报告题目：Bait游戏&搜索算法

王崧睿 221502011 detect0530@gmail.com

引言：

在个人过去的实践中，搜索算法是低效暴力的代名词，但是通过本课程的学习，从深度优先宽度优先到代价优先，再到A*算法，我才发现搜索算法的强大之处。优秀的启发式函数可以大大提高搜索效率，搜索算法的强大之处在于其可以解决各式各样的问题，比如本次实验中的Bait游戏，可以通过搜索算法来解决。并在一次次优化算法的过程中，我也对搜索算法有了更深的理解。

2 实验内容

2.1 Task1: 深度优先搜索

2.1.1 记录走过的状态

要求使用深度优先搜索完成Bait游戏，我们首先要确保一定可以遍历完所有的情况（即completion），因为游戏规定精灵可以上下左右移动，那么每张地图都构成一个图，那么首要问题就是避免死循环，即避免走“回头路”，我们使用

```
private ArrayList<StateObservation> Visited= new ArrayList<StateObservation>();
```

定义一个状态数组表示已经走过的状态，仔细阅读源码框架后，$StateObservation$类表示了当前局面的所有信息，因为实际上我们不能只是简单的记录走过的坐标集合，因为一旦精灵拿到钥匙或者推开箱子都会使得局面状态改变而使得同样坐标的位置可以再次走到，所以我们需要记录每个局面的状态，即记录每个局面的$StateObservation$。

同时注意到，比较两个$StateObservation$是否相同的方法不能简单用$==$，在$StateObservation$类中给了我们$equalPosition$接口。于是我们创建一个方法用来判断当前局面是否已经走过：

```
    private boolean CheckifVisited(StateObservation stataObs) {
        for (int i = 0; i < Visited.size(); i++) {
            if (stataObs.equalPosition(Visited.get(i))) {
                return true;
            }
        }
        return false;
    }
```

2.1.2 act接口调用

阅读框架源码知$controler$每次行动调用的是act函数，也就是说我们搜索结束后，不能直接返回$action$数组，而是：

- 1. 第一次act时，搜索得到正确actionlist，返回第一个元素。
- 2. 第二次act时，返回actionlist的第二个元素。
- 3. 依此类推.....

```
//act函数内：

if(OK==true){
    numstep++;
    return DFSActions.get(numstep);
}

......

Visited.clear();
DFSActions.clear();
DFS(stateObs,elapsedTimer);
if(DFSActions.size()==0){
    System.out.println("action nil");
    return Types.ACTIONS.ACTION_NIL;
}
else{
    OK=true;
    return DFSActions.get(0);
}
```

2.1.3 深度优先搜索的实现

我采用递归的方式实现DFS，每次递归时，先判断当前局面是否已经走过，如果走过则返回，否则将当前局面加入$Visited$数组，然后判断当前局面是否为终局，如果是则返回，否则遍历当前局面的所有可行动作，对每个动作进行递归，直到找到终局或者所有动作都走过。

在搜索的时候，我始终使用$DFSaction$数组记录当前的动作序列。

```
 boolean DFS(StateObservation stataObs, ElapsedCpuTimer elapsedTimer){
    if(CheckifVisited(stataObs)){
        return false;
    }
    Visited.add(stataObs);
    StateObservation stCopy = stataObs.copy();
    ArrayList<Types.ACTIONS> actions = stataObs.getAvailableActions();
    //  StateObservation stCopy = stataObs.copy(),re=null;
    for(Types.ACTIONS action:actions) {
        stCopy.advance(action);
        DFSActions.add(action);
        if (stCopy.getGameWinner() == Types.WINNER.PLAYER_WINS) {
            return true;
        } else if (CheckifVisited(stCopy) || stCopy.isGameOver()) {
            stCopy = stataObs.copy();
            DFSActions.remove(DFSActions.size() - 1);
```

```
                continue;
            } else if (DFS(stCopy, elapsedTimer)) {
                return true;
            } else {
                stCopy = stataObs.copy();
                DFSActions.remove(DFSActions.size() - 1);
                continue;
            }
        }
        return false;
    }
```

2.1.4 TASK1 完整代码展示

```
package controllers.dfs;

import java.awt.Graphics2D;
import java.util.ArrayList;
import java.util.Random;
import javax.swing.*;
import java.awt.*;

import core.game.ForwardModel;
import core.game.Observation;
import core.game.StateObservation;
import core.player.AbstractPlayer;
import ontology.Types;
import tools.ElapsedCpuTimer;
import tools.Vector2d;
import java.awt.*;
import java.awt.image.BufferedImage;



public class Agent extends AbstractPlayer {

    /**
     * Random generator for the agent.
     */
    protected Random randomGenerator;

    /**
     * Observation grid.
     */
    protected ArrayList<Observation> grid[][];

    /**
     * block size
     */
    protected int block_size,numstep=0;
    private boolean OK=false;
```

```
    private ArrayList<StateObservation> Visited= new ArrayList<StateObservation>
();
    private ArrayList<Vector2d> Key= new ArrayList<tools.Vector2d>();
    private ArrayList<Types.ACTIONS>  DFSActions= new ArrayList<Types.ACTIONS>();

    public Agent(StateObservation so, ElapsedCpuTimer elapsedTimer)
    {
        randomGenerator = new Random();
        grid = so.getObservationGrid();
        block_size = so.getBlockSize();
    }

    private boolean CheckifVisited(StateObservation stataObs) {
        for (int i = 0; i < Visited.size(); i++) {
            if (stataObs.equalPosition(Visited.get(i))) {
                return true;
            }
        }
        return false;
    }
    boolean DFS(StateObservation stataObs, ElapsedCpuTimer elapsedTimer){
        if(CheckifVisited(stataObs)){
            return false;
        }
        Visited.add(stataObs);
        StateObservation stCopy = stataObs.copy();
        ArrayList<Types.ACTIONS> actions = stataObs.getAvailableActions();
      //  StateObservation stCopy = stataObs.copy(),re=null;
        for(Types.ACTIONS action:actions) {
            stCopy.advance(action);
            DFSActions.add(action);
            if (stCopy.getGameWinner() == Types.WINNER.PLAYER_WINS) {
                return true;
            } else if (CheckifVisited(stCopy) || stCopy.isGameOver()) {
                stCopy = stataObs.copy();
                DFSActions.remove(DFSActions.size() - 1);
                continue;
            } else if (DFS(stCopy, elapsedTimer)) {
                return true;
            } else {
                stCopy = stataObs.copy();
                DFSActions.remove(DFSActions.size() - 1);
                continue;
            }
        }
        return false;
    }

    public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer
elapsedTimer) {
        if(OK==true){
            numstep++;
            return DFSActions.get(numstep);
```

```
        }
        grid = stateObs.getObservationGrid();

        Visited.clear();
        DFSActions.clear();
        DFS(stateObs,elapsedTimer);
        if(DFSActions.size()==0){
            System.out.println("action nil");
            return Types.ACTIONS.ACTION_NIL;
        }
        else{
            OK=true;
            return DFSActions.get(0);
        }
    }
```

2.1.5 Algorithm Performance

在$10s$的限制里可以解决第1,2,3张地图。但是很明显看出来路径有随机性，很多时候绕了路。

2.2 Task2: 深度受限的搜索+初步Astar

起初看题以为是加个$MaxDepthLimitt$就完事了，但是注意到题目有要求用启发式函数。我们不妨把这两者结合，即：

- 搜索中仍有深度要求，搜到指定的参数$MaxDepthLimit$后就返回。
- 对于深度达到$MaxDepthLimit$或者终局的局面，我们用启发式函数来评估这个局面的好坏。

