SOURCE CODE

Import Packages

```
import PIL.Image, PIL.ImageTk
import random
import time
import threading
from PIL import Image
from PIL import ImageTk
```

Initialization

```
class PacketPriority:
    def __init__(self, master):
        self.master = master
        self.frame = Frame(self.master)
        self.a1 = Label(self.master, text='Packet Priority Queuing Model an Emergency Packet Scheduling',bg='lightblue', fg='black', font=("Helvetica", 16))
        self.a1.pack()
        self.a1.place(x=50, y=30)
        self.a1 = Label(self.master, text=' in IoT Network Using Reinforcement Learning',bg='lightblue', fg='black', font=("Helvetica", 16))
        self.a1.pack()
        self.a1.place(x=100, y=60)
        mm = PIL.Image.open("dttr.jpg")
        img2 = PIL.ImageTk.PhotoImage(mm)
        panel2 = Label(self.master, image = img2)
        panel2.image = img2 # keep a reference!
        panel2.pack()
        panel2.place(x=100,y=100)
        self.b2 = Button(self.master, text=" << Packet Information >> ", command=self.simulate)
        self.b2.pack()
        self.b2.place(x=260, y=450)
```

Sense Data

```
class Sense_data():
    def __init__(self, master):
            self.master = master
            self.frame = Frame(self.master)
            self.a10 = Label(self.master, text='Packet Priority Queuing Model',bg='lightblue',
fg='blue', font=("Helvetica", 16))
            self.a10.pack()
            self.a10.place(x=150, y=10)
            self.a11 = Label(self.master, text='Monitor',bg='lightblue', fg='blue', font=("Helvetica",
12))
            self.a11.pack()
            self.a11.place(x=130, y=90)
for cs in cc:
              i=1
              j=0
              number=cs
              self.n=number
              f=cs
              #print("f="+str(f) +"k="+str(k))
              while i <= number
                if k==1:
                  #f=cs
                  xr = random.randint(60,180)
                  yr = random.randint(50,150)
                elif k==2:
                  #f=cs
                  xr = random.randint(250,350)
                  yr = random.randint(50,150)
                elif k==3:
                  #f=cs
                  xr = random.randint(50,150)
                  yr = random.randint(230,350)
                else:
```

```
                #f=cs
                xr = random.randint(250,350)
                yr = random.randint(230,350)
          tt="N"+str(i)
              a = xr
              b = yr
              r = 5
              x0 = a - r
              y0 = b - r
              x1 = a + r
              y1 = b + r
              rh=10
              xh0 = a - rh
              yh0 = b - rh
              xh1 = a + rh
              yh1 = b + rh
              bob="b"+str(i)
              if k==1 and f==i:
                self.cir = self.canvas.create_oval(xh0, yh0, xh1, yh1,
                                                   fill="red", tags=bob)
                self.canvas.pack()
                self.txt = self.canvas.create_text(xr-10, yr-10, text="CH", font=("purisa",8),
fill="#660033", tags=bob)
              elif k==1:
                self.cir = self.canvas.create_oval(x0, y0, x1, y1,
                                                   fill="#FF9900", tags=bob)
                self.canvas.pack()
                self.txt = self.canvas.create_text(xr-10, yr-10, text=tt, font=("purisa",8),
fill="black", tags=bob)
              if k==2 and f==i:
                #print("c")
