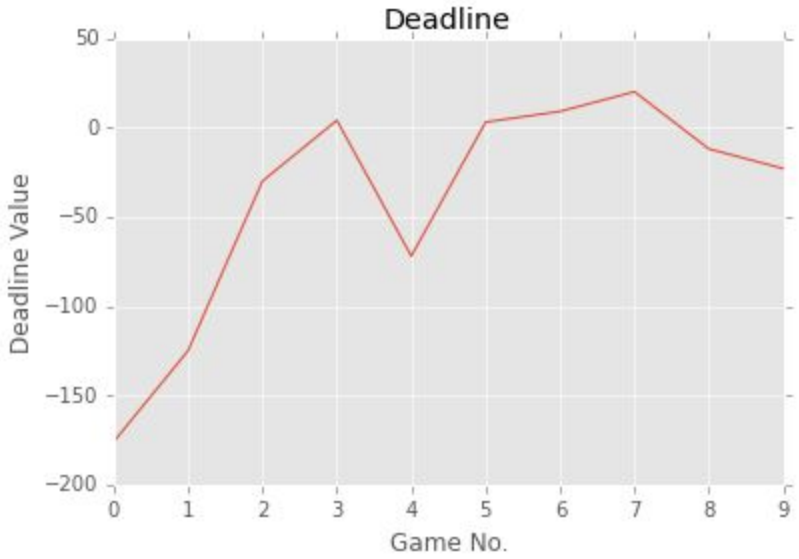
Udacity Machine Learning: SmartCab

v2 2016/04/27
v1 2016/04/25

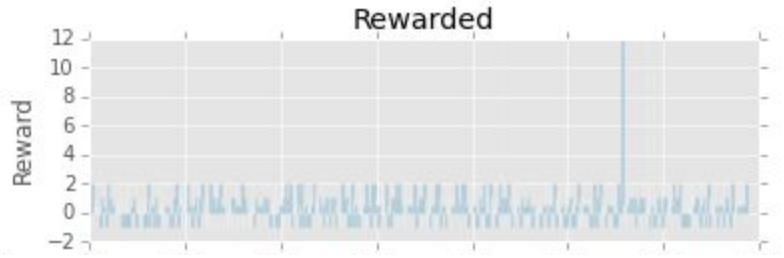
Implement a basic driving agent

In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?

It does reach the target; appears accumulate a lot of -1 rewards for not obeying to the 'law' e.g. attempts to run through red lights. Movements, as one would expect, is random.



The above plot displays the value of the final **deadline** after each of the 10 iterations.



The above plot displays the reward distribution for each turn, it illustrates a fluctuation between positive and negative rewards - showing no learning was achieved.

Identify and update state

Justify why you picked these set of states, and how they model the agent and its environment.

The following is a list of variables I used to represent state:

- *light* {green,red}
- *next_way_point* {none, forward, left, right}
- flag indicating whether the *next_way_point* was obstructed or not based on the road rules (taken from the DummyAgent class from the environment.py file). The labels included {free, obstructed}.

light was chosen as I deemed it as a critical signal to adhere to the rules (avoid penalties). *next_way_point* provides hints (/direction) to where the target resides, obviously important to help navigate our agent to the goal.
Lastly I wanted someway to incorporate the awareness of the other agents (cars) but to avoid having too many states that would jeopardise 'effective learning', therefore I compressed them into this one variable that, I hope, would capture a lot of that information.

Deadline (*time steps remaining*) was also made available but not used for the same argument as not including all the directions for obstructions i.e. would have the state space too large and require more learning.

