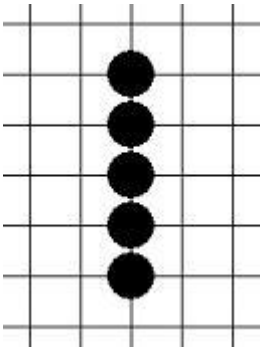


Alpha-Beta 剪枝算法

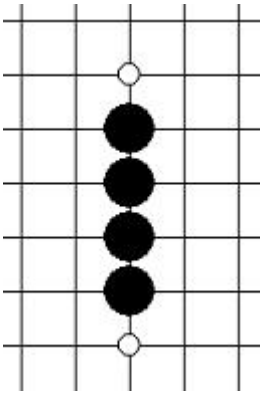
算法改进想法

1. 针对第一个问题，我们的想法是只向一个方向搜索固定几位即可，例如四到五位。这是因为我们认为超过四位的落子形成的棋局类型与该空白点关系并不是很大。我们也认为这样做能在尽可能不降低 AI 棋力的前提下提升算法的效率。

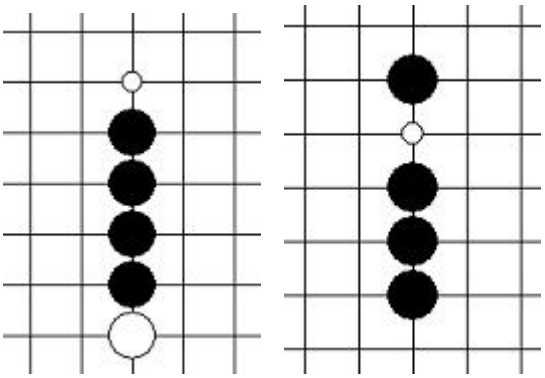
2. 关于棋局的类型，我们查阅了一些资料以及其他五子棋 AI 算法，发现棋局类型大致可以分为 5 类：5 连、4 连、3 连、2 连、1 连；其中又可以分成三种情况：活（即连子的两端都空白）、冲（即连子一端被堵死或四个连子中有一个空白点）、死（连子两端被堵死）。下图给出了几个例子。



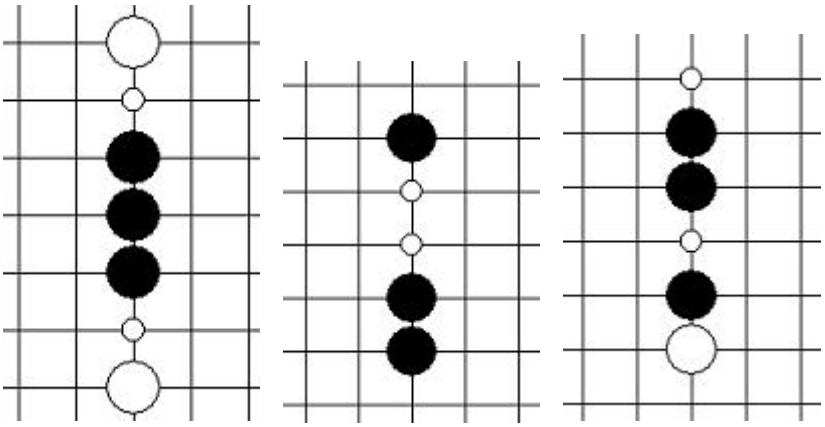
连五，判断胜负



活四，有两个连五点



冲四，有一个连五点



跳三，可以形成冲四

棋局类型判断以及打分牵扯到许多五子棋专业技巧，我们通过查阅相关资料、阅读其他五子棋 AI 算法源码，最终得出了如下的棋局类型以及打分等级：

```
//棋局类型
const int WIN5;//5 连珠
const int ALIVE4;//活 4
const int DIE4;//死 4
const int LOWDIE4;//死 4 的低级版本
const int ALIVE3;//活 3
const int JUMP3;//跳 3
const int DIE3;//死 3
const int ALIVE2;//活 2
const int LOWALIVE2;//低级活 2
const int DIE2;//死 2
const int NOTHREAT;//没有威胁

//打分等级
const int LevelOne = 100000;//成五
const int Leveltwo = 10000;//成活 4 或 双死 4 或 死 4 活 3
const int Levelthree = 5000;//双活 3
```

```
const int Levelfour = 1000;//死 3 高级活 3
const int Levelfive = 500;//死四
const int Levelsix = 400;//低级死四
const int Levelseven = 100;//单活 3
const int LevelEight = 90;//跳活 3
const int LevelNight = 50;//双活 2
const int LevelTen = 10;//活 2
const int LevelEleven = 9;//低级活 2
const int LevelTwelve = 5;//死 3
const int LevelThirteen = 2;//死 2
const int LevelFourteen = 1;//没有威胁
const int LevelFiveteen = 0;//不能下
```

以上是我们罗列出的棋局类型。有些棋局类型非常好判断，然而也有些棋局类型我们目前仍感觉难以找到合适的方法来辨别，目前只是单纯地将这种棋局类型列举上去。在这一点上我们需要进一步地去思考。

3. 经过小组讨论，我们认为搜索和判断应该分成两个成员函数实现，这虽然牺牲了一部分效率，但是我们认为能够提高 AI 的棋力。这是考虑到如果要全面准确的判断棋局类型，需要把八个方向整合成四条线，即横、竖、从左上到右下、从左下到右上。

4. 我们目前的算法仍然非常浅显，需要更进一步的优化。这也需要我们利用课余时间尽可能地查阅网上关于 AI 算法的博客、阅读更多的代码来提升自己的水平。

总结

在这之前，我们组并没有接触过任何与 AI 相关的东西，在讨论整个 AI 算法的过程中，我们的思路很多也是来源于网上的博客以及相关的源码，而在这期间我们接触到了许多算法，学到了不少知识。但是如何学以致用是我们接下来这一阶段要考虑的问题。同时，我们的 AI 虽然有了初步的设计思路，但是还没有真正成型，这是我们需要补上的进度。我们接

下来要做的主要是根据以上的改进想法，从网上查阅相关资料，继续完善我们的算法，希望不久之后能做出不错的成品。