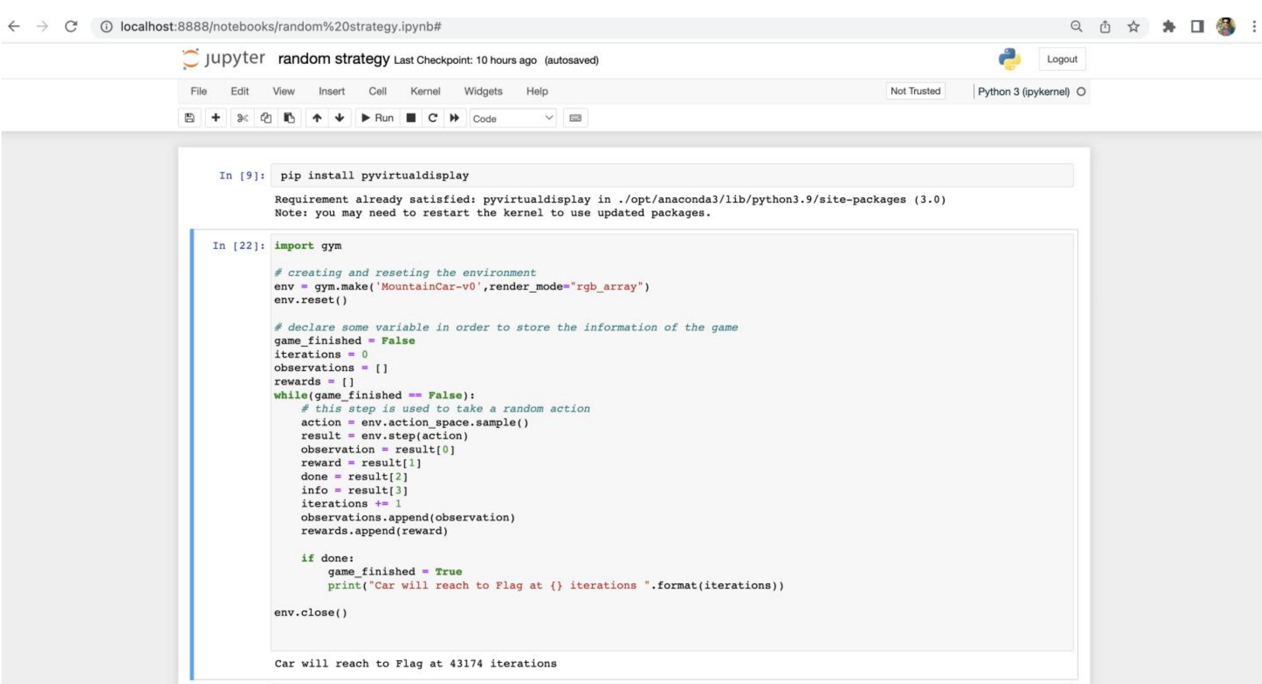
Name : CHOVATIYA JAY MAHESHBHAI
STUDENT ID : 1161980

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.

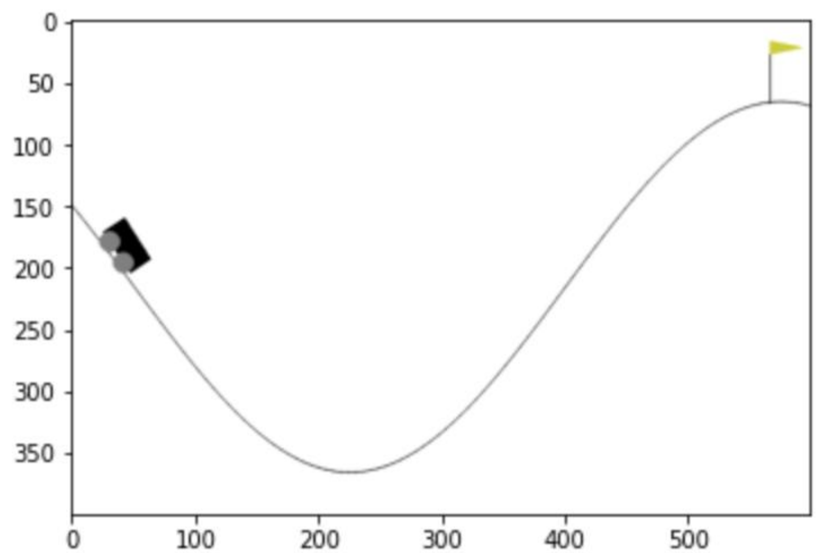
Outputs : I have done this program by using jupyter and attached the file in folder by the name of car_randomstrategy.ipynb .



