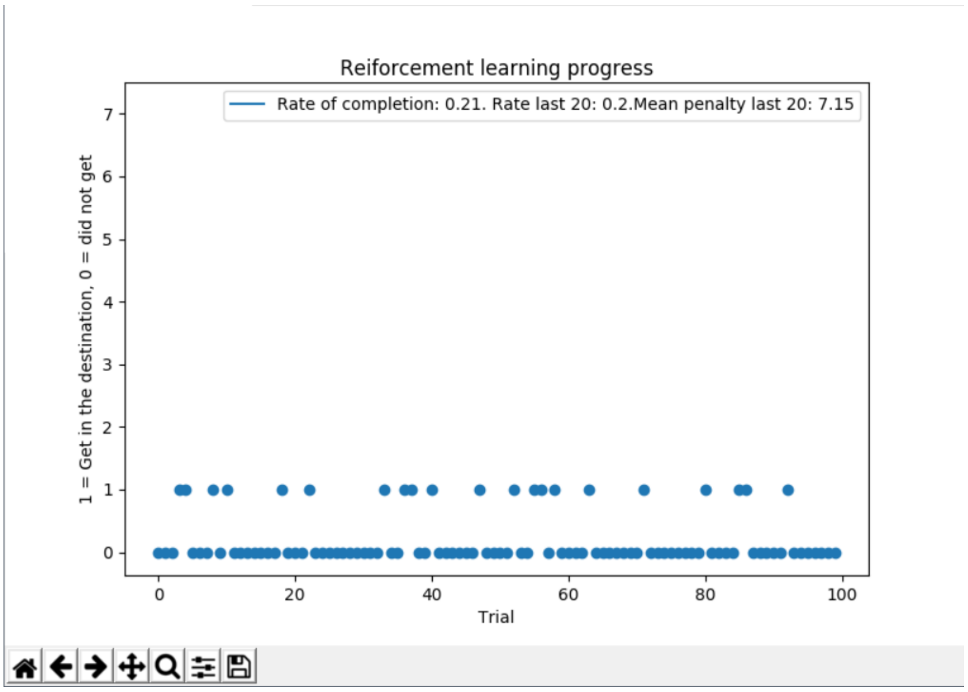
Machine Learning Engineer Nanodegree.

Reinforcement learning.

Question 1: Implement a Basic Driving Agent

Implement a random agent and o*bserve 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?

Answer: Well, random actions are very funny to watch, the car doesn't have any pattern, does many forbidden moves and doesn't seem to care about the right direction it should follow. It does, however, get eventually to the destination, which is not a surprise because of the high number of trials. I've run it three times with 100 trials and got a completion rate of ca. 20%(0.2,0.22,0.19), which was actually a surprise for me for being so high, I wouldn't guess it intuitively. The mean penalty(number of forbidden moves), however, is, as expected, very high. In the figure below one may see the accuracy of one random test.



Question 2: Inform the Driving Agent

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?

Answer: For the states I've divided each one depending on 4 features, [input.get('Light'), input.get('oncoming'), input.get('Left'), self.next_waypoint]. In this way it is possible for the agent to: 1 - Identify where it is suppose to go to minimize time, 2 - Identify forbidden moves, 3 - Find another away when the 'GPS' direction is a forbidden move. The input.get('Right') wasn't selected because it is irrelevant due to the USA right of way and the deadline was also neglected because it would extremely increase the state space and could make the car break traffic rules to get quicker to the destination when the deadline is low.

