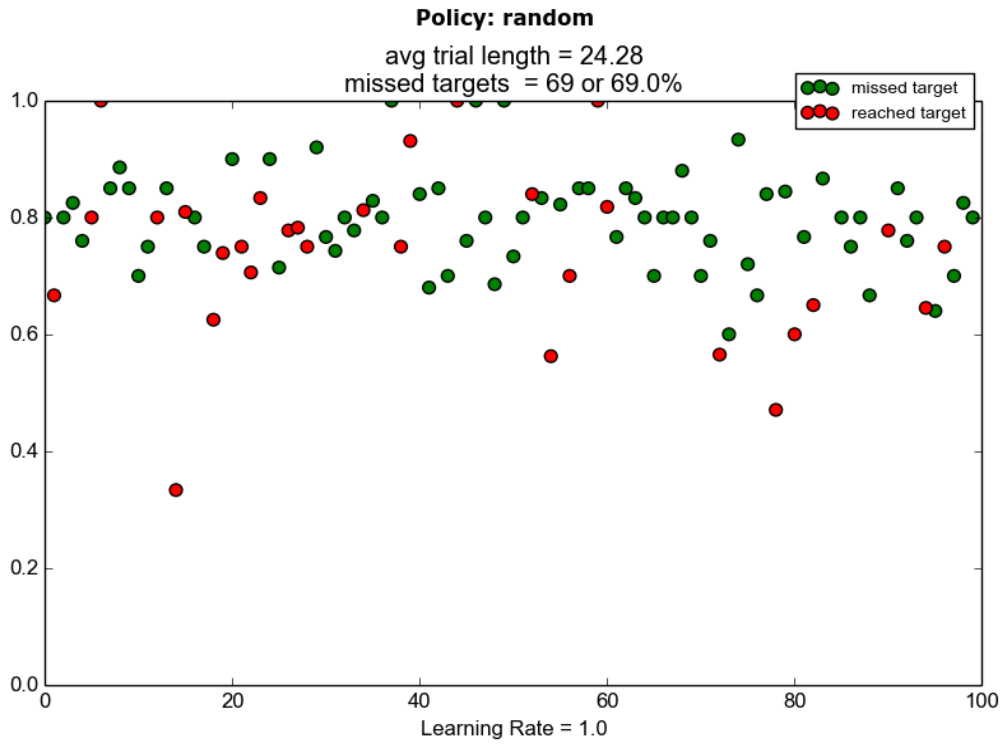
Project 4: SmartCab

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 bad_moves/(total_trial_moves) ratio on the y-axis vs. the trial number on the x-axis, that is, x runs from trial 0 to trial 99. I tried 3 different policies for comparison to the implementation of Q-learning.

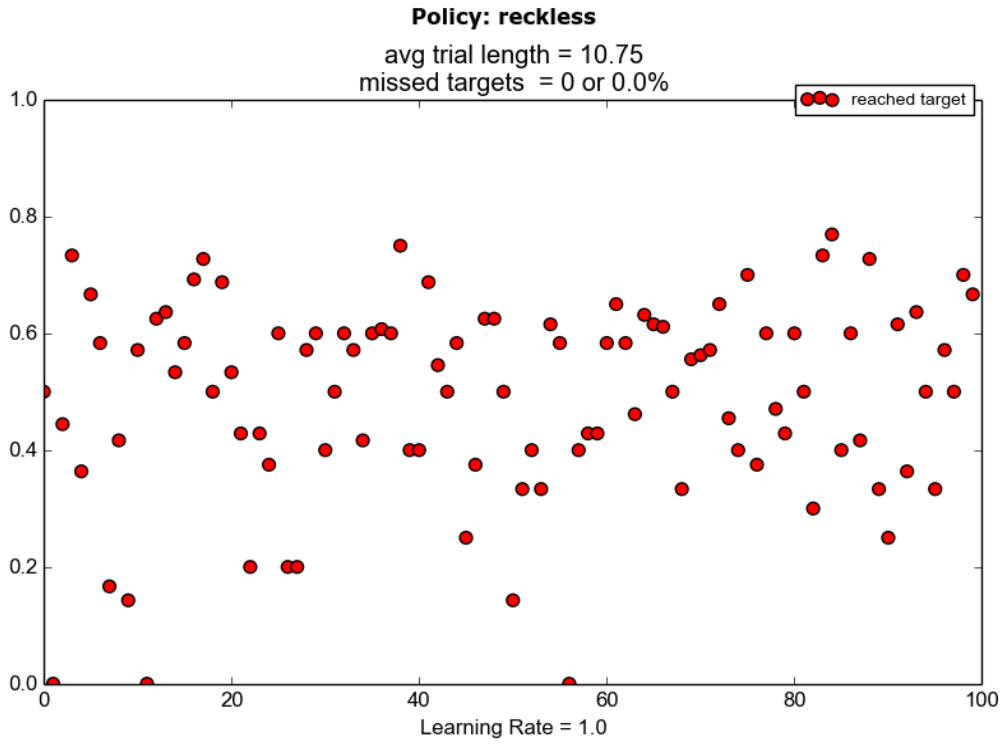
Random choice:

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', 'right', or None. After correcting for a bug in my tally method, 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 only reaches its objective randomly only 31% of the time. Its error rate is very high, with around 95% of all trials having over 60% of their moves being incorrect ones. Additionally as can be expected there is no improvement in performance from trial 0 to trial 100, with the bad move ratio pretty uniformly being about 80% on average. The average amount of time (or moves) each trial took was 24.28, which doesn't yet seem unreasonable (it will in comparison to the following policies), if it weren't for the fact that 69% of those trials ended with the agent missing the target entirely.



“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. 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 10.75 moves, less than half the time it took the Random policy. Again as it would be expected, like the random policy reckless shows no improvement or change in its behavior as it progresses from trial 0 to trial 99.



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.

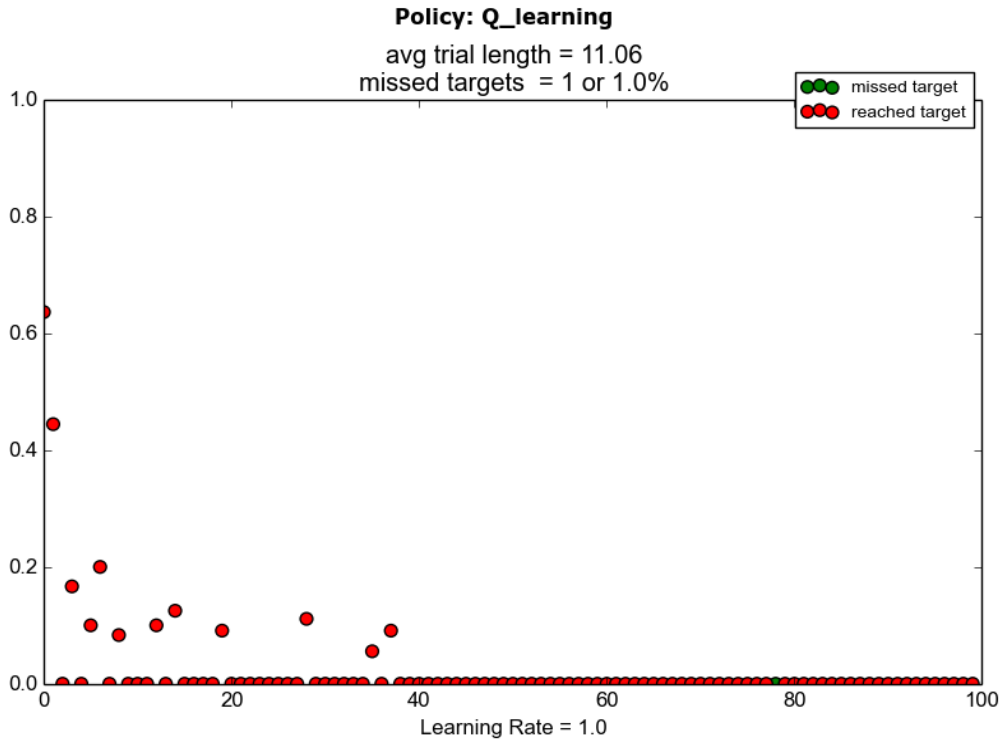
Q-learning:

