**A.I. Game-bot**

**BY:-**

**Vineeth Chandran :-BL.EN.U4CSE15130**

**Raguvir :-BL.EN.U4CSE15104**

**Ashwini :-BL.EN.U4CSE15013**

**Shreyansh :-BL.EN.U4CSE15502**

**Subject Code :- 15CSE387**

**Subject Name :- Open Labs**

What Is the Project About?

**What is a bot**: In video games, a bot is a type of **AI expert system software** that plays a video game in the place of a human.

The goal of our project is to make an **AI Game Bot**. It will use reinforcement learning in which the game bot will learn how to play a game by repeatedly playing it and learning by trial and error ( learning from its mistakes).

The inspiration came from the breakthrough achieved by Google’s AlphaGo which beat the world’s best Go (A Chinese board game similar to chess) player which is said to be so complex that the number of legal board positions are greater than the 10211. After which Elon Musk’s OpenAI achieved another feat of beating the world’s best D.O.T.A II (a massively multiplayer online game based on strategy) player.

OpenAI released the python library **Universe.** Universe allows an AI agent to use a computer like a human does: by looking at screen pixels and operating a virtual keyboard and mouse.

Gym is a toolkit for developing and comparing reinforcement learning (RL) algorithms. With Universe, any program can be turned into a Gym environment.

Packages Used:-

Gym by OpenAI

TFlearn by Tensorflow

Universe

Numpy

Cuda

Gym :- Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games.

Tflearn :- TFlearn is a modular and transparent deep learning library built on top of Tensorflow. It was designed to provide a higher-level API to TensorFlow in order to facilitate and speed-up experimentation, while remaining fully transparent and compatible with it.

Universe:- Universe is a software platform for measuring and training an AI's general intelligence across the world's supply of games, websites and other applications. This is the universe open-source library, which provides a simple Gym interface to each Universe environment.

NumPy:- NumPy is a library for the Python programming language, adding support for large, muti-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Cuda :- CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units).

Work Division:-

Raguvir:-

Developed the initial population function which involves training the data with by using random values between 0 and 1, where 0 is for left and 1 is for right. Only those models will be taken whose score is greater than the minimum score.

Code snippet:-

if score >= score\_requirement:

accepted\_scores.append(score)

for data in game\_memory:

if data[1] == 1:

output = [0,1]

elif data[1] == 0:

output = [1,0]

training\_data.append([data[0], output])

Vineeth:-

Made it possible to train the bot manually. Here the user will train the bot by playing a few number of games. This sample will be stored in a list called training\_data.

Code snippet:-

for \_ in range(goal\_steps):

action = random.randrange(0,2)

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

if len(prev\_observation) > 0:

game\_memory.append([prev\_observation, action])

prev\_observation = observation

score +=reward

if done:

break

Ashwini:-

Made the train\_model function which sends the training\_data list to the neural\_network\_model function where the bot is trained to play the game.

Code snippet:-

def train\_model(training\_data, model=False):

X = np.array([i[0] for i in training\_data]).reshape(-1, len(training\_data[0][0]), 1)

y = [i[1] for i in training\_data]

if not model:

model = neural\_network\_model(input\_size = len(X[0]))

Shreyansh:-

Made the neural\_network\_model function used to train the bot based on the training\_data list. There are 6 layers to the DNN including the output layer.

Code snippet:-

network = input\_data(shape=[None, input\_size, 1], name='input')

network = fully\_connected(network, 128, activation='relu')

network = dropout(network, 0.8)

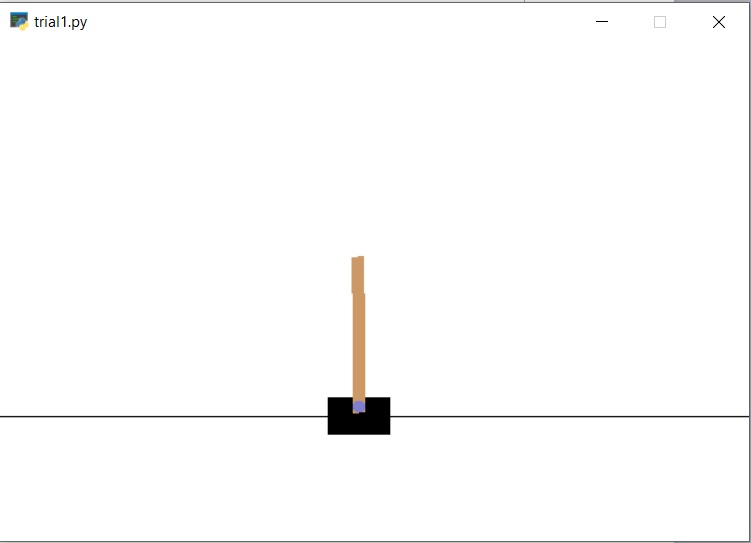
network = fully\_connected(network, 256, activation='relu')

