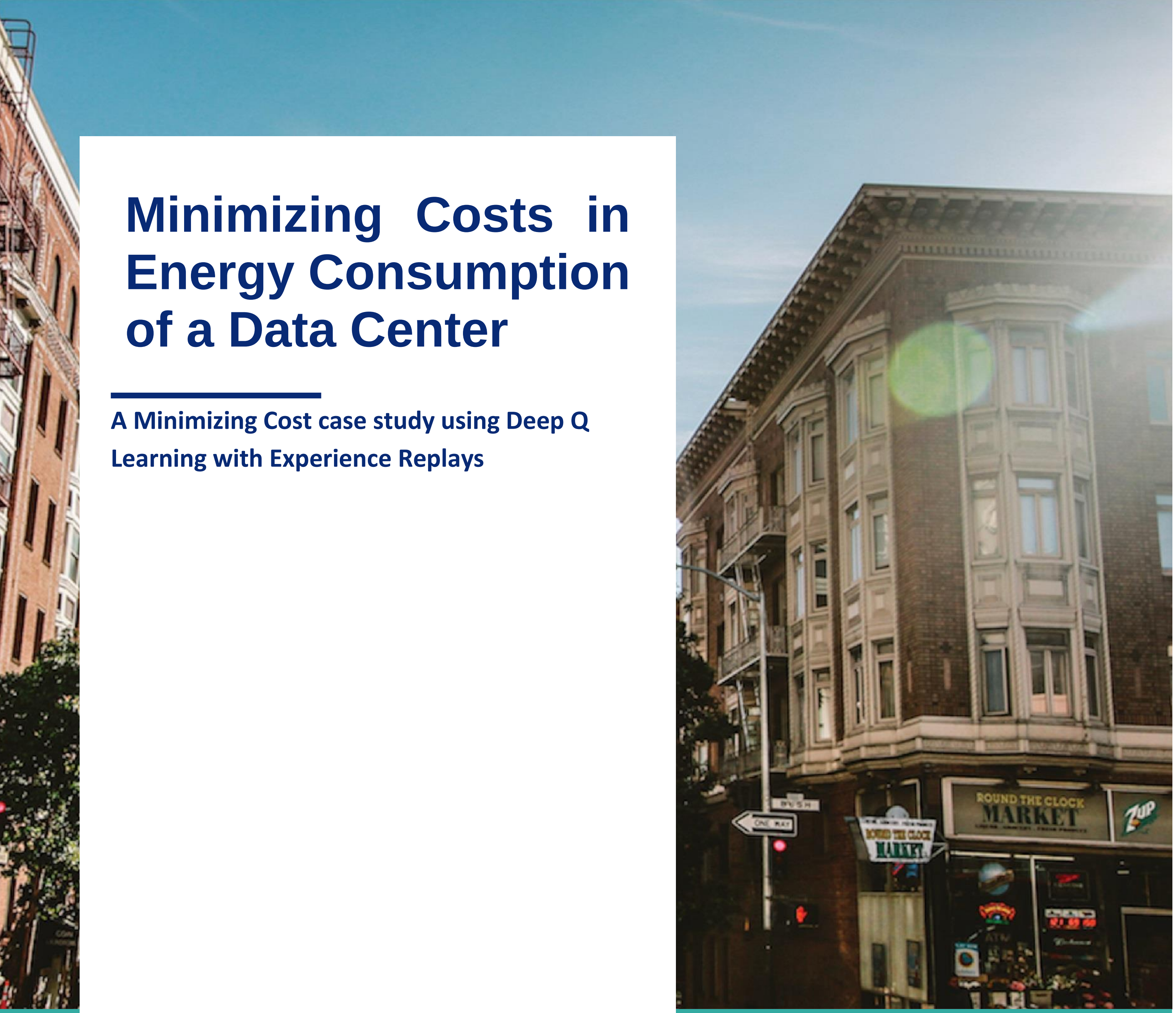


Minimizing Costs in Energy Consumption of a Data Center

A Minimizing Cost case study using Deep Q Learning with Experience Replays

MAY 20

Authored by: Anuj Malkotia

Problem to solve:

In 2016, DeepMind AI minimized a big part of Google's cost by reducing Google Data Centre Cooling Bill by 40% using their DQN AI model (Deep Q-Learning). In this case study, we will do something very similar. We will set up our own server environment, and we will build an AI that will be controlling the cooling/heating of the server so that it stays in an optimal range of temperatures while saving the maximum energy, therefore minimizing the costs. And just as DeepMind AI did, our goal will be to achieve at least 40% energy saving.

Environment to define:

Before we define the states, actions and rewards, we need to explain how the server operates. We will do that in several steps. First, we will list all the environment parameters and variables by which the server is controlled. After that we will set the essential assumption of the problem, on which our AI will rely to provide a solution. Then we will specify how we will simulate the whole process. And eventually we will explain the overall functioning of the server, and how the AI plays its role.

Parameters:

- the average atmospheric temperature over a month
- the optimal range of temperatures of the server, which will be [18_C; 24_C]
- the minimum temperature of the server below which it fails to operate, which will be - 20_C
- the maximum temperature of the server above which it fails to operate, which will be 80_C
- the minimum number of users in the server, which will be 10
- the maximum number of users in the server, which will be 100
- the maximum number of users in the server that can go up or down per minute, which will be 5
- the minimum rate of data transmission in the server, which will be 20
- the maximum rate of data transmission in the server, which will be 300

- the maximum rate of data transmission that can go up or down per minute, which will be 10

Variables:

- the temperature of the server at any minute
- the number of users in the server at any minute
- the rate of data transmission at any minute
- the energy spent by the AI onto the server (to cool it down or heat it up) at any minute
- the energy spent by the server's integrated cooling system that automatically brings the server's temperature back to the optimal range whenever the server's temperature goes outside this optimal range

All these parameters and variables will be part of our server environment and will influence the actions of the AI on the server. Then let's give and explain below the two core assumptions of the environment. It is important to understand that these assumptions are not AI related, but just used to simplify the environment so that we can focus the maximum on the AI solution.

Assumptions:

We will rely on the following two essential assumptions:

Assumption 1: The temperature of the server can be approximated through Multiple Linear Regression, by a linear function of the atmospheric temperature, the number of users and the rate of data transmission:

$$\text{server temperature} = b_0 + b_1 \times \text{atmospheric temperature} + b_2 \times \text{number of users} + b_3 \times \text{data transmission rate}$$

where $b_0 \in \mathbb{R}$, $b_1 > 0$, $b_2 > 0$ and $b_3 > 0$.

Eventually, let's assume further that after performing this Multiple Linear Regression, we obtained the following values of the coefficients: b0 = 0, b1 = 1, b2 = 1:25 and b3 = 1:25. Accordingly:

$$\text{server temperature} = \text{atmospheric temperature} + 1.25 \times \text{number of users} + 1.25 \times \text{data transmission rate}$$

Assumption 2: The energy spent by a system (our AI or the server's integrated cooling system) that changes the server's temperature from Tt to Tt+1 within 1 unit of time (here 1 minute), can be approximated again through regression by a linear function of the server's absolute temperature change:

$$E_t = \alpha|\Delta T_t| + \beta = \alpha|T_{t+1} - T_t| + \beta$$

where:

$$\begin{cases} E_t \text{ is the energy spent by the system onto the server between times } t \text{ and } t+1 \text{ minute} \\ \Delta T_t \text{ is the change of the server's temperature caused by the system between times } t \text{ and } t+1 \text{ minute} \\ T_t \text{ is the temperature of the server at time } t \\ T_{t+1} \text{ is the temperature of the server at time } t+1 \text{ minute} \\ \alpha > 0 \\ \beta \in \mathbb{R} \end{cases}$$

Again, let's explain why it intuitively makes sense to make this assumption with $\alpha > 0$. That's simply because the more the AI heats up or cools down the server, the more it spends energy to do that heat transfer. Indeed for example, imagine the server suddenly has overheating issues and just reached 80°C, then within one unit of time (1 minute) the AI will need much more energy to bring the server's temperature back to its optimal temperature 24°C than to bring it back to 50°C for example. And again for simplicity purposes, we just suppose that these correlations are linear. Also (in case you are wondering), why do we take the absolute value? That's simply because when the AI cools down the server, $T_{t+1} < T_t$, so $\Delta T < 0$. And of course an energy is always positive so we have to take the absolute value of ΔT.

