1. CLASS Agent
    a. Constructor Method:
        i. Parameters: self, neighbors, x, y, BaseStation, Batchsize (default 64)

        ii. The instance variables for the agent are set in this method:

            (a) **Queue**: The FIFO queue of all packets.
            (b) **Neighbours**: The list of all neighbors of the agent. From params.
            (c) **DQN_object**: The deep Q network for the agent.
            (d) **Position**: (x, y) Coordinates of the agent in the grid. From params.
            (e) **Batchsize**: The number of experiences from the buffer used to train the **DQN**. Default 64 experiences from params.
            (f) **State size** - Size of state, set to 1.
            (g) **Action size** - Size of action, set to 1.
            (h) **Target Base station**: Base station object to which packets are going.
            (i) **Latest loss** - The most recent value of the loss function calculated from training of the policy network of the agent. Set to 0.
            (j) **Losses** - The list of all losses during training.
            (k) **Latest_queue** - The queue after each run of the network.
            (l) **q_values** - The q values that are appended after every run.

        iii. Returns: None (since it is constructor method)

    b. getCurrentState
        i. Parameters: self (no external parameters)
        ii. Logic:
            The **state** of every agent consists of the **size of its queue**, the **size of its neighbors' queues** (the queue length of a base station is 0) and the **ttl of the packet at the head of the queue.** If there are no packets, the maximum TTL of all packets is returned.
        iii. Returns: the list containing the **state**.

    c. initDQN
        i. Parameters: self, device (CPU or GPU)
        ii. Logic: Initializes the deep Q Network for each agent by passing the state and action size instance variables. This is stored in the DQN_object instance.
        iii. Returns: None

d. loadModel
    i. Parameters: self, filename (for configuration)
    ii. Logic: calls loadModel() on the DQN_object with the file name.
    iii. Returns: None.

e. pushQueue
    i. Parameters: self, packet to push
    ii. Logic: Appends the packet into the queue of the agent.
    iii. Returns: True if the push was successful, false if Queue is full.

f. popQueue
    i. Parameters: self (no external parameters)
    ii. Logic: Pops the queue and returns the packet.
    iii. Returns: -1 if the queue is empty, otherwise the packet.

g. getTopPacket
    i. Parameters: self
    ii. Logic: Fetches the last packet in the latest queue.
    Returns: –1 if the latest queue is empty, otherwise the last packet in latest queue.

h. nextAction
    i. Parameters: self, current state of agent
    ii. Logic: Uses the DQN_object policy network and the given state to obtain a new state using the selectAction() method on the DQN Object.
    iii. Returns: the new state

i. trainAgent
    i. Parameters: self, state, action, nextState, reward
    ii. Logic: Stores the state, action, nextState, reward tuple in the memory of DQN Object, then calls the learn method using the batchsize which will return the training loss, which is to be stored in the latest_loss object of the DQN.

j. saveLoss
    i. Parameters: self
    ii. Logic: Appends the training loss into the losses array for the agent.
    iii. Returns: None

k. getLoss
    i. Parameters: self
    ii. Logic: returns the losses array for the agent.
    iii. Returns: losses

l. acceptPacket
    i. Parameters: self, packet object
    ii. Logic: add the packet to the latest queue array.
    iii. Returns: None (How does this relate to pushQueue?)

m. getPosition
    i. Parameters: self
    ii. Logic: returns the position tuple of the agent.
    iii. Returns: position

    .

n. addNeighbour
    i. Parameters: self, neighbor to add.
    ii. Logic: add the neighbor UAV object to the neighbors list, then increment action and state sizes, since the neighbor information is used to formulate state and action for the given UAV.
    iii. Returns: None

o. Identification methods: isUAV, isBase, isIOT, isBaseStation, isBlock
    i. Parameters: self
    ii. Logic: Identifies the class of node.
    iii. Returns true only in case of isUAV, otherwise returns false.

p. getReward (important)
    i. Parameters: self (no external parameters)
    ii. Logic:
        - First, get the ttl of the top packet using getTopPacket.get_ttl()
        - Calculate the unscaled reward using agent_to_agent_scale * the Q value from the dqn_object's getQValue method (from the Q network) by passing the current state using getCurrentState.
        - Scale the reward based on the type - square, exponential, or fractional. All of these use the ttl of the top packet and the maximum ttl as explained in the paper.

iii. Returns: the scaled reward.

