Running in Simulator:
The driving agent doesn't produce any errors in the simulator. The state, action, and reward are properly presented in the heading. The (desired) next waypoint is labeled right next to the agent as it navigates through the environment. It eventually reaches the destination, and the simulation trial restarts. The dummy agents are wandering the environment properly, and the agent stops at intersections at red lights. The interpreter is properly outputting and updating the deadline, inputs, action and reward with the print statement.

Choosing the State:
After testing different combinations of states, I boiled down the factors into two essential parts: the next waypoint (in order to reach the destination) and the state of the traffic light at the current intersection. If the agent were to receive a reward, it should know that a positive or negative reward was based on these two factors: aligning the action with the next waypoint and the action it followed during the state of the traffic light. I also considered whether to include if it sensed the presence of other cars in the intersection, but this would multiply the amount of current states by two (adding a True and False factor). With these two factors, the agent would still learn a feasible policy with a simple state of knowing the next waypoint and sensing the state of the traffic light. If I added the deadline to the state, it would exponentially increase the amount of states by a magnitude of twenty or more. The curse of dimensionality will be a factor, and the agent would not be able to learn a good policy within a deadline and within 100 trials. This would also cause some overfitting, as the agent would have to learn a policy within a specific state with each deadline mark.

Choosing the Action:
An action was chosen at random epsilon percent of the time. (1 - epsilon) percent of the time the action was determined based on the current state's q-value.

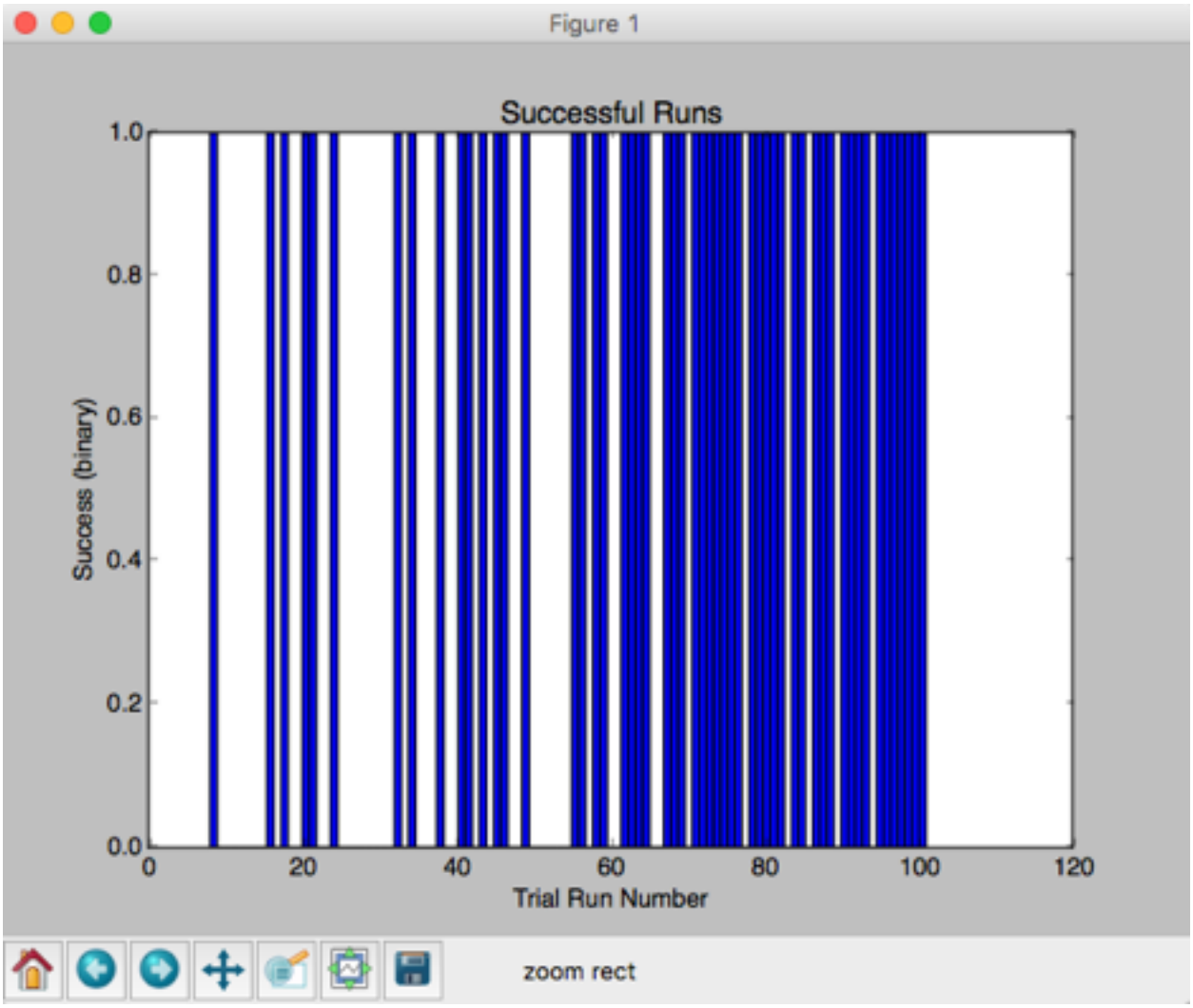
Changes in Behavior:s
Initially, the agent behaves randomly. Its actions doesn't align with the next waypoint at all. It will sit at a green traffic light a lot. As the trial advances, the agent follows the next waypoint sometimes and it tends to move more at green lights. It also doesn't know how to obey traffic lights in the beginning, but it learns to follow these laws as it receives negative rewards.

