计算概论大作业不围棋实验报告

信息科学技术学院　　2000013180　　王思远

1. 实验目的与要求

1.1. 通过解决不围棋 bot 这一实际问题，提高设计搜索算法的能力以及实现能力、调试能力

1.2. 加深对 c++语言的理解

2. Bot 设计

2.1. UCT 算法

首先，在研究传统贪心算法、αβ剪枝优化的 minmax 搜索、UCB 算法以及 UCT 算法并对其拟合能力进行比较后，我选择了 UCT 算法作为设计不围棋 bot 的基础。

在不围棋算法中，我们首先定义了完全博弈树 T 和博弈树 T'的概念。显然，完全博弈树是非常大的，不可能在可以接受的时间范围内对其进行完全的遍历。因此，基于完全博弈树的搜索算法，就是构造出接近最优解的博弈树。

UCT 算法其实就是将 UCB 算法应用于蒙特卡洛树上。蒙特卡洛树的构造方法如下:

(1) 选择：从 T'根节点向下搜索，每次递归至胜率最大的子节点，遇到叶子节点（在该点，棋局已经结束）或者非全扩展节点（有它的后继棋局没有被加入其子节点）时结束。

(2) 扩展：对第一步选择的节点 u 随机扩展一个子节点 v，若是叶子节点则使得 v=u，不向下进行拓展。

(3) 模拟：对 v 的局面进行随机模拟，以其胜负情况作为 v 的评分。

(4) 回溯：对 v 的父节点一直到根节点的路径进行回溯，更新胜率。

(5) 回到第(1)步。

这样不断对博弈树进行扩展，最后选择根节点的胜率最大的子节点作为下不围棋的行动，可以找到一个相当优秀的解。然而，蒙特卡洛树方法有一个缺点，就是它缺乏对模拟次数较少的节点的“好奇心”。这些模拟次数较少的节点，其模拟得到的胜率与实际胜率相比可能相差很多。简单地通过选择模拟胜率最高的子节点进行扩展，很容易陷入贪心而错过最优解。

UCB公式： $\bar{x}_j + \sqrt{\frac{2\ln n}{n_j}}$

$\bar{x}_j$为观测到的第j个臂的平均回报，

n_j为 目前为止按压第j个臂的次数，

n为到目前为止按压所以臂的次数和。

UCB：upper confidence bound 上限置信区间

$\bar{x}_j$表示期望回报，$\sqrt{\frac{2\ln n}{n_j}}$表示探索的程度，可以理解为$\bar{x}_j$的不确定程度

$\bar{x}_j + \sqrt{\frac{2\ln n}{n_j}}$表示的是$\bar{x}_j$的置信区间上限，即$\bar{x}_j$可能的最大期望回报

这时，通过 UCB 算法来选取 1.选择 时的最优子节点，就可以提高算法的“好奇心”，更难错过最优解。这就是 UCT 算法的基本思路。

在实现了基础的 UCT 算法并提交到 botzone.org 后，我发现在 1s 的时间限制内，UCT 算法的拟合精度甚至不如设计较为精巧的贪心算法。也就是说，必须对 UCT 算法进行优化。

2.2. 对叶子节点的优化

在上述博弈树 T'中，叶子节点都是“人类不会去尝试的节点”，即选择自杀或提走对方的子。叶子节点的数量相当多（完全博弈树的最后一层都是这种节点），扩展出这种子节点占用了大量的时间和空间，拉低了 UCT 算法的运行效率。因此，我设计了一个类(ValidCounter)用于在 $O(n^2)$的时间范围内计算一个已知节点的所有非叶子节点的儿子(n=9，为棋盘大小)，计算思路如下：

叶子节点分两类：

(1) 下这一手是自杀

a) 这个点是对方的眼

b) 这个点是自己的唯一气，且这个点周围没有空格，且这个点不是自己的非唯一气

(2) 下这一手提走了对方的子，即这个点是对方的唯一气

这样，对棋局进行一次深度优先搜索，找到每个白块和每个黑块，记录下每一个点是否是自己的唯一气、自己的非唯一气、周围是否有空格、是否是对方的眼、是否是对方的唯一气，就可以 O(1) 地判断它是否为不可落子点。去掉不可落子点，对剩下的点进行随机排序，即可作为该节点在博弈树中的儿子集合。