q. Run (very important)
   i. Parameters: self, train (set to true by default)
   ii. Logic:
      - Uses self.getCurrentState() and stores in a variable state.
      - Next, decrease the time to live of each packet using decrease_ttl() method of packet.
      - Find the packet at the head of the queue and obtain it using popQueue. If -1 is returned, then the queue is empty, therefore no reward (0).
      - Then, it selects the next action using the select_action() method of the dqn_object. This returns the index of the neighbor to which the packet has to be forwarded. If it returns the number of neighbors, then the packet is dropped. Check the
      - The Q value of the current state is obtained using the getQValue method on DQN and stored in the q_values list.
      - CASES:
         a. If the next action is the number of neighbors, the packet is dropped, and the experience is sent to trainAgent. A value of -100 is returned to indicate this.
         b. If the time to live of the top packet is negative, then the packet has expired and is not sent. The experience is sent to trainAgent and -100 is returned to indicate this.
         c. None of the above adverse cases are true.
            i. The packet is sent to the neighbor. This is done by accessing the neighbor object using self.neighbors[nextaction] and calling acceptPacket for this neighbor with the topPacket as the parameter.
            ii. Uses self.getCurrentState() again to obtain the next state. **Recall that this contains the queue length of the neighbors as well.** The queue length of the neighbor that was sent the packet is increment using the index ‘nextAction + 1’ (+1 because the 0th index is the queue length of the self agent)
            iii. There are two rewards: one is the reward that is being stored in the memory of the DQN. This is obtained using getReward() on the neighbor. This reward is divided by the manhattan distance between the sender of the packet and the basestation (**as explained in the paper)** and sent to trainAgent (memory of DQN)

iv. The second reward is what is returned by run. It is calculated using the time to live of the topPacket/maximum time to live, and scaled by the Manhattan distance. If the distance is **one** or **two** this indicates that the sending node is neighboring the base station and that the packet was sent.**(clarify and update this)**

- When the network is in testing mode, the status of what is happening is being printed. This includes the position, state, ttl of top packet, and whether the packet was dropped, expired, or sent to a neighbor (the neighbor is also printed).

iii. Returns:

- 0 if the queue is empty.
- -100 if the packet was dropped or if it expired.
- The reward_to_be_returned if the packet was sent.

r. getVal

i. Parameters: self (no external parameters)

ii. Logic:

- Returns the queue length of the given agent.

iii. Returns: Length of queue.

s. Reset

i. Parameters: self

ii. Logic:

- Clears the queue for testing.

iii. Returns: None

t. Update_state

i. Parameters: self

ii. Logic:

- Flushes everything in latest_queue to the parent instance variable queue.
- Resets latest_queue.

2. CLASS BaseStation

a. Constructor method (__init__)

i. Parameters: self, x, y (position)

ii. Logic: Sets the following instance variables for the base station:

- Position: (x, y)

- Packets received (packetRecv): 0
- List of packets received: []
- Total ttl: 0 (Used to calculate the average ttl later on)

iii. Returns: None

b. acceptPacket
   i. Parameters: self, packet object that is being received
   ii. Logic:
      - Increments the packet counter and adds the packet object to the array of packets received.
      - Accesses the ttl of the packet using get_ttl() and adds it to total ttl.
   iii. Returns: None

c. getReward
   i. Parameters: self
   ii. Logic: Used for sending the reward to the agent which sent the packet to it. Same functionality as in getReward in agent, but does not use the Q network to determine the unscaled reward. It finds the ttl of the latest packet received and uses **scale_base_reward** to find a scaled reward using **scale_base_reward** * ttl**2, and returns this scaled reward.
   iii. Returns: scaled reward.

d. getPosition
   i. Parameters: self
   ii. Logic: Returns the coordinates of the base-station.
   iii. Returns: coordinates in tuple form.

e. Identification functions: isUAV, isBase, isIot, isBlock
   i. Parameters: self
   ii. Logic: Identifies the type of node: In this case, only isBase returns true.
   iii. Returns: true for isBase, false for all other methods.

f. getVal
   i. Parameters: self
   ii. Logic: Returns the number of packets received for the agent.
   iii. Returns: number of packets received.

g. Reset
    i. Parameters: self
    ii. Logic: Clears all instance variables except position for testing.
    iii. Returns: None.

3. CLASS Block
    a. Constructor Method (__init__)
        i. Parameters: position
        ii. Logic: Sets the position instance of the block node.
        iii. Returns: Nothing.

    b. Identification functions (isUAV, isBase, isIOT, isBlock)
        i. Parameters: self
        ii. Logic: Returns true for isBlock, false for everything else.
        iii. Returns: (specified above)

    c. Reset
        i. Parameters: self
        ii. Logic: Called while resetting the network for testing. Does nothing since block has no functionality but function is still needed to prevent attributeError as every single node's reset() method is called at once.
        iii. Returns: none

4. CLASS IotNodes
    a. Constructor Method (__init__)
        i. Parameters: rate, default time to live, (x, y) position, transmission rate (default set to 10)
        ii. Logic: Sets the following instance variables for iot node objects:
            - Rate: from params
            - def_ttl: from params
            - position: from params
            - Neighbors: empty array (this is populated in generate() method of map)
            - Total_packets: number of packets generated, set to 0.
            - Queue: FIFO for sending packets, empty array.
        iii. Returns: None

    b. generatePackets

i. Parameters: self
ii. Logic: It is called for every unit time. It generates packets equal to the rate of packet generation. It then increments total_packets and appends those many packets to the back of the queue instance variable.
iii. Returns: None.

c. findNeighbour
   i. Parameters: self
   ii. Logic: It returns (i) the baseStation if it is one of the neighbors or (ii) the agent of the least queue size.
      - First searches for a neighbor that is the base station. If it finds the baseStation as a neighbor then it returns the baseStation object