

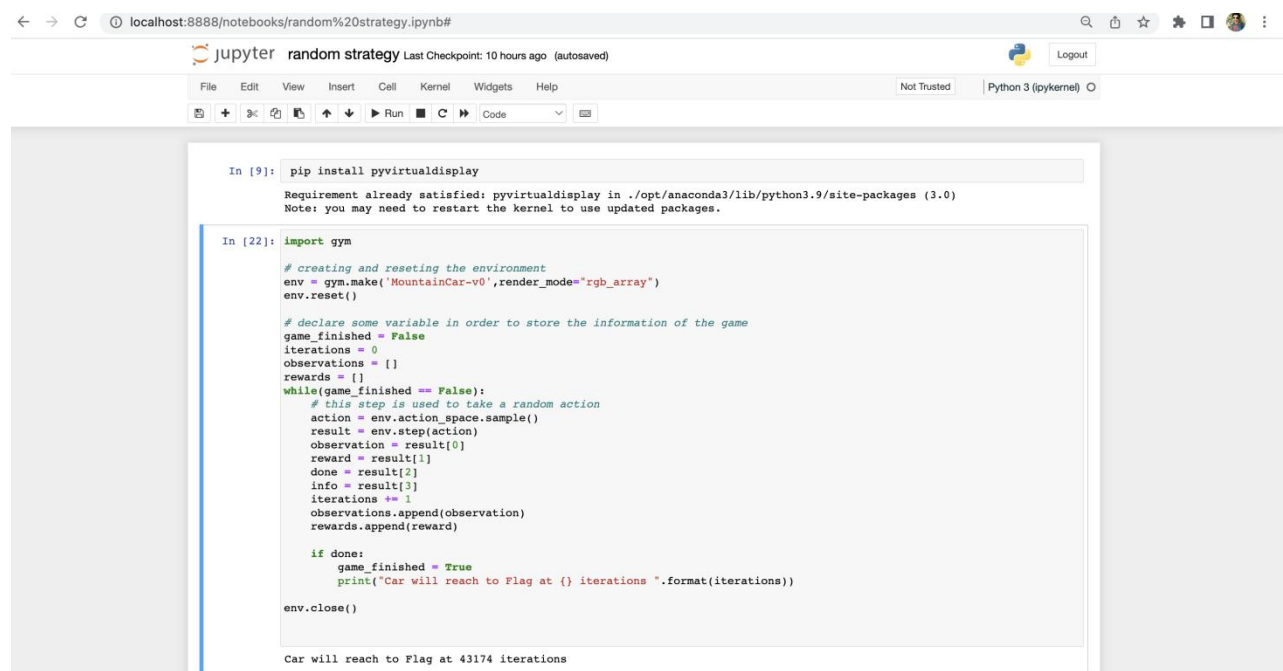
Name : CHO VATIYA JAY MAHESHBHAI
STUDENT ID : 1161980

Mountain Car Problem:

There are two techniques that I have used to solve the mountain car problem. The first and foremost one is random strategy. To elaborate, this method is effective, but it has also the drawback that data collected from one condition is not utilized to assess another. Apart from it, it could require traveling to every state, which might take long of training time.

In the random strategy i have created the environment by using “gym.make()”. Afterwards,reset the environment for the base state and then i have declare some variable in order to store the information.The primary method is step(), and it allows you to perform an action in the environment.This method returns the reward and the state at interval+1 as well as a value called done, which is i have mentioned in the program game_finished=True if the car reaches the target. It will print episodes completed after the 43174 steps respectively.The approach incorporates implementations of the previously determined parameters.