Implement Q-Learning

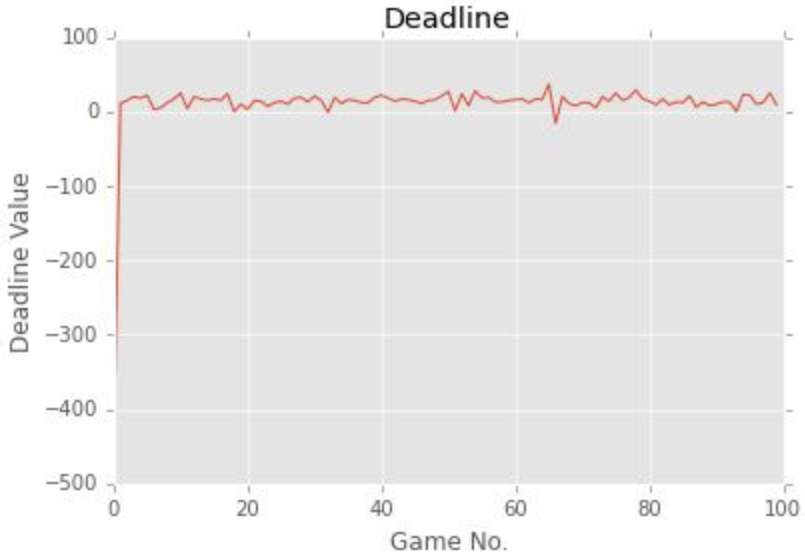
What changes do you notice in the agent's behavior?

I did notice some patterns form, in some instances it looked stuck in local minima (especially at the start) but eventually formed a pattern that appeared to efficiently navigate towards the goal.

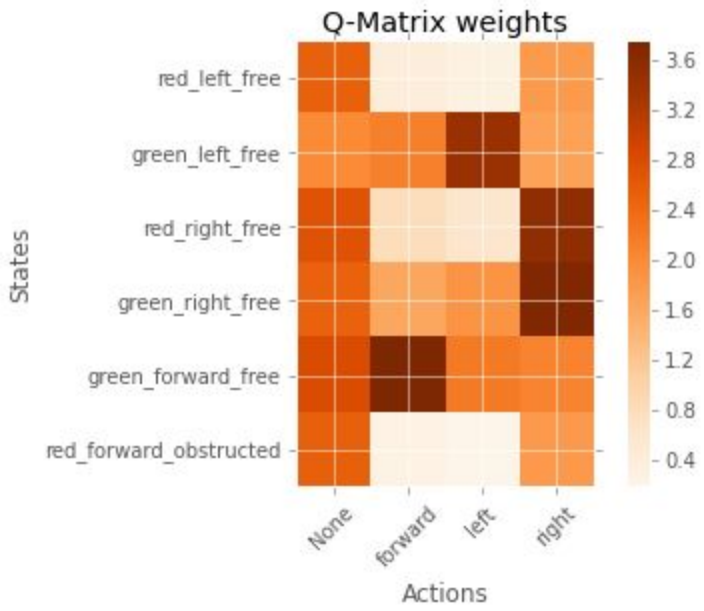
The following plots illustrates the behaviour and performance of the agent; the profile was:

- Learning rate was fixed at 0.5,
- Discount factor was fixed at 0.5,
- Epsilon = 0.1 (fixed)
- Trials = 100
- Enforcing deadline = False

The above plots illustrate the reward for **each** action (line graph) and how often the agent explored vs. using what had been learnt (scatter plot).



The above plot shows the deadline achieved for each trial; It can be observed that the initial trial, where the quality matrix was empty, performed poorly but as time went on, the agent performed a lot better.



The final plot shows the (final) weights for each action for each state. It appears the agent has learn't the road rules.

NB; this implementation had elements of reinforcement learning (Exploration-Exploitation dilemma) to avoid being stuck at an initial state.

Enhance the driving agent

Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

Changes included:

- I wanted to make more effective use of exploration i.e. force all actions to be explored for each state; for this reason I implemented filtering (when exploring) such that duplicates and zeros took precedence i.e. if the state vector contained 2 actions with the same value, then the the random selection would be out of these.
- I also added similar filtering to the exploitation branch (i.e. when selecting the strongest action based on the quality values) such that if there were an instance where 2 or more of the highest ranked actions then a randomly selection was performed on this subset. *This and the above modification were implemented to avoid local minima.*
- *epsilon* reduced over time using the equation 1/trial count; under the premise that we should be more confident with our decision as we gain more experience.
- Finally the *learning rate* and *discount factor* were adjusted and compared, as described below.

