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，以演化算法作为基础。针对PositionSearch，FoodSearch，Classic这三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，引入Exploration reward， Population diversity等方式，解决了陷入local optimal的方式，在三个地图都能够完成任务。对于FoodSearch，将food数量以及Position综合考量，修改了fitness函数，在三张地图中都达到了较好的效果。最后针对Classic问题，减少actionDim以应对幽灵这一类动态变化的因素，同时加入了避免幽灵，状态score等因素计算fitness，在一些任务中会发挥出较好的性能。

1、引言

将游戏中的可执行动作序列视作个体，通过游戏规则(这里类似于domain knowledge)，构建评估函数，计算fitness。从而可以将Pacman建模为演化算法可以实施的问题。本实验通过演化算法，尝试解决PositionSearch，FoodSearch和Classic三种环境种的任务。

2、实验内容

Task1

下面是第一次的尝试，由于最终效果很不理想，所以只进行简单的概述。在1.3中详细阐释Search任务的最终版本。

1.1 部件设计

Representation

action space是离散的5个动作（向east, west, north, south四个方向前进以及stop），但stop认为并没有什么帮助，所以只考虑四种前进，通过integer表示。（速度按照游戏设定默认为1.0不进行修改）每个解包括N个动作，故表示为$\{0,1,2,3\}^N$的一个向量。解码时，如果在N个动作执行前已经到达终点，则后续不必考虑。（在hw1中测试过，pacman吃掉豆子后会直接结束游戏）

Population size

初始化为40个，actionDim修改为20

Recombination

由于游戏中动作的一致性，所以采取one-point crossover

Mutation

简单的random setting

Fitness Evaluation

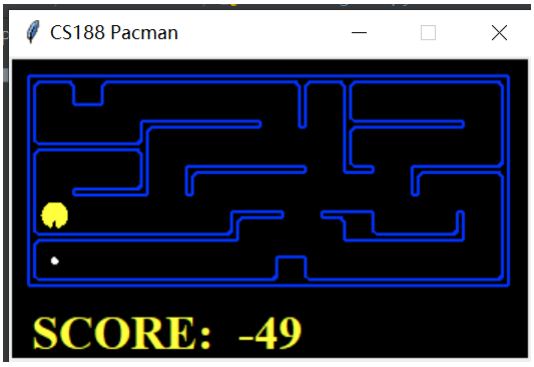
考虑positionHeuristic，值越小，fitness越好。

1.2 整体流程

通过self.actions存储一系列动作，如果self.actions存有信息，直接取用，否则通过调用getActions生成当前局面的新的actions。具体到getActions函数，是演化算法的具体流程：(i)、计算当前population的fitness；(ii)根据fitness进行parent selection；(iii)通过mutation和recombination形成子代；(iv)通过survivor selection决定下一代的population。

这里parent selection根据fitness的值通过torch.distribution.Categorical进行采样，子代按照fitness based的方法，取最大的popSize个。

按照上述方法实现后，即使在smallMaze上也很难吃到食物，pacman往往会陷入图示的local optimal，且无法逃出。对于bigMaze的复杂情况，更是难以处理，即使成功，分数也一般是负数。



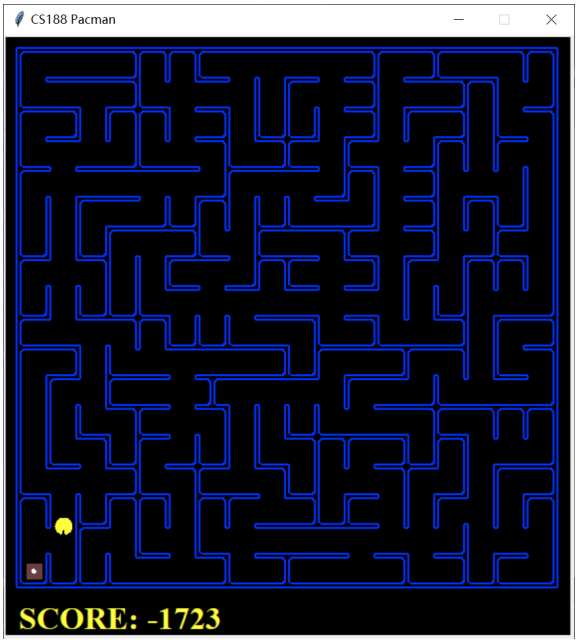
1.3 My Final Version

在尝试了较多种方法后，（期间一度被这个local optimal整破防），最终实现如下所述，首先是在small Maze和open Maze上的效果。

```
Pacman emerges victorious! Score: 249
Average Score: 342.2
Scores:        470.0, 303.0, 261.0, 428.0, 249.0
Win Rate:      5/5 (1.00)
Record:        Win, Win, Win, Win, Win
```