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. Originally, in my actual implementation of this, the Q function accounted for all three directions the Learning Agent could encounter another agent (or not) at an intersection; oncoming, left, right. This led to a fairly large state space. 4x4x4 = 64 states given by 4 possible actions from 3 directions, times 2 possibilities for the traffic light, times 3 possible way-points (“None” way-point excluded) for a total of 384 states. However the state space can be more properly tailored for the learning problem at hand and reduced drastically. We are trying to get the agent to avoid two basic types of infractions;

1) Making a left turn on a green light while there is “simultaneous” oncoming traffic moving forward towards the agent, or traffic making a right turn into the path of the agent attempting the left turn.

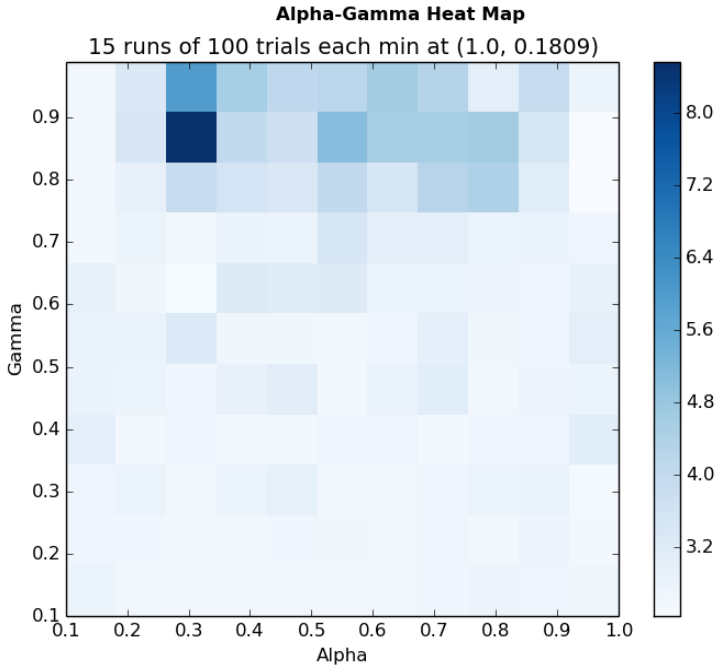
2) Making a right turn on a red light while there is traffic approaching from the left of the agent.

This means that at a green light the LearningAgent only needs to be aware of oncoming traffic information (4 possible states, None, “left”, “right”, “forward”) and the way-point (3 states, “forward”, “left”, or “right”) for a total of 12 states. Similarly on a red light, the LearningAgent just needs to know what is occurring on its left had side (again 4 possible states), and for each of those possibilities there are 3 for the way-point. So another 12 for a total of 24 states in the state space. I exclude the None way-point from consideration because it is only None when the LearningAgent reaches its target, in which case the simulation would be over. 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*24 = 96 possible action-states to some value. As opposed to 1536 possible action-states for the original scheme. The deadline might seem like it could be part of the state. However if our objective is learning traffic rules, these are independent of time. However, I consider a play on these notions later with the semi-Reckless policy. Below is a plot of a run of Q-learning with learning rate, alpha = 1.0, and discount sum rate, gamma = 0.2.

