> MLE Nanodegree
> Philipp Vogler
> March 2016

P4: Train a Smartcab to Drive

1. Implement a basic driving agent

** In your report, mention what you see in the agent’s behavior. Does it eventually make it to the target location? **

The line action = random.choice([None, 'forward', 'left', 'right'])` makes the agent choose a random action. It makes its way through the world and to the target location by chance. Usually, it takes quite a lot of time until the target location is reached. The agent does not learn.

2. Identify and update state

** Justify why you picked these set of states, and how they model the agent and its environment.**

I put everything into the state that the sensors record and that is relevant to the action decision. This includes the recommended next waypoint and the inputs about the environment (left, light, oncoming, right).
The deadline and the time step are not essential to the decision. They also would increase the number of states significantly, if recorded. I did not include deadline and time step in the state. They are used to adjust the learning and the exploration rate.
Later I also scraped the 'left' input from the state, because it is not relevant in a right-of-way environment.

3. Implement Q-Learning

 ** What changes do you notice in the agent’s behavior? **

 In the beginning, the actions are still random, due to the random initialization of the Q table.
 After many runs, the agent starts to prefer certain actions in certain states. In some cases, this is useful to reach the target destination in other cases it leads to problems like deadlock situations or right loops. The agent also prefers the "None-action", due to the reward structure.

4. Enhance the driving agent

** Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform? **

* Reduce the count of states/keys in the Q table
    In my fist Q table, there was the possibility for the recommendation for the next waypoint to be NONE, which is a state

that the planner does not provide.
    Also, I realized that with right-of-way rules the left- sensor is quite unimportant for the decision. I removed it from the state.
    These measures reduced the number of states in the Q table significantly.

* Set the initial values of the Q table in regard to the rewards.
    To have "realistic" start values in the Q table, I initialized them with numbers from 0 to 2. This is what the regular maximum reward is. (Except hitting the target.)

* Improving the action selection method
    I changed the action selection in a way that the recommendation to reach the next waypoint is favored about the other possible actions. This counteracts the preference for the "None-action".

* Introduction of gamma to solve the exploration-exploitation dilemma
    I introduced an exploration rate that should help to cover more states in the Q table. The way it is implemented with regard to the deadline minimizes the unwanted side effect of exploration under time pressure.
    It helped in particular against the clockwise cycling of the agent.

* Introduction of a learning rate
    The learning rate remarkably improved the speed of the agents learning. The results improve after fewer runs.

** 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? **

No, the agent gets better, but not optimal. With the exploration rate, there is a random element in the decision process that prohibits 100% optimal behavior. Furthermore, the agent is not preventing penalties at all costs.