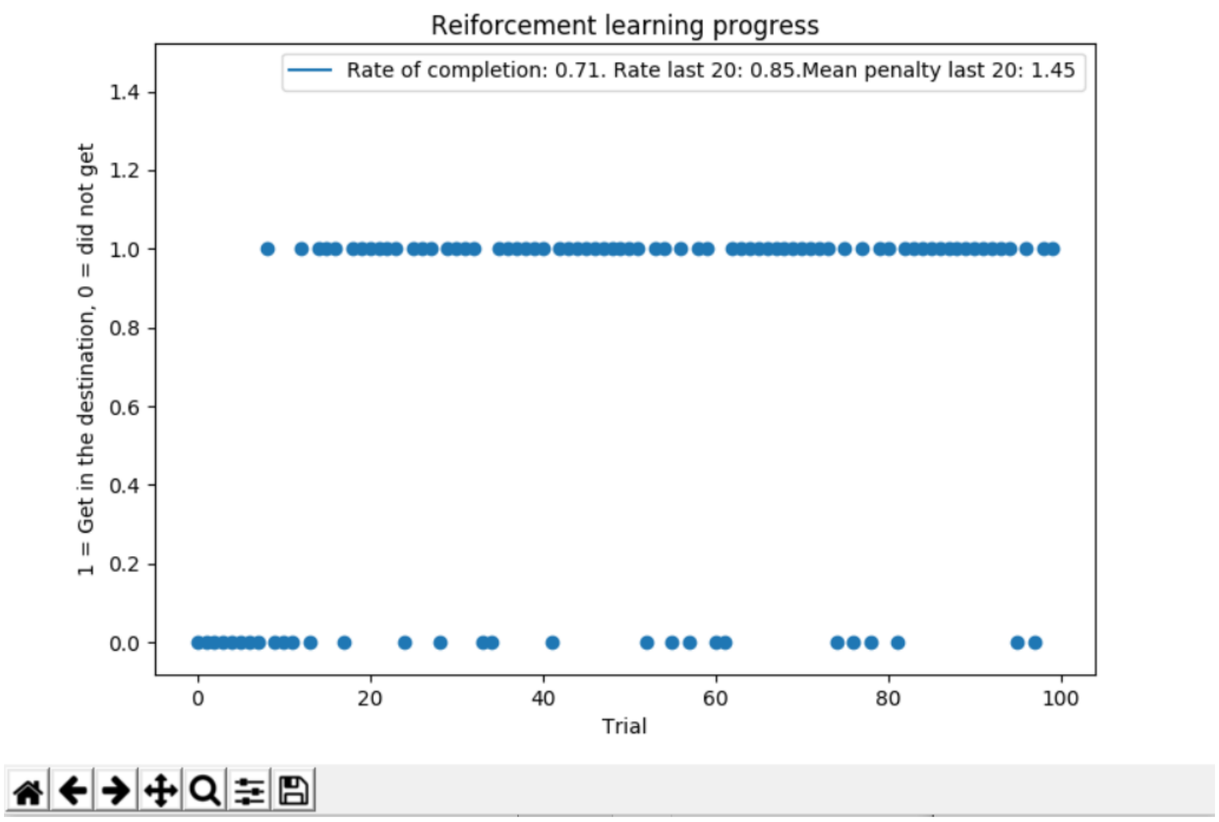
How many states in total exist for the ***smartcab*** in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

Answer: In this way of dividing states one has theoretically: (2 x Light) * (4 x oncoming) * (4 x Left) * (3 x way point) = 96 states . In practice we will see less states, I've run it three times with 100 trials and got an average of 40 states (43,39,37). Even the higher number would be a reasonable one if we run it enough times(enough being maybe 200-500 episodes, nothing monstrous), the lower number then is pretty reasonable.

Question 3: Implement a Q-Learning Driving Agent

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?

Answer: After implementing the Q learning we see that the agent starts to make more sense in its choices, tending to avoid forbidden movies and try to get as soon as it can to the destination, that was accomplished because now the agent get feedback for the states and it is able to evaluate its utilities and learn better actions. After implement it without playing with the parameters (alpha = 0.5, epsilon = 0.75 [or 0.25 depending how you see it] with a exponential with trials decay from 0.9999, gamma = 0.999) the mean general completion rate went to 69% (0.7,0.67,0.71) , a last 20 trials completion rate of 88% (0.95,0.85,0.85) and the mean penalty 1.36 (1.25, 1.4, 1.45). It is consistently better as the random walk, both in completion rate as also in the reduced penalties. In the figure below there's a picture of one of those trials.



Question 4: Improve the Q-Learning Driving Agent

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?

Answer: To improve the Learning I remained with the epsilon decay, so the agent would gradually stop to make random moves the more it learn and then remain with the best founded policy after most of the training and the value very close to 1 was so it could reasonable decay exponentially with the number of trials. Then I did a grid search in the parameters alpha and epsilon(I didn't change the discount factor because it could influence the car to go 'faster' and break more traffic rules), the table with the average (rounded from 3 rounds) completion rate in last 20 trials in a 100 trials run can be seem in the table below and the goal was to first check which ones would give the best completion rate:

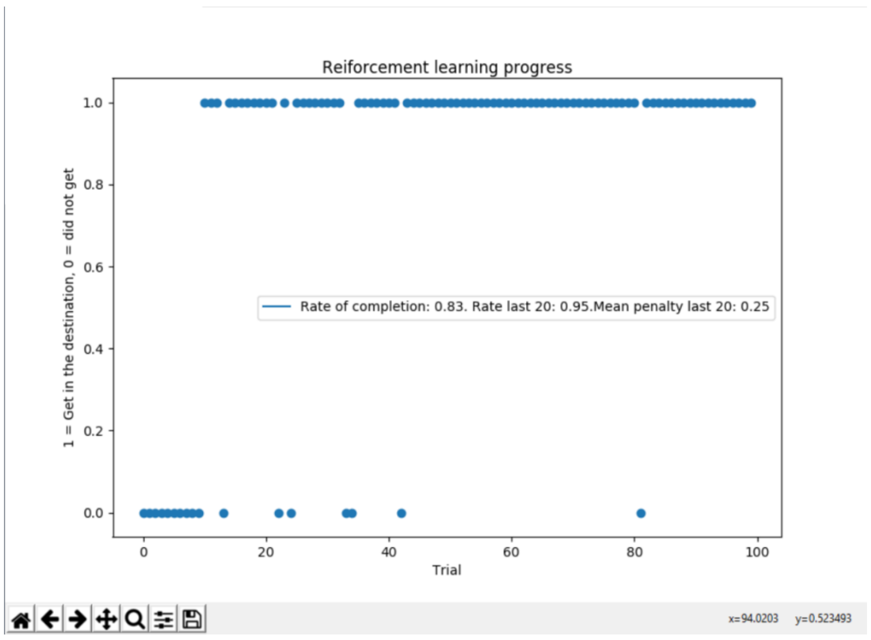
	Epsilon = 0.4	Epsilon = 0.75	Epsilon = 0.9

Alpha = 0.2	65%, Mean penalty: 3.6	86%, Mean penalty: 1.2	**96%, Mean penalty: 0.35**
Alpha = 0.5	48%, Mean penalty: 3.6	88%, Mean penalty: 1.4	93%, Mean penalty: 0.5
Alpha = 0.8	53%, Mean penalty: 3.5	95%, Mean penalty: 0.8	**98%, Mean penalty: 0.3**

We see that a higher Epsilon is needed to improve the result (increase exploration) and the alpha can also influence. After this first trial we have Alpha = 0.2, 0.8 and Epsilon = 0.9 as best results. We then run a trial of 100 times each to get a more reasonable mean and get the best parameters. We also check the mean penalty for the last 20 trials to evaluate how 'safe' our model is:

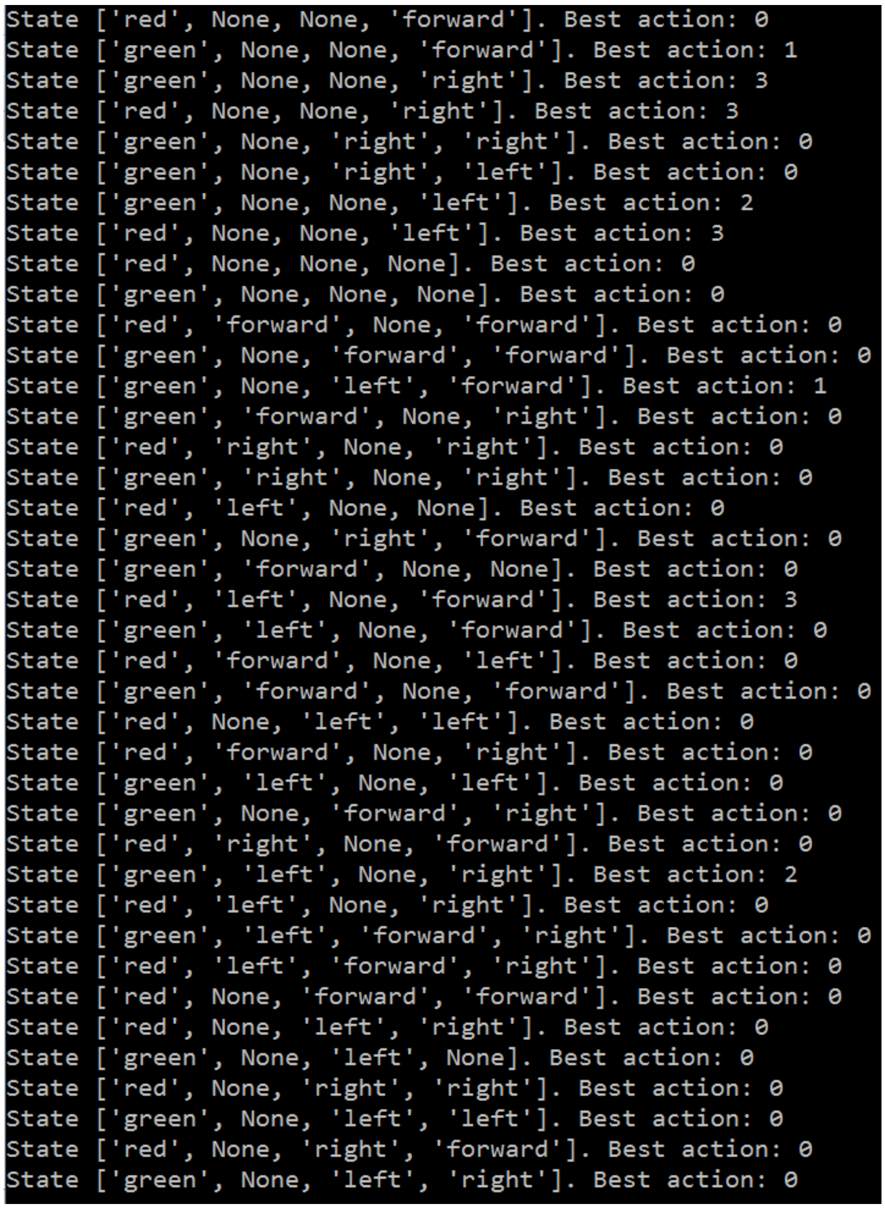
	Epsilon = 0.9
Alpha = 0.2	95.5% ; Mean penalty: 0.386
Alpha = 0.8	**96.5% ; Mean penalty: 0.25**