Outputs : I have done this program by using jupyter and attached the file in folder by the name of car_randomstrategy.ipynb .



```
In [9]: pip install pyvirtualdisplay
Requirement already satisfied: pyvirtualdisplay in ./opt/anaconda3/lib/python3.9/site-packages (3.0)
Note: you may need to restart the kernel to use updated packages.

In [22]: import gym
# creating and resetting the environment
env = gym.make('MountainCar-v0',render_mode="rgb_array")
env.reset()

# declare some variable in order to store the information of the game
game_finished = False
iterations = 0
observations = []
rewards = []
while(game_finished == False):
    # this step is used to take a random action
    action = env.action_space.sample()
    result = env.step(action)
    observation = result[0]
    reward = result[1]
    done = result[2]
    info = result[3]
    iterations += 1
    observations.append(observation)
    rewards.append(reward)

    if done:
        game_finished = True
        print("Car will reach to Flag at {} iterations ".format(iterations))

env.close()

Car will reach to Flag at 43174 iterations
```

Notes : This code may not function properly in some local setups. Because It does use some Jupyter Notebook-specific commands, It has been tried and is reliable on Google Colab. Here, i have tested and run the program by using google colab in which i tried to implement the same random strategy but use the graphics to show how car is reached at flag and attached the file by the name of problem_1.ipynb.

Stage 1: automobile car is about to start taking motion towards flag.