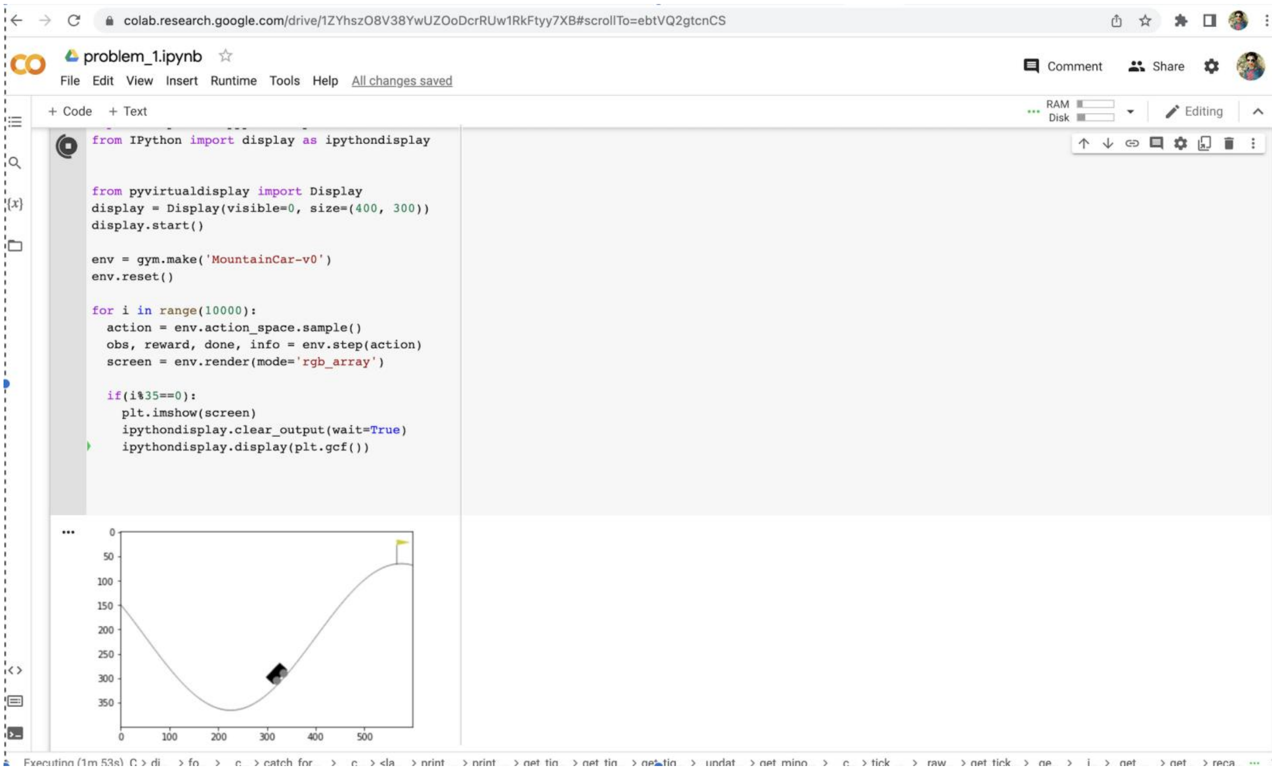
<u>Notes</u> : This code may not function properly in some local setups. Because It does use some Jupyter Notebook-specific commands, It has been tried and is reliable on Google Colab.Here, i have tested and run the program by using google colab in which i tried to implement the same random strategy 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.

