Project 4: Train a Smartcab to Drive

Question 1:

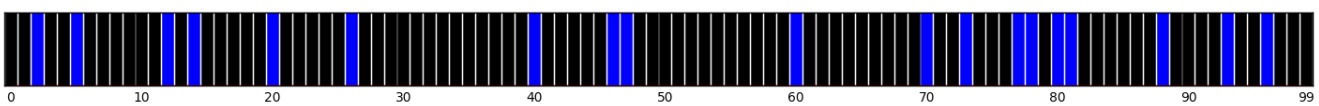
Observe what you see with the agent's behavior as it takes random actions. Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?

Answer:

The smartcab made it to its destination 19 times out of 100 trials. I notice that the successes are spread out evenly throughout the trials (see plot below). I also notice that cars can teleport between parallel sides of the city.

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline

results = [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
           1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
           1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]
win = []
lose = []
index = 0
for result in results:
    win.append(index) if result == 1 else lose.append(index)
    index += 1

print
print "Blue: Success"
print "Black: Failure"

plt.figure(figsize=(18, 1))

markerline, stemlines, baseline = plt.stem(win, [1 for _ in win], markerfmt='
 ')
plt.setp(stemlines, linewidth=9, color='blue')

markerline, stemlines, baseline = plt.stem(lose, [1 for _ in lose],
markerfmt=' ')
plt.setp(stemlines, linewidth=9, color='black')

plt.tick_params(axis='y', left='off', right='off', labelleft='off')
plt.tick_params(axis='x', top='off', bottom='off')
plt.xticks(range(0, 100, 10) + [99])
plt.xlim(-0.5, 99.5);
```

```
Blue: Success
Black: Failure
```



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?

Answer:

Substate	Possible values
light	'green'; 'red'
oncoming	None; 'forward', 'left', 'right'
left	None; 'forward', 'left', 'right'
right	None; 'forward', 'left', 'right'
self.next_waypoint	'forward', 'left', 'right'

Total states: $2 * 4^3 * 3 = 384$

Justification:

I included all and only the substates that are necessary to determine rewards. For this claim, I assumed that only the traffic light, traffic rules, and planner waypoint influence rewards.

I didn't include the specific traffic rules in the state because I need the agent to be able to learn any set of traffic rules. This is also the reason why I included `right` in the state, despite its irrelevance to the traffic rules of this environment.

I didn't include the deadline in the state because it would multiply the number of states by a number between 20 to 60, resulting in 7680 to 23040 states instead of 384. For rigor, I tested agents that had the deadline in their state and not one completed more than 90 trials.

Question 3:

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?

Answer:

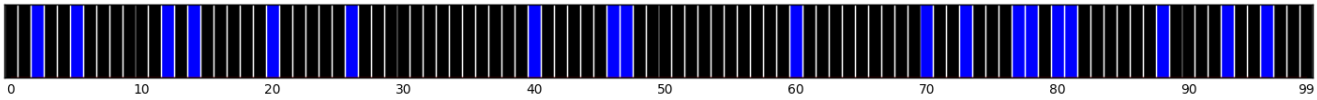
The random-moving agent failed about just as often in the last 50 trials as it did in the first 50. The Q-learning agent failed a few times in the first 50 trials but succeeded every time in the last 50 trials. In other words, the Q-learning agent improved over time, while the random-moving agent didn't. This happened because the Q-learning agent updated its behavior based on the rewards it received, while the random-moving agent didn't. I've compared the performance of the two agents in two plots below.

In [4]:

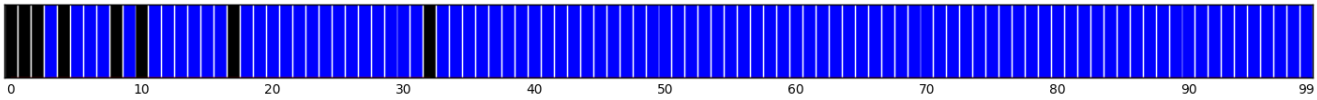
```
import matplotlib.pyplot as plt
%matplotlib inline

random_agent = [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
                1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]

learning_agent = [0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

print "\nBlue: Success"
print "Black: Failure\n"

