Project Report: Smartcab

Implement a Basic Driving Agent

To begin, your only task is to get the **smartcab** to move around in the environment. At this point, you will not be concerned with any sort of optimal driving policy. Note that the driving agent is given the following information at each intersection:

- The next waypoint location relative to its current location and heading.
- The state of the traffic light at the intersection and the presence of oncoming vehicles from other directions.
- The current time left from the allotted deadline.

To complete this task, simply have your driving agent choose a random action from the set of possible actions (`None`, `'forward'`, `'left'`, `'right'`) at each intersection, disregarding the input information above. Set the simulation deadline enforcement, `enforce_deadline` to `False` and observe how it performs.

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: *The agent would behavior more like taking a random walk with random actions. The agent will randomly choose the optional behavior which may not be optimal. The smartcab will make it to the destination sometimes, but most of the time if will fail to find the destination given fixed step. With 500 runs, only 19.6% of the time it reaches destination.*

Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the **smartcab** and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination of inputs as an implicit state. At each time step, process the inputs and update the agent's current state using the `self.state` variable. Continue with the simulation deadline enforcement`enforce_deadline` being set to `False`, and observe how your driving agent now reports the change in state as the simulation progresses.

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?*

OPTIONAL: *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: *I chose to include intersection state and next_waypoint state as my states to consider. For the intersection state, I need it because I don't want the smartcab to be in danger of violation of traffic rules. I need next_waypoint state because the reward is related with the action for such step and the location of the next step.*

Implement a Q-Learning Driving Agent

With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm

for your driving agent to choose the *best* action at each time step, based on the Q-values for the current state and action. Each action taken by the **smartcab** will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement `enforce_deadline` to`True`. Run the simulation and observe how the **smartcab** moves about the environment in each trial.
The formulas for updating Q-values can be found in **this** video.
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: *For the agent I implemented(agent2.py and run.sh), it will behave more reasonably than the random action. This is because we have a Q-value based estimation for each step we move and each step is chosen to maximize the reward.*

Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the**smartcab** is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (`alpha`), the discount factor (`gamma`) and the exploration rate (`epsilon`) all contribute to the driving agent's ability to learn the best action for each state. To improve on the success of your **smartcab**:

- Set the number of trials, `n_trials`, in the simulation to 100.
- Run the simulation with the deadline enforcement `enforce_deadline` set to `True` (you will need to reduce the update delay `update_delay` and set the `display` to `False`).
- Observe the driving agent's learning and **smartcab's** success rate, particularly during the later trials.
- Adjust one or several of the above parameters and iterate this process.

This task is complete once you have arrived at what you determine is the best combination of parameters required for your driving agent to learn successfully.

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*: I tested with different combination of parameters and here is the accuracy (the number of times it reaches destination at given step)*

Alpha	Epsilon	Gamma	Accuracy (%)
0.8	0.1	0.1	95
0.8	0.1	0.01	96
0.8	0.2	0.1	89
0.8	0.2	0.01	92
0.9	0.1	0.1	89
0.9	0.1	0.01	97
0.9	0.2	0.1	87
0.9	0.2	0.01	92

From the result, we see that the accuracy will be affected by the parameter a lot and the agent will perform best with alpha = 0.9, epsilon = 0.1 and gamma = 0.01. It should be noted that the accuracy will be varied by 1 or 2% each run because a lot of random parameters (start and end location, deadline etc) are included in each run

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: *In answer this question, I printed out the updated states for each move in the 100 run for the parameters above. From that, roughly the agent will be able to get the destination with less than five times more steps required (in comparison with theoretical limit of abs*$|x_{desi}-x_{start}|$ *+ abs*$|y_{desi}-y_{start}|$ *which is the ideal case for free-move agent). Five times isn' t a great delay since for each step we need to consider the traffic condition. In comparison with the random walk model, I would say the Q-learning is a success.*