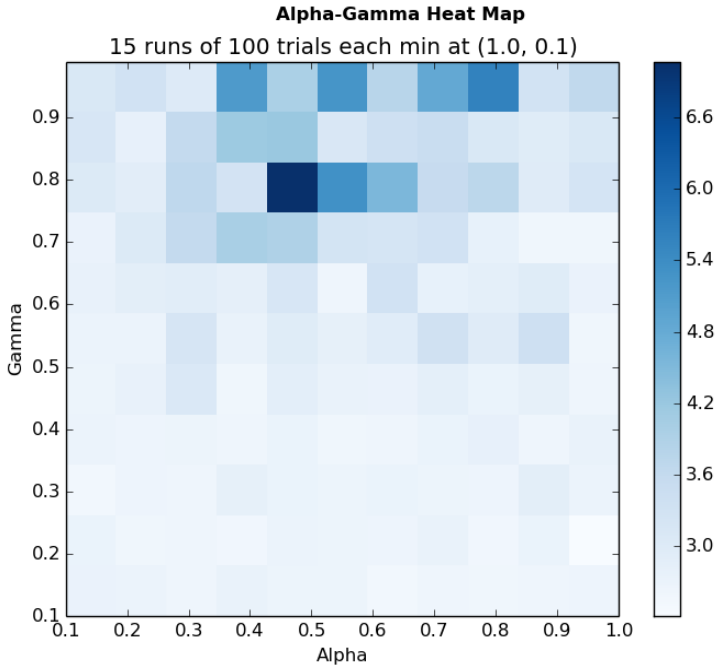


Several things can be observed in relation to the Reckless and Random policies. First, most notably, is the LearningAgent's change in behavior as the trials run from 0 to 99. The first trial has a highish error to move ratio of ~0.65, but the ratio sharply falls as more trials are run to virtually zero by the 40th trial. As trials are run, erroneous moves leave their mark as negative reward on the Q function. Since the LearningAgent decides its next action as the one having the highest value given by Q at the present state, bad moves are increasingly less likely to be picked. That is, the Q-value of a bad move keeps getting reduced by repeated addition of negative rewards. The second thing to notice is that in this run the LearningAgent missed one target (though in that trial it made no mistakes) which shows how its behavior lies between that of Random and Reckless. Like random it misses targets (although usually only between 1 and 3) but like Reckless its time to target is much smaller, in this run of 100 trials an average of 11.06. Reckless will mostly be some moves quicker than Q-learning since it will never deviate from its shortest path to target as given by the planner. On the other hand Q-learning might make moves that increase its distance to target based on the fact that they have a higher value in Q. Admittedly not all runs generated such neatly behaving graphs as the one above, with sometimes more erratic behavior towards the middle and end, but the trend to sharply decrease bad_move ratio ratio to close to zero remains.
I arrived at the values for the learning rate alpha and discount gamma after a somewhat obsessive and not completely

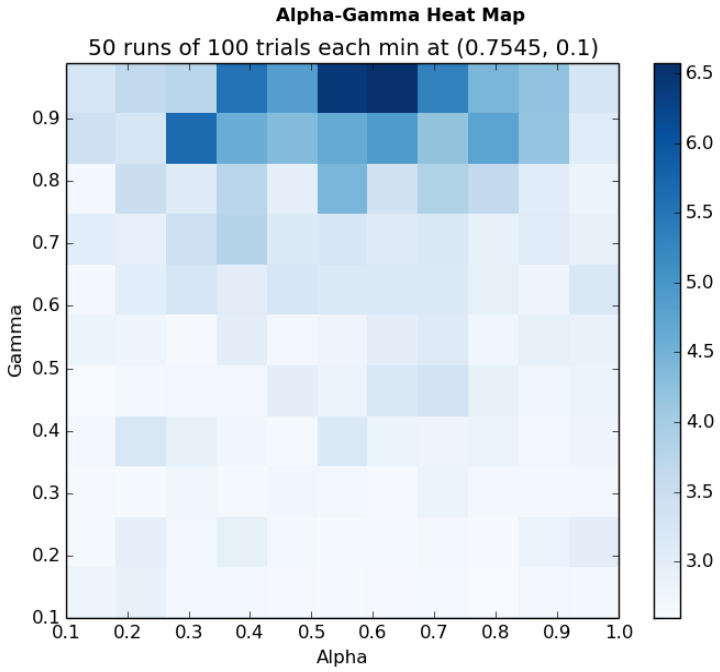
satisfactory exhaustive search. Essentially I tried 12 evenly spaced values between 0.1 and 1 for alpha, and 12 evenly spaced values between 0.1 and .99 for gamma (somewhat arbitrarily though gamma should be strictly less than 1 for this to be a contraction I think). For each alpha/gamma combination 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. I then generated a 12 by 12 heat maps with the average of average times dictating the heat intensity value. This in the hope of finding good regions where alpha/gamma pairs might exist. The heat values I encountered had only modest variation between themselves leaving the maps whitewashed, I therefore used their log values in the final maps to increase contrast. Below are two samples of 15 runs of 100 trials.



