## Tasks

### Setup

You need Python 2.7 and **[pygame](http://pygame.org/" \t "_blank)** for this project: [https://www.pygame.org/wiki/GettingStarted](https://www.pygame.org/wiki/GettingStarted" \t "_blank)  
For help with installation, it is best to reach out to the pygame community [[help page](http://www.pygame.org/wiki/info" \t "_blank), [Google group](https://groups.google.com/forum/" \l "!forum/pygame-mirror-on-google-groups" \t "_blank), [reddit](https://www.reddit.com/r/pygame/" \t "_blank)].

### Download

Download [smartcab.zip](https://s3.amazonaws.com/content.udacity-data.com/courses/nd009/projects/smartcab.zip" \t "_blank), unzip and open the template Python file agent.py (do not modify any other file). Perform the following tasks to build your agent, referring to instructions mentioned in README.md as well as inline comments in agent.py.

Also create a project report (e.g. Word or Google doc), and start addressing the questions indicated in italicsbelow. When you have finished the project, save/download the report as a PDF and turn it in with your code.

### Implement a basic driving agent

Implement the basic driving agent, which processes the following inputs at each time step:

* Next waypoint location, relative to its current location and heading,
* Intersection state (traffic light and presence of cars), and,
* Current deadline value (time steps remaining),

And produces some random move/action (None, 'forward', 'left', 'right'). Don’t try to implement the correct strategy! That’s exactly what your agent is supposed to learn.

Run this agent within the simulation environment with enforce\_deadline set to False (see run function inagent.py), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current state, action taken by your agent and reward/penalty earned are shown in the simulator.

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

There is no guarantee that the agent will ever make it to the target location. All directions are uniformly chosen at random, so there is no way of telling how likely it is that the agent will ever reach its destination.

### Identify and update state

Identify a set of states that you think are appropriate for modeling the driving agent. The main source of state variables are current inputs, but not all of them may be worth representing. Also, you can choose to explicitly define states, or use some combination (vector) of inputs as an implicit state.

At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.

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

These are the states that can be immediately observed by the learning agent. The agent should pay attention to its future location, whether the light is green and allowing it to move, and if there are cars oncoming from any direction. The deadline is important to include in the state once we start enforcing it because it will mean that a terminal state has been reached. This shortens each trial, but we receive no penalty for not finishing on time. Some states were unnecessary, however. The oncoming traffic from the right is nonessential if you pay attention to the color of the light. On the other hand, after reading some discussion and looking over the code, it appears that the oncoming and left oncoming traffic also do not matter because the agent is already programmed to acknowledge right-of-way. This is helpful because it severely reduces the input space that the agent needs to learn. If this were not the case, then it could still be argued that the only other input dimension to add to the state would be oncoming traffic. That would make sure that the agent learned not to make left turns when there is oncoming traffic.

### Implement Q-Learning

Implement the Q-Learning algorithm by initializing and updating a table/mapping of Q-values at each time step. Now, instead of randomly selecting an action, pick the best action available from the current state based on Q-values, and return that.

Each action generates a corresponding numeric reward or penalty (which may be zero). Your agent should take this into account when updating Q-values. Run it again, and observe the behavior.

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

The agent starts to take the None action more as it observes a +1 for every time it does not run a red light. It really isn’t encouraged to find the goal because it receives rewards for following correct rules of the road. It does not receive a huge negative reward for not finding the destination, so it makes sense that it would try to receive multiple small rewards instead of searching and making a mistake.

### Enhance the driving agent

Apply the reinforcement learning techniques you have learnt, and tweak the parameters (e.g. learning rate, discount factor, action selection method, etc.), to improve the performance of your agent. Your goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive.

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?

I tried implementing the ego-centric and allo-centric models in the tutorial provided at this link:

<https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration/>

This was to encourage more exploration without encountering the issue of the agent choosing the None option because it gives short-term returns.

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?