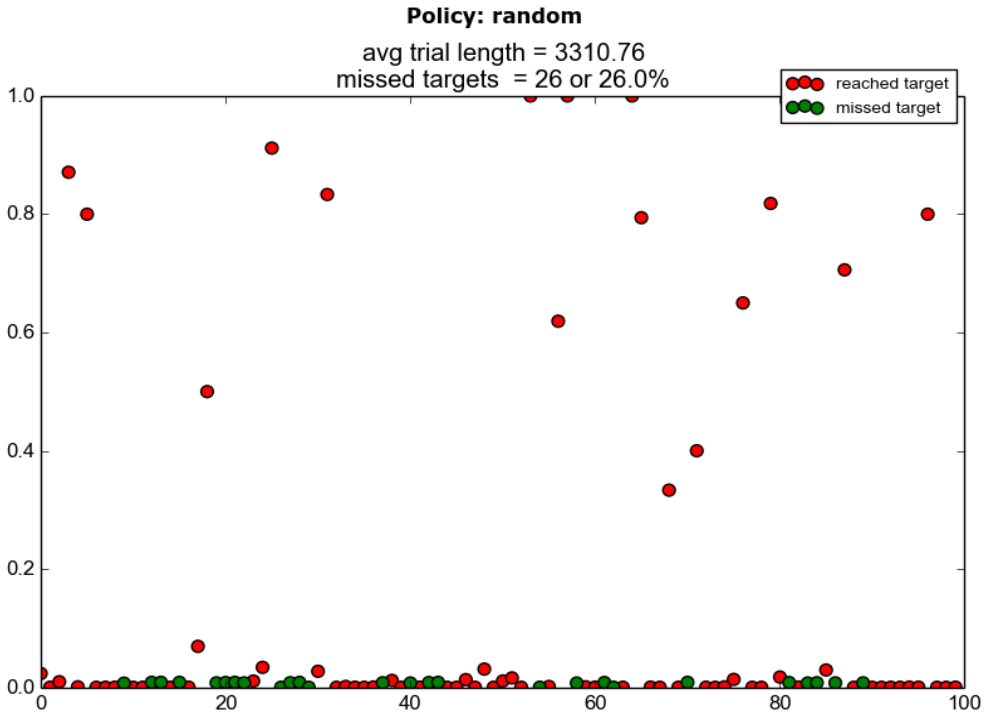
Project 4: SmartCab

To get a rough visualization of the Learning Agent's behavior I tallied the number of times its reward was negative (incorrect moves made by the agent) and divided that number by the total number of moves the agent took during a trial. Then I generated a scatter plot of this bad_move/(total_moves) ratio vs. the total number of trials, mainly, 100 of them. I tried 3 different policies for comparison to the implementation of Q-learning.

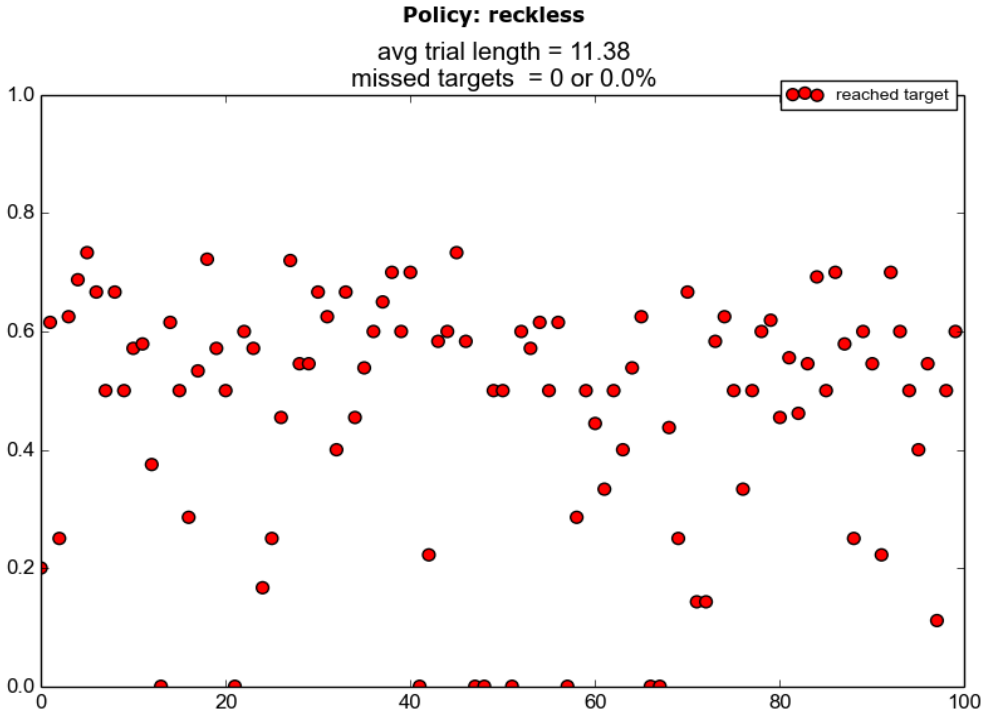
Random choice:

Here the choice of the next given action to take is exactly as that of the Dummy Agents that drive around the grid. At each step a random action is taken between 'forward', 'left', and 'right' excluding the None action, so that the agent always at least moves. The results for a given run are shown below. The green dots represent times the agent never reached its objective. From here it can be seen that the agent surprisingly reaches its objective randomly about 74% of the time. In addition, for only about 16% of the trials did it have a markedly high mistake ratio. However for the majority of those (~80%) the ratio was pretty high, above 0.6, meaning that it erred on more than half the moves the agent took during that trial. The biggest problem with this policy is the average time it takes to actually finish (if it does at all, i.e. more than the deadline = -100 absolute cutoff). Here we see that the average length of completion was 3310 moves.



“Reckless” choice:

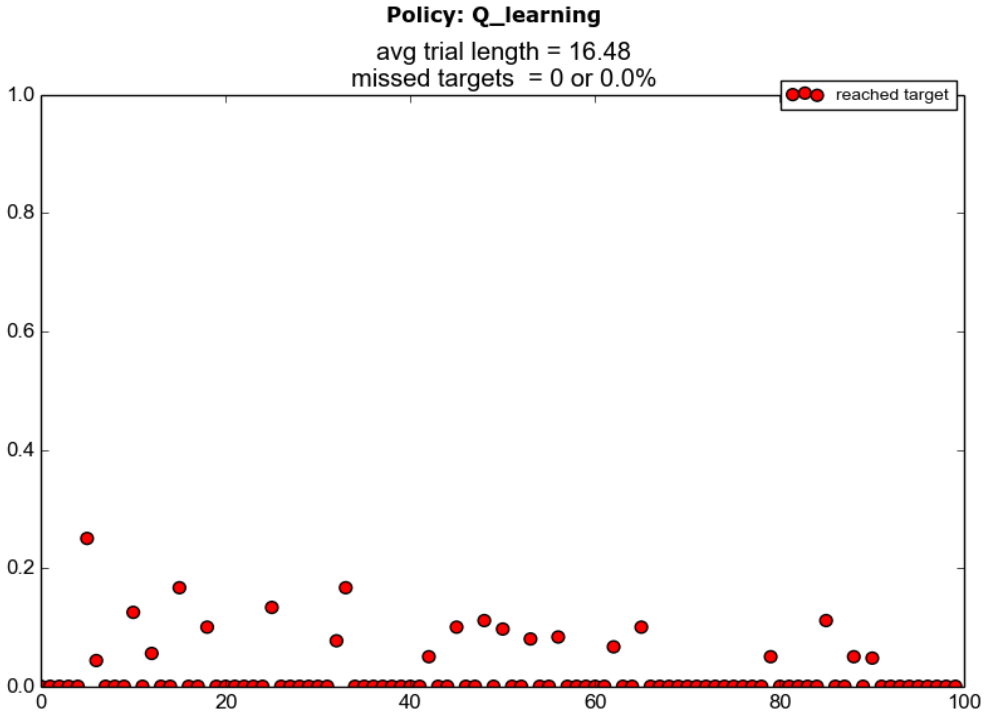
Here the choice for the next move is simply the next way-point as dictated by the planner. This was actually the first policy I implemented during the course of the project. It provides a lower bound on the amount of moves (or “time”, since with our implementation they are the same thing) the agent needs to take to reach its objective. Here the agent takes the most direct path, according to the planner, towards the objective. Ignoring in the process any negative reward accumulated by disregarding traffic rules. As can be seen in the figure all the dots are red, so the agent always reached the objective, but its error ratio is mostly evenly distributed in a band between 0.2 and 0.8. In almost no trial did it not commit some infraction, and on average slightly more than half of its moves were bad moves. Its average time to target was just 11 moves, a mere 0.33% of the time it took the Random policy.



