Install and import necessary packages

```
!pip install gym
!apt-get install python-opengl -y
!apt install xvfb -y

!pip install gym[atari]

!pip install pyvirtualdisplay
!pip install piglet
```

```
Requirement already satisfied: gym in /usr/local/lib/python3.10/dist-packages (0.25.2)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from gym) (1.23.5)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gym) (2.2.1)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from gym) (0.0.8)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package python-opengl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
xvfb is already the newest version (2:21.1.4-2ubuntu1.7~22.04.8).
0 upgraded, 0 newly installed, 0 to remove and 32 not upgraded.
Requirement already satisfied: gym[atari] in /usr/local/lib/python3.10/dist-packages (0.25.2)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from gym[atari]) (1.23.5)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gym[atari]) (2.2.1)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from gym[atari]) (0.0.8)
Requirement already satisfied: ale-py~=0.7.5 in /usr/local/lib/python3.10/dist-packages (from gym[atari]) (0.7.5)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.10/dist-packages (from ale-py~=0.7.5->gym[atari]) (6.1.1)
Requirement already satisfied: pyvirtualdisplay in /usr/local/lib/python3.10/dist-packages (3.0)
Requirement already satisfied: piglet in /usr/local/lib/python3.10/dist-packages (1.0.0)
Requirement already satisfied: piglet-templates in /usr/local/lib/python3.10/dist-packages (from piglet) (1.3.0)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from piglet-templates->piglet) (3.1.1)
Requirement already satisfied: attrs in /usr/local/lib/python3.10/dist-packages (from piglet-templates->piglet) (23.2.0)
Requirement already satisfied: astunparse in /usr/local/lib/python3.10/dist-packages (from piglet-templates->piglet) (1.6.3)
Requirement already satisfied: markupsafe in /usr/local/lib/python3.10/dist-packages (from piglet-templates->piglet) (2.1.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse->piglet-templates->piglet) (0.42.0)
Requirement already satisfied: six<2.0,>=1.6.1 in /usr/local/lib/python3.10/dist-packages (from astunparse->piglet-templates->piglet) (1.16.0)
```

To activate virtual display we need to run a script once for training an agent, as follows:

```
from pyvirtualdisplay import Display
display = Display(visible=0, size=(1400, 900))
display.start()
```

```
<pyvirtualdisplay.display.Display at 0x786e987ecd90>
```

```
# This code creates a virtual display to draw game images on.
# If you are running locally, just ignore it
import os
if type(os.environ.get("DISPLAY")) is not str or len(os.environ.get("DISPLAY"))==0:
    !bash ../xvfb start
    %env DISPLAY=:1
```

```
import gym
from gym import logger as gymlogger
from gym.wrappers.record_video import RecordVideo
gymlogger.set_level(40) # error only
import tensorflow as tf
import numpy as np
import random
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import math
import glob
import io
import base64
from IPython.display import HTML

from IPython import display as ipythondisplay

"""
Utility functions to enable video recording of gym environment and displaying it
To enable video, just do "env = wrap_env(env)""
"""

def show_video():
  mp4list = glob.glob('video/*.mp4')
  if len(mp4list) > 0:
    mp4 = mp4list[0]
    video = io.open(mp4, 'r+b').read()
    encoded = base64.b64encode(video)
    ipythondisplay.display(HTML(data='''<video alt="test" autoplay
                loop controls style="height: 400px;">
                <source src="data:video/mp4;base64,{0}" type="video/mp4" />
             </video>'''.format(encoded.decode('ascii'))))
  else:
    print("Could not find video")


def wrap_env(env):
  env = RecordVideo(env, './video')
  return env
```

OpenAI Gym

OpenAI gym is a python library that wraps many classical decision problems including robot control, videogames and board games. We will use the environments it provides to test our algorithms on interesting decision problems.

Gym documentation: https://www.gymlibrary.dev/content/basic_usage/#

Refer to the docstrings of github code to understand the attributes of the environment.

https://github.com/openai/gym/blob/dcd185843a62953e27c2d54dc8c2d647d604b635/gym/envs/classic_control/mountain_car.py#L18C1-L18C20

Environment object

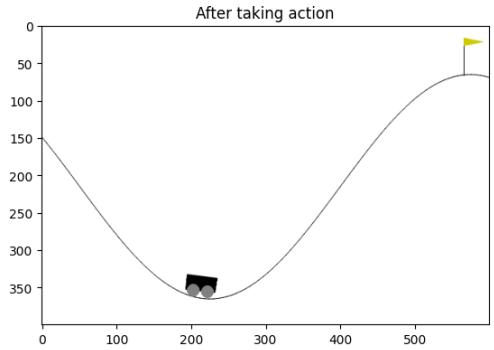
- env.observation_space : state space, all possible states.
- env.action_space : all possible actions the agent can take.

