重力四子棋作业

电 62 班 李明轩 2015010705

一、 算法介绍

算法采用信心上限树算法（UCT 树）及蒙特卡洛模拟，这种算法**适用于通过不断的模拟给出结论的场合，不需要很强的先验知识**。

对于每个节点，按照如下算式，计算子节点的权重：

$$\frac{Q(v)}{N(v)}+c\sqrt{\frac{\ln\left(N(parent(v))\right)}{N(v)}}$$

其中 Q(v)是当前节点总得分，N(v)是当前节点总访问次数，N(parent(v))是其父节点的总访问次数。上式中，第一项是下棋的平均得分，第二项代表了子节点的未知程度，**从而使得访问次数较少的节点也能够被充分地探索**。

UCT 树主要就是选择子节点→探索新节点→随机模拟→结果回溯的过程。随机模拟时等概率选择落子点，直至分出胜负。每模拟一次，当我的程序胜时计 1 分，对手胜时计-1 分，平局计 0 分,将结果加入到各祖先节点的 Q(v)中。

程序花费 2.5 秒将树构建完毕后，按照第一层中$\frac{Q(v)}{N(v)}$最大的节点进行落子。

二、 编程实现

程序中主要包含一个 UCT 类及 Node 结构体。前者主要用于构建 UCT 的各个方法，后者用于存储每个节点的各种信息。

UCT 类中主要的变量和方法如下：

```
int size;  //树的规模，即总节点数量
int M, N;  //棋盘规模
Node* nodes; //节点数组
int** currentBoard; //临时棋盘布局，用于蒙特卡洛模拟
int* currentTop;
int** origBoard;    //初始棋盘布局，在此基础上用top记录子节点信息
int* origTop;
double c;          //超参数c
int lastX, lastY, noX, noY;
clock_t start_t, end_t;  // end_t - start_t < 2.5s
```

```
UCT(……); //构造函数
~UCT();  //析构函数
void UpdateCurrentBoard(int node_id);  //更新临时棋盘
int Expand(int node_id, int i);        //扩展
int BestChild(int node_id, double c);  //选择
double DefaultPolicy(int node_id);     //模拟
void Backup(int node_id, double delta); //回溯
int TreePolicy(int node_id);    //探索
Point UCTSearch();              //主方法
Point OneStepWin();             //寻找一步获胜的策略，没有再用UCT算法
void PrintNodes();
```

Node 结构体主要变量如下：

```
    bool can_expand[MAX_N]; //子节点是否可以拓展，即各列是否还能落子
    bool expanded[MAX_N];   //子节点是否已拓展
    int player; // 1:oppo 2:me
    int winner; // 0:not finish; 1:oppo win; 2: me win
    int parent_id; //父节点id,用于回溯
    int avail_child_num;  //可以拓展的子节点数，即可以落子的列数
    int childs_id[MAX_N]; //子节点id,用于扩展和模拟
    int top[MAX_N];       //子节点top位置，结合origBoard及祖先节点落子位置，可以推断出当前棋盘
    double visit_num;     //N(v)
    double total_reward;  //Q(v)
    bool gameover;        //游戏结束,不能再扩展了

    int newX; //相比于父节点，多下了哪个位置？
    int newY;
```

主要踩过的坑：

1. 慎用指针。许多函数中都需要创建一些临时数组，一开始都是用 int* x = new int[N]的方式，并且没考虑 delete。模拟的时候发现内存占用涨的飞快，琢磨了很久才发现问题，因此后来尽量不用指针，都改成 int x[N]这样的形式了
2. 指针用起来很麻烦，后来索性 Node 就不用 Node* nextNode 的形式了，直接创建一个 Node 数组，父节点、子节点 id 都存起来，方便访问，也方便一次性 delete。
3. 原先是每个 Node 都存储了一个棋盘 Board[][]，这样占用空间大，在 saiblo 网站测评的时候探索的深度有限。考虑了一下可以对每个节点只存储 top 及新的落子点 newX,newY，这样省空间。结合 origBoard 及祖先节点的落子信息就可推断出当前节点的棋盘样子了，不用每个节点都存个棋盘。
4. Start_clock 越早赋值越好，不要进入循环了才开始计时，不然总时长可能会超。

最后在本地和 saiblo 网站都测试成功。提交到 saiblo 网站时要把 strategy.cpp 中的#include "UCT.h"改成#include "UCT.cpp"，不然会编译错误。

三、 效果展示

和所有 2n.dll 都对局一次，结果存储在 results 文件夹下的 2n.txt。整体统计如下。

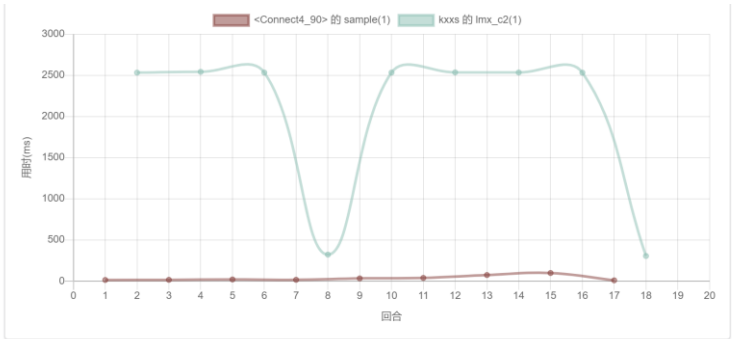
胜数	平数	负数	总数
73	0	27	100

此外对于 92~100.dll 各对决 5 次(即各 10 局)，胜率如下：

对手	胜率
90%	70%
92.dll	80%
94.dll	20%
96.dll	60%
98.dll	50%
100.dll	50%

可见算法是有一定效果的。

此外在 saiblo 网站进行评测，结果如下。
每回合用时 2.5s 左右，使用内存 400MB。



基本是 1000pts 守门员水平。

四、 总结收获

对于蒙特卡洛方法和UCT的思想有了更深的认识，体会到了随机模拟的效率和效果，并且对于C++小白来说也是一次深刻的编程体验。感谢老师和助教的付出。