In [1]:
```
from IPython.display import Image
```

Implement a Basic Driving Agent

1. 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?

Making the smartcab take random actions means that the smartcab is just driving around the city with no particular purpose. It eventually reaches its destination when the deadline is not enforced, but it's very frustrating to see that sometimes it's very close to reaching its destination and just begins to drive away from it. The cab pays no attention to the rewards as it does not care about reaching the destination on time or avoiding infractions.

Inform the Driving Agent

2. 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?

- **Traffic light color**. This state is important because the cab needs to know if it has to stop at an intersection or if it can continue driving.
- **Cars around us and their directions**. This is important in order to understand if we can move forward, turn left on a green light, or turn right on a red light, according to U.S. Right-of-Way rules. The information about oncoming traffic is only relevant if we know our planned direction, which takes us to the next point.
- **Next waypoint**. It is important to know the "recommended" or "planned" direction to have a sense of the direction in which we should be moving. We might want to follow this recommendation or not based on other inputs, but knowing this is helpful to eventually reach our destination.
- **Deadline**. This is important to know since we know the cab won't receive any reward if it does not reach its destination on time. It is important to notice that the deadline is expressed as the number of additional steps we can take before we run out of time. If the deadline is, say 40, this means that if this input is not reduced, we could end up having up to 40 additional states for each combination of inputs described above (see next section). Therefore, I chose to define a new variable to determine if we should "hurry up" (meaning that we have less than 20 steps) or maybe we can explore a little bit more.

All these inputs will help us identify the state the smartcab is in. In this particular case the states will have the format: *{light}-{oncoming}-{left}-{hurry_up}-{next_waypoint}*, resulting in something that looks like this: **green-None-None-yes-forward**

Note: Given the U.S. Right-of-way rules, the input traffic coming from the "right" is not needed for this model. Removing it reduces the size of the state space.

2.1 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?

There exist a total 2 x 4 x 4 x 2 x 4 = 256 states in this environment:

- Lights = 2
- Oncoming = 4
- Left = 4
- Hurry Up = 2
- Next Waypoint = 4

Combining these inputs into a single string, we get inputs like 'green-None-None-yes-forward', which can later be used to take actions.

256 states are a reasonable number of states for this problem since the smartcab will make around 100 trips in 1 round, and can take around 30-40 steps in each trip. This means that the cab will be making about 3000-4000 steps in total, enabling the smartcab to visit the most common states more than once, and learn during the whole process. If we were to add the raw deadline as part of the state then the problem would be more difficult, since we would have around 128 * 40 = 5,120 different states, making it very unlikely for the car to be in the same state more than two times, and thus learn.

Implement a Q-Learning Driving Agent

3. 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?

Now the car has an objective, or at least knows the policy that will get it closer to that objective (maximizing rewards). By following the actions that maximize the rewards, the cab tries to reach its destination and take actions that earns it points, as opposed to taking random actions.

This behavior is occurring precisely because the smartcab is trying to maximize its rewards, and now it has memory of the reward it has obtained in previous steps, as well as the future reward that making an action will give it.

Improve the Q-Learning Driving Agent

4. 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 agent performs best when the immediate reward has a higher weight than the future reward. In a sense this is an expected behavior, since the cab only has the input of the environment at a particular moment (before it acts), but the environment might have changed when it reaches the next state. In other words, the environment is dynamic. For example, the cab *knows* that if it makes a right turn now, the light at the next intersection is currently green. But the reality is that when the cab reaches that intersection, it might have changed to red. Knowing this in the previous step might have motivated the cab to take some other action.

The agent also performs best when it follows what it learns (exploiting) but still has some room to explore, and when it learns a lot from past experiences. This is also expected since once the cab knows that if it speeds ahead on a red light it will get penalized, taking this same action in another part of the map or at a later point in tame will not change the outcome.

