Observe what you see with the agent's behavior as it takes random actions. Does the smart cab eventually make it to the destination? Are there any other interesting observations to note?

At this point the smart cab might make it to the destination, but since all actions are taken randomly, it would be because of luck. Regarding to other interesting observations, the smart cab is completely naive at it would try to do illegal turns or go straight through a red traffic light; the agent could even drive off the side of the grid. The smart cab knowledge of what's going out in it's environment is null, and that's why it doesn't care/prefer doing the best actions to arrive to destination.

What states have you identified that are appropriate for modeling the smart cab and environment? Why do you believe each of these states to be appropriate for this problem?

The appropriate states for modeling the smart cab and environment consist of a tuple built up by the following information:

- $Light \in \{green, red\}$: Traffic light color at current intersection. It is appropriate for this problem given that it allows the car to decide if what it should do in a given intersection. Not knowing about it would lead to try invalid movements (penalizing the vehicle and losing time) or crashes.
- $Next \in \{left, forward, right\}$: Direction suggested by planner that will lead us to the destination point. It is appropriate for this problem given that it gives the smart cab an idea of where the destination point is located. Without it, the smart cab has no idea where it should go to.
- $Oncoming \in \{left, forward, right\}$: Relative direction of other vehicle at current intersection.
- $Left \in \{left, forward, right\}$: Direction that vehicle at the left (if any) is going to take after current intersection.

The last two elements of the tuple are appropriate for this problem, given that if the smart-cab knows what decisions other cars are taking, it can decide to find a better path to arrive to destination. Not knowing where other cars are moving could lead to car crashes are going through busy routes.

Notice that the $Right$ sub-state is not needed for modeling the environment of the smart cab, given that we're assuming that all vehicles in the environment follow the U.S. Right-of-Way rules. According to those rules, we should pay attention to the traffic coming from our right if the

traffic lights are green and we are trying to turn left. Nevertheless, if the traffic light is green for the smart cab it means that no traffic from the right hand side should go straight the traffic light, and thus we don't need to check whether there's a vehicle to the right of the intersection (from the smart cab frame of reference).

The $Deadline$ sub-state is not used for modeling the environment or smartcab given that it would increase the size of the space in about 20 times (or more, depending on the dest. point). On the other hand, the deadline sub-state provides us some information about how fast we should arrive to the destination point. Nevertheless, we want to arrive always in the least amount of time, so this information is not very relevant and the benefit of this information is much smaller than the disadvantages of increasing our space.

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?

For the model description given above, In total, there are 96 possible different states, given by:

$$2\,(light) * 3\,(oncoming) * 3\,(left) * 3\,(next) = 54\,.$$

It seems like a big number of states given that the agent has to explore every single one at least one time (preferably more than one), in order to arrive to the global optimum. Nevertheless, given that there are 100 iterations of different trips from origin to destination point, and that for each trip the agent might take 20 or more steps, there is enough time for the agent to visit every space state at least 37 times.

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?

As the agent tries to get to the destination point, the agent reaches the destination with shorter trips. This means that as time goes on, the agent starts making more accurate predictions on what are the best actions to make, and it's decisions are less random. This happens because for every single transition that the agent could make from a state *s,* it knows which action *a* will give the highest reward to its behaviour. And every time the agent explores a new state or arrives to a previously visited state, its utility for that given state gets more and more accurate. It is important to notice that even though the smart cab has learned for a long time, there is a

chance that it still makes a random choice, so it can explore further the total space of the environment.

Other noticeable changes in the agent's behaviour occur after the initial trials have occurred. Initial trials look like random walk and start moving towards a more intelligent path finding. The smart cab starts learning to follow the rules of the road, even though it seems that it is getting stuck in local minima, where you might find it going in circles at times. The agent learns to follow the best next_waypoint but takes into account the environment state, so if the planner tells the agent to go right and there's a vehicle there, it will try to maximize the possible reward.

One last thing to notice is that little by little, the behaviour of the smart cab starts changing, and sometimes it might seem like after iteration i the agent behaviour changes suddenly. This was really interesting for me.

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?

For the implementation of the Q-Learning algorithm I used a parameter named `INIT_VAL`, which determined the initial value for every state in the model space. It was assigned to a high value (in comparison to the rewards magnitudes) of `200`, and was done this way following the idea of '*optimism in the face of uncertainty'*. This idea consists in that using high initial Q-values lead to an exploratory-learning agent, because it will delay exploiting familiar paths, since those will end up with lower Q-values than its initial estimate. In other words, using high initial values implies having an "optimistic" agent that initially believes all possible actions will yield excellent rewards.

Additionally, the implementation made applies a term known as epsilon decay. It consists in decreasing the epsilon value as the agent trains more. In this case, epsilon decreases every time the agent completes a trip in the given time. It is a valid modification to the algorithm that is based in the fact that if the agent is doing better, the chance of a random act should decrease until it reaches a threshold (because it means it has learnt) . That threshold is 0.05 or 5%, and the rate used for the decay is 0.1.

