```
# #@markdown #**Anti-Disconnect for Google Colab**
# #@markdown ## Run this to stop it from disconnecting automatically
# #@markdown  **(It will anyhow disconnect after 6 - 12 hrs for using the free version of Colab.)**
# #@markdown  *(Colab Pro users will get about 24 hrs usage time)*
# #@markdown ---

# import IPython
# js_code = '''
# function ClickConnect(){
# console.log("Working");
# document.querySelector("colab-toolbar-button#connect").click()
# }
# setInterval(ClickConnect,60000)
# '''
# display(IPython.display.Javascript(js_code))

# !pip install setuptools==65.5.0
# !pip install wheel==0.38.4

# !pip install gym_super_mario_bros==7.3.0
# !pip install nes_py stable-baselines3[extra]

import time
from glob import glob


import numpy as np
import os
from stable_baselines3.common.results_plotter import load_results, ts2xy
from stable_baselines3.common.callbacks import BaseCallback
from gym.wrappers import RecordVideo
from time import time

class SaveOnBestTrainingRewardCallback(BaseCallback):
    """
    Taken from https://stable-baselines3.readthedocs.io/en/master/guide/examples.html
    Callback for saving a model (the check is done every ``check_freq`` steps)
    based on the training reward (in practice, we recommend using ``EvalCallback``).
    :param check_freq:
    :param chk_dir: Path to the folder where the model will be saved.
      It must contains the file created by the ``Monitor`` wrapper.
    :param verbose: Verbosity level.
    """
    def __init__(self, save_freq: int, check_freq: int, chk_dir: str, verbose: int = 1):
        super(SaveOnBestTrainingRewardCallback, self).__init__(verbose)
        self.check_freq = check_freq
        self.save_freq = save_freq
        self.chk_dir = chk_dir
        self.save_path = os.path.join(chk_dir, 'models')
        self.best_mean_reward = -np.inf

    def _init_callback(self) -> None:
        # Create folder if needed
        if self.save_path is not None:
            os.makedirs(self.save_path, exist_ok=True)

    def _on_step(self) -> bool:
        if self.n_calls % self.save_freq == 0:
            if self.verbose > 0:
                print(f"Saving current model to {os.path.join(self.chk_dir, 'models')}")
            self.model.save(os.path.join(self.save_path, f'iter_{self.n_calls}'))

        if self.n_calls % self.check_freq == 0:
```

```
          # Retrieve training reward
          x, y = ts2xy(load_results(self.chk_dir), 'timesteps')
          if len(x) > 0:
              # Mean training reward over the last 100 episodes
              mean_reward = np.mean(y[-100:])
              if self.verbose > 0:
                print(f"Num timesteps: {self.num_timesteps}")
                print(f"Best mean reward: {self.best_mean_reward:.2f} - Last mean reward per episode:
{mean_reward:.2f}")

              # New best model, you could save the agent here
              if mean_reward > self.best_mean_reward:
                  self.best_mean_reward = mean_reward
                  # Example for saving best modell
                  if self.verbose > 0:
                    print(f"Saving new best model to {os.path.join(self.chk_dir, 'best_model')}")
                  self.model.save(os.path.join(self.chk_dir, 'best_model'))

        return True

def startGameRand(env):
    fin = True
    for step in range(100000):
        if fin:
            env.reset()
        state, reward, fin, info = env.step(env.action_space.sample())
        env.render()
    env.close()

def startGameModel(env, model):
    state = env.reset()
    while True:
        action, _ = model.predict(state)
        state, _, _, _ = env.step(action)
        env.render()

def saveGameRand(env, len = 100000, dir = './videos/'):
    env = RecordVideo(env, dir + str(time()) + '/')
    fin = True
    for step in range(len):
        if fin:
            env.reset()
        state, reward, fin, info = env.step(env.action_space.sample())
    env.close()

def saveGameModel(env, model, len = 100000, dir = './videos/'):
    env = RecordVideo(env, dir + str(time()) + '/')
    fin = True
    for step in range(len):
        if fin:
            state = env.reset()
        action, _ = model.predict(state)
        state, _, fin, _ = env.step(action)
    env.close()

# Import Environment libraries
import gym_super_mario_bros
from nes_py.wrappers import JoypadSpace
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT

# Start the environment
env = gym_super_mario_bros.make('SuperMarioBros-v0') # Generates the environment
# env = gym_super_mario_bros.make('SuperMarioBrosRandomStages-v0') # Generates the environmen
env = JoypadSpace(env, SIMPLE_MOVEMENT) # Limits the joypads moves with important moves
```

```
# Import preprocessing wrappers
from gym.wrappers import GrayScaleObservation
from stable_baselines3.common.vec_env import VecFrameStack, DummyVecEnv, VecMonitor
from matplotlib import pyplot as plt

# Apply the preprocessing
env = GrayScaleObservation(env, keep_dim=True) # Convert to grayscale to reduce dimensionality
env = DummyVecEnv([lambda: env])

# Alternatively, you may use SubprocVecEnv for multiple CPU processors
env = VecFrameStack(env, 4, channels_order='last') # Stack frames
env = VecMonitor(env, "./train/TestMonitor") # Monitor

#from utils import SaveOnBestTrainingRewardCallback

from stable_baselines3 import DQN
from stable_baselines3 import PPO

# CHECKPOINT_DIR = './train/'
# LOG_DIR = './logs/'

# # #@markdown #**PPO**

# callback = SaveOnBestTrainingRewardCallback(save_freq=100000,
check_freq=1000,chk_dir=CHECKPOINT_DIR)

# model = DQN('CnnPolicy',
#             env,
#             batch_size=192,
#             verbose=1,
#             learning_starts=10000,
#             learning_rate=5e-3,
#             exploration_fraction=0.1,
#             exploration_initial_eps=1.0,
#             exploration_final_eps=0.1,
#             train_freq=8,
#             buffer_size=1000,
#             tensorboard_log=LOG_DIR
#             )
# model.learn(total_timesteps=1000000, log_interval=1, callback=callback)

# model =DQN.load('./DQN_4/train/best_model')
model =DQN.load('./DQN_4/train/models/iter_1000000')
startGameModel(env, model)
# saveGameModel(env,model)
```