In your report, mention what you see in the agent’s behavior. Does it eventually make it to the target location?

I got the simple version of the program running with the primary agent making uniformly random choices about what to do during each time step (“[None, 'forward', 'left', 'right']”). When I ran the program this way, the primary agent is basically wandering around the map randomly. There are times when it is waiting at a light for several time steps and changes its mind about which directions to go since its random direction is getting updated every time step. It is also getting a lot of penalties from the reward function, because it is randomly breaking traffic laws by trying to turn/go straight when it is not allowed etc., and accidentally crashing into other cars (agents). Despite all of this, if you sit and watch long enough, it does eventually make it to the target location.

Justify why you picked these set of states, and how they model the agent and its environment.

I think the smart cab will need to know the following information in order to “learn” how to get to the target location while maximizing its reward in this system/game:

* Recommended next\_waypoint
  + This is a good default decision for the direction to go in the next time step. This would be an ideal decision if there weren’t any other cars, or traffic lights. The other variables below might cause the primary agent to deviate from this recommended action. `self.planner.next\_waypoint()`
* Time steps left until the deadline
  + This gives information about how many time steps are left before the agent might incur a penalty for not arriving in time. This might change incentives and therefore decisions for the agent when the time steps until the deadline is getting closer to zero. `self.env.get\_deadline(self)`
* Light status at the intersection
  + This will help the agent “learn” if it will be legal to take an action during the current time step. Essentially letting the agent know if it will incur a penalty for taking an action that would break the law. This can be captured using ` self.env.sense(self)`.
* The potential actions for other cars at the ‘oncoming’, ‘left’, and ‘right’ positions at the current intersection
  + Again, this information will help the agent “learn” what traffic situations with other vehicles will cause it to incur a penalty. This can be captured using ` self.env.sense(self)`.
* The delta distances between the current and target location in the X and Y directions
  + This is very similar to the L1 distance, and could actually be used to compute the L1 distance. This “learned” L1 distance can be used by the agent to determine how far it has to go until the target destination and compare it to how many time steps are left. This might change the urgency of getting to the target destination and therefore the agents actions.
  + This also has the advantage of allowing the agent to make better guesses about what directions to go if the recommended next\_waypoint action will cause penalties in the current time step. For example, let’s assume I’m sitting at an intersection and the light is red and the recommended action is to go straight. If I know that my target location is one block ahead and one block to the right from my current position/heading, it might make more sense to turn right now and left at the next intersection, instead of waiting for the light to turn green to go straight, and then turn right.
* Heading for primary agent
  + In order for the logic in the 2nd reason for delta X and Y distances above to work, you would need to know the current heading of the vehicle. So, this will get added to allow for this “learning” to take place.