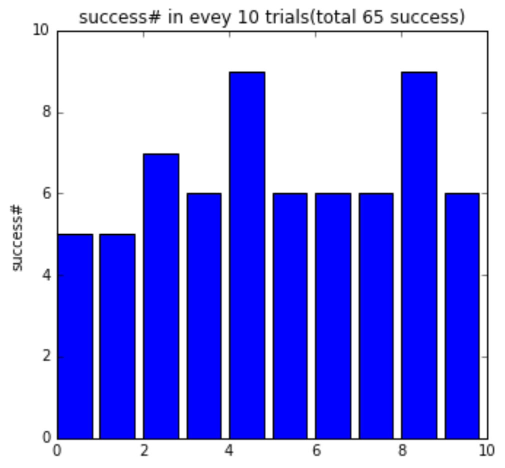
Implement a Basic Driving Agent

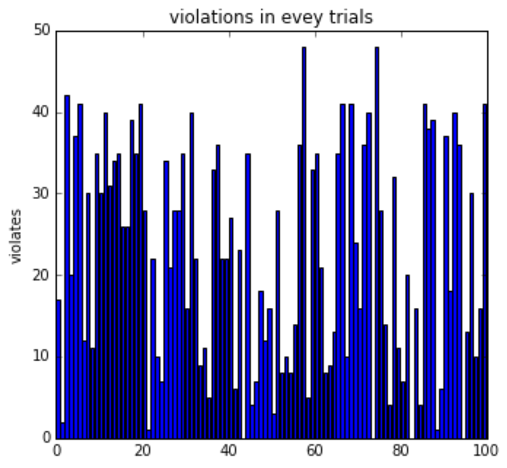
To complete this task, simply have your driving agent choose a random action from the set of possible actions (None, 'forward', 'left', 'right') at each intersection, disregarding the input information above. Set the simulation deadline enforcement, enforce_deadline to False and observe how it performs.

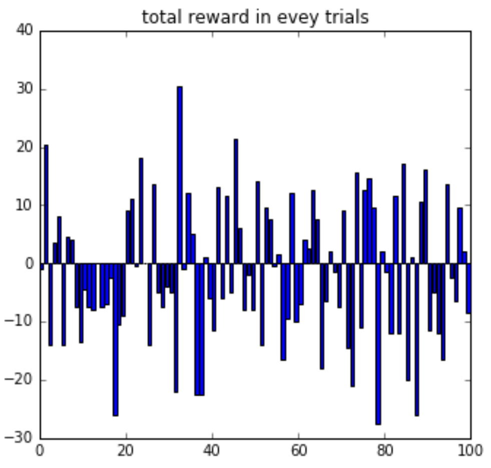
QUESTION 1: Observe what you see with the agent's behavior as it takes random actions. Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?

Answer: The agent made random decision at each intersection, disregarding the traffic lights and traffic coditions, below is an output from this random-action case, 'rewards' is an accumulative reward score in each trial, since we have 100 trials here, there are 100 entries in total, sometimes the agent can violate a lot of rules getting a negtive score. In 65 trials the agent eventually get to the destination, thought the agent is making random decision, i guess this is because the grid is too small so the agent still get very high chance to reach the destination. One interesting thing is the agent can has up to 50 violations in one trial and still reachs the destination, i'm expecting to see a large inprovement on this later.

```
In [1]:
```







Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the smartcab and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination of inputs as an implicit state. At each time step, process the inputs and update the agent's current state using the self.state variable. Continue with the simulation deadline enforcement enforce_deadline being set to False, and observe how your driving agent now reports the change in state as the simulation progresses.

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: At first i want to use 'location', 'destination', 'heading', also include the 'inputs', but then i found that would be too many states, and will not deliver a good Q-table within 100 trials, so instead of using these three features explicitly, i choose to use 'next_waypoint' to combine those three features. And 'next_waypoint' is a good generalized feature, the only thing the agent need is the correct way to get closer to the destination, no matter where is it and how far is it from the destination. As for the 'inputs', the traffic light is a must, so as the oncoming traffic, since we need to avoid traffic when turning left, and the information of traffic coming from the left is needed when turing right at a red light. But the traffic from right seems can be omitted. So in conclusion, total four features are select to present to state: 'next_waypoint', 'light', 'oncoming traffic' and 'left traffic'.

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?

Answer: Total 72 states exist, i would say this is in a reasonable range, but maybe a little too many for 100 trials, in other words, 100 trials maybe not be enough for the agent to learn and deliver a perfect Q-table, each state got 4 actions, which is 288 values to fill, plus there is very little traffic, some states even may not occur in 100 trials. Also because there is little traffic, it's not nesseccary for the Q-table to be perfect, so i say the number of states is in a reasonable range. (My question: Is there an equation or a guideline of learning time of n-states/m-actions Q-learning? In this case 72 states/4 actions, each trial has about 30 actions, at least how many trial should be performed?)

