HSEA-HW1

匡亚明学院　张子谦　　191240076　　191240076@smail.nju.edu.cn

本人承诺该实验全程由本人独立完成，无抄袭或给予他人抄袭行为，代码在截至日期后上传至本人github

摘要：Pacman游戏是一个操纵avator吃掉所有food（特定环境下，需要躲避幽灵，获取尽可能高分数）的游戏。本实验在已有的代码框架下，以A*算法为基础，针对PositionSearch和FoodSearch这两类问题以及不同的游戏地图，进行实验，观察算法表现，提高算法性能。对于PositionSearch，以Manhattan距离作为启发式函数，保证了admissible和consistent，并进一步尝试了通过结合simulate trajectories与惩罚项，构建新的启发式函数，对openMaze等复杂的情况，很大程度上减少了expand node的数目。对于FoodSearch，分别尝试了到食物的最远Manhattan距离，到食物的最远真实距离，以及到食物的最近距离+该食物到其他食物的最远Manhattan距离，以及引入了地图剩余food的数量这四种启发式函数设计综合比较性能和时间复杂度，最终的效果提升十分显著；同时为应对更加复杂的情况，新构建了一个限定搜索深度的OneStepFoodSearchAgent，在找到最优解前，每步动作都进行一定的A*搜索，以应用于第二项与第三项任务。

1、引言

如图所示，将游戏中的每个状态视作节点，action视作边，转移模型由游戏逻辑定义，Pacman便可视作一个典型的路径搜索的模型。本实验拟以A*算法为基础，分别尝试解决PositionSearch和FoodSearch这两类问题，并在实际的游戏环境中进行实验。

2、实验内容

Task0：理解代码框架

- **util.py**中固定了随机数种子，定义了堆、栈、优先队列、Counter字典等数据结构，以及一些归一化、采样等辅助函数。
- **game.py**中

  *Agent*相当于Abstract类，用于定义游戏实体（即吃豆人和幽灵），通过特定的policy选取action。

  *Configuration*标识agent的位置(x,y pair)与当前朝向(direction)，并通过generateSuccessor方法根据action生成后继configure。

  *AgentState*标识agent的状态，包括configure, speed等数据。

  *Grid*是方格世界，记录了每个2dim的位置信息。

  *Actions*定义了一些agent执行action前后的辅助函数。

  *GameStateData*记录了游戏过程中的状态信息。

  *Game*控制游戏进行，主要根据run方法跑一局游戏，期间依次由agent 产生action并执行，通过本身定义的transition对游戏状态进行更新。
- **pacman.py** 将game.py中的一些类进行了一定的封装，*GameState*允许获取游戏中agent的legalactions，产生state的后继（transition model），获取分数等。*classicGameRules*, *PacmanRules*, *GhostRules*描述了agent与environment交互的规则。

我们需要进行完成的内容包括 **search.py**与 **searchAgent.py**两部分，通过命令行参数指定**searchAgent.py**中的Agent进行加载，Agent通过fn，heuristic参数指定**search.py**中的搜索函数（A*）和所采用的启发式函数

Task1：PositionSearch

- 完成**aStarSearch()**函数

  注意到参数中包括了problem，针对Task1，主体为**searchAgent.py**中的*PositionSearchProblem*，即探路问题，按照搜索问题的标准定义了initial state，goal test, path cost, actions(默认)以及transition model(由games.py中接口辅助实现，封装在getSuccessor方法中，返回legal的后继节点)

  基于Graph-Search，伪代码如下：

```
class Node:
    def __init__(self, state, cost, actions):#节点存储(位置，路径代价信息，  路径（便于返回））
        self.state = state
        self.cost = cost
        self.actions = actions
def aStarSearch(problem, heuristic=nullHeuristic):
    node = Node(init_state, 0, [])#初始状态
    frontier = priority_queue()
    frontier.update(node, node.cost + heuristic(node.state))
    #reached = dict()
    #reached[node.state] = heuristic(node.state)
    reached = set()
    while !frontier.empty():
        node = frotier.pop()
        if isgoal(node.state):
            return node.actions
        if node.state in reached:
            continue
        reached.add(node.state)
        for child in problem.getSuccessor(node.state):
            s = child.state
            c = node.cost + child.cost#child.cost是单步动作的代价
            actions = node.actions + [child.action]
            frontier.update(Node(s, c, actions), c+heuristic(s))
            """if s not in reached.keys() or c + heuristic(s) < reached[s]:
                reached[s] = c + heuristic(s)
                frontier.update(Node(s, c, actions), c+heuristic(s))"""
```