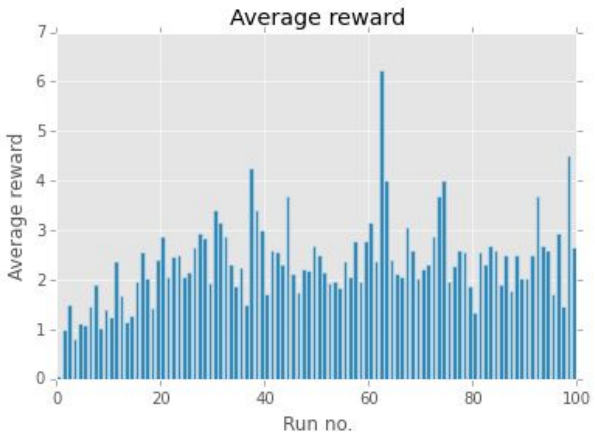
NB; the motivation for a lot of the procedural code was to try and avoid local minima.

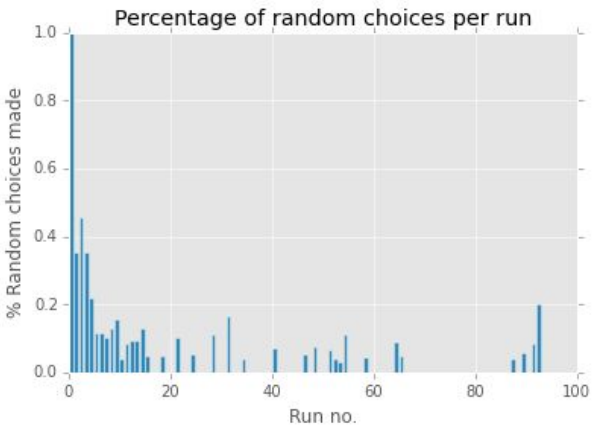
The following is the results of 5 ‘experiments’ (tweaking variables) which are used to determine the best variables/policy for our Smartcab.

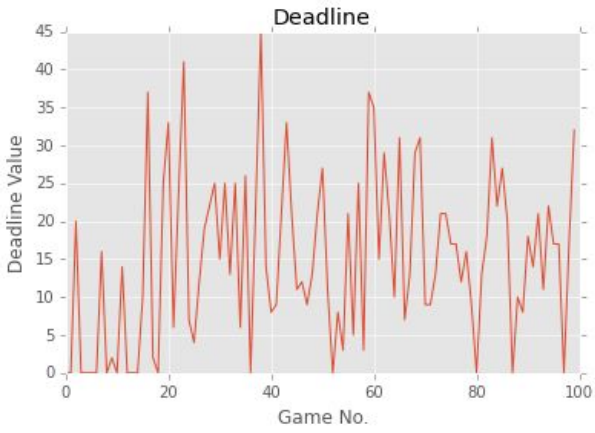
Experiment 1

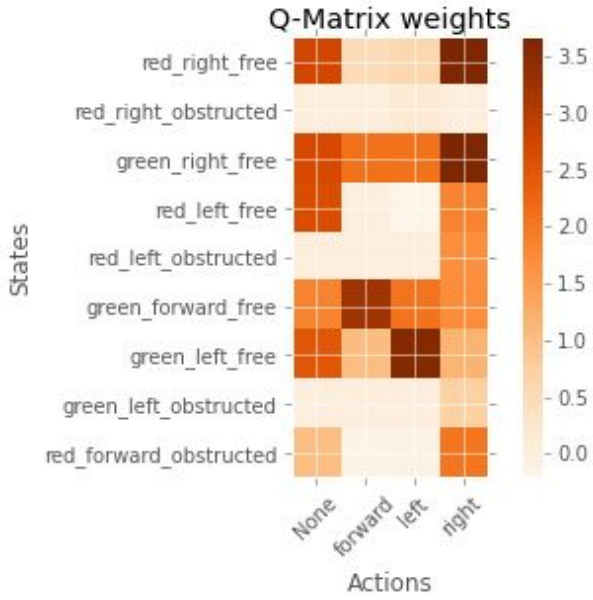
- **Learning rate was fixed at 0.5**
- **Discount factor was fixed at 0.5**
- Epsilon = 1/current runs
- Trials = 100
- Enforcing deadline = True

The following plots are presented to help compare performance of the modifications.





