五子棋实验报告

沈冠霖 2017013569

夏金城 2018013404

摘要

程序使用 Minimax 算法和 alpha-beta 剪枝实现了五子棋 AI。在获取下一步操作中，程序根据五子棋的特性做了剪枝和排序优化。在搜索过程中，程序用散列表进一步剪枝优化。除此之外，程序还实现了悔棋和复盘，和玩家交互鲁棒性较好。

关键词

Minimax 算法，alpha-beta 剪枝，散列表

1. 程序结构

全局常量变量，和一些关键函数实现：define.h,define.cpp

与玩家交互的函数，包括选择颜色，玩家下棋，悔棋，复盘，还有对应的异常处理等：makemove.cpp

估值算法实现：evaluate.cpp

判断游戏结束：gameover.cpp

获取所有下一步操作：createmoves.cpp

搜索算法：searchmove.cpp

主函数：start.cpp

2. 数据结构，变量，异常处理设计

首先，为了处理玩家选择黑白棋的两种情况，程序定义了全局变量 Player,Computer 来代表当前玩家，电脑的颜色，用 CurrendSide 代表当前下棋的一方，用 Opposite 函数求得对方。这样避免了重复的 if else，简化了逻辑，增加了程序可读性。

其次，为了实现悔棋，复盘操作，程序定义了两个 vector：ComputerList,PlayerList.用于记录双方的下棋位置。这样悔棋只需要 pop_back，复盘只需要按顺序正向输出即可。

程序仍然用 chessBoard 二维数组来表示棋盘。不过为了避免出现越界访问，程序封装了 GetPlace 函数。

最终，对于与玩家交互的部分，程序做了很多异常处理，包括处理错误指令，错误的下棋位置，错误的悔棋等，增加了程序鲁棒性。

3. 估值算法和游戏结束判定

程序通过估计整个棋局的所有棋型所占分数来获取当前棋局的分数。比如要估计黑棋所占分数，就遍历所有黑色棋子，先找到其八个方向上所在的所有棋型，取每个方向的最大棋型，然后将其分数求和。

同时，程序记录了所有得分棋型的形状和点，以便于进行判重。同时，如果一个点在多个方向都有得分棋型，程序给其设置了额外加分。

判断游戏是否结束也基于棋型判断法---如果能找到 11111 棋型，则游戏结束，有一方胜利。如果棋盘满了还没有 11111 棋型，则和棋。

4. 获取下一步的操作

4.1 优化思路

对于五子棋来说，所有在棋盘上没被落子的位置都可以是下一步的操作。但是这样最多有 15*15=225 种，而其许多操作是没有必要的。（无优化的函数是 CreateMovesNaive()）

因此，程序进行了一定的剪枝---剪掉所有周围（八个方向）没有任何棋子相邻的位置。这样经过实验，不会显著影响 AI 的强度，但是可以大大提高搜索效率，尤其是刚开局的时候，可以把 225 种分支减少到十多种，大大提高效率。（做了这种优化的函数是 CreateMovesWithNeighbor()）

其次，很多情况下，新的最优解是落在上一次落子位置边上的（比如落在对方上次落子边上的封堵防守，落在自己上次落子边上的进攻）。我们也把上一次双方落子周围的位置单独拿出来优先搜索。（同时做了两种优化的函数是 CreateMoves()）

4.2 实验结果

(以下数据结果在 Release 模式下进行 5 次试验取平均值得到，其中每组对比实验保证走法和总步数保持相近）

1.不排除没有任何棋子相邻位置和排除的性能对比

性能	总搜索次数	平均搜索次数	总搜索时间(ms)	平均搜索时间(ms)
无优化	3847926.8	459267.4	33066.0	4153.2
排除相邻位置优化	12175.4	1775.0	148.6	24.8

2.不排序和排序的对比

性能	总搜索次数	平均搜索次数	总搜索时间(ms)	平均搜索时间(ms)
不排序	38441.6	4305.4	829.0	103.4
排序	21087.0	2389.0	558.8	54.8

由以上结果可知，两次优化前后，算法的时间性能和搜索次数性能均有明显的提升。

5. 搜索算法

5.1 Minimax 算法和 alpha-beta 剪枝

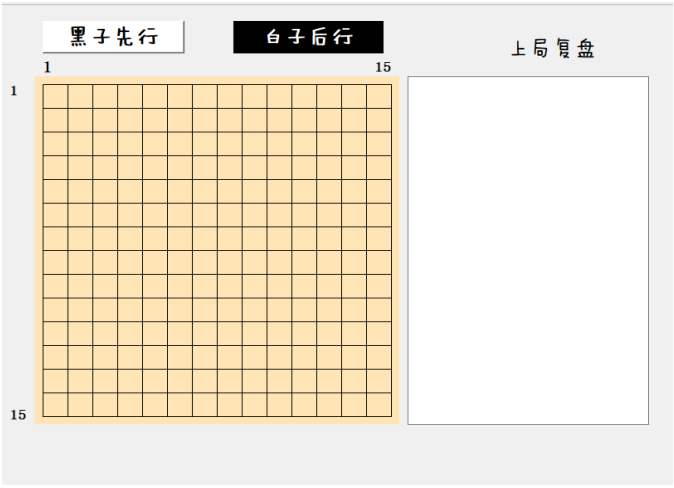
算法采用了如下逻辑：

```
int AlphaBeta(int depth, int alpha, int beta)
{
   if (depth == 0 || gameover())
      return Evaluate();
   GenerateLegalMoves();
   while (MovesLeft())
   {
      MakeNextMove();
      val = -AlphaBeta(depth - 1, -beta, -alpha);
      UnmakeMove();
      if (val >= beta)
         return beta;
      if (val < alpha)
         alpha = val;
   }
   return alpha;
}
```