We end up concluding that alpha = 0.8 and epsilon = 0.9 are the best choice of the selected options. Below there's a plot of one run with the model:



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?

Answer: The optimal policy would be to always go to the GPS direction as long as it isn't a forbidden movement. If so, just move to another legal direction and/or wait until the direction is not a forbidden move anymore. We can analyze in the figure below the final table with the Best action (maximal Q value) for each state where : 0 = None, 1 = Forward, 2 = Left, 3 = Right. We perceive that the Q table is a mix between waiting to the GPS way to not be forbidden and to go to another direction, while in mostly of the free paths it has chosen the right answer. Therefore I would say it got close to the right policy.



```
State ['red', None, None, 'forward']. Best action: 0
State ['green', None, None, 'forward']. Best action: 1
State ['green', None, None, 'right']. Best action: 3
State ['red', None, None, 'right']. Best action: 3
State ['green', None, 'right', 'right']. Best action: 0
State ['green', None, 'right', 'left']. Best action: 0
State ['green', None, None, 'left']. Best action: 2
State ['red', None, None, 'left']. Best action: 3
State ['red', None, None, None]. Best action: 0
State ['green', None, None, None]. Best action: 0
State ['red', 'forward', None, 'forward']. Best action: 0
State ['green', None, 'forward', 'forward']. Best action: 0
State ['green', None, 'left', 'forward']. Best action: 1
State ['green', 'forward', None, 'right']. Best action: 0
State ['red', 'right', None, 'right']. Best action: 0
State ['green', 'right', None, 'right']. Best action: 0
State ['red', 'left', None, None]. Best action: 0
State ['green', None, 'right', 'forward']. Best action: 0
State ['green', 'forward', None, None]. Best action: 0
State ['red', 'left', None, 'forward']. Best action: 3
State ['green', 'left', None, 'forward']. Best action: 0
State ['red', 'forward', None, 'left']. Best action: 0
State ['green', 'forward', None, 'forward']. Best action: 0
State ['red', None, 'left', 'left']. Best action: 0
State ['red', 'forward', None, 'right']. Best action: 0
State ['green', 'left', None, 'left']. Best action: 0
State ['green', None, 'forward', 'right']. Best action: 0
State ['red', 'right', None, 'forward']. Best action: 0
State ['green', 'left', None, 'right']. Best action: 2
State ['red', 'left', None, 'right']. Best action: 0
State ['green', 'left', 'forward', 'right']. Best action: 0
State ['red', 'left', 'forward', 'right']. Best action: 0
State ['red', None, 'forward', 'forward']. Best action: 0
State ['red', None, 'left', 'right']. Best action: 0
State ['green', None, 'left', None]. Best action: 0
State ['red', None, 'right', 'right']. Best action: 0
State ['green', None, 'left', 'left']. Best action: 0
State ['red', None, 'right', 'forward']. Best action: 0
State ['green', None, 'left', 'right']. Best action: 0
```