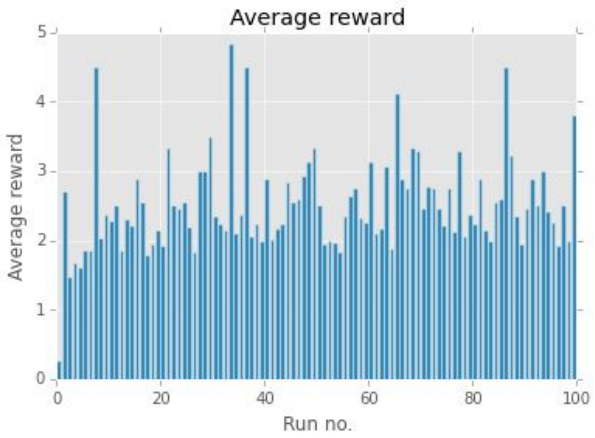


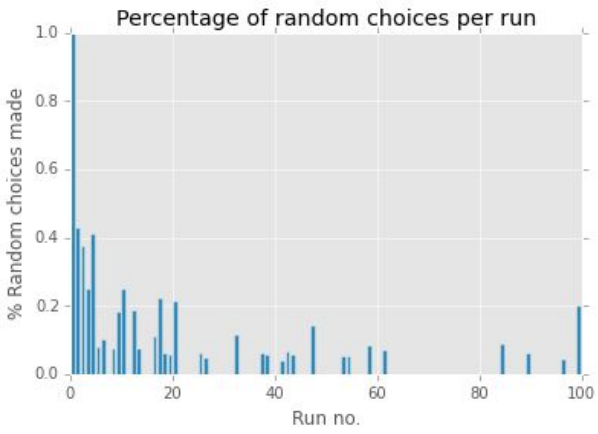


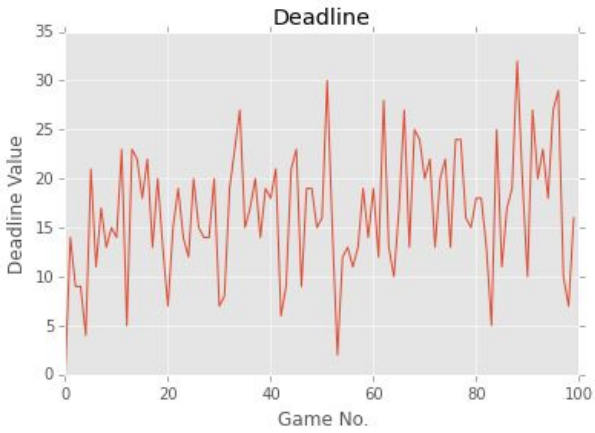
Experiment 2

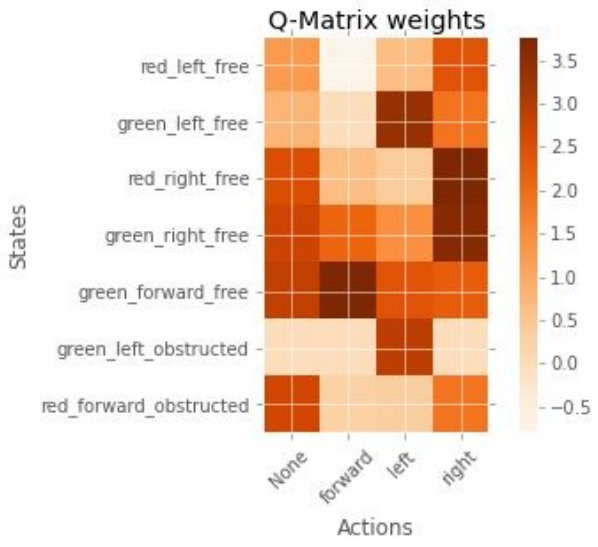
- **Learning rate was fixed at 0.8**
- **Discount factor was fixed at 0.5**
- Epsilon = 1/current runs
- Trials = 100
- Enforcing deadline = True

The following plots are presented to help compare performance of the modifications.







