

问题： 在你的报告中观察并记录智能车在采取随机动作时的行为。它最终到达目标位置了吗？还有什么其他有趣的现象值得记录下来？

答案：

如果采取随机动作，智能车很少到达最终目标位置，因为采取随机的动作。但刚开始做项目的时候，我理解错了第一部分的意图，我将 `action` 设置为 `next_waypoint`，之后我发现智能车的 `reward` 只会出现 2 和 -1，但一般情况下都能到达目的地，而且当 `reward` 等于 -1 时，智能车再 `pygame` 的界面中是无法移动的。

现象 1：

但在这里我也仔细研究了 `next_waypoint` 算法，最佳路线是优先选择东西向作为首先移动的方向，然后再选择南北方向。这是因为他先计算出当前位置和目的地的方向向量，然后根据方向向量和当前的车头朝向来选择车辆下一步要走的方向。比如，目的地是在当前位置的正东方向，而且车头也是朝东的，那么下一步的移动方向就是向前；而如果车头是朝西的，那么系统会让智能车掉头朝东走。而如果，目的地是在当前位置的东南方向，智能车会优先选择向东走，然后再向南。

现象 2：

如果智能车走到了地图的边缘，并且下一个方向是朝着

入.

在这里我没有选择 **deadline** 的原因是:即使当 **deadline** 非常小,情况非常紧急时,智能车在运行时也不会违反交通规则直接越过红灯进入下一个状态,所以也不需要加入.

合适的情况总体来说大致分为四种:

1、`self.next_waypoint == 'right' and inputs['light'] == 'red' and inputs['left'] == 'forward'`

智能车最佳的选择是 **None**

2、`self.next_waypoint == 'forward' and inputs['light'] == 'red'`

智能车可行的选择是 **right or None**,最佳的选择是 **None**

3、`self.next_waypoint == 'left' and if(inputs['light'] == 'red' or (inputs['oncoming'] == 'forward' or inputs['oncoming'] == 'right'))`

智能车无法向左转,但可以选择其他方式,智能车可行的 **action** 有很多种情况,在这里就不一一列举了,在 **next_waypoint** 是最佳路线的情况下,最好的选择应该是 **None**.

4、这是除了上述三种情况下的最后一种情况,也是最好的情况下,跟着 **next_waypoint** 走就一定能用最快的速度到达终点了.

可选题目: 在这个环境中,智能出租车总共可能有多少状态?这个数字足够让我们的智能车做 **Q-Learning**,使得它在

每个状态可以做出基于训练的决策吗？如果是，为什么？如果不是，也说一下原因。

再这样的情况下，智能车共有 $2(\text{traffic light}) * 4(\text{forward}) * 4(\text{right}) * 4(\text{left}) * 3(\text{next_waypoint})$ ，但不包含 None，484 种状态。

我认为是可以的，因为在上述的四种整体情况中，首先在智能车行驶 100 次的情况下，如果每次 Qlearning 优化大约 20 次，那么 Qtable 大致会更新 2000 次，会包含绝大部分的情况，并且在这 484 中情况下大多数 state 是在第 4 种，而 $\text{action} = \text{next_waypoint}$ 总会得到正的优化。

问题：与一直选择随机动作相比，你发现智能车的行为有了什么样的变化？为什么会发生这种变化？

智能车在一开始会随机选择路线行驶,但随着 Qtable 的不断丰富能够很明显的发现 $\text{reward} < 0$ 的情况出现的越来越少了.因为当智能车选择错误的动作时,Qtable 在这一决策上的值会越来越小,而当智能车选择正确的动作时,Qtable 在这种决策上的奖励值会越来越大,所以在之后遇到相同情况时,智能车总会选择奖励值最大的动作作为最佳决策.

