算法基本思路与实现方法：

我编写的算法核心是信心上限树算法，具体上基本依照下图中的伪代码进行实现（来源为网络学堂发布的课件）；

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

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。
同时，在树搜索整个过程之前先依次检查当前局势是否有满足这两个条件的点，若有则直接落子，不进行树搜索。
这一改进加快了算法收敛的过程，相对于初始版本有效地提高了棋力。