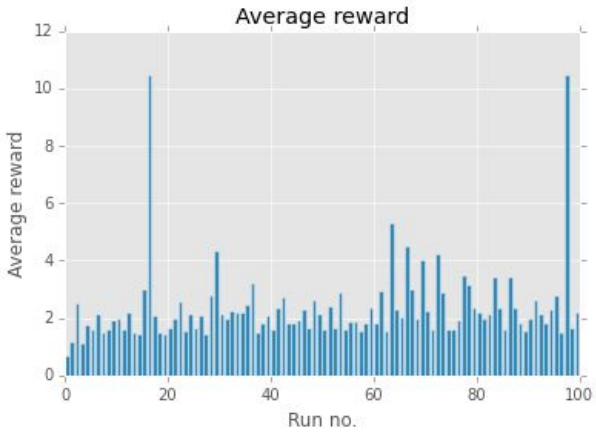


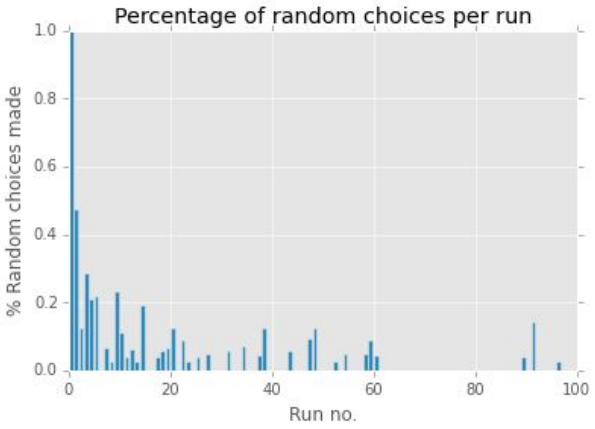
Experiment 3

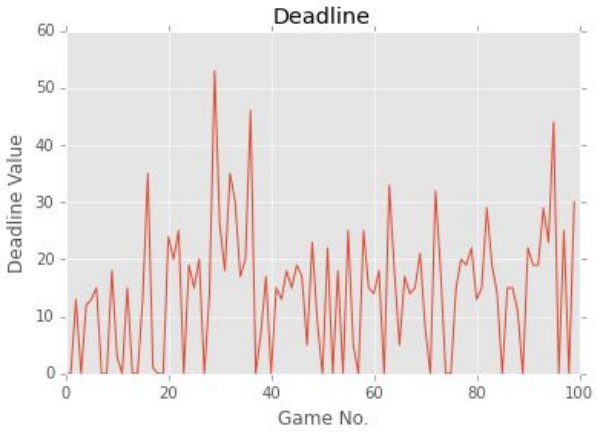
- **Learning rate was fixed at 0.5**
- **Discount factor was fixed at 0.8**

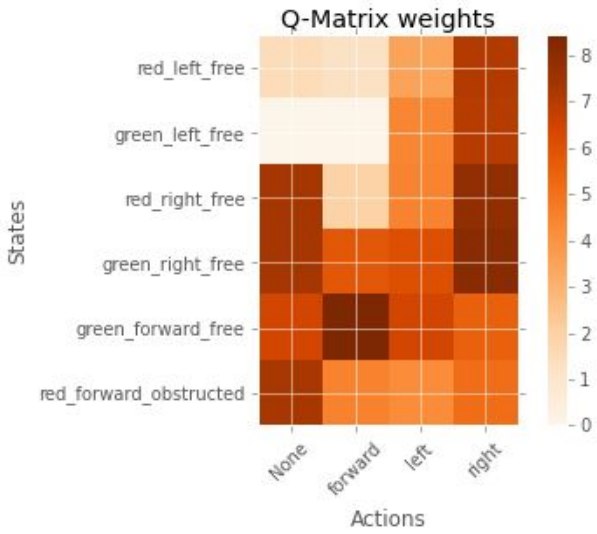
- Epsilon = 1/current runs
- Trials = 100
- Enforcing deadline = True

The following plots are presented to help compare performance of the modifications.