Eventually, for further simplicity purposes we will also assume that the results of the regression are $\alpha = 1$ and $\beta = 0$, so that we get, the following final equation based on Assumption 2:

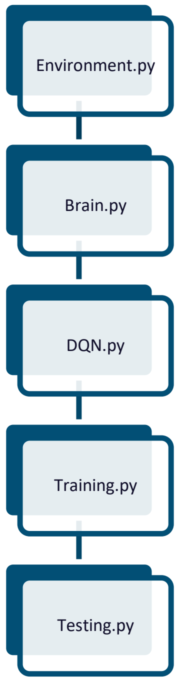
$$E_t = |\Delta T_t| = |T_{t+1} - T_t| = \begin{cases} T_{t+1} - T_t & \text{if } T_{t+1} > T_t \text{, that is if the server is heated up} \\ T_t - T_{t+1} & \text{if } T_{t+1} < T_t \text{, that is if the server is cooled down} \end{cases}$$

CREATING OUR MINIMIZING COST CASE STUDY WITH PYTHON:

We will create the following files:

1. **Environment.py** : The Framework for our minimizing cost case study
2. **Brain.py:** Artificial brain for our model
3. **Dqn.py:** Deep Q Learning Algorithm model
4. **Training.py:** For training our model when it is created which will consist of brain and dqn in environment
5. **Testing.py:** Testing our AI on the server

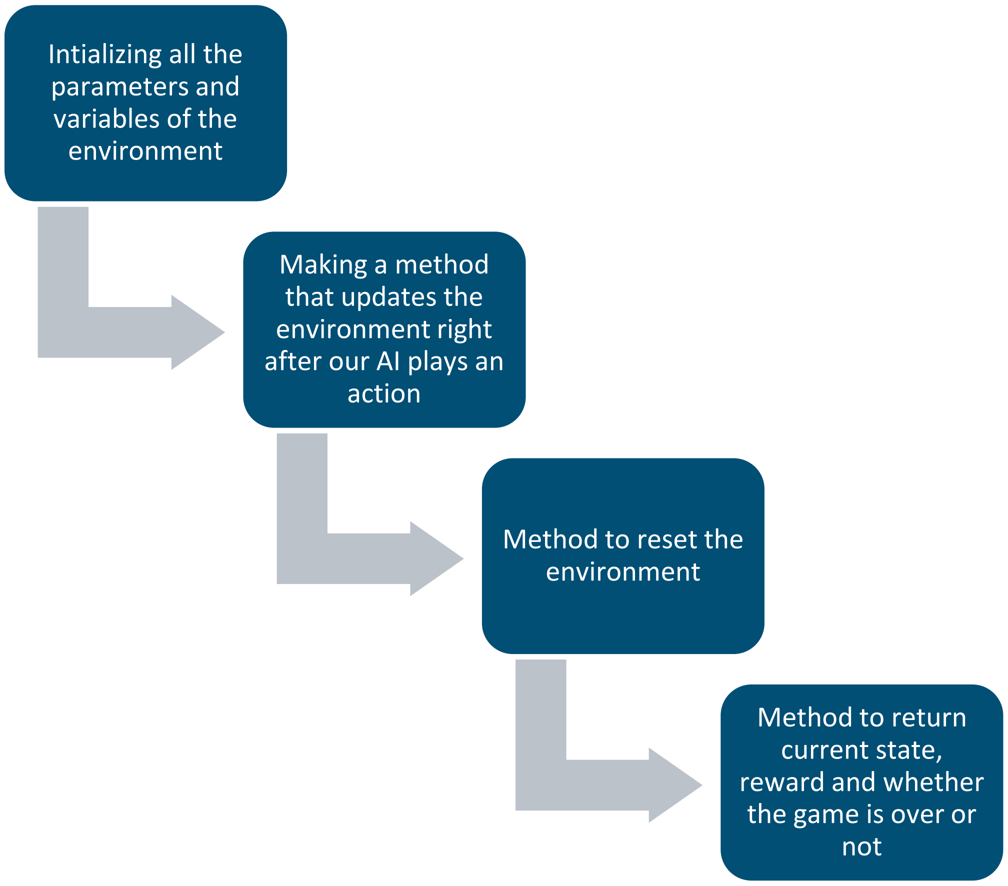
The Hierarchy on which we will create these files are:



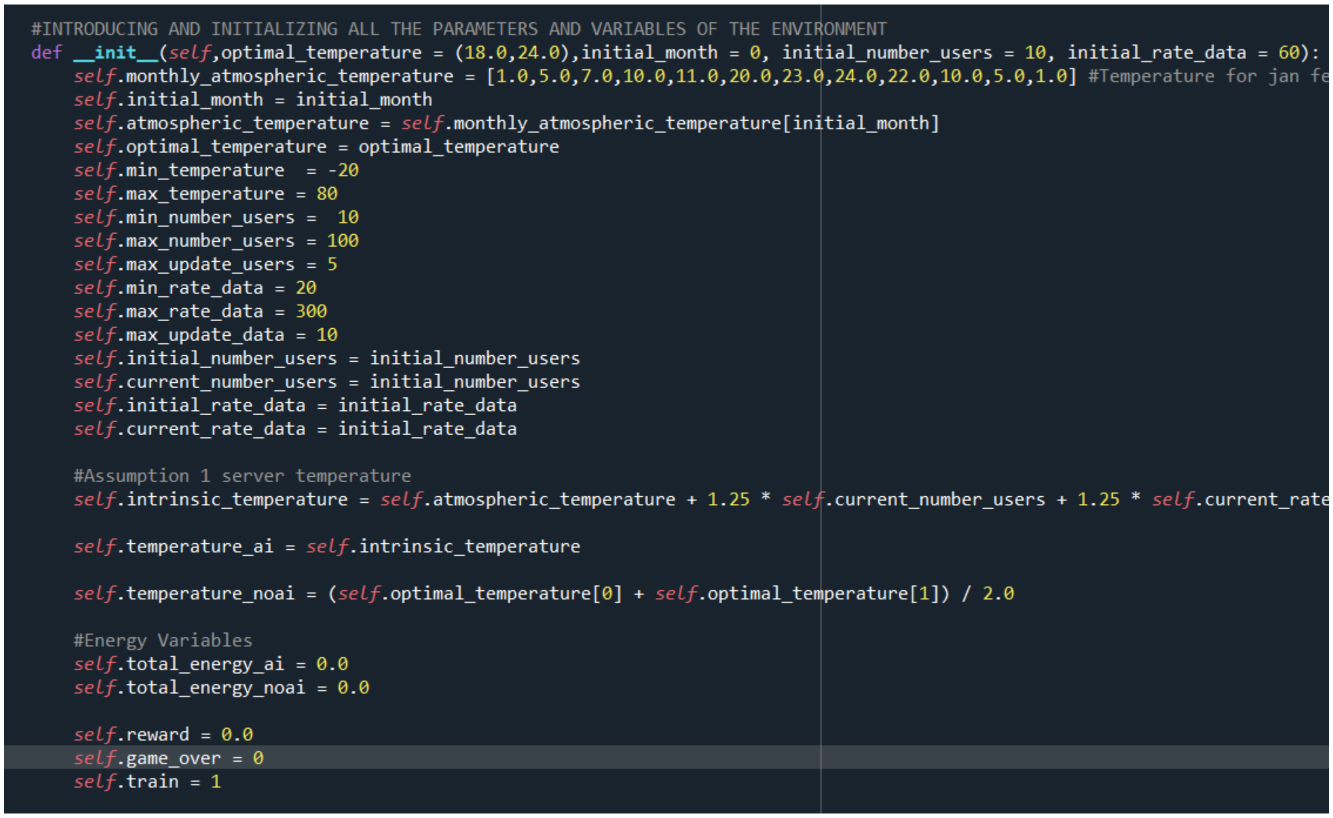
Step 1: Building the Environment

In this first step, we are going to build the environment inside a class. Why a class? Because we would like to have our environment as an object which we can create easily with any values of some parameters we choose. For example, we can create one environment object for one server that has a certain number of connected users and a certain rate of data at a specific time, and one other environment object for another server that has a different number of connected users and a different rate of data at some other time. And thanks to this advanced structure of the class, we can easily plug-

and-play the environment objects we create on different servers which have their own parameters, hence regulating their temperatures with several different AIso that we end up minimizing the energy consumption of a whole data center, just as Google DeepMind for Google’s data centers did with their DQN algorithm.
This class follows the below sub-steps, which are part of the general AI Framework inside Step 1 – Building bthe environment:

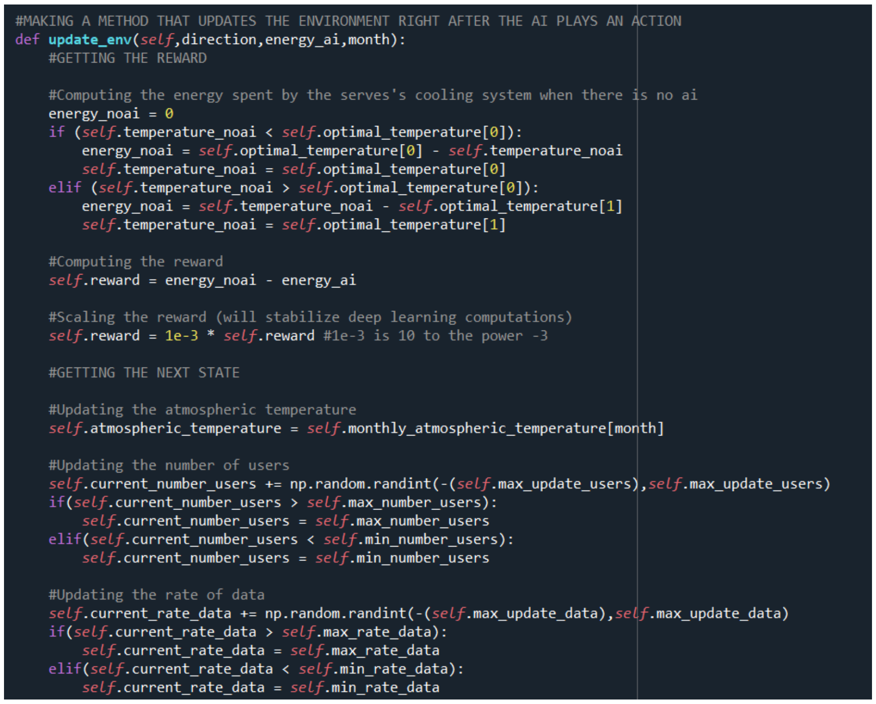

Intializing all the parameters and variables of the environment
Making a method that updates the environment right after our AI plays an action
Method to reset the environment
Method to return current state, reward and whether the game is over or not

1. **Introducing and initializing all the parameters and variables of the environment.**

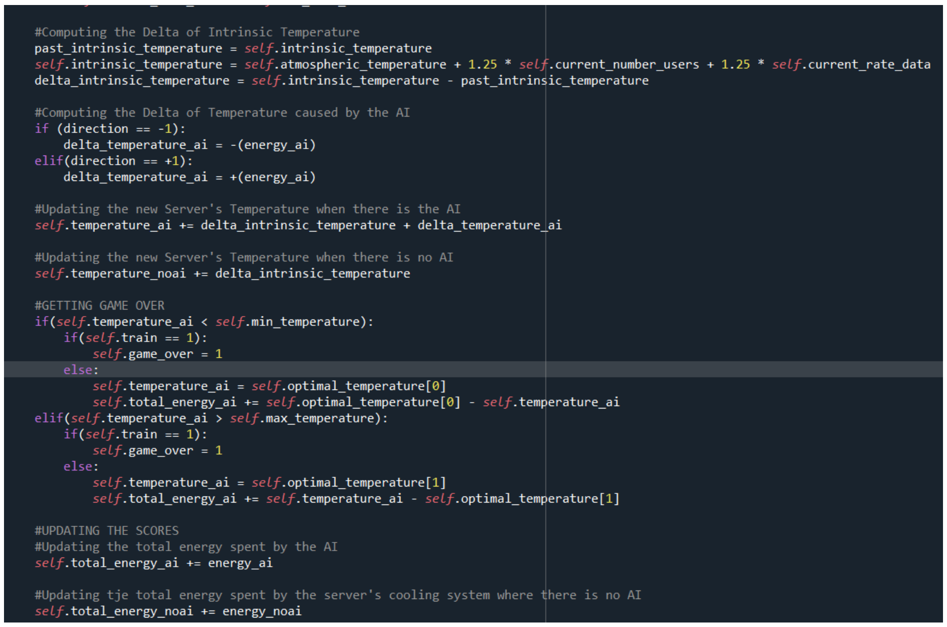


```
#INTRODUCING AND INITIALIZING ALL THE PARAMETERS AND VARIABLES OF THE ENVIRONMENT
def __init__(self,optimal_temperature = (18.0,24.0),initial_month = 0, initial_number_users = 10, initial_rate_data = 60):
    self.monthly_atmospheric_temperature = [1.0,5.0,7.0,10.0,11.0,20.0,23.0,24.0,22.0,10.0,5.0,1.0] #Temperature for jan fe
    self.initial_month = initial_month
    self.atmospheric_temperature = self.monthly_atmospheric_temperature[initial_month]
    self.optimal_temperature = optimal_temperature
    self.min_temperature  = -20
    self.max_temperature = 80
    self.min_number_users =  10
    self.max_number_users = 100
    self.max_update_users = 5
    self.min_rate_data = 20
    self.max_rate_data = 300
    self.max_update_data = 10
    self.initial_number_users = initial_number_users
    self.current_number_users = initial_number_users
    self.initial_rate_data = initial_rate_data
    self.current_rate_data = initial_rate_data

    #Assumption 1 server temperature
    self.intrinsic_temperature = self.atmospheric_temperature + 1.25 * self.current_number_users + 1.25 * self.current_rate

    self.temperature_ai = self.intrinsic_temperature

    self.temperature_noai = (self.optimal_temperature[0] + self.optimal_temperature[1]) / 2.0

    #Energy Variables
    self.total_energy_ai = 0.0
    self.total_energy_noai = 0.0

    self.reward = 0.0
    self.game_over = 0
    self.train = 1
```

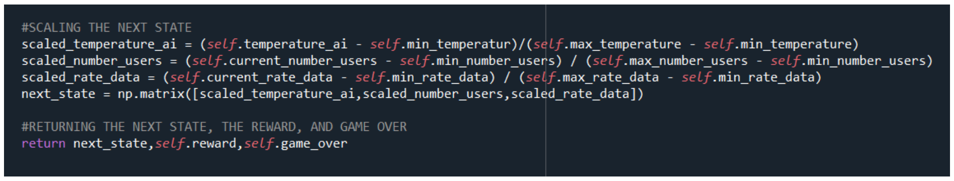
2. **Making a method that updates the environment right after the AI plays an action.**



```
#MAKING A METHOD THAT UPDATES THE ENVIRONMENT RIGHT AFTER THE AI PLAYS AN ACTION
def update_env(self,direction,energy_ai,month):
    #GETTING THE REWARD

    #Computing the energy spent by the serves's cooling system when there is no ai
    energy_noai = 0
    if (self.temperature_noai < self.optimal_temperature[0]):
        energy_noai = self.optimal_temperature[0] - self.temperature_noai
        self.temperature_noai = self.optimal_temperature[0]
    elif (self.temperature_noai > self.optimal_temperature[0]):
        energy_noai = self.temperature_noai - self.optimal_temperature[1]
        self.temperature_noai = self.optimal_temperature[1]

    #Computing the reward
    self.reward = energy_noai - energy_ai

    #Scaling the reward (will stabilize deep learning computations)
    self.reward = 1e-3 * self.reward #1e-3 is 10 to the power -3

    #GETTING THE NEXT STATE

    #Updating the atmospheric temperature
    self.atmospheric_temperature = self.monthly_atmospheric_temperature[month]

    #Updating the number of users
    self.current_number_users += np.random.randint(-(self.max_update_users),self.max_update_users)
    if(self.current_number_users > self.max_number_users):
        self.current_number_users = self.max_number_users
    elif(self.current_number_users < self.min_number_users):
        self.current_number_users = self.min_number_users

    #Updating the rate of data
    self.current_rate_data += np.random.randint(-(self.max_update_data),self.max_update_data)
    if(self.current_rate_data > self.max_rate_data):
        self.current_rate_data = self.max_rate_data
    elif(self.current_rate_data < self.min_rate_data):
        self.current_rate_data = self.min_rate_data
```