                self.cir = self.canvas.create_oval(xh0, yh0, xh1, yh1,
                                    fill="#669900", tags=bob)
                self.canvas.pack()
                self.txt = self.canvas.create_text(xr-10, yr-10, text="CH", font=("purisa",8),
fill="#669900", tags=bob)
              elif k==2:
                #print("d")
                self.cir = self.canvas.create_oval(x0, y0, x1, y1,
                                    fill="#00FF00", tags=bob)
                self.canvas.pack()
                self.txt = self.canvas.create_text(xr-10, yr-10, text=tt, font=("purisa",8),
fill="black", tags=bob)
              if k==3 and f==i:
                self.cir = self.canvas.create_oval(xh0, yh0, xh1, yh1,
                                    fill="#990000", tags=bob)
                self.canvas.pack()
                self.txt = self.canvas.create_text(xr-10, yr-10, text="CH", font=("purisa",8),
fill="#990000", tags=bob)
              elif k==3:
                self.cir = self.canvas.create_oval(x0, y0, x1, y1,
                                    fill="yellow", tags=bob)
                self.canvas.pack()
                self.txt = self.canvas.create_text(xr-10, yr-10, text=tt, font=("purisa",8),
fill="black", tags=bob)
              if k==4 and f==i:
                self.cir = self.canvas.create_oval(xh0, yh0, xh1, yh1,
                                    fill="#0033CC", tags=bob)
def movement(self):
    k=1
    while k <= self.n:
      bb="b"+str(k)
      ab = random.randint(-10, 10)
      cd = random.randint(-10, 10)
```

```
            movenode=self.canvas.move(bb, ab, cd)
            k += 1
        self.canvas.after(700, self.movement)
##Gas
        gv4=""
        gr1=randint(10,80)
        gr2=randint(10,80)
        gr3=randint(1,30)
        ga1=randint(50,70)
        ga2=randint(1,7)
        gv1=str(ga1)+"."+str(gr3)
        gv2=str(ga2)+"."+str(gr3)
        gv3="0"+"."+str(gr3)
        if ga2>4:
            gas_st=1
            gv4="Abnormal"
        else:
            gv4=""
            gas_st=0
        gas_val="GAS: NH3: "+gv1+", CO:"+gv2+", SO2:"+gv3+" "+gv4
        get_msg.append(gas_val)
        gas_val2="GAS: NH3: "+gv1+", CO:"+gv2+", SO2:"+gv3
        get_msg2.append(gas_val2)
        #######
        ##fire
        fr1=randint(20,200)
        fr2=randint(30,34)
        fr3=""
        if fr1>100:
            fir_st=1
            fr3="Abnormal"
        else:
```

```
    fr3=""
    fir_st=0
fir_val="Fire: Smoke: "+str(fr1)+", T:"+str(fr2)+" "+fr3
get_msg.append(fir_val)
fir_val2="Fire: Smoke: "+str(fr1)+", T:"+str(fr2)
get_msg2.append(fir_val2)
##health
hr1=randint(40,120)
hr2=randint(30,34)
hr3=""
if hr1<60 or hr1>80:
    hea_st=1
    hr3="Abnormal"
else:
    hr3=""
    hea_st=0
hea_val="Health: HB: "+str(hr1)+", T:"+str(hr2)+" "+hr3
get_msg.append(hea_val)
hea_val2="Health: HB: "+str(hr1)+", T:"+str(hr2)
get_msg2.append(hea_val2)
sr1=randint(40,120)
sr2=randint(30,34)
sr3=randint(40,100)
env_val="Environmental: M: "+str(sr1)+", T:"+str(sr2)+", H:"+str(sr3)
get_msg.append(env_val)
env_val2="Environmental: M: "+str(sr1)+", T:"+str(sr2)+", H:"+str(sr3)
get_msg2.append(env_val2)
#######
#self.a1 = Label(self.master, text=gas_val,bg='lightblue', fg='blue', font=("Helvetica", 12))
#self.a1.pack()
#self.a1.place(x=50, y=50)
farr="|".join(get_msg)
```

```
        farr2="|".join(get_msg2)
        f1=open("data1.txt","r")
        data1=f1.read()
        f1.close()
        f1=open("data2.txt","r")
        data2=f1.read()
        f1.close()
```