for agent in ((random_agent, 'Random-moving agent:'), (learning_agent, 'Q-Lear
ning agent:')):
    print agent[1]
    plt.figure(figsize=(18, 1))
    win = []
    lose = []
    index = 0
    for result in agent[0]:
        win.append(index) if result == 1 else lose.append(index)
        index += 1

    markerline, stemlines, baseline = plt.stem(win, [1 for _ in win], markerfm
t=' ')
    plt.setp(stemlines, linewidth=9, color='blue')

    markerline, stemlines, baseline = plt.stem(lose, [1 for _ in lose], marker
fmt=' ')
    plt.setp(stemlines, linewidth=9, color='black')

    plt.tick_params(axis='y', left='off', right='off', labelleft='off')
    plt.tick_params(axis='x', top='off', bottom='off')
    plt.xticks(range(0, 100, 10) + [99])
    plt.xlim(-0.5, 99.5);
    plt.show()
```

Blue: Success
Black: Failure

Random-moving agent:



Q-Learning agent:



Question 4:

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?

Answer:

Out of the different sets of parameters I tried, the followed set was the best:

Alpha == 0.5, Gamma == 0.5, Epsilon == 0.0, Initial_Q == 1

The corresponding agent had **99** wins and a total reward of **1234.5**.

Wins	Reward	Alpha	Gamma	Epsilon	Initial Q
92	1202.5	0.5	0.5	0.0	10
21	-175.0	0.0	0.0	0.0	10
100	993.5	1.0	1.0	0.0	10
26	109.5	1.0	1.0	0.1	10
32	247.5	1.0	1.0	0.2	10
83	1264.0	0.5	0.5	0.1	10
79	1187.5	0.5	0.5	0.1	10
98	1091.5	0.5	0.5	0.0	0
55	976	0.5	0.5	0.0	0
100	1111.5	0.5	0.5	0.0	1
100	1122.5	0.5	0.5	0.0	1
99	**1234.5**	**0.5**	**0.5**	**0.0**	**1**
97	1061.0	0.5	0.5	0.0	1
99	1114.5	0.5	0.5	0.0	1
96	1030.0	0.5	0.5	0.0	-1
55	852.0	0.5	0.5	0.0	-1
59	848.5	0.5	0.5	0.0	0
99	1060.5	0.5	0.5	0.0	random.randint(-2,2)
58	488.0	0.5	0.5	0.0	random.randint(-2,2)
20	-223.5	0.71	0.42	0.99	-1
79	1203.5	0.45	0.42	0.34	-2
38	359.5	0.07	0.61	0.78	-7
25	154.5	0.57	0.59	0.9	1
79	1233	0.83	0.82	0.25	3

Question 5:

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?

Answer:

My agent is close to the optimal policy. I base this claim on the following interpretation of "optimal policy":

> *A policy that causes an agent to reach the destination in the minimum possible time while not incurring any penalties.*

I also base this claim on the following assumptions:

> 1. My agent learns to follow the planner's waypoints when those waypoints would not cause a penalty.
> 2. By following the planner's waypoints, the minimum time will be achieved.

Another interpretation of "optimal policy" is:

> *A policy that causes an agent to maximize its reward.*

This policy would end up causing perverse behavior in the agent. Through experiment, I found that an optimal agent (according to the first interpretation) scores around 1000 reward points with 100 wins. However, my learning agent, which makes some mistakes (with respect to the first interpretation), tends to score around 1200 reward points with around 99 wins. My agent is able to score higher than the optimal agent because some of its mistakes lead to a profit of 1.5 points. For example, if the agent moves away from the destination, it loses 0.5 points; if the agent then follows the planner, it gains 2.0 points; this results in a profit of 1.5 reward points. By moving away from the destination (i.e. making a certain kind of mistake), the agent will have more opportunities to gain rewards; with respect to an agent that gets to the destination as fast as possible. In other words, an agent that tries to maximize its rewards in this problem will waste time, which is why I call the behavior "perverse".

Below is a table comparing a planner-following agent, a Q-learning agent, and an optimal (hard-coded) agent:

Wins	Reward	Alpha	Gamma	Epsilon	Initial Q
100	472.0	(planner)	(planner)	(planner)	(planner)
99	1234.5	0.5	0.5	0.0	1
100	Around 1000	(optimal)	(optimal)	(optimal)	(optimal)

In [5]:
```
%%html
<style>
table {float:left}
</style>
```