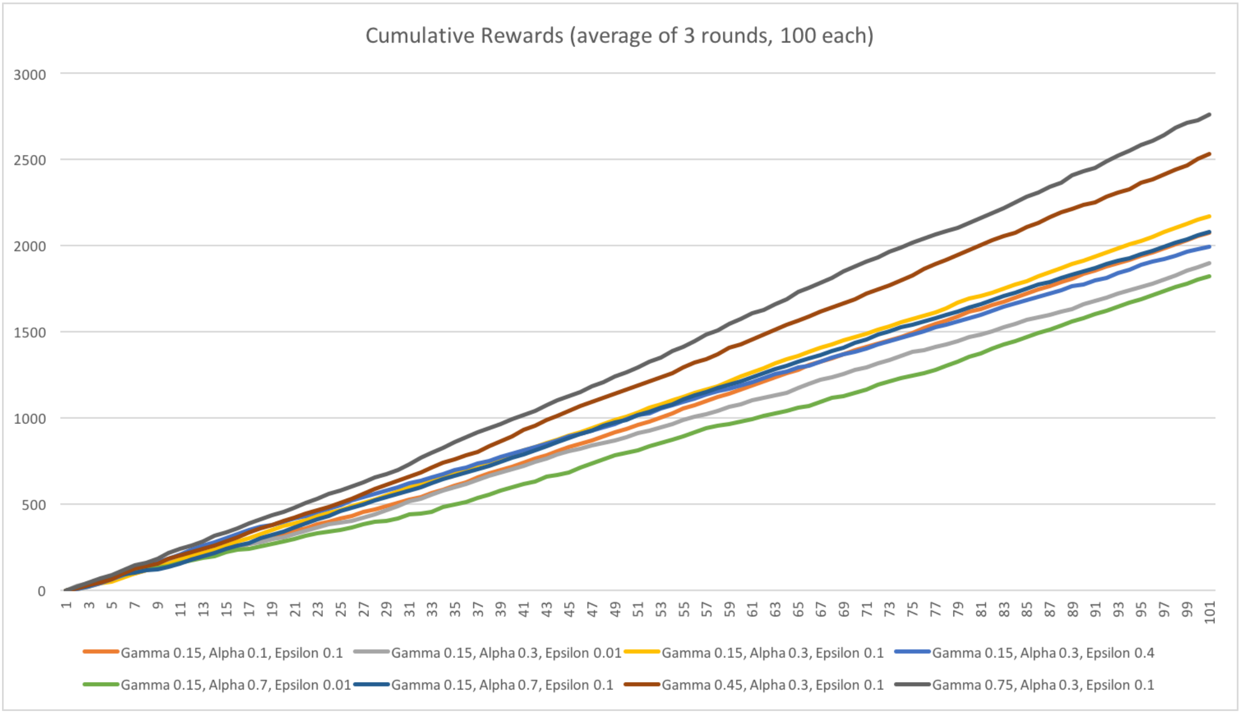
Below I summarize a set of parameters that I tested and the cumulative rewards after the 100 trips. In each case, I ran the simulation 3 times and averaged out the results in order to get a better idea of the potential reward. For example, in scenario H (Gamma=0.75, Alpha 0.30, Epsilon=0.10) the cumulative rewards of the 3 rounds were 2752, 2706, and 2823. The average is 2760, which is the number included in the table.

Scenario	Gamma	Alpha	Epsilon	Cumulative Reward (avg 3 rounds of 100 trips each)
A	0.15	0.70	0.01	1825
B	0.15	0.30	0.01	1899
C	0.15	0.30	0.40	1996
D	0.15	0.10	0.10	2074
E	0.15	0.70	0.10	2082
F	0.15	0.30	0.10	2172
G	0.45	0.30	0.10	2535
H	0.75	0.30	0.10	2760

As seen in the image below, all the different scenarios start with the same cumulative rewards, but the dark-gray line (Gamma 0.75, Alpha 0.3, Epsilon 0.1) is the one that is learning to maximize the results at a higher rate early on.

```
In [2]: Image('cumulative_rewards.png')
```