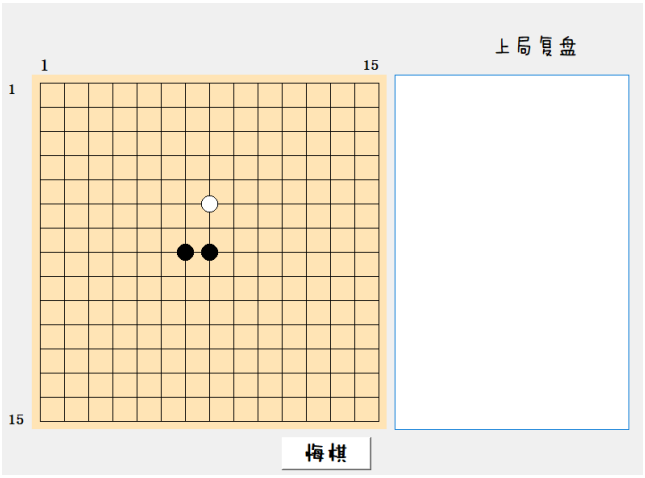

这个是 alpha-beta 剪枝算法的一种变式版本。因为五子棋 AI 中，双方估值都是估计的自己的最大值，那么对手的最大值取负，就是你要的 min 最小值。因此，用-beta –alpha 替代 alpha beta，用负极大值替代极小值，就能实现 maxmin 函数的 min 部分，max 部分则完全不需要改动。剪枝已经大大优化了算法了，结果参考实验部分。

5.2 图形界面

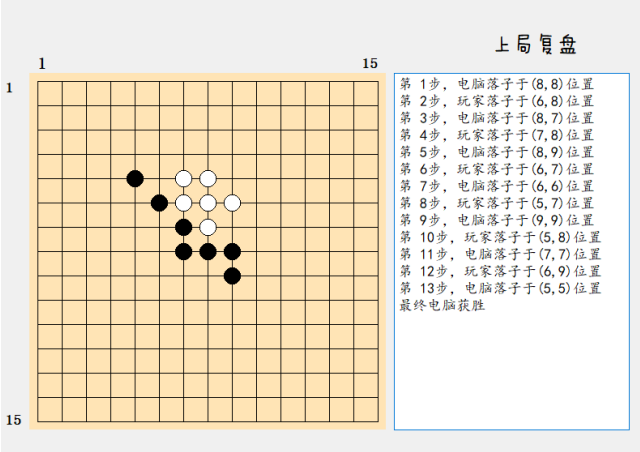
图形界面采用 Qt 实现。将原框架中的核心代码导入 Qt 中，同时进行如下修改:命令行中的输入选择先手后手命令和悔棋命令改成按钮操作，将输入坐标改为点击棋盘相应位置进行下棋，达到游戏结束条件弹出提示框进行提示，同时游戏结束，旁边显示栏进行复盘信息显示。图形界面如下:



点击棋子选择按钮后可以开始进行鼠标下棋和悔棋操作。图形界面如下（以白棋后手为例）：



当达到游戏结束条件时进行提示并显示复盘信息。图形界面如下（以电脑获胜为例）：



5.3 实验结果

(以下数据结果在 Release 模式下进行 5 次试验取平均值得到，其中每组对比实验保证走法和总步数保持相近）

不剪枝和剪枝的性能对比

性能	总搜索次数	平均搜索次数	总搜索时间(ms)	平均搜索时间(ms)
不剪枝	215522.2	21542.4	4281.0	478.8
剪枝	29087.6	2789.4	547.0	63.8

由以上实验结果可知，采用 alpha-beta 剪枝算法后，程序的的时间性能和搜索次数性能均有明显的提升。

6. 实验总结

这次实验让我们深入了解了搜索的基本步骤，包括寻找初始状态，可行的状态转移，构建评估函数，以及搜索算法。以及让我们进一步熟悉了 alpha-beta 剪枝算法等技术，提高了编程能力。

7. 参考资料

[1] 百度百科：alpha-beta 剪枝算法（参考了基本逻辑）
https://baike.baidu.com/item/AlphaBeta%E5%89%AA%E6%9E%9D%E7%AE%97%E6%B3%95/18261650

[2] 简书：博弈树 alpha-beta 剪枝搜索的五子棋 AI（参考了获取可行步的优化，还有评估函数）
https://www.jianshu.com/p/8376efe0782d