四子棋

陈新 2018013443

1 环境

Windows

2 算法简介

采用信心上限树UCT算法，构建UCT树，并一步步模拟四子棋对战落子过程，计算每个结点的收益，从而确定最佳落子点

每次决策时，都以当前状态为根节点建立UCT树，隔层的结点分别为AI、USER落子的状态。每次UCT过程都从根节点，若当前结点所有子结点均已拓展完毕，则通过信心上限计算出最好的子结点继续循环；若当前结点有尚未拓展的子节点，则拓展子节点，跳出循环。

信心上限计算公式为：

其中为手臂j的胜利次数，为手臂j的总访问次数，T为父节点的访问次数，c为常数参数，我选取了1.38

跳出循环后，对被选中的结点展开模拟落子，直到决出胜负。然后根据胜负决定一个收益（AI胜为1，平局为0，USER胜为-1），不断回溯更新结点的总收益。

最终决定落子点时，寻找根结点的子结点中平均收益最大的落子。

3 代码结构与具体实现

1. Node类（结点）：
   1. 成员变量

Node\* father;

int x, y; //记录该节点的上一步对手的落子位置

int visitedSum; //该节点的访问总数

int turn; //记录该节点的棋权为AI还是USER

int depth; //该节点的深度（暂时没什么用）

int expandableSum; //N列中，未满、可以落子的有几列

int expandableIndex[12]; //前expandableSum个位置记录可以扩展的结点列标

Node\* children[12]; //子结点

double profit; //该结点收益

* 1. 成员函数（有具体功能的）

bool Expandable() const; //是否可以扩展子节点，即棋局是否还能往后面下

bool gameEnd(int\*\* board, int\* top) const; //游戏是否已经结束

Node\* expand(int\*\* board, int\* top); //扩展子结点

Node\* bestChild(int\*\* board, int\* top); //寻找子结点中的最优解

void backUp(double deltaProfit); //回溯更新自己以及各父结点收益

1. UCT类
   1. 成员变量

Node\* father;

int x, y; //记录该节点的上一步对手的落子位置

int visitedSum; //该节点的访问总数

int turn; //记录该节点的棋权为AI还是USER

int depth; //该节点的深度（暂时没什么用）

int expandableSum; //N列中，未满、可以落子的有几列

int expandableIndex[12]; //前expandableSum个位置记录可以扩展的结点列标

Node\* children[12]; //子结点

double profit; //该结点收益

* 1. 成员函数（有具体功能的）

bool Expandable() const; //是否可以扩展子节点，即棋局是否还能往后面下

bool gameEnd(int\*\* board, int\* top) const; //游戏是否已经结束

Node\* expand(int\*\* board, int\* top); //扩展子结点

Node\* bestChild(int\*\* board, int\* top); //寻找子结点中的最优解

void backUp(double deltaProfit); //回溯更新自己以及各父结点收益

由于UCT算法的性能主要决定于进行模拟的次数，所以效率显得非常重要。我取消了UCT树中每个结点的棋盘局面记录，因为每次模拟，都只会以根节点未开始，选择其中某条路线走到底，如此一来棋盘局面也是可以模拟得来的，因此就节省了空间和动态申请空间的耗时，从约27万次模拟优化到约44万次。

4 对抗结果

Saibo网站账号：kasumi777

批量测试 #1757

[https://www.saiblo.com/batch/1757](https://www.saiblo.com/batch/775)

胜97 负3 平0 胜率97%

5 总结收获与改进方案

**总结收获：**

这次作业加深了我对蒙特卡洛、UCT的理解，让我能够应用算法进行实际问题的解决。

**改进方案：**

我试过更新每个结点的**必胜/必负/不确定**状态，来进行剪枝，但是没完成debug

在UCT模拟时，避免进入对手必胜的结点，理论上可以使得尽量多的模拟次数用在对己方有可能胜利的分支，会对那些分支有更准确的胜率刻画。

做法如下：

UCT树的叶子结点是胜负已分的，状态为AI必胜/USER必胜二者中的一个。

若当前结点轮到AI落子，子结点中一旦有AI必胜的结点则更新当前结点为AI必胜；若子结点全部是USER必胜，则当前结点更新为USER必胜，否则为不确定。

若当前结点轮到USER落子，子结点中一旦有USER必胜的结点则更新当前结点为USER必胜；若子结点全部是AI必胜，则当前结点更新为AI必胜，否则为不确定。

则一个结点如果状态为不确定，那么必定有至少一条确定胜负的子路线没有被模拟过。

在模拟时，避免进入结点方必败的分支。