基于上述两点，我们一次可以找到多个深度为指定值的局面并使用启发式函数选择最好的一个$ActionList$执行。

2.2.1 启发式函数的设计

本题只是设计一个简单的启发式函数，初心是过TASK1即可，行之有效的启发式函数设计待我在后面的$A*$算法时来介绍。

- 赢得游戏 score + 1*W_1
- 丢失游戏 score - 1*W_2
- 计算得分 score + GetGameScore()*W_3
- 得到钥匙 score + 1*W_1
- 箱子推到了钥匙位置上 score - W_4
- Distance：
    1. 拿到钥匙前：score - DistanceToKey()*W_5

2. 拿到钥匙后：score + DistanceToDoor()*W_6

```
    private int heuristic(StateObservation stateObs){
        int score=0;
        ArrayList<Observation>[] movingPositions = stateObs.getMovablePositions();
        if(stateObs.getGameWinner()==Types.WINNER.PLAYER_WINS){
            System.out.println("Winnnnnn");
            for(int i=0;i<Visited.size();i++){
               System.out.println("Visited x=" +
(Visited.get(i).getAvatarPosition().x/50+1) + " y=" +
(Visited.get(i).getAvatarPosition().y/50+1));
            }

            score+=1000000;
        }
        if(stateObs.getGameWinner()==Types.WINNER.PLAYER_LOSES){
            System.out.println("Loseeeee");
            for(int i=0;i<Visited.size();i++){
                System.out.println("Visited x=" +
(Visited.get(i).getAvatarPosition().x/50+1) + " y=" +
(Visited.get(i).getAvatarPosition().y/50+1));
            }
            score-=1000000;
        }
        score+=(int)stateObs.getGameScore()*100;
      //  System.out.println("score= " + score);

        boolean flag=false;
        for(int i=0;i<Visited.size();i++){
            if(keypos.equals(stateObs.getAvatarPosition())){
                flag=true;
            }
        }
        if(keypos.equals(stateObs.getAvatarPosition())) flag=true;


       // System.out.println(stateObs.getAvatarPosition() + " " + keypos);

        if(flag&&KKK) score+=500;

        if(Getkey(stateObs).isEmpty()){
            score-=GetDistance(stateObs.getAvatarPosition(),goalpos);
        }
        else{
            if(GetBox(stateObs).contains(keypos)) score-=100000;

            score-=GetDistance(stateObs.getAvatarPosition(),keypos);
        }
       // System.out.println("score= " + score);

        return score;
    }
```

2.2.2 搜索的实现

和第一题类似，只不过加了个$MaxDepthLimit$的限制，然后选择最优的启发式$socre$对应的$ActionList$即。

细节上说，设置一个Has_Init开关，控制现在是否已经初始化，如果现在还没初始化，则每次搜索前先清空$Visited$数组和$DFSActions$数组，然后再进行搜索。

```
    void LFS(StateObservation stataObs, ElapsedCpuTimer elapsedTimer,int dep){
        if(dep==MaxDepth||stataObs.getGameWinner()==Types.WINNER.PLAYER_WINS){
            int score=heuristic(stataObs);
            if(score>MaxScore){
                MaxScore=score;
                MaxAction= (ArrayList<Types.ACTIONS>) DFSActions.clone();
            }
            return;
        }

        if(Has_Init){
            if(CheckifVisited(stataObs)){
                return;
            }
        }
        else{
            Visited.clear();
            DFSActions.clear();
            MaxAction.clear();
            Has_Init=true;
        }

        Visited.add(stataObs);
        StateObservation stCopy = stataObs.copy();
        ArrayList<Types.ACTIONS> actions = stataObs.getAvailableActions();
        for(Types.ACTIONS action:actions) {
            stCopy.advance(action);
            DFSActions.add(action);

            if(CheckifVisited(stCopy)){
                stCopy=stataObs.copy();
                DFSActions.remove(DFSActions.size()-1);
                continue;
            }

            if (stCopy.getGameWinner() == Types.WINNER.PLAYER_WINS) {
                int score=heuristic(stCopy);
                if(score>MaxScore){
                    MaxScore=score;
                    MaxAction= (ArrayList<Types.ACTIONS>) DFSActions.clone();
                }
```

```
                stCopy=stataObs.copy();
                DFSActions.remove(DFSActions.size()-1);
                Has_Init=true;
                return;
            }
            else{
                LFS(stCopy,elapsedTimer,dep+1);
                Visited.remove(stCopy);
                stCopy=stataObs.copy();
                DFSActions.remove(DFSActions.size()-1);

                continue;
            }
        }

    }
```

