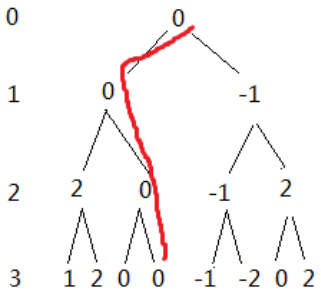
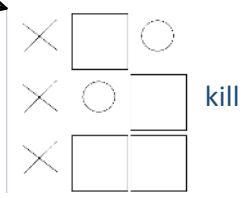
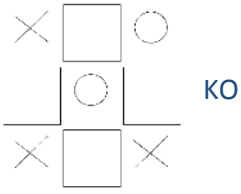
Readme MinMax算法

2016年4月4日 14:38

1. 思路
   a. 使用深度限制min/max算法，向下探索三层，根据对某个state做出评估，之后从下到上回溯，根据player = -1/1 求出当前层的min/max，直到最上层，找到应该走的路线（如图）
   b. int calc_V(vector<TicTacToeState> state, int depth, int player)，通过该函数，指定depth，算出某个state的估值；
   c. 计算估值的方法，主要分为3种，记为kill、KO、draw，return 为 -100/100 -50/50 0
   - Kill----已经3个连成了一条线（如图）
   - KO---下一步，无论对方走在哪里，都可以kill对方（如图）
   - Draw---接近平局，或者已经游戏结束







   d. 一个优化

   假设第一步就要使用depth=3的搜索，则需要对比8*7*6种情况，对性能损耗严重，而且前几步比较得到的大部分为非kill、非KO的结果，对AI的侵略性有损。经过人工计算，第一步只有拿中间和拿角的情况是不会输的，故当 chess_count == 1的时候，不进行搜索，而是直接放在中央cell或者任意角cell
   - Chess_count == 1，计算量最大，故直接给计算结果，很合理

2. 代码审计
   a. 缺点
   - 在初始版里，有大量的memory leak，而vector的特点看来，vector.clear()并不会真正释放内存，而malloc的指针已经被丢弃，故只能使用

     vector<TicTacToeState>().swap(state);

     这样的方法去释放内存，从而解决memory leak
   - 数据结构设计问题，重复计算各行各列的sum，应该在结构体建立时，就计算其sum，存入TicTacToeState

   b. 优点
   - 估值的计算，使用简单的100/50/0来表示，简化流程

3. 运行结果（无图）

   AI给出的解，永远是平局