问题：把你实现基本 Q-Learning 时的参数调节过程记下来。哪个参数组合智能车表现最好？它最终的表现有多好？

alpha 应该叫做学习率,它的作用是在每一个 **Qtable** 更新过程中学习当前策略带来的奖励值,取值范围在 $[0,1]$,当 **alpha** = 0 时,**Qtable** 完全不更新,一直都采用原来的策略.当 **alpha** = 1 时,**Qtable** 每次都选择当前的策略,抛弃原有的 **Qtable**.我认为在使用 **Qlearning** 时应该在一开始选择相对较大的 **alpha**,但是随着学习的深入,**alpha** 应当随之减小,否则不停的更新可能会非常浪费计算资源.

gamma 叫做折扣因子,是指在当前策略下,未来的策略对现在的影响,取值范围也在 $[0,1]$ 之间

epslion 是随机项,取值在 $[0,1]$ 之间,当 **epslion** = 0 时,智能车完全根据 **Qtable** 选择最优决策,也就是不存在 **random**;当 **epslion** = 1 时,智能车完全随机,这从一开始让智能车随机选择动作完全一样.而在优化过程中,我认为 **epslion** 应该初始化为一个相对比较大的数字,尽可能的探索 **Qtable** 的空间,随后会越来越小,使整个 **Qlearning** 策略学习避免后期随机项带来错误的决策.

一开始我首先试着在 **state** 中加入了智能车的当前 **location** 和 **destination**, 因为我在没有依据的情况下认为 **route_to** 不一定是最佳路线,在不同的 **location** 和 **destination** 下最优的决策应该是不同的(就好像我们使用地图导航时,总会有不同的到达路径),但后来我试着和伙伴们交流,才理解在这个 **6*8** 的世界中这几乎就是最佳的策略。

一开始我试着改变了不同的参数以下是几种：

	alpha	gamma	epslion	最后 10 次 中采取的动作	最后 10 次中 action 错 误的次数	Action 正 确率	最后 10 次 中到达目 的地的次 数	成功率
1	0.2	0.8	0.1	210	57	72.86%	9	90.00%
2	0.2	0.8	0.2	218	74	66.06%	6	60.00%
3	0.2	0.8	初始化 0.8, 每一 次更新后 alpha * 0.9	205	53	74.15%	10	100.00 %
4	0.5	0.8	初始化 0.8, 每一 次更新后 alpha * 0.9	152	22	85.53%	10	100.00 %
5	0.8	0.8	初始化 0.8, 每一 次更新后 alpha * 0.9	137	10	92.70%	10	100.00 %
6	0.8	0.5	初始化 0.8, 每一 次更新后 alpha * 0.9	124	1	99.19%	10	100.00 %
7	0.8	0.3	初始化 0.8, 每一 次更新后 alpha * 0.9	160	1	99.38%	10	100.00 %

			初始化					
			0.8, 每一					
8	0.8	0.2	次更新后	184	1	99.46%	10	100.00
			alpha *					%
			0.9					

1、Epsilon = 0.1, alpha = 0.2, gamma = 0.8:

在这个情况下，最后的测试中经常会出现很多 **reward = -1** 的情况，也就是说 **action** 偏移了最佳的路线，效果并不好。

2、Epsilon = 0.2, alpha = 0.2, gamma = 0.8:

在这个情况下，因为 **epsilon** 太大，总会有 **20%** 的可能不管在什么 **state** 下，**action** 是随机选择的，导致会出现很多的 **reward = -1**。

在这里我发现了问题所在，那就是一开始训练时，我们希望随机选择能给 **Qtable** 很多正向的激励，但到最后，其实 **Qtable** 不再那么需要随机性来做决策了，因此我认为最佳的参数不应该是固定的值而应该是迭代变化的值。因此我在 **reset** 函数里修改了算法：

3、Epsilon = 0.8, alpha = 0.2, gamma = 0.8, 在每一次 reset 时, **self.epsilon = self.epsilon * 0.9**

这一次我发现在最后几次的学习中，因为 **epsilon** 已经非常非常小了，所以来自 **epsilon** 的干扰非常少，最后十次的行驶都能够成功到达目的地，但在行驶过程中还是出现了不少错误，看来这个思路有效。

4、**Epsilon = 0.8, alpha = 0.5, gamma = 0.8**, 在每一次 reset 时, **self.epsilon = self.epsilon * 0.9**