在act函数里，也根据Has_Init开关的不同施行不同的造作：

- 如果没初始化，则意味着现在需要搜索一遍，即进入LFs函数
- 如果已经搜索过了，那么依次执行搜索好的$BestActionList$就可以了。

2.2.3 TASK2 完整代码展示

```
package controllers.dls;

import java.awt.Graphics2D;
import java.util.ArrayList;
import java.util.Random;
import javax.swing.*;
import java.awt.*;

import core.game.ForwardModel;
import core.game.Observation;
import core.game.StateObservation;
import core.player.AbstractPlayer;
import ontology.Types;
import ontology.avatar.MovingAvatar;
import tools.ElapsedCpuTimer;
import tools.Vector2d;
import java.awt.*;
import java.awt.image.BufferedImage;

public class Agent extends AbstractPlayer {

    protected Random randomGenerator;

    protected ArrayList<Observation> grid[][];
```

```
    protected int block_size,numstep=0,MaxScore=-10000000;
    private boolean OK=false,Has_Init=false,KKK=false;

    private final int MaxDepth=7;



    private ArrayList<StateObservation> Visited= new ArrayList<StateObservation>
();
    private ArrayList<Types.ACTIONS>  DFSActions= new ArrayList<Types.ACTIONS>
(),MaxAction= new ArrayList<Types.ACTIONS>();
    private Vector2d goalpos,keypos;
    /**
     * Public constructor with state observation and time due.
     * @param so state observation of the current game.
     * @param elapsedTimer Timer for the controller creation.
     */

    public Agent(StateObservation so, ElapsedCpuTimer elapsedTimer)
    {
        randomGenerator = new Random();
        grid = so.getObservationGrid();
        block_size = so.getBlockSize();
        ArrayList<Observation>[] fixedPositions = so.getImmovablePositions();
        ArrayList<Observation>[] movingPositions = so.getMovablePositions();
        goalpos = fixedPositions[1].get(0).position; //目标的坐标
        keypos = movingPositions[0].get(0).position ;//钥匙的坐标
    }

    //检查是否走过
    private boolean CheckifVisited(StateObservation stataObs) {
        for (int i = 0; i < Visited.size(); i++) {
            if (stataObs.equalPosition(Visited.get(i))) {
                return true;
            }
        }
        return false;
    }

    //获取箱子的坐标
    private ArrayList<Vector2d> GetBox(StateObservation stataObs){
        ArrayList<Observation>[][] observationGrid= stataObs.getObservationGrid();
        ArrayList<Vector2d> Box= new ArrayList<tools.Vector2d>();
        for(int i=0;i<observationGrid.length;i++){
            for(int j=0;j<observationGrid[i].length;j++){
                if(observationGrid[i][j]!=null){
                    for(int k=0;k<observationGrid[i][j].size();k++){
                        if(observationGrid[i][j].get(k).itype==8){
                            Box.add(observationGrid[i][j].get(k).position);
                        }
                    }
                }
            }
```

```
        }
        return Box;
    }

    //获取钥匙的坐标
    private ArrayList<Vector2d> Getkey(StateObservation stataObs){
        ArrayList<Observation>[][] observationGrid= stataObs.getObservationGrid();
        ArrayList<Vector2d> Key= new ArrayList<tools.Vector2d>();
        for(int i=0;i<observationGrid.length;i++){
            for(int j=0;j<observationGrid[i].length;j++){
                if(observationGrid[i][j]!=null){
                    for(int k=0;k<observationGrid[i][j].size();k++){
                        if(observationGrid[i][j].get(k).itype==6){
                            Key.add(observationGrid[i][j].get(k).position);
                        }
                    }
                }
            }
        }
        return Key;
    }
    //得到两点的曼哈顿距离
    int GetDistance(Vector2d a,Vector2d b){
        return (int)Math.abs(a.x-b.x)/50+(int)Math.abs(a.y-b.y)/50;
    }

    //启发式估价函数
    private int heuristic(StateObservation stateObs){
        int score=0;
        ArrayList<Observation>[] movingPositions = stateObs.getMovablePositions();
        if(stateObs.getGameWinner()==Types.WINNER.PLAYER_WINS){
            score+=1000000;
        }
        if(stateObs.getGameWinner()==Types.WINNER.PLAYER_LOSES){
            score-=1000000;
        }
        score+=(int)stateObs.getGameScore()*100;

        boolean flag=false;
        for(int i=0;i<Visited.size();i++){
            if(keypos.equals(stateObs.getAvatarPosition())){
                flag=true;
            }
        }
        if(keypos.equals(stateObs.getAvatarPosition())) flag=true;
        if(flag&&KKK) score+=500;

        if(Getkey(stateObs).isEmpty()){
            score-=GetDistance(stateObs.getAvatarPosition(),goalpos);
        }
        else{
            if(GetBox(stateObs).contains(keypos)) score-=100000;

            score-=GetDistance(stateObs.getAvatarPosition(),keypos);
```

```
        }

        return score;
    }
    void LFS(StateObservation stataObs, ElapsedCpuTimer elapsedTimer,int dep){


        if(dep==MaxDepth||stataObs.getGameWinner()==Types.WINNER.PLAYER_WINS){
            int score=heuristic(stataObs);

            if(score>MaxScore){
                MaxScore=score;
                MaxAction= (ArrayList<Types.ACTIONS>) DFSActions.clone();
            }

            return;
        }

        if(Has_Init){
            if(CheckifVisited(stataObs)){
                return;
            }
        }
        else{
            Visited.clear();
            DFSActions.clear();
            MaxAction.clear();
            Has_Init=true;
        }

        Visited.add(stataObs);
        StateObservation stCopy = stataObs.copy();
        ArrayList<Types.ACTIONS> actions = stataObs.getAvailableActions();
        for(Types.ACTIONS action:actions) {
            stCopy.advance(action);
            DFSActions.add(action);

            if(CheckifVisited(stCopy)){
                stCopy=stataObs.copy();
                DFSActions.remove(DFSActions.size()-1);
                continue;
            }

            if (stCopy.getGameWinner() == Types.WINNER.PLAYER_WINS) {

                int score=heuristic(stCopy);
                if(score>MaxScore){
                    MaxScore=score;
                    MaxAction= (ArrayList<Types.ACTIONS>) DFSActions.clone();
                }

                stCopy=stataObs.copy();
                DFSActions.remove(DFSActions.size()-1);
                Has_Init=true;
```

```
                return;
            }
            else{
                LFS(stCopy,elapsedTimer,dep+1);
                Visited.remove(stCopy);
                stCopy=stataObs.copy();
                DFSActions.remove(DFSActions.size()-1);

                continue;
            }
        }

    }

    public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer
elapsedTimer) {
        if(Has_Init){
            numstep++;int now=numstep;
            if(numstep>MaxAction.size()-1){
                Has_Init=false;numstep=0;MaxScore=-100000;
            }
            else{
                return MaxAction.get(now);
            }
        }

        if(!Getkey(stateObs).isEmpty()) KKK=true;
        else KKK=false;MaxScore=-10000000;
        System.out.println(stateObs.getAvatarPosition());
        LFS(stateObs,elapsedTimer,0);
        System.out.println("MaxScore= " + MaxScore + " " +  MaxAction.size());
        for(int i=0;i<MaxAction.size();i++){
            System.out.println("action " + MaxAction.get(i));
        }

        if(MaxAction.size()==0){
            System.out.println("action nil");
            return Types.ACTIONS.ACTION_NIL;
        }
        else{
            return MaxAction.get(0);
        }
    }

    private void printDebug(ArrayList<Observation>[] positions, String str) {
        if (positions != null) {
            System.out.print(str + ":" + positions.length + "(");
            for (int i = 0; i < positions.length; i++) {
                System.out.print(positions[i].size() + ",");
            }
            System.out.print("); ");
            for(int i=0;i<positions.length;i++){
                for(int j=0;j<positions[i].size();j++){
                    System.out.println(str + " x=" +
```

```
positions[i].get(j).position.x/50 + " y=" + positions[i].get(j).position.y/50);
                }
            }
        } else System.out.print(str + ": 0; ");
    }
}
```

2.2.4 Algorithm Performance

程序的执行效果与各项参数密切相关，首先是各项启发式函数的系数常数，经过调整得到以下参数：

- 赢得游戏 $score + 1 * W_1(100000)$
- 丢失游戏 $score - 1 * W_2(100000)$
- 计算得分 $score + GetGameScore() * W_3(100)$
- 得到钥匙 $score + 1 * W_1(500)$
- 箱子推到了钥匙位置上 $score - W_4(100000)$
- Distance：
    1. 拿到钥匙前：$score - DistanceToKey() * W_5(1)$
    2. 拿到钥匙后：$score + DistanceToDoor() * W_6(1)$

另外的一个关键参数是$MaxDepthLimit$，这个参数十分关键，设小了会导致程序找不到成功的路径(比如距离小到无法在一次搜索中推开箱子拿到钥匙)，设置大了会导致程序运行时间过长，甚至超过单步的时间限制。

经过尝试，设置$MaxDepthLimit = 3$ 能通过第一关。而设置$MaxDepthLimit = 15$能通过第二关。

2.3 Task3: Astar启发式搜索

2.3.1 A*算法问题概述：

本题要求使用$A*$算法来解决Bait游戏，在BFS的基础上加入了启发式函数，即每次搜索时，优先搜索启发式函数值最小的节点，这样可以大大提高搜索效率。

其算法的核心在于启发式函数的设定，优秀的启发式函数可以极大程度加速搜索。

在本题中，设定为单步时限$100ms$，经过参数反复打磨与设定，我的程序可以在第1,2,3,5张图中在$100ms$内直接找到获胜策略。

设计好了启发式函数后，整体框架其实类似于BFS，只不过在每次搜索时，我们需要优先搜索集合中启发式函数值最小的节点。这一点可以用$java$中的$TreeSet$内置的红黑树加速实现（因为到后期节点数会很多，所以用$TreeSet$来存储节点可以大幅度优化时间开销）。

2.3.2 启发式函数的设计

先阐述一下游戏的整体流程：

推动箱子或者填坑 --> 拿到钥匙 --> 推动箱子或者填坑 --> 到达终点

整个过程和箱子的位置、是否拿到钥匙、箱子是否进坑密切相关。

于是自然有以下参数：

- 0. 深度$Dep * c_0$
- 1. 游戏得分$GetGameScore * c_1$
- 2. 是否赢得游戏$WinorLose * c_2$
- 3. $Distance * c_3$ （拿到钥匙前Distance为到钥匙的距离，拿到钥匙后Distance为到终点的距离）

看起来现在的启发式函数会直奔钥匙并保证不会掉坑里并且会选择得分高的方式（推箱子入坑，吃蘑菇等），但是在第二张图中会出现下面的情况：

即不会往两边的箱子走，因为往两边走意味着远离钥匙，我们的启发式函数不鼓励我们这样做。

另外是容易出现推箱子进入死角的情况。

为了避免上述两种不利情况，我们为箱子单独增加参数：

理论上，我们需要为每一个box都计算一个单独但是权值，但是实际上因为走一步影响的box有限，故只需统计走一步能影响到的box数就可以。形式化的，即我们只对于当前精灵位置曼哈顿距离小于等于二的box进行权值计算。

- 4.1 自由度`*C_4_1` （自由度表示箱子左右或者上下都是空的，自由度越大，箱子越容易被推动，值为0/1/2）
- 4.2 离当前box最近的坑的距离`*C_4_2` (我们期望推着箱子往坑走)

