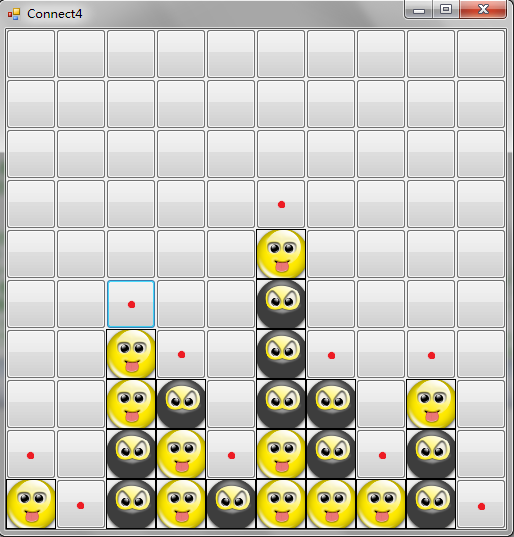
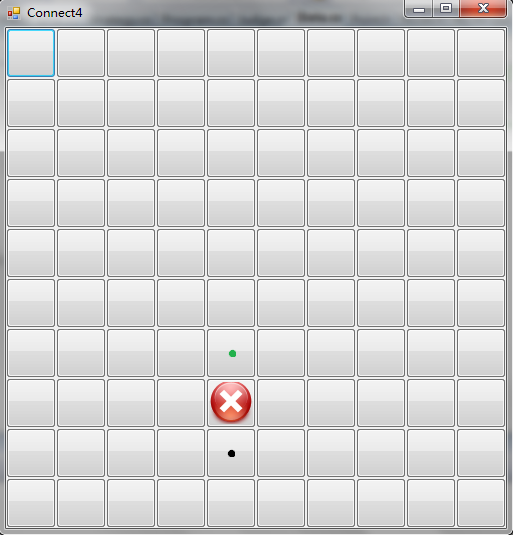
**四子棋实验报告**

黄 欢

（计算机科学与技术系 计33 2013011331）

1. **题目概述**

四子棋，即游戏双方分别持不同颜色的棋子，设A持白子，B持黑子，以某一方为先手依次落子。假设A为先手，落子规则如下：在M行N列的棋盘中，棋手每次只能在每一列当前的最底部落子，如图中的红点所示，如果某一列已经落满，则不能在该列中落子。在图形界面中，如果在某一列的任意一个按钮上点击，会自动在该列的最低端落子。棋手的目标是在横向、纵向、两个斜向共四个方向中的任何一个方向上，使自己的棋子连成四个（或四个以上），并阻止对方达到同样的企图。先形成四连子的一方获胜，如果直到棋盘落满双方都没能达到目标，则为平局。

本次实验棋盘的大小是随机的，而不是固定不变的（宽度和高度的范围均为[9,12]）。且每次棋盘生成之后，会同时在棋盘上随机生成一个不可以落子的位置，如图中的红叉所示。

1. **算法思路及方法**

本次实验中我采用了将信心上限（UCB1）算法思想用于蒙特卡罗规划（Monte-Carlo Tree Search）的信心上限树（UCT）算法，兼顾探索和利用，在有限的计算时间里，对较好的节点更深入地进行探索以保证选择更接近最优解。

在该算法中，每一个节点包含四项基本信息，分别为其所对应的状态，所对应的来自父节点的行为，随机模拟收益（例如获胜次数），以及节点的被访问次数。

**伪代码如下：**

算法3：信心上限树算法（UCT）

**function** UctSearch()

以状态创建根节点;

**while** 尚未用完计算时长 **do**:

TreePolicy();

DefaultPolicy(s());

Backup();

**end while**

**return** (BestChild());

**function** TreePolicy()

**while** 节点不是终止节点 **do**:

**if** 节点是可扩展的 **then**:

**return** Expand()

**else**:

BestChild(

**return**

**function** Expand()

选择行动中尚未选择过的行动

向节点添加子节点，使得=

**return**

**function** BestChild(

**return**

**function** DefaultPolicy()

**while** 不是终止状态 **do**:

以等概率选择行动

**return** 状态的收益

**function** Backup(

**while** **do**:

的父节点

在本实验中，在每一回合收到getPoint指令后，程序建立一个根节点0表示当前局面。每个节点连向孩子的边代表处于该节点表示的局面下可以选择的着法，而相应的孩子就是选择该着法后转移到的局面。树一开始只包含根节点，在之后的expand过程中节点会被添加到树中。

然后执行若干次迭代过程，每次迭代都会依次执行treePolicy、expand、defaultPolicy和backup四个过程。

* treePolicy。从根出发，根据tree policy走到一个孩子节点，重复此过程直到碰到一个可扩展的节点为止。可扩展的节点指的是带有未访问孩子的节点。
* expand。此时我们已经走到了一个可扩展节点。根据tree policy选择未探索过的着法中最有探索价值的，给当前节点添加一个孩子并走到这个孩子节点。
* defaultPolicy。根据default policy在当前节点表示棋盘下模拟双方对弈。
* backup。根据对弈的结果更新当前节点及其所有祖先的统计信息。