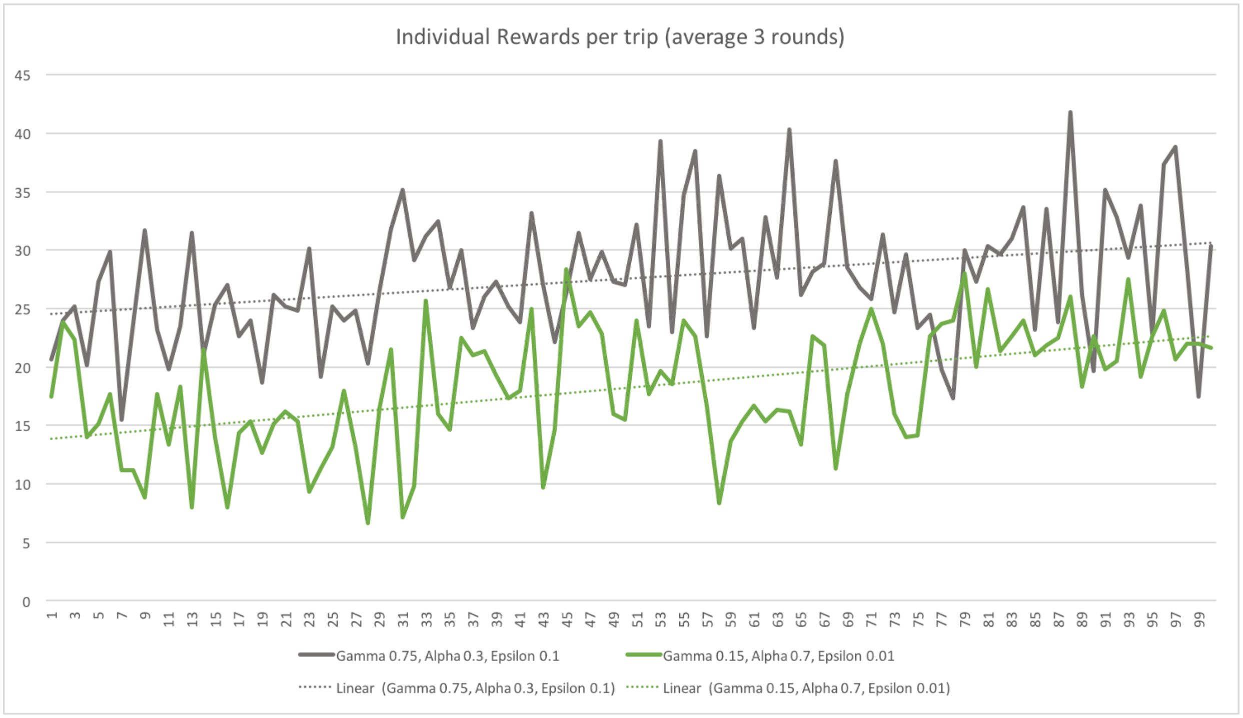
```
Out[2]:
```



A way to understand why one line grows faster than the others is by understanding the individual rewards the cab gets on each trip. As we can see in the graph below, where we compare the best performing parameters (Gamma 0.75, Alpha 0.30, Epsilon 0.10) vs the worst performing parameters (Gamma 0.15, Alpha 0.70, Epsilon 0.01) we see that the best performing parameters are consistently achieving higher rewards. An interesting fact to note is that both show improvements as trips are made, meaning that in both cases the agent is learning.

```
In [3]: Image('individual_rewards.png')
```

