***QUESTION 1:*** *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?*

Answer: When we set the agent’s behavior as a random action b/w none, left, forward and right, we observe the following:

* Agent takes one of the random actions. If the move was not okay (violated traffic rules or took unsafe actions relative to the other dummy agents) then agent gets penalized with a reward = -1
* If the move was okay but the agent action was “none” – do nothing then reward = 0
* If the agent makes an okay move and action is not none, then the cab will move. If it moves in the direction same as the next\_waypoint which is displayed on the pygame console, meaning it moves in the direction of destination then the reward is 2.0. If it moves in any other direction the reward is -0.5
* Also for each trial a starting point and a destination point are identified. The deadline is 5\*(distance b/w start and end)
* Agent is considered successful if they reach their destination within the steps deadline for that trial. They then get a bonus reward of 10 points
* Since I set the enforce deadline to False, the game continue beyond deadline = 0 into negative deadline but there is a hard time limit of -100. If the agent has still not reached their destination, trial ends
* The last interesting observation is that other agents (non-red cars) always move in the direction of the next way\_point meaning their behavior is accurate and predictable

I observed that agents will often reach their destination but in this random action case, that rarely happens when deadline>0, on an average the agent reaches their destination with a deadline around - 55, implying we are making 55 extra moves than is desirable. This is quite a bit of gap to close.

***QUESTION 2:*** *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?*

***OPTIONAL:*** *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?*

The states that I have defined are: self.state = (inputs['light'], inputs[‘oncoming’], inputs[‘left’], inputs[‘right’], self.next\_waypoint). This is based on the local information about the agent. The state and actions pair will result in a reward and this information will be used to perform Q learning in the next exercise. So the states chosen are the ones that influence reward and I would like to use as a starting point for Q learning. From inputs I have chosen information on whether the light is red or green. As well as the status of the other 3 cars on the intersection so that the agent learns to follow the rules of the road. I have chosen next\_waypoint since agent gets a reward or penalty depending on whether it moves to the next\_waypoint or not. I have not chosen deadline since that will add too many states and it will slow down the learning rate. Total no of state combinations here is: 2X4X4X4X3 = 384 possible states with the option I have chosen above.

***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?*

Once a Q learning agent is correctly implemented, the program will start storing state, action tuples along with q value. At each point it will choose the action that maximizes q value. We can use alpha, gamma and epsilon to adjust the behavior of the agent. Specifically I notice the following:

* In the initial 9-10 trails the agent behaves randomly and doesn’t usually reach its destination within the stated deadline. This is because the agent is still building its q dictionary of state,action tuples
* In the initial trails, I saw the agent moving in circles several times implying that it was stuck in a local minimum. The epsilon parameter really helps the agent take random actions and add more observations to its q dictionary.
* Towards the later trails, the agent is likely to reach its destination 90%+ times. The agent moves in the direction of the next\_waypoint or takes an action “None” if the move is not okay to make. This behavior is occurring because the agent is learning and choosing actions based on reinforcement learning rather than just choose actions randomly.

***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?*

To improve the performance of my Q-learning algorithm, I played with several parameters listed below. To measure performance of the agent I logged all the results in a log.txt file and manually looked at two things - 1. Total no. of successful trails in the 100 trails. This code is added in agent.py and commented out and 2. Manually checking no of successful trials in the last 10 runs.

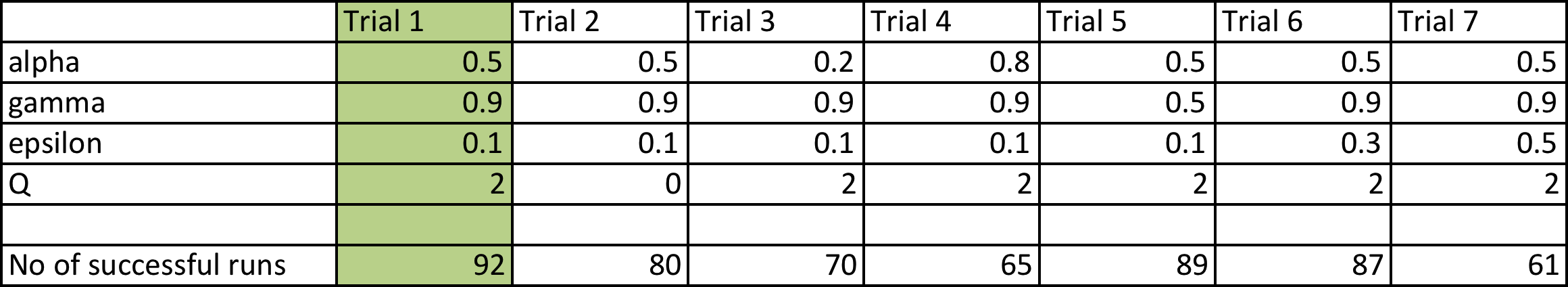
1. Epsilon parameter – I tested values 0.1, 0.3 and 0.5. Epsilon parameter determines what percent of time the agent will select random actions and expand it’s learning outside its optimal chosen state. I only wanted to test b/w a narrow range of epsilon parameters. **The best performance was obtained with epsilon of 0.1.**