- 4.3 是否可以一步入洞，即箱子的一侧是精灵对侧是坑 *C_4_3 （这种情况我们期望把箱子推进坑）
- 4.4 是否挡住了钥匙 *C_4_4 （这种情况赋很大的负权值直接毙掉就可以）

上述参数可以有效解决箱子进入死角或者推着箱子乱走的情况。

同时，对于精灵，我们要引导其往箱子走。

- 5 与精灵最近的box距离$*C_5$ （我们期望精灵往箱子走）

至此，启发式函数的设计已经完成。剩下的工作是对于每个参数赋予合适的权值。（炼丹太痛苦了

经过反复调试，我最终得到的参数如下：

```
double
W_0=-2,W_1=100,W_2=100000000,W_3=-5,W_4_1=30,W_4_2=10,W_4_3=1000,W_4_4=-10000000,W
_5=-10;
```

并成功在总时间$100ms$内通过了第1,2,3,5张图。（远远比要求的单步$100ms$优秀。

code如下：

```
    private double heuristic(StateObservation stateObs,boolean HasGetKey,int dep){
        double
score=0,W_0=-2,W_1=100000000,W_2=100,W_3=-5,W_4_1=30,W_4_2=10,W_4_3=1000,W_4_4=-10
,W_5=-10;
        double
Dep,WinorLose=0,Nowscore=0,Distance,Box_1=0,Box_2=0,Box_3=0,Box_4=0,GG=0,MinDistan
ceBox=0;

        Dep=dep;

        if(stateObs.getGameWinner()==Types.WINNER.PLAYER_WINS){
            WinorLose=1;
        }
        if(stateObs.getGameWinner()==Types.WINNER.PLAYER_LOSES){
            WinorLose=-1;
        }


        Nowscore=stateObs.getGameScore();

        if(HasGetKey){
            Distance=GetDistance(stateObs.getAvatarPosition(),goalpos);
        }
        else{
            Distance=GetDistance(stateObs.getAvatarPosition(),keypos);
        }

        ArrayList<Vector2d> Box=GetBox(stateObs);
        ArrayList<Observation>[][] observationGrid= stateObs.getObservationGrid();
        int[][] Map=new int[observationGrid.length][observationGrid[0].length];
```

```
        for(int i=0;i<observationGrid.length;i++){
            for(int j=0;j<observationGrid[i].length;j++){
                if(observationGrid[i][j].size()!=0){
                    for(int k=0;k<observationGrid[i][j].size();k++){
                        Map[i][j]=observationGrid[i][j].get(k).itype;
                    }
                }
                else{
                    Map[i][j]=-1;
                }
            }
        }

        int ax=(int)stateObs.getAvatarPosition().x/50,ay=
(int)stateObs.getAvatarPosition().y/50;

        for(int i=0;i<Box.size();i++){

            int x=(int)Box.get(i).x/50,y=(int)Box.get(i).y/50;
            int n=observationGrid.length,m=observationGrid[0].length;

            if(x==keypos.x/50&&y==keypos.y/50&& !HasGetKey) GG=1;

            boolean is=false;
            if(ax==x&&(ay==y+1||ay==y-1)) is=true;
            if(ay==y&&(ax==x+1||ax==x-1)) is=true;
            if(ax==x-1&&ay==y-1) is=true;
            if(ax==x+1&&ay==y+1) is=true;
            if(ax==x-1&&ay==y+1) is=true;
            if(ax==x+1&&ay==y-1) is=true;
            if(!is) continue;

            if(x<n-1&&x>0){
                if(Map[x+1][y]==-1&&Map[x-1][y]==-1) Box_1++;
            }
            if(y<m-1&&y>0){
                if(Map[x][y+1]==-1&&Map[x][y-1]==-1) Box_1++;
            }

            if(x<n-1) if(Map[x+1][y]==2) Box_2++;
            if(x>0) if(Map[x-1][y]==2) Box_2++;
            if(y<m-1) if(Map[x][y+1]==2) Box_2++;
            if(y>0) if(Map[x][y-1]==2) Box_2++;

            if(x<n-1&&x>0){
                if(Map[x+1][y]==2&&Map[x-1][y]==-1) Box_3++;
                if(Map[x-1][y]==2&&Map[x+1][y]==1) Box_3++;
            }
            if(y<m-1&&y>0){
                if(Map[x][y+1]==2&&Map[x][y-1]==-1) Box_3++;
                if(Map[x][y-1]==2&&Map[x][y+1]==-1) Box_3++;
            }

            Box_4+=GetDistanceHole(x*50,y*50,stateObs);
```

```
        }

        if(HasGetKey) GG--;

        MinDistanceBox=GetMinDistanceBox(stateObs);


score=W_0*Dep+W_1*WinorLose+W_2*Nowscore+W_3*Distance+W_4_1*Box_1+W_4_2*Box_2+W_4_
3*Box_3+W_4_4*Box_4+GG*-1000000+W_5*MinDistanceBox;
        return score;
    }
```

2.3.3 A*算法的实现

与在2.3.1概述中说的一样，我们用红黑树数据结构实现。

为了方便代码实现，实现了基础类Node。

```
public Node(double score, StateObservation stateObs, ArrayList<Types.ACTIONS>
Actions, int depth,boolean GetKey) {
        this.score = score;
        this.stateObs = stateObs;
        this.Actions = Actions;
        this.depth = depth;
        this.GetKey=GetKey;
}
```

其中$score$为启发式函数值，$stateObs$为当前局面，$Actions$为当前局面的$ActionList$，$depth$为当前深度，$GetKey$为是否拿到钥匙。

同时定义红黑树排序关键字是$score$。

```
    public int compareTo(Node other) {
        // 降序排列
        int ok = Double.compare(other.score, this.score);
        if(ok==0){
            if(stateObs.equalPosition(other.stateObs)){
                return 0;
            }
            else return 1;

        }
        else return ok;
    }
```

Astar算法的实现框架代码如下:（和之前两个的框架类似，只不过变成了优先队列展开而已

```
ArrayList<Types.ACTIONS> Astar(Node StartNode, ElapsedCpuTimer elapsedTimer,int
dep){
        int tot=0;
        Todolist.add(StartNode);
        while(!Todolist.isEmpty()){
            Node now=Todolist.pollFirst();
            System.out.println("score= " + now.getScore() + " depth= " +
now.getDepth() + " GetKey= " + now.GetKey + "Positon= " +
(now.getStateObs().getAvatarPosition().x/50)+","+
(now.getStateObs().getAvatarPosition().y/50)+"-------------");
            tot++;
            StateObservation stataObs=now.getStateObs();
            Visited.add(stataObs);//记录走过的节点
            ArrayList<Types.ACTIONS> actions =
now.getStateObs().getAvailableActions();
            for(Types.ACTIONS action:actions){
                StateObservation stCopy = now.getStateObs().copy();
                stCopy.advance(action);
                ArrayList<Types.ACTIONS>  AstarAction= (ArrayList<Types.ACTIONS>)
now.getActions().clone();
                AstarAction.add(action);
                if(stCopy.getGameWinner()==Types.WINNER.PLAYER_WINS){
                    OK=true;
                    return AstarAction;
                }

if(stCopy.getGameWinner()==Types.WINNER.PLAYER_LOSES||CheckifVisited(stCopy)){
                    AstarAction.remove(AstarAction.size()-1);
                    continue;
                }

                //判断是否取得了钥匙
                boolean NowGetKey=false;
                if(Getkey(now.getStateObs()).size()==1&& Getkey(stCopy).isEmpty())
{
                    NowGetKey=true;
                }

                //如果Todolist有该位置的节点，比较两个节点的score，取最优的
                Node node=contain(Todolist,stCopy);
                if(node!=null){
                    double score0=node.getScore();

                    double score1=heuristic(stCopy,NowGetKey|now.GetKey,dep+1);
                    if(score0>=score1){
                        AstarAction.remove(AstarAction.size()-1);
                    }
                    else{
                        Node Newnode=new Node(score1,stCopy,AstarAction,
dep+1,NowGetKey|now.GetKey);
                        Todolist.remove(node);
                        Todolist.add(Newnode);
                    }
```

```
                }
                //反之，则加入红黑树中排序
                else{
                    Node Newnode=new
Node(heuristic(stCopy,NowGetKey|now.GetKey,dep+1),stCopy,AstarAction,
dep+1,NowGetKey|now.GetKey);
                    Todolist.add(Newnode);
                }
            }
        }

        System.out.println("GGGGGGGGGGGGGGGGGGGGG");
        return (new ArrayList<Types.ACTIONS>());

    }
```