Implement a Q-Learning Driving Agent

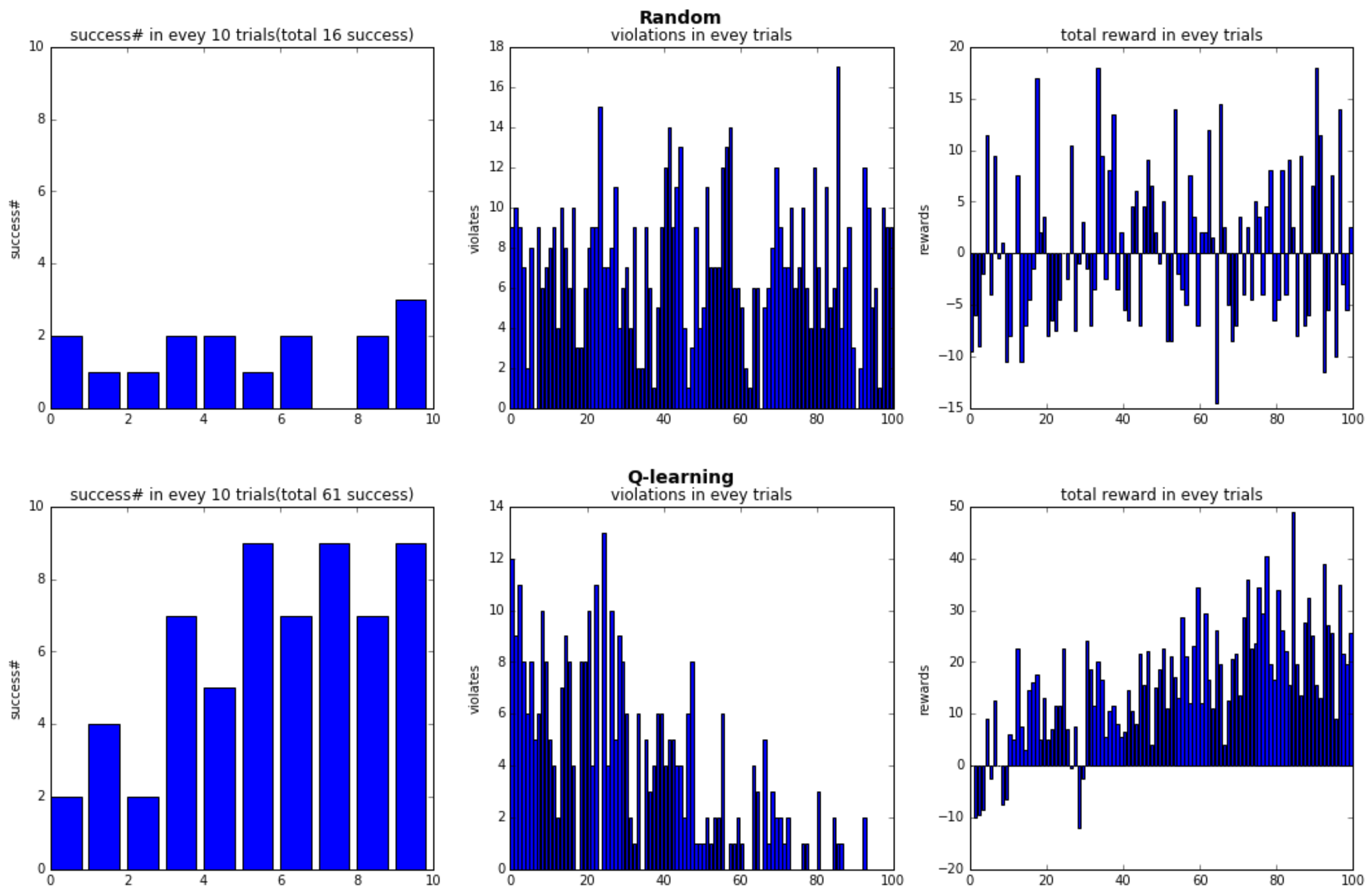
With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm for your driving agent to choose the best action at each time step, based on the Q-values for the current state and action. Each action taken by the smartcab will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement enforce_deadline to True. Run the simulation and observe how the smartcab moves about the environment in each trial.

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: Since the 'enforce_deadline' is set to false in previous case, i ran another 'random' case with 'enforce_deadline' set to True, so it make more sense when compare to 'Q-learning' case. In this Q-learning process, i set the gamma to 0.8, alpha and epsilon are both 1.0 at the beginning, and decrease by 0.01 per trial, so at the last trial they will be 0.