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw2\search-code-2> python pacman.py -l openmaze -p EvolutionSearchAgent -n 5
Pacman emerges victorious! Score: 448
find the target
Pacman emerges victorious! Score: 371
find the target
Pacman emerges victorious! Score: 413
find the target
find the target
Pacman emerges victorious! Score: 449
Average Score: 425.0
Scores:        448.0, 371.0, 413.0, 444.0, 449.0
Win Rate:      5/5 (1.00)
Record:        Win, Win, Win, Win, Win
```

且能够在bigmaze中完成target（虽然分数丢人orz）



1、fitness的计算

单纯考虑位置，往往会陷入local optimal，所以对于一个individual(动作序列)，我们综合考虑以下指标：1、执行此动作序列后最终位置与food的manhattan距离。2、此序列中是否有往返的行为（即先向左，再向右；先向上，再向下），如果有这样的无效行为，则会减少fitness（代码中是如果没有往返，就增加一些fitness）3、是否尝试跨越block，如果该动作尝试跨越block，那么会扣除一定的分数。4、path reward，这一部分会在第二部分 exploration reward中进行介绍。

getFitness函数（简化版）如下：（由于框架本身默认撞上墙壁的动作是异常，所以这里记录下来，在最后返回时将这些动作替换为STOP）

```
def getFitness(self, problem, individual):
    s = problem.getStartState()
    if s not in self.memory_pool.keys():
        self.memory_pool[s] = 1
    for i, ac in enumerate(individual):
        if problem.isGoalState(s):
            done =True
        #1、if back and forth ?
        if i+1 < len(individual) and not is_back_and_forth(individual[i],
individual[i+1]):
            no_back_forth += 1
        get nextx, nexty
        #2、if cross the wall?
        if problem.walls[nextx][nexty]:
            block += 1
            stop_idx.append(i)
        else:
            s = (nextx, nexty)
            if s not in self.memory_pool.keys():
                self.memory_pool[s] = 1
            path_reward += 5 / self.memory_pool[s]
            #path_reward += 10 / self.memory_pool[s] #for big maze
    f = 1. * path_reward + 5. * no_back_forth - 10. * block
    f -= 2 * positionHeuristic(s)
    return f, done, stop_idx, s
```

2、Exploration reward

这一部分的引入对解决bigmaze问题和逃出local optimal都起到了很大的作用（第三部分也同样起到了这样的作用）。对于每个格点位置，我们通过一个memory pool（以字典方式）存储在实际过程中被访问的次数，定义每个格点的reward为 initial_reward / visited_cnt。这里initial_reward定为了5，在bigmaze中设定为10。对于每个individual，它的fitness会将执行该序列得到的reward之和作为path_reward加入fitness中。

当然reward的设置也可以是多样的，例如用$-logp(s)$，通过计数作为$p(s)$的无偏估计，作为reward也能达到探索的效果。

这样pacman便不会一直陷在local optimal中不出来，并在bigMaze这样的复杂环境下通过调节参数，鼓励探索，找到目标（画风莫名开始变得像RL了）

3、Population diversity

仍然是因为local optimal而引入。即使调高popSize，也很能按照第一版的方法依靠mutation的产生出动作序列跳出，即使产生，也会由于其当前暂时的低fitness而被拒绝（没有考虑到long-term的return）。所以这里通过KMeans对final_state（即每个动作的最后位置）聚类划分，根据每个聚类的数目分别取用fitness最高的若干个。这里并没有考虑对动作序列进行聚类分析是因为一个动作序列第一位采取STOP的方式，其他位和另一个动作完全相同。这样在一般的距离度量下可能判断两者不属于同一个类，但实际效果是相同的，所以我们还是只考虑最后的结果。

函数（删减版）如下