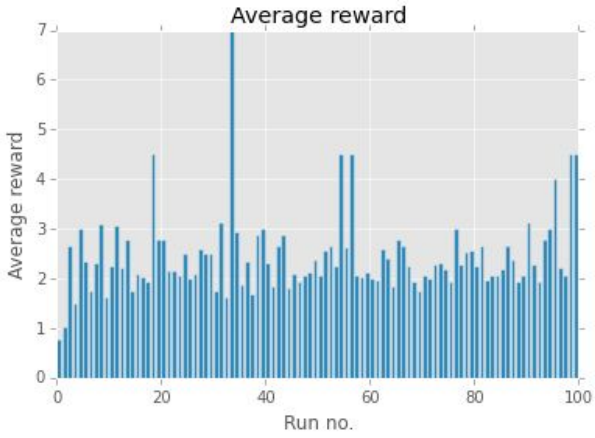


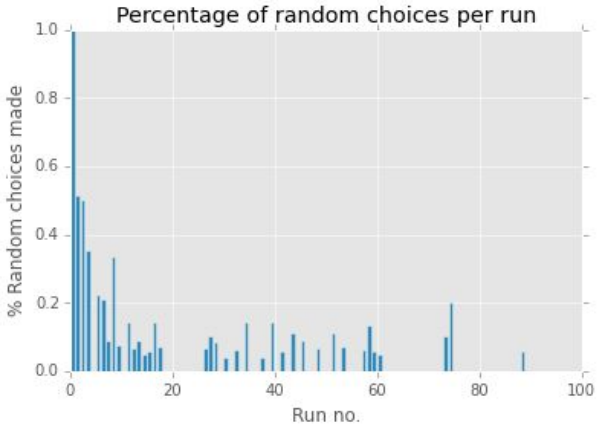


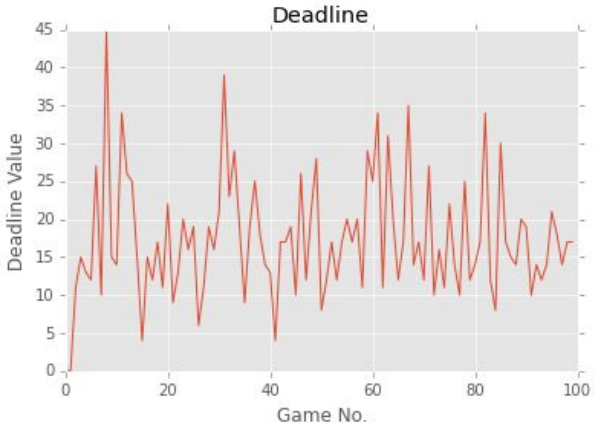
Experiment 4

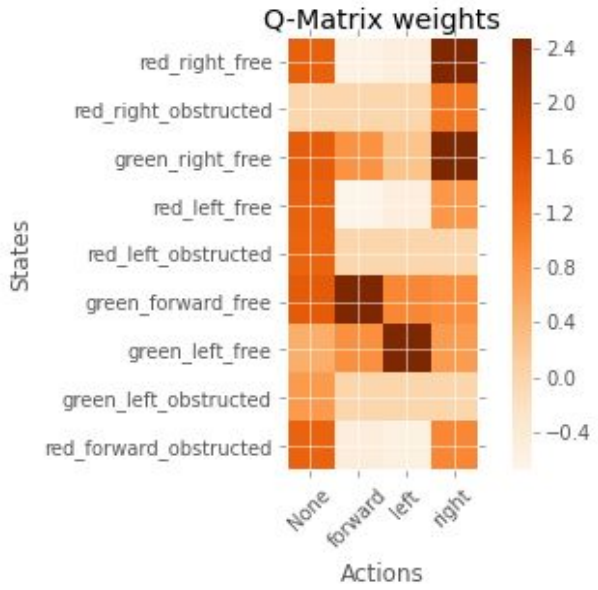
- **Learning rate was fixed at 0.5**
- **Discount factor was fixed at 0.2**
- Epsilon = 1/current runs
- Trials = 100
- Enforcing deadline = True

The following plots are presented to help compare performance of the modifications.