2. Value for alpha parameter – I chose values 0.2, 0.5 and 0.8 to test across a range of possible combinations.  A learning rate of 0 will make the agent not learn anything, while a learning rate of 1 would make the agent consider only the most recent information. **The best performance was obtained by a learning rate of 0.5.**

3. Value for gamma parameter - Chosen values include 0.5 and 0.9. **Best performance for gamma = 0.9.** The discount factor determines the importance of future rewards. A factor approaching 1 will make Q-learning strive for a long-term high reward.

4. Initial value of Q parameter – Tried both initializing at Q = 0 and at 2.0. Much better performance with Q = 2.0

The results from my testing are in the table below:



As can be seen, the trial 1 is the best performance I got. The following parameters give the best performance:

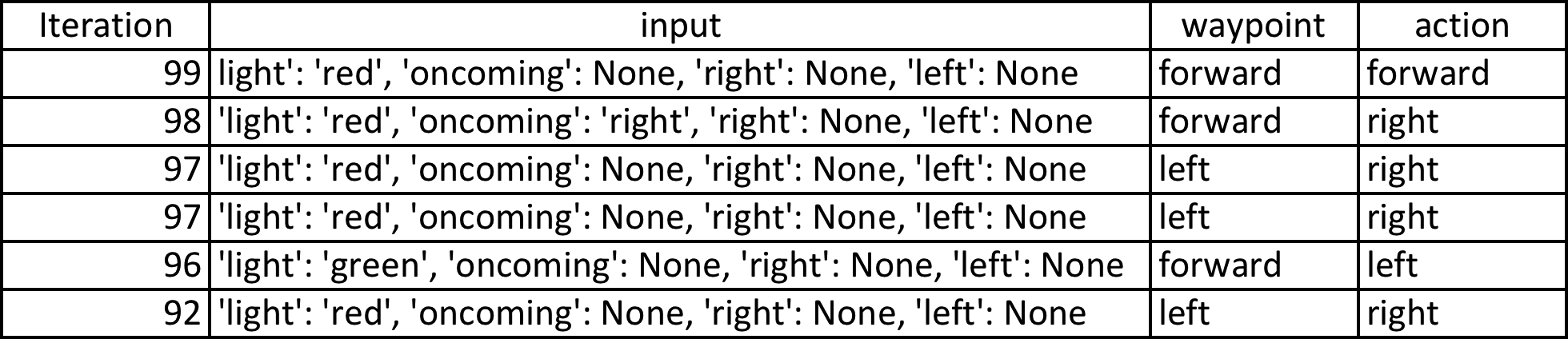
* State defined as self.state = (inputs['light'], inputs[‘oncoming’], inputs[‘left’], inputs[‘right’], self.next\_waypoint)
* Epsilon 0.1
* Alpha: 0.5
* Gamma = 0.9
* Initial Q = 2.0

**The agent successfully reaches destination in 92 of the total 100 trails! And it reaches the destination in successfully in all the 10 last trials.** This result is better than I expected. Q-learning algorithm is very powerful in teaching the agent to make the right set of moves.

***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?*

Yes my agent successfully reached the destination in 92 trails and all 10 of the last 10 trials. An optimal policy here would be a mapping of state, action pairs that tells you the best action to take when you are in a given state. The optimal policy here would instruct the agent to follow the next\_waypoint if it is safe to make a move else if there is a red light or oncoming traffic to make move = None, i.e. wait. When it is okay to move then make the same move as the next way\_point as that is the fastest way to reach its destination. This doesn’t imply that the agent will always reach the destination before deadline if the traffic/light conditions are unfavourable and the agent is forced to not move (move = None) however under the optimal policy, the agent should reach the destination most of the time.

I think my agent is getting close to following the optimal policy but there are some gaps. The table below looks at the last 10 trails and instances where the agent got to destination in time but incurred negative reward



I found one rare circumstance when the agent disobeyed the traffic law and made a move that was not okay which is the first instance in the table above when the agent moves forward on a red light. I think this state should have been explored so one reason it is doing that is because of the epsilon parameter – taking random actions in 10% of the trials.

I found several instances when the agent follows the rules of the road but moves to the destination in a sub-optimal (indirect) route. One common case was the instance 3 and 4 in the table above. The next waypoint was left but the agent moved right because the light was red. The correct action must be to do nothing and then move left when it was safe to do so. Maybe this is a state that the agent hasn’t explored yet. Or the epsilon parameter could be causing it to take random actions in 10% of the trials.