**算法具体流程如下：**

1. 初始化棋局，以0为父节点nf，扩展第一层子节点，每个节点代表一个可行位。
2. 对于nf的每个子节点ni，用UCB公式算出其UCB值，选取UCB值最大的节点nl。
3. 如果nl不是叶子节点，那么以它为父节点nf，转2。
4. 如果nl是叶子节点，那么将它进行模拟对局，向上更新每一个节点的赢盘值win和总盘值tot；若游戏还没有结束，那么为nl扩展子节点。
5. 如果不满足终止条件（本程序中为经过的时间小于4.5s），那么以0为父节点，转2。
6. 在以0为父节点的子节点，选取胜率最高的节点，作为落子点。
7. **程序实现**
8. **定义结构体Node**

每个节点都对应一个棋盘，记录当前的博弈状况。

|  |  |
| --- | --- |
| Int mlchild, mrchild | 记录最左孩子和最右孩子的下标，初始构造为“-1”。 |
| Int father | 记录父节点的下标，初始构造为“-1”。 |
| Int lastx, lasty | 记录要走到该节点，其父节点的落子位置，初始构造为“-1”。 |
| Int flag | 记录该节点的状态：  -1表示未拓展；  0表示已拓展；  1表示程序失败，游戏结束；  2表示程序胜利，游戏结束；  3表示打成平局，游戏结束。  初始构造为“-1”。 |
| Bool user | 记录下棋方，true表示本程序行走，false表示对方行走，初始构造为“false”。 |
| Int tot | 记录总盘数，初始构造为“0”。 |
| Int win | 记录胜盘数，初始构造为“0”。 |

由于创建指针和释放内存十分耗时，所以在本实验中采用了建立Node[MAXTREE]数组来模拟树状结构，再通过Node类中的参数来进行层间控制。既避免了链表创建和销毁时的消耗，又保证了路径的连续性。此外，本实验中一些通用的参数设为全局变量，以方便调用，提高运行效率。

1. **创建根节点**

函数getPoint（）开始的棋盘状态board为根节点，下标为0，由构造函数进行初始化。

1. **对博弈树不断进行拓展**

当拓展不超过4500ms时，继续拓展，并且注意处理好不可落子点（noX，noY）。拓展过程中的每次迭代都会依次执行treePolicy、expand、defaultPolicy和backup四个过程。

从根出发，根据tree policy走到一个孩子节点，重复此过程直到碰到一个可扩展的节点为止。博弈树内部节点为已拓展节点，flag值必为0；可拓展节点的flag值为-1.并且在查找和拓展过程中，更新新的棋盘（nowBoard）。根据tree policy选择未探索过的着法中最有探索价值的，给当前可扩展节点添加一个孩子并走到这个孩子节点。根据default policy在当前节点表示棋盘下模拟双方对弈，调用userWin和machineWin，若游戏结束，更新flag值；若游戏尚未结束，则继续对当前节点进行扩展。最终根据对弈的结果更新当前节点及其所有祖先的统计信息，并通过“悔棋”恢复棋盘，继续下一轮拓展。

决定最有探索价值的孩子节点时，只需确定孩子节点的UCB值，即为估价函数，具体为

右式中，第一项表示其父节点落子所得到的胜率，第二项是为了维持博弈树的平衡性。这也正是蒙特卡洛方法优化作用所体现的地方。在实现过程中，为了防止分母为零的情况，将分母部分再加上epsilon值。

1. **选取最优落子点**

在根节点中找出 最大的孩子，其记录的父亲的落子点（lastx，lasty）即为根节点下一步的落子位置（x，y）。

1. **测评结果**

从实验效果上看，效果没有预期好，可能是由于内存的原因，当然最主要的是程序实现没有那么强大，算法本身还是很强大的。测评结果采样次数比较少，如果要有更准确的数据，需要更多的数据，结果还与系统的运行状况等诸多因素相关。有时候也会报错，已经连续调试了好多天，或许还是有些问题。具体评测结果如下：

|  |  |
| --- | --- |
| 对战方 | 我方胜率（采样次数较少） |
| 2.dll | 1.00 |
| 10.dll | 1.00 |
| 50.dll | 1.00 |
| 60.dll | 0.55 |
| 80.dll | 0.55 |
| 90.dll | 0.53 |
| 100.dll | 0.10 |

在同学的推荐下，我学习并采用了蒙特卡洛算法，该算法更为精妙，也更高效，代码量也精简许多。如果在本试验中采用“位运算”，重复利用之前的结果，还可以进一步优化。如果时间充裕，相信一定能实现得更好。

通过本次实验，我看到了算法的精妙，十分有趣，也锻炼了代码能力，虽然没有那么强大，却也进步不少。在未来的学习中，我将在人工智能方面更深地探索。

感谢老师和助教的指导与付出！