I included some analyses to observe the car's progress of being able to successfully reaching its destination within the time limit. Here are the desired results:

List of trial numbers, where the agent successfully reached the destination:
[8, 16, 18, 20, 21, 24, 32, 34, 38, 41, 42, 43, 45, 46, 49, 55, 56, 58, 59, 62, 63, 64, 65, 67, 68, 69, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 84, 85, 87, 88, 89, 90, 91, 92, 93, 95, 96, 97, 98, 99, 100, 101]

Number of successful runs:
54

In the figure below, the blue bars indicate each time that the agent had successfully reached its destination within the time limit. At first, it rarely reached its destination. By trial number 60, it is



able to consistently reach its destination with a high success rate. By trial number 90, it reaches its destination 90% of the time. In total, it reached the destination 54 times.

Agent Learns Feasible Policy:
Here are the net rewards per trial run:

List of net rewards per trial run: # all positive
[0, 18.0, 19.0, 7.0, 24.0, 13.5, 6.0, 17.0, 6.5, 31.5, 22.5, 20.5, 19.5, 13.0, 21.5, 25.5, 19.0, 37.0, 38.5, 56.0, 43.5, 23.5, 24.0, 46.0, 37.5, 31.0, 34.0, 23.0, 31.5, 27.5, 17.5, 18.0, 18.5, 23.5, 24.0, 26.0, 38.5, 43.5, 19.0, 23.5, 21.5, 32.0, 24.5, 34.5, 17.5, 11.0, 21.0, 20.5, 29.5, 38.5, 36.0, 20.0, 34.0, 21.0, 33.0, 29, 44.5, 30.5, 30, 33.0, 59.5, 30.5, 36, 15.0, 30.5, 56.0, 41.0, 46, 31, 26.5, 31, 26, 36.5, 30, 24, 31, 22, 22.5, 35, 26, 35.5, 35, 46.0, 24, 38.5, 41.0, 22, 30, 40, 24, 35, 20, 23, 43.5, 30.5, 38.0, 18, 27, 38, 10.5]

Improvements Reported: # include utility values, graph of
I tuned three parameters: alpha (learning rate), gamma (discount rate), and epsilon (randomness). Alpha and epsilon were both initially set to 1. They decayed at the same rate also. At each trial, they were decayed at a rate of 1 over the trial number (1 / trial). I set the variable, gamma = 0.82 ** trial. I chose the base number for gamma at 0.82 by averaging the number of successful runs over three tests, for each gamma base number: 0.7, 0.75, 0.775, 0.8, 0.82.

