Smart-Cab Driving Agent

Implement a basic driving agent

A basic driving agent was based on picking randomly the action from the legal actions (without worrying about the safety of decision) in the environment. The deadline for completing the task of reaching destination was turned off & agent ran for 30 runs.

```
valid_actions = ['forward', 'left', 'right']
#Q1 chose random action from valid_actions irrespective of action being allowed
action = random.choice(valid_actions[1:])
# Execute action and get reward
self.env.act(self, action)
```

Simulator.run(): Trial 0
Environment.act(): Primary agent has reached destination! deadline left 15
Simulator.run(): Trial 1
Environment.act(): Primary agent has reached destination! deadline left -29
Simulator.run(): Trial 2
Environment.act(): Primary agent has reached destination! deadline left 26
Simulator.run(): Trial 9
Environment.act(): Primary agent has reached destination! deadline left 9
Simulator.run(): Trial 10
Environment.step(): Primary agent hit hard time limit (-100)! Trial aborted.
Simulator.run(): Trial 11

Agent is able to make to destination in some of the trials but mostly violating/close to the deadline. There are intermittent failures where it fails to reach the destination.
The grid size is pretty small, its probable random moves might take agent to destination. I tried the random agent with with double the grid size, agent consistently fails to reach destination.

Inform the agent & states of agent

State of agent in environment is characterized by parameters of environment at any intersection

1) Traffic Light : Red,Green
2) Oncoming traffic : Left, Right, forward, None.
3) Traffic on left : Left, Right, forward, None.
4) Traffic on right : Left, Right, forward, None.
4) Desired direction to move : Left, Right, forward.

List of possible action are [Left, Right, forward, None]
If we take encode all these parameters in the problem, it leads to 2*4*4*4*3 = 384 states. The number of states are huge, with some redundant. This might lead to higher training time, as agent also need to visit the redundant states.

Domain knowledge
Given the logic of traffic light in agent's environment & intent of traffic in other lanes we can identify the validity of moves.

Forward action - Allowed
Traffic light is green

Left action -Allowed
Traffic light is green
No traffic in oncoming lane, both going forward & right.

Right action -Allowed
Traffic light is Red
No traffic in left lane, both going forward.

Right action -Allowed
Traffic light is green.

In none of the allowed actions, do we need to check traffic on our right lane.(Not true in real world, more complex as you take right on Red, need to check for u-turn in right lane). We can simplify the state by ignoring the traffic in right lane.

This reduces number of states to 2*4*4*3 = 96.
Actions possible in each are [Left, Right, forward,None]. None is valid move in case of safety hazard. It returns with reward of 0.
This gives a 96x4 Q value matrix

All these environment variables are needed to train a agent to reach the destination guided by the "Desired direction to move", while following the rules framed by other variables. Dropping any one of these variables will violate the learning goals for agent.

For example, removing any of the variables about traffic condition might result in action leading safety violations. While removing desired direction might result in agent which follows traffic rules, but is not guided to any specific direction. We could also include the deadline as one variable, but our training process & maze traversal is guided by desired optimal direction. If agent learns to follow the rules & take desired direction deadline violation won't happen unless the number of moves allowed are less than the one needed for optimal route.

```
def create_state(self):
# create state based on all parameters of agent
# can be simplified further based on knowledge of traffic rules
    inputs = self.env.sense(self)
    #print inputs
    return "{}|{}|{}|{}".format(self.planner.next_waypoint(), inputs["light"], inputs["oncoming"], inputs["left"])
```

Implement a Q-Learning Driving Agent

Agent is setup for training with all Q-values initialized to 20, higher than maximum possible reward. This allows the agent to positively take the actions in beginning.

The actions are chosen randomly at the beginning epsilon = 0.05, learning rate = 0.5 & discount = 0.3. Agent is run for 100 trails with deadline turned ON.

Agent misses the destination in the initial runs, however it consistently reaches destination for later runs. Table below shows example of its learned Q values for two states.

State				Action	Q val
Direction to move	Traffic Light	Onward traffic	Left traffic	[Left,right,forward,None]	
Forward	Green	None	None	forward	3.31
Forward	Green	None	None	left	-0.71
Forward	Green	None	None	right	-0.71
Forward	Green	None	None	None	0

In this state desired way next is forward while traffic situation allows agent to do so (clear intersection with green) so the forward action is chosen by highest Q-value of 3.31

State				Action	Q val
Direction to move	Traffic Light	Onward traffic	Left traffic	[Left,right,forward,None]	
Forward	Red	None	None	forward	-1.0
Forward	Red	None	None	left	-1.42
Forward	Red	None	None	right	-0.71
Forward	Red	None	None	None	7.42

In this state desired way next is forward while traffic situation is red, so agent chose to stay at its place with Q-value of 7.42.

Thus we have a trained agent which manages to reach its destination with in the deadline.

Improve the Q-Learning Driving Agent

On further analysis of agent with training parameters epsilon = 0.05, learning rate = 0.5 & discount = 0.3. We note the agent completes its task but still there are not of traffic infractions. Example in 100 run trial we get 58 traffic violations, 6 deadline violations (all in first ten trials).

On further analysis of state * actions pair in Q_value table, it seems not all states are visited as frequently. We try to introduce concept of simulated annealing where epsilon starts with high value allowing more random moves to be possible in beginning of training. Epsilon is scaled with each trial, down the training sequence actions are guided more by learned policy.

This helps us to cover the state * actions pairs, thus populating the appropriate Q_values for each traffic infraction state & action pairs.

Exp	epsilon	scaling_epsilon	Learning (α)	Discount (γ)	Infractions	Deadline Violations	Trials
1	0.05	1	0.5	0.3	**103**	**16**	100
2	0.1	1	0.5	0.3	**85**	**11**	100
3	**0.1**	**0.9**	**0.5**	**0.3**	**69**	**8**	**100**
4	**0.1**	**0.9**	**0.5**	**0.3**	**190**	**57**	**1000**
5	0.1	0.9	0.7	0.3	**71**	**10**	100
6	0.1	0.9	0.7	0.3	166	51	1000
7	0.1	0.9	0.8	0.3	**58**	**12**	100
8	0.1	0.9	0.8	0.3	163	63	1000
9	0.1	0.9	0.7	0.4	**59**	**10**	100
10	0.1	0.9	0.7	0.4	169	71	1000

Above table shows experiments performed for agent

Exp1 is static epsilon of 0.05, the agent learn to drive to destination, however the number of traffic infractions are quite high. This is because of not enough coverage of (state,action) in Q-value table.

Exp2 tries to increase number of random decisions in the beginning of training process by increasing epsilon. This does help in reducing both infractions & failed traversals.

Exp3 identifies that we need to increase random actions taken in beginning of training process. Simple form of annealing is used where we start with epsilon of 0.1 & scale by 0.9 at every new training cycle. This helps reducing both infractions & failed traversals as compared to previous experiments. Exp4 is same training process over 1000 cycles, the number of infractions & deadline violations doesn’t increase linearly indicating successful convergence of training.

Further experiments try to further try various combinations of learning rates & discounts. It seems our original values of learning_rate=0.5 & discount=0.3 are optimal. Exp3 & 4 shows more or less optimal solution.

We can improve upon the learning process by adding in the domain knowledge, as in setting up the Q_values of (state,action) pairs violating the rules of environment to high -ve values. This will lead to

reduced infractions. Another possibility is to have more advanced version of simulated annealing where we increase the probability of random actions deep into training process.