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

No, it appears the **smartcab** never makes it to the destination. As of this point in the program, the actions appear to be completely random, with zero learning occurring. It seems every time, the **smartcab** runs out of time and the trial is aborted.

QUESTION: *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?*

OPTIONAL: *How many states in total exist for the **smartcab** in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?*

The states I believe are appropriate for modeling the **smartcab** and environment are:

1 The status of the light – red or green. This is important because not yielding to the light if necessary will break the law and thus produce a negative reward.

2-3 Whether or not there is traffic to the left or oncoming. This determines the award, positive or negative, and the significance of the reward, large or small, that could be used to make decisions in the future. Left and oncoming need to be know in order to learn to avoid accidents.

4 Waypoint is another state – where the vehicle should go.

There are 4 states that exist, described above, I believe are useful for modeling the driving agent and environment. This seems reasonable since there are several factors that determine the state which can be used for the goal of Q-learning to learn and make informed decisions about each state. Deadline could possibly be left out, as it might not be as important as getting to the destination safely is. Getting there without breaking any rules maximizes the rewards, and assuming it is heading to the destination in a reasonably fast matter, it should be okay. Also, if the agent moves in the correct direction when it's legal, this results in a larger reward than staying in place and not moving, therefore, it would be to the benefit of the agent to move towards the target promptly, without the need to incorporate the 'Deadline' state.

Another state that can be left out is inputs['right'], this is because according to the rules and USA right of way laws.

QUESTION: *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?*

It seems that the agent's behavior changes to make decision that are no longer random and instead are based on some logic, although the agent does not always make the most beneficial decision. I believe this is happening now because the agent is calculating which immediate next decision will maximize the reward. From my observation, this seems to result in the agent avoiding cars as well as waiting for them to pass instead of just turning left or right. Now, the agent also seems to occasionally reach the target.

QUESTION: *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 different parameters turned in the basic implementation of Q-Learning were alpha and gamma. The set of parameters the agent performed best were gamma = 0.5, alpha = 0.5, and the max_q function used. These were tuned by adjusting them until the success rate was 100% and the number of penalties (negative rewards) were less than 10% of the actions. Like mentioned in the previous question, the agent performs fairly well, although reaching the destination is not the absolute fastest it seems like it could be. If other inputs were considered such as deadline that may increase the rate the agent reaches the destination, however, it may do this at the expense of avoiding collisions. There may be a trade-off. We also optimized the ‘val’ to equal 0.4. This was determined through adjusting all of the mentioned variables manually and recording the results in the table below.

100 Trials Each										
Initial Q	10									
Alpha	0.25	0.5	0.75	0.25	0.75					
Gamma	0.25	0.5	0.75	0.75	0.25					
Penalties Total	**161**	**159**	**417**	**528**	**91**					
Initial Q	-10									
Alpha	0.25	0.5	0.75	0.25	0.75					
Gamma	0.25	0.5	0.75	0.75	0.25					
Penalties /	**-**	**0**	**-**	**-**	**-**					
Initial Q	0									
Alpha	0.25	0.5	0.75	0.25	0.75					
Gamma	0.25	0.5	0.75	0.75	0.25					
Penalties /	**-**	**0**	**-**	**-**	**-**					
Initial Q	5									
Alpha	0.25	0.5	0.75	0.25	0.75	0.9	0.9	0.9	0.99	0.9
Gamma	0.25	0.5	0.75	0.75	0.25	0.25	0.1	0.01	0.01	0.01
Penalties Total	**119**	**202**	**431**	**407**	**31**	**57**	**30**	**31**	**25**	**25**

After finding a pattern that having a val of 0 or a negative number made the reward 0, it was clear that wouldn’t work. Then by testing different values of initial Q, we saw that having a value of 5 was better than 10 and then increasing the alpha and decreasing the gamma produced the fewest penalties (negative rewards). It seemed that eventually the model almost completely chooses all positive rewards and finds the destination with these tuned values.

It is possible that including other values like epsilon and tuning that with these and further messing with the initial Q could possible help the model learn faster but this can easily become time intensive. Maybe an algorithm can be made to tune this!?

QUESTION: *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?*

Yes, the agent does come close to finding an optimal policy and yes it does incur some “penalties” (negative rewards). Running the code:

```
if (reward < 0):
        self.penalty += 1
```

```
print "Negative reward: deadline = {}, inputs = {}, action = {}, reward = {}, waypoint
{}".format(deadline, inputs, action, reward, self.next_waypoint)
```

showed that there still were instances where the agent made the wrong decision. During the some trials before adjusting the parameter mentioned previously. These penalties where for the situation shown below:

> **State** = light was red, there was no oncoming, right, or left cars,
> And
> **Action** = a right turn.

I believe there is possibly a tradeoff between moving as fast as possible and minimizing penalties. Sometimes to reach the obstacle in the least amount of time, doing so is only possible if rewards are not always maximized. An optimal policy would be one that achieves the goal of reaching the destination in an efficient manner, without any accidents and minimizing the penalties. It may also be possible though to reduce the penalties through other adjustments such as with an e-greedy exploration, mentioned in one of the reviews.

The final optimal values that were chosen were alpha = 0.9, gamma = 0.1, and val=5 for the reasons stated in the previous answer.