2.3.4 TASK3 完整代码展示

```
package controllers.Astar;

import java.util.ArrayList;
import java.util.PriorityQueue;
import java.util.Random;
import java.util.TreeSet;

import core.game.Observation;
import core.game.StateObservation;
import core.player.AbstractPlayer;
import ontology.Types;
import tools.ElapsedCpuTimer;
import tools.Vector2d;

import static java.lang.Double.min;

// Node类用于描述一个状态，包括状态的得分、状态本身、状态的动作、状态的深度、是否拿过钥匙
class Node implements Comparable<Node> {
    private double score;
    private StateObservation stateObs;
    private ArrayList<Types.ACTIONS> Actions;
    private int depth;
    public boolean GetKey;

    public Node(double score, StateObservation stateObs, ArrayList<Types.ACTIONS>
Actions, int depth,boolean GetKey) {
        this.score = score;
        this.stateObs = stateObs;
        this.Actions = Actions;
        this.depth = depth;
        this.GetKey=GetKey;
    }
```

```
    public Node(Node other) {
        System.out.println(other.stateObs==null);
        this.score = other.score;
        this.stateObs = stateObs.copy(); // 假设stateObs也需要克隆
        this.Actions = (ArrayList<Types.ACTIONS>) other.Actions.clone(); // 假设
Actions也需要克隆
        this.depth = other.depth;
        this.GetKey=other.GetKey;
        // 复制其他属性
    }
    public double getScore() {
        return score;
    }
    public StateObservation getStateObs() {
        return stateObs;
    }
    public ArrayList<Types.ACTIONS> getActions() {
        return Actions;
    }
    public int getDepth() {
        return depth;
    }

    //排序按照score启发式得分降序排序
    @Override
    public int compareTo(Node other) {
        // 降序排列
        int ok = Double.compare(other.score, this.score);
        if(ok==0){
            if(stateObs.equalPosition(other.stateObs)){
                return 0;
            }
            else return 1;

        }
        else return ok;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Node other = (Node) o;
        return stateObs.equalPosition(other.stateObs);
    }

    @Override
    public String toString() {
        return Double.toString(score);
    }
};

public class Agent extends AbstractPlayer {
```

```
    protected Random randomGenerator;

    protected ArrayList<Observation> grid[][];

    private TreeSet<Node> Todolist = new TreeSet<>();

    private boolean OK=false;
    protected int block_size,numstep=0;
    private ArrayList<StateObservation> Visited= new ArrayList<StateObservation>
();
    private ArrayList<Types.ACTIONS>  MaxAction= new ArrayList<Types.ACTIONS>();
    private Vector2d goalpos,keypos;
    public Agent(StateObservation so, ElapsedCpuTimer elapsedTimer)
    {
        randomGenerator = new Random();
        grid = so.getObservationGrid();
        block_size = so.getBlockSize();
        ArrayList<Observation>[] fixedPositions = so.getImmovablePositions();
        ArrayList<Observation>[] movingPositions = so.getMovablePositions();
        goalpos = fixedPositions[1].get(0).position; //目标的坐标
        keypos = movingPositions[0].get(0).position ;//钥匙的坐标
    }

    //检查是否走过相同的状态
    private boolean CheckifVisited(StateObservation stataObs) {
        for (int i = 0; i < Visited.size(); i++) {
            if (stataObs.equalPosition(Visited.get(i))) {
                return true;
            }
        }
        return false;
    }
    //获得两点的距离
    int GetDistance(Vector2d a,Vector2d b){
        return (int)Math.abs(a.x-b.x)/50+(int)Math.abs(a.y-b.y)/50;
    }

    //检查openlist是否走过相同的状态
    private Node contain(TreeSet<Node> Todolist,StateObservation stataObs){
        for(Node node:Todolist){
            if(node.getStateObs().equalPosition(stataObs)){
                return node;
            }
        }
        return null;
    }
    private ArrayList<Vector2d> GetBox(StateObservation stataObs){
        ArrayList<Observation>[][] observationGrid= stataObs.getObservationGrid();
        ArrayList<Vector2d> Box= new ArrayList<tools.Vector2d>();
        for(int i=0;i<observationGrid.length;i++){
            for(int j=0;j<observationGrid[i].length;j++){
                if(observationGrid[i][j]!=null){
                    for(int k=0;k<observationGrid[i][j].size();k++){
                        if(observationGrid[i][j].get(k).itype==8){
```

```
                            Box.add(observationGrid[i][j].get(k).position);
                        }
                    }
                }
            }
        }
        return Box;
    }

    //得到与box最近的坑的距离
    private double GetDistanceHole(int x,int y,StateObservation stateObs){
        ArrayList<Observation>[][] observationGrid= stateObs.getObservationGrid();
        int n=observationGrid.length,m=observationGrid[0].length;
        double sum=1e9;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                for(int k=0;k<observationGrid[i][j].size();k++){
                    if(observationGrid[i][j].get(k).itype==2){
                        sum=min(sum,GetDistance(new
Vector2d(x,y),observationGrid[i][j].get(k).position));

                    }
                }
            }
        }
        if(sum==1e9) return 0;
        else return sum;

    }

    //得到与精灵最近的额box的距离
    private double GetMinDistanceBox(StateObservation stateObs){
        ArrayList<Observation>[][] observationGrid= stateObs.getObservationGrid();
        ArrayList<Vector2d> Box= new ArrayList<tools.Vector2d>();
        for(int i=0;i<observationGrid.length;i++){
            for(int j=0;j<observationGrid[i].length;j++){
                if(observationGrid[i][j]!=null){
                    for(int k=0;k<observationGrid[i][j].size();k++){
                        if(observationGrid[i][j].get(k).itype==8){
                            Box.add(observationGrid[i][j].get(k).position);
                        }
                    }
                }
            }
        }
        double sum=1e9;
        for(int i=0;i<Box.size();i++){
            sum=min(sum,GetDistance(stateObs.getAvatarPosition(),Box.get(i)));
        }
        if(sum==1e9) return 0;
        else return sum;
    }

    //得到key的坐标集合
```

```
    private ArrayList<Vector2d> Getkey(StateObservation stataObs){
        ArrayList<Observation>[][] observationGrid= stataObs.getObservationGrid();
        ArrayList<Vector2d> Key= new ArrayList<tools.Vector2d>();
        for(int i=0;i<observationGrid.length;i++){
            for(int j=0;j<observationGrid[i].length;j++){
                if(observationGrid[i][j]!=null){
                    for(int k=0;k<observationGrid[i][j].size();k++){
                        if(observationGrid[i][j].get(k).itype==6){
                            Key.add(observationGrid[i][j].get(k).position);
                        }
                    }
                }
            }
        }
        return Key;
    }

    //启发式函数
    private double heuristic(StateObservation stateObs,boolean HasGetKey,int dep){
        double
score=0,W_0=0,W_1=100000000,W_2=100,W_3=-5,W_4_1=30,W_4_2=10,W_4_3=1000,W_4_4=-10,
W_5=-10;
        double
Dep,WinorLose=0,Nowscore=0,Distance,Box_1=0,Box_2=0,Box_3=0,Box_4=0,GG=0,MinDistan
ceBox=0;

        Dep=dep;

        if(stateObs.getGameWinner()==Types.WINNER.PLAYER_WINS){
            WinorLose=1;
        }
        if(stateObs.getGameWinner()==Types.WINNER.PLAYER_LOSES){
            WinorLose=-1;
        }


        Nowscore=stateObs.getGameScore();

        if(HasGetKey){
            Distance=GetDistance(stateObs.getAvatarPosition(),goalpos);
        }
        else{
            Distance=GetDistance(stateObs.getAvatarPosition(),keypos);
        }

        ArrayList<Vector2d> Box=GetBox(stateObs);
        ArrayList<Observation>[][] observationGrid= stateObs.getObservationGrid();
        int[][] Map=new int[observationGrid.length][observationGrid[0].length];
        for(int i=0;i<observationGrid.length;i++){
            for(int j=0;j<observationGrid[i].length;j++){
                if(observationGrid[i][j].size()!=0){
                    for(int k=0;k<observationGrid[i][j].size();k++){
                        Map[i][j]=observationGrid[i][j].get(k).itype;
                    }
```

```
                }
                else{
                    Map[i][j]=-1;
                }
            }
        }

        int ax=(int)stateObs.getAvatarPosition().x/50,ay=
(int)stateObs.getAvatarPosition().y/50;

        for(int i=0;i<Box.size();i++){

            int x=(int)Box.get(i).x/50,y=(int)Box.get(i).y/50;
            int n=observationGrid.length,m=observationGrid[0].length;

            if(x==keypos.x/50&&y==keypos.y/50&& !HasGetKey) GG=1;

            boolean is=false;
            if(ax==x&&(ay==y+1||ay==y-1)) is=true;
            if(ay==y&&(ax==x+1||ax==x-1)) is=true;
            if(ax==x-1&&ay==y-1) is=true;
            if(ax==x+1&&ay==y+1) is=true;
            if(ax==x-1&&ay==y+1) is=true;
            if(ax==x+1&&ay==y-1) is=true;
            if(!is) continue;

            if(x<n-1&&x>0){
                if(Map[x+1][y]==-1&&Map[x-1][y]==-1) Box_1++;
            }
            if(y<m-1&&y>0){
                if(Map[x][y+1]==-1&&Map[x][y-1]==-1) Box_1++;
            }

            if(x<n-1) if(Map[x+1][y]==2) Box_2++;
            if(x>0) if(Map[x-1][y]==2) Box_2++;
            if(y<m-1) if(Map[x][y+1]==2) Box_2++;
            if(y>0) if(Map[x][y-1]==2) Box_2++;

            if(x<n-1&&x>0){
                if(Map[x+1][y]==2&&Map[x-1][y]==-1) Box_3++;
                if(Map[x-1][y]==2&&Map[x+1][y]==1) Box_3++;
            }
            if(y<m-1&&y>0){
                if(Map[x][y+1]==2&&Map[x][y-1]==-1) Box_3++;
                if(Map[x][y-1]==2&&Map[x][y+1]==-1) Box_3++;
            }

            Box_4+=GetDistanceHole(x*50,y*50,stateObs);

        }

        if(HasGetKey) GG--;

        MinDistanceBox=GetMinDistanceBox(stateObs);
```