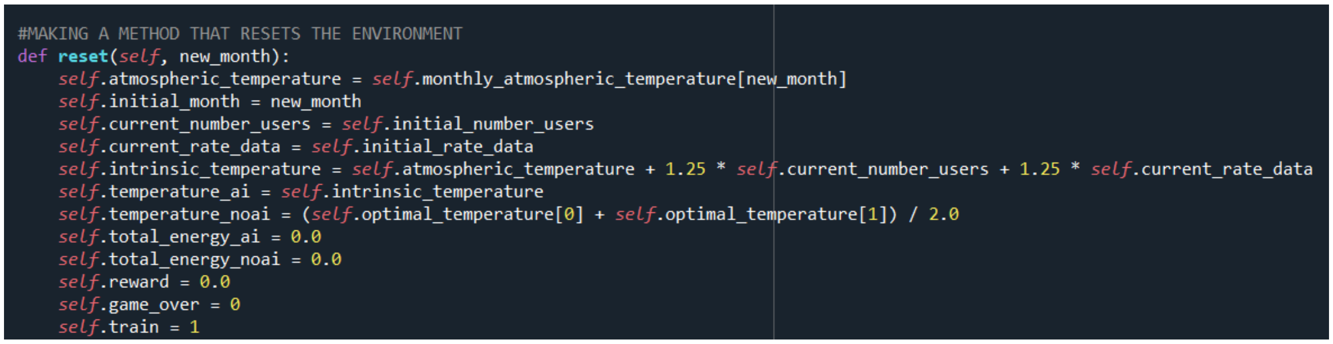


```
#Computing the Delta of Intrinsic Temperature
past_intrinsic_temperature = self.intrinsic_temperature
self.intrinsic_temperature = self.atmospheric_temperature + 1.25 * self.current_number_users + 1.25 * self.current_rate_data
delta_intrinsic_temperature = self.intrinsic_temperature - past_intrinsic_temperature

#Computing the Delta of Temperature caused by the AI
if (direction == -1):
    delta_temperature_ai = -(energy_ai)
elif(direction == +1):
    delta_temperature_ai = +(energy_ai)

#Updating the new Server's Temperature when there is the AI
self.temperature_ai += delta_intrinsic_temperature + delta_temperature_ai

#Updating the new Server's Temperature when there is no AI
self.temperature_noai += delta_intrinsic_temperature

#GETTING GAME OVER
if(self.temperature_ai < self.min_temperature):
    if(self.train == 1):
        self.game_over = 1
    else:
        self.temperature_ai = self.optimal_temperature[0]
        self.total_energy_ai += self.optimal_temperature[0] - self.temperature_ai
elif(self.temperature_ai > self.max_temperature):
    if(self.train == 1):
        self.game_over = 1
    else:
        self.temperature_ai = self.optimal_temperature[1]
        self.total_energy_ai += self.temperature_ai - self.optimal_temperature[1]

#UPDATING THE SCORES
#Updating the total energy spent by the AI
self.total_energy_ai += energy_ai

#Updating tje total energy spent by the server's cooling system where there is no AI
self.total_energy_noai += energy_noai
```



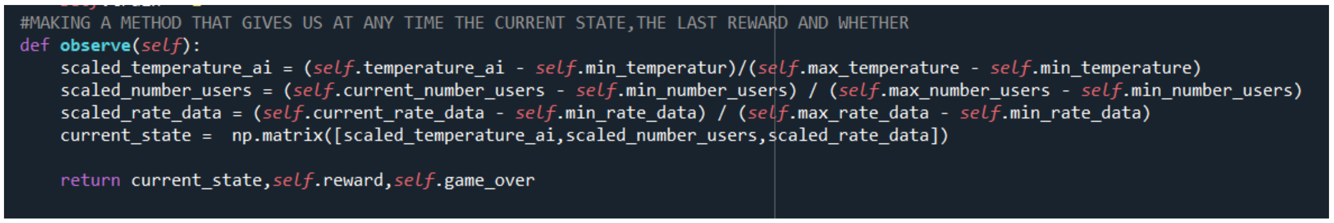
```
#SCALING THE NEXT STATE
scaled_temperature_ai = (self.temperature_ai - self.min_temperatur)/(self.max_temperature - self.min_temperature)
scaled_number_users = (self.current_number_users - self.min_number_users) / (self.max_number_users - self.min_number_users)
scaled_rate_data = (self.current_rate_data - self.min_rate_data) / (self.max_rate_data - self.min_rate_data)
next_state = np.matrix([scaled_temperature_ai,scaled_number_users,scaled_rate_data])

#RETURNING THE NEXT STATE, THE REWARD, AND GAME OVER
return next_state,self.reward,self.game_over
```

3. Making a method that resets the environment.



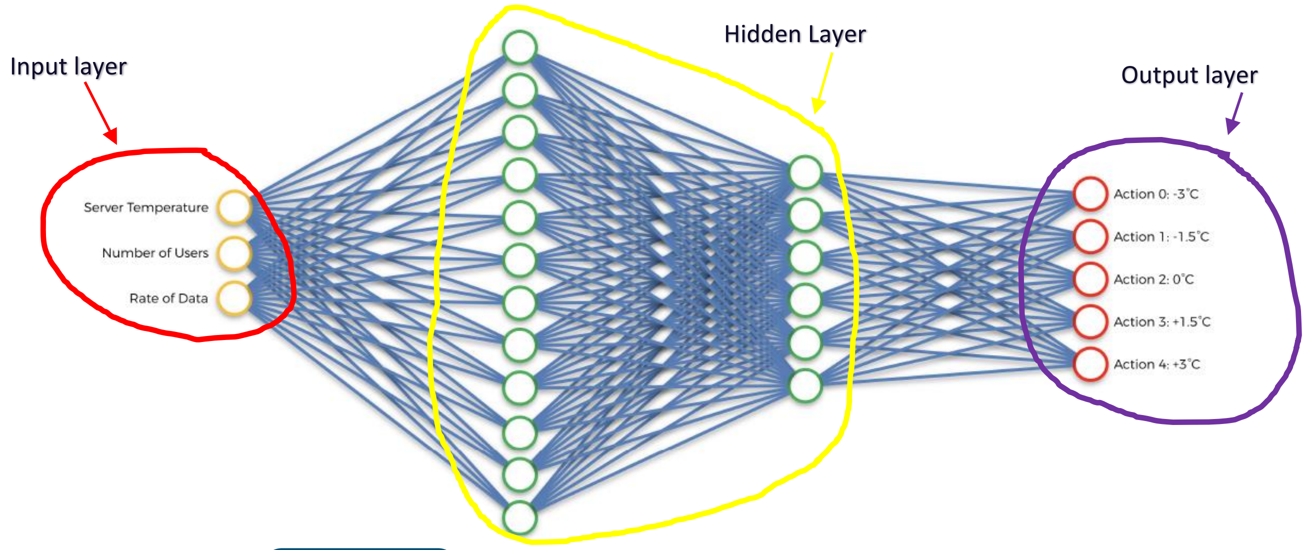
```
#MAKING A METHOD THAT RESETS THE ENVIRONMENT
def reset(self, new_month):
    self.atmospheric_temperature = self.monthly_atmospheric_temperature[new_month]
    self.initial_month = new_month
    self.current_number_users = self.initial_number_users
    self.current_rate_data = self.initial_rate_data
    self.intrinsic_temperature = self.atmospheric_temperature + 1.25 * self.current_number_users + 1.25 * self.current_rate_data
    self.temperature_ai = self.intrinsic_temperature
    self.temperature_noai = (self.optimal_temperature[0] + self.optimal_temperature[1]) / 2.0
    self.total_energy_ai = 0.0
    self.total_energy_noai = 0.0
    self.reward = 0.0
    self.game_over = 0
    self.train = 1
```

