作业1: Bait游戏 实验报告

151220129 计科 吴政亿

任务一 深度优先搜索

变量简介

变量类型	变量名	变量含义
ArrayList	closeList	存储已经走过的历史路径
boolean	isCalculated	是否得到了能走到终点的答案
ArrayList<Types.ACTIONS>	depthFirstAction	存储dfs中的每一步动作
int	nowStep	当前步骤在depthFirstAction的下标

函数简介

返回类型	函数名	传入参数	函数功能
boolean	isInCloseList	StateObservation obs	检测是否在历史状态中
boolean	getDepthFirst	StateObservation stateObs, ElapsedCpuTimer elapsedTimer	计算从stateObs出发的深度优先路径，如果找到则返回true
Types.ACTIONS	act	StateObservation stateObs, ElapsedCpuTimer elapsedTimer	根据局面stateObs调用getDepthFirst并返回当轮动作
void	debugPrint	Types.ACTIONS act	输出act动作信息
void	debugPrintAllAction	ArrayList<Types.ACTIONS> actions	输入actions中所有动作信息

核心代码

```
boolean getDepthFirst(StateObservation stateObs, ElapsedCpuTimer elapsedTimer){
    if(stateObs in closeList)
        return false;
    else
        closeList.add(stateObs);
        stCopy = stateObs.copy();
    for(all actions in stateObs){
        try this action in stCopy;
        depthFirstAction.add(action);
        if(win)
            return true; // all actions are in 'depthFirstAction'
        else if(stateObs in closeList || Game over){
            stCopy = stateObs.copy(); // reset stCopy
            depthFirstAction.delete(action);
            continue;
        }
        else{ // a new state
            if(getDepthFirst(stCopy,elapsedTimer))
                return true;
            else {
                stCopy = stateObs.copy(); // reset stCopy
                depthFirstAction.delete(action);
                continue;
            }
```

```
        }
    }
    return false;
}
```

任务二 深度受限的深度优先搜索

变量简介

变量类型	变量名	变量含义
ArrayList	closeList	存储已经走过的历史路径
ArrayList	stateDepth	历史路径里对应的每一个的深度
ArrayList<Types.ACTIONS>	limitDepthFirstAction	存储limitdfs中的每一步动作
double	bestCost	精灵与目标在state中最优状态的距离
Vector2d	goalpos	门的位置
Vector2d	keypos	钥匙的位置

函数简介

返回类型	函数名	传入参数	函数功能
void	initAgent	null	初始化Agent
boolean	isInCloseList	StateObservation obs	检测是否在历史状态中
void	limitDepthFirst	StateObservation stateObs, ElapsedCpuTimer elapsedTimer, int depth	计算从stateObs出发的受限层数为depth的深度优先路径
Types.ACTIONS	act	StateObservation stateObs, ElapsedCpuTimer elapsedTimer	根据局面stateObs调用limitDepthFirst并返回当轮动作
void	debugPrint	Types.ACTIONS act	输出act动作信息
void	debugPrintAllAction	ArrayList<Types.ACTIONS> actions	输入actions中所有动作信息
double	getDistance	Vector2d vec1, Vector2d vec2	返回vec1与vec2的曼哈顿距离
boolean	avatarGetKey	StateObservation stateObs	判断精灵是否得到钥匙
double	heuristic	StateObservation stateObs	启发式函数，返回局面评分
void	debugPos	Vector2d vec, String head	输出vec的位置信息

核心代码

```
protected void limitDepthFirst(StateObservation stateObs, ElapsedCpuTimer elapsedTimer, int depth){
    if(Reach the end of limitDepthFirst){
        nowStateScore = heuristic(stateObs);
        if(nowStateScore better than bestCost)
            bestAction = now actions;
        return;
    }
    else if(stateObs is not game start){
```

```
        if(stateObs in closeList && depth same)
            return;
        else{
            closeList.add(stateObs);
            stateDepth.add(depth);
        }
    }
    else{ // at the beginning of limitdfs, init all
        closeList.clear();
        stateDepth.clear();
    }

    for(all actions in stateObs){
        try this action in stCopy;
        limitDepthFirstAction.add(action);
        if(Game win) {
            nowStateScore = heuristic(stateObs);
            if(nowStateScore better than bestCost)
                bestAction = now actions;
            stCopy = stateObs.copy(); // reset stCopy
            limitDepthFirstAction.delete(action);
            continue;
        }
        else{
            limitDepthFirst(stCopy,elapsedTimer,depth);
            stCopy = stateObs.copy(); // reset stCopy
            limitDepthFirstAction.delete(action);
            continue;
        }
    }
}
```

