Tasks

Project Report

You will be required to submit a project report along with your modified agent code as part of your submission. As you complete the tasks below, include thorough, detailed answers to each question*provided in italics*.

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 movement was random. It sometimes went to the 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?*

Answer: Use traffic light, next waypoint, oncoming car and left car as a state. Global location is not appropriate for status because this cab cannot get that information, though it's better to have it.

Additional comment: Use traffic light to judge based on the traffic rule, next waypoint to include the factor of current place and goal, oncoming car to judge whether it can turn left on a green light and left car to judge whether it can turn right on a red light. I don't need to use right car information because it doesn't affect any decisions. For deadline, it could be better to use if I let the car to change the strategy, but this time I didn't use it and just used e-greedy method.

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?*

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: By using q value, smartcab continues to improve.

Additional comment: If smartcab fail to follow the traffic rule, it can't get the reward and it improves to follow the traffic rule. If I didn't use e-greedy, smartcab falled into the local minimum sometimes and continued to chose inefficient paths sometimes. By introducing some randomness, smartcab could improve.

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: When I set big number as epsilon like 0.3, success rate became bad. Through some experiments, when alpha was 0.1, gamma was 0.9 and epsilon was 0.1, it worked the best.
Additional comment: Added experimental result below. Epsilon shouln't be bigger than 0.1 because it encourages to choose the random path. For alpha (learning rate), it doesn't make so much difference this time, but it's better to have a small number generally in order to avoid sudden update of q value. For gamma (discount factor), it worked better to have a bigger number. Discount factor would decide how to deal with the future revenue's expectation and higher number means higher reliability for the current policy.

Epsilon	Alpha	Gamma	**Success Rate (average of 5 trials)**
0.1	0.1	0.9	**0.838 (0.84, 0.84, 0.9, 0.85, 0.76)**
0.1	0.1	0.8	**0.744 (0.84, 0.8, 0.58, 0.91, 0.59)**
0.1	0.2	0.9	**0.778 (0.85, 0.67, 0.67, 0.86, 0.84)**
0.1	0.2	0.8	**0.748 (0.69, 0.85, 0.7, 0.77, 0.73)**
0.2	0.1	0.9	**0.74 (0.79, 0.49, 0.82, 0.79, 0.81)**
0.2	0.1	0.8	**0.67 (0.56, 0.79, 0.51, 0.74, 0.75)**

0.2	0.2	0.9	**0.77** **(0.69, 0.84, 0.73, 0.86, 0.73)**
0.2	0.2	0.8	**0.746** **(0.78, 0.73, 0.64, 0.77, 0.81)**

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: Yes, it worked well though it didn't work perfectly because of e-greedy method.

Additional comment: The optimal solution is to explore various paths at the beginning, but to converge at the last moment. When I see the q-learning table, the waypoint and action are becoming same through the iteration.