On the other hand, each parameter $\alpha\,(alpha)$, $\beta\,(beta)$, $\gamma\,(gamma)$ used to tune the basic implementation of Q-Learning could take any of the following values:

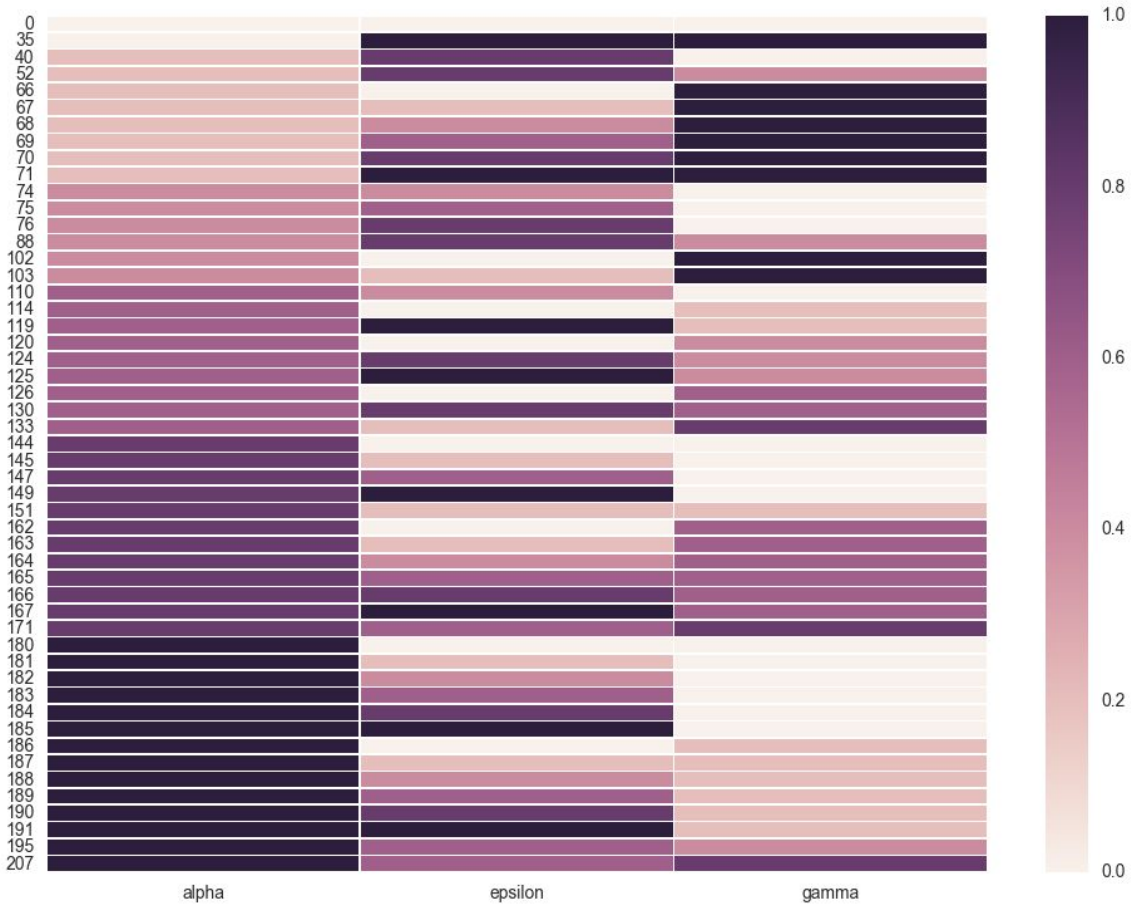
0.0	0.2	0.4	0.6	0.8	1.0

Thus, giving a total of 216 different combinations. Training the car was done 20 times with a number of 100 trials every time, for each distinct set of parameters alpha, beta and gamma. It was done in this way in order to have a normalized dataset that wouldn't depend on the randomness of the origin-destination configuration that the simulation creates. Results for the data sampled with each combination of parameters was averaged.