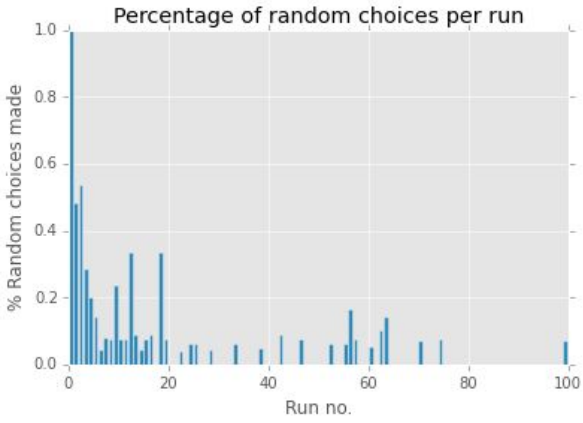


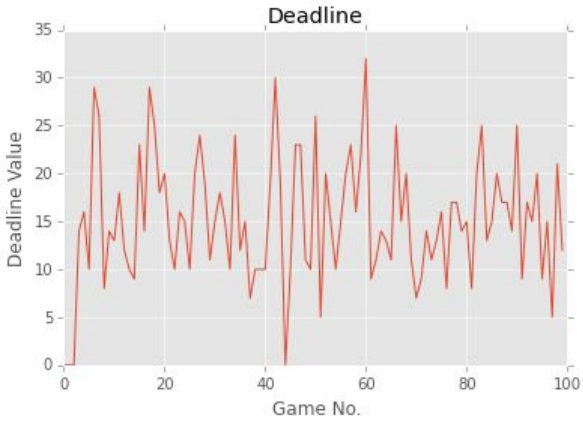


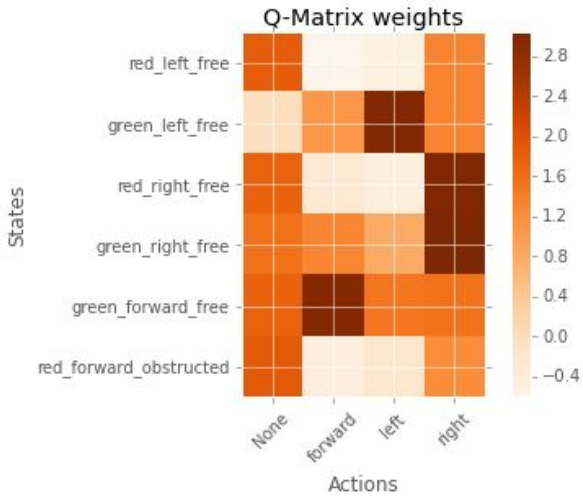
Experiment 5

- **Learning rate was fixed at 1/current runs**
- **Discount factor was fixed at 0.4**
- Epsilon = 1/current runs
- Trials = 100
- Enforcing deadline = True

The following plots are presented to help compare performance of the modifications.







Conclusion

For each experiment; learning rate and discount factor were adjusted and performance measured against each other (comparing average reward, deadline, and the quality matrix).

Learning rate determines how quickly the agent learns from each move i.e. if 0 then the agent ignore any feedback, whereas 1 would mean the agent replaces existing knowledge with what was just learnt, and 0.5 would mean it would find a mean between what it already knows and has just learnt.
Discount factor determines how much weight is given to the expected future reward, i.e. if 0 then any future reward is ignored, while a value of 1 would mean that it is given a high weight and therefore will heavily influence the quality value.

Experiments 2 and 4 achieves the best results based on their resulting deadline value at the end each run.

Experiment 2 variables:

- Learning rate was fixed at 0.8
- Discount factor was fixed at 0.5

Experiment 4 variables:

- Learning rate was fixed at 0.5
- Discount factor was fixed at 0.2

Both of these decreased the emphasis on future expected reward and, therefore, increasing emphasis on the immediate reward (current turn). Consequently this is what you would want for such a simulation (/model) as incorrect decisions could lead to devastating results.