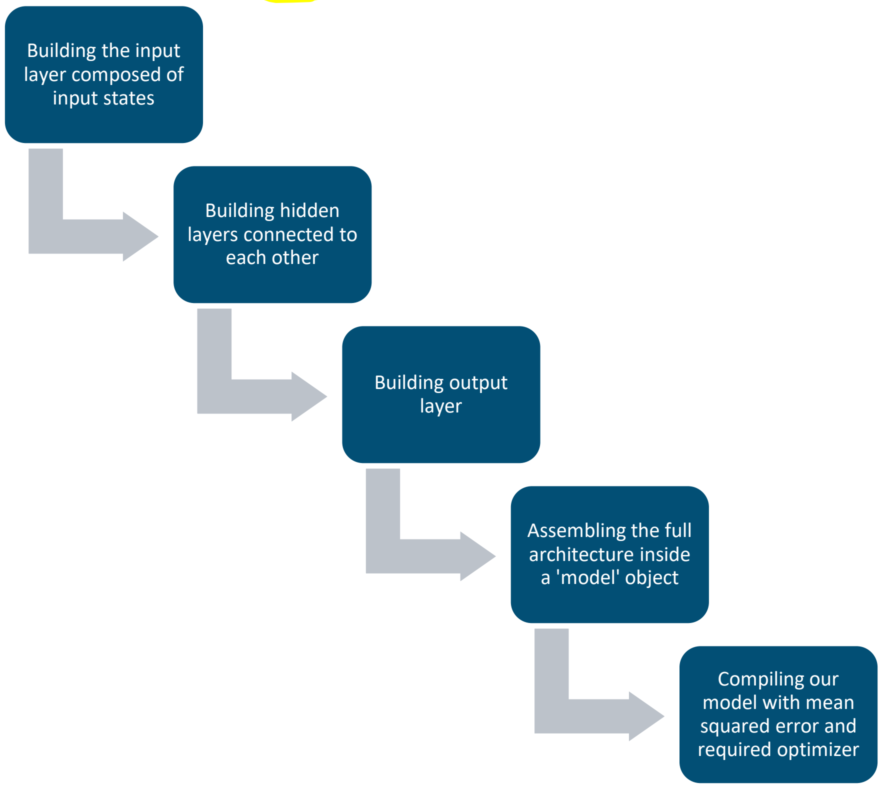
4. Making a method that gives us at any time the current state, the last reward obtained, and whether the game is over.

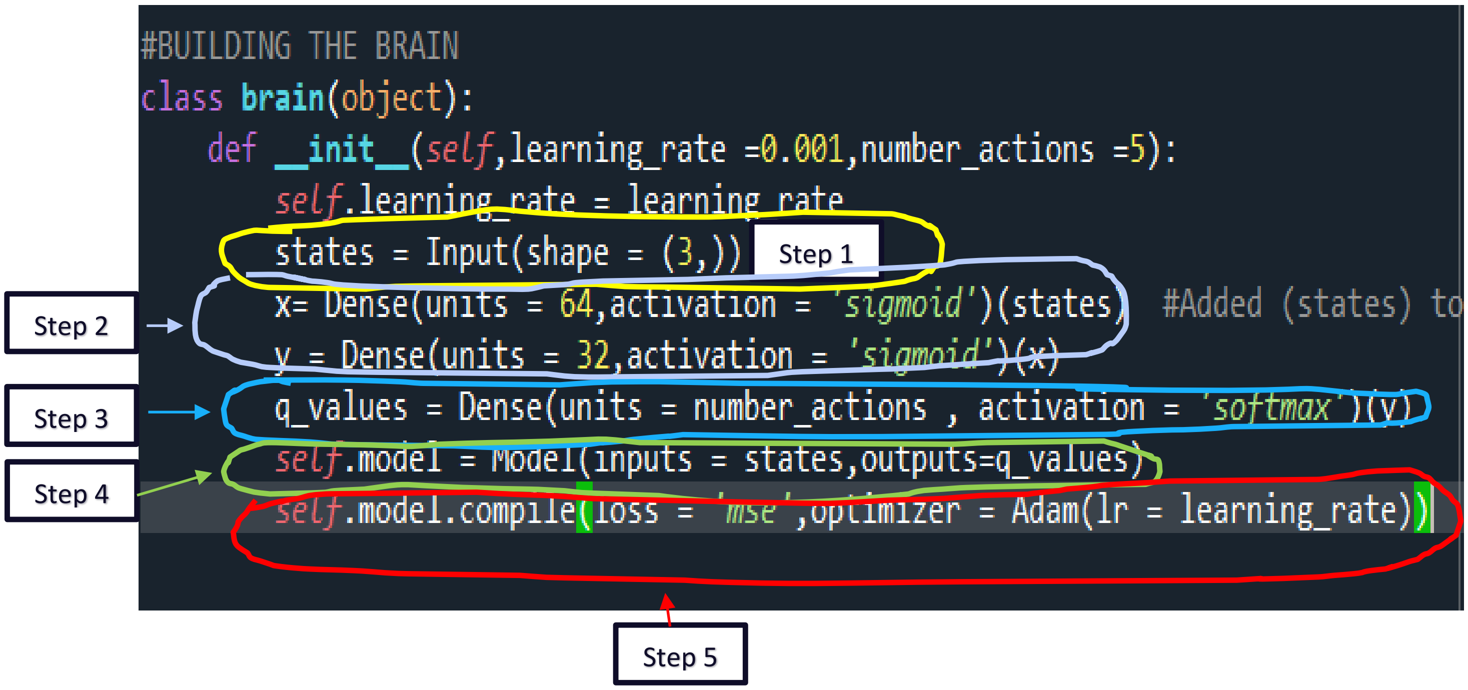


```
#MAKING A METHOD THAT GIVES US AT ANY TIME THE CURRENT STATE,THE LAST REWARD AND WHETHER
def observe(self):
    scaled_temperature_ai = (self.temperature_ai - self.min_temperatur)/(self.max_temperature - self.min_temperature)
    scaled_number_users = (self.current_number_users - self.min_number_users) / (self.max_number_users - self.min_number_users)
    scaled_rate_data = (self.current_rate_data - self.min_rate_data) / (self.max_rate_data - self.min_rate_data)
    current_state =  np.matrix([scaled_temperature_ai,scaled_number_users,scaled_rate_data])

    return current_state,self.reward,self.game_over
```

Step 2 : Building the Brain



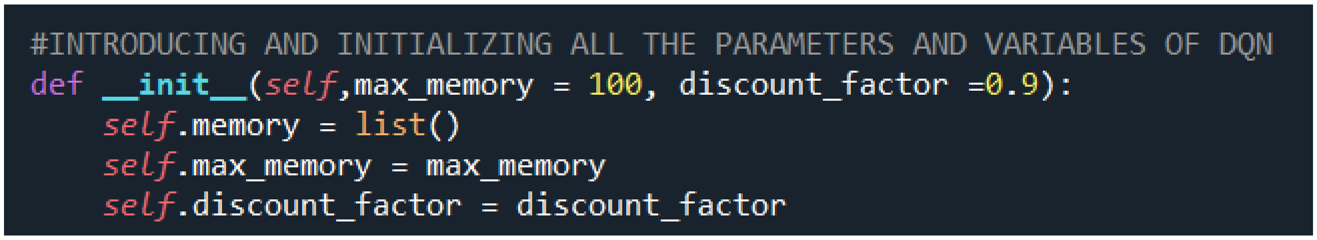

Building the input layer composed of input states
Building hidden layers connected to each other
Building output layer
Assembling the full architecture inside a 'model' object
Compiling our model with mean squared error and required optimizer



```
#BUILDING THE BRAIN
class brain(object):
    def __init__(self,learning_rate =0.001,number_actions =5):
        self.learning_rate = learning_rate
        states = Input(shape = (3,))
        x= Dense(units = 64,activation = 'sigmoid')(states)  #Added (states) to
        y = Dense(units = 32,activation = 'sigmoid')(x)
        q_values = Dense(units = number_actions , activation = 'softmax')(y)
        self.model = Model(inputs = states,outputs=q_values)
        self.model.compile(loss = 'mse',optimizer = Adam(lr = learning_rate))
```

