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

Below is the table summarizing 10 trial simulations, (the distance is calculated with periodic boundary condition. The map is 8*6)

Trial	Start	Destination	Manhattan Distance	Steps	Rewards
1	(8,1)	(2,1)	2	2	14
2	(8,1)	(8,5)	4	130	72.5
3	(5,5)	(7,2)	5	103	63.5
4	(3,4)	(1,6)	4	79	29.5
5	(1,4)	(6,3)	4	58	25
6	(7,5)	(3,4)	5	470	233.5
7	(3,6)	(2,1)	2	199	113.5
8	(4,2)	(1,4)	5	50	29
9	(7,1)	(6,6)	2	9	12
10	(4,6)	(2,4)	4	133	84.5

We can see that there are really lucky trials (trial 1 and 9), really unlucky ones (trial 6), and many medium cases, as one may suppose for random walk. They all eventually get to the destination, as there is no time limit.

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

I basically copy the behavior of dummy agents. There're three features chosen to represent the agent: the action it wants to take (self.next_waypoint), the status of traffic light, and the status of oncoming traffic. If the agent's coming action will violate traffic law, then the action won't be taken. I choose it because it's basically how we drive. Additionally, I set self.state to a vector tracking total steps and reward up to current state.

BTW, there is a bug (or feature?) DummyAgent.update() in environment.py. The

elif self.next_waypoint == 'straight'

should be

elif self.next_waypoint == ‘forward’,

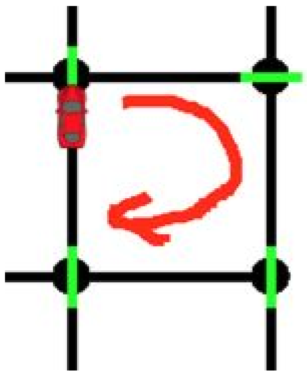
which agrees to the status defined in planner.py. The current dummy agents are so rude that they will run the red light if they're going straight.

Q-learning: What changes do you notice in the agent's behavior?

We're told not to change file other than agent.py. However, I find several places confusing, doesn't make sense, or hard to be overridden in agent.py, so I made some changes in environment.py file. I explain them in detail below.

First, I change the Environment.reset(). The assignment says "our goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive." It doens't specify whether these trials have same start and destination. The Simulator function given to us resets start and destination every trial. However, I don't think that makes sense. If the problem keeps changing, there won't be learning possible. So I change the reset function. Now the starting point and destination are only randomly generated once at very first one runs the code, then they're kept unchanged for the number of trials set in the code.

Second, the reward strategy is changed. The original strategy has two problems. First, it compares actions to good solutions generated in planner.py, which uses global position information. We're told no GPS is available, so such reward is kind of cheating in my opinion. Second, it gives any OK move (that doesn't violate traffic laws) positive reward. This is against our goal to get to destination ASAP. Even worse, it severely restrains the range of possible discount factor. The discount factor has to be small enough, otherwise the future becomes too important that the agent prefers to run forever (because it earns reward, not penalty, for its every move). For example, it may prefer to detour and loop forever (shown below). This is often observed in my simulations. Punishing rather than rewarding each step solves the problem.



Third, the compute_dist() function is wrong. It doesn't take periodic boundary condition into consideration. i correct it.

Fourth, the Q table is added as an object to the file, and added to be an attribute of the environment. I suspect we're expected to add it to agent.py and make it an attribute of the agent. That works, but I think Q table is associated with the global environment, not to a specific agent. My way seems to be more reasonable.

Lastly, some messages printed to screen are changed, to make things clearer.

With all these stuff in mind, now we can get to Q learning. I set learning_rete = 0.7 and discount_factor = 0.8, then executes 10 runs (different start and destination), each includes 100 trials (same start and destination). Time enforcement is applied. Generally speaking, the agent always fails at first, but 50 trials are enough to find a success policy. Furthermore, 7 out of 10 runs finally find the optimal policy. All end up with positive reward (remember I punish every move). Quite a good start.

Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

I changed the learning rate from 0.7 to 0.5 (emphasize learning more). I ran another 10 runs, they all converged to optimal policies in 50 trials.

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?

Yes.

Check out commit

3350aef9d088bdfb67fb253bd0770aed0f8d7583

to see random walk agent;

9aeb263aa3dd89657572e6ad357da09e69e0a552

to see random walk agent with traiffic laws;

c9c4809edd59be935903b71229158e7b1b11de6e

to see Q-Learning;

b4efed94533af894aa60f869d7cb23f649856c83

to see the improved Q-Learning agent.