Game AI Design of Gomoku

Design And Analysis Of Algorithms

姓名：沈彦昊

学号：09016427

Week 3

1th report

项目介绍：

该项目需要设计一个棋盘大小为 15*15 的五子棋 AI 算法。

无禁手等附加条件限制。

出现一方在横纵左斜右斜四方向有五子连珠即为胜利。

设计思路：

1.棋盘**格式格式**为 15*15 char 型二维数组

玩家为‘X’；AI 为‘O’用以显示当前棋盘情况

2.将五子棋每一步的走法展开，可以组成一颗**博弈树**。

在博弈树中，从根节点为 0 开始，所有可能出现的走法即为下一层元素。

五子棋 AI 棋力的强弱与**预测走法的深度**有关。

3.**博弈树中要选择最优走法**，即使自己收益最高，使对手收益最低。

我们假设得分越高，对 AI 越有利。因此在计算在 AI_turn 时候要求选择得分最高分支，在 Player_turn 时候认为玩家会选择得分最低下法，选择得分最低分支。

例如该算法可以预测未来三招的落子，需要对博弈树的三层分支进行有效判断，得出对 AI 最有利/使玩家收益最下选择。

4.对博弈树中每层元素的得分比较需要有高效的**评估函数。**此处引入五子棋术语，如活四，死四，活三等。不同的连珠情况对应不同的权值，对落子点进行充分权值评估后，可以找到一个 Max_Value 的点，说明该点可带来的效益最高，优先级最高。

参阅资料：

1. https://baike.baidu.com/item/%E4%BA%94%E5%AD%90%E6%A3%8B%E6%9C%AF%E8%AF%AD/11009079?fr=Aladdin

〖死四〗不能成五的四连。

〖三〗可以形成四再形成五的三枚同色棋子组成的棋型。

〖活三〗再走一着可以形成活四的三。

〖连活三〗两端都是威胁的活三。简称"连三"。

〖跳活三〗中间夹有一个威胁的活三。简称"跳三"。

〖眠三〗再走一着可以形成冲四的三。

〖死三〗不能成五的三。

〖二〗可以形成三、四直至五的两枚同色棋子组成的棋型。

〖活二〗再走一着可以形成活三的二。

〖连活二〗连的活二。简称"连二"。

〖五连〗五枚同色棋子在一条线上邻接连串。

〖长连〗五枚以上同色棋子在一条线上邻接成串。

〖成五〗五连和长连的统称。

〖威胁〗下一手可以成五或者活四的点。

〖四〗五连去掉1子的棋型。

〖活四〗有两个威胁的四。

〖冲四〗只有一个威胁的四。

了解五子棋的术语，对评估函数中权值判断有着重要作用。

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）。

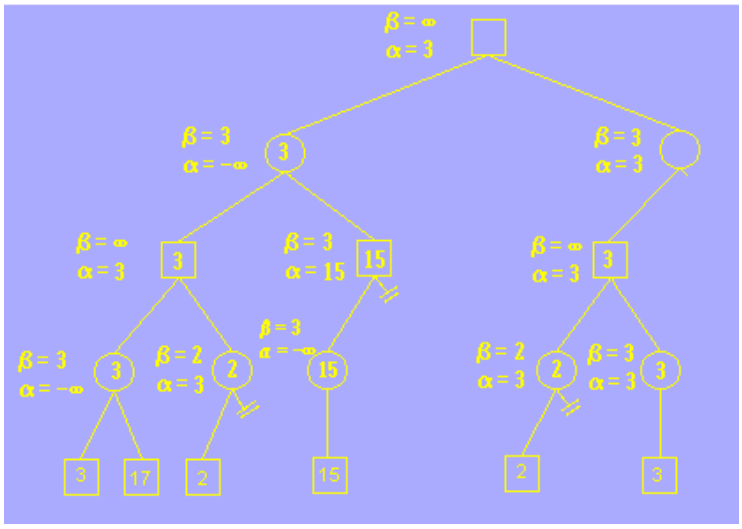
2．http://blog.csdn.net/baixiaozhe/article/details/51872495

http://blog.csdn.net/u013351484/article/details/50789521

Alpha-Beta 剪枝算法(Alpha Beta Pruning)