  （按照PPT中UniformCost Search的框架，对于reached的判断如注释所示，但这里根据个人习惯进行了一些修改。）

  完成后可以用nullHeuristic，(h(s)=0)，也即Uniform Cost Search进行探索尝试，测试结果如下：

```
C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l smallMaze -p SearchAgent
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 19 in 0.0 seconds
Search nodes expanded: 92
Pacman emerges victorious! Score: 491
Ending graphics raised an exception: 0
Average Score: 491.0
Scores:        491.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l openMaze -p SearchAgent
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 682
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l bigMaze -p SearchAgent
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

- 完成**myHeuristic()**函数

  先进行简单的尝试，用当前state到目标位置的Euclidean距离和Manhattan距离分别作为启发式函数，其consistent性质显然，最优性得以保障。分别得到expand的结点数对比如下：

	nullHeuristic	Euclidean	Manhattan
large	620	557	549
open	682	550	535
small	92	56	53

  可见相较于nullHeuristic，有了一部分提升，但不够显著，而Euclidean和Manhattan两种启发式函数区别不大。下尝试对启发式函数进行一定的修改。

  考虑到具体问题，因为wall的存在，导致启发式函数$h(s)$和实际$h^*(s)$在尤其是largemaze的地图中差距较大，而无论是Euclidean还是Manhattan距离，都忽视了wall的存在，这样的设计缺乏一定的合理性。于是有两种改进的思路。

```
C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l smallMaze -p SearchAgent -a fn=astar,heuristic=myHeuristic
[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
Pacman emerges victorious! Score: 491
Ending graphics raised an exception: 0
Average Score: 491.0
Scores:        491.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l openMaze -p SearchAgent -a fn=astar,heuristic=myHeuristic
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.1 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l bigMaze -p SearchAgent -a fn=astar,heuristic=myHeuristic
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Ending graphics raised an exception: 0
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

一方面，可以尝试调整启发式函数的权重，对于priority_queue，评判标准为$g(s)+\gamma*h(s)$，但这样无法保证路径的最优性，通过尝试，当参数小于1时，最优性虽能保证，但是expand的节点个数更多了，性能并未得到提升。而当参数大于1时，虽然expand节点数有了下降，尤其是在openMaze的情形中，下降至138个node，但也在该地图上从最优的456分降到了450分。具体情况如表格所示。（以Manhattan启发式函数为例，元素为分数-节点对）

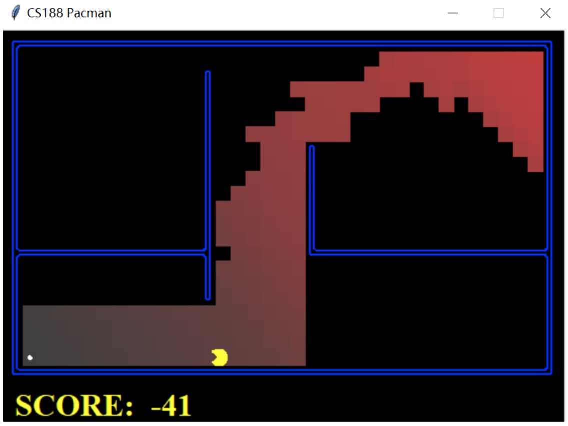
	0.8	**1**	**2**
large	300/300 - 560	300/300 - 549	300/300 - 510
open	456/456 - 555	456/456 - 535	450/456 - 138
small	491/491 - 57	491/491 - 53	491/491 - 55