原来学习率设置的太低也会影响智能车的决策, 提高之后错误次数明显的减少了。

5、**Epsilon = 0.8, alpha = 0.8, gamma = 0.8**, 在每一次 reset 时, **self.epsilon = self.epsilon * 0.9**

我试着再加大一些学习率, 但我知道当 **alpha = 1** 时, 智能车只会根据新的 **Qtable** 进行决策而抛弃原有的决策, 因此我选择了 **0.8** 并且不再增加 **alpha**, 这时 **action** 错误的次数已经很少了, 但是还是有错误。于是我打算调整 **gamma** 值, 看看有没有帮助。

6, 7, 8、**Epsilon = 0.8, alpha = 0.8, gamma = (0.5, 0.3, 0.2)**, 在每一次 reset 时, **self.epsilon = self.epsilon * 0.9**

经过调整我发现 **gamma** 的值为 **0.2** 时, **action** 错误的次数已经非常少了, **action** 成功率达到了 **99.46%**。

但这时我还是有一个疑问, 如果 **Qtable** 不能收敛的化, 那么学习这么一项简单的功能岂不是要耗费很多的计算资源, 因此我试着调整 **alpha**, 让他也开始逐渐变小, 保证 **Qtable** 收敛, 经过五次的测试, 结果如下:

	alpha	gamma	epslion	最后 10 次 中采取的动作	最后 10 次中 action 错 误的次数	Action 正确率	最后 10 次 中到达目 的地的次数	成功率
9	初始化 0.8, 每一 次更新后 alpha * 0.95	0.2	初始化 0.8, 每一 次更新后 alpha * 0.9	157	0	100.00%	10	100.00%
10	初始化 0.8, 每一 次更新后 alpha * 0.95	0.2	初始化 0.8, 每一 次更新后 alpha * 0.9	130	0	100.00%	10	100.00%
11	初始化 0.8, 每一 次更新后 alpha * 0.95	0.2	初始化 0.8, 每一 次更新后 alpha * 0.9	141	1	99.29%	10	100.00%
12	初始化 0.8, 每一 次更新后 alpha * 0.95	0.2	初始化 0.8, 每一 次更新后 alpha * 0.9	134	0	100.00%	10	100.00%
13	初始化 0.8, 每一 次更新后 alpha * 0.95	0.2	初始化 0.8, 每一 次更新后 alpha * 0.9	136	1	99.26%	10	100.00%

当 **Epslion = 0.8, alpha = 0.8, gamma = 0.2**, 在每一次 reset 时, **self.epslion = self.epslion * 0.9, self.alpha = self.alpha * 0.95** 时, 经过五次测试, 智能车均能准确到达目的地, 同时

action 平均准确率达到了 **99.71%**。同时我发现如果一开始训练的**错误 action** 少的话，智能车更有可能出错。

此外，我还尝试了将 **gamma = 0.5**，但这时的结果却非常不稳定。这也帮助我理解了，其实每一次选择动作时，从未来获得的奖励不应该太大，就好像步子迈大了容易扯着蛋。

问题： 你觉得你的智能车已经几乎找到了最佳策略吗？例如，能够在最短时间内到达目的地，不遇到任何惩罚。在这个问题中，你觉得应该怎样定义最佳策略？

我认为已经快要找到最佳策略了，我认为最佳的策略就是

第一在训练的开始，利用随机因子 **epslion** 不断的试错，积累足够的经验，随后逐步缩小 **epslion**，避免随机因子做出错误决策。

第二在训练的开始，需要尽可能的学习不同的策略带来的奖励，但在 **alpha** 学习到一定程度时，需要适当的减少学习率，个人认为在这个项目中不存在所谓的过拟合状态，但也因在 **Qtable** 训练完成后，确保 **Qtable** 是稳定的，而不是持续不断的变化。

第三，对于未来的折扣因子 **gamma**，不论在什么时候，未来对现在的影响都不应该是最重要的，因此不应该赋予未

来过高的期望，而更应该着眼当下，快乐就好。