In [3]:
```
!pip install gym pyvirtualdisplay > /dev/null 2>&1
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
!apt-get update > /dev/null 2>&1
!apt-get install cmake > /dev/null 2>&1
!pip install --upgrade setuptools 2>&1
!pip install ez_setup > /dev/null 2>&1
!pip install gym[atari] > /dev/null 2>&1
```

```
Collecting setuptools
  Downloading https://files.pythonhosted.org/packages/ae/42/2876a3a136f8bfa9bd703518441c
8db78ff1eeaddf174baa85c083c1fd15/setuptools-56.0.0-py3-none-any.whl (784kB)
     |████████████████████████████████| 788kB 11.7MB/s
ERROR: datascience 0.10.6 has requirement folium==0.2.1, but you'll have folium 0.8.3 wh
ich is incompatible.
Installing collected packages: setuptools
  Found existing installation: setuptools 54.2.0
    Uninstalling setuptools-54.2.0:
      Successfully uninstalled setuptools-54.2.0
Successfully installed setuptools-56.0.0
```

In [3]:
```
import gym
from gym.wrappers import Monitor
import glob
import io
import base64
import numpy as np
from random import random, uniform, randint
from tqdm import tqdm_notebook as tqdm
import pandas as pd
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
from IPython.display import clear_output, HTML
from IPython import display as ipythondisplay
# from pyvirtualdisplay import Display
#from keras.models import Sequential
#from keras.layers import Dense
#from keras.optimizers import Adam
```

In [102…
```
class NeuralNet():
  def __init__(self,
               input_space=1,
               output_space=1,
               hidden_layer_sizes=(64, 64),
               activation_func='relu',
               input_name='state',
               output_name='actions',
               alpha = 0.0001,
               optimizer = 'SGD',
               error_func = 'MSE'
               ):
    self.output_space = output_space
    inputs=keras.Input(shape=(input_space,),name=input_name)
    previous_layer = inputs
    for hidden_layer_size in hidden_layer_sizes:
      x = keras.layers.Dense(hidden_layer_size,
                             activation=activation_func)(previous_layer)
      previous_layer = x
    outputs = keras.layers.Dense(output_space, name=output_name)(x)
```

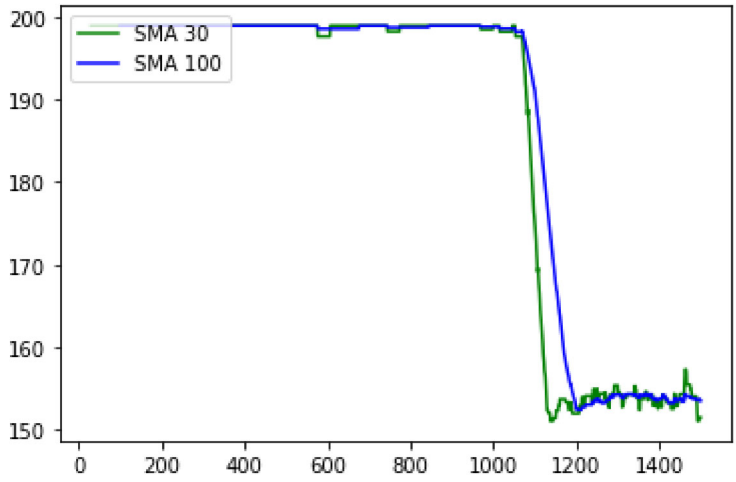
```
    self.model = keras.Model(inputs=inputs, outputs=outputs)
    self.alpha=alpha
    if optimizer == 'SGD':
      self.optimizer=keras.optimizers.SGD(learning_rate=alpha)
    elif optimizer == 'Adam':
      self.optimizer=keras.optimizers.Adam(learning_rate=alpha)
    else:
      # TODO
      self.optimizer=keras.optimizers.SGD(learning_rate=alpha)

    if error_func == 'MSE':
      self.error_func = keras.losses.MeanSquaredError()
    else:
      # TODO
      self.error_func = tf.keras.losses.CategoricalCrossentropy(reduction=tf.keras.loss

  def get_model(self):
    return self.model

  def predict(self, obs):
    return self.model.predict(obs)

  def predict_q(self, x_train):
    x_train = tf.constant([x_train], dtype='float')
    with tf.GradientTape() as tape:
      predictions = self.model(x_train, training=True)
    return predictions

  def update(self, x_train, G, action, verbose=False):
     x_train = tf.constant([x_train], dtype='float')
     with tf.GradientTape() as tape:
      #  predictions = self.model(x_train, training=True)
      #  print('pred all, ', predictions)
      #  pred = predictions[0, action]
      #  print('pred all, ', predictions)
      #  print('pred action, ', predictions[0, action])
      #  y_train = predictions[0].numpy()
      #  y_train[action] = G
      #  y_train = tf.constant(y_train)
      #  ppError = self.error_func([G], [pred])
       predictions = self.model(x_train, training=True)
       y_train = predictions[0].numpy()
       y_train[action] = G
       y_train = tf.constant([y_train])
       ppError = self.error_func(y_train, predictions)
       if verbose: print(f'Error: {ppError}')
     grads = tape.gradient(ppError, self.model.trainable_weights)
     self.optimizer.apply_gradients(zip(grads, self.model.trainable_weights))
     return ppError
```