另一方面，考虑wall因素，尝试simulate一些路径（忽视wall的阻碍作用），即随机的采取向x方向或y方向走一步，在路径中遇到wall会给予一些penalty惩罚项。最后对simulate的所有路径取最小值作为当前状态的启发式函数值（由于是找最优路径，相当于确定性策略，所以取最小而不是平均）。

```
def simulate(state, problem, penalty):
    goal_x, goal_y = problem.goal
    now_x, now_y = state
    cost = manhattan(goal, state)
    while now_x != goal_x and now_y != goal_y:
        r = random.randint(0, 1)
        if r == 0:
            now_x - > goal_x
        else:
            now_y - >goal_y
        if problem.walls[now_x][now_y]:
            cost += penalty
    while now_x != goal_x:
        now_x - > goal_x
        if problem.walls[now_x][now_y]:
            cost += penalty
    while now_y != goal_y:
        now_y - >goal_y
        if problem.walls[now_x][now_y]:
            cost += penalty
    return cost
```

先进行简单的尝试，只模拟一次(simple_simulate函数对应实现），沿着x方向走到goal_x的位置，再沿着y方向走到goal_y的位置，遇到wall，增加3的penalty，测试下三张地图都得到了最优解，拓展节点数分别为519, 516，61。除了smallMaze地图，节点数都有一定的下降。

下一步是完整的实现，每个状态设置simulate frequency为30， random.seed(7)(用于复现) penalty调整为5，三种地图都达到了最优解，同时expand节点数目也有了下降，如下图所示，expand节点数分别为 **498**， **203**， **40**。




```
C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l smallMaze -p SearchAgent -a fn=astar,heuristic=myHeuristi
[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: 40
Pacman emerges victorious! Score: 491
Average Score: 491.0
Scores:        491.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l openMaze -p SearchAgent -a fn=astar,heuristic=myHeuristic
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.5 seconds
Search nodes expanded: 203
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l bigMaze -p SearchAgent -a fn=astar,heuristic=myHeuristic
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 1.0 seconds
Search nodes expanded: 498
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

但是我们知道，这种依赖于模拟轨迹的启发式函数在复杂的地形下也无法保证最优。但相较于调整权重，个人认为这种模拟的方法更可靠一些。一方面，权重的大小可能要根据实际地图进行不断的调整，不具有自适应性，会比较繁琐；另一方面，对于simulate的情况，当采样充分大时，其几乎能保证最优，同时减少expand节点数目。对于simulate产生的时间增加问题，也可以通过固定每次simulate的步数，到达指定步数后再次通过简单的启发式函数返回估计值，以解决时间复杂度过高的问题。

尝试将simulate的frequency提高，但性能的提升并不显著。由于simulate的方法仍然是基于Manhattan距离进行的改进，所以对于largeMaze这种地图较为复杂的情况，expand节点个数基本是在500左右，很难通过模拟次数的增加降低，且耗时还会更长，故保留参数rollout30次。

Task2：FoodSearch

task2需要找到一条路径，以最少的步数，吃掉地图中的所有豆子。先尝试用Manhattan距离，此时goal是吃掉所有的豆子，显然无法直接用于计算Mahattan距离，所以尝试取距离当前位置距离最远的豆子到此时位置的manhattan距离作为启发式函数。

对于第三个测试，其expand的节点数目过于庞大，显然是不合理的，故尝试改变启发式函数，由于当前情境下有多个食物，所以可以尝试直接探索出到最远食物的路径长度，作为启发式函数（显然admissible且consistent），不过由于每一步都要对所有的food做一次uniform search(这里cost恒为1的情况下等价于bfs)，此时耗时较大，但相应的，拓展的节点数目也大大降低。

在代码编写过程中，关于真实距离的计算，可以新生成一个PositionSearchProblem，通过之前的astar搜索，求解，不过由于该方法是在探索时判断是否为goal，所以重新编写了bfs函数，在expand过程中直接判断。因为每一步的cost相等，所以其最优性也可以保证。时间由1425.1sec降为800sec左右（视具体CPU有所波动，作为参考，实验所用设备型号为Intel Core i7-11800H @ 2.30GHz）。