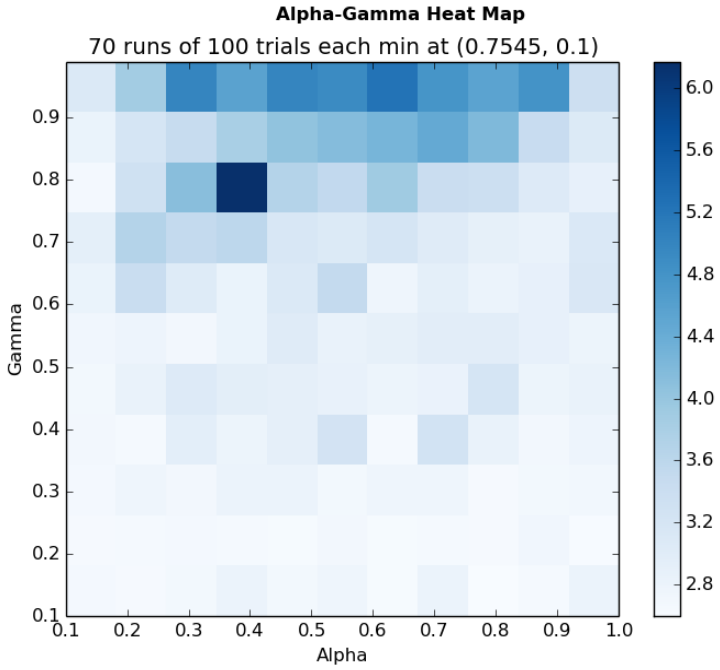
##

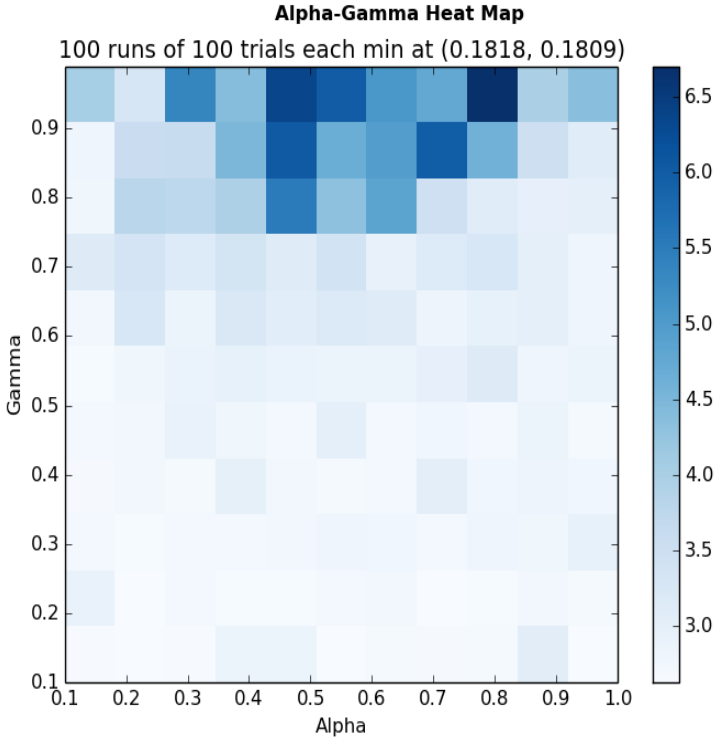


On the previous submission of this project these maps basically looked like random noise, but my implementation of the Q-learning equation was incorrect on several accounts. Here are maps in increasing number of runs.



It seems to me that *maybe* there is a main “heat source” centered at around alpha/gamma (0.5-0.6, 0.99) that radiates symmetrically outward.





