Self-Driving Agent

Summary

The intention of this exercise is to teach a self-driving car (Agent) to automatically drive to reach the final destination maximizing rewards. Q learning algorithm is implemented.

Key Factors:

Rewards:

Any time the agent acts, a reward is reached, the possible rewards are:

- 10: Agent reaches destination
- 2: Agent action is valid and equals the next_waypoint() policy
- 1: Agent action is None
- 0.5: Agent action is valid but is not the next_waypoint() policy
- -1: Agent action is not valid

Planner:

Agent uses a planner to reach destination, **this planner identifies the next waypoints** agent should follow to reach destination in a grid environment. However planner is limited since the process to find the optimum waypoints firstly checks for optimal waypoints in the North-South direction and, in case None is found, then in the East-West direction. As a consequence, in cases where there are two optimal waypoints, only the waypoint in the North-South direction is reported.

States:

Agent' state includes the following information:

- Street light: Agent's movements are determined by traffic light color and so rewards
- Next waypoint: Agent's actions must be oriented to the next waypoint defined by the Planner in order to reach destination, rewards = +2 when a valid action drives the agent to the next waypoint

Since rewards are related to street light colors and next waypoint, agent' state need to include this information in order to learn which action maximizes rewards and so learn how to drive towards destination. Some more information could be added to the agent' state, however since rewards are just function of these variables, there is no benefit in including more info while it would make the problem more complex to solve.

IMPORTANT I: Traffic conditions are not considered since rewards does not include them as a variable. This is an environment deficiency probably originated because there are not much trafic and its implementation is probably difficult.

IMPORTANT II: Time steps remaining or distance to destination are not included since rewards are not defined as a function of any of these factors. A way to improve this project is probably to include a reward of -0.5 when the agent performs a valid action that increases its relative distance to destination, with this information included in the reward definition, the relative distance to destination would be another factor in the agent' state definition.

Q-Learning:

In order to provide the ability to the agent of learning from previous states, a Q-matrix is generated. Each element of this Q matrix is an agent state&action which includes the following information:

- Street light color
- Next waypoint
- Action taken

Each state&action in the the Q-matrix has a particular value that is updated accordingly to the reward obtained. At the very beginning of the game, all possible pairs of states and actions had been assigned to a value of 0 but as the agent walks through the different states and takes different actions received rewards are used to update values. Values are updated with the received reward of the current state plus the maximum Q-value for the next state among all possible actions.

Q-learning agent:

Agent target is to reach destination while maximizing rewards received, for such reason q-learning agent follows the Planner policy but "senses" environment states (street light + Next waypoint) to decide which action take. The way the agent moves around the environment is the following:

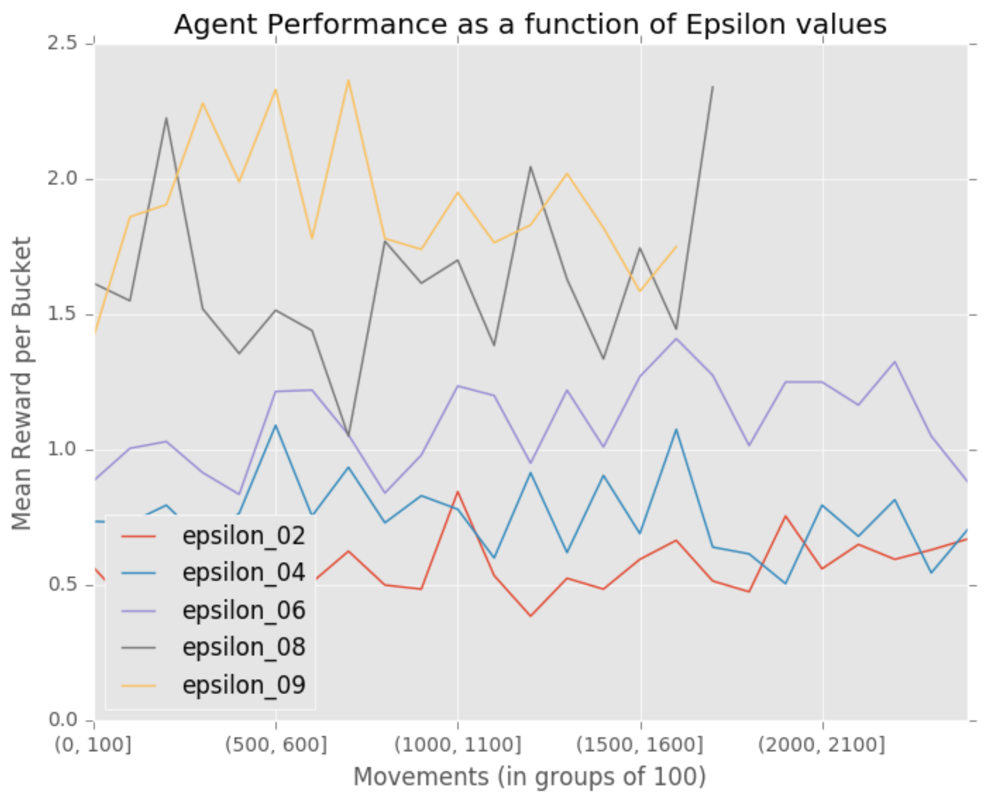
1. Sense current state (state1)
2. Choose an action (action1) that maximizes Q-equation from this current state
3. Take the selected action and move agent to the next state (state2) and get a reward
4. Sense the new state and update the Q-equation for (state1,action1) using the reward just received and the (state2,action2), being action2 the action that maximizes Q-equation for state2

