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 smartcab is randomly driving around in the grid. By chance it can reach the destination. Actually, with the enforce_deadline=False there is a 67% chance that the smartcab will make it to the destination before the hard deadline of -100. In this case the expected reward is about -2.7.*

If we enforce a deadline, then chances are only 20% that the smartcab will reach destination before the deadline has elapsed. In this case the expected reward is about 0.4.

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: *A state is all the information necessary to make a decision that you expect will take you to a new (higher value) state. For this project I tried to keep the number of states to the minimum but providing the smartcab with the maximum amount of information. I have identified the following states:*

- *nextWaypoint {forward,right,left}, which is the next waypoint provided by the planner. In order for the smartcab to take a decision (action), it needs to know first of all where it needs to go to in the next time step.*
- *stateTrafficLight {'green','red'}*
- *oncoming {'forward','left','right'}: moving direction of oncoming traffic*
- *left {'forward','left','right'}: moving direction of traffic coming from the left*
- *right {'forward','left','right'}: moving direction of traffic coming from the right*

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: There are in total 3*2*3*3*3= 162 states in this smartcab environment. Q-learning evaluates the value of all possible state/action pairs. The smartcab can take 4 possible actions in each state, which gives us 4*162=648 possible state/action pairs. With this setup it takes less than 1 hour to do 1 million trials, which should be enough to learn and make informed decisions about each state.

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: The smartcab is reaching more often the destination before the deadline expires and is moving less erratic (driving in circles or in the last step driving away from the destination instead of towards the destination. In the Q-learning algorithm, the epsilon parameter defines the trade-off between exploration and exploitation. The bigger epsilon the more the smartcab will explore random actions and visit new state action pairs. The smaller epsilon, the more the smartcab will chose the best action for a specific state. In each time step the state action value is updated with the reward that resulted from that specific action. By applying the Q-learning algorithm many times, the smartcab will learn for which actions it receives the highest awards and for which actions it will be punished. In other word the smartcab will learn the traffic rules.

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: For me the best result was achieved epsilon=0.008, alpha= 0.29 and gamma=0.7. For this combination of parameters, the model achieved a success rate of approx. 80% and an average reward of 22.

Finding the right combination of the 3 parameters required a lot of computational effort. I also tried an approach where each of the 3 parameters would linearly decrease over time from an initial value of 100, to a value of 0 for the last trial. This method has a success rate of 70% with an average reward of 26.

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: No. An optimal policy would in the first place represent a safe driving behaviour of the smartcab and in a second place result in high rewards i.e. arrival in time. One can encourage certain behaviour by tuning the rewards. In the code that was provided, the smartcab pays a penalty of -1 in case of a traffic violation. With these reward settings, the smartcab will make 24% of its actions in an illegal way. If we for example increase the penalty from -1 to -1.5 in case of a traffic violation, then only 5% of the smartcab actions are illegal because it becomes too costly. By tuning the rewards and penalties, one can optimise the smartcab policy towards the desired behaviour.