- env.state : Current state the agent is in.
- env.reset() : reset environment to initial state, return first observation
- env.render(): show current state.
- env.step(action) : commit action a and return (new observation, reward, is done, info)

MountainCar

```
# env = gym.make('CartPole-v0')
env = gym.make('MountainCar-v0')

# env = wrap_env(env)

print('observation space:', env.observation_space)
print('action space:', env.action_space)

obs = env.reset()

print('initial observation:', obs)

action = env.action_space.sample()  # take a random action
print("action: ", action)

obs, r, done, info = env.step(action)
print('\nnext observation:', obs)
print('reward:', r)
print('done:', done)
print('info:', info)
plt.title("After taking action")
plt.imshow(env.render('rgb_array'))
```

```
observation space: Box([-1.2  -0.07], [0.6  0.07], (2,), float32)
action space: Discrete(3)
initial observation: [-0.56240416  0.        ]
action:  2

next observation: [-0.5611138   0.00129038]
reward: -1.0
done: False
info: {}
<matplotlib.image.AxesImage at 0x797e3b7baa10>
```



Random action

```
'''CartPole problem use random action'''
# env = gym.make('CartPole-v0')
env = gym.make('MountainCar-v0')
env = wrap_env(env)  # defined before for rendering online

observation = env.reset()

total_reward = 0

while True:
  env.render()

  # your agent goes here
  action = env.action_space.sample()   # take a random action
  observation, reward, done, info = env.step(action)
  # print(reward)
  total_reward+=reward

  if done:
    break;

env.close()
show_video()
print(total_reward)
```

0:00 / 0:06

```
-200.0
```

Intutive action

- Accelerate to the left when car is on the left.
- Accelerate to the right when the car is on the right.

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


def policy(env):
  if env.state[1]>0:
    action = 2
  else:
    action = 1
  return action


env = gym.make('MountainCar-v0')
env = wrap_env(env)  # defined before for rendering online

observation = env.reset()

while True:
  # env.render()

  # your agent goes here
  action = policy(env)   # take a random action
  observation, reward, done, info = env.step(action)
  # print(reward)

  if done:
    break;

env.close()
show_video()
```


📓 Select a gym environment and understand the evironment's states, actions and rewards by going through the Openai Gym's documentation and github codes.

- Explain in your own words what the env is about and what needs to be achieved. Give a one liner explaining what the observation space and action space is for the selected environment.
- Import the environment and make the agent take "random actions". Print the total reward the agent received at the end of an episode.
- Calculate the average total reward by simulating multiple episodes. (atleast 1000)
- Try modifying the actions to maximise the total reward. If the chosen environment is complex to hardcode the actions, you can just explain in your own words.

Cliff Walking

```
env = gym.make('CliffWalking-v0')
print("State space: ", env.observation_space)
print("Action space: ", env.action_space)
env.reset()
print("Current state: ",env.s)
```

```
State space:  Discrete(48)
Action space:  Discrete(4)
Current state:  36
```

Random action

```
env = gym.make('CliffWalking-v0')
env = wrap_env(env)  # defined before for rendering online

observation = env.reset()

while True:
  # your agent goes here
  action = env.action_space.sample()  # take a random action
  #print(action)
  observation, reward, done, info = env.step(action)
  # print(reward)

  if done:
    break;

env.close()
show_video()
```

Explain in your own words what the env is about and what needs to be achieved. Give a one liner explaining what the observation space and action space is for the selected environment.

The `CliffWalking-v0` environment is a standard gridworld, with a start state and a goal state. There is a cliff in between the start and goal state. If the agent falls into the cliff, it gets a reward of -100 and is sent back to the start state. The goal of the agent is to reach the goal state from the start state while avoiding the cliff.

The observation space is a discrete space of size 48, representing the 48 grid cells of the environment (a 4x12 grid). Each state corresponds to a cell in the grid. The state is represented as a single integer between 0 and 47.

The action space is a discrete space of size 4, representing the four possible actions the agent can take at each state: move up, down, right, or left. The actions are represented as integers: 0 (up), 1 (right), 2 (down), and 3 (left).

- Calculate the average total reward by simulating multiple episodes. (atleast 1000)

```
'''CliffWalking problem using random action'''
# env = gym.make('CartPole-v0')
env = gym.make('CliffWalking-v0')
env = wrap_env(env)  # defined before for rendering online

observation = env.reset()

total_reward = 0

while True:
  env.render()

  # your agent goes here
  action = env.action_space.sample()   # take a random action
  observation, reward, done, info = env.step(action)
  # print(reward)
  total_reward+=reward

  if done:
    break;

env.close()
#show_video()
print(total_reward)
```

```
-305426
```

Try modifying the actions to maximise the total reward. If the chosen environment is complex to hardcode the actions, you can just explain in your own words.

```
env = gym.make('CliffWalking-v0')
env = wrap_env(env)  # defined before for rendering online

observation = env.reset()

total_reward = 0

while True:
    env.render()

    # Check the agent's current position
    row = observation // 12
    col = observation % 12

    if row == 3 and col < 11:  # If the agent is on the bottom row and not at the goal
        action = 0  # Move up
    elif col < 11:  # If the agent is not at the goal
        action = 1  # Move right
    else:  # If the agent is at the goal
        action = 2  # Move down

    observation, reward, done, info = env.step(action)
    total_reward += reward

    if done:
        break;

env.close()
print(total_reward)
```

```
-13
```