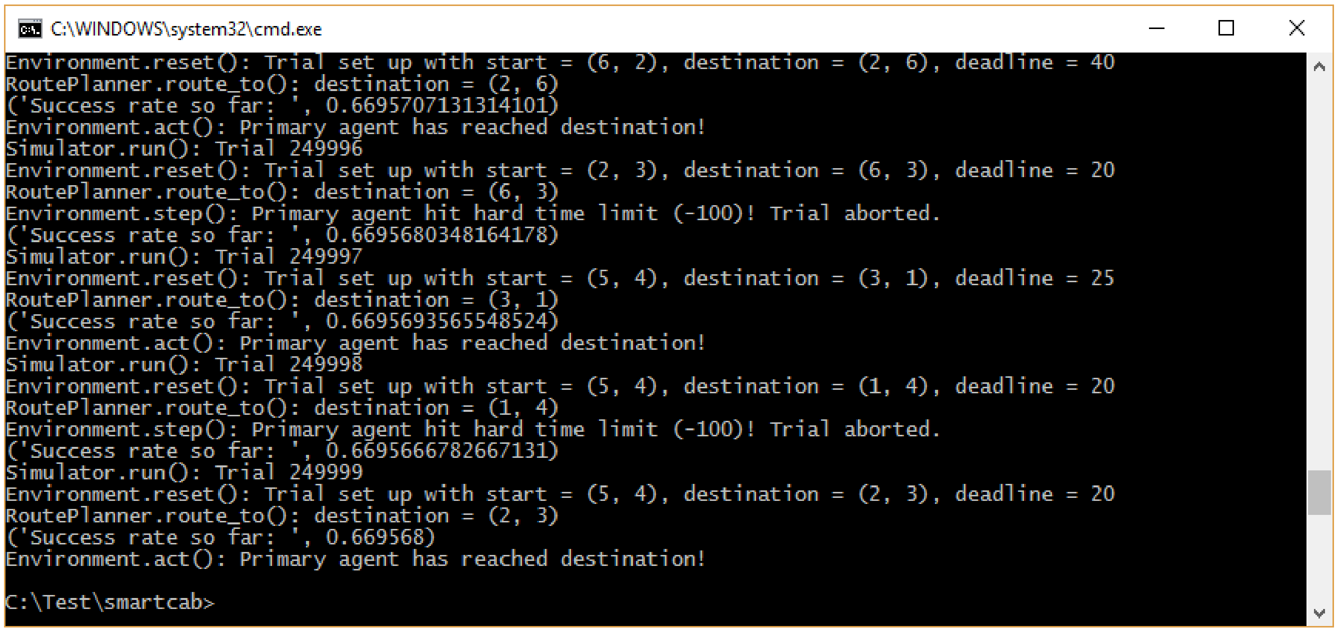
Implement a Basic Driving Agent

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?*

The smartcab eventually makes it to the destination in many trials. However, because it is driving without regard to the next waypoint, the route is not efficient and the success rate is not as high as it could be. After 250,000 trials (without enforcing the deadline, but a hard limit of t = -100), the cab arrived at its destination about 2/3 of the time.



```
Environment.reset(): Trial set up with start = (6, 2), destination = (2, 6), deadline = 40
RoutePlanner.route_to(): destination = (2, 6)
('Success rate so far: ', 0.6695707131314101)
Environment.act(): Primary agent has reached destination!
Simulator.run(): Trial 249996
Environment.reset(): Trial set up with start = (2, 3), destination = (6, 3), deadline = 20
RoutePlanner.route_to(): destination = (6, 3)
Environment.step(): Primary agent hit hard time limit (-100)! Trial aborted.
('Success rate so far: ', 0.6695680348164178)
Simulator.run(): Trial 249997
Environment.reset(): Trial set up with start = (5, 4), destination = (3, 1), deadline = 25
RoutePlanner.route_to(): destination = (3, 1)
('Success rate so far: ', 0.6695693565548524)
Environment.act(): Primary agent has reached destination!
Simulator.run(): Trial 249998
Environment.reset(): Trial set up with start = (5, 4), destination = (1, 4), deadline = 20
RoutePlanner.route_to(): destination = (1, 4)
Environment.step(): Primary agent hit hard time limit (-100)! Trial aborted.
('Success rate so far: ', 0.6695666782667131)
Simulator.run(): Trial 249999
Environment.reset(): Trial set up with start = (5, 4), destination = (2, 3), deadline = 20
RoutePlanner.route_to(): destination = (2, 3)
('Success rate so far: ', 0.669568)
Environment.act(): Primary agent has reached destination!

C:\Test\smartcab>
```