LSTM Prioritization

```
##LSTM
    def load_data(stock, seq_len):
        amount_of_features = len(stock.columns)
        data = stock.as_matrix() #pd.DataFrame(stock)
        sequence_length = seq_len + 1
        result = []
        for index in range(len(data) - sequence_length):
            result.append(data[index: index + sequence_length])
        result = np.array(result)
        row = round(0.9 * result.shape[0])
        train = result[:int(row), :]
        x_train = train[:, :-1]
        y_train = train[:, -1][:,-1]
        x_test = result[int(row):, :-1]
        y_test = result[int(row):, -1][:,-1]
        x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], amount_of_features))
        x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], amount_of_features))
        return [x_train, y_train, x_test, y_test]
    def build_model(layers):
        model = Sequential()
        model.add(LSTM(
            input_dim=layers[0],
            output_dim=layers[1],
            return_sequences=True))
        model.add(Dropout(0.2))
        model.add(LSTM(
            layers[2],
            return_sequences=False))
        model.add(Dropout(0.2))
        model.add(Dense(
            output_dim=layers[2]))
        model.add(Activation("linear"))
        start = time.time()
        model.compile(loss="mse", optimizer="rmsprop",metrics=['accuracy'])
        print("Compilation Time : ", time.time() - start)
        return model
    def build_model2(layers):
            d = 0.2
            model = Sequential()
            model.add(LSTM(128, input_shape=(layers[1], layers[0]), return_sequences=True))
            model.add(Dropout(d))
            model.add(LSTM(64, input_shape=(layers[1], layers[0]), return_sequences=False))
            model.add(Dropout(d))
            model.add(Dense(16,init='uniform',activation='relu'))
            model.add(Dense(1,init='uniform',activation='linear'))
            model.compile(loss='mse',optimizer='adam',metrics=['accuracy'])
            return model
def chPriority(self):
        f3=open("det2.txt","r")
        v3=f3.read()
        f3.close()
        v4=int(v3)+1
        mss=""
```

Deep Queue Scheduling

```
def DeepQLearning(num_of_nodes):
        enviroment = "Packet"
```

```
action_space="1"
action=0
alpha=1
reward=0
for customer in range(0, num_of_nodes):
    # Reset the enviroment
    state = enviroment
    # Initialize variables
    reward = 0
    terminated = False
    j=1
    n=num_of_nodes
    while j<n:
        # Take learned path or explore new actions based on the epsilon
        if random.uniform(0, 1) < num_of_nodes:
            i=0
            k=0
            while i<=num_of_nodes:
                i+=3
                k+=1
            action = i
        else:
            action = np.argmax(q_table[state])
        # Take action
        gamma=1
        #next_state, reward, terminated, info = action
        q_table=num_of_nodes/3
        # Recalculate
        q_value = k
        max_value = q_table #np.max(q_table[next_state])
        new_q_value = (1 - alpha) * int(q_value) + alpha * (reward + gamma * max_value)
        # Update Q-table
```

```
            #q_table[state, action] = new_q_value
            state = new_q_value
            j+=1
        #if (queue + 1) % 100 == 0:
        #    clear_output(wait=True)
            #print("Queue: {}".format(queue + 1))
            #enviroment.render()
def QueuePredict(enviroment, optimizer):
        # Initialize atributes
        _state_size = enviroment
        _action_size = "1" #enviroment.action_space.n
        _optimizer = optimizer
        expirience_replay = int(enviroment/2)
        # Initialize discount and exploration rate
        gamma = 0.6
        epsilon = 0.1
        # Build networks
        q_network = optimizer
        target_network = expirience_replay
def store(state, action, reward, next_state, terminated):
    expirience_replay.append((state, action, reward, next_state, terminated))
def _build_compile_model():
    model = Sequential()
    model.add(Embedding(_state_size, 10, input_length=1))
    model.add(Reshape((10,)))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(_action_size, activation='linear'))
    model.compile(loss='mse', optimizer=self._optimizer)
    return model
def alighn_target_model():
    target_network.set_weights(q_network.get_weights())
```

```
def act(state):
    if np.random.rand() <= epsilon:
        return enviroment.action_space.sample()
    q_values = q_network.predict(state)
    return np.argmax(q_values[0])
def retrain(batch_size):
    minibatch = random.sample(expirience_replay, batch_size)
    for state, action, reward, next_state, terminated in minibatch:
        target = q_network.predict(state)
        if terminated:
            target[0][action] = reward
        else:
            t = target_network.predict(next_state)
            target[0][action] = reward + gamma * np.amax(t)
        q_network.fit(state, target, epochs=1, v
```