Learning process of the Q-learning agent:

Q-learning agent uses Q-learning equation to identify the best action for any possible state. At the very beginning Q-learning equation equals zero for all possible states&actions, in this phase agent simply performs random actions, of course this has negative consequences since in many occasions it performs forbidden actions with negative rewards associated. However as the agent keeps driving and moves around through different states, occasionally it falls in a state it was previously, and since it follows Q-learning principle of taking the action that maximize rewards, agent takes the action that maximizes rewards according to its accumulated experience. Apart of this, since there are several possible actions with positive rewards associated, a greedy policy is included to allow the agent to explore alternative actions to the currently more suitable. For example, in this particular problem agent receives a reward of +1 when action=None, so if the agent at the beginning of the game discovers that not moving has associated a positive reward, then it always choose this action as optimum, but since the greedy policy is included, eventually it discovers that moving accordingly to the Planner has a bigger reward associated: +2. This is how the agent learns to follow the optimal policy and stop when it is not possible.

Greedy Policy (epsilon), Learning Rate (alpha) and Discount Factor (gamma) tuning:

As previously described, a Greedy Policy is introduced to allow the agent to occasionally not to take the optimal action according to its experience. Next graph shows how different values of epsilon affected to the agent performance over 100 trials. In the X axis are included buckets of 100 agent movements and Y-axis represents the mean reward per bucket. Low epsilon values have associated low means while high values of epsilon high means, another conclusion is that higher epsilons have associated a more efficient driving, since it tends to reach destination in fewer movements, for example epsilon values higher than 0.8 finish 100 trials in less than 2000 movements.

