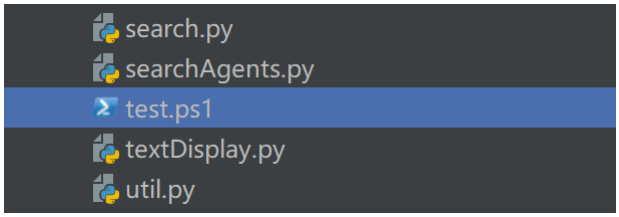
实验报告--pacman

201300086 史浩男

实验报告应包括但不限于：任务叙述+解决方法+实验效果+必要分析。以及复现实验效果的操作说明

代码测试说明

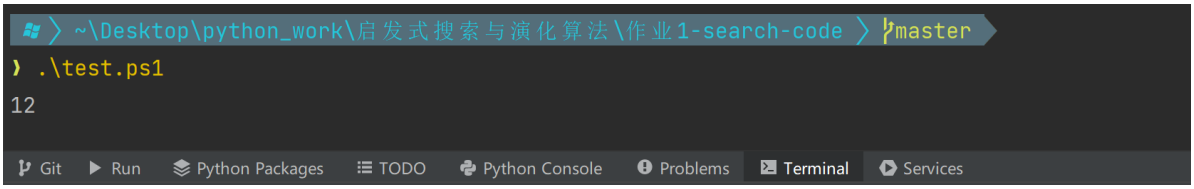
为了方便测试，我在项目目录里加入了脚本`test.ps1`



在项目目录中打开脚本，输入两个数字：第一个数字代表测试第几个任务，第二个数字代表测试第几个地图

如图片中的”12“代表测试第一个任务的第二个地图，对应参数：

```
python pacman.py -l openMaze -p SearchAgent -a fn=astar,heuristic=myHeuristic
```



任务一

一、任务叙述

在`search.py`中的空函数`aStarSearch`中实现A* 的图搜索代码和启发式函数代码。

自行选择并编写效果更好的启发式函数，并在报告中对两种启发式函数的效果进行分析

二、实现启发式函数

`util.py`中已经封装好了计算曼哈顿距离的函数，我猜测这就是我们在此任务中需要的启发式函数

```
def myHeuristic(state, problem=None):
#采用封装好的曼哈顿距离
    return util.manhattanDistance(state, problem.getGoalState())#自己添加的
getGoalState()
    #return util.EuclideanDistance(state, problem.getGoalState())#尝试欧式距离
```

除此之外，我还在`util.py`中添加了计算欧式距离的函数，通过比较效果，我发现曼哈顿距离的效果明显优于欧式距离，证明了在这样一个`pacman`只能横纵移动的游戏中，一维曼哈顿距离是更有效的度量方式

三、实现A*图搜索代码

`util.py`中已经封装好了两个优先队列`PriorityQueue 和``PriorityQueueWithFunction`

为了使代码实现更为简介，我选择了带权函数的优先队列`PriorityQueueWithFunction`。因此，我将cost函数和启发式函数合并起来，函数定义在`aStarSearch`内：

```
#A*自定义g+h
def astar_priorityFunction(item):
    state, actions = item
    g = problem.getCostOfActions(actions)
    h = heuristic(state, problem)
    return g + h
```

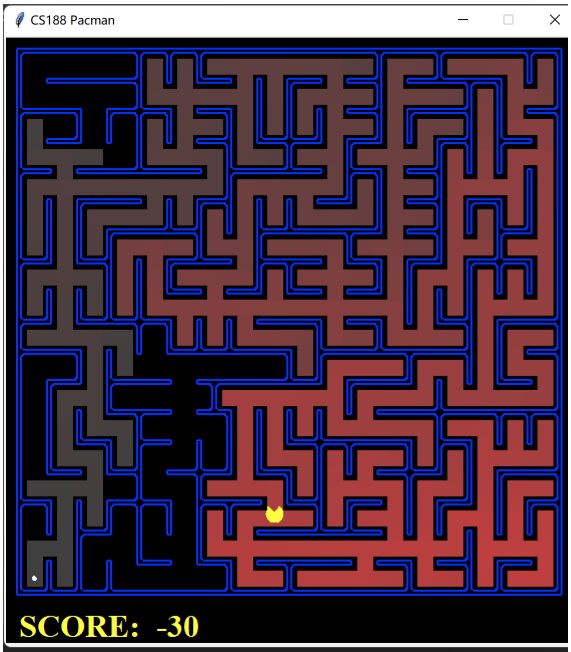
并将合并后的总cost函数传递给优先队列（不能传递参数，debug好久……）

```
stateQ= util.PriorityQueueWithFunction(astar_priorityFunction)#只传函数名，不传参
```