任务三 A*搜索

根据自己自行测试，可以在有限时间内完成第二关与第三关的搜索并成功通关。

变量简介

变量类型	变量名	变量含义
ArrayList	closeList	存储已经走过的历史路径
PriorityQueue	openList	存储尚未走的已探索到的步骤
boolean	getAnswer	是否得到了能走到终点的答案
ArrayList<Types.ACTIONS>	aStarAction	存储aStar中的每一步动作
Vector2d	goalpos	门的位置
Vector2d	keypos	钥匙的位置

函数简介

返回类型	函数名	传入参数	函数功能
void	initAgent	null	初始化Agent
boolean	isInCloseList	StateObservation obs	检测是否在历史状态中
boolean	isInOpenList	StateObservation obs	检测是否在尚未走的已搜索到的区域中

返回类型	函数名	传入参数	函数功能
void	aStar	StateObservation stateObs, ElapsedCpuTimer elapsedTimer, int depth	计算从stateObs出发的受时间限制的aStar路径
Types.ACTIONS	act	StateObservation stateObs, ElapsedCpuTimer elapsedTimer	根据局面stateObs调用aStar并返回当轮动作
double	getDistance	Vector2d vec1, Vector2d vec2	返回vec1与vec2的曼哈顿距离
boolean	avatarGetKey	StateObservation stateObs	判断精灵是否得到钥匙
double	heuristic	StateObservation stateObs	启发式函数，返回局面评分

核心代码

```
protected void aStar(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {
    initAgent();
    openList.add(startNode); // 将初始状态加入openList
    while(!openList.isEmpty())
    {
        Node tmp = openList.poll(); // 取得分最高的节点tmp
        aStarAction = tmp.actions; // 将aStarAction初始化为tmp从起点到当前位置的所有动作
        closeList.add(tmp.stateObs.copy()); // 将tmp的状态加入closeList中标记已走过

        for(all actions in stateObs){
            stCopy = tmp.stateObs.copy();
            try this action in stCopy;
            aStarAction.add(act);
            if(Game win) {
                getAnswer = true;
                return ; // 最终的序列步骤在aStarAction中
            }
            else if(Game over || stateObs in closeList) { // 如果游戏结束或发现曾走过，则回溯
                aStarAction.delete(action);
                continue;
            }
            else if(stateObs in openList){ // 如果发现当前局面在openList待尝试
                if(new actions better than old){ // 如果当前走法优于之间走法
                    refresh openList; // 则更新动作
                }
                aStarAction.delete(action); // 回溯
            }
            else{ // 这是一个新的动作
                openList.add(new Node(stCopy,heuristic(stCopy),aStarAction)); // 加入新的动作
                aStarAction.delete(action); // 回溯
            }
        }
    }
}
```

任务四 蒙特卡洛树搜索

算法框架

```
while(时间限制内){
    treePolicy选择一个当前可达状态selected；
    对子状态执行rollOut，得到得分；
    从selected开始执行backUp；
}
通过mostVisitedAction返回次数最大的动作并作为结果；
```

函数简介

函数名	传入参数	函数功能
rollOut	null	不断随机向下搜索，当游戏结束或达到递归层数后对状态评分。更新Agent得分bound后返回得分。
backUp	SingleTreeNode node, double result	传入一个节点与他的得分，对这个节点与他的所有祖先节点，访问次数+1，总分+=得分。
treePolicy	null	如果当前节点有子节点未访问，则返回其中一个，否则调用uct从所有子节点中选择一个。
uct	null	根据子节点总分，访问次数，Agent的得分bound计算节点分数，然后选择得分最高的返回。

