人工智能导论

第二次课程作业

Prof. Jianmin Li

马鸿鹏2014012459

Submitted in March 31, 2017

1 Reflex Agent

Solution

在第一个问题中，需要填写缺失的evaluationFunction，这个函数的目的是为当前的状态打一个分数。

我的思路是这样的，适合简单考虑的因素是食物，怪，当远离食物时施加一个惩罚，当远离怪的时候施加一个奖励，Pacman和这些物品之间的距离用曼哈顿距离来衡量，因为这个函数更符合Pacman和怪物在迷宫之中的移动

公式如下：

$$ManhattanDistance(\vec{x},\vec{y}) = \sum_i abs(x_i - y_i) \tag{1}$$

为了防止Pacman撞上怪物，当Pacman和怪物距离为0时增加一个超级大的惩罚，这是为了防止Pacman主动撞上怪物。其次对Pacman和怪物距离为1的时候加上一个小惩罚，因为这种情况很危险，ghost下一回合就可以吃掉Pacman。PS：这是一种极其简单的处理，在后面的Bonus题会在这个基础上有明显的改进。

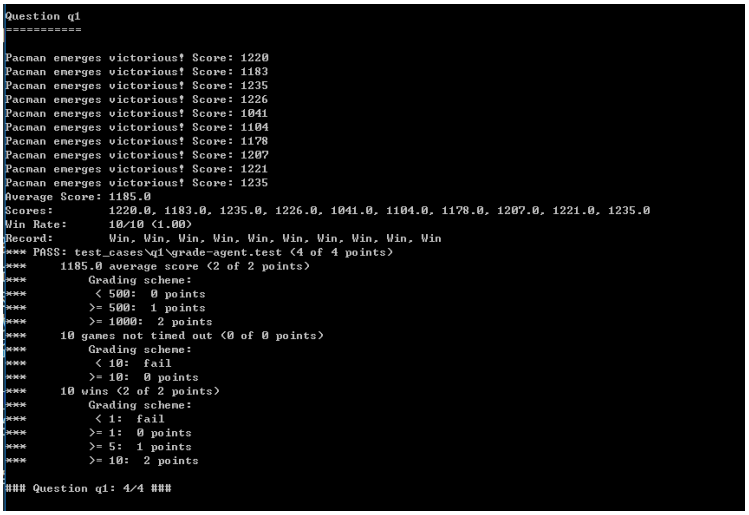

Fig 1. Reflex Agent Result

2 MinimaxAgent, $\alpha - \beta$ pruning Agent

Solution

这道题基本就是按照课上讲过的Minimax和$\alpha - \beta$实现一下，算法上面主要按照课上讲的内容实现即可。出发点上是Pacman采取MaxMin策略，最大限度地提高在最糟糕情况下eval function的结果，而Ghost采取MinMax策略，最大限度地惩罚Pacman，这个算法只有并不能确保给出最优解（Heuristic），这受限于使用的evalFunction。

题外话，博弈论中的MaxMin Strategy

$$\text{Maxmin Strategy for player i} = arg\max_{s_i}\min_{s_{-i}} u_i(s_1, s_2) \tag{2}$$

其中$s_i$player i 采取的策略s_{-i}是其他player采取的策略。在有限步，两个player，零和游戏的纳什均衡点时Maxmin策略和Minmax策略相等，而本题中实现的算法并不能算是一个确定能找到最优解（分数最高）策略。

3 Expection Max Agent

Solution

这种情况下考虑对手使用的策略服从某个分布，而在本题中则是平均分布（Ghost随机选择方向）。实际上相当于替换在MiniMax算法中的Min-value函数，使之变成Ghost的行动在某一分布下的期望。

比较需要注意的是李老师在课上讲的对对手的预测并不是对手可能行动的平均，而是在一定分布下自己得分期望值，这个算法最大化玩家得分的期望。

4 better evaluation function

Solution

```
if len(Food.asList()) > 0:
    for food in Food.asList():
        foodDist.append(util.manhattanDistance(food, Pos))
    foodScore = -average(foodDist)

for capsule in Capsules:
    capsuleDist.append(util.manhattanDistance(capsule, Pos))
if len(capsuleDist) > 0:
    capsuleScore = -sum(capsuleDist)

if(len(GhostStates) > 0):
    for ghost in GhostStates:
        ghostDist.append(util.manhattanDistance(
            ghost.getPosition(), Pos))
    _mid = multiplyListByElement(
        ghostDist, ScaredTimes, (lambda x: x[0] + x[1]))
    ghostScore += sum(filter((lambda x: x == 1), _mid)) * -200
    ghostScore += sum(filter((lambda x: x == 0), _mid)) * -99999
    ghostScore += sum(filter((lambda x: x < 0), _mid)) * 200
    # ghostScore += sum(filter((lambda x: x > 1), _mid))
    # print _mid, ghostScore

maxBestScore = currentGameState.getScore() + foodScore + \
    ghostScore + capsuleScore * 10
```

Fig 2. Better Evalutaion Function Source Code

我的改进主要在以下方向

- 考虑食物的影响，离所有的食物距离均值（对总和做修正）越近越好
- 优先考虑虚弱Ghost的胶囊，它的权重是普通食物的10倍（这是一个人为指定的数字）
- 对正常的Ghost和快要恢复正常的Ghost采取回避策略，而对虚弱Ghost采取优先击杀的策略

最后的Evaluation Function

$$\text{Evaluation Function} = \text{Current State Score} + \text{Average(Food Distance)}$$

$$+ \text{Sum(Capsule Distance)} * \text{weight} + \text{Ghost Score} \quad (3)$$

这个函数很多改进的方向，比如对参数的调优，考虑墙壁的影响等等