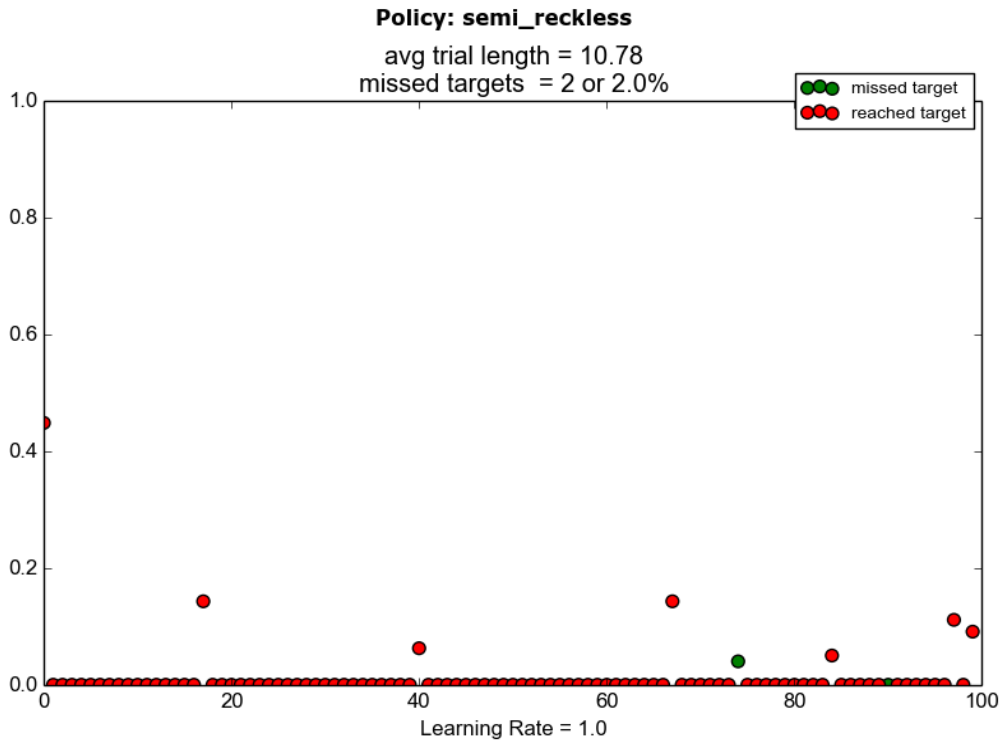
Though its shape fluctuates somewhat from run to run it indicates a region where you *don't* want to pick you alpha/gamma. In regards to the alpha/gamma pair that resulted in minimum heat, it almost always occurred with an alpha of 1 or close to 1, and a gamma almost invariably below 0.2. alpha = 1, 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. But sometimes the alpha/gamma minimum looked more like (0.1, 0.1). From the maps, if indeed heat spreads radially from somewhere close to (0.5, 0.99), then the corners (0.1, 0.1) and (1.0, 0.1) would be the farthest from it. So minimums could come from either. In the end based on mostly on these observations I chose alpha/gamma = (1.0, 0.2), where gamma is more of an average of observed values.

Semi-Reckless:

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. 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 = 0.2 from the previous exploration. The average of the average times to target in 100 trials for both policies were as follows;

Semi-Reckless: mean = 11.6476666667 std = 0.578409793217
Q-Learning: mean = 11.9746666667 std = 0.712866202188

This seems kind of nice because you would expect semi-Reckless to go just a little faster as the time winds down. The above standard deviations also imply that semi-Reckless is slightly more consistent on its tiny edge over Q-learning. 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.



Conclusion:

Finally this leaves the question of what an ideal, “perfect” policy would look like. It should lead the agent to not make any bad_moves at all, but match Reckless in speed. I don't see how it's possible to *perfectly* reconcile these two requirements. At the end of the day if the final target is across a red light, then matching the way-point's demand *and* not incur a penalty is impossible. It should be possible to achieve minimum penalty however, by programming the agent to explicitly to resolve the two problem scenarios described in the Q-learning section above. In the case of a green light, with the agent facing oncoming traffic moving forward or making a right, and its way-point indicating a left turn, it should choose action None (in other words yield to traffic). In case of a right turn on a red light with traffic moving forward from the left, the agent should choose action None (again yielding the right of way). In addition the agent should always choose None action at a red light when the way-point is forward. In this manner the agent will never deviate from the shortest path as dictated by the planner (matching Reckless's speed) and incur the lightest possible penalty (-0.5 in the code I think). Both Q-learning and semi-Reckless strive to approximate this ideal as they evolve, since in both policies the two action-states mentioned above will incur less penalty over time than not yielding to traffic.