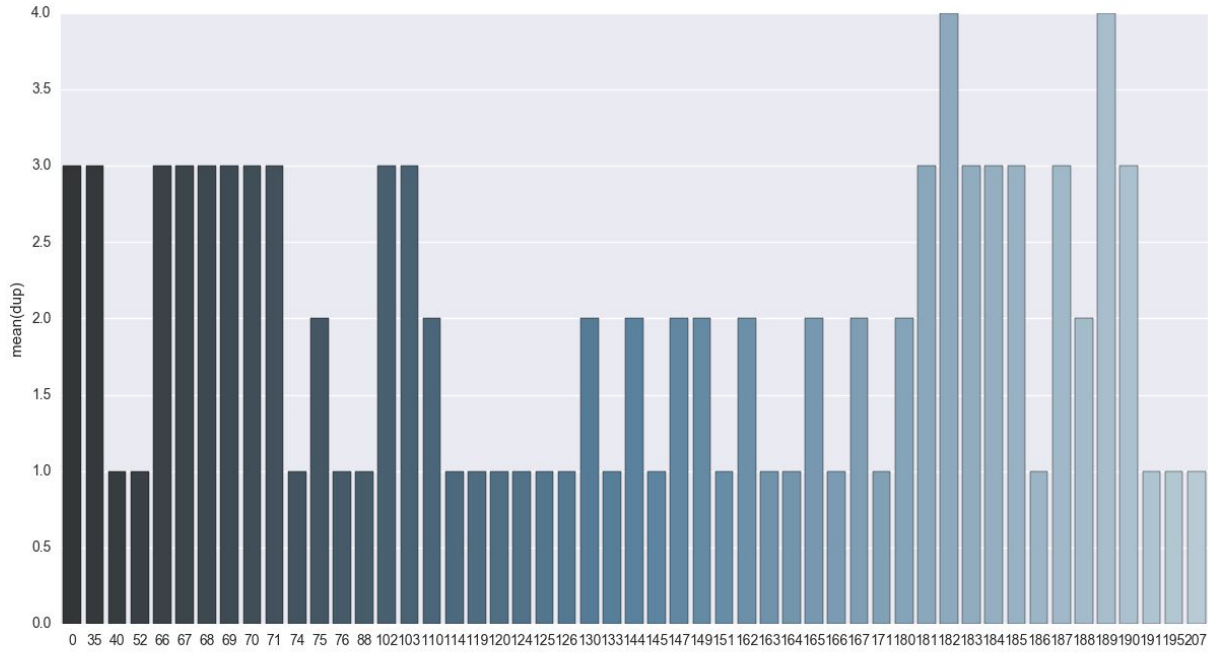
In order to find which set of parameters make the agent perform best, it is necessary to figure out what does 'best' mean. For this case, we will say that best means that the smart cab learns fast, arrives to destination in every single trip and that it doesn't take too long traveling to the destination. According to this, 10 different parameters were measured for each simulation, with distinct parameters:

- ***Percent to deadline:*** The proportion of time / deadline required to arrive to destination (for all 100 iterations). The lower, the better.
- ***Percent to deadline last 10:*** Same as above, but only for last 10 iterations.
- ***Penalties:*** Amount of times the agent is penalized (gets a negative reward) in all the 100 iterations.
- ***Penalties last 10:*** Same as above, but only for last 10 iterations.
- ***Scores:*** What were the total scores during all 100 iterations. The higher, the better.
- ***Scores last 10:*** Same as above, but for only last 10 iterations.
- ***Steps:*** Sum of total number of steps required to arrive to destination over all 100 iterations.
- ***Steps last 10:*** Same as above, but only for last 10 iterations.
- ***Successes:*** How many times did smart cab arrive to destination (from all 100 iterations). The higher, the better.
- ***Successes last 10:*** Same as above, but only for last 10 iterations.

The idea of using a "last 10" measure for each of the global ones is that it leads us to know how well our trained model performs. Additionally, for each one of these parameters, the top 10 parameter configurations was obtained and is shown below (each parameter configuration has a unique id shown on the left of the plot):



Some configurations appeared in the top 10 of more than one of the 10 measures taken:



As seen, configuration #182 and #189 seem to be well balanced configuration since they appear in the top 10 of 4 different measurements. To decide which one of these two configurations was more important, the value for each configuration attribute (except alpha, beta and gamma) was ranked in comparison to every other configuration. After this, the configuration that had the least sum of ranks was selected, which in this case was configuration #182.

id	Penalties	Penalties_10	% to deadline	% to dead._10
182	32.40	1.90	0.439258	0.399052
189	73.00	2.15	0.472685	0.391640