After implement Q-learning, the first change i noticed is the success rate becomes higher and higher as the trials goes, especially in the late part, the success rate in every 10 trials is 70% to 90%, while it's about 20% in the 'random' case. This is because in the late part, the Q-table has been updated many times, and the agent has the higher chance to make the 'best' action as the epsilon goes down. Second change i noticed is the number of violations in every trial keep decreasing, in last several trials the violations basically remains zero, while this number could be pretty high in 'random' case. The third thing is the total reward score getting larger and larger, after about 30 trials negtive reward doesn't appear, the increasing success rate and decreasing violations both made contribution on this.

```
In [2]:
```



Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the smartcab is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (alpha), the discount factor (gamma) and the exploration rate (epsilon) all contribute to the driving agent's ability to learn the best action for each state.

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: I tuned 3 different values for epsilon: 0.1, 1 - 0.01*t and 1/(1.06^t), 't' stands for number of trials. For each epsilon value, the average score of 10 tests is used to plot below picture, each test is independent from others and every test has 100 trials.

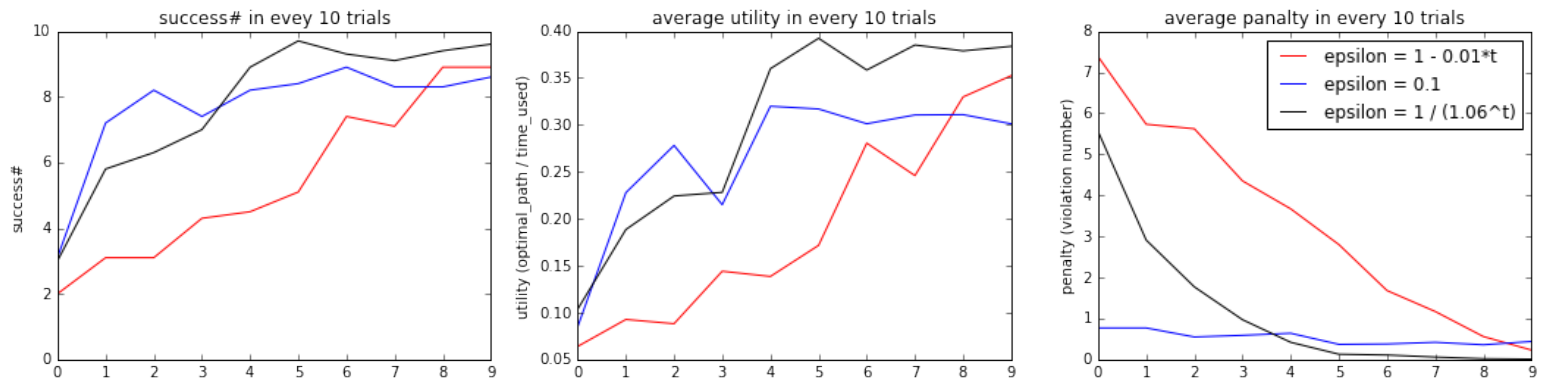
Among these three epsilon values, when epsilon equals 1/(1.06^t), the agent shows the highest sucess rate and highest utility, especially in the late part. At last the driving agent is able to reach the destination in about 95% times and in zero penalty, but usually not in the minnimum possible time.

When epsilon equals 1 - 0.01*t, the success rate and utility increases and penalty decreases, but in the last part, the penalty still not zero, success rate and utility is second highest among three cases.

When epsilon always equals 0.1, the agent shows about 80% success rate and low penalty at early stage, but did not get improved as the process goes, so at last its perfomance is the poorest among three.

I would say the driving agent is doing pretty good, since it almost finish its job every time and get no penalty, in the case of epsilon equals 1/(1.06^t).

```
In [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: The agent gets pretty close to the optimal policy in the end, in my opinion, i use three factors the describe the optimal policy, first is the success rate, second is the penalty and third is the utility. Success rate is the number of success deliver in 10 consecutive trials. Utility is a value to show how efficient the agent finish its job, it is the optimal path divided by the actual path the agent used to reach the destination. So '1' means the agent goes the shortest path and meets no reds, and '0' means the agent couldn't reach the destination in the time limit. And penalty is the number of '-1' reward get in every 10 consecutive trials.

So my agent does great on the penalty, since it can keep zero penalty in the end. The success rate is not 100%, but still about 95% in the end. The utility is much more lower than the ideal value '1'. This because there are always red lights in the path, and the planner is not considering the map could be wrapped around, while the car actully can cross the boundry and appear in the other side, the shortest path is calculated based on the map can be wrapped around, so this values is not get close to '1'.

```
In [ ]:
```