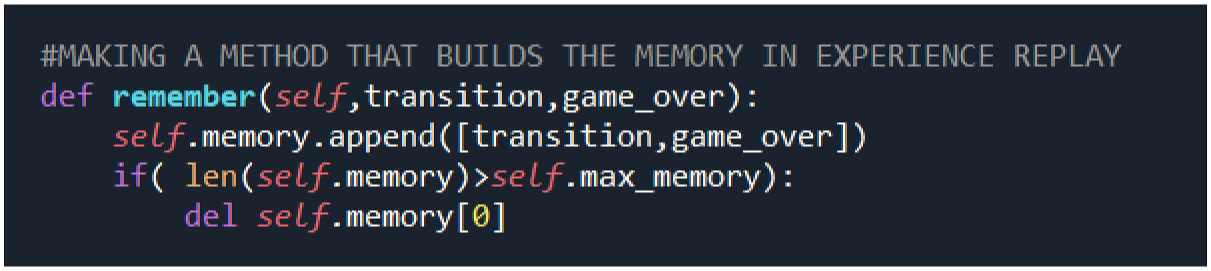
Step 3: Building DQN

1. Introducing and initializing all the parameters and variables of the DQN model.



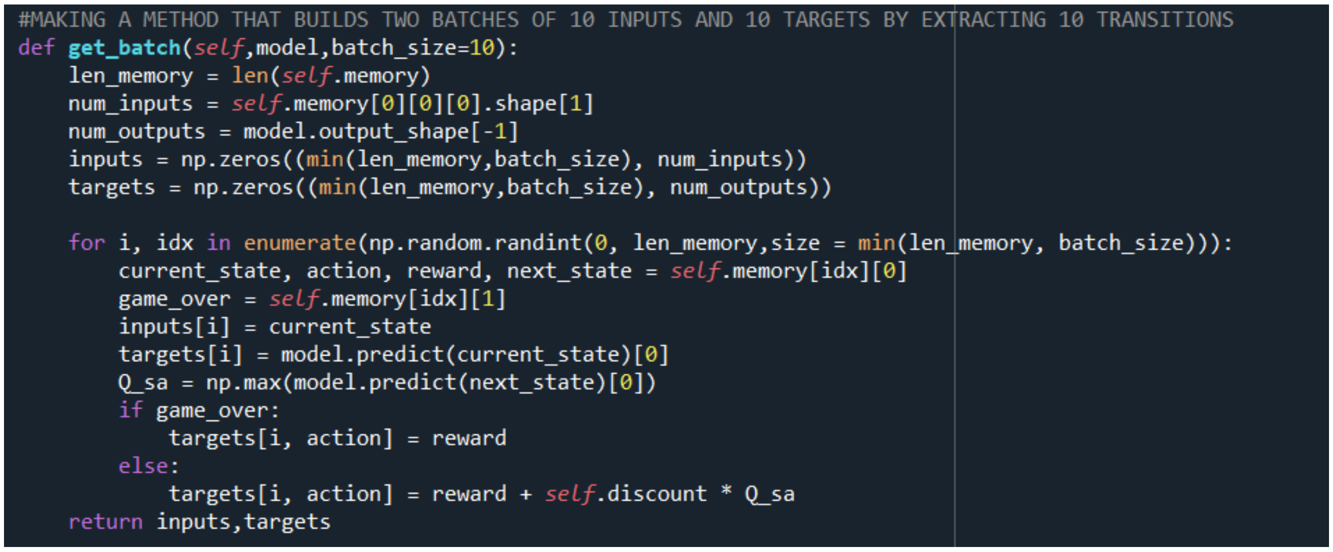
```
#INTRODUCING AND INITIALIZING ALL THE PARAMETERS AND VARIABLES OF DQN
def __init__(self,max_memory = 100, discount_factor =0.9):
    self.memory = list()
    self.max_memory = max_memory
    self.discount_factor = discount_factor
```

2. Making a method that builds the memory in Experience Replay.



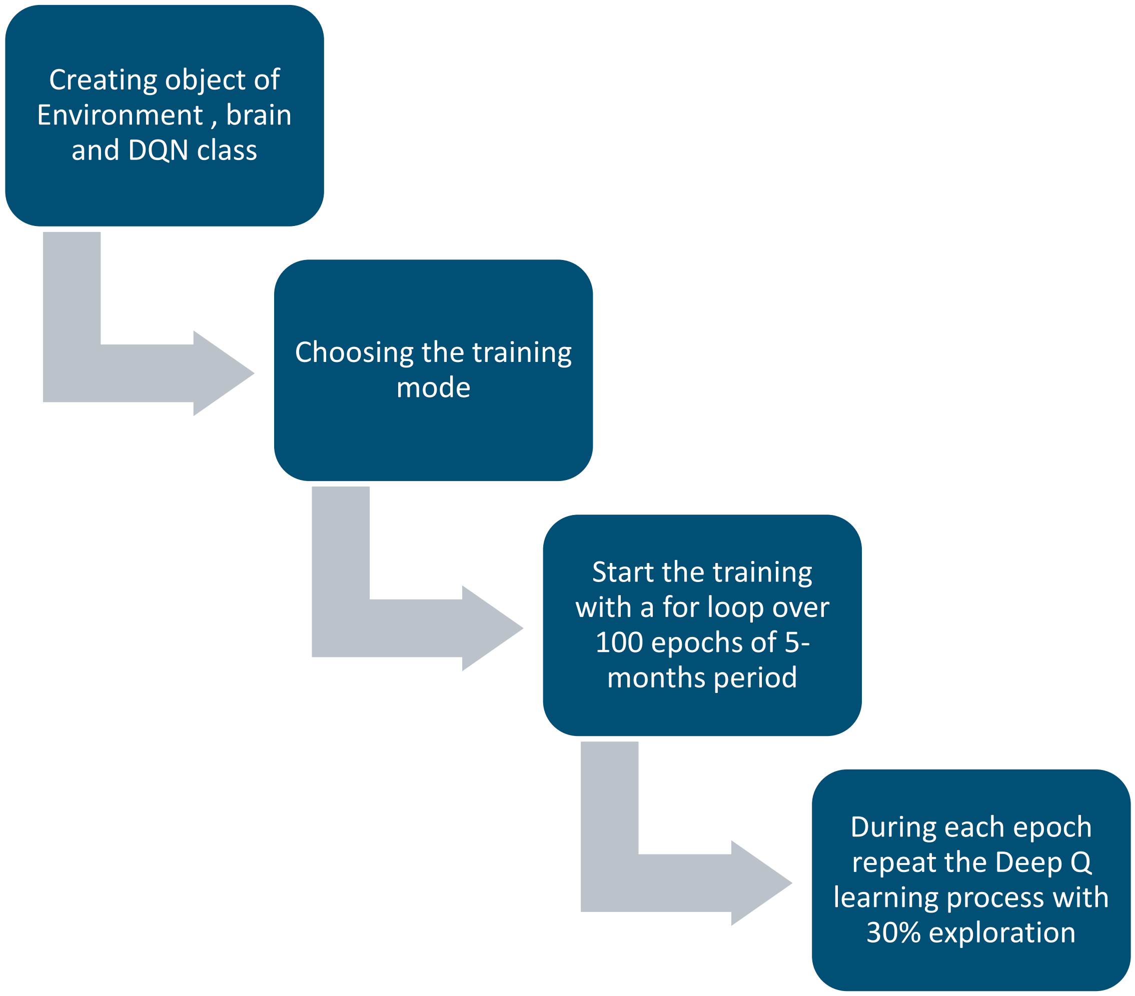
```
#MAKING A METHOD THAT BUILDS THE MEMORY IN EXPERIENCE REPLAY
def remember(self,transition,game_over):
    self.memory.append([transition,game_over])
    if( len(self.memory)>self.max_memory):
        del self.memory[0]
```