```
def survivor_selection(self, fitness, final_s):
    cluster_res = KMeans(n_clusters=2).fit(final_s)
    group_res = cluster_res.predict(final_s)
    group0 = np.argwhere(group_res == 0).squeeze()
    group1 = np.argwhere(group_res == 1).squeeze()
    size0 = int(self.popSize * len(group0) / len(final_s))
    size1 = int(self.popSize * len(group1) / len(final_s))
    modify size
    assert size0 + size1 == self.popSize
    idxs0 = group0[np.argsort(fitness[group0])[-size0:]]
    idxs1 = group1[np.argsort(fitness[group1])[-size1:]]
    idxs = np.concatenate((idxs0, idxs1))
    return idxs
```

具体部分参数数值：

actionDim（个体长度）	**20**
T(迭代次数)	100
popSize（种群大小）	40
pm（bitwise mutation概率）	0.2
fitness比重（见getFitness代码）	1:5:10:2

Task2&3

虽然任务2要求直接在这些环境中测试，但考虑到PositionHeuristic等函数的设计只针对了Task1中这些单个食物的任务，problemType等情况，故在这里将Task2和Task3合并起来，针对现在的两个环境分别修改并测试（这里SearchAgent都响应进行重写了）。

Search环境

这里重新写了FoodEvolutionSearchAgent()类（继承了EvolutionSearchAgent类）。运行时通过命令行指定参数 -p FoodEvolutionSearchAgent即可。

最主要修改的地方是getFitness方法，也即对评估函数的改进。

针对FoodSearch型任务时，在Fitness的计算中加入了吃掉的food数量和距离当前最远food的manhattan距离（实践中发现需要为了防止地图只剩下一个food后无目的的寻找），分别正相关、负相关。设定一定比例的参数（5和-1）。同时保留了no_back_forth（不进行无谓的往复行动），block（撞到墙壁）两个参数。

同时对于3项Search任务不难发现Food的位置有一定的连贯性，相较于Task1，很少陷入local optimal，可以连续的得到一些正向的反馈（即吃掉食物，得到高的fitness）反而更加容易达到目标。因此在survivor_selection的过程中，便没有考虑多样性，直接选取最大的popSize个个体进入下一代。

在三张地图各运行5次结果如下：

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw2\search-code-2> python pacman.py -l Search1 -p FoodEvolutionSearchAgent -n 5
Pacman emerges victorious! Score: 531
Find the target, use fitness 1 times
Pacman emerges victorious! Score: 531
Find the target, use fitness 1 times
Pacman emerges victorious! Score: 531
Find the target, use fitness 1 times
Pacman emerges victorious! Score: 531
Find the target, use fitness 1 times
Pacman emerges victorious! Score: 531
Average Score: 531.0
Scores:        531.0, 531.0, 531.0, 531.0, 531.0
Win Rate:      5/5 (1.00)
Record:        Win, Win, Win, Win, Win
```

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw2\search-code-2> python pacman.py -l Search2 -p FoodEvolutionSearchAgent -n 5
Pacman emerges victorious! Score: 596
find the target, use fitness 301 times
Pacman emerges victorious! Score: 595
find the target, use fitness 301 times
Pacman emerges victorious! Score: 597
find the target, use fitness 319 times
Pacman emerges victorious! Score: 595
find the target, use fitness 301 times
Pacman emerges victorious! Score: 596
Average Score: 595.8
Scores:        596.0, 595.0, 597.0, 595.0, 596.0
Win Rate:      5/5 (1.00)
```

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw2\search-code-2> python pacman.py -l Search3 -p FoodEvolutionSearchAgent -n 5
find the target, use fitness 604 times
Pacman emerges victorious! Score: 751
find the target, use fitness 301 times
Pacman emerges victorious! Score: 772
find the target, use fitness 661 times
Pacman emerges victorious! Score: 753
find the target, use fitness 301 times
Pacman emerges victorious! Score: 771
find the target, use fitness 301 times
Pacman emerges victorious! Score: 772
Average Score: 763.8
Scores:        751.0, 772.0, 753.0, 771.0, 772.0
Win Rate:      5/5 (1.00)
Record:        Win, Win, Win, Win, Win
```

考虑到遗传算法的随机性，在如上图分数很可观的情况下，对参数的数值便并没有做过多调整。

Classic 环境

这里重新写了ClassicEvolutionSearchAgent()类（继承了EvolutionSearchAgent类）。运行时通过命令行指定参数 -p ClassicEvolutionSearchAgent即可。

