四子棋实验报告

1. 算法思路

本次作业主要使用蒙特卡罗树搜索以及信心上限树算法，从当前的棋局出发，选取出算法下最优的落子点。

1.1 蒙特卡罗树搜索

蒙特卡罗树搜索是一种动态的搜索策略，由已知的初始状态出发，通过不断的尝试新的状态，来逐步扩展树的节点；而每当扩展出一个新的状态，便对此状态进行一次随机的模拟，并将获得的收益反向更新到根节点。整个过程分为四步：

选择和扩展：从根节点出发，不断扩展至新状态，形成新的子节点，若已无法继续扩展，则通过特定的选择标准，选择出最优的子节点，继续扩展。

模拟：对上步中得到的新状态进行模拟，双方轮流随机落子，统计棋局最后的结果

回溯：将所获得的收益向父节点传播，更新父节点的胜率、场次等等信息，完成搜索。

1.2 信心上限树搜索

信心上限树可以说是蒙特卡洛树搜索的一种改进，在典型的蒙特卡罗树搜索中，选择最优子节点的标准一般仅考虑了当前已知节点的胜率，但显然，在探索已知价值更高的节点的同时，也应兼顾未曾被探索的节点，因而信心上限树采取了如下的评价标准：

$$I_v=\frac{Q(v)}{N(v)}+c\sqrt{\frac{2\ln(N(parent(v))}{N(v)}}$$

2. 代码实现

算法的实现与课程资料所给的 UCT 算法伪代码相同，关键的代码实现及说明见下

```
Node* Uct::uctSearch() {
    clock_t startTime = clock();
    int cnt = 0;
    while ((double)(clock() - startTime) / (double)CLOCKS_PER_SEC < TIME_LIMIT) {
        cnt++;
        Node *v = treePolicy();          //选择或扩展节点
        int delta = v->defaultPolicy(); //模拟计算收益
        v->backup(delta);               //回溯
    }
    //printf("the cnt is %d\n", cnt);
    Node *bestChild = root->bestChild();
    return bestChild;
}
```

Uct 类下的 uctSearch()为主搜索函数，treePolicy()为选择或扩展节点函数，而其余的模拟、回溯等函数皆定义于节点类 Node 中

节点类中的主要成员变量如下

```
int state[K][K];
int top[K];          //棋局状态
int winsNum, visitedNum;
int x, y;            //落子位置
int player;          //棋手身份
Node *parent;
Node *children[K];
int topIndex;        //扩展位置
int childrenNum;     //子节点数量
int status;          //节点所处状态
```

其中 status 用于存储节点所处状态，如若胜利，则为 1，平局为 3，尚未结束则为 0

3. 优化细节

1）对课程所提供的 AI 进行了若干次实验，特别是根据与 90-100 号 AI 对局的胜率，发现 C 取 0.7 时效果比较好，故最终选取 C=0.717；

2）本次作业中，在选取最优子节点时，增加了对必胜位置和必输位置的判断（具体的实现参考了所提供的 Judge 函数），当遇到必胜位置（自己落子在此处即可连成 4 个获得胜利）或者必输位置（敌方落子在此处即可连成 4 个）便立刻返回。

3）此外，在对局初期的时候，为了将棋子下至敌方棋子周围，在选择子节点时还采取了此节点周围敌方棋子数量这一指标，当然，当模拟的次数足够多时，这一指标应越发不再重要，即与模拟的次数成反相关，故而更改 UCT 的评价指标为

$$I_v' = I_v + c' * \frac{cnt(enemy\_around(v))}{\log(N(v))}$$

最终的评价函数为：

$$I_v = \frac{Q(v)}{N(v)} + c\sqrt{\frac{2\ln(N(parent(v)))}{N(v)}} + c' * \frac{cnt(enemy\_around(v))}{\log(N(v))}$$

4. 实验效果

本地测试结果

id	wina	winb	losea	loseb	tie	bug
2018010156	47	43	3	6	0	

结果	胜	负	平
比例	0.9	0.09	0

Saiblo 平台账号结果：

批量测试 #13398

93	7	0	100	100	93%
胜	负	平	已测评局数	总局数	胜率

被测试 AI

<2021IAI 2018010156>
final v8

结果	胜	负	平

比例	0.93	0.07	0

说明：在本地测试时，第一次所测得的结果并不太好，特别是在与 50.dll 及其以下的 AI 的对局中，出现了较多的负局，但之后再单独与它们测试时，发现胜率基本都为 1，这一结果也与平台上的测试相符，具体的原因现在还未搞清。

5.总结收获

本次四子棋大作业，让我对 UCT 算法的原理及应用更加熟悉。同时，通过网站提交后还能在线与所写的 AI 进行对战，觉得蛮有乐趣与成就感的（目前我还没下赢过我写的 AI），但不知为何，同样的代码在网站上批量测试时有时会出现 tle，可能是因为服务器波动？不过，总的来说，这次作业还是让我收获很大。