3. Making a method that builds and returns two batches of 10 inputs and 10 targets

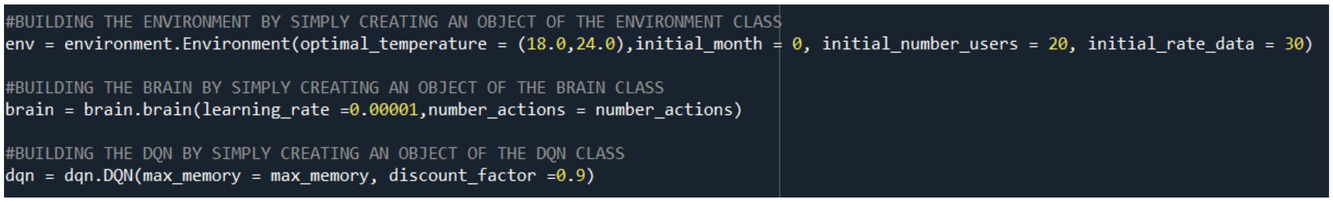


```
#MAKING A METHOD THAT BUILDS TWO BATCHES OF 10 INPUTS AND 10 TARGETS BY EXTRACTING 10 TRANSITIONS
def get_batch(self,model,batch_size=10):
    len_memory = len(self.memory)
    num_inputs = self.memory[0][0][0].shape[1]
    num_outputs = model.output_shape[-1]
    inputs = np.zeros((min(len_memory,batch_size), num_inputs))
    targets = np.zeros((min(len_memory,batch_size), num_outputs))

    for i, idx in enumerate(np.random.randint(0, len_memory,size = min(len_memory, batch_size))):
        current_state, action, reward, next_state = self.memory[idx][0]
        game_over = self.memory[idx][1]
        inputs[i] = current_state
        targets[i] = model.predict(current_state)[0]
        Q_sa = np.max(model.predict(next_state)[0])
        if game_over:
            targets[i, action] = reward
        else:
            targets[i, action] = reward + self.discount * Q_sa
    return inputs,targets
```

Step 4: Training the AI

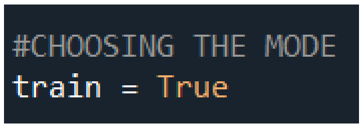

Creating object of Environment , brain and DQN class
Choosing the training mode
Start the training with a for loop over 100 epochs of 5-months period
During each epoch repeat the Deep Q learning process with 30% exploration

- **Creating object of Environment , brain and DQN class**



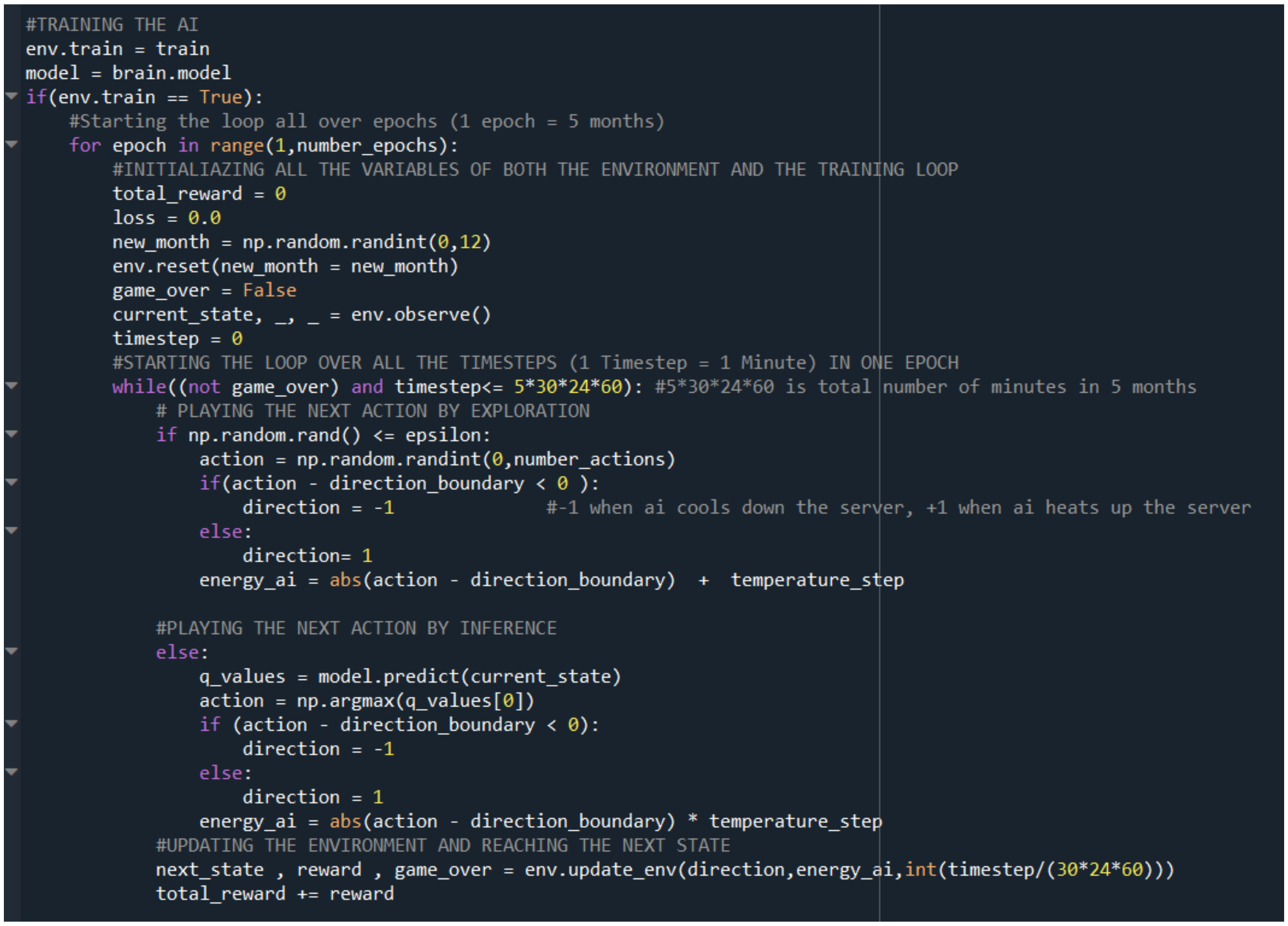
```
#BUILDING THE ENVIRONMENT BY SIMPLY CREATING AN OBJECT OF THE ENVIRONMENT CLASS
env = environment.Environment(optimal_temperature = (18.0,24.0),initial_month = 0, initial_number_users = 20, initial_rate_data = 30)

#BUILDING THE BRAIN BY SIMPLY CREATING AN OBJECT OF THE BRAIN CLASS
brain = brain.brain(learning_rate =0.00001,number_actions = number_actions)

#BUILDING THE DQN BY SIMPLY CREATING AN OBJECT OF THE DQN CLASS
dqn = dqn.DQN(max_memory = max_memory, discount_factor =0.9)
```

- **Choosing the training mode**



```
#CHOOSING THE MODE
train = True
```

- **Start the training with a for loop over 100 epochs of 5-months period**



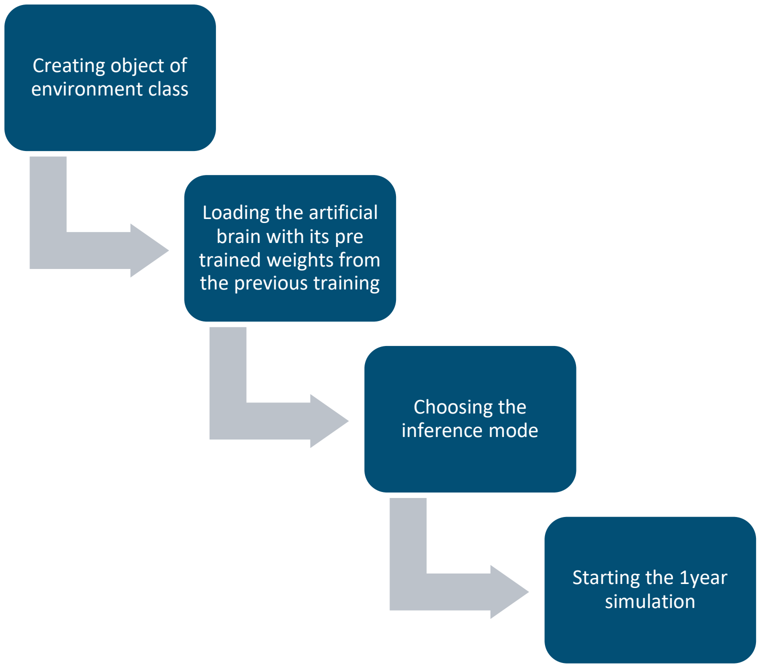
```
#TRAINING THE AI
env.train = train
model = brain.model
if(env.train == True):
    #Starting the loop all over epochs (1 epoch = 5 months)
    for epoch in range(1,number_epochs):
        #INITIALIAZING ALL THE VARIABLES OF BOTH THE ENVIRONMENT AND THE TRAINING LOOP
        total_reward = 0
        loss = 0.0
        new_month = np.random.randint(0,12)
        env.reset(new_month = new_month)
        game_over = False
        current_state, _, _ = env.observe()
        timestep = 0
        #STARTING THE LOOP OVER ALL THE TIMESTEPS (1 Timestep = 1 Minute) IN ONE EPOCH
        while((not game_over) and timestep<= 5*30*24*60): #5*30*24*60 is total number of minutes in 5 months
            # PLAYING THE NEXT ACTION BY EXPLORATION
            if np.random.rand() <= epsilon:
                action = np.random.randint(0,number_actions)
                if(action - direction_boundary < 0 ):
                    direction = -1              #-1 when ai cools down the server, +1 when ai heats up the server
                else:
                    direction= 1
                energy_ai = abs(action - direction_boundary)  +  temperature_step

            #PLAYING THE NEXT ACTION BY INFERENCE
            else:
                q_values = model.predict(current_state)
                action = np.argmax(q_values[0])
                if (action - direction_boundary < 0):
                    direction = -1
                else:
                    direction = 1
                energy_ai = abs(action - direction_boundary) * temperature_step
            #UPDATING THE ENVIRONMENT AND REACHING THE NEXT STATE
            next_state , reward , game_over = env.update_env(direction,energy_ai,int(timestep/(30*24*60)))
            total_reward += reward
```