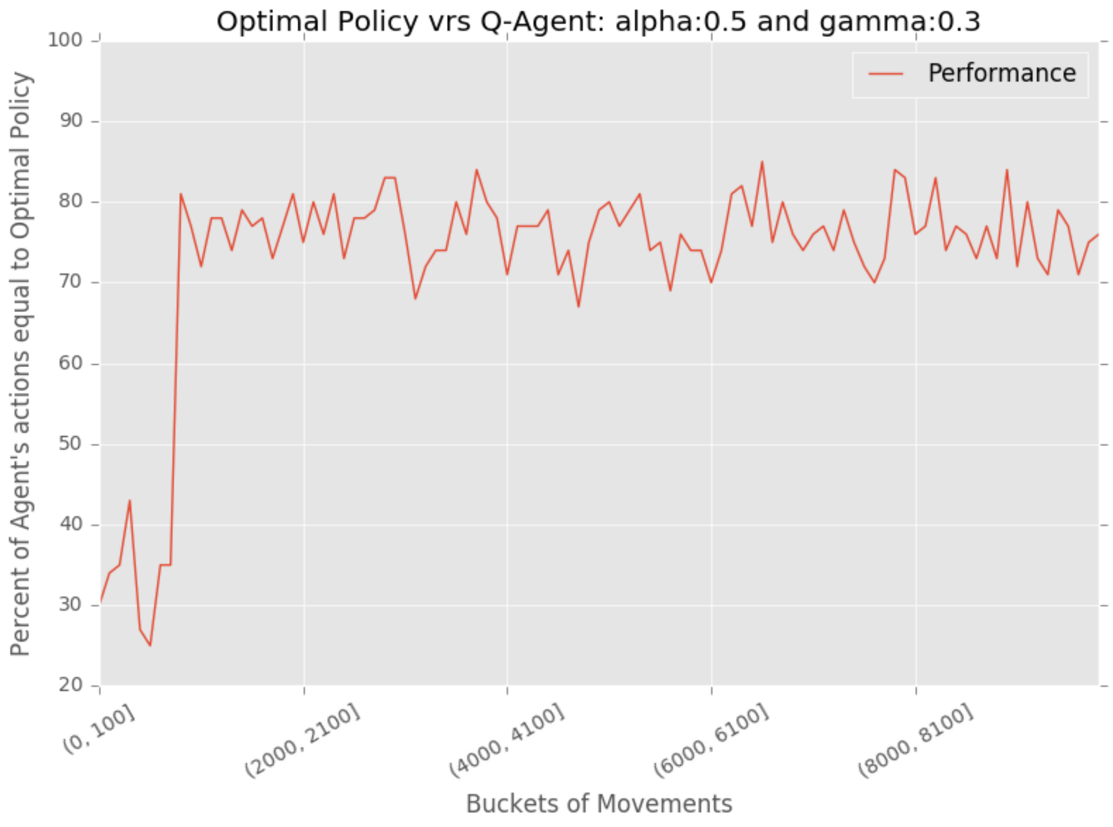


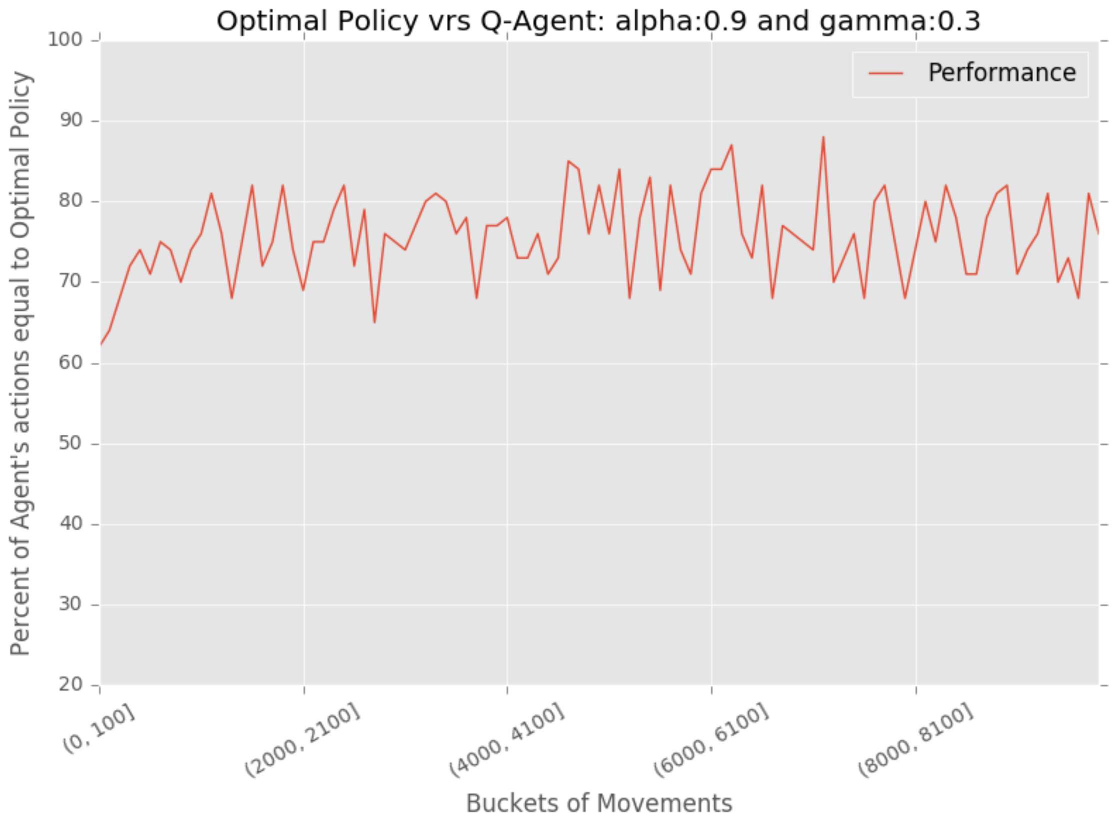
The parameters used in the Q-value update process are:

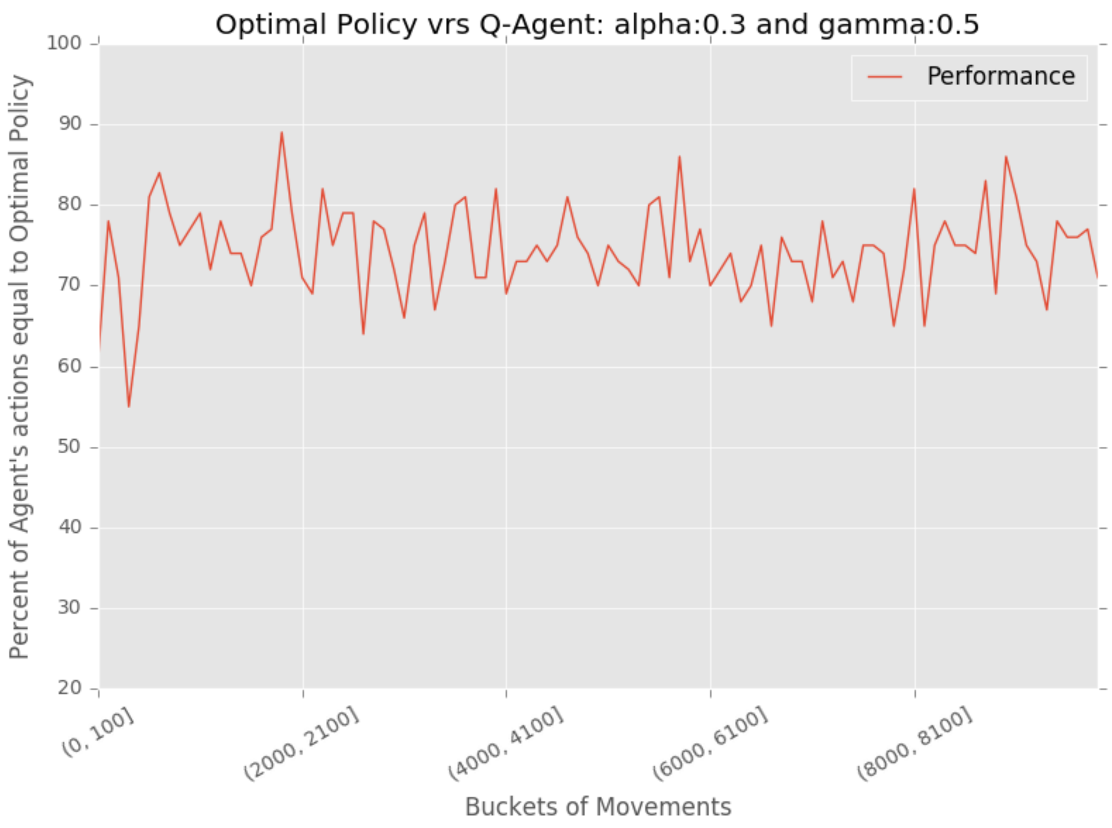
- the learning rate (alpha), set between 0 and 1. Setting it to 0 means that the Q-values are never updated, hence nothing is learned. Setting a high value such as 0.9 means that learning can occur quickly.
- discount factor (gamma), also set between 0 and 1. This models the fact that future rewards are worth less than immediate rewards. Mathematically, the discount factor needs to be set less than 0 for the algorithm to converge.

