Design for Gomoku AI

Week 3rd

By Liu Aishan (09016411)

序

心路历程：一开始试图使用控制台应用程序来写五子棋，由于做此类完整的游戏经验不足，

为了能够后面方便测试，开始写算法之前耗费了不少时间和力气去思考框架以及棋盘的表示问题，由于多个类（AI，人，棋盘）调用二维数组，因此把数组声明成了全局的，感觉思路不是十分清晰，应该把重心放在五子棋的算法上，于是打算使用 Cocos2d-x 引擎来便利基础部件的使用与插入。
下图为之前的思路。

Report_1

1. 基本思路。
   实现这个程序一开始想要从类的角度入手考虑，用 C++面向对象的程序语言。
   类：棋盘（Chess Broad）
   该类的 public 成员包括一个二维数组存储三种整型数字分别表示 还未下子 黑棋下子 白棋下子。
   该类的成员函数包括：开局；~~判断是否有棋胜出（）~~；重新开始一局（重新赋值矩阵 ）；（判断人的一方是否悔棋）；
   AI（是棋盘类的子类？因为需要一直对公有成员进行操作）
   该类成员目前考虑不够成熟全面，下面就目前想到的进行罗列，不足的地方或者思路狭窄处望老师指正，谢谢！
   目前想到的 AI 类只包括成员函数不用包含私有成员的属性，成员函数包括：下子时调用的函数（包括对当前矩阵情况的评估以及根据评估算法的结果给选定的矩阵的位置进行赋值以及判断自己是否胜出）（搜索已被自己赋值的点的八个位置，若有子则 count++，若没子或者有对方的子则结束搜索，可以用递归吗？（感觉是重复同一方法进行搜索））；
2. 待解决问题。
   每走一步都要看整个棋盘自己有没有赢很废柴，有没有方法可以给连得大于某个值的位置进行存储也可以对因为被破坏不可能再获胜的位置进行删除，直接对这个部分进行查询即可。
   往哪里开始下子 黑白棋？
   怎么调整自己的代码在平台上的和别人的 AI 进行 PK？
   蒙特卡洛树
3. 待改进。
   在生成棋盘的时候没有用动态生成因为觉得赋值为零比较消耗步骤，如果可以初始化全零那么在一局结束和开始的时候直接 new 和 delete 会简化一点，现在的话重新开始只能重新赋值全零。

以下为以 Cocos2d-x 为基底去设计五子棋 AI 的框架与目前的算法思路。由于前面耗费时间长，所以五子棋还没有完整实现，目标为在下一次的 Week-5,实现完整对局，通过测试具备较为合理的分值比较系统。

目前参考的文章：cocos2dx 基础篇(5)——浅析几个重要类 （作者：夏天的风）

《算法导论》
百度百科
Alpha-Beta 剪枝算法(Alpha Beta Pruning)详解（作者：小白的 2015）

算法所使用的数据结构：试图使用树的数据结构进行构造。

思路来源：在算法课上讲到动态规划部分时，有一道例题是两个人在一列随意排列的数中依次抽取首或尾的数字，且双方一样聪明，设计算法使 A 某抽取数字之和最大。课上有同学提出了一个算法是利用二叉树列举当前连接当前值（父节点）通过以正负区分对手选值和己方选值（结点之间的边）到达下一步的己方总值情况（子节点）。虽然在这道问题上，二叉树考虑所有情况的算法的时间复杂度在指数规模上，效率较低，但是为我提供了思路和启发，对于面对“博弈”二字还十分陌生的我来说，我认为，这种思路在一定程度上包含了博弈的思维，因为影响最终取胜与否的不仅仅是当前这步自己的选择，还有在自己进行选择后对手的选择，依次递推为自己分值达到最大化进行深度的考察。这种不只考虑一步的深度的思考以及正负值来代表对手和己方选择对己方的影响的两个思维角度的转化给了我很大启发，让我觉得这种想法也可以运用到其他的博弈相关的问题中，比如五子棋。

遇到问题：不像取数一样，五子棋的选择不是二选一，在大的方面去看涵盖了 15*15 个位置中所有未放棋子的空白位置，也就是要通过比较在空白的位置中选择最佳的位置。而“最佳”二字大概就是决定这个 AI 是否智能是否“聪明”的关键了,因此确定“最佳成为了核心”。另一方面，如动态规划中的题目可见，这种思路的效率比较低，会涵盖问题中的一些重复的情况，加上棋盘上点一开始很多空白，要进行比较的可选择项太多，可能程序在运行起来会比较低效，因此在遍历这些位置上，判断五子的过程中以及构建树的结构时能否通过改进来减少不必要情况的计算成为了代码是否清晰高效的关键，也是在实现五子棋完整功能后需要积极去改进的部分。

对于“最佳”位置的情况的几点考虑：1.最主要的部分是对于五子，四子，三子，可构成双三子，对方成四子，三子，双三子这些情况的比较。**2.**刚开始还没有头绪去码代码的时候，在线上对弈平台 Botzone 观看了几局对弈发现，对于刚开局时前几枚棋子的落子位置的选择上要避免与对手纠缠，有一局是执先手的子一直在棋盘上随机落子，而后手一直在黑子刚下的新的随机位置相邻的八个位置随机依附，因此这种情况下在落子是防守还是进攻的选择上需要进行考虑。**3.**对于到达边界的情况的也要作为权衡考虑的因素。

目前通过网络了解到的有关算法：针对问题中的第二方面的树的构建预计会比较庞大有赘余的问题了解到 Alpha-Beta 剪枝算法。

Week-5 goal：

1. 经过测试得到针对不同情况得到与其相对应的比较合理的分值，列出表格在实验报告中有所体现；
2. 针对剪枝算法进行认真学习并运用到代码中在实验报告中张贴在五子棋程序中相对应的代码并涵盖自己提炼的关键知识。
3. 实现完整对局并且具备可以重新开局的功能。