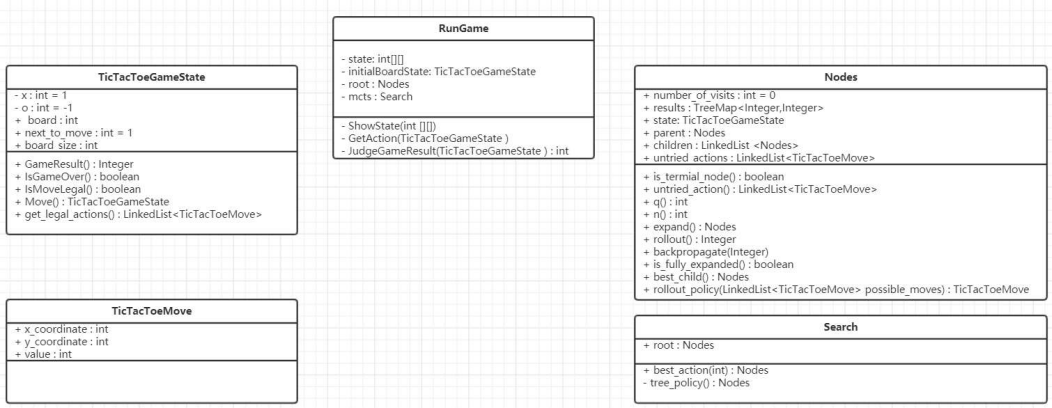
人工智能大作业-蒙特卡洛树搜索树实现三子棋游戏-代码理解及其实现

笔记本： 我的第一个笔记本
创建时间： 2019/6/5 10:13 **更新时间：** 2019/6/7 15:02
作者： 1735512161@qq.com

人工智能大作业-蒙特卡洛树搜索树实现三子棋游戏-代码理解及其实现

程序的UML图:



代码理解:

RunGame.java:

属性:
用一个`int[][]` 表示棋盘、

`new` 一个初始棋盘对象、

`new` 一个`mcts` 树根节点、通过根节点`new` 一个`mcts` 树

方法:
`void ShowState(int[][] board)`：打印出棋盘的状态

`TicTacToeMove GetAction(TicTacToeGameState state)`：输入玩家的移动并判断是否合法

`int JudgeGameResult(TicTacToeGameState state)`：判断游戏结果

`main()`：
首先直接调用`mcts`树的`best_action(1000)`，模拟`1000`次落子(保证了效率和落子的准确性)，再打印出电脑落子以后棋盘的状态，然后就是一个`while`循环，玩家和电脑依次落子，每层循环最后都要判断是否游戏结束，不结束就继续下棋，结束就退出程序。

TicTacToe.java:

该文件实现了两个类:

`TicTacToeMove`类:
用于表示落子的情况(x坐标、y坐标、value-标识下棋对象)

`TicTacToeGameState`类:

属性:
静态常量:`x = 1`、`o = -1`，分别标识电脑落子和玩家的落子

`board`：棋盘的状态

`board_size`：棋盘的大小

`next_to_move`：标识下一步要走棋的对象(电脑->玩家、玩家-电脑)

方法:
`Integer GameResult()`：判断游戏结果，也就是判断对角线(左下->右上+左上->右下)，三个横起的，三个竖起的，加起来=3，表示电脑赢，=-3，表示玩家赢，如果既有3也有-3就表示平局，如果不是上述情况则表示游戏还未结束，返回null。

`boolean IsGameOver()`：判断游戏是否结束，调用上述函数即可判断。

`boolean IsMoveLegal(TicTacToeMove move)`：判断一次移动(传入一个`TicTacToeMove`对象)是否合法。

`TicTacToeGameState Move(TicTacToeMove move)`：移动棋子(传入一个`TicTacToeMove`对象)，首先判断移动是否合法，如果不合法直接抛出自定义异常，如果合法就给board数组在(move.x,move.y)位置上赋上move.value,
接着讲next_to_move改值(1->-1、-1->1)，然后返回当前的棋盘状态(`TicTacToeGameState`对象)。

`LinkedList<TicTacToeMove> get_legal_actions()`：得到合法动作集合，也就是遍历整个棋盘找出所有board=0的位置，作为move对象放入到LinkedList<TicTacToeMove>集合中并返回。

Nodes.java:

属性:
`number_of_visits`：节点被访问的次数(用于计算n)

results：这是一个用TreeMap实现的映射关系，key: -1、1 value: 不太清楚(用于计算q)

parent：表示该节点的前置节点

childern：表示该节点的已访问子节点，用LinkedList <Nodes> 来实现

untried_actions：表示该节点的子节点中没有被访问的节点，用LinkedList<TicTacToeMove> 来实现

state：表示该节点对应的棋盘状态

方法:

boolean is_termial_node()：判断该节点是否为终端节点，直接调用此时state.GameOver ()方法即可

LinkedList<TicTacToeMove> untried_action()：用于得到该节点的untried_actions，直接调用get_legal_actions() 返回一个LinkedList<TicTacToeMove>

q()：计算节点的q值，为后面的UCT函数作准备，节点的q值也即总模拟奖励，计算方式为: 如果该节点为电脑走的，即this.parent.state.next_to_move=1，所以wins对应的就是这个值，-1对应的就是loses的值，两者相减就是总模拟奖励，玩家走子同理。

n()：计算节点的n值，也是为后面的UCT函数作准备，就是直接返回节点的number_of_visits属性

Nodes expand()：选择一个该节点的未被访问过的节点(untried_actions)，进行一次移动(state.Move(action))，返回一个状态，然后加入到已访问节点中(children.add(child_node))

Integer rollout()：Rollout策略函数，根据节点的状态来进行移动: 也就是调用state.get_legal_actions()得到一个可能的移动对象集合，然后调用rollout_policy()随机得到一个移动move对象，最后调用Move()得到移动之后的状态，再返回判断游戏的结果。

void backpropagate(Integer reward)：反向传播，不断更新上面的节点的访问次数n，和results中的q值，当然n值就是number_of_visits，每次访问一次+1，q值就是由results映射表得到的，所以这里也在更新results中的值，每访问一次，以reward为键值，对应的value+1。

boolean is_fully_expanded()：判断该节点是否为完全展开节点，直接判断未访问节点是否为0即可

Nodes best_child()：置信上限函数UCT的应用处，得到节点的q和n，然后利用公式计算出UCT最大的节点返回，该节点即为当前节点要往下走的最优节点，不过我写的java实现在此处有bug。

TicTacToeMove rollout_policy(LinkedList<TicTacToeMove> possible_moves)：在可能的移动中(possible_moves 为rollout() 方法传进来的参数)，随机的返回一个TicTacToeMove 对象。

Search.java:

属性:
`root`：表示这颗mcts树的根节点

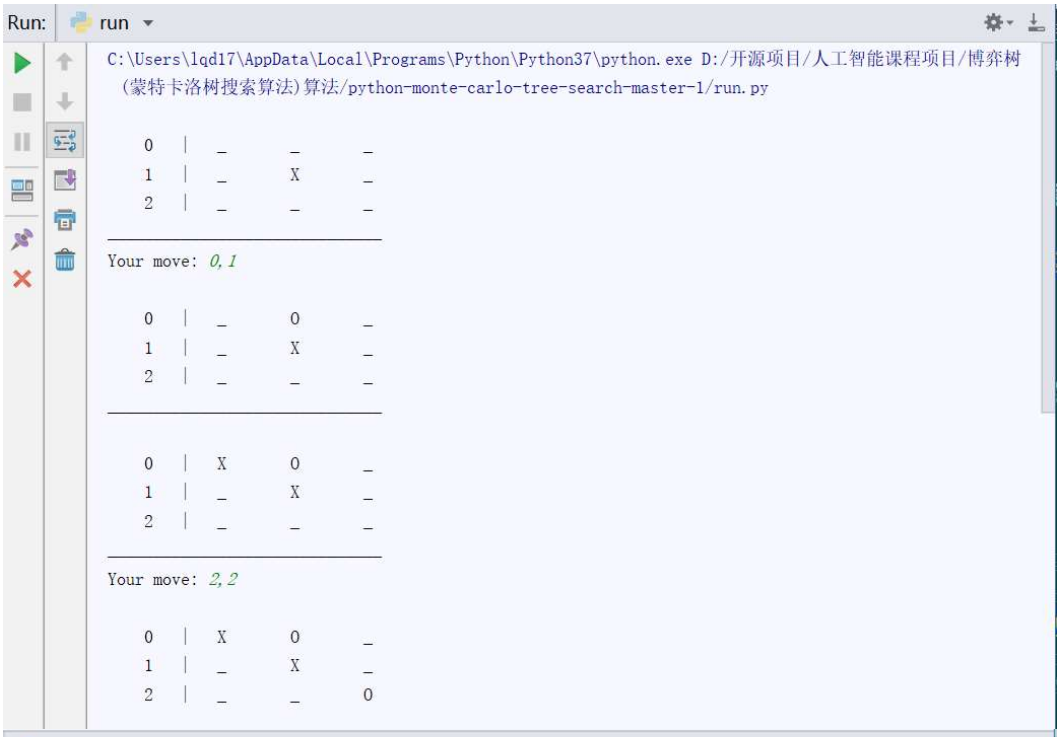
方法:
`Nodes best_action(int simulations_number)`：找出最优落子，传进来的是模拟次数，太大了会使得计算时间过长，太小了又不能保证落子的最优，所以要根据实际情况来选择最佳的模拟次数，每次模拟都会通过`tree_policy()` 函数得到一个最优的起始节点，然后会调用`rollout()` 函数得到一个可能的移动集合并选择一个进行移动，最后返回一个移动后的游戏结果，然后会调用`backpropagate()` 函数，从该节点开始往上回溯，当经过了指定的模拟次数后，会直接调用`this.root.best_child()` 函数来得到最优落子。