Alpha-Beta 剪枝用于裁剪搜索树中没有意义的不需要搜索的树枝，以提高运算速度。

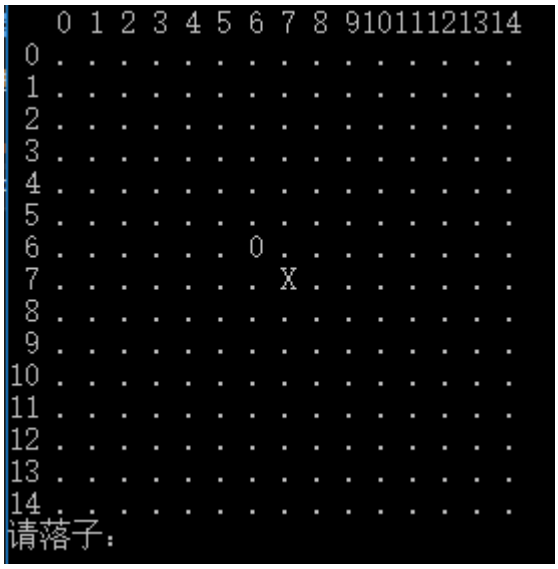
该算法与加深预测步数、提高搜索效率有关。



当前实现程度：

(实现想法/设计思路/优缺点/改进方向)

1. 界面



O 为 Player　X 为 AI

实现方法：char 型二维数组

优点： 因为棋盘是 15*15 的固定格式，使用二维数组可以快速定位，读取/修改。

2. 博弈树预测层次

当前进度：只就简单的进行一层的搜索。未进行递归实现多层次遍历。

Alpha-Beta 剪枝算法暂未实现。

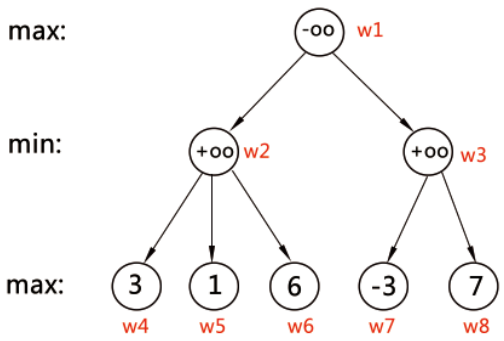
缺点：只能进行最简单的一步判断。无法看穿对手的引诱棋、诱杀等下法。因此棋力很低。

未来计划/改进方案：

1.极大极小值搜索算法

使用递归的深度优先搜索：

在 AI_turn（偶数层）上找到子结点中权值最大，在 Player_turn（奇数层）上找到子结点权值最小。



假如有如上图的博弈树，设先手为 A ，后手为 B ；则 A 为 max 局面，B 为 min 局面。上图中 A 一开始有 2 种走法，因为 A 是 max 局面，所以它会取 f(w2) 和 f(w3) 中大的那个，设定叶子结点局面的估价值。

例如上图的搜索过程为 w1 --> w2 --> w4 ，然后回溯到 w1 --> w2 得到 f(w2) = 3 ，接着 w1 --> w2 --> w5 得到 f'(w2) = 1，因为 w2 在第二层，**是 min 局面，所以它会选择得到的结果中小的那个，即用 f'(w2) 替代 f(w2) ，即 f(w2) = 1**，接着 w1 --> w2

--> w6 得到 f'(w2) = 6 > f(w2) ，直接忽略。因此如果 A 往 w2 走的话将会得到一个估价值为 f(w2) = 1 的局面；类似地，如果往 w3 走的话将会得到一个估价值为 f(w3) = -3 的局面。而 **A 是 max 局面，所以它会选择估价值大的走法，f(w2) = 1 > f(w3) = -3，因此它下一步走 w2。**

2. Alpha-Beta 剪枝算法

在上例子中， f(w2) = 1，接着 w1 --> w2 --> w6 得到 f'(w2) = 6 > f(w2) ，直接忽略。因为如果下在这里，是会使对手获益，即使自己受损。因此该结点和该结点的子结点不需要再考虑，即为**剪枝**。可达成提高效率目标。

