In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?

I see exactly what I would expect from the current random implementation. It does not obey traffic laws and performs a random action. Since the enforce deadline variable is set to False, the car eventually reaches its destination everytime by chance alone.

Justify why you picked these set of states, and how they model the agent and its environment.

My state uses 5 variables: next_waypoint, light, oncoming, left, and deadline. Next_waypoint has 3 options: forward, right, left; light has 2 options: red, green, oncoming and left both have 4 options: None, forward, right, left; deadline has 2 options: normal and rush.

I made the deadline have a boolean value, because I figured the cab might optimize itself to break certain driving rules to reach its destination when there was little time left. I set this rush value equals to times less than and equal to 5 remaining time units. However when there is sufficient time remaining i think it will get optimized to always obey the traffic laws.

Next_waypoint is the most important state variable because it is the only variable that my algorithm can learn from to efficiently direct the car to its destination.

The light, oncoming, and left variables are necessary and sufficient to provide a car enough information to drive in accordance with traffic laws. The right state is not needed, because I assume the other drivers will be obeying traffic laws. As long as my car follows traffic laws, then I never need to know about the right side. I can turn right on red if I know the left is clear. I can turn left on a green if I know the straight away is clear.

Implement Q-Learning. Run it again, and observe the behavior. What changes do you notice in the agent's behavior?

For my implementation I used 3 untuned variables epsilon, alpha, and gamma. I used my intuition when picking these values. I used the variable epsilon and a random number generator to decide if my cab should explore or exploit my q table. I set epsilon to 1000 and for each decision and generated a random number between 1-1000 inclusive and compared it with epsilon. If the number was larger I would exploit, and if equal or smaller and would explore and decrement epsilon by one. In doing this the algorithm would explore heavily during the initial phase, then slowly decrease exploration as it built up the q-table.

I set alpha to .8 which causes the q-new function to weigh the new reward and the max utility of the new state a lot higher than q-old. I set gamma to .2 because I didn't think there was much value in knowing the utility of future states, since they are not as informative in this simulation.

Also I used a stochastic view of the light values for calculating Q-max of S'. When a car takes an action and reaches a new light there is a 50% chance of that light being either green or red. But, if a car takes no action and remains at a light there is a bayesian probability since you know what color the light already is. Looking at the environment code, I factored in probability into Q-max of S' when the car remained in place.

When I ran this with enforce_deadline = True the cab drove randomly in the first several trials but then starts to drive with some understanding of the policy. Near the end you can clearly see the cab driving efficiently towards its destination. The cab reached its destination in ~74.5/100 runs before tuning the variables.

Enhance driving experience

I used a grid-search method to tune the epsilon, alpha, and gamma variables. For epsilon I used values of [900,1000,1100,1300,1400,1500] and for alpha and gamma I used values from 0-1 by .05 increments. This works out to 2646 different variable combinations. The values that yielded the highest reached destination % was epsilon=900, alpha=.95, gamma=.05. Using these values the cab reached its destination ~78.7/100 trials.

I think the cab performs really well. During the later portions the cab is clearly making smart choices. An ideal policy would be to follow the planner's next waypoint in according with the traffic laws. Even if the cab did this, it wouldn't guarantee that the cab made it to its destination in time though. But overall I think this should be the ideal policy. My cab failed to achieve an optimal driving policy, because I've noticed it occasionally making choices that are not what the planner advised. Sometimes near the end it will take a wrong turn and has to loop around to get back to where it was. When I ran it for over 100 trials I do notice the car making less wrong actions, and following the next waypoint more closely.

However, I did notice a bug in the city layout that affects the outcome. The cab can drive off the grid and will be warped to the opposite side of the grid, but the planner never takes this into account. So sometimes my car will take the wrong turn and warp all the way across the grid, and the planner routes then back several more steps than necessary. I feel like the planner should either incorporate this warp or the warp should not be a valid move.