```
C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l  Search3  -p AStarFoodSearchAgent
Path found with total cost of 31 in 811.4 seconds
Search nodes expanded: 52991
Pacman emerges victorious! Score: 779
Average Score: 779.0
Scores:        779.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l  Search2  -p AStarFoodSearchAgent
Path found with total cost of 16 in 0.1 seconds
Search nodes expanded: 138
Pacman emerges victorious! Score: 614
Average Score: 614.0
Scores:        614.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l  Search1  -p AStarFoodSearchAgent
Path found with total cost of 6 in 0.0 seconds
Search nodes expanded: 6
Pacman emerges victorious! Score: 534
Ending graphics raised an exception: 0
Average Score: 534.0
Scores:        534.0
Win Rate:      1/1 (1.00)
Record:        Win
```

除了直接计算最远food的距离，也可以尝试通过bfs找到最近的food，然后计算该food与距离最远food的manhattan距离，两者相加，作为启发式函数的返回值。对于单个状态的启发式函数计算时间优于trueDistance，但对于Search3的特殊环境，导致manhattan距离的计算与真实距离总是由较大的偏差，所以expand的节点数较大，由此时间为853.2sec，也大于trueDistance方法。

```python
def foodHeuristic(state, problem):
    position, foodGrid = state
    foodlist = foodGrid.asList()
    if not foodlist:
        return 0
    #distances = []
    #for foodpos in foodlist:
        #1、manhattan距离
        #dist = abs(position[0] - foodpos[0]) + abs(position[1] - foodpos[1])
        #2、trueDistance
        #dist = uniformSearch(position, foodpos, problem.startingGameState)
        #dist = bfs(position, foodpos, problem.walls)
        #distances.append(dist)
    #return max(distances)
    #3、true+manhattan
    def bfs_nearest():
        """bfs返回最近的food_pos和cost"""
        return nearest_food_pos, path_cost
    nearest_food, nearest_cost = bfs_nearest()
    dist = 0
    for food in foodlist:#计算manhattan距离
        dist = max(dist, abs(food[0] - nearest_food[0]) + abs(food[1] -
nearest_food[1]))
    return nearest_cost + dist
```

	nullHeuristic	Manhattan	trueDistance	true+manhattan
Search1	14	9	6	7
Search2	707	189	138	93
Search3	\	114949	52991	75818

上述三种启发式函数都仅仅考虑到了到食物的距离，而没有考虑到食物的数量的变换，于是在启发式函数中添加了食物数量这一项，与最远manhattan相加，效果如下，提升十分显著。与trueDistance结合后效果也极佳，节点数分别为 **6， 105， 45**

```
C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l  Search1  -p AStarFoodSearchAgent
Path found with total cost of 6 in 0.0 seconds
Search nodes expanded: 6
Pacman emerges victorious! Score: 534
Average Score: 534.0
Scores:        534.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l  Search2  -p AStarFoodSearchAgent
Path found with total cost of 16 in 0.0 seconds
Search nodes expanded: 66
Pacman emerges victorious! Score: 614
Average Score: 614.0
Scores:        614.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l  Search3  -p AStarFoodSearchAgent
Path found with total cost of 31 in 0.0 seconds
Search nodes expanded: 53
Pacman emerges victorious! Score: 779
Ending graphics raised an exception: 0
Average Score: 779.0
Scores:        779.0
Win Rate:      1/1 (1.00)
Record:        Win
```

尽管尝试了上述启发式函数设计，降低了expand节点数目与时间，但由于SearchAgent定义，在registerInitalState中总会默认进行探索到游戏结束，将actions存在列表中，当调用getAction时，从中依次的返回。这样的情况对于所以考虑每次进行一定次数的探索，如果搜索到最终的目标状态就保存并存入actions列表，之后从中依次返回；如果到达规定的expand次数后仍然没有达到goal state，则返回当前actions序列中的第一个，待到下一次调用getAction时继续探索。这样的方法将会更加适用于复杂的环境（例如task3）

具体的实现，在searchAgent.py中由新创建的OneStepFoodSearchAgent类与OnStepFoodHeuristic函数中可以查看，cmd键入命令（**注意修改此时Agent已修改，不是AStarFoodSearchAgent，该Agent的默认参数设置与task3适配，如果要运行需要修改self.max_depth和self.heuristic参数为被注释掉的代码**）和效果如下，耗时同样很短，且在三个任务中都达到了最优的探索效果，在Search3中只expand了150个节点，虽然相较于引入foodcount的astar算法性能差一些，但显然是更普适一些的算法。