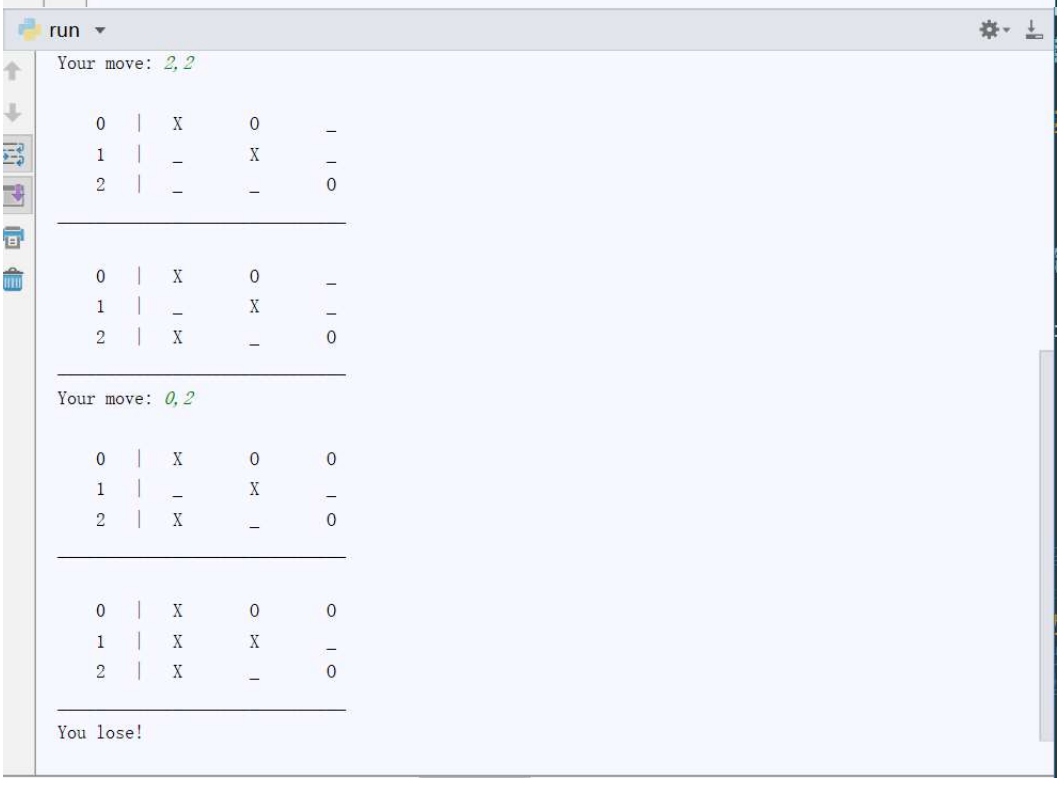
`Nodes tree_policy()`：从这颗mcts树的根节点开始开一个循环，直到遍历到终端节点才退出，如果节点没有被完全扩展，就调用当前节点的expend()函数不断扩展，当此节点完全扩展开了，就调用当前节点的best_child()函数得到最优子节点，并返回到best_action()函数。

代码实现:

我的代码实现

运行结果:





注释:

> 该项目的代码实现是根据网上一位博主的代码来完成的。
> 他是根据官方代码来改写实现的，而我是在他的基础上用java替代了py，但是在编写的过程中，由于python的语法不熟悉，导致有一些bug还没有解决。