人工智能导论
四子棋作业报告

齐强 2017011436 计 75

2019 年 5 月 13 日

目录

1 代码结构

1. Uct.h：包含 State Uct 两个类

```
class Uct{
+-- 26 lines: public:-------------------------------------------------------------
};

// random drop one chess
void Uct::playPolicy(State& node){
+--  9 lines: node.ly = rand() % node.col;---------------------------------------
}

// calculate last down chess influence to the State
int Uct::profitPolicy(State& node){
+-- 13 lines: if(node.lx == -1 && node.ly == -1)-----------------------------------
}

// simulate one time to the expanded node
int Uct::simulatePolicy(State* node){
+--  9 lines: copy a new-State to operate , avoid influence the origin node------
}

void Uct::backupPolicy(State* node, int profit){
+--  5 lines: while(node){-------------------------------------------------------
}

State* Uct::expandPolicy(State* node)const{
+-- 10 lines: while(!node->isEnd()){-------------------------------------------
}

State* Uct::searchPolicy(vector<int> tp, vector<vector<int>> bd){
+-- 12 lines: root = new State(row, col, nx, ny, lx, ly, tp, bd);--------------
```

```
#define TIME_LIMIT 0.5
#define NOTEND 2
#define RADIO 1.5

class State{
    public:
+--- 23 lines: transfer in during construct-----------------------------------

    public:
+--- 33 lines: State(int row_, int col_, int nx_, int ny_, int lx_, int ly
};

bool State::isEnd(){
+--  7 lines: if(lx == - 1 && ly == -1)-------------------------------------
}

void State::clear(){
+--  3 lines: for(int i = 0;i < col;i ++)-----------------------------------
}

bool State::isExpandable(){
+--  5 lines: for(int i = 0;i < col;i ++){----------------------------------
}

// most visit child
State* State::mvc(){
+-- 11 lines: State* res = NULL;-----------------------------------------
}

// highest profit child
State* State::hpc(){
+-- 20 lines: State* res = NULL;-----------------------------------------
}

State* State::expand(){
+-- 31 lines: for(int i = 0;i < col;i ++){-------------------------------
}

// state debug info output
void State::debug() {
+-- 44 lines: printf("current State:\n");-------------------------------
}
```

2. Judge.h Judge.cpp：在给定代码基础上做了简单修改

2 算法思路

1. 经典“蒙特卡洛方法”+ “信心上限树“算法
2. 通过构建构建 UCT 树、随机模拟对战过程、计算收益, 从而确定最佳落子点
3. 选择模拟对战节点方法：
   (a) 从根出发, 若当前点可以扩展新的子节点, 则扩展并选中新扩展的点;
   (b) 若当前点无法扩展新的子节点, 则选中以当前点为根的子树中 profitRadio 最高的叶节点
   (c) profitRadio 计算规则 $\frac{Q(v)}{N(v)} + c\sqrt{\frac{2\ln(N(u))}{N(v)}}$
   (d) u 为当前节点,v 为考察的子节点。N (v) 为 v 被回溯的次数,Q(v) 为 v 回溯的总得分

3 优化尝试

1. 对于常数 C 的调节：
   进行了几次实验，通过调节 C 的大小，根据胜率进行了几次二分实验，最后确定大概 C 为 0.5 时，效果比较好
2. 对于增加模拟次数的尝试：
   调高模拟次数，能够显著地提高获胜概率，所做的尝试有
   - 将 vector 改为数组（失败，代码改动较多，未能及时完成）
   - 将二维 board vector 改为一维 vector（失败）
   - 改变时间判断方法，由每模拟一次判断一次时间，改为模拟 1000 次判断一次时间，能够增加一些模拟次数

4 测试效果

写了一些自动测试脚本，调用 compete.exe

对于前 50 个 TestCase dll，胜率基本 >0.8 并接近于 1

下面是对于 90-100 dll 的结果记录

dll	90	92	94	96	98	100
C = 0.5	0.4	0.5	0.2	0.2	0.3	0.2
C = 0.8	0.5	0.8	0.7	0.2	0.7	0.2
C = 1.5	0.4	0.5	0.7	0.8	0.3	0.2

5 遇到困难与解决方法

1. 对于空间问题的解决
2. 对于 expand 的正确逻辑实现出现了很多问题
3. 对于很多玄学 bug，各种卡掉....

6 总结收获

本次完成大作业的过程中, 学习到了蒙特卡洛模拟方法、信心上限树算法学习了更多的 debug 的技巧...