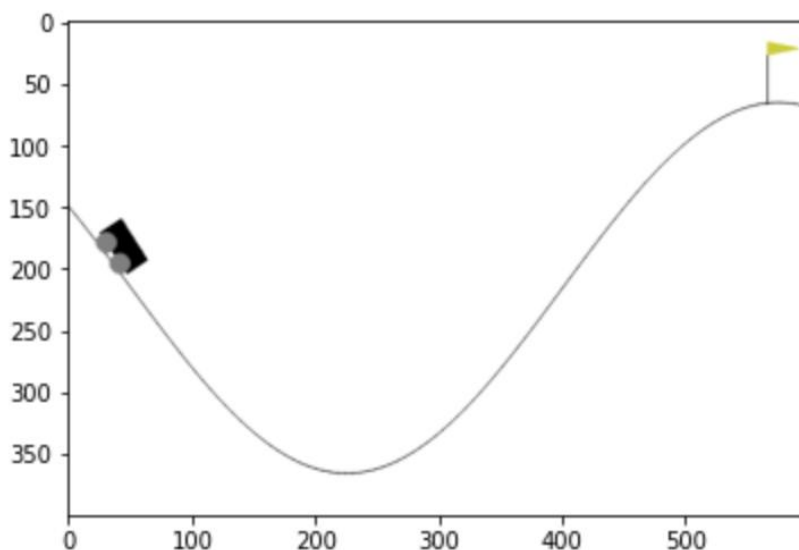
```
from IPython import display as ipythondisplay

from pyvirtualdisplay import Display
display = Display(visible=0, size=(400, 300))
display.start()

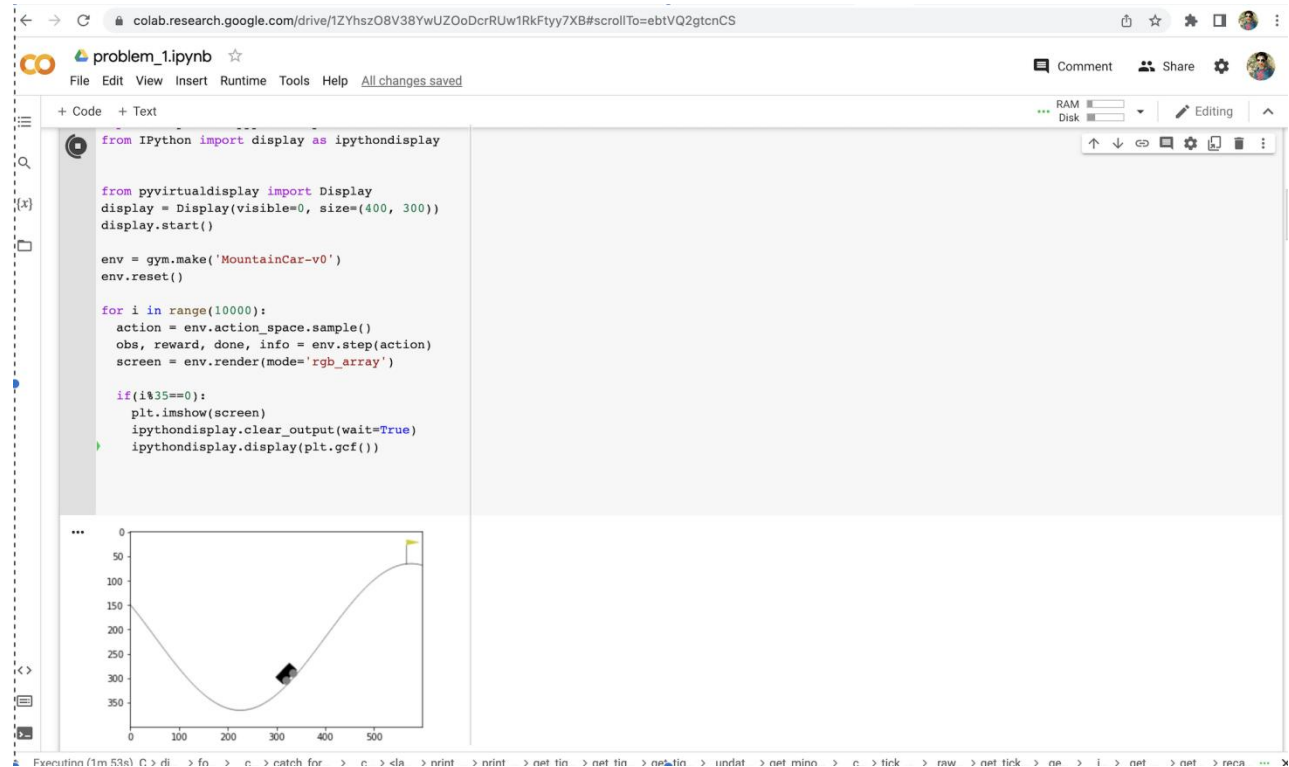
env = gym.make('MountainCar-v0')
env.reset()

for i in range(10000):
    action = env.action_space.sample()
    obs, reward, done, info = env.step(action)
    screen = env.render(mode='rgb_array')

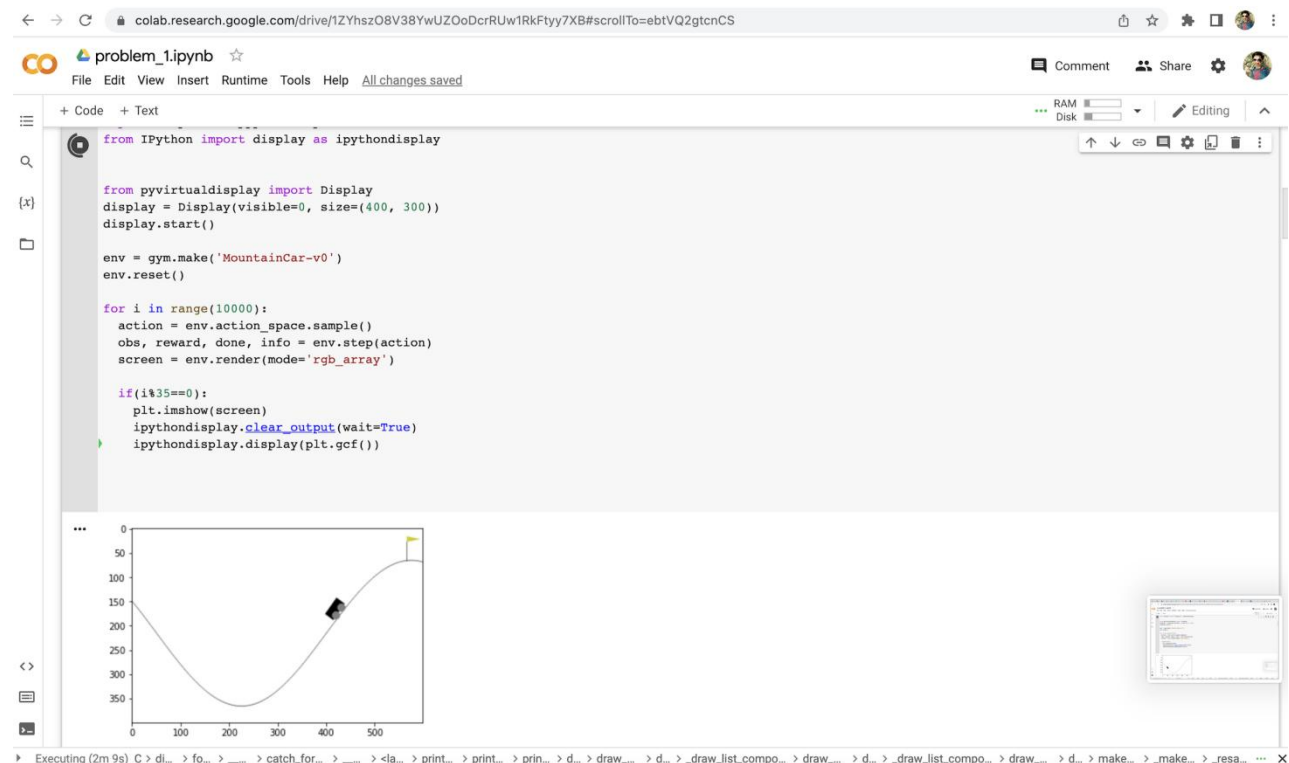
    if(i%35==0):
        plt.imshow(screen)
        ipythondisplay.clear_output(wait=True)
        ipythondisplay.display(plt.gcf())
```