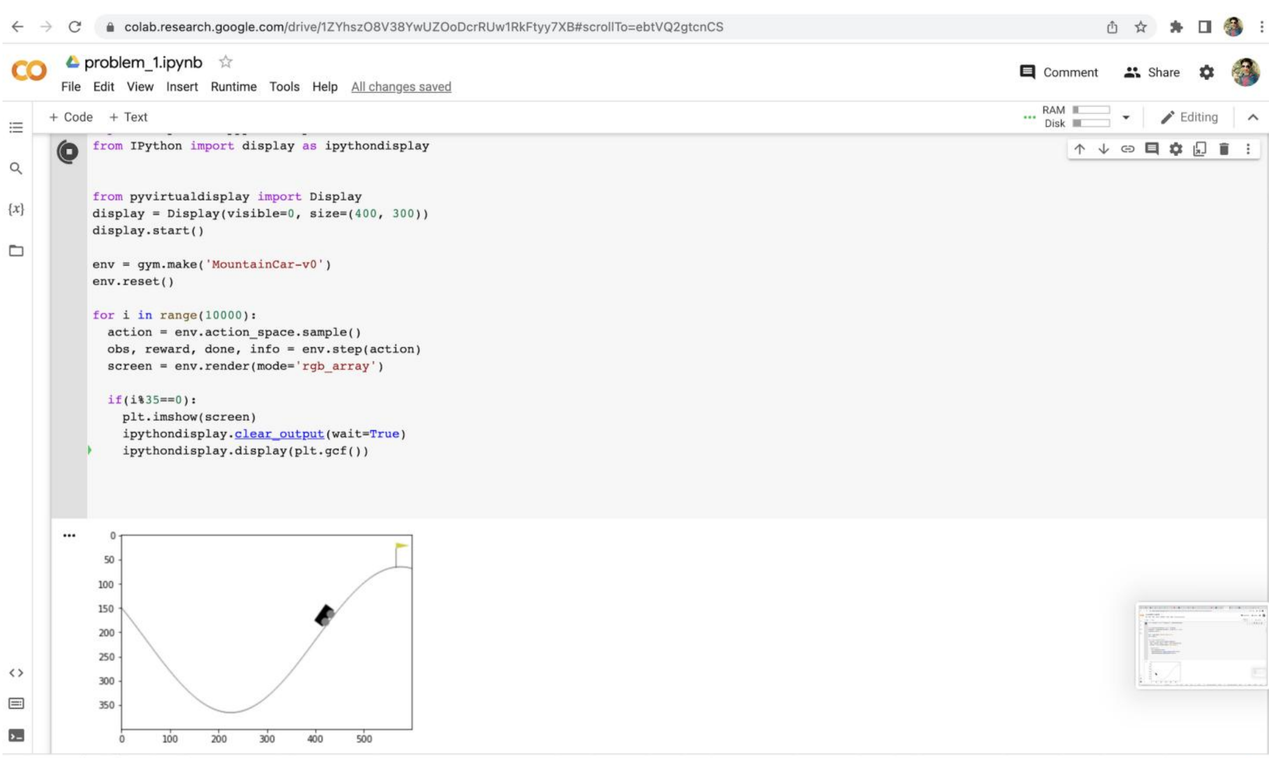
```
from IPython import display as ipythondisplay


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:

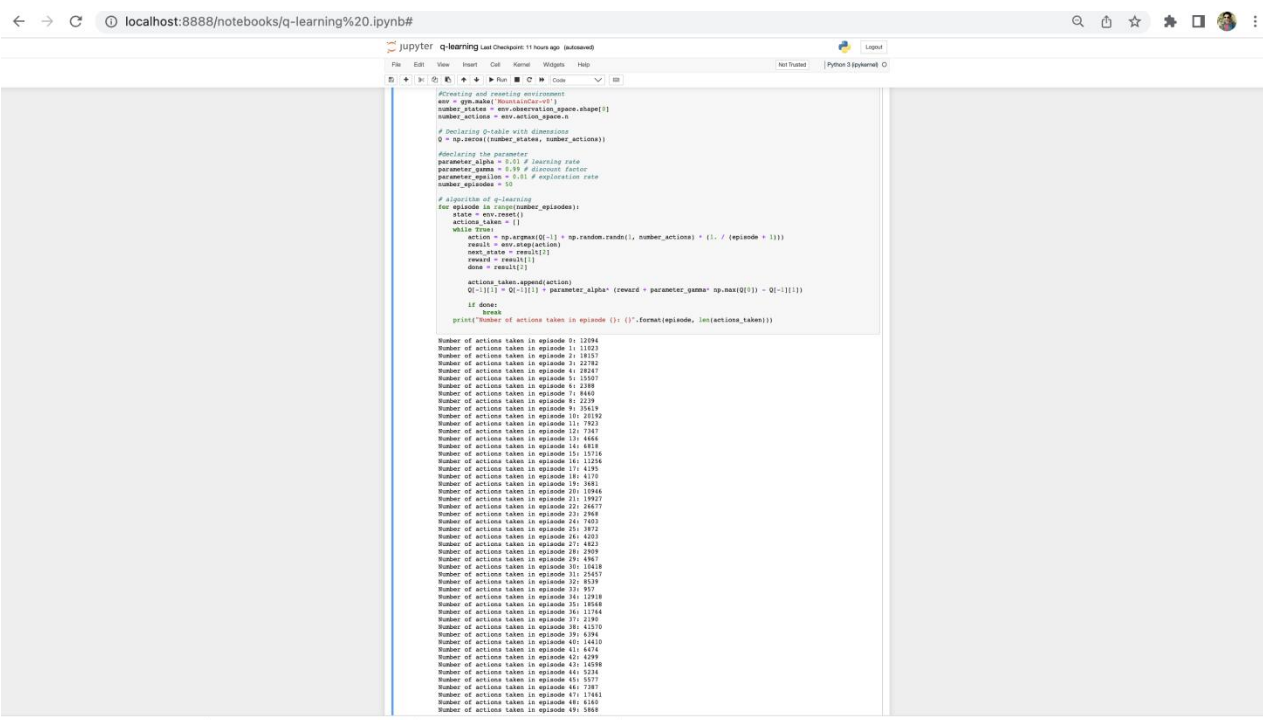


Stage 3: When car is about to reached at flag.



<u>Q-learning:</u> 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.

<u>Outputs :</u>



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.