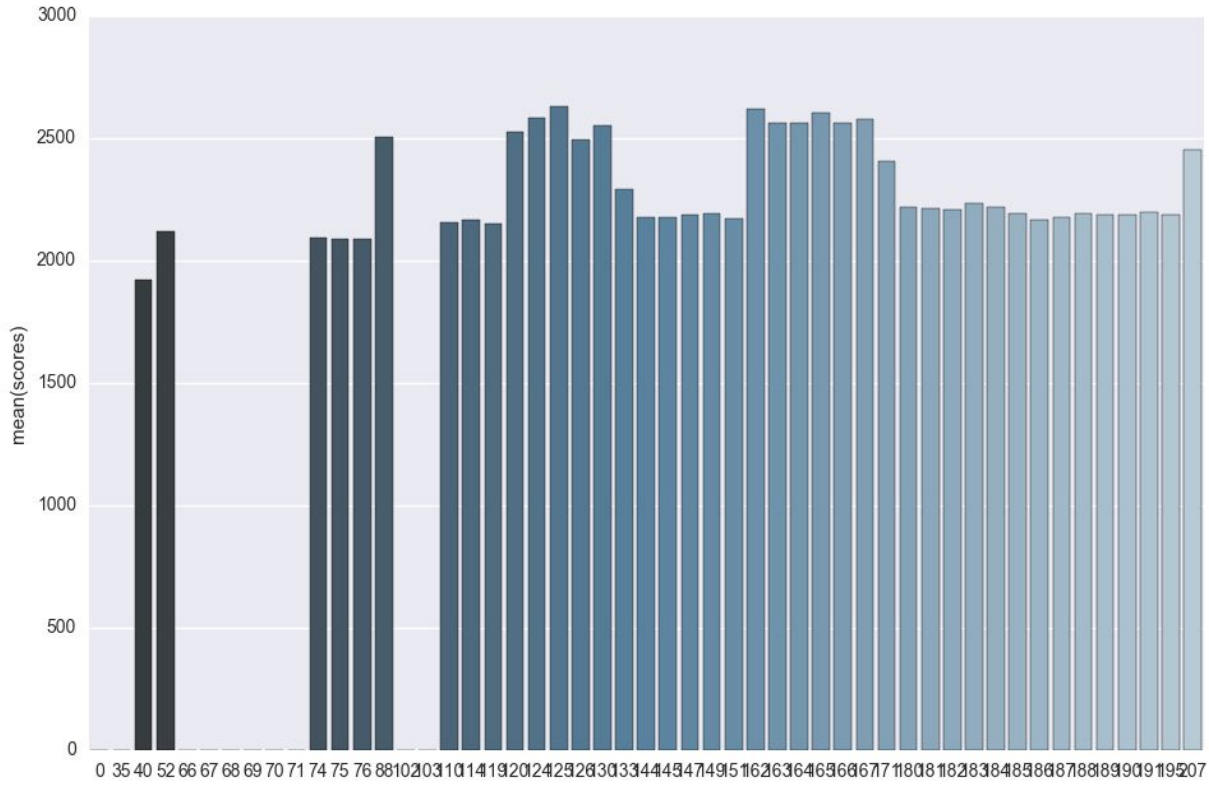
id	Steps	Steps_10	Successes	Successes_10
182	1309.40	223.050	98.60	10.00
189	1367.65	221.150	97.75	10.00

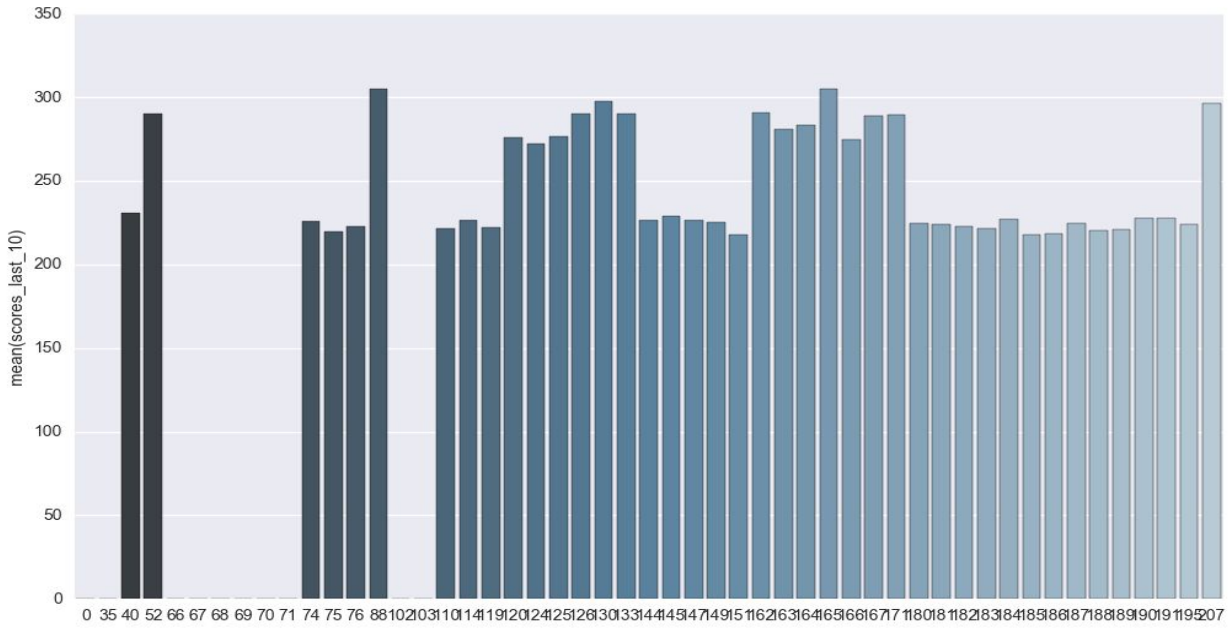
id	Rank (sum of)

182	31.5000
189	34.5000

According to the information given above, config #182 is much better than #189 in all measurements except one (*% to dead _last 10,* where #189 is better by less than 0.01). It is also much better than most configurations for every other measure. Additionally, configuration #182 was the one that ranked first in comparison with all other configurations.