Stage 2:

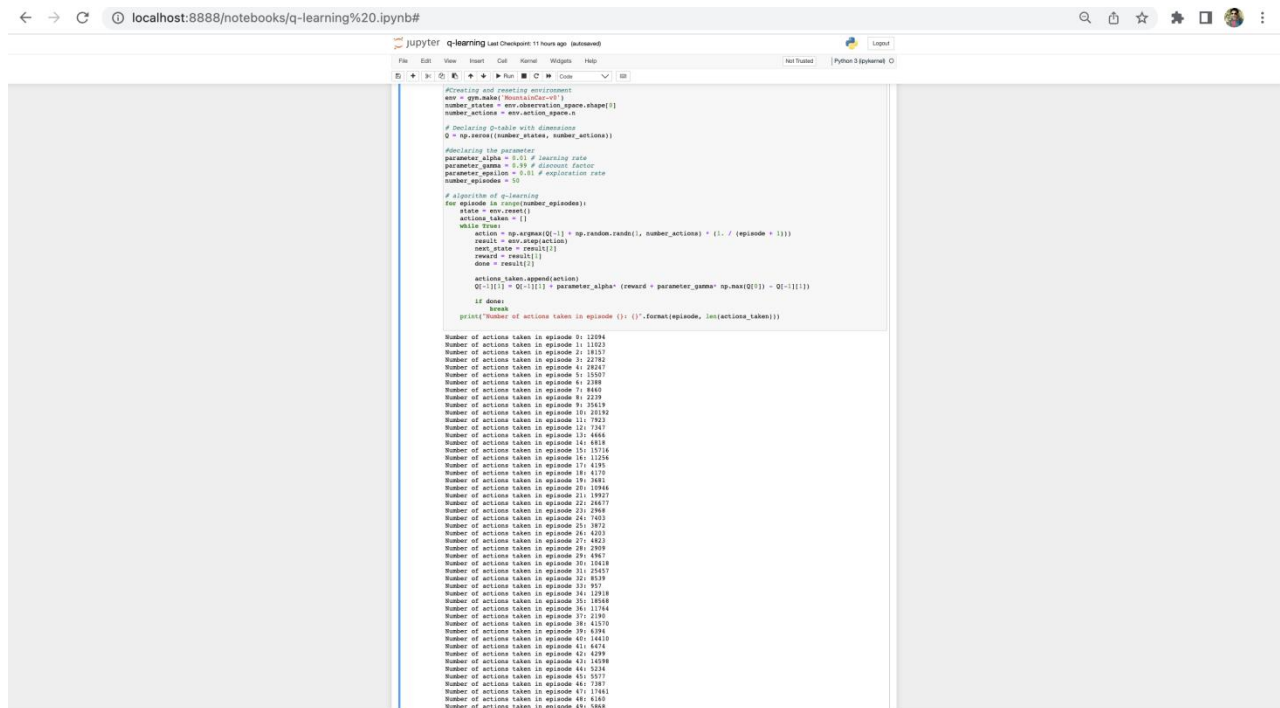


Stage 3: When car is about to reached at flag.