2.3. 对模拟策略的优化

在 UCT 算法中，模拟是相当消耗时间的。而且由于落子的随机性，模拟的结果很可能不尽如人意。在学习了 ID“冷冻章鱼”的开源代码后，我发现可以基于不围棋“让对手可落子的点尽可能少，让自己可落子点尽可能多”的经典策略，设计这样的模拟方案：

$$simulate(x) = \begin{cases} 1 & v = 0 \\ -1 & u = 0 \\ (u-v)/(u+v) & else \end{cases}$$

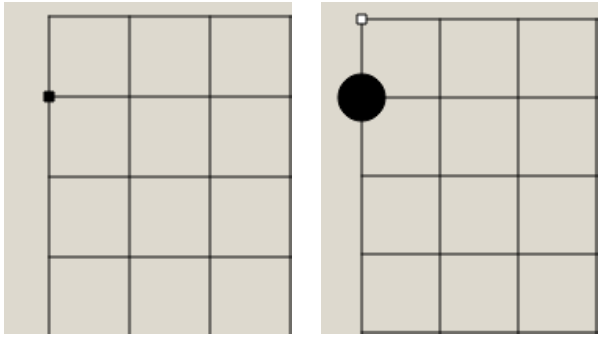
其中，u 表示该局面下己方可落子点数量，v 表示该局面下对方可落子点数量。这种模拟策略在用 2.2. 的方法初始化 u 和 v 后是可以 O(1) 完成的，并且在造眼、破眼上有着相当不错的能力。

3. 图形界面与功能设计

在图形界面的设计上，我选择了 Qt。我为每一个界面设计了一个类，还设计了按钮类、裁判类用于保存游戏进度以及裁判是否为终局、ai 玩家类用于使用 UCT 算法对给出的棋局给出最好选择。最后，通过 Qt 特有的信号和槽系统将它们连接起来，以实现需要的功能。

3.1. 落子标记

通过监测鼠标运动，实时更新位置，绘制落子标记



3.2. 按钮跳动特效

```
void MyPushButton::zoomDn(int delta)
{
    QPropertyAnimation *animation=new QPropertyAnimation(this,"geometry");
    animation->setDuration(200);

    animation->setStartValue(QRect(this->x(),this->y(),
                                   this->width(),this->height()));
    animation->setEndValue(QRect(this->x(),this->y()+delta,
                                 this->width(),this->height()));

    animation->setEasingCurve(QEasingCurve::OutBounce);
    animation->start();
}
```

我使用 QPropertyAnimation 中的动画，对按钮进行向下平移然后向上平移实现了按钮跳动特效。

3.3. 暂停功能

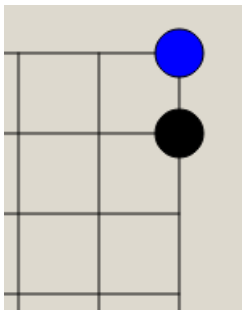
在按下 pause 按钮后，使用全局变量禁用了用户鼠标点击下子这个函数的调用。

3.4. 存档读档功能

在按下 save 按钮后，使用 QFile 将当前棋局输出至文件 SaveBoard.log 中。按下继续游戏按钮后，使用 QFile 从这个文件中读取棋盘，如果不存在该文件则使用 QMessageBox 提示玩家没有存档文件。

3.5. 提示功能

将当前局面传给 bot 类，计算出最推荐的落子点，并使用蓝色标出。



3.6. 成就功能

在胜利/失败后，使用 QFile 输出 Achievement.log 中的成就数据，并在成就界面使用 QFile 输入成就数据，用 QTableWidget 显示。

4. 总结

改进的 UCT 算法实现后在 botzone 中取得了较优的天梯排名，使用 Qt 实现的图形界面可以正常编译运行，达到了预期的效果。