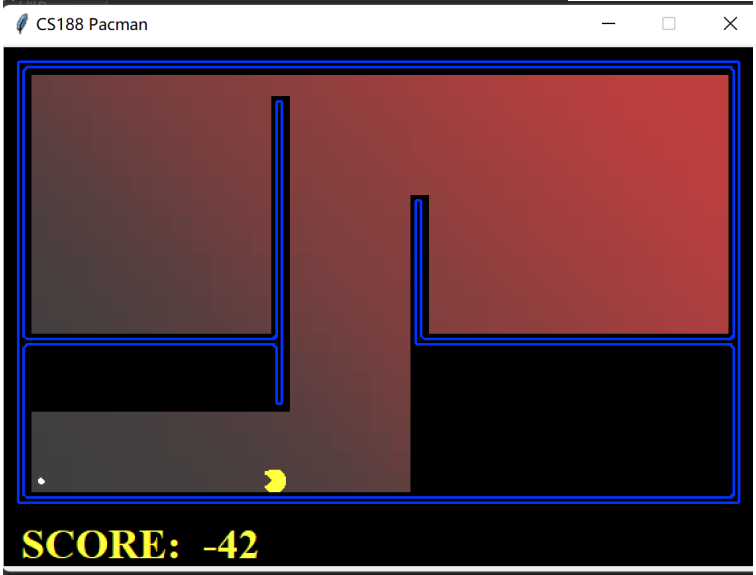
每次从队列中取出当前state和action，最后返回一个actions列表，队列完整实现如下：

```
# 初始即目标
    if problem.isGoalState(problem.getStartState()):
        return []

    actions=[]
    visited=[]
    stateQ= util.PriorityQueueWithFunction(astar_priorityFunction)
    stateQ.push(item=(problem.getStartState(),actions))

    while stateQ.isEmpty()==False:
        nowstate,nowactions=stateQ.pop()
        if problem.isGoalState(nowstate):
            return nowactions
        if nowstate not in visited:
            visited.append(nowstate)
            nowsuccessors=problem.getSuccessors(nowstate)
            for nextstate,action,cost in nowsuccessors:
                stateQ.push((nextstate,nowactions+[action]))
    return actions
```

四、效果




```
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 549
```




```
) .\test.ps1
12
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 535
```


```
) .\test.ps1
13
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 19 in 0.0 seconds
Search nodes expanded: 53
```

任务二

一、任务叙述

在尽可能少的步骤中吃掉所有的豆子，一个解决方案被定义为一条收集 `Pacman` 世界中所有食物的路径。你需要完成 `searchAgents.py` 中 `foodHeuristic` 函数的编写。期待你实现满足admissible和consistent的启发式函数

二、启发式函数探索与分析

我一共探索了9种启发式函数：

p.s.每个方法后面的第一个三元组代表了分别在三个地图上测试的 `expand node` 大小，x代表没有跑出结果。第二个三元组代表在三个地图上的得分 `score`

1、原始（14，707，x）

2、最远点曼哈顿（9，189，x）

3、到所有残存食物曼哈顿距离总和/总食物数（10，302，x）

4、到所有残存食物曼哈顿距离总和/残存食物数（10，209，x）

5、到所有残存食物曼哈顿距离总和/（残存食物数/2）（7，302，x）

6、到所有残存食物曼哈顿距离总和/（总食物数/2）（7，209，x）

分析：以上的6种方法，都只能在前两个测试图中跑出结果，最大的地图上会卡住

7、到所有残存食物曼哈顿距离总和（6，46，48）（534，603，779）

分析：于是我尝试了这种明显不满足admissible的启发式函数，解决了最大地图上会卡住的问题。但这种方法不能保证获得最高得分，比如在第二个地图上得分603，而不是最高分614

8、剩余食物数量（12，106，31）

9、剩余食物数量+到最近食物距离（7，70，31）

分析：我终于想到了最适合本问题的启发式函数。测试证明方法9的expand node最小，得分最高

```
    #方法9：剩余食物数量+距离最近食物的距离
    lst = list(map(lambda x: util.manhattanDistance(position, x),
foodGrid.asList()))
    if len(lst) > 0:
        return len(lst) + min(lst)
    else:
        return 0
```