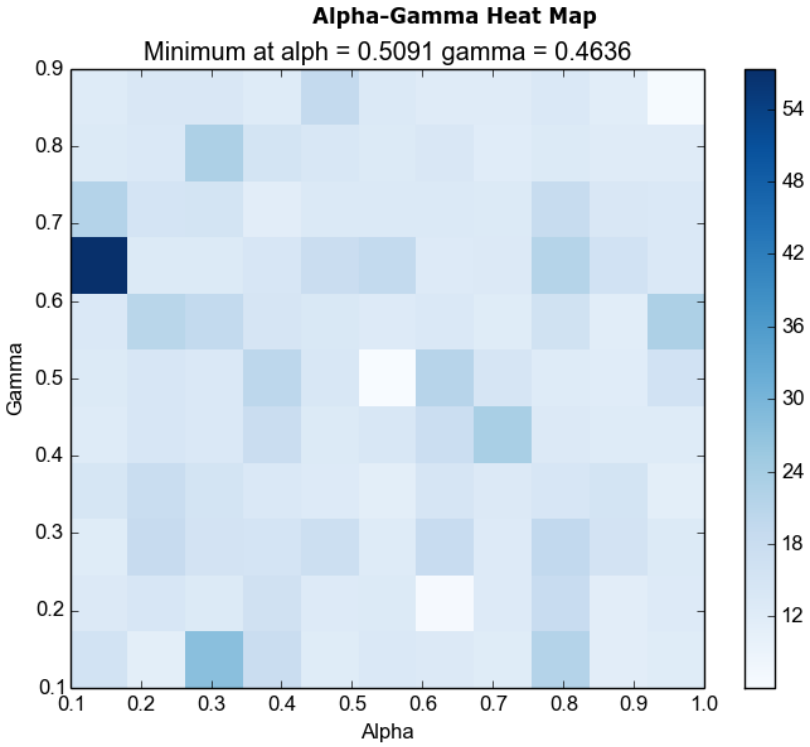
The figures above are for a single run of the program, not an average of multiple runs. Variation between multiple runs did not seem that significant between the Random and Reckless policies. Admittedly I did not follow through with a calculation of averages or other statistics to back up that observation more rigorously.

Q-learning:

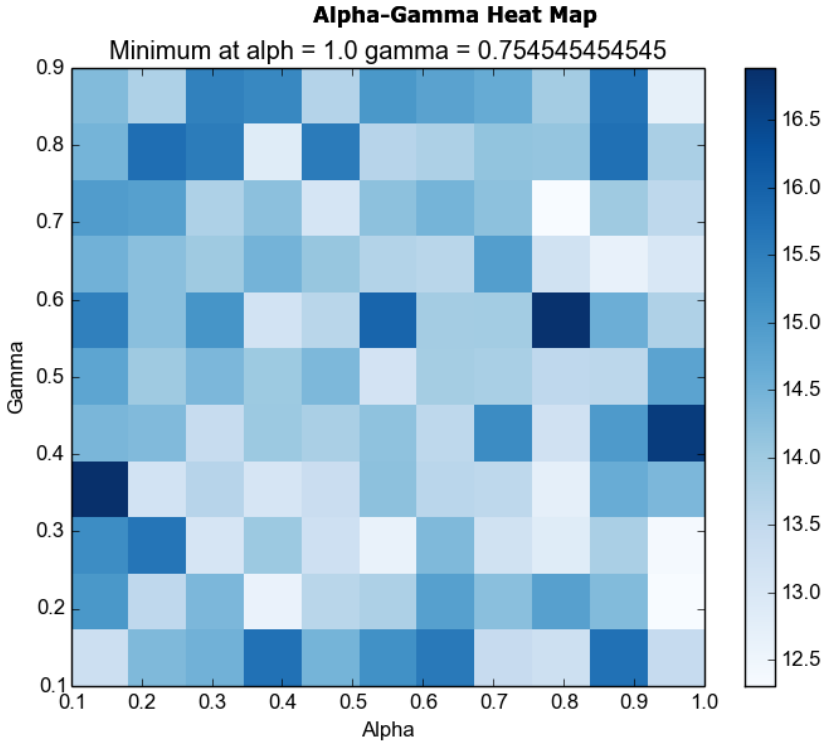
For the Q-learning implementation the state space that seems relevant consists of the inputs to the agent at each intersection; Traffic light color, and whether a car is approaching from the forward, left, or right directions, or not at all. However, it seems to me important to distinguish between the case where say, there is an oncoming car and the Learning Agent's way-point indicates a left turn on a green light (a potential collision) and the case where in the same situation the way-point indicates a right turn (where no collision would occur). Ideally the agent would accumulate negative weight for the former and positive for the latter. So the way-point is also included as a state parameter. In the my actual implementation of this, the Q function accounts for three directions the Learning Agent could encounter another agent (or not) at an intersection; oncoming, left, right. For each of these the other agent could, either not be there, turn 'left', turn 'right', or go 'forward'. So far a total of 4x4x4 = 64 states. For each of these the light could be either 'green' or 'red, making a total so far of 128. Now, since the way-point is included, that means that for each of all of those states so far the Learning Agent could be wanting to turn 'left', 'right', 'forward', or take no action. When I looked at the logic in planner.py (which sends the way-point), as I understood it, the way-point is only None when the Learning-Agent has reached its target. I decided to exclude the None action from Q's consideration, since the simulation would be at its end anyway. So the total number of states is 3*128 = 384. Q itself is bigger than that because for each of those states in the environment the Learning Agent will take one of 4 actions. So the Q function as an, (action , state) → value mapping, takes each of 4*384 = 1536 possible state action combinations to some value. Below is the plot of a typical run of Q-learning with learning rate, alpha = 0.5, and discount sum rate, gamma = 0.5.



I settled on those values after a somewhat disappointing search for some optimal combination of alpha and gamma. Essentially I tried 12 evenly spaced values between 0.1 and 1 for alpha, and 12 evenly spaced values between 0.1 and .9 for gamma (somewhat arbitrarily though gamma should be strictly less than 1 for this to be a contraction I think). For each alpha/gamma combination I ran multiple runs of 100 trials each. I then averaged the average time it took the Learning Agent to reach its target over the 100 trials. At first I did this over just 2 runs and the results seemed somewhat interesting, as can be seen in this heat map of alpha vs gamma.



I thought that *maybe* there were the beginnings of a sink in the center where the reported minimum occurred at (0.51, 0.46). As I increased the number of runs to 15 however, things began looking more like this;



Alas, increasing the number of runs further, always yielded similar looking graphs, with no real structure that I could discern. However, though I couldn't find some visual pattern that would justify the alpha/gamma pair that yielded minimum average times to target, this minimum very consistently occurred where alpha = 1. Which could be expected from the equation V = (1-a) **V + a** X seen in class. Here a = 1 would yield complete learning (from X) and complete disregard for the current state V, in a sense learning faster. Here are some more alpha/ gamma pairs for 15 runs as described above;

(1.0, 0.39039)
(1.0, 0.75454)
(1.0, 0.39039)

and over 70 runs;