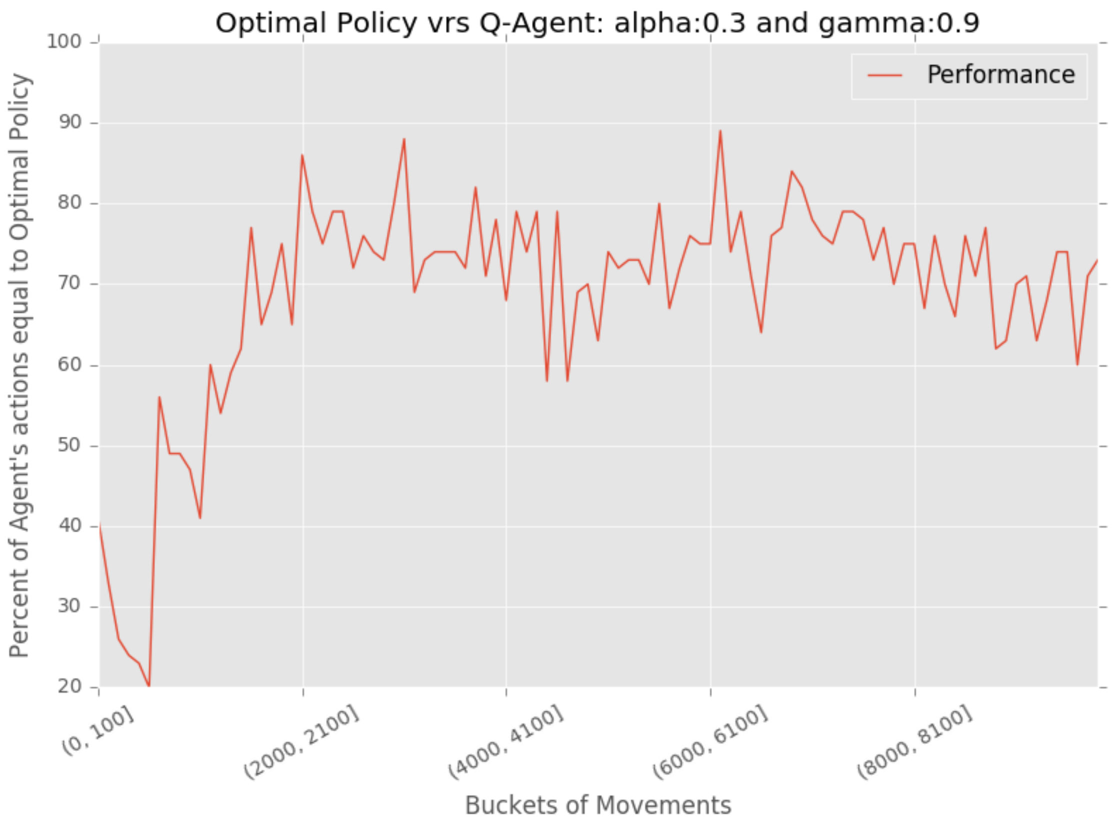
In order to tune Gamma and Alpha values, agent's performance is benchmarked with the Optimal policy, which is defined as follows: Agent follows the Planner expect on those situations when it is not possible, in that case agent's action is None.

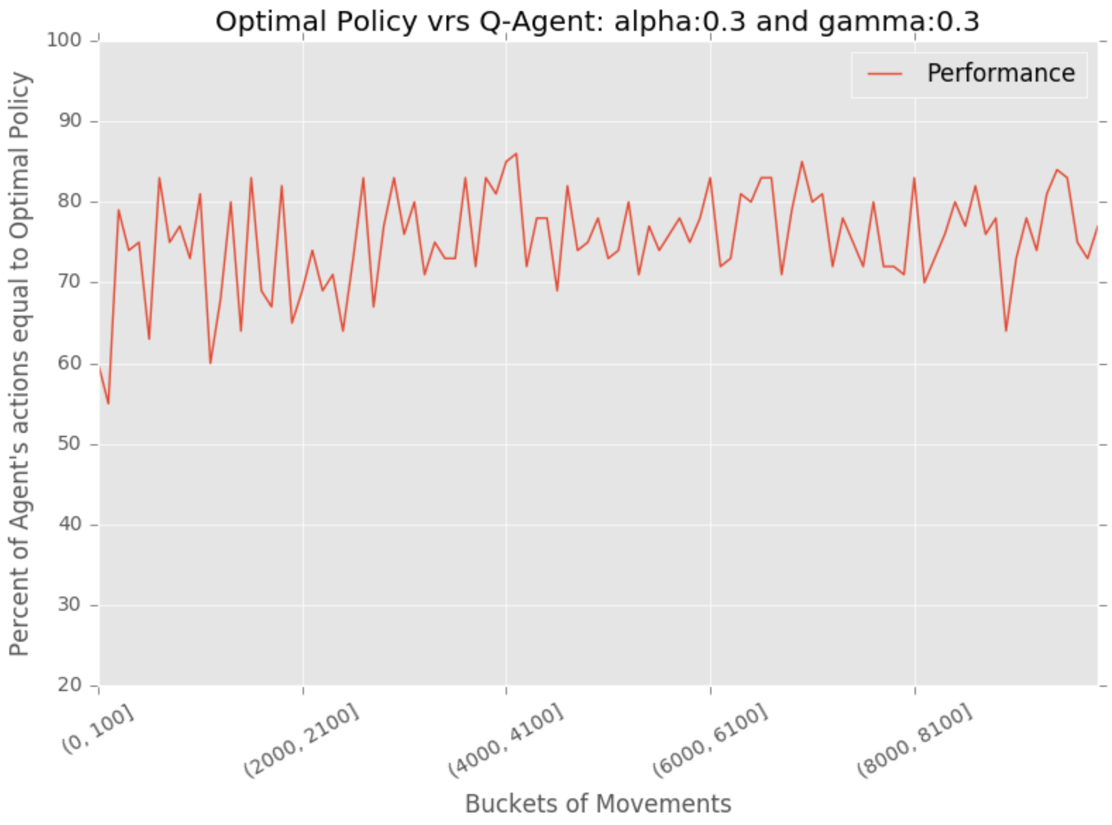
Following graphs shows agent's performance (using epsilon=0.9) for 800 consecutive trials for different combinations of alpha and gamma with the objective of identifying which parameters combinations maximize agents performance according to the Optimal policy defined. In the Y-axis it is represented the percentage of times agent follows the Optimal Policy for the different 100-movement buckets in X-axis.