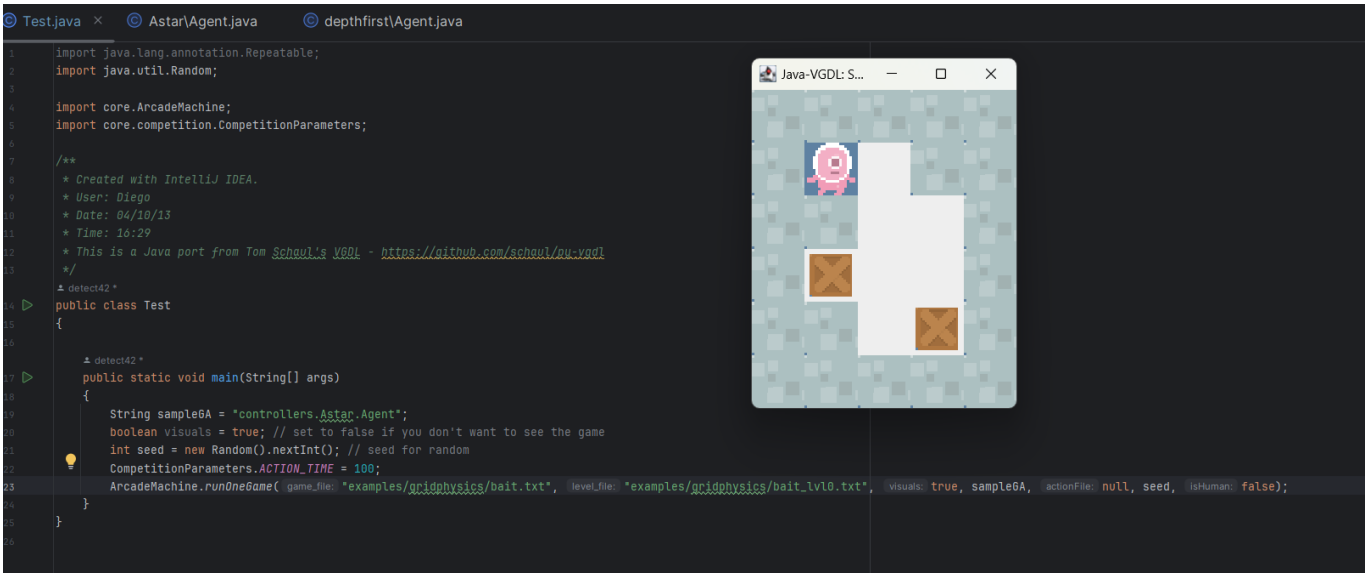
```
score=W_0*Dep+W_1*WinorLose+W_2*Nowscore+W_3*Distance+W_4_1*Box_1+W_4_2*Box_2+W_4_
3*Box_3+W_4_4*Box_4+GG*-1000000+W_5*MinDistanceBox;
        return score;
    }
    ArrayList<Types.ACTIONS> Astar(Node StartNode, ElapsedCpuTimer
elapsedTimer,int dep){
        int tot=0;
        Todolist.add(StartNode);
        while(!Todolist.isEmpty()){
            Node now=Todolist.pollFirst();
            System.out.println("score= " + now.getScore() + " depth= " +
now.getDepth() + " GetKey= " + now.GetKey + "Positon= " +
(now.getStateObs().getAvatarPosition().x/50)+","+
(now.getStateObs().getAvatarPosition().y/50)+"-------------");
            tot++;
            StateObservation stataObs=now.getStateObs();
            Visited.add(stataObs);
            ArrayList<Types.ACTIONS> actions =
now.getStateObs().getAvailableActions();
            for(Types.ACTIONS action:actions){
                StateObservation stCopy = now.getStateObs().copy();
                stCopy.advance(action);
                ArrayList<Types.ACTIONS>  AstarAction= (ArrayList<Types.ACTIONS>)
now.getActions().clone();
                AstarAction.add(action);
                if(stCopy.getGameWinner()==Types.WINNER.PLAYER_WINS){
                    OK=true;
                    return AstarAction;
                }

if(stCopy.getGameWinner()==Types.WINNER.PLAYER_LOSES||CheckifVisited(stCopy)){
                    AstarAction.remove(AstarAction.size()-1);
                    continue;
                }

                boolean NowGetKey=false;
                if(Getkey(now.getStateObs()).size()==1&& Getkey(stCopy).isEmpty())
{
                    NowGetKey=true;
                }

                Node node=contain(Todolist,stCopy);
                if(node!=null){
                    double score0=node.getScore();

                    double score1=heuristic(stCopy,NowGetKey|now.GetKey,dep+1);
                    if(score0>=score1){
                        AstarAction.remove(AstarAction.size()-1);
                    }
                    else{
                        Node Newnode=new Node(score1,stCopy,AstarAction,
dep+1,NowGetKey|now.GetKey);
```