While tuning gamma, I ran the trial three times with each rate:
gamma = 0.8 ** trial
List of last 10 trial numbers, where the agent successfully reached the destination:
[]
Number of successful runs:
23
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 94, 95, 96, 98, 99, 100, 101]
Number of successful runs:
49
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 93, 95, 96, 97, 98, 99, 101]
Number of successful runs:
63

gamma = 0.7 ** trial
List of last 10 trial numbers, where the agent successfully reached the destination:
[]
Number of successful runs:
3
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 93, 94, 97, 98, 99, 101]
Number of successful runs:
74
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 96, 100]
Number of successful runs:
34

gamma = 0.75 ** trial
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 93, 94, 95, 96, 97, 98, 99, 100, 101]
Number of successful runs:
78
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 94, 95, 97, 98, 100, 101]
Number of successful runs:
65
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 94, 95, 97, 98, 99, 100, 101]
Number of successful runs:

64

gamma = 0.775 ** trial
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 93, 95, 96, 97, 98, 99, 100, 101]
Number of successful runs:
57
List of last 10 trial numbers, where the agent successfully reached the destination:
[93, 94, 95, 96, 98, 99, 100, 101]
Number of successful runs:
29
List of last 10 trial numbers, where the agent successfully reached the destination:
[100]
Number of successful runs:
11

gamma = 0.82 ** trial
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 93, 94, 96, 100, 101]
Number of successful runs:
66
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 93, 94, 95, 96, 97, 98, 99, 100, 101]
Number of successful runs:
76
List of last 10 trial numbers, where the agent successfully reached the destination:
[92, 93, 94, 95, 96, 97, 98, 99, 100, 101]
Number of successful runs:
70

Final Performance:
Here is the optimal policy for each of the six states:

```
for state, q_values in q_table.iteritems():
    print state
    print q_values
>>>
('forward', 'red') # 1
{'forward': 0.916403088077901, 'right': 0.9097482777530781, None: 1.8647011985087083,
'left': 0.9219947419002325}
('right', 'green') # 2
{'forward': 1.4857296178104615, 'right': 1.2443045890748672, None: 1.2436912189536018,
'left': 1.2725610574244341}
('right', 'red') # 3
{'forward': 0.9668227717651976, 'right': 1.6226575258038178, None: 0.9450434347890134,
'left': 0.9596962384734086}
('left', 'green') # 4
{'forward': 1.7348453909923336, 'right': 1.743091304005982, None: 1.7563510086151892,
'left': 2.0727582044612527}
```

('left', 'red') # 5
{'forward': 0.9189742412832963, 'right': 1.200381330402584, None: 0.9151762059763113, 'left': 0.9077261550719182}
('forward', 'green') # 6
{'forward': 2.358558612352207, 'right': 1.534619239448997, None: 1.5176265388727608, 'left': 1.5208372526409422}

For the first state, its intuitive to take no action when the light is red and the next waypoint is forward. The highest q-value supports this.
For the second state, the immediate reasonable action would be to turn right when the light is green and the next waypoint is right. However, the policy shows that the optimal action to take in that state in order to maximize long term reward is to turn left instead.
In state three, when the next waypoint is right and the light is red, the optimal action is to turn right.
In state four, the optimal policy aligns with the next waypoint.
In state five, the optimal policy says to turn right when the light is red and the next waypoint is left.
In state six, the agent should go forward when the light is green and the next waypoint is forward.
The highest q-values in each state is the action that that the agent should follow according to the policy.
After having observed the agent's actions and environment during the last ten trials, we see that every time the light turns red the agent either does nothing or turns right. It never turned left or went forward under any circumstances when the light was red. From this sample, we know that the car learned how to drive according to the state of the traffic light. During the last ten trials, the car didn't follow the next waypoint 74 times, and this happened in two cases. One case happens when the car takes no action and the next waypoint is forward. The other case happens when the car turns right when the next waypoint is left, and this only occurred four times of the 74.
Also, we observed how the car moved from its initial location to the destination. It had either moved closer to the destination or waited at a traffic light, so it always took the shortest path. During the last ten trials, it never moved farther to avoid a long traffic light, unless the next waypoint directed it to turn right. Of the times that the agent failed to reach the destination, it had had the misfortune of waiting at long traffic lights along the way, and it was one unit distance away from the destination when the environment reset. Since I included the next waypoint and the traffic light into the state, the agent learned to give these two priority over everything else since it only understood these two factors. In conclusion, based on the two factors I've implemented into its state, the agent had learned the optimal policy.