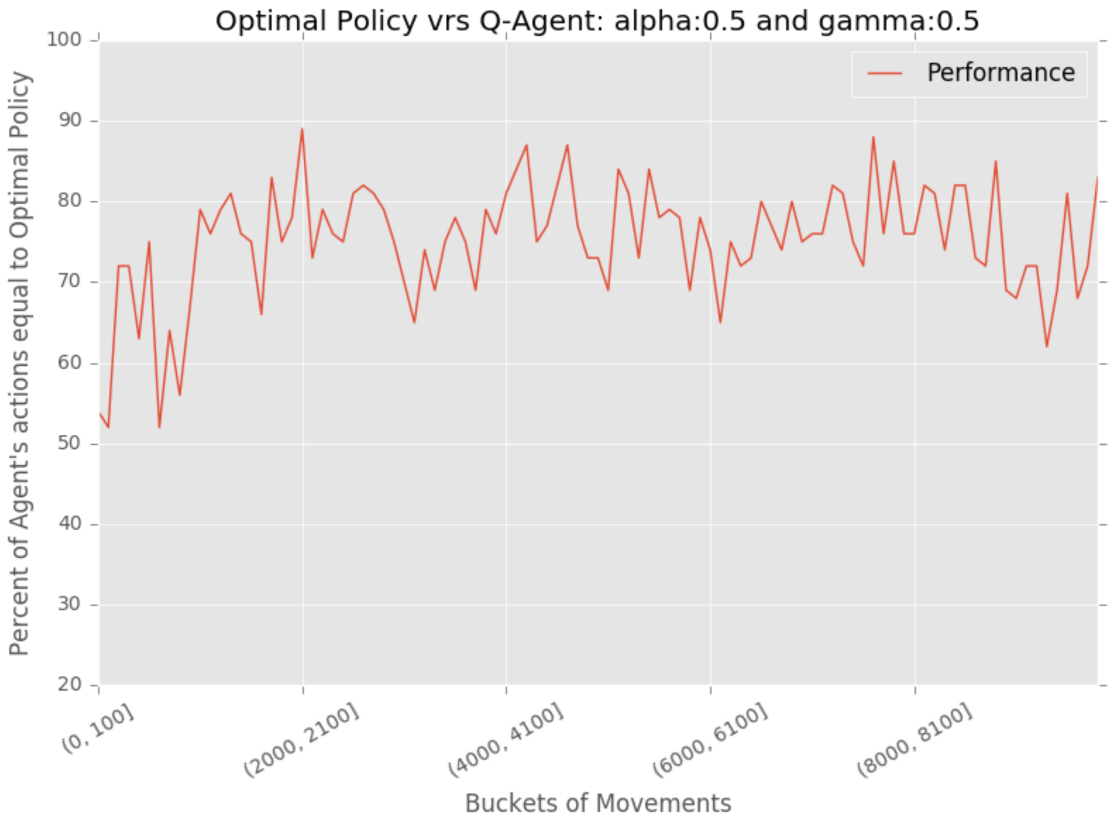

Optimal Policy vrs Q-Agent: alpha:0.5 and gamma:0.3
Performance
Percent of Agent's actions equal to Optimal Policy
100
90
80
70
60
50
40
30
20
(0, 100]
(2000, 2100]
(4000, 4100]
(6000, 6100]
(8000, 8100]
Buckets of Movements