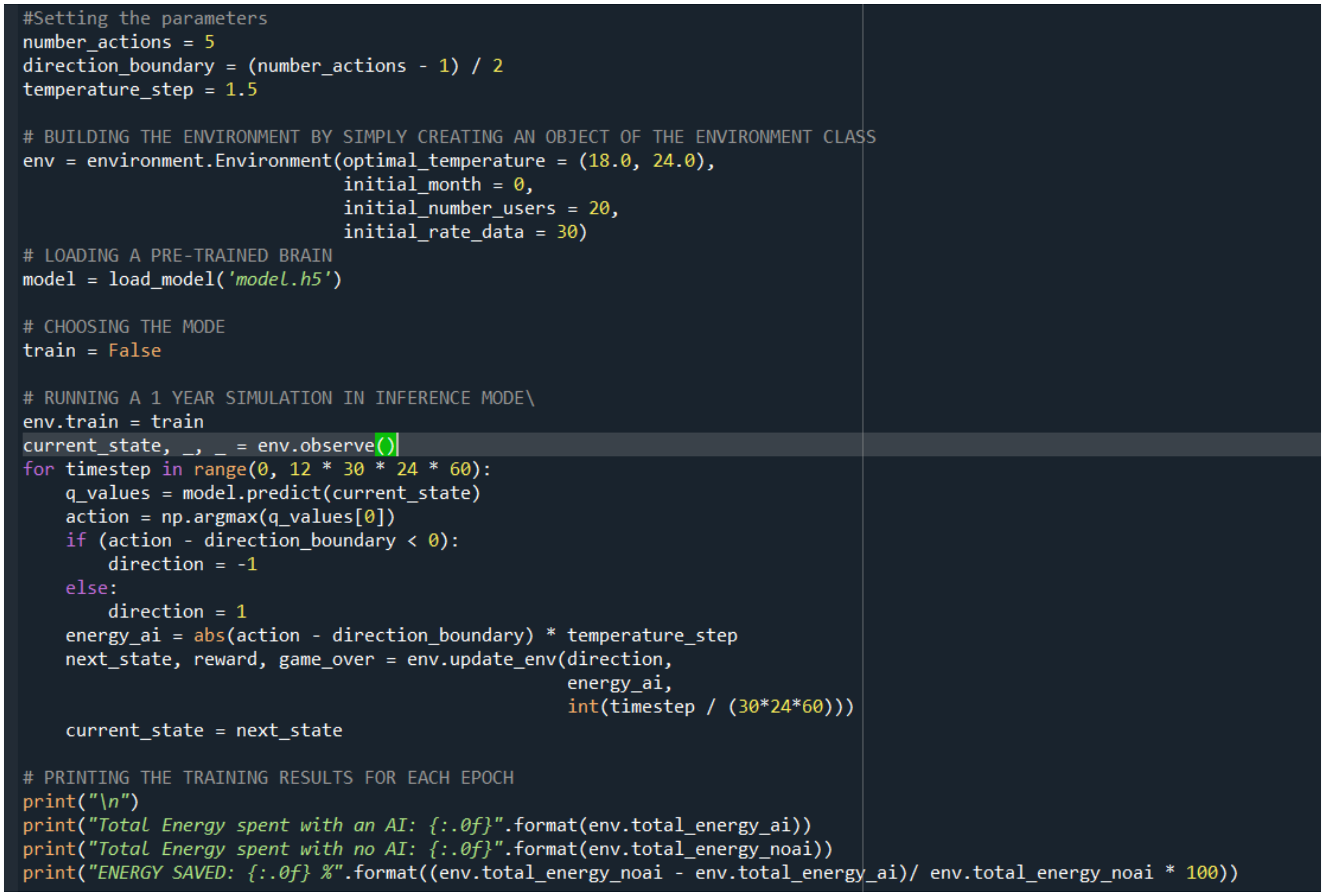
```
#STORING THIS NEW TRANSITION INTO THE MEMORY
dqn.remember([current_state,action,reward,next_state],game_over)

#GATHERING IN TWO SEPARATE BATCHES THE INPUTS AND THE TARGETS
inputs,targets = dqn.get_batch(model,batch_size= batch_size)

#COMPUTING THE LOSS OVER THE TWO WHOLE BATCHES OF INPUTS AND TARGETS
loss += model.train_on_batch(inputs, targets)
timestep += 1
current_state = next_state
```

- **During each epoch repeat the Deep Q learning process with 30% exploration**

```
#PRINTING THE RESULTS FOR EACH EPOCH
print('\n')
print("Epoch : {:03d}/{:03d}".format(epoch, number_epochs))
print("Total Energy spent with an AI: {:.0f}".format(env.total_energy_ai))
print("Total Energy spent with no AI: {:.0f}".format(env.total_energy_noai))

#SAVING OUR MODEL
model.save('model.h5')
```

Step 5: Testing our AI on Server


Creating object of environment class
Loading the artificial brain with its pre trained weights from the previous training
Choosing the inference mode
Starting the 1year simulation



```python
#Setting the parameters
number_actions = 5
direction_boundary = (number_actions - 1) / 2
temperature_step = 1.5

# BUILDING THE ENVIRONMENT BY SIMPLY CREATING AN OBJECT OF THE ENVIRONMENT CLASS
env = environment.Environment(optimal_temperature = (18.0, 24.0),
                              initial_month = 0,
                              initial_number_users = 20,
                              initial_rate_data = 30)
# LOADING A PRE-TRAINED BRAIN
model = load_model('model.h5')

# CHOOSING THE MODE
train = False

# RUNNING A 1 YEAR SIMULATION IN INFERENCE MODE\
env.train = train
current_state, _, _ = env.observe()
for timestep in range(0, 12 * 30 * 24 * 60):
    q_values = model.predict(current_state)
    action = np.argmax(q_values[0])
    if (action - direction_boundary < 0):
        direction = -1
    else:
        direction = 1
    energy_ai = abs(action - direction_boundary) * temperature_step
    next_state, reward, game_over = env.update_env(direction,
                                                   energy_ai,
                                                   int(timestep / (30*24*60)))
    current_state = next_state

# PRINTING THE TRAINING RESULTS FOR EACH EPOCH
print("\n")
print("Total Energy spent with an AI: {:.0f}".format(env.total_energy_ai))
print("Total Energy spent with no AI: {:.0f}".format(env.total_energy_noai))
print("ENERGY SAVED: {:.0f} %".format((env.total_energy_noai - env.total_energy_ai)/ env.total_energy_noai * 100))
```

Hence what we have built is surely excellent for our business client, as our AI will save them a lot of costs! Indeed, remember that thanks to our object oriented structure (working with classes and objects), we can very easily take our objects created in this implementation that we did for one server, and then plug them into other servers, so that in the end we end up saving the total energy consumption of a whole data center! That's how Google saved billions of dollars in energy related costs, thanks to their DQN model built by Deep Mind AI.