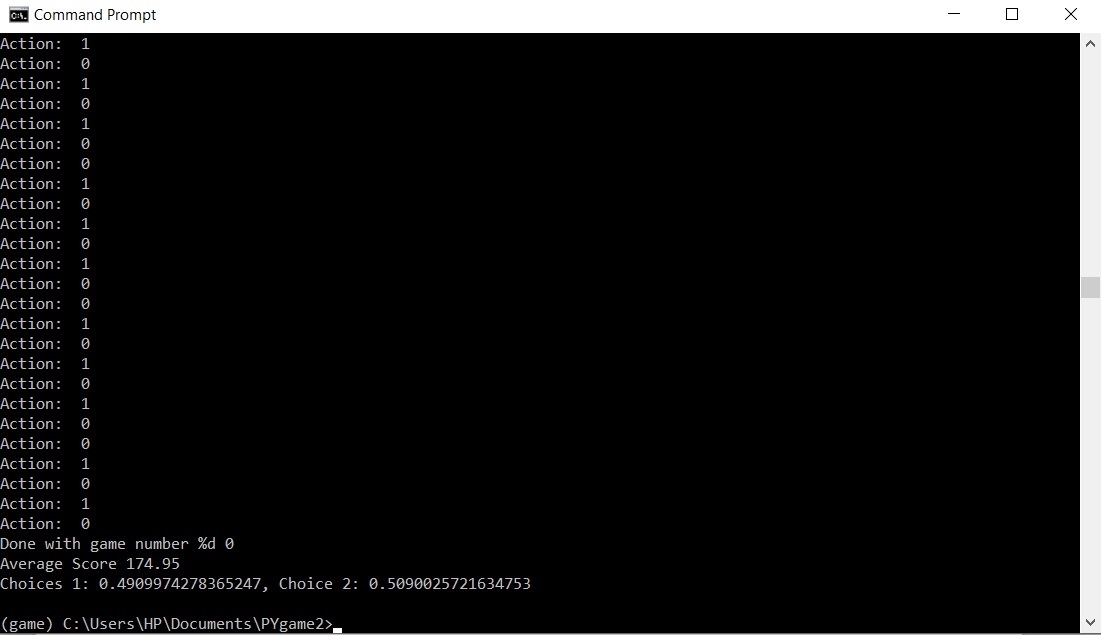
network = dropout(network, 0.8)

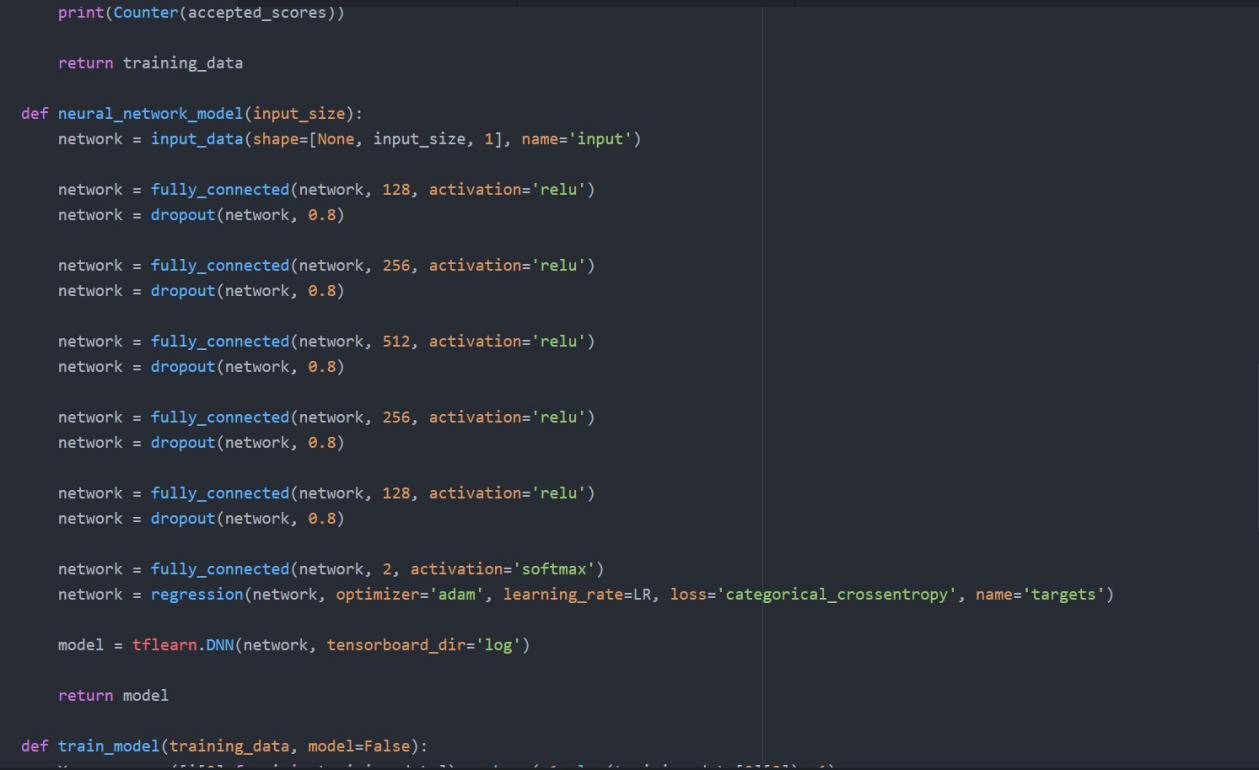
network = fully\_connected(network, 512, activation='relu')

network = dropout(network, 0.8)

Screenshot:-







Conclusion:-

The game bot learns from the training data that involves a random action of moving the cart left and right with a set of actions denoted by 0 or 1.

This data is the trained using a dense neural network which has 5 hidden layers and one output layer.

The initial number of games is 1000 which means the training set will have the 1000 samples out of which the ones above MIN\_SCORE(50) will be used for training the bot.

Once the bot is trained we make it play 20 games and calculate the mean score.

Our bot achieves an average score of 200 most of the times but the value will vary because the training data-set may not always be good.