Because the smartcab is driving without regard to any traffic laws or the reward function, it can choose actions which are either illegal (such as trying to move forward when the light is red) or not the next waypoint. Because of these sub-optimal moves, the smartcab tends to accumulate more negative rewards than positive rewards.

Lastly, because of the settings governing the Environment class, the cab is allowed to drive off the edge of the grid and re-appear at the opposite end.

Inform the Driving Agent

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?*

The 'smart' cab maintains an egocentric of the intersection it is faced with. The states necessary for modeling the smartcab and the environment include:

- state of the light at the intersection
- state of oncoming traffic at the intersection
- state of traffic at the intersection on the left
- state of traffic at the intersection on the right
- direction of the next waypoint relative to the intersection

Initially, one might think that the state space includes the precise grid location of the smartcab, but for the current situation, this will not be necessary. The smartcab is on a uniform grid without any variation in topography or roads. If the state space included grid coordinates / headings this would blow up the state space and make the problem infeasible due to the computational cost. With respect to the map, the only variable that matters is the next waypoint. The smartcab does not act based on its grid location or the location of the destination.

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?*

Using the definition of the state from above, the number of states in existence for the smartcab can be thought of as similar to the sample space of possible 'outcomes' that the smartcab faces at the intersection:

= Status of light * Status of oncoming car * Status of car on right * Status of car on left * Possible Waypoints

= ['red', 'green'] * [None, 'forward', 'left', 'right']^3 * ['forward', 'left', 'right']

= 2 * 4 * 4 * 4 * 3 = 384

For each of the three other directions at the intersection, the smartcab treats the non-existence of a car the same as a car which is taking no action (perhaps the smartcab can perceive it as parked or idled).

For 100 trials, which could last about 20+ states, it is likely that the algorithm will be able to learn the ideal behavior properly, as most of the time the intersection will be absent of other cars. However, given the small number of trials, it will be necessary to tune the learning parameters alpha, epsilon, and gamma so that the exploration/exploitation trade-off is balanced.

Implement a Q-Learning Driving Agent

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?*

At first, the agent continues to move in a random behavior, not always hitting the next waypoint or obeying traffic laws. As it continues learning state/action Q-values, its accuracy in reaching the destination before the deadline increases. The smartcab begins to learn that the next waypoint is important while still obeying traffic laws. This behavior is occurring as the smartcab encounters the same state over and over again, and gradually "learns" the traffic laws through positive and negative rewards. The more frequently the agent has visited a state, the more likely the Q-values properly reflect the rewards for taking each action.

Improve the Q-Learning Driving Agent

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?

The parameters which could be tuned for the implementation of Q-learning include:

- alpha, the learning rate
- epsilon, the probability of a random choice of action at each state
- gamma, the weight between future and present

Several permutations of the above settings were conducted as 10 runs of 100 trials and the results are recorded below:

Alpha	**Epsilon**	**Gamma**	**Median Success Rate**	**Median Ratio Penalized Moves**	**Median Total Reward**
0.5	0.05	0.5	0.89	0.1505	2390
0.75	0.05	0.5	0.89	0.08427	2208.5
0.75	0.05	0.75	0.811	0.1493	2328.5
0.9	0.05	0.5	0.92	0.0776	2207.95
0.9	0.01	0.5	0.58	0.04465	1691.25
0.9	0.08	0.5	0.925	0.1238	2266
0.9	0.08	0.8	0.855	0.184	2331.25
0.9	0.08	0.3	0.945	0.10615	2244
0.9	0.08	0.1	0.93	0.07265	2173
0.95	0.05	0.05	0.91	0.05535	2122.75
0.95	0.08	0.25	0.94	0.0922	2178.5

The smartcab manages to reach its destination before the deadline roughly 94.5% of the time when alpha is 0.95, epsilon is 0.08, and gamma is 0.3. The next highest success rate is with only a small change in gamma to 0.25. It seems the smartcab does best when the learning rate is high, there is small chance of random exploration, and future rewards are discounted heavily (low gamma).

The smartcab has come close to an ideal policy since it reaches the destination within the allotted time without incurring a large number of penalized actions. Some penalized actions are necessary since the smartcab needs to both explore and exploit.

The lowest ratio of penalized moves occurred with the lowest epsilon. This shows that despite a low rate of exploration resulted in a low rate of penalized moves. However, this also resulted in a very low success rate.

In order to try to gain a better success rate, the initial Q value will be adjust from a default 0 to slightly higher at 2. This follows the guidance of 'being optimisitc in the face of uncertainty.' Under these conditions the following results were obtained:

Alpha	**Epsilon**	**Gamma**	**Initial Q**	**Median Success Rate**	**Median Ratio Penalized Actions**	**Median Total Reward**
0.9	0.08	0.3	2	0.965	0.12025	2310.25
0.9	0.01	0.1	2	0.99	0.0325	2224

Since the initial Q value dictates that more exploration will be done, the epsilon was lowered to reduce the probability of exploration occurring randomly. This has yielded a very high success rate at 0.99 and lowered the ratio of penalized actions under 4%.

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?

An ideal policy would be more specifically tailored to traffic laws. While the state space is large at 384, the cases generally can be described by the following situations.

1) Next waypoint = Forward; Light = Green
2) Next waypoint = Right; Light = Green
3) Next waypoint = not Right; Light = Red
4) Next waypoint = Left; Light = Green, Oncoming = Left, None

5) Next waypoint = Left; Light = Green, Oncoming = Forward, Right
6) Next waypoint = Right; Light = Red, Left = Straight
7) Next waypoint = Right; Light = Red, Left = Right, Left

The smartcab is permitted to legally take action towards its next waypoint in states 1, 2, 4, 7. The smartcab is not permitted to legally take action towards its next waypoint in states 3, 5, 6. For cases 1, 2, 4, and 7, the smartcab should proceed to the next waypoint. For cases 3, 5, and 6, the smartcab should idle one time period, observe the state, and then act according to the above choices.

Since each traffic light stays red on average 4 seconds, and the deadline for the smartcab to arrive is set at 5 * distance, there should be enough time for it to reach the destination by obeying both traffic laws and the next waypoint.

An interesting observation is if the smartcab and other drivers are observing traffic rules, the driver who is on the right at the intersection has no impact on the smartcab's next action. Therefore the original 384 states could have been condensed to 96 from the beginning. The code has been altered in agent96.py to limit the number of states. Running this model under the ideal starting variables results in the following:

Alpha	**Epsilon**	**Gamma**	**Initial Q**	**Median Success Rate**	**Median Ratio Penalized Actions**	**Median Total Reward**
0.9	0.01	0.1	2	0.995	0.0246	2228.5

With a higher initial Q value, along with optimized alpha, epsilon, and gamma, the smartcab makes quick work of learning the ideal policy in a reduced state space. It nearly always arrives in time to its destination and less than 3% of its actions are penalized. It appears the agent is doing the bare minimum of learning necessary and then applying it as a policy successfully. The performance with a state size of 96 is marginally superior than with 384, as expected.