Q-learning: In this algorithm learning rate denotes alpha how much the previous steps are factored into the calculation. It strives to include current state values of the algorithm. apart from it, the learning rate is zero here so the value of q is not changed and multiply it by factor 1. Because of number of states, it slows down the process when applying Q-learning to programming.

Outputs :



The screenshot shows a Jupyter Notebook interface with a code editor on the left and an output area on the right. The code defines a Gym environment, initializes Q-table, and runs a Q-learning algorithm for 50 episodes. The output area displays the number of actions taken in each episode, showing a decreasing trend over time.

```
#Creating and resetting environment
env = gym.make('MountainCar-v0')
number_states = env.observation_space.shape[0]
number_actions = env.action_space.n

# Initializing Q-table with dimensions
Q = np.zeros((number_states, number_actions))

#Declaring the parameter
parameter_alpha = 0.05 # learning rate
parameter_gamma = 0.99 # discount factor
parameter_epsilon = 0.05 # exploration rate
number_episodes = 50

# Algorithm of Q-learning
for episode in range(number_episodes):
    state = env.reset()
    actions_taken = []
    while True:
        action = np.argmax(Q[state, :] - np.random.randn(1, number_actions) * (1. / (episode + 1)))
        result, new_state, done, _ = env.step(action)
        next_state = result[0]
        reward = result[1]
        done = result[2]
        data = result[3]
        actions_taken.append(action)
        Q[state][action] = Q[state][action] + parameter_alpha * (reward + parameter_gamma * np.max(Q[next_state, :]) - Q[state][action])
        state = next_state
        if done:
            break
    print("Number of actions taken in episode {}: {}".format(episode, len(actions_taken)))
```

Number of actions taken in episode 0: 12084
 Number of actions taken in episode 1: 11023
 Number of actions taken in episode 2: 2837
 Number of actions taken in episode 3: 22792
 Number of actions taken in episode 4: 2924
 Number of actions taken in episode 5: 15507
 Number of actions taken in episode 6: 3389
 Number of actions taken in episode 7: 8460
 Number of actions taken in episode 8: 5239
 Number of actions taken in episode 9: 7049
 Number of actions taken in episode 10: 25192
 Number of actions taken in episode 11: 7923
 Number of actions taken in episode 12: 7347
 Number of actions taken in episode 13: 4466
 Number of actions taken in episode 14: 6816
 Number of actions taken in episode 15: 15716
 Number of actions taken in episode 16: 11226
 Number of actions taken in episode 17: 4195
 Number of actions taken in episode 18: 4179
 Number of actions taken in episode 19: 2681
 Number of actions taken in episode 20: 10946
 Number of actions taken in episode 21: 14937
 Number of actions taken in episode 22: 24677
 Number of actions taken in episode 23: 2868
 Number of actions taken in episode 24: 7403
 Number of actions taken in episode 25: 1852
 Number of actions taken in episode 26: 4203
 Number of actions taken in episode 27: 4423
 Number of actions taken in episode 28: 2809
 Number of actions taken in episode 29: 4967
 Number of actions taken in episode 30: 13418
 Number of actions taken in episode 31: 25457
 Number of actions taken in episode 32: 8539
 Number of actions taken in episode 33: 957
 Number of actions taken in episode 34: 12018
 Number of actions taken in episode 35: 10168
 Number of actions taken in episode 36: 11784
 Number of actions taken in episode 37: 2130
 Number of actions taken in episode 38: 4270
 Number of actions taken in episode 39: 6394
 Number of actions taken in episode 40: 14410
 Number of actions taken in episode 41: 4474
 Number of actions taken in episode 42: 14598
 Number of actions taken in episode 43: 14598
 Number of actions taken in episode 44: 5234
 Number of actions taken in episode 45: 5577
 Number of actions taken in episode 46: 7287
 Number of actions taken in episode 47: 17461
 Number of actions taken in episode 48: 6160
 Number of actions taken in episode 49: 5868

Overall, i recapitulated that Q-learning is better techniques compared to the random strategy. Because it determines the best course of action and randomly selects this action based on the current state with the intention of giving the maximum reward. Whereas, on the other hand, it makes decisions based on randomness without learning or optimizing. It won't become more effective after some iteration.