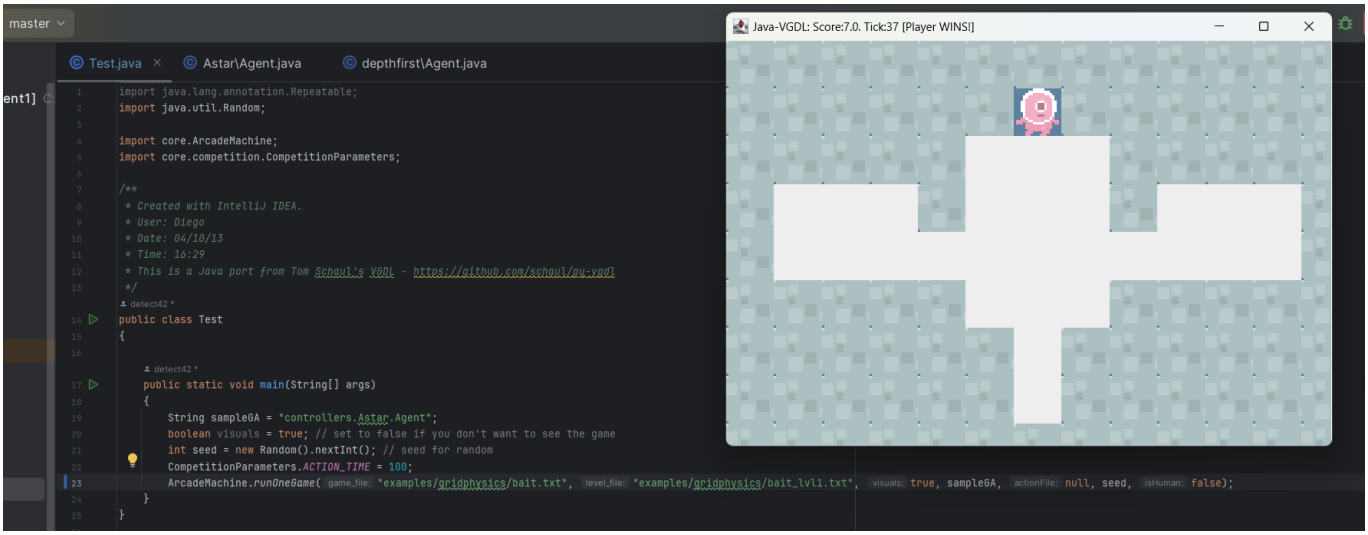
```
                        Todolist.remove(node);
                        Todolist.add(Newnode);
                    }
                }
                else{
                    Node Newnode=new
Node(heuristic(stCopy,NowGetKey|now.GetKey,dep+1),stCopy,AstarAction,
dep+1,NowGetKey|now.GetKey);
                    Todolist.add(Newnode);
                }
            }
        }

        System.out.println("GGGGGGGGGGGGGGGGGGGGG");
        return (new ArrayList<Types.ACTIONS>());

    }

    public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer
elapsedTimer) {

        if(OK==false){
            Node StartNode=new
Node(heuristic(stateObs,false,0),stateObs.copy(),new ArrayList<Types.ACTIONS>
(),0,false);
            MaxAction= Astar(StartNode,elapsedTimer,0);OK=true;
            System.out.println("AstarAction.size()= " + MaxAction.size());
            return MaxAction.get(0);
        }
        else{
            numstep++;
            return MaxAction.get(numstep);
        }
    }

    private void printDebug(ArrayList<Observation>[] positions, String str) {
        if (positions != null) {
            System.out.print(str + ":" + positions.length + "(");
            for (int i = 0; i < positions.length; i++) {
                System.out.print(positions[i].size() + ",");
            }
            System.out.print("); ");
            for(int i=0;i<positions.length;i++){
                for(int j=0;j<positions[i].size();j++){
                    System.out.println(str + " x=" +
positions[i].get(j).position.x/50 + " y=" + positions[i].get(j).position.y/50);
                }
            }
        } else System.out.print(str + ": 0; ");
    }
}
```