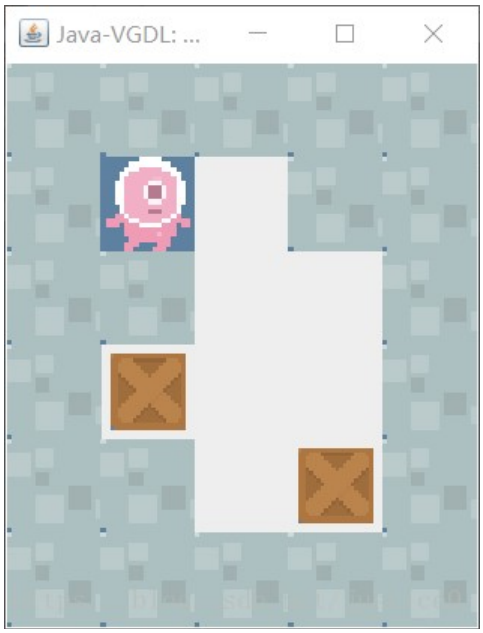
核心代码

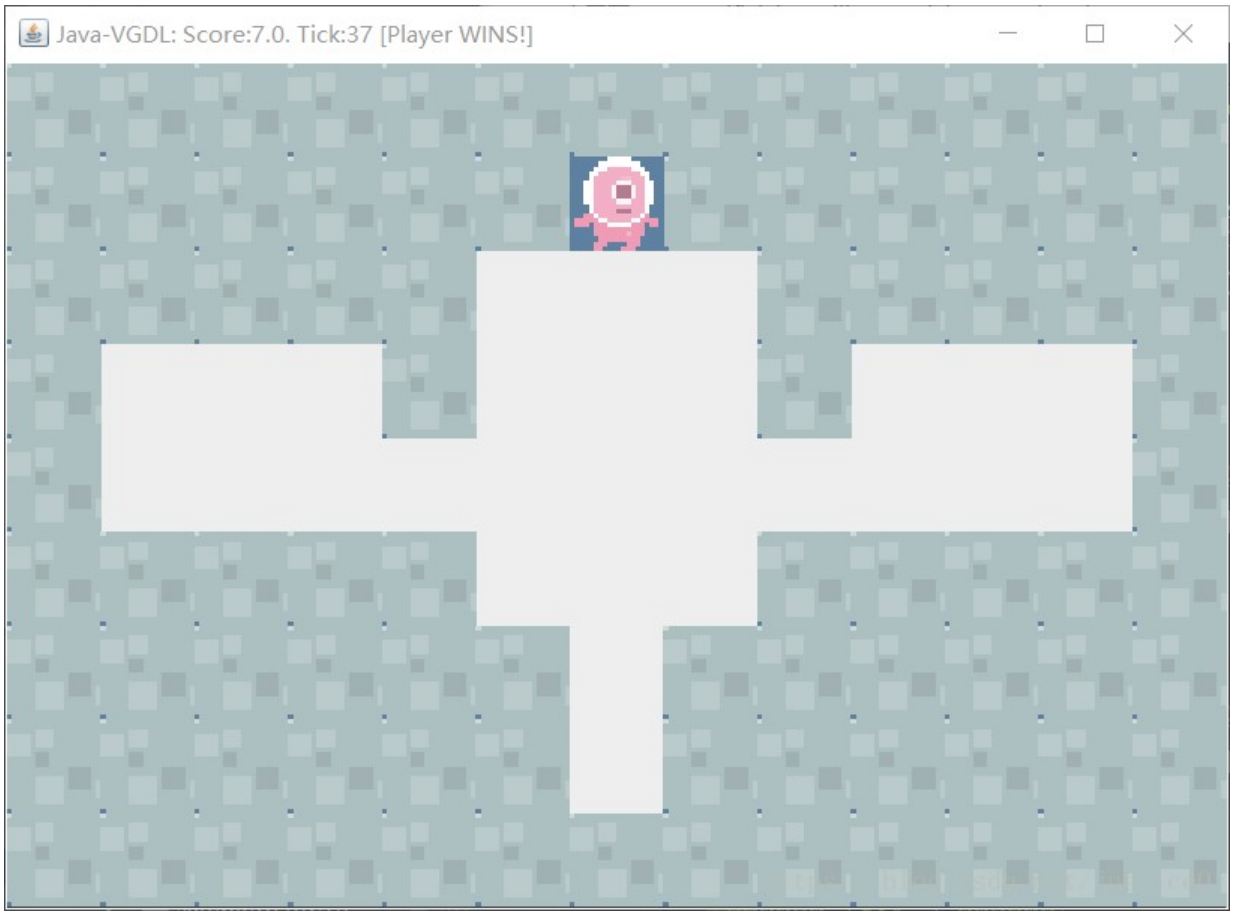
uct算法作为关键，他的子节点计算方式如下：

childValue(平均估值) = $normalize(\frac{childTotalValue}{childVisitTimes+\epsilon}), (\epsilon = 1 * 10^-6)$
uctValue(节点分数) = $childValue + \sqrt{\frac{2 \ln parentVisitTimes+1}{childVisitTimes+\epsilon}} + \xi$, ξ是噪声

简而言之，访问次数少的，平均分高的子节点节点分数更高，更容易被roolOut选中。由于在act中我们返回的是访问次数最多的子节点作为状态，所以可以看出uct对于可能成功的子节点更友好。

运行结果




Java-VGDL: Score:7.0. Tick:37 [Player WINS!]


Java-VGDL: Score:11.0. Tick:88 [Player WINS!]