Task 1.2C

GitHub: https://github.com/applecrumble123/Reinforcement_Learning/tree/master/smart-cab

Policy is referred to as the mapping of the agent's perceived state to the best action at a given timestamp. It is the agent's attempt to maximize the total amount of reward over the long run by a means of exploration.

For this experiment, Q-learning is used by the agent to make the optimal action best on the given state. Q-learning is a model-free RL algorithm where the agent will know the state in a given environment. It uses an off-policy Reinforcement Learning where the optimal policy is independent regardless of the agent's motivation and it learns the optimal policy with the help of a greedy policy. Therefore, there is a balance between exploration and exploitation to get the optimal policy.

There is only 1 action-value function for Q-learning as the agent will only consider the max Q-value for a given action, leading to the next state. The agent will take an action while in the current state, giving it a reward and sending it to the next state, then update the Q-value. Since the next state, which is usually selected based on the concept of transitional probability, is given by the environment and where the agent will end up, the Q-learning transitional probability is equals to 1, as seen in Appendix 1.

Each Q-value represents the quality of an action taken in the perceived state and a higher value imply that the chances of getting greater rewards are higher. The Q-values are initialized randomly. The agent will then receive different rewards when being exposed to different environment and the Q-values will be updated using the following equation:

$$Q(state, action) \leftarrow (1-\alpha)[Q(state, action)] + \alpha[reward + \gamma Max_a Q(next\ state,\ all\ actions)]$$

α (alpha) refers to the learning rate for which $0 \leq \alpha \leq 1$. It controls the extent to which the Q-values are updated every iteration. γ (gamma) refers to the discount factor for which $0 \leq \gamma \leq 1$. It determines the how much importance is given to future rewards. A high value would mean the agent would consider it as a long-term effective reward while a low value meant an immediate reward.

This equation shows that the Q-value is updated ($\leftarrow$) by using a weight $(1-\alpha)$ of the old Q-value and then adding the learned value, which takes into consideration the reward after taking the current action in the current state and, the discounted maximum reward from the next state after the current action is taken. This means the agent can make the optimal action after considering the reward for the current state and action combination and the max future reward for the next state for a given action.

The Q-values are stored in the Q-table, which would be initialized to 0.

The process goes as:

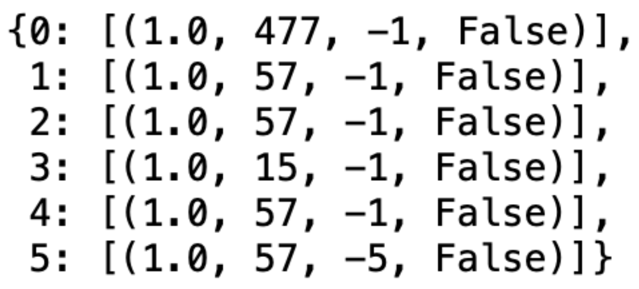
1. One action is selected from all states with their possible actions in the current state.
2. The agent will move on to the next state as a result of the selected action and the highest Q-value is obtained from all possible actions in that given state.
3. The Q-table is then updated using the equation above and then set the next state as the current state.
4. If the goal is reached, repeat the process again.

The epsilon, $\in$, in the range of [-0, 1], is introduced to prevent overfitting. If the number is less than epsilon, the agent will explore the agent space, else, it will exploit to the Q-table to make the optimal decision. A lower value of epsilon will result in more penalties as the agent is exploring and making random decisions. Therefore, the agent will be able to balance between exploration and exploitation.

As seen in Appendix 2, when reinforcement learning is not enforced, the agent will take 13754 steps to drop the passenger at the correct location and it incurred 2230 penalties. However, with Reinforcement Learning, as seen in Appendix 3, there are 0 penalties with 21 steps taken.

