Implementation of basic driving agent

After implementing the basic driving agent that processes at each time step:

- Next waypoint location, relative to its current location and heading,
- Intersection state (traffic light and presence of cars), and,
- Current deadline value (time steps remaining),

And produces random output (None, 'forward', 'left', 'right')

It is noticed that the agent wanders without any specific destination, with no regard to traffic light or presence of the other cars, given enough time it reaches the destination as there's the probability that it ends up there just by chance

Inputs

Sample:

deadline = 24, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, next_waypoint = left, reward = 12

- Deadline: number of moves available
- Light: traffic signal light
- For oncoming, left, right
  - None: that way is clear
  - Forward, left, or right: the direction of the car in that way
- Next waypoint: a direction from the simple planner to the way of the destination
- Reward: the reward received based on the action chosen.

Chosen statuses

- Light: traffic signal light
- Next waypoint: a direction from the simple planner to the way of the destination
- Light violation: a binary that indicates if the planner direction would violate the current light
  - Except for None action, all others in green light would not be a violation, and for red light only right action wouldn't be violating
- Right of way: a state that can be one of (no left, no right, has right of way) that represents the right of way on green and red light based on right-of-way rules
- Hurry: a binary that indicates the remaining moves are less than a threshold

Implementation of Q-Learning

Next step was to implement Q-Learning with values of alpha: .7 and gamma .9

Intuition for choosing the final values:

As we want the agent to learn in its first few steps then rely more on what he learned afterwards

Alpha (learning rate of the agent): it makes since to have relatively high alpha, since it's decaying, but not very close to one as it would be close to full learning (less weight on it learns)

Gamma (the value or weight of future reward): a high gamma makes it value future reward almost as much as immediate reward, as opposed to low gamma which would make it favor immediate reward.

The basic agent picks the action based on q table if state was encountered before; otherwise a random action is chosen.

When testing this initial implementation, I observed that the agent mostly preferred not to move, since that gave him a small reward and no reason to explore other actions.

Enhancements

To fix this issue, I've added manually small reward value for actions that it didn't take before to encourage the agent to explore

For example if initial values were [0, 0, 0, 0] and after the action it became [1, 0, 0, 0], the values are to be changed to [1, 1.05, 0, 0] then [1, 1.05, 1.1, 0] and so on.

Running the simulation again,, it's observed that the agent doesn't reach the destination most of the time, although it started to move and explore, after some investigation it turns that at some point the agent will take a random action in a new state and reach the destination getting a high reward, and when in the same state it takes the same (this time wrong) action but doesn't reach the destination

For example it turns left at red light intersection and gets big reward for that, and later it turns left at every red light.

To negate that effect, I added a simple hack to the reward that whenever the reward is higher than 5, we subtract 10 from it, making sure the agent doesn't receive big reward for an action that would be considered bad in most cases.

After running another test another issue came up with how we encourage the agent to explore, an example would be a new state with green light and forward waypoint, where the agent has randomly chosen to go forward (which is optimum) and next time it encounters this states it goes right receiving smaller reward, but when it adds to amount we already added to make it explore it's then higher than the correct action, by doing this, it goes in loops and fails to reach the destination.

At this point since there is no penalty for not reaching the destination, I removed the state "hurry" which is based on the deadline, to simplify things then return to it after the above issues are fixed.

As this method of exploration didn't work well, I removed it and made the following enhancements to the design

- On a new state it takes the next waypoint given by planner as an action (only first time on new state)
- On an existing state, the agent checks if there are actions it didn't try before, and try them on each visit to this state, once it tried them all, it picks the best of them on later visits
    - The above steps insure that the agents explores and learns the outcome of his options first before making use of what he learned
- Divided alpha value by the number of runs (which accumulates across trials), by doing so it makes the agent learn for small number of runs and then relies on what he learned for the rest of the trials
    - By doing so, the big reward that caused bad decision becomes irrelevant, so I removed the modification of the reward implemented earlier

Testing the above implementation, it appears that the agent fails to reach the destination for the first 1 to 3 trials, and then reaches it almost consistently for the rest of the trials.

Summary of Q-Learning Agent performance

Below table is generated based on taking random action based on a calculated probability against epsilon

Learning Rate	Discount Factor	Epsilon	Pass Rate	Explored Statuses	Penalty Ratio
0.7	0.9	0.9	62/100 %62.0	10	309/2144 = %14.41
0.7	0.3	0.9	70/100 %70.0	9	281/1964 = %14.31
0.7	0.3	0.6	23/100 %23.0	10	201/2812 = %7.15
0.3	0.7	0.9	20/100 %20.0	10	355/2891 = %12.28

The below table is generated by systematically making the agent explore all possible action first in a given state, then take exploitive action if all actions are known in its current state

Learning Rate	Discount Factor	Pass Rate	Explored Statuses	Penalty Ratio
0.7	0.3	96/100 %96.0	9	6/1640 = %0.37
0.3	0.7	43/100 %43.0	10	6/2251 = %0.27
0.6	0.4	99/100 %99.0	9	8/1507 = %0.53
0.8	0.9	59/100 %59.0	10	8/1972 = %0.41

The agent always reach its destination in the last ten trials, while following the rules of the road, although on red light it doesn't always take the shortest path, as we will see examining the Q-Table below taken from one of the trials.

Note: the values are ordered as [None, Forward, Left, Right]

light='green', light_violation=0, row='has_row', next_waypoint='forward'): [0.18409999999999999,

3.2394098577773063, Takes right action while following the rules

0.058333333333333333,

0.06976312091503269],

light='green', light_violation=0, row='has_row', next_waypoint='left'): [0.1438181818181818,

0.03284166666666666,

1.8226061090506565, Takes right action while following the rules

0.03814192307692307],

light='green', light_violation=0, row='has_row', next_waypoint='right'): [0.0604175,

0.033056333333333333,

0.077,

0.7886855778914873], Takes right action while following the rules

light='red', light_violation=0, row='has_row', next_waypoint='right'): [0.01044776119402985,

-0.0077912124769411355,

-0.007537151200519143,

0.32044669902265177], Takes right action while following the rules

light='red', light_violation=1, row='has_row', next_waypoint='forward'): [1.5390549535260452,

Chooses to stay in place as following the suggested action wouldn't be following the rules

-0.15983333333333333,

-0.0599375,

0.03888888888888889],

light='red', light_violation=1, row='has_row', next_waypoint='left'): [0.19490071106656304, Chooses to stay in place as following the suggested action wouldn't be following the rules

-0.0102484375,

-0.010090769230769231,

0.005303030303030303025]}