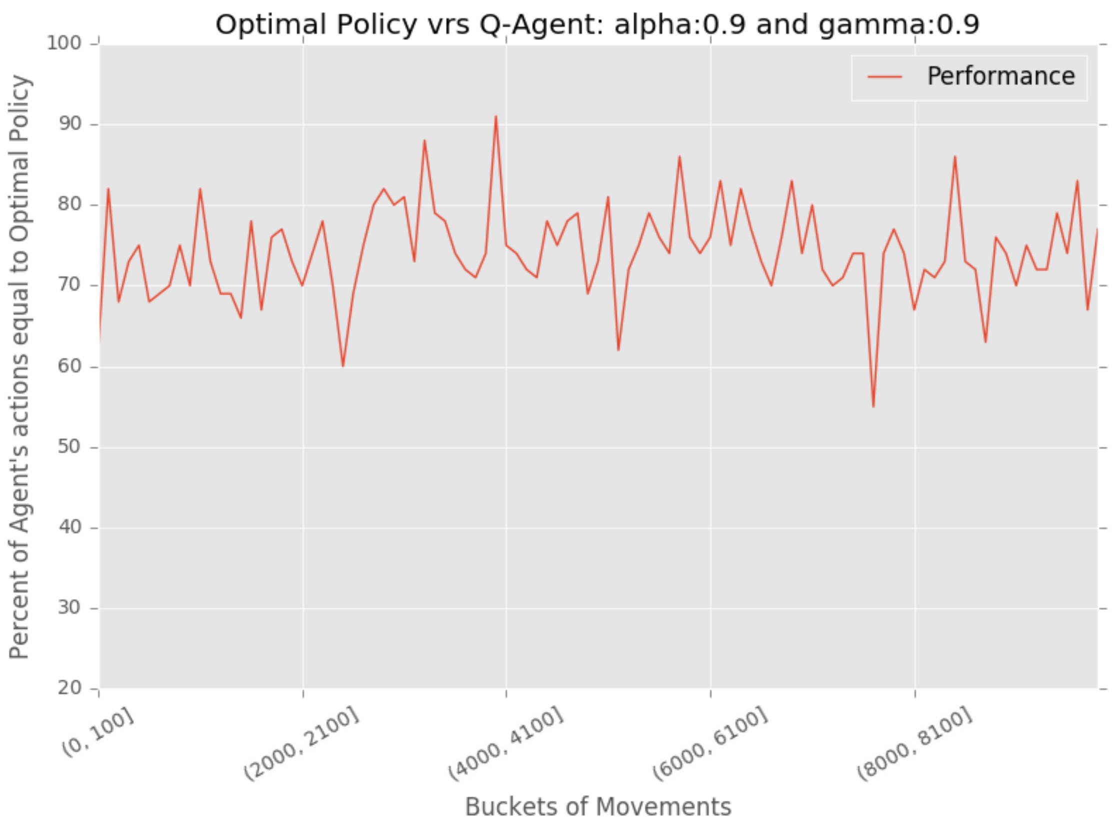

Optimal Policy vrs Q-Agent: alpha:0.9 and gamma:0.3
Performance
Percent of Agent's actions equal to Optimal Policy
100
90
80
70
60
50
40
30
20
(0, 100]
(2000, 2100]
(4000, 4100]
(6000, 6100]
(8000, 8100]
Buckets of Movements


Optimal Policy vrs Q-Agent: alpha:0.3 and gamma:0.5
Performance
Percent of Agent's actions equal to Optimal Policy
100
90
80
70
60
50
40
30
20
(0, 100]
(2000, 2100]
(4000, 4100]
(6000, 6100]
(8000, 8100]
Buckets of Movements


Optimal Policy vrs Q-Agent: alpha:0.3 and gamma:0.9
Performance
Percent of Agent's actions equal to Optimal Policy
100
90
80
70
60
50
40
30
20
(0, 100]
(2000, 2100]
(4000, 4100]
(6000, 6100]
(8000, 8100]
Buckets of Movements


Optimal Policy vrs Q-Agent: alpha:0.3 and gamma:0.3
Performance
Percent of Agent's actions equal to Optimal Policy
100
90
80
70
60
50
40
30
20
(0, 100]
(2000, 2100]
(4000, 4100]
(6000, 6100]
(8000, 8100]
Buckets of Movements