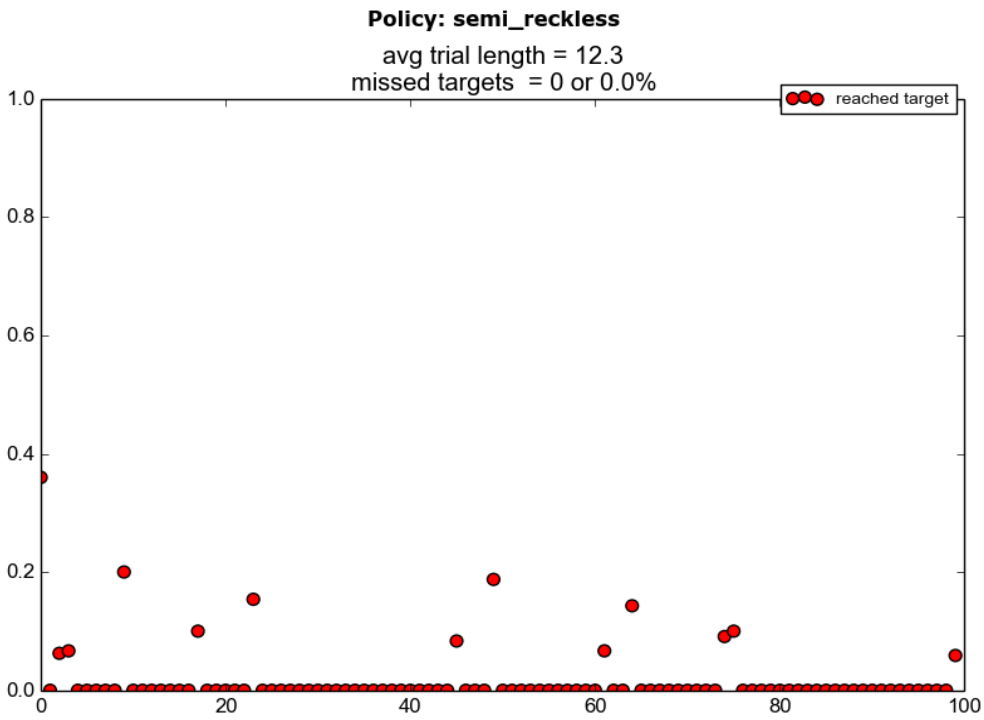
(1.0, 0.87272)

So one thing is that gamma did seem to vary wildly between a number *either* close to 0.4 xor close to 0.8. Don't know if that is significant as of yet. In the end I settled on alpha = 1 and the average of 0.4 and 0.8 for gamma.

Semi-Reckless:

Finally as a curiosity I implemented something of a hybrid policy between the Reckless policy and Q-learning as seen in class. In Q-learning at each step, the Learning Agent chooses the action for which Q assigns the maximum value given the current state. That is next_action = argmax Q(action, state). Where the argmax is taken over the 'action' which is one of {None, 'forward', 'left', 'right'}, call it max_action. In semi-Reckless, you first take the *value* of that action, i.e., Qv = Q (max_action, state) and the *value* of the way-point (the action Reckless proposes) Rv = Q(way-point, state). Then you choose the action that corresponds to the highest value between u * Rv and (1- u)* Qv, where u = 1 / (deadline + 1). So at the start of the trial, deadline is large, and regular Q-learning will be favored over being Reckless. But as the deadline counts down and approaches zero the action usually given by Q-learning will be abandoned and the Learning Agent will simply start taking the shortest path to the target (killing anything in the way). I did 30 runs of 100 trials each to compare Q-learning and semi-Reckless. The alpha/gamma parameters were left as alpha = 1 and gamma = 6, from the previous exploration. The average of the average times to target in 100 trials for both policies were as follows;
semi-Reckless: mean = 13.1286666667 std = 3.0070081846
Q-Learning: mean = 14.7636666667 std = 5.429347096

This seems kind of nice because you would expect semi-Reckless to go just a little faster as the time winds down. It also seems to be more consistent about this slight edge over Q-learning. That being said, I don't think I would want to be in *that* taxi. For completeness, I include below, the graph of bad move ratio vs moves in trial for semi-Reckless. Though over multiple runs this graph and Q-learning's seem to behave very similarly.


Policy: semi_reckless
avg trial length = 12.3
missed targets = 0 or 0.0%
reached target
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100