Q-Learning-Copy1

May 15, 2019

In [1]:
```
import gym
env = gym.make("Taxi-v2").env

env.render()
```

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

In [2]:
```
env.reset() # reset environment to a new, random state
env.render()

print("Action Space {}".format(env.action_space))
print("State Space {}".format(env.observation_space))
```

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

Action Space Discrete(6)
State Space Discrete(500)
```

In [3]:
```
state = env.encode(2, 1, 2, 0) # (taxi row, taxi column, passenger index, destination
print("State:", state)

env.s = state # corresponding state value
env.render()
```

```
State: 228
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
In [4]: env.P[228]  # initial reward table (default)
```

```
Out[4]: {0: [(1.0, 328, -1, False)],
         1: [(1.0, 128, -1, False)],
         2: [(1.0, 248, -1, False)],
         3: [(1.0, 208, -1, False)],
         4: [(1.0, 228, -10, False)],
         5: [(1.0, 228, -10, False)]}
```

```
In [6]: #env.P[28]
```

```
In [5]: env.s = 328  # set environment to illustration's state

        epochs = 0
        penalties, reward = 0, 0

        frames = [] # for animation

        done = False

        while not done:
            action = env.action_space.sample()
            state, reward, done, info = env.step(action)

            if reward == -10:
                penalties += 1

            # Put each rendered frame into dict for animation
            frames.append({
                'frame': env.render(mode='ansi'),
                'state': state,
                'action': action,
                'reward': reward
                }
            )

            epochs += 1
```

```
        print("Timesteps taken: {}".format(epochs))
        print("Penalties incurred: {}".format(penalties))
```

```
Timesteps taken: 1575
Penalties incurred: 565
```

```
In [6]: from IPython.display import clear_output
        from time import sleep

        def print_frames(frames):
            for i, frame in enumerate(frames):
                clear_output(wait=True)
                #print(frame['frame'].getvalue( ))
                fm = frame['frame']
                print(fm)
                print(f"Timestep: {i + 1}")
                print(f"State: {frame['state']}")
                print(f"Action: {frame['action']}")
                print(f"Reward: {frame['reward']}")
                sleep(.1)


        print_frames(frames)
```

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Timestep: 1575
State: 0
Action: 5
Reward: 20
```

1 in the above snippet, oue agent takes almost ----- step and makes lots of wrong dop off by getting many penalties to deliver 1 person to the dest.

2 RL-Q

In [7]:
```
import numpy as np
q_table = np.zeros([env.observation_space.n, env.action_space.n])
print (q_table)
```

```
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 ...
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
```

In [9]:
```
%%time
"""Training the agent"""

import random
from IPython.display import clear_output

# Hyperparameters
alpha = 0.1
gamma = 0.6
epsilon = 0.1

# For plotting metrics
all_epochs = []
all_penalties = []

for i in range(1, 100001):
    state = env.reset()

    epochs, penalties, reward, = 0, 0, 0
    done = False

    while not done:
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample() # Explore action space
        else:
            action = np.argmax(q_table[state]) # Exploit learned values

        next_state, reward, done, info = env.step(action)
```

```
                old_value = q_table[state, action]
                next_max = np.max(q_table[next_state])

                new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
                q_table[state, action] = new_value

                if reward == -10:
                    penalties += 1

                state = next_state
                epochs += 1

            if i % 100 == 0:
                clear_output(wait=True)
                print(f"Episode: {i}")

        print("Training finished.\n")
```

```
Episode: 100000
Training finished.

Wall time: 1min 11s
```

3 Now that the Q-table has been established over 100,000 episodes, let's see what the Q-values are at our illustration's state:

```
In [10]: q_table[328]
```

```
Out[10]: array([ -2.41737279,  -2.27325184,  -2.41719425,  -2.36368694,
                -11.25015806, -11.14915607])
```

4 Evaluating the agent

```
In [11]: """Evaluate agent's performance after Q-learning"""

        total_epochs, total_penalties = 0, 0
        episodes = 100

        for _ in range(episodes):
            state = env.reset()
            epochs, penalties, reward = 0, 0, 0

            done = False

            while not done:
                action = np.argmax(q_table[state])
```

```
                state, reward, done, info = env.step(action)

                if reward == -10:
                    penalties += 1

                epochs += 1

            total_penalties += penalties
            total_epochs += epochs

        print(f"Results after {episodes} episodes:")
        print(f"Average timesteps per episode: {total_epochs / episodes}")
        print(f"Average penalties per episode: {total_penalties / episodes}")
```

```
Results after 100 episodes:
Average timesteps per episode: 12.75
Average penalties per episode: 0.0
```

```
In [1]: import gym
```

```
In [ ]:
```