QUESTION1:Observe what you see with the agent's behavior as it takes random actions. Does the eventually make it to the destination? Are there any other interesting observations to note?

With enforce_deadline as False, the smartcab will eventually make it to the destination.

Observation:The route from start to destination is not optimal because the cab does not handle with specific situation and does not remember or utilize previous experience. So, each time, it takes different iteration numbers to finish the route. Most iterations exceed the default limit.

QUESTION2: What states have you identified that are appropriate for modeling the smartcab and environment? Why do you believe each of these states to be appropriate for this problem?

The states include the traffic lights (red / green), the oncoming traffic (forward/none, the right traffic(right/none), the left traffic(left/none).These selections come from the traffic rule in US. The light and traffic situation does matter the decision of the car.

Besides, I also use next_waypoint in the model. It could be used to defined the target to reach the destination. The agent can use this information to act properly to reach the destination in an efficient manner.

QUESTION3:What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?

After implementing the learning, I find the car could make the right behavior meeting specific situation. And it could reach the destination faster compared to before.

The reason for that is because the system chose the larger reward when making decision. So the cab is more likely to choose activity with better result. Also, because we also have next_waypoint in the model, the whole process should be accelerated.

QUESTION4: Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?

The parameters I choose to change include learning rate. Gamma and Epsilon. Detailed result shown below:

Learning Rate	Gamma	Epsilon	
0.2	0.5	1	22
0.2	0.5	0.4	73
		0.2	88
		0.1	97
		0.05	97
	0.6	0.4	73
		0.2	83
		0.1	88
		0.05	97
	0.8	0.4	72
		0.2	85
		0.1	91
		0.05	94
	0.9	0.4	68
		0.2	77
		0.1	91
		0.05	94
0.1	0.5	0.4	76
		0.2	87
		0.1	96
		0.05	96
	0.6	0.4	72
		0.2	75
		0.1	91
		0.05	91
	0.8	0.4	70
		0.2	80
		0.1	88
		0.05	93
	0.9	0.4	67
		0.2	85
		0.1	81
		0.05	91
0.05	0.5	0.4	68
		0.2	88
		0.1	99
		0.05	98
	0.6	0.4	70

		0.2	76
		0.1	91
		0.05	94
	0.8	0.4	70
		0.2	84
		0.1	94
		0.05	97
	0.9	0.4	62
		0.2	79
		0.1	89
		0.05	94

The best parameter combination is {“learning rate”:0.05, “Gamma”:0.5, “Epsilon”:0.1}, it achieves 99 success in 100 trials.

QUESTION5:: Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties? How would you describe an optimal policy for this problem?

The agent find the optimal policy.

One optimal policy is described below:

```
Environment.reset(): Trial set up with start = (5, 6), destination = (6, 3), deadline =
RoutePlanner.route_to(): destination = (6, 3)
LearningAgent.update(): deadline = 20, inputs = {'light': 'red', 'oncoming': None,
'right': None, 'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 19, inputs = {'light': 'red', 'oncoming': None,
'right': None, 'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 18, inputs = {'light': 'red', 'oncoming': None,
'right': None, 'left': None}, action = None, reward = 0.0
LearningAgent.update(): deadline = 17, inputs = {'light': 'green', 'oncoming': None,
'right': None, 'left': None}, action = left, reward = 2.0
LearningAgent.update(): deadline = 16, inputs = {'light': 'green', 'oncoming': None,
'right': None, 'left': None}, action = left, reward = 2.0
LearningAgent.update(): deadline = 15, inputs = {'light': 'green', 'oncoming': None,
'right': None, 'left': None}, action = forward, reward = 2.0
Environment.act(): Primary agent has reached destination!
LearningAgent.update(): deadline = 14, inputs = {'light': 'green', 'oncoming': None,
'right': None, 'left': None}, action = forward, reward = 12.0
```