三、admissible

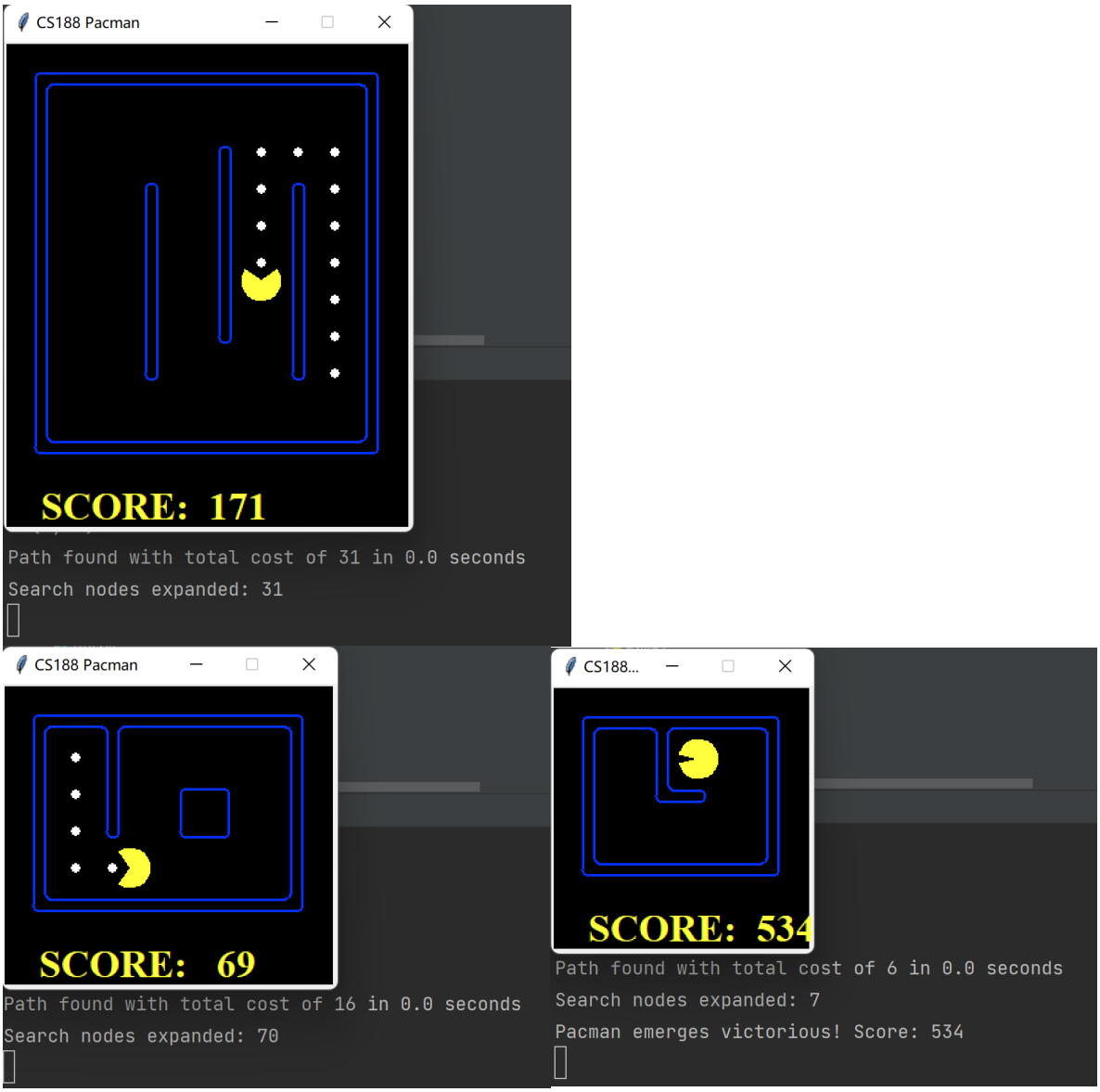
"剩余食物数量+到最近食物距离"这个方法显然是admissible的。

”剩余食物数量“是吃到食物的所有action个数的下限

”到最近食物距离“是所有没吃到食物的所有action个数的下限

所以"剩余食物数量+到最近食物距离"是总步数的下限，任何时刻都满足，符合admissible条件

四、效果



任务三

只有第一个小地图可以成功运行，测试时运行脚本后输入”31“

后两个地图会卡住

我在框架代码中没有发现关于大豆子和幽灵位置和状态的调用接口，也就是说任务三中高效合理的算法目前无法完成

致谢：

感谢张毅霖同学对我在探索任务二中启发式函数时的一些启发