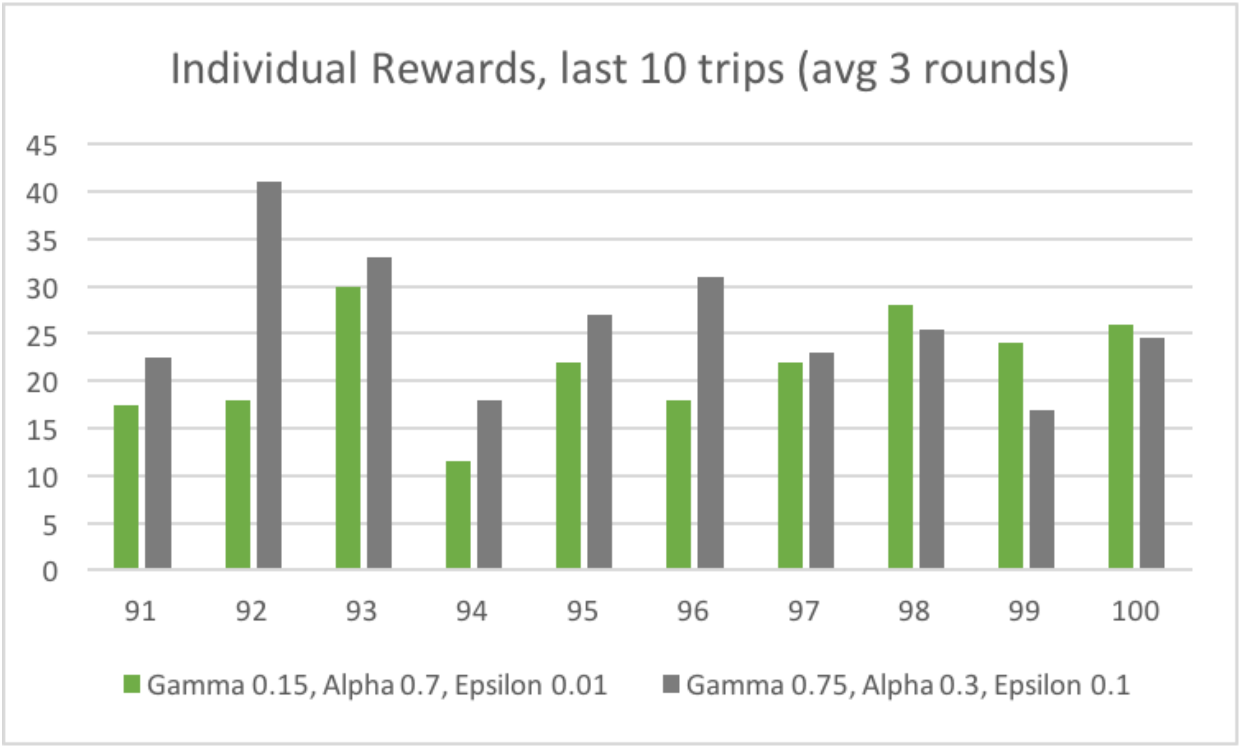
Out[3]:



In the image below (Individual Rewards, last 10 trips), we see that the worst performing model has still learned a great deal, but is being outperformed by the other model. The total rewards for the last 10 trips are 217 and 263, respectively.

```
In [4]: Image('individual_rewards_last_10_trips.png')
```

Out[4]:



5. 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?

The optimal policy of the best performing model (Gamma 0.75, Alpha 0.3, Epsilon 0.1) is to maximize the rewards by driving around and even having small infractions, and yes, reach the destination when it makes sense.

In the table below we can see the comparisons between the worst performing model (left) and the best performing model (right). We see that in the first one, the agent is trying really hard to reach the destination on time, while on the second one, the agent reaches the destination in 8 out of the last 10 trips.

Last 10 trips: did the agent reach the destination on time? (1.0 = yes)

Trip	Gamma 0.15, Alpha 0.7, Epsilon 0.01	Gamma 0.75, Alpha 0.3, Epsilon 0.1
91	1.0	1.0
92	1.0	1.0
93	1.0	1.0
94	1.0	1.0
95	1.0	-
96	1.0	1.0
97	1.0	1.0
98	1.0	1.0
99	1.0	1.0
100	1.0	-

It’s important to notice also that in the worst performing model, the agent is trying very hard to avoid doing something "wrong" (only 2 out of 10 trips have infractions or negative rewards), but in the best performing model, the cab does not really care if it will get a negative reward (either by not following the plan or by having a small infraction) if this means that it will enable it to maximize the reward for that trip.

Negative Rewards / Total Rewards, last 10 trips:

Row Labels	Gamma 0.15, Alpha 0.7, Epsilon 0.01	Gamma 0.75, Alpha 0.3, Epsilon 0.1
91	7.7%	30.0%
92	-	26.1%
93	-	30.0%
94	50.0%	-
95	-	28.6%
96	-	48.4%
97	-	42.9%
98	-	28.6%
99	-	33.3%
100	-	28.6%

It is interesting to notice that the "wrong" actions that the cab is making in the best performing model are ones related to not following the plan as it was originally conceived (for example, red light, no cars coming, and plan is telling it to go forward... instead of waiting for a green light and then going forward, the agent just turns right since this will enable it to continue getting rewards in other states). It does this because it has learned that even if that action will have an immediate negative impact, the gain in the following states will be worth it. These are some examples:

- ('red-None-None-yes-left', 'right')
- ('red-None-None-yes-forward', 'right')
- ('red-None-None-yes-forward', 'right')
- ('red-None-left-yes-forward', 'forward')
- ('red-None-None-yes-forward', 'right')
- ('green-left-None-yes-right', 'forward')