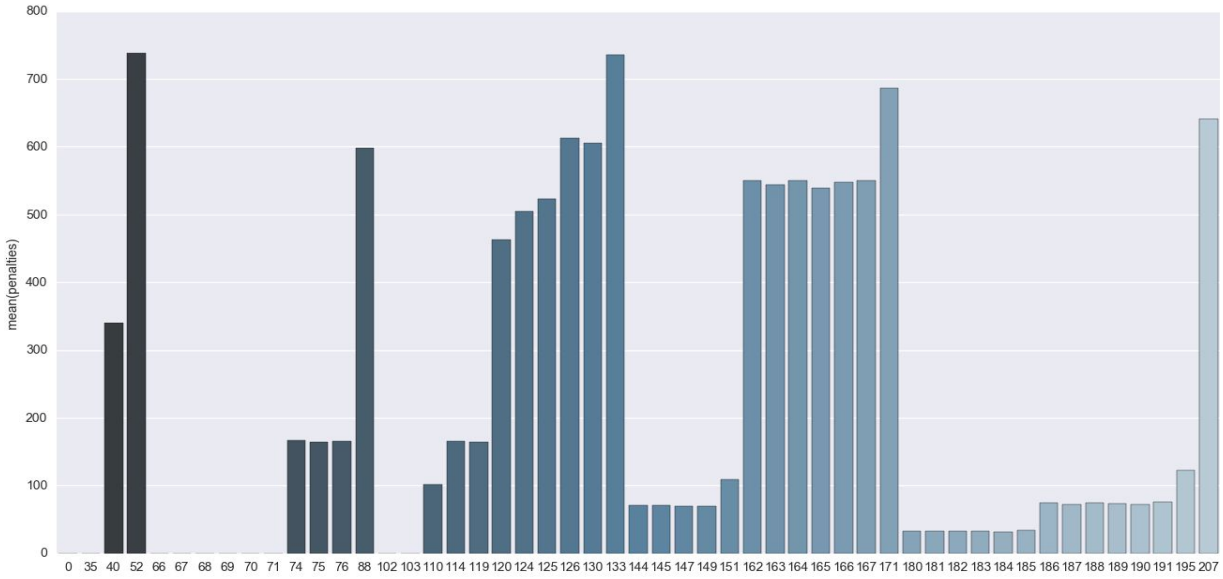
It is important to notice given that the scores for the configuration #182 are lower than many of the other configurations, both for all the total scores and for the last 10 iterations score.