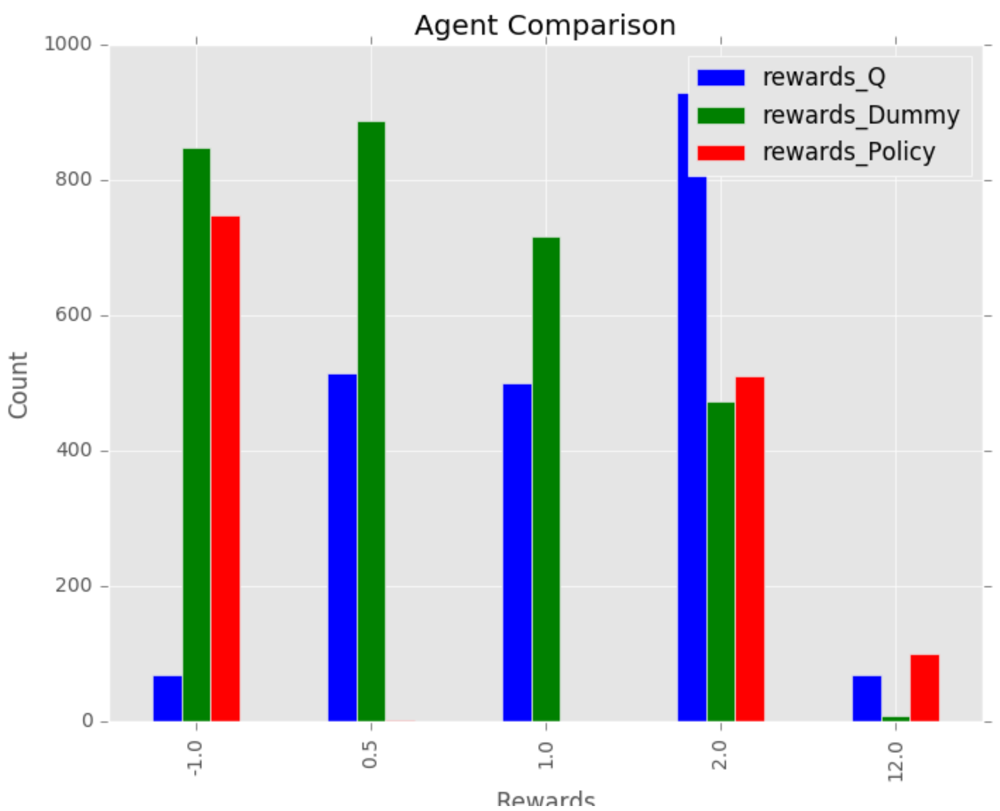

Optimal Policy vrs Q-Agent: alpha:0.5 and gamma:0.5
Performance
Percent of Agent's actions equal to Optimal Policy
100
90
80
70
60
50
40
30
20
(0, 100]
(2000, 2100]
(4000, 4100]
(6000, 6100]
(8000, 8100]
Buckets of Movements



According to these graphs, agent's performance seems to be 70%-80% around the Optimal Policy after a few hundreds of movements, however alpha=0.9 and gamma=0.9 is used as it presents more constant results since the beginning of the trials.

Results:

In the following graph it is represented the total number of different rewards for a simulation of 100 trials for the Q-learning agent (let's call it Q-Agent), agent that always follows the Planner (let's call it P-Agent) and a dummy agent that takes random actions (let's call it Dummy).



As we can see Q-agent maximizes +2 rewards (since it follows Q-equation principle) while P-agent more often reaches destination but with significant -1 rewards (since it drives towards destination all the time and in many occasions actions taken are not allowed). To end, Dummy hardly reaches destination (since it moves randomly).

Conclusion:

Q-Agent moves around the environment following the Q learning principle of maximize rewards at every state. As a consequence at the very beginning of the game it moves randomly but it soon tends to repeat those movements that produced a positive rewards, for example right-loops or not taken any action are quite common. However as the agent gains more and more experience, and the greedy policy helps him to investigate alternative actions, agent learns more and resolves situations differently. As said, at the beginning the agent basically moves basically randomly around the environment, but step-by-step it discovers those state&action with positive rewards and try to repeat them as much as possible (right loops or none action), and as it gains more and more experience, agent starts to drive towards the destination more consistently and so following the optimal policy.