```
C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l  Search1  -p OneStepFoodSearchAgent
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Search nodes expanded:  12
Pacman emerges victorious! Score: 534
Average Score: 534.0
Scores:        534.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l  Search2  -p OneStepFoodSearchAgent
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Search nodes expanded:  188
Pacman emerges victorious! Score: 614
Ending graphics raised an exception: 0
Average Score: 614.0
Scores:        614.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l  Search3  -p OneStepFoodSearchAgent
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Search nodes expanded:  150
Pacman emerges victorious! Score: 779
Ending graphics raised an exception: 0
Average Score: 779.0
Scores:        779.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Task3

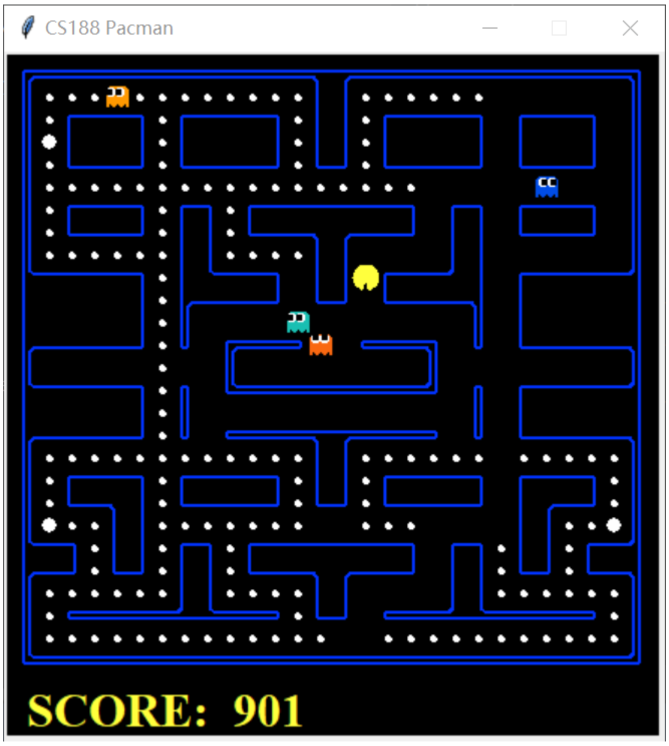
task3的运行需要先将

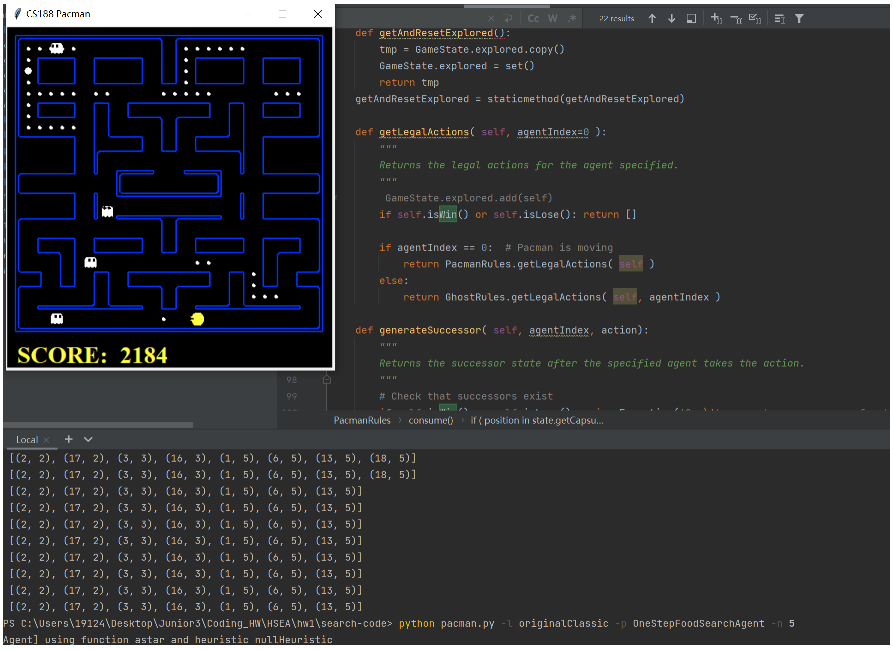
原本的AstarSearchAgent在task3这样的庞大的状态空间要进行一次搜索所需的时间复杂度是极高的，只在第一个环境中能运行，且表现一般，无法做到躲避幽灵。所以使用Task2中写的OneStepFoodSearchAgent进行探索，但表现依旧不够好，一方面是搜索空间大，很难在若干步后找到goal state，另一方面针对foodsearch的agent完全没有考虑被幽灵追击以及能量食物的状态，所以下尝试在OneStepFoodSearchAgent中尝试实现：

```
C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Search nodes expanded:  14
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Pacman died! Score: -180
Average Score: -180.0
Scores:        -180.0
Win Rate:      0/1 (0.00)
Record:        Loss

C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code>python pacman.py -l
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Pacman died! Score: -402
Average Score: -402.0
Scores:        -402.0
Win Rate:      0/1 (0.00)
Record:        Loss
```

首先，加入了对游戏胜利or失败的判断，此时Agent已经会简单的躲避，不会为了food完全忽略幽灵的追击。