3．评估函数

当前进度：首先完成 Value[Size][Size]的初始化

```
const int Value[SIZE][SIZE] =
{
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,1,1,1,1,1,1,1,1,1,1,1,1,1,0},
{0,1,2,2,2,2,2,2,2,2,2,2,2,1,0},
{0,1,2,3,3,3,3,3,3,3,3,3,2,1,0},
{0,1,2,3,4,4,4,4,4,4,4,3,2,1,0},
{0,1,2,3,4,5,5,5,5,5,4,3,2,1,0},
{0,1,2,3,4,5,6,6,6,5,4,3,2,1,0},
{0,1,2,3,4,5,6,7,6,5,4,3,2,1,0},
{0,1,2,3,4,5,6,6,6,5,4,3,2,1,0},
{0,1,2,3,4,5,5,5,5,5,4,3,2,1,0},
{0,1,2,3,4,4,4,4,4,4,4,3,2,1,0},
{0,1,2,3,3,3,3,3,3,3,3,3,2,1,0},
{0,1,2,2,2,2,2,2,2,2,2,2,2,1,0},
{0,1,1,1,1,1,1,1,1,1,1,1,1,1,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

该初始化的思路为：越靠近棋盘中心，权值越高。所以最开始几步，其他判定条件限制较少时候，Ai 选择下在中心附近。

接下来对该点权值的判断有两大难点：**搜索与权值的选择**

搜索：

我们选择已经有落子的点开始向八个方向发出“射线”，并**记录与该点的连续同色点有几个。**例如：如果有连续四个，即完成五子连珠，游戏结束。如果在 chessmap[i][j] 上方向有连续三个，下方向无对手棋子，则 chessmap[i][j-4]与 chessmap[i][j+1]的 Value 值提升至次高级，除了对手有冲死（即为威胁）情况。

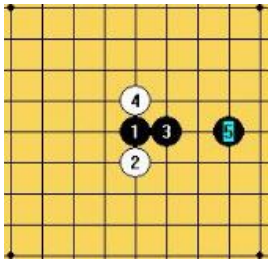
搜索过程需要注意**控制边界。**如果一遇到边界值即“射线”停止，计算连续值结束。

权值设定 v1.0：

```
if t1 >= 4 Value[i][j] += 10000000;          //出现五子连珠

if t1 == 3 Value[i][j] += 100000;            //出现四子连珠

if t1 == 2 Value[i][j] += 1000;              //出现三子连珠

if t1 == 1 Value[i][j] += 10;                //出现二子连珠
```

缺点：

只考虑了单方向的连续得分，**未考虑跳三（XX?X）等跳跃情况。**会对棋力有会很大削弱。



改进方案：

增加判定细化条件，增加更多的判定函数。争取完成一个全面的判定系统。

如上图这种情况需要对八反方向中的两个方向进行组合讨论，增加来了讨论复杂度。

最终，遍历每个没有落子的点，求出 Max_Value/Min_Value ,根据所在层数选择极大值点或者极小值点。

算法评估

由设计思路可知，我们的算法是对可选位置进行多层次的评估，选出在未来几步中为自己都带来优势最大的落子位置。

1.算法的搜索深度与 Ai 的棋力有关系。但是每加深一层搜索，时间复杂度、空间复杂度都成指数级增长。因此存在棋力与效率的矛盾。由此得出，**通过更巧妙的方法避开博弈树中的重复评估的点**是提升算法效率的主要方向。

2.Ai 的棋力还与**评估函数中评估值的准确性**息息相关。目前只用了最为简单的 1000/10/10/1 等 Value，接下来的算法改进过程中需要重新细化置值。

3.目前**棋局的判断还过于简单**，没有跳跃型的判定。需要增加更多的评估函数增强 Ai 对棋局的了解。

总结

设计一个这样的五子棋的 AI，是我第一次初步接触这类博弈类型的项目，也是我第一次接触 AI。虽然这个简单的五子棋程序离到真正的人工智能还差很远，但通过这两周的初步接触，查阅资料，从他人的博客中汲取灵感，学习更高效的算法，是一个不断发现问题，解决问题的过程，也为以后更具挑战性的项目积累了自主解决问题的能力基础。

前两周主要工作在分析问题，设计思路，并且对存在的主要问题找到了相关解决方案。在接下来的时间内希望能完成的程序能达到期望水平。