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: With the random chosen actions,meaning the agents don't learn from the previous driving behavior and improve their actions. The smartcabs performed terrible. Because of the random chosen action, they disregards the traffic lights, or if there is a oncoming car. Because there is no limit deadlines, the random agent eventually can reach 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: the states I choose are “traffic” and “next_waypoint”. The traffic choice is because we want the smartcab to perform correctly since there is a traffic law we have to obey. And the traffic law has a close relation with rewards. So traffic is good one to choose. The reason I chose “Next_waypoint” is because it is the path which leads from the current location to the destination. Working with the traffic law together, it can help decide the best action for next step.

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 think there should be 4 in total. Traffic, Next_waypoint, deadline and destination. The reason why I didn't choose deadline and destination is because they are both set up in the beginning of the each trial, especially the destination, it won't change on our way to the destination. Although knowing destination is kinda nice, but there are lots of ways to reach the destination, it won't affect a lot in deciding which action should be taken at each traffic intersection. The deadline is also a setup value in the beginning of each trial, and it decreases by one at each step, which doesn't seem helpful to decide what's the best action next.

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: Q-learning is implemented in Qlearn.py. Compared with the smartcab's random actions with single digits success of reaching the destination, With the Q-learning coming to play, the smartcab can now learn how to drive not only based on the traffic rules, but also learn how to drive to win rewards. With the Q-learning implementation, the smartcab can learn from each state it drives through, remember and compare with all the possible actions associated with reward, and pick the best actions with the best reward. The chance of success of reaching the destination raises significantly.

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?*

Alpha	Gamma	Epsilon	Success rate(#success to destination/100)
0.2	0.2	0.5	69/100
0.2	0.2	0.2	91/100
0.9	0.2	0.2	93/100
0.9	0.2	0.02	97/100
0.9	0.2	0.01	98/100
0.9	0.7	0.01	98/100
0.9	0.02	0.01	97/100

Answer: Based on the parameters' combination test, the best results come from both (Alpha = 0.9, Gamma = 0.2, Epsilon =0.01) and (Alpha = 0.9 Gamma = 0.7, Epsilon = 0.01), with almost 98% of agents success to destination.

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: After applied the learning optimal policy, the agent gets to the destination much quicker in the end(fewer steps relatively). Which means the agent is cable to find a optimal route to reach the destination faster. Also, the smartcab always gets positive rewards after reach the destination, which means the smartcab also learned how to make actions by following the traffic rules. So the possible optimal policy in this problem would be the policy which has higher rate of success, capable to follow all the traffic rules and find the best routes to the destination.