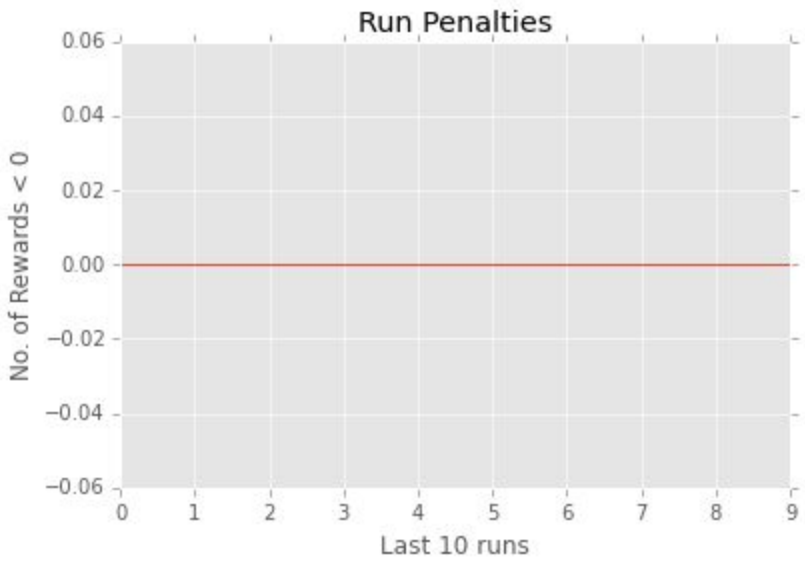
My *thinking* to why this works; the discount factor influences how much weight you give to future rewards, as mentioned above, a factor of 0 would mean our strategy is short-sighted, only relying on immediate rewards, while a factor of 0.9 would mean we put more emphasis on future rewards. If the environment was deterministic i.e. you would expect the same reward each time for subsequent states then you would set this close to 1.0. In our case, the next state is deterministic but environment (beyond that) the environment is stochastic therefore our strategy needs to be short sighted and focus on immediate rewards.

It can also be observed that the agents learn't the road rules illustrated in the q-matrix colourbar - the darker colours represent the stronger weights given to an action for each state. This shows that the agent moves in the direction of the waypoint whilst adhering to the road rules.

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?

Based on the context of having to navigate a vehicle from point A to point B; I would define the optimal policy one that is most safe i.e. adhere to the road rules, and secondly navigating to the destination via the shortest distance (as opposed to fastest route - maybe to meet emission goals). Obviously this is guided by the rewards received by the environment but assuming some high-level domain expert would set these.

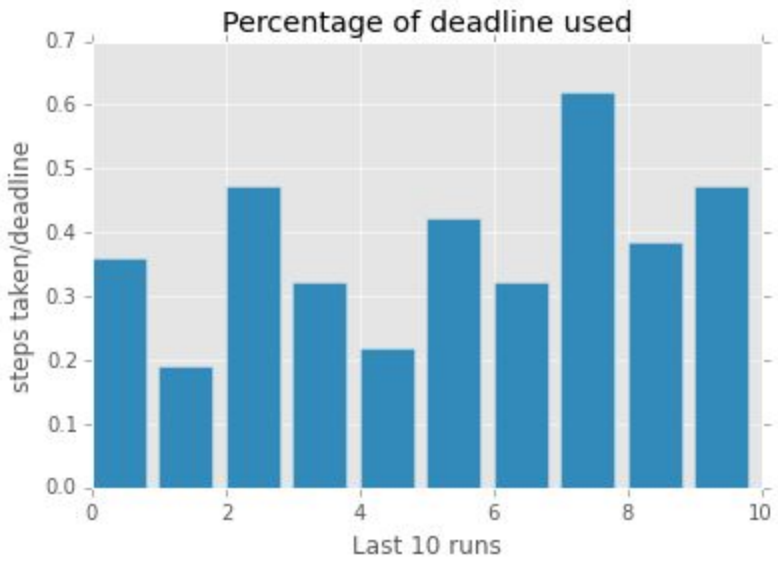
Using this as a definition for the optimal policy I would argue that the agent achieves this. The following plot shows that no penalties (rewards < 0) were received during the last 10 runs.



This illustrates that the agent adhered to the road rules, and therefore can be considered safe.

In terms of taking the shortest route; the plot below is shows the rewards received for each move during the last run, it shows that no reward < 1 was received therefore can be concluded that the agent followed the shortest path based on the directions provided by the environment/planner.

Finally; the last plot helps reinforce (;)) this conclusion by showing the percentage of moves used from the allocated moves available. On average it reaches the destination using less than 50% of the allocated moves.


Percentage of deadline used
steps taken/deadline
Last 10 runs
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2
4
6
8
10