Project Report

Implement a Basic Driving Agent

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?

Having the model only perform random actions results in 'seemingly' random results. Sometimes the virtual vehicle will get to its ultimate destination on time but very rarely. Having the time constraint disabled allows for the vehicle to always make it to the destination, however in a large majority of cases the time it takes to do so is well beyond the allotted time. Another observation is the next waypoint input. Interestingly having this data can be a great value to have because give the model a clue as to which action can be prioritized to try. Of course when all actions currently are random, the model will ignore this input and will try all actions as randomly equal. The feedback as rewards is something else to be noticed. When the model tries to perform an action and the action is illegal or incorrect, the feedback reward will be negative. When the car performs a correct action or an action that will result in the car getting to its next waypoint safely, the feedback reward will be positive. This is something we can use to have the model itself train the rules of the road and not really have to tell it explicitly the rules. Given enough training and gathering this kind of feedback, the model can just guide itself based off previous feedback.

Inform the Driving Agent

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?

We'll have to take almost all input into account because the rules might change depending on the state of mostly all inputs. This includes the state of the light, which of course will obviously change what the vehicle will be allowed to do but also the state of other cars around the vehicle and oncoming traffic. So just with these we have Oncoming, Right, Left and Light as building blocks of our state. We will also have to take into account which way we want to go because that will a driving factor into what action we will want to take so we will also add next_waypoint.

So our state will be a compilation of other input states of Light, Oncoming, Left, Right and the next waypoint (or in other words a preferred action). Decided to exclude the deadline decrementer because which will result in way too many states as only one state can be visited per run.

Implement a Q-Learning Driving Agent

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?

The agent reaches the destination almost always now in the latter half of runs in the deadline time compared to almost never on time with random actions. I also see that there is a lot less negative reward values being achieved after the agent has had a chance to initially visit (or explore) some of the state/action executions around the map.

Improve the Q-Learning Driving Agent

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?

Below is the value combinations I have tried and the resulting percent of trials that have ran to completion within the deadline. The combination of values that performed the best was
epsilon = .1 alpha = .1 gamma = .8
also of note is that increasing epsilon to encourage more exploration did train the the agent on states not yet establish, however it did so in vain because there was a deadline and by the having more random actions concluded in more missed deadlines even though the Q table was more complete. It may be better off in keeping the epsilon at a lower value and only fill on those Q values more

conservatively. Another approach probably worth investigating is to have epsilon start out high and decrease it over the duration of the run so encourage exploration at the start and rely on known rewards towards the end.
Having alpha = .1 seemed to also perform better. This may be because this will take into account the current rewards as a heavier weight than the next states rewards which makes sense because in this type of model learning the next state isnt always going to be the same and we would only care about the next waypoint. Just because the next states rewards would be high on this iteration, doesnt mean its going to be the same for the next iteration of the exaxt same state. It would imply that we mostly care about the immediate rewards vs future rewards. Especially in a model so dynamic such as training an agent with other 'not so predictable' agents in the same world.

epslion	alpha	gamma	reached out of 100
0.1	0.1	0.8	98
0.1	0.1	0.5	95
0.1	0.1	0.2	82
0.1	0.25	0.8	87
0.1	0.25	0.5	64
0.1	0.25	0.2	35
0.1	0.5	0.8	91
0.1	0.5	0.5	75
0.1	0.5	0.2	64
0.25	0.1	0.8	92
0.25	0.1	0.5	88
0.25	0.1	0.2	87
0.25	0.25	0.8	86
0.25	0.25	0.5	81
0.25	0.25	0.2	72
0.25	0.5	0.8	88
0.25	0.5	0.5	84
0.25	0.5	0.2	53
0.5	0.1	0.8	64
0.5	0.1	0.5	56
0.5	0.1	0.2	71
0.5	0.25	0.8	67
0.5	0.25	0.5	59
0.5	0.25	0.2	72
0.5	0.5	0.8	65
0.5	0.5	0.5	67
0.5	0.5	0.2	55

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?

Here is the final run of the 100 trials for the above values for epslion, alpha and gamma

Environment.reset(): Trial set up with start = (8, 5), destination = (2, 5), deadline = 30 RoutePlanner.route_to(): destination = (2, 5)
LearningAgent.update():next_waypoint=right, deadline = 30, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = 2.0
LearningAgent.update():next_waypoint=forward, deadline = 29, inputs = {'light': 'red', 'oncoming': None, 'right': None,'left': None}, action = None, reward = 0.0
LearningAgent.update():next_waypoint=forward, deadline = 28, inputs = {'light': 'red', 'oncoming': None, 'right': None,'left': None}, action = None, reward = 0.0
LearningAgent.update():next_waypoint=forward, deadline = 27, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update():next_waypoint=forward, deadline = 26, inputs = {'light': 'red', 'oncoming': None, 'right': None,'left': None}, action = None, reward = 0.0
LearningAgent.update():next_waypoint=forward, deadline = 25, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update():next_waypoint=forward, deadline = 24, inputs = {'light': 'red', 'oncoming': None, 'right': None,'left': None}, action = None, reward = 0.0
LearningAgent.update():next_waypoint=forward, deadline = 23, inputs = {'light': 'red', 'oncoming': None, 'right': None,'left': None}, action = None, reward = 0.0
Going Random
LearningAgent.update():next_waypoint=forward, deadline = 22, inputs = {'light': 'red', 'oncoming': None, 'right': None,'left': None}, action = None, reward = 0.0
LearningAgent.update():next_waypoint=forward, deadline = 21, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = 2.0
LearningAgent.update():next_waypoint=forward, deadline = 20, inputs = {'light': 'green', 'oncoming': None, 'right': 'forward', 'left': None}, action = forward, reward = 2.0
Environment.act(): Primary agent has reached destination!

Looks like our agent did not incur any negative rewards (yay!), however we need to keep in mind that there was one random action which *could* have resulted in a negative reward. What we could have done to completel eliminate that random is to decay epsilon down to 0 as a function of the decreasing deadline. However if we do that we risk getting into a state we cannot get out of. (although in this model it doesnt look like those exist)

With the absent of the epsilon-generated random, it looks like we do have an optimal policy. The agent safely waits at the red light if it needs to and in the cases that it needs to make a right at a red light safely, it does. When there is a green light, the agent proceeds safely.

An optimal policy for this problem is very peculiar because we would want, at all costs, to not make a catastrphic move. Even if that means waiting at a stoplight and I would think in my own sense that missing a deadline would probably be more favorable than making a deadly illegal move. In the context of this model, this would translate to having absolutely zero negative rewards. Having the agent trained to make a action decision based on the positive rewards in this context certainly achieves that. Also there doesnt seems to be any *more* rewards in getting to the destination faster than the deadline. Getting to the destination at the deadline has the same outcome as getting to the destination and halfway of the deadline time. So I would think this means that there is no incentive to 'speed' or make illegal move just to gain a step in time because maybe just waiting and taking no action at all (or None), if needed, would probably result in a better outcome in the long run than risking making a catastrphic move without motive.