人 工 智 能 导 论

重 力 四 子 棋

实 验 报 告

计 86 罗境佳 2018013469

1. 算法原理

本次四子棋大作业主要采用了信心上限树算法（以下简称 UCT）实现，针对当前的棋局选择最佳的落子点。

算法的大致过程为，以当前状态为根，根据根结点中所存的 top 数组（即下一局对手可以选择走的地方）逐层扩展，每新扩展一个结点，便随机落子进行到最终状态（即可以分出胜负）作为结点的初始评分。当一层被扩展完时，选择该层中得分最高的结点继续扩展下一层，隐式进行了 alpha-beta 剪枝。在时间限制内，尽可能多扩展结点，最终选择根节点的得分最高的子结点作为本局落子进行返回。

其中结点 v 的得分计算方式如下：

$$\frac{Q(v)}{N(v)} + c_1 * \sqrt{\frac{2 * \ln(\mathrm{N(parent(v))} + 1.001)}{\mathrm{N(v)}}} + c_2 * \frac{count\big(enemy\_\mathrm{around}(v)\big)}{log(N(v) + 1.001)}$$

$Q(v)$表示在实验中结点 v 的子树的得分（胜为 1，平为 0，负为-1），$N(v)$表示结点 v 子树的总试验次数，$enemy_\mathrm{around}(v)$表示与结点 v 相连的敌人数量，为在试验初期将棋子向阻断敌方的位置引导所设。

2. 算法实现

算法实现大致与书中介绍的 UCT 算法伪代码相同。

2.1 class UcTree

UCT 的实现类。

```
TreeNode *treePolicy()
    p = root;
    while (!p->isTerminal)
        if (p 的子结点未全部扩展)
            return p->expand()
        else
            p = p->bestChild()
```

```
Point uctSearch()
    while (used_time < TIME_LIMIT)
        v = treePolicy()
        delta = v->randomExpand()
        v->backup(delta)
    return root->bestChild()
```

2.2 class TreeNode

UCT 的结点类。

```
bool isTerminal()
    if (USER && userWin()) return STATUS_WIN
    if (MACHINE && machineWin()) return STATUS_WIN
    if (isTie()) return STATUS_TIE

TreeNode *bestChild()
    for (c in children)
        if (c->detectGoldenChance()) return c
    for (c in children)
        if (c->detectDeathTrap() == -1) return c
        记录暂时得分最高的子结点
    返回 children 中得分最高者

TreeNode *expand()
    if (有未扩展的 top 状态)
        p = new TreeNode(该 top 状态)
        将 p 与 this 连接
        return p
    else (所有状态都已扩展完)
        return nullptr

int randomExpand()
    if (该结点胜负已定) return DELTA
    s[][] = state, t[] = top
    while (未决出胜负)
        next_act = t[random() % N]
        s = nect_act(s), t = next_act(s)
    if (赢) return 1
    if (平局) return 0
```

```
void backup(delta)
    while (指针 p 没有到达根结点)
        Q += delta, N += 1
        delta = 0 - delta
        if (delta > 1) delta = delta / 10
        p = p->parent

int detectDeathTrap()
    if (该位置是敌方棋子地方会赢) return -1
    else return count(enemy)

int detectGoldenChance()
    return 若该位置是我方棋子我方会赢
```

3. 实验结果

3.1 本地结果

id	wina	winb	losea	loseb	tie
2018013469	44	43	2	2	1

结果	胜	负	平
比例	0.92	0.04	0.01

3.2 Saiblo 平台结果

Saiblo 平台账号：jingjia

AI：？？？(ver. 7)

批量测试编号：#1668

结果	胜	负	平
比例	0.92	0.08	0.00

4. 创新与思考

4.1 对必胜位置和必输位置的预先判断

在实验初期发现四子棋对于“必胜位置”（即将自己的棋子下在该位置能够连成 4 个）与“必输位置”（若将该位置让给敌方则敌方能够连成 4 个）并不敏感。于是在 bestChild() 函数中增加了必胜位置与必输位置的判断，即先对每个 top 中的状态进行 detectGoldenChance()和 detectDeathTrap()的检测，若符合这两种情况则立刻返回。这样可以避免被敌方一招将死。

除此之外，为了将棋子引导到能够阻断敌方棋子的位置，而非棋盘中很偏远的地方，在 bestChild() 函数中增设了 count(enemy_around(v)) 的指标，该计算也由 detectDeathTrap()函数完成。而在模拟次数足够多时，胜率则更占上风。

4.2 通过增大 delta 提高对最近几步的模拟准确度

在实验过程中，常常发现“白给”的状况。例如敌方棋子已练成斜着的 3 个，而我方棋子给敌方斜着的第 4 个棋子垫了位置，帮助敌方赢得了棋局。我实现的 UCT 以敌方下完子后的状态为根结点，自己下一步的出棋为根的子结点，而上文所说的胜局即为根的孙子结点。思考后，我认为一个结点应该对附近的后代结点的胜负采取更大的权重，反映在本结点的胜率上。

于是，对于树上的已取得胜负的叶子结点（该胜负情况被结点本身的状态包含，而非随机模拟取得的结果），由 randomExpand()返回一个很大的 DELTA（设为 10000），这样能够较大程度影响附近结点的胜率。而在 backup()的过程中，该 delta 等比减小，每次变为之前的-1/10，防止结点胜率受较远结点的胜负的过度影响。在 backup()的过程中，|delta|收敛到 1 则停止。

5. 总结与收获

在做四子棋大作业的过程中，对原先一知半解的 UCT 算法更加熟悉了，也想出了一些优化方法，并且复习了 OOP 等基础知识，在疯狂 debug 的过程中也对指针、内存等等的理解加深了，收获很大～