Implement a Basic Driving Agent

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?

ANSWER:
hen set the agent to take random actions, the SmartCab would drive around randomly. Most of the time, it does not reach to the destination. But very rarely, eventually, it would get lucky and it would make it to the destination. But it appears to be completely random and due to luck. It also appears that the SmartCab would sometimes break transportation rules and run into accidents, but it does not seem to learn from mistakes. Its behavior does not seem to improve even after receiving penalty, nor does it appear to responding to rewards.

Inform the Driving Agent

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?

ANSWER:
The self.state [self.state=(inputs['light'], inputs['oncoming'], inputs['left'], inputs['right'], self.next_waypoint)] attribute for LearningAgent is set to be taking the following three conditions into consideration as for the purpose of modeling the SmartCab and its environment: 1) the transportation light (green or red); 2) the traffic situation at the intersection (whether or not there is traffic in the oncoming, left, and right directions); 3) the planned next way point. The overall goal of one SmartCab trip is to get to the destination ***without violating transportation rules nor running into accidents*** *as fast as possible, hence the state of transportation light of an intersection is important state information to inform the LearningAgent so that it could learn not to violate transportation rule (would receive negative rewards if it violates). The information regarding to traffic state around the SmartCab and its next way point is crucial for teaching the LearningAgent to drive safely. The agent's variable [deadline] is another factor which is potentially useful state information, however it was not added into forming the LearningAgent's self.state attribute. Because adding [deadline] expands the number of self.state by a factor of 20~30. The solution to the MDP problem asks the algorithm to visit every states infinitely often, given the setting of the number of simulation trial runs (~100), a relatively smaller size of self.state is more desirable. In addition, the practical training result of this self.state setting without [deadline] info seems to be good enough. (details see the following sections)*

QUESTION: 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:
There are 32 states in total exist for the SmartCab in this environment. This number seems reasonable for Q-Learning to learn and make informed decisions about each state. If set the trial run number of the simulation to be 100, and assume that for each trial simulation run, the SmartCab takes 20 actions and experiences 20 states on average before it terminates (either because of successfully reach the destination or trip aborts due to run out of time), so the

SmartCab is expected to experience ~2000 states in total. This number seems to be reasonably large enough to ensure that all 32 states would be appeared/seen so that Q-Learning could learn about.

Implement a Q-Learning Agent

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?

ANSWER:
*After implementing Q-Learning, the agent appears to be no longer randomly driving around, and it seems to be responding to the rewards and penalties that its receiving after taking certain actions while its at certain states. Its rate of delivering customer to destination before time runs out obviously has improved as compared to when it's taking random behaviors. The reason for this difference is that Q-Learning algorithm [Q(state, action) = Immediate_Reward + discount_factor * argmax (Q(future_state, future_action))] is teaching the agent by rewarding agent's correct behavior (positive rewards for taking the right action at the right state), and punishing its incorrect behavior (negative rewards if it violates transportation rule or runs into accidents).*

Improve the Q-Learning Driving Agent

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?

ANSWER:
- *Tuned omega values: [0.7, 0.8, 0.9, 1.0, 2.0]. The learning rate is constructed using omega value: apha = 1/(t^(omega))*
- *Tuned gamma/discount factor values: [0.5, 0.6, 0.7, 0.8, 0.9]*
- *Tuned epsilon/exploration factor values: [0.1, 0.2, 0.3, 0.4, 0.5]*

The performances regarding to different settings of tuned parameters are stored in data.csv. It is loaded and analyzed in the Q-Learning Parameter Tuning-Performance Analysis section presented in the Report.ipynb.
*The optimal parameter setting is **omega = 0.7, gamma (discount factor) = 0.6, epsilon (exploration rate) = 0.3**.*
Using this parameter setting, the SmartCab achieved a success customer deliver rate of 0.96; it received an average penalty of -3.8, an average net reward of 21.81 during the 100-trial-run simulation. Its average time costs for delivering customer is 13.46 time step.*

*(*Note that the performance of the optimal parameter setting reported here is the output based on one specific parameter tuning simulation run. Due to the stochasticity of some other factors in the game, different runs would result slightly different output. Various results of simulation runs using the optimal tuned parameter setting suggest that the success rate regarding to customer delivery before time runs out typically varies in from 0.88 ~ 0.98)*

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?

ANSWER:
The LearningAgent appears to get close to finding an optimal policy which has a high customer delivery rate in relatively small time costs with high net reward and low penalty punishments. More details regarding to the parameter tuning and optimal parameter trained agent performance are analyzed and demonstrated in the following Q-Learning Parameter Tuning-Performance Analysis section in the Report.ipynb. An optimal policy for this problem would be a policy which produces a series of optimal actions for each state that the SmartCab is in, and this series of optimal actions would guide the SmartCab to accomplish the mission (delivery the customer to its destination) in the shortest time step possible with no violation of traffic rule and no occurrence of accidence (hence results in low penalties and high rewards).