In [119…

```
class agent():
  def __init__(self,
               env_name='MountainCar-v0',
               layer_size = (64,64),
               optimizer='Adam',
               alpha=0.5
               ):
    self.env = gym.make(env_name)
    self.model = NeuralNet(
```

```
        input_space=self.env.observation_space.shape[0],
        output_space=self.env.action_space.n,
        hidden_layer_sizes=layer_size,
        optimizer=optimizer,
        alpha=alpha
    )


  def choose_action(self, epsilon=0.5, policy=None, greedy=False, obs=None):
    if policy and (greedy or (uniform(0, 1) >= epsilon)):
      return np.argmax(self.model.predict_q(obs))
    else:
      return self.env.action_space.sample()

  def run(self,
          epsilon=0.5,
          policy=False,
          greedy=False,
          max_iter = 50000,
          verbose=False):
    obs = self.env.reset()
    action = self.choose_action(obs=obs)
    for i in range(max_iter):
      obs_, reward_, done, info = self.env.step(action)
      action_ = self.choose_action(epsilon=epsilon,
                                   policy=policy,
                                   greedy=greedy,
                                   obs=obs_)
      yield obs, action, reward_, obs_, action_, done
      obs, action = obs_, action_
      if verbose:
        print("step i",i,"action=",action)
        print("obs=",obs,"reward=",reward_,"done=",done,"info=",info)
      if done:
        break
    self.env.close()

  def test(self, n_episodes = 10):
      for i in tqdm(range(n_episodes), position=0):
        results = []
        for j, (obs,
                action,
                reward_,
                obs_,
                action_,
                done) in enumerate(self.run(epsilon=0.5,
                                            policy=True,
                                            greedy=True,
                                            max_iter = 50000,
                                            verbose=False)):
          pass
        results.append(j)
      return np.mean(results)

# RL Book page. 209
  def train(self,
            n = 8,
            epsilon = 0.5,
            gamma = 0.9,
            n_episodes = 2000,
            verbose=False
```

```
            ):
    train_history = []
    position_target = 0.5

    for i in tqdm(range(n_episodes), position=0):
      episodes = []
      for j, (obs,
              action,
              reward_,
              obs_,
              action_,
              done) in enumerate(self.run(epsilon=0.5,
                                          policy=True,
                                          greedy=False,
                                          max_iter = 50000,
                                          verbose=False)):
        if self.env.spec.id == 'MountainCar-v0':
          reward_ -= 1
        elif self.env.spec.id == 'CartPole-v0':
          if j == 199:
            reward_ = 1
          elif done:
            reward_ = -1
          else:
            reward_ = 0
        # reward_ = 1 if obs[0] > position_target else -np.abs(position_target - obs[0])
        # print(reward_)
        episodes.append((obs, action, reward_, obs_, action_, done))
        tau = j - n + 1
        if tau >= 0:
          G = np.sum([
                      gamma ** (k-tau-1) + episodes[k][2]
                      for k in range(tau+1, tau+n)
                      ])
          # print('1, ', G)
          if not done or j == 199:
            G += gamma ** n * self.model.predict_q(obs_)[:, action_]
          print('pred, ', self.model.predict_q(obs_))
          self.model.update(obs, G, action, verbose=verbose)
      train_history.append(j)
      if i % 100 == 0:
        clear_output(wait=True)
        # plt.plot(train_history, 'k.', label='data')
        # df = pd.DataFrame(train_history, columns=['data'])
        # df['SMA10'] = df['data'].rolling(window=10).mean()
        # plt.plot(df['SMA10'], "r", label='SMA 10')

        df['SMA30'] = df['data'].rolling(window=30).mean()
        plt.plot(df['SMA30'], "g", label='SMA 30')

        df['SMA100'] = df['data'].rolling(window=100).mean()
        plt.plot(df['SMA100'], "b", label='SMA 100')

        plt.legend(loc=2)
        plt.show()
    return train_history

mountain_car_agent = agent(env_name='MountainCar-v0', layer_size = (1024, 1024))
history = mountain_car_agent.train(verbose=False, n_episodes=1500)
```



In [120…

```
average_result = mountain_car_agent.test()
print('Test Result is ', average_result)
```

```
Test Result is  163.7
```