主要修改了getFitness和部分参数。

对于fitness的计算，保留了FoodSearch中的计算方式。此外利用state和individual的动作序列产生next_state，计算score。同时加入了对胶囊的计算。以及考虑执行动作序列中每个状态如果与幽灵距离低于一个阈值的话，就给予一定的惩罚（减少fitness）。

```
def getFitnyess(self, state, problem, individual):
    state_copy = state.deepCopy()
    ghost_pos = state_copy.getGhostPositions()
    dis_threshold = 5
    near_cnt = 0
    for i, ac in enumerate(individual):
        for ghost_p in ghost_pos:
            dis2ghost = abs(location[0]-ghost_p[0]) + abs(location[1]-
ghost_p[1])
            if dis2ghost < dis_threshold:
                near_cnt += 1
        if state_copy.isWin():
            done = True
            break
        elif state_copy.isLose():
            f = -100000
            break
        if i + 1 < len(individual) and not is_back_and_forth(individual[i],
individual[i + 1]):
            no_back_forth += 1
        generate nextx, nexty
        if problem.walls[nextx][nexty]:
            block += 1
            stop_idx.append(i)
        else:
            state_copy = state_copy.generateSuccessor(0, ac)
            location = (nextx, nexty)
            if eat food food_cnt+=1
            if eat capsules capsules_cnt+=1
    max_dis = max(dis to food if food left)
    if f != -100000:
        f = 5 * food_cnt + 5 * capsules_cnt + 2 * no_back_forth - 2 * block -
max_dis + 10 *        state_copy.getScore() -5 * near_cnt
    return f, done, stop_idx, location
```

此外，对于参数也进行了修正。actionDim仅设为5。因为对于ghost的动作无法预测，所以过长的序列会导致执行过程中很容易被ghost抓到导致游戏失败。其他参数也尝试进行了一定的修改，不过由于随机性较强，且运行时间比较长，所以在效果变化不明显的情况下，仍选择保留了原始的参数。

以下为minimax，power两个地图上的运行结果。

在较复杂的power地图上，仍然有一次取得了游戏的胜利（真的不容易orz），与上次作业中通过启发式搜索实现的OneStepFoodAgent性能相近（不过耗时较长一些）。

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw2\search-code-2> python pacman.py -l minimaxClassic -p ClassicEvolutionSearchAgent -n 5
Pacman emerges victorious! Score: 515
Pacman died! Score: -494
find the target, use fitness 142 times
Pacman emerges victorious! Score: 516
Pacman emerges victorious! Score: 515
find the target, use fitness 211 times
Pacman emerges victorious! Score: 516
Average Score: 313.6
Scores:        515.0, -494.0, 516.0, 515.0, 516.0
Win Rate:      4/5 (0.80)
Record:        Win, Loss, Win, Win, Win
```

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw2\search-code-2> python pacman.py -l powerClassic -p ClassicEvolutionSearchAgent -n 5
Pacman emerges victorious! Score: 1735
Pacman died! Score: -316
Pacman died! Score: -342
Pacman died! Score: 449
Pacman died! Score: -243
Average Score: 256.6
Scores:        1735.0, -316.0, -342.0, 449.0, -243.0
Win Rate:      1/5 (0.20)
Record:        Win, Loss, Loss, Loss, Loss
```

```
(base) PS C:\Users\19124\Desktop\Junior3\Coding_HW\HSEA\hw2\search-code-2> python pacman.py -l originalClassic -p ClassicEvolutionSearchAgent -n 5
Pacman died! Score: 62
Pacman died! Score: 783
Pacman died! Score: 115
Pacman died! Score: -350
Pacman died! Score: 747
Average Score: 271.4
Scores:        62.0, 783.0, 115.0, -350.0, 747.0
Win Rate:      0/5 (0.00)
Record:        Loss, Loss, Loss, Loss, Loss
```

同时在实际游戏中发现avator学会了躲避幽灵，吃掉胶囊，以及吃掉后追击幽灵等操作（由getScore隐式的给出）。

针对不同的地图环境，也分别调整了部分参数。例如在minimax，power这类小地图中，dis2ghost的阈值相应需要调小一些，避免在ghost分散四周的情况下难以做出好的行动，相应的在计算fitness权重时也调高一些，保证pacman能意识到并逃离。而在original这种较大的地图中，dis2ghost的阈值则可以调高一些，及时的规避风险。

其他一些参数也分别调整进行了实验，不过由于耗时较长且存在随机性，很难充分测试找到最佳参数。过程不再赘述。

3、Conclusion

实验中实现了演化算法，针对Position, Food, Classic三种环境分别设计Fitness函数，调整参数，在实际环境中测试算法效果。