重力四子棋游戏小程序

游戏规则介绍：

重力四子棋，是五子棋的一种变体；首先，正如其名字，它只要求一方按横、竖、斜连成四子即为获胜，棋盘落满子而无人获胜为平局；其次，每次落子必须落在每一列最低的未落子点上，故名重力四子棋。

由于原始的重力四子棋存在必胜方法，因此我在这里实现的是一种改动规则的重力四子棋，它对规则的改动有两点：

1、 棋盘大小随机。棋盘的长与宽均在【9，12】范围内随机，因此有 4 * 4 = 16 种棋盘大小；
2、 不可落子点。为防止随机到的棋盘产生必胜法，程序随机在棋盘上产生一个不可落子点。

需求分析：

1、 通过命令行输入落子点坐标，实现人-人对战与人-机对战，能够合理判定胜负和平局；
2、 利用蒙特卡洛投点法和信心上限树算法实现一个 AI，并通过允许玩家调整 AI 每步计算时间的方式调整 AI 强弱。
3、 允许玩家使用初始的随机设置进入游戏，也允许使用自定义设置（棋盘大小、先手后手等）进入游戏。

总体设计：

系统主要功能分为三块：输入、处理和输出。

输入方面，程序为了让用户在输入错误时程序不至于崩溃，且能流畅的修改游戏设置，设计了 setting 和 valid_input 函数；输出方面，系统根据不同的输出需求（棋盘的不同表达形式，输出流的不同等）重载了多个 output 函数；处理方面，程序中设计了 new_top 函数和 judge 系列函数来完成胜负判定和棋盘更新。

此外，我学习了蒙特卡洛法和信心上限树的相关知识，通过设计 node 系列函数和 getpoint 函数，完成了一个可以在规定时间内给出落子选择，且有一定棋力的 AI，并且通过修改规定的思考时间完成了 AI 强弱的调整。

为了满足设计基本要求，本程序设利用函数重载机制，重载了 output 函数；在新建棋盘时使用了动态内存分配，输出比赛结果时使用了文件操作；在存储历史落子点坐标 lastX，lastY 时采用了栈的数据结构。

详细设计：

在程序的一开始，考虑到我的游戏的自创性，我为玩家提供了游戏规则介绍的选项，同时也提供了修改游戏设置的功能。

在游戏规则介绍页面，我直接输出了预设的游戏规则文本；在修改游戏设置页面，我使得玩家能够按自己的意愿决定修改设置的先后顺序，并且为误操作做了一定的鲁棒性处理。

进入游戏后，我为玩家提供了单机与热座两种对战模式，并对棋盘的输出格式进行了一定处理，使得输出的棋盘尽量清晰，避免因为看不清棋盘产生误操作。

游戏结束后，程序会自动将结束时的棋盘输出至 output.txt。

以下是结合蒙特卡洛树的信心上限树算法的伪代码，这也是我实现程序中 AI 的基础。

算法 3：信心上限树算法（UCT）

function UCTSEARCH(s_0)
以状态s_0创建根节点v_0;
while 尚未用完计算时长 **do**:
v_l ←TREEPOLICY(v_0);
Δ←DEFAULTPOLICY(s(v_l));
BACKUP(v_l, Δ);
end while
return a(BESTCHILD(v_0, 0));

function TREEPOLICY(v)
while 节点v不是终止节点 **do**:
if 节点v是可扩展的 **then**:
return EXPAND(v)
else:
v ← BESTCHILD(v, c)
return v

function EXPAND(v)
选择行动$a \in A(state(v))$中尚未选择过的行动
向节点v添加子节点v'，使得$s(v')$= $f(s(v), a), a(v') = a$
return v'

function BESTCHILD(v, c)
return $argmax_{v' \in children\ of\ v}\left(\frac{Q(v')}{N(v')} + c\sqrt{\frac{2ln(N(v))}{N(v')}}\right)$

function DEFAULTPOLICY(s)
while s不是终止状态 **do**:
以等概率选择行动$a \in A(s)$
$s \leftarrow f(s, a)$
return 状态s的收益

function BACKUP(v, Δ)
while $v \neq NULL$ **do**:
$N(v) \leftarrow N(v) + 1$
$Q(v) \leftarrow Q(v) + \Delta$
Δ← −Δ
$v \leftarrow v$的父节点

系统调试：

在本程序的编写过程中，棋盘部分的调试过程相对容易，大部分精力都用在了调整输出格式和提高输入鲁棒性上。

而在 AI 部分，AI 本身的调试和性能的测试都消耗了不少时间。而且我发现原始的信心上限树算法有时并不能取得很好的效果，还容易犯一些低级错误；因此在这一算法的基础上，我基于重力四子棋的特性，对算法做了一些改进：

DefaultPolicy 函数中选择行动 a 时，不是简单的等概率选择，而是从左向右检查是否有落子即获胜的点，若有即落子；再检查是否有对方落子我方即失败的点，若有即落子占掉这个位置；如果前两种情况都不出现才会以等概率选择行动。

经过这一改进，我完成的 AI 获得了一定的布局能力，能够战胜绝大多数没有接触过重力四子棋的同学了。

结果分析：

样例数据已经包含在含有生成的 exe 文件的文件夹中，依照系统内置的操作指南进行操作即可进行测试。

总结：

这次作业，让我有以下几点感悟：

第一，人机交互（Human-Computer Interaction, HCI）确实是一个重要的领域，没有优秀的 HCI 设计，再好的灵感、再有趣的游戏也会因为难以操作的系统和鲁棒性不够导致的连续崩溃而被用户放弃。

第二，不能迷信 AI。AI 是我们设计出来，试图让计算器模拟人类的一些算法；但事实上，很多算法都有其问题所在，我们不能本着对 AI 的迷信奉行拿来主义，而要根据自身的需求做取舍和优化。