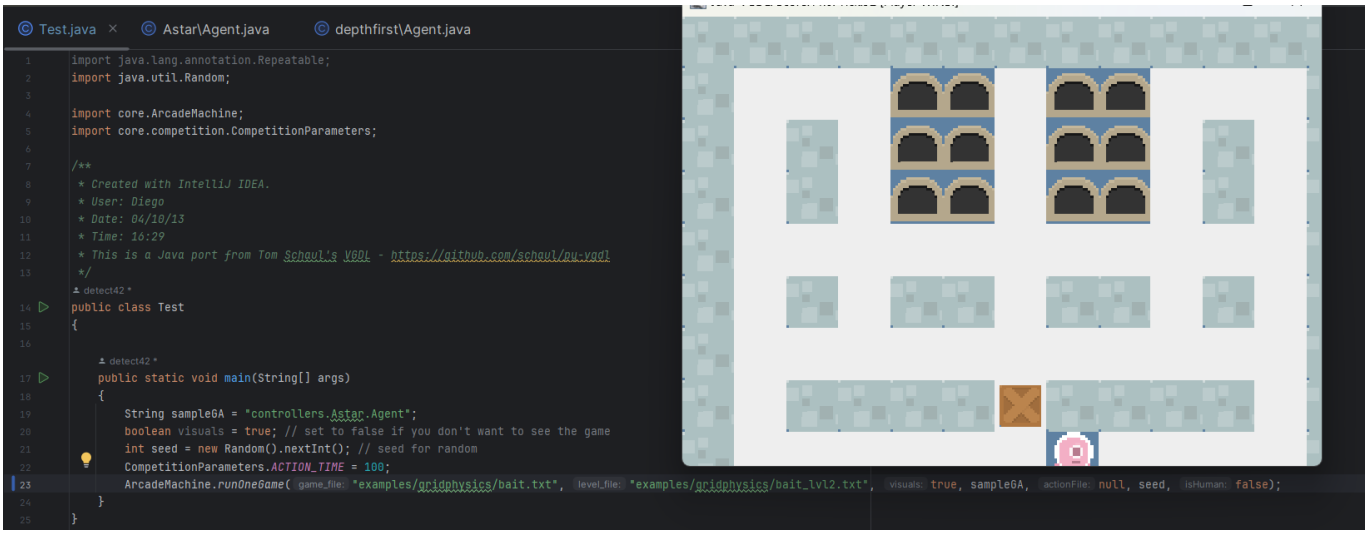
2.3.5 Algorithm Performance

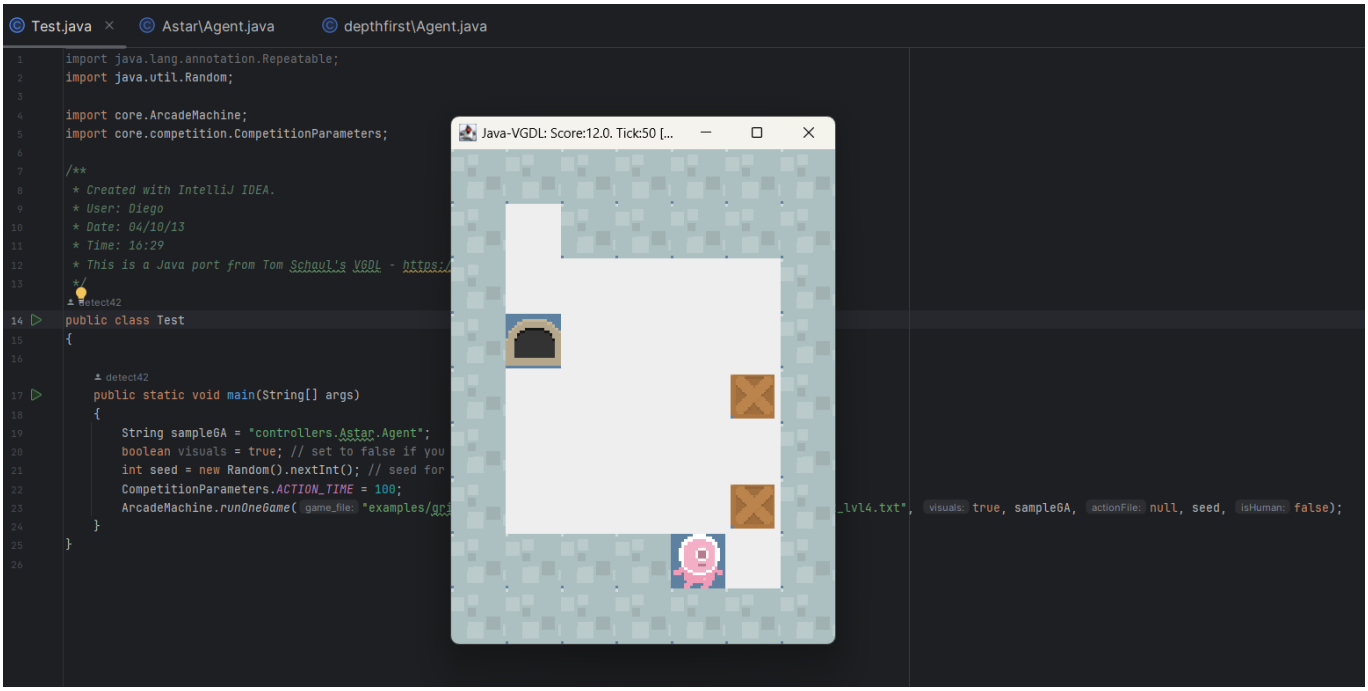
至此Astar算法实现完毕，经过测试可以在**总时间**$100ms$内通过第1,2,3,5张图。从结果上证明了参数的设定和权值具有较高的泛化性和可行性。

附图为运用Astar算法在单步100ms通过第一二三五关的截图（虽然设置的是单步，但是其实在第一步（100ms）内就找到了一条正确路径，后续不需要再搜索而是只需要读存好的actionlist就可以了）









2.4 TASK4：

2.4.1 蒙特卡洛树搜索概述

首先了解一下蒙特卡洛树搜索。

1. **初始化**：
   - 创建一个根节点，代表当前的游戏状态或问题。
   - 初始化根节点的统计信息，如访问次数和累积奖励。
2. **选择**：
   - 从根节点开始，通过一定策略选择子节点，直到达到叶子节点（尚未被扩展的节点）。
   - 选择子节点的策略通常是根据已有信息，来平衡探索未知节点和利用已知节点之间的权衡。
3. **扩展**：
   - 对于已选中的叶子节点，根据可行的行动扩展它，生成一个或多个子节点。

- 如果子节点中的某个代表游戏结束状态，那么不再扩展它，而是直接进入模拟步骤。

4. **模拟**：
   - 针对扩展的子节点，执行随机模拟或随机走子，直到达到游戏结束状态或者达到一个预定的深度限制。
   - 在模拟中通常使用随机策略，以模拟游戏状态的不确定性。
5. **回溯**：
   - 从模拟结束的节点开始，将模拟结果的奖励值（通常是游戏胜利与否的评估）反向传播回父节点，并更新父节点的统计信息，如访问次数和累积奖励。
   - 这一步鼓励树搜索在每个节点上逐渐累积关于最佳动作的信息。
6. **重复**：
   - 重复执行选择、扩展、模拟和回溯步骤，直到达到某个停止条件，例如时间限制或迭代次数。
7. **选择最佳行动**：
   - 在根节点的所有子节点中选择具有最高价值（通常是累积奖励除以访问次数）的行动作为最佳行动。
   - 返回最佳行动作为决策的结果。

蒙特卡洛树搜索通过不断地模拟和探索可能的决策路径，逐渐提高对最佳行动的估计，从而在复杂的决策问题中表现出色。实际上，只要迭代的次数足够多，其选择最优解的可能性会逼近1。

在给出的$SampleMCTS$中，其实现了一个十分简化的蒙特卡洛树搜索。

首先设定一个搜素深度和评价函数（由胜利与否和得分共同决定）。

然后在每一步中，用绝大部分时间去建树，迭代当前的最优解往下搜索，当到达搜索深度或者游戏结束时，反向传播不断更新父节点的值。

最终在时间耗尽前返回路过频率最高的那个action。（因为我们的评价函数与胜利、得分相关，即往这个方向走分多且容易胜利

以上。

2.4.2 样例代码分析

由于蒙特卡洛树搜索算法是一步一步跑的，所以在act部分，每次传入当前的状态和剩余时间的接口即可。

```
public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer)
{

    ArrayList<Observation> obs[] = stateObs.getFromAvatarSpritesPositions();
    ArrayList<Observation> grid[][] = stateObs.getObservationGrid();

    //Set the state observation object as the new root of the tree.
    mctsPlayer.init(stateObs);

    //Determine the action using MCTS...
```

```
    int action = mctsPlayer.run(elapsedTimer);

    //... and return it.
    return actions[action];
}
```

在主体部分，rollback是随机往下面走并记录得分和胜利情况。 backUp回溯过程中更新父节点的值

```
while(remaining > 2*avgTimeTaken && remaining > remainingLimit){
        ElapsedCpuTimer elapsedTimerIteration = new ElapsedCpuTimer();
        SingleTreeNode selected = treePolicy();
        double delta = selected.rollOut();
        backUp(selected, delta);

        numIters++;
        acumTimeTaken += (elapsedTimerIteration.elapsedMillis()) ;

        avgTimeTaken  = acumTimeTaken/numIters;
        remaining = elapsedTimer.remainingTimeMillis();
        //System.out.println(elapsedTimerIteration.elapsedMillis() + " --> " +
acumTimeTaken + " (" + remaining + ")");
    }
    //System.out.println("-- " + numIters + " -- ( " + avgTimeTaken + ")");
}
```

另外值得一提的部分是uct函数的实现，该函数用于判断根节点的children中最有潜力的节点。

具体来说，每个节点的权值被定义经过的总分数的平均值再加上一个小小的随机扰动。

得到可以量化的分值后就可以选择最佳子节点了。

```
public SingleTreeNode uct() {

    SingleTreeNode selected = null;
    double bestValue = -Double.MAX_VALUE;
    for (SingleTreeNode child : this.children)
    {
        double hvVal = child.totValue;
        double childValue =  hvVal / (child.nVisits + this.epsilon);


        childValue = Utils.normalise(childValue, bounds[0], bounds[1]);

        double uctValue = childValue +
                Agent.K * Math.sqrt(Math.log(this.nVisits + 1) / (child.nVisits +
this.epsilon));

        // small sampleRandom numbers: break ties in unexpanded nodes
        uctValue = Utils.noise(uctValue, this.epsilon, this.m_rnd.nextDouble());
//break ties randomly
```

```
            // small sampleRandom numbers: break ties in unexpanded nodes
            if (uctValue > bestValue) {
                selected = child;
                bestValue = uctValue;
            }
        }

        if (selected == null)
        {
            throw new RuntimeException("Warning! returning null: " + bestValue + " : "
    + this.children.length);
        }

        return selected;
    }
```

3 结语

在本次作业中，我实现了深度优先搜索，深度限制+Astar,Astar完全版，已经了解了蒙特卡洛算法。

从结果上来看，发现Astar算法的效果最好，而且在不同的地图中，只需要调整一下权值就可以很好的适应不同的地图。

而所给的样例蒙特卡洛算法的效果并不好，可能是因为其实现的比较简单，而且没有考虑到钥匙箱子的交互问题。

通过本次作业的学习，我对搜索算法有了不一样的认识，也对搜索算法的实现有了更深的理解。（debug呜呜呜

以上。