进一步根据state判断food和胶囊，以及getScore方法，综合获得启发式函数，但此时pacman会出现反复横跳的行为（如下图所示位置处），尝试加入eps参数，按照epsilon-greedy策略去探索路径。调整后，偶尔会获得很高的分数。





最后在三个环境先分别测试得到的分数如下（同样，由于没有设置random.seed()，重新运行可能结果有一定出入）：

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code> python pacman.py -l originalClassic -p OneStepFoodSearchAgent -n 5
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Pacman died! Score: 1719
Pacman died! Score: 208
Pacman died! Score: 222
Pacman died! Score: 196
Pacman died! Score: 1048
Average Score: 678.6
Scores:        1719.0, 208.0, 222.0, 196.0, 1048.0
Win Rate:      0/5 (0.00)
Record:        Loss, Loss, Loss, Loss, Loss
```

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code> python pacman.py -l minimaxClassic -p OneStepFoodSearchAgent -n 5
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Pacman emerges victorious! Score: 516
Pacman emerges victorious! Score: 516
Pacman emerges victorious! Score: 516
Pacman emerges victorious! Score: 516
Pacman died! Score: -494
Average Score: 314.0
Scores:        516.0, 516.0, 516.0, 516.0, -494.0
Win Rate:      4/5 (0.80)
Record:        Win, Win, Win, Win, Loss
```

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw1\search-code> python pacman.py -l powerClassic -p OneStepFoodSearchAgent -n 5
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Pacman emerges victorious! Score: 2111
Pacman died! Score: 315
Pacman died! Score: -446
Pacman died! Score: -374
Pacman died! Score: -321
Average Score: 257.0
Scores:        2111.0, 315.0, -446.0, -374.0, -321.0
Win Rate:      1/5 (0.20)
Record:        Win, Loss, Loss, Loss, Loss
```

4、总结

实验中实现了Astar算法，针对位置搜索，食物搜索问题，分别设计启发式函数，前者主要通过Manhattan距离等度量方式获得，尝试通过simulate结合一定的penalty设计启发式函数，虽然无法保证最优，仍然获得了较好的效果。后者将距离度量与food的数量结合，大大减少expand结点的数目。针对task3，另外设计OneStepAgent，在一定expand次数后停止搜索，结合state.Score在不同环境下运行，部分情况下能获得较好的效果，但仍然有待改进。