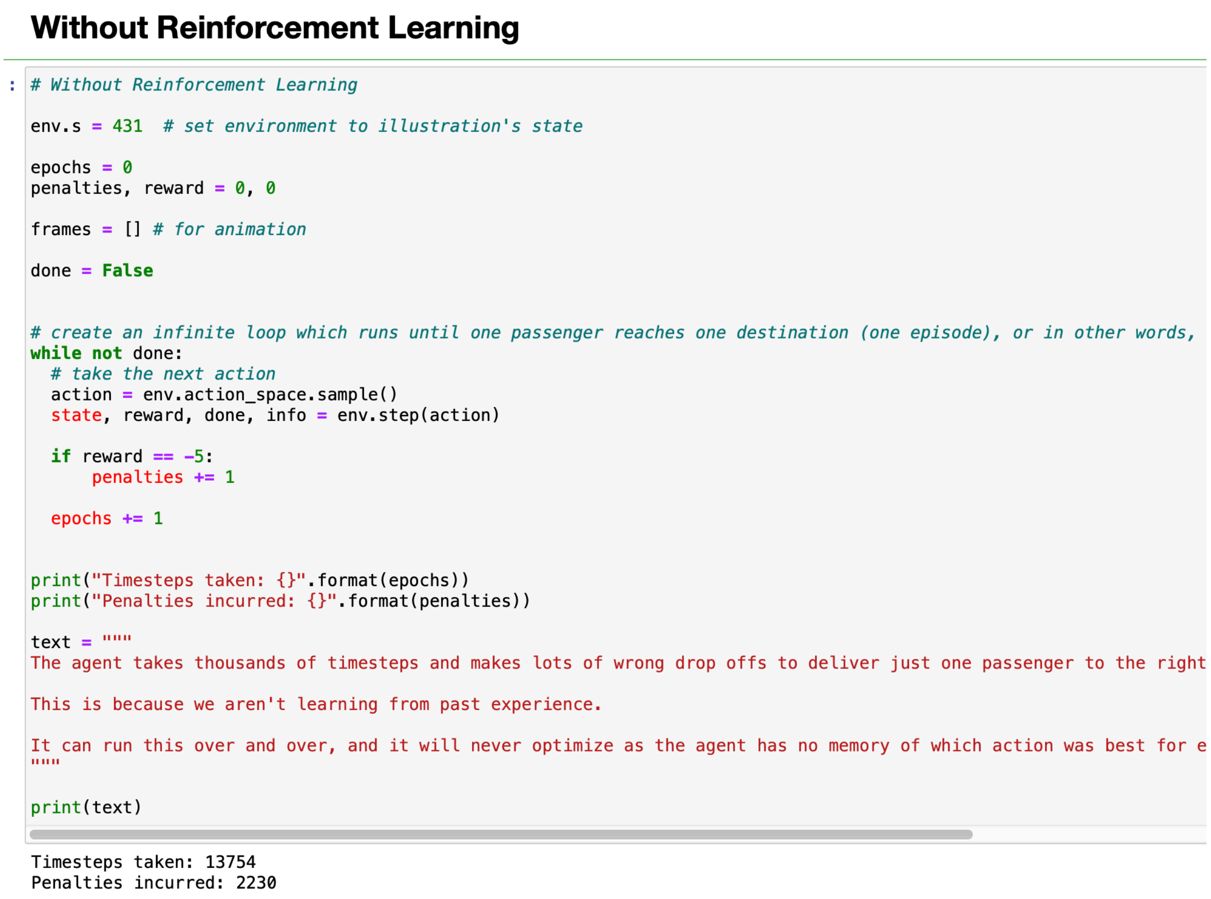
Appendix

Appendix 1: Reward table given an environment



```
{0: [(1.0, 477, -1, False)],
 1: [(1.0, 57, -1, False)],
 2: [(1.0, 57, -1, False)],
 3: [(1.0, 15, -1, False)],
 4: [(1.0, 57, -1, False)],
 5: [(1.0, 57, -5, False)]}
```

Appendix 2: Without Reinforcement Learning



Without Reinforcement Learning

```
: # Without Reinforcement Learning

  env.s = 431  # set environment to illustration's state

  epochs = 0
  penalties, reward = 0, 0

  frames = [] # for animation

  done = False


  # create an infinite loop which runs until one passenger reaches one destination (one episode), or in other words,
  while not done:
    # take the next action
    action = env.action_space.sample()
    state, reward, done, info = env.step(action)

    if reward == -5:
        penalties += 1

    epochs += 1


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

  text = """
  The agent takes thousands of timesteps and makes lots of wrong drop offs to deliver just one passenger to the right

  This is because we aren't learning from past experience.

  It can run this over and over, and it will never optimize as the agent has no memory of which action was best for e
  """

  print(text)
```

```
Timesteps taken: 13754
Penalties incurred: 2230
```

Appendix 3: With Reinforcement Learning

```python
"""Evaluate agent's performance after Q-learning"""


# Resets the environment and returns a random initial state
state = env.reset()
epochs, penalties, reward = 0, 0, 0

done = False

while not done:
    # use the current q table with its current state to do the next action
    action = np.argmax(q_table[state])
    state, reward, done, info = env.step(action)

    if reward == -5:
        penalties += 1

    epochs += 1

print("Time steps to drop off passenger: {}".format(epochs))

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

```
Time steps to drop off passenger: 21
Penalties incurred: 0
```

Reference

Medium. 2021. *The Complete Reinforcement Learning Dictionary*. [online] Available at: <https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e> [Accessed 17 March 2021].

Sagar, R. 2021. *On-Policy VS Off-Policy Reinforcement Learning: The Differences*. [online] Analytics India Magazine. Available at: <https://analyticsindiamag.com/reinforcement-learning-policy/> [Accessed 17 March 2021].

Learndatasci.com. 2021. *Reinforcement Q-Learning from Scratch in Python with OpenAI Gym*. [online] Available at: <https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/> [Accessed 17 March 2021].