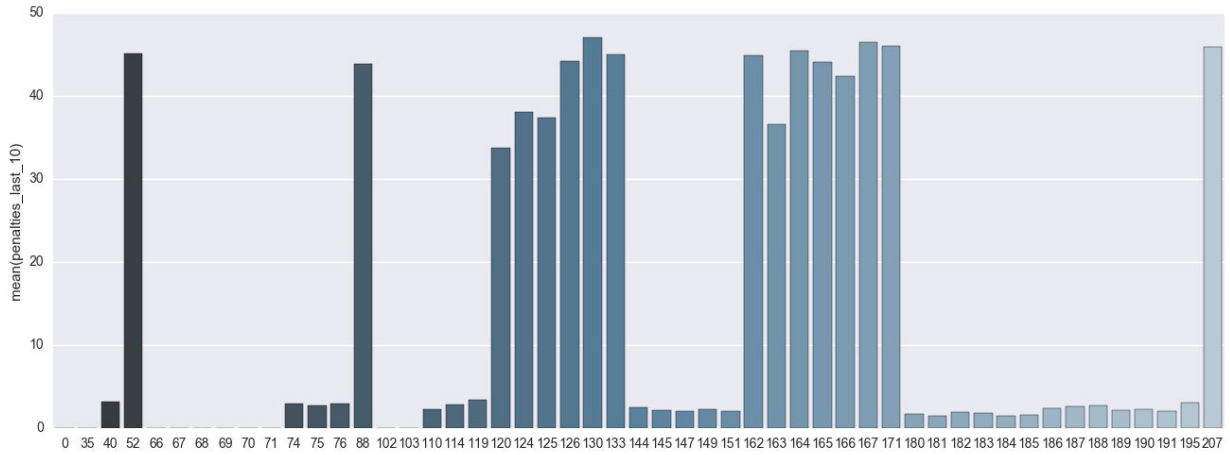


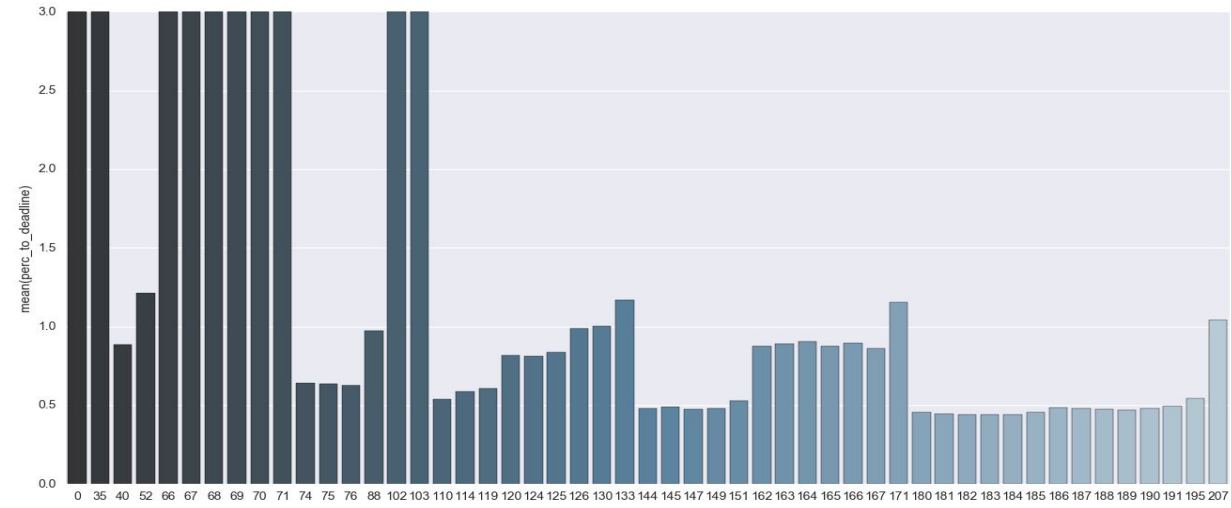


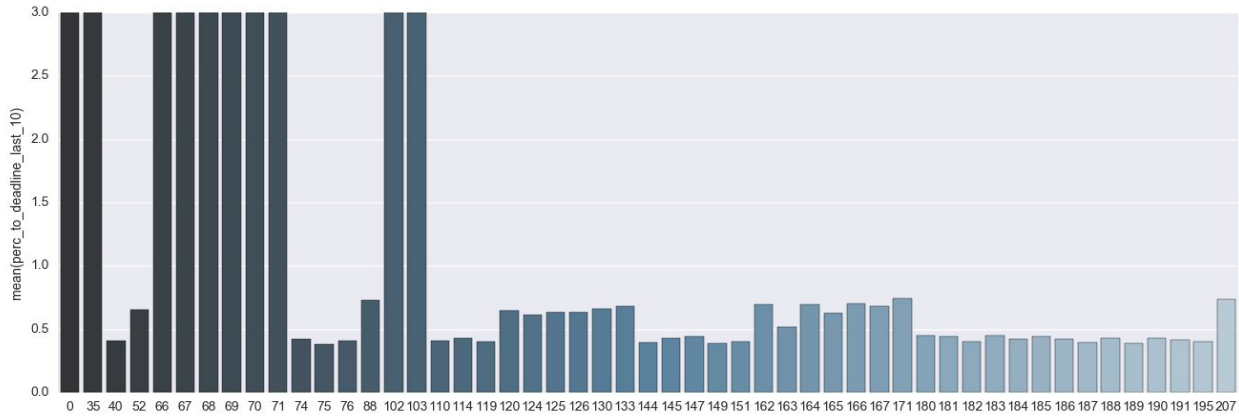
Nevertheless, what's going on here is that this configuration is taking the least average percent to deadline, making it earn less rewards than other configurations which might make mistakes. The cost (negative reward) for a mistake is -1 or -0.5, while the reward for moving correctly is 2. So as a smart car I might follow loops so that my positive rewards outbalance the negative ones, making me gain an overall higher score.

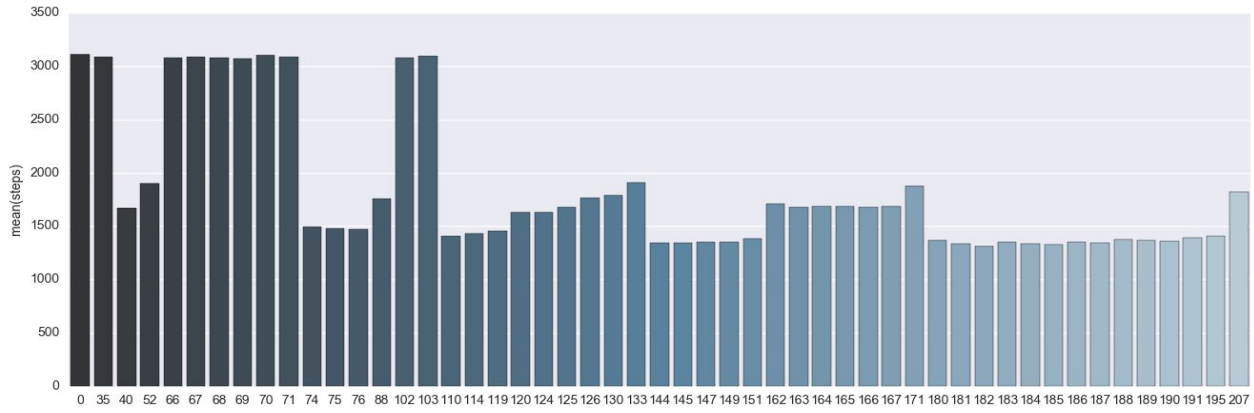
Below, you will find a graph that explains how all top 10s configurations performed for each of the 10 measurements taken. As an implication of what was explained previously, configuration #182 will appear to perform well in all of the graphs:

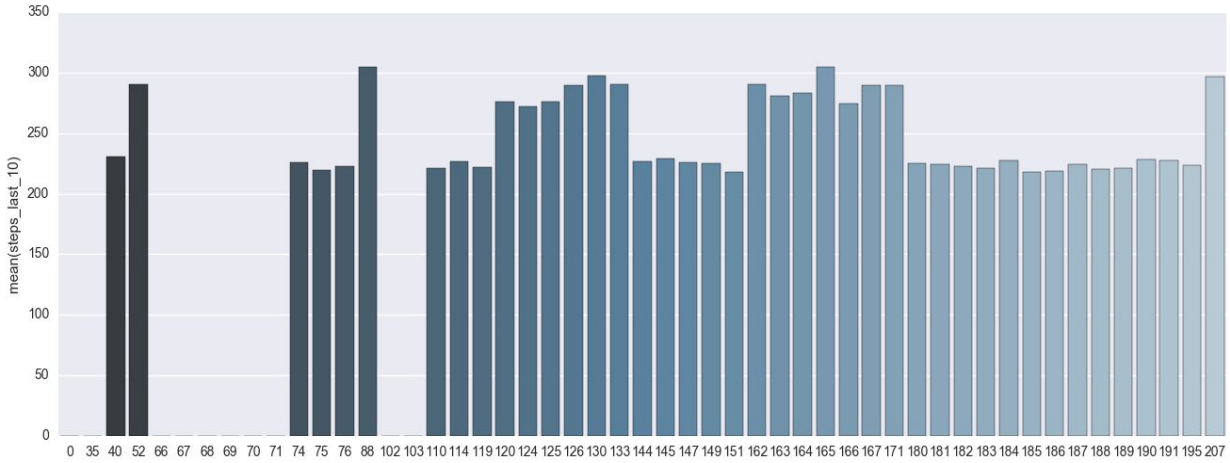


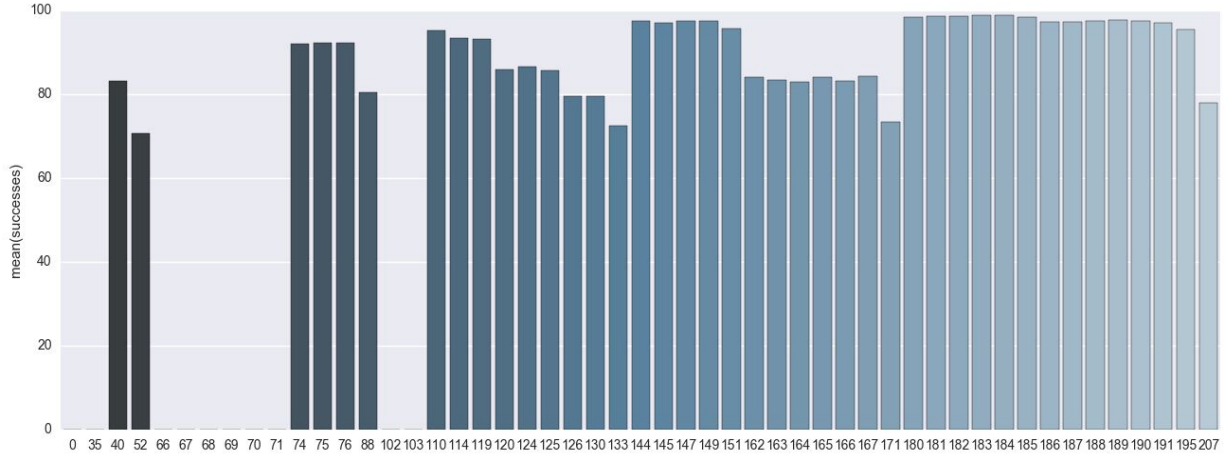

mean(penalties_last_10)
50
40
30
20
10
0
0 35 40 52 66 67 68 69 70 71 74 75 76 88 102 103 110 114 119 120 124 125 126 130 133 144 145 147 149 151 162 163 164 165 166 167 171 180 181 182 183 184 185 186 187 188 189 190 191 195 207

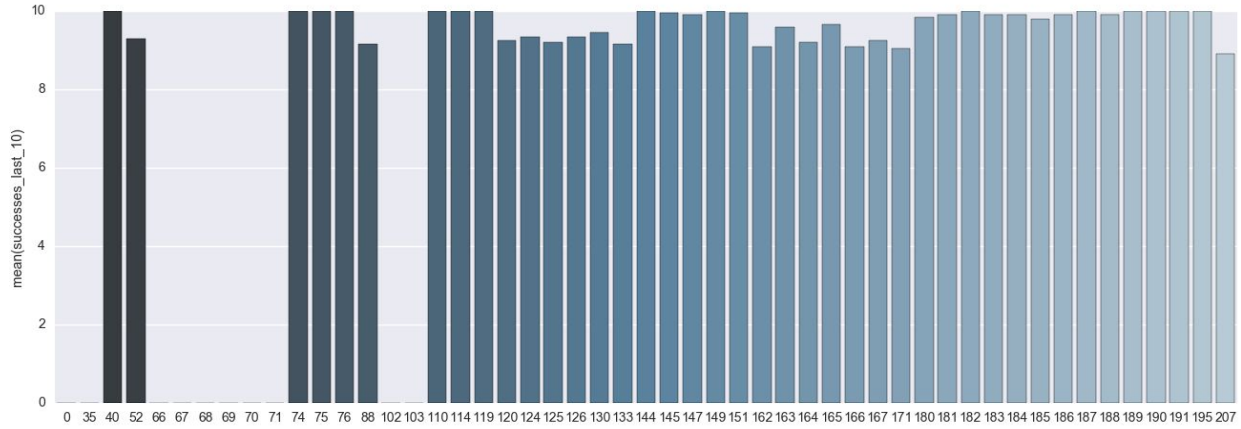

mean(perc_to_deadline)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0 35 40 52 66 67 68 69 70 71 74 75 76 88 102 103 110 114 119 120 124 125 126 130 133 144 145 147 149 151 162 163 164 165 166 167 171 180 181 182 183 184 185 186 187 188 189 190 191 195 207


mean(perc_to_deadline_last_10)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0 35 40 52 66 67 68 69 70 71 74 75 76 88 102 103 110 114 119 120 124 125 126 130 133 144 145 147 149 151 162 163 164 165 166 167 171 180 181 182 183 184 185 186 187 188 189 190 191 195 207


mean(steps)
3500
3000
2500
2000
1500
1000
500
0
0 35 40 52 66 67 68 69 70 71 74 75 76 88 102 103 110 114 119 120 124 125 126 130 133 144 145 147 149 151 162 163 164 165 166 167 171 180 181 182 183 184 185 186 187 188 189 190 191 195 207


mean(steps_last_10)
350
300
250
200
150
100
50
0
0 35 40 52 66 67 68 69 70 71 74 75 76 88 102 103 110 114 119 120 124 125 126 130 133 144 145 147 149 151 162 163 164 165 166 167 171 180 181 182 183 184 185 186 187 188 189 190 191 195 207


mean(successes)
100
80
60
40
20
0
0 35 40 52 66 67 68 69 70 71 74 75 76 88 102 103 110 114 119 120 124 125 126 130 133 144 145 147 149 151 162 163 164 165 166 167 171 180 181 182 183 184 185 186 187 188 189 190 191 195 207



In conclusion, the best configuration is the one that arrives the fastest, learns fast, performs correct/safe actions, and succeeds in most of the iterations. Learning the fastest means having the most successes, arriving the fastest means having the lowest amount of steps to arrive to destination and lowest percent to deadline, and performing safe actions is having the lowest amount of penalizations. For our scenario, configuration #180 with params: $\alpha = 1.0,\ \varepsilon = 0.4,\ \gamma = 0.0$ is the best configuration as it maximizes all these functions (shown in the graphs above).

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 optimal policy for this problem would be: the smart cab reaches destination always, it reaches in the least possible time, and it doesn't have any accidents or tries invalid moves (it does not incur any penalties).

Yes, the agent gets close to finding an optimal policy. For this case the optimal policy is reaching the destination in the least possible time, and is shown in the avg. percent to deadline graphs. It also reaches the point where all its iterations are successful (agent arrives to destination). Nevertheless, the trained car still incurs in penalties: for the average 223.050 steps that it took in total for the last 10 iterations, it incurred in average in 1.90 penalties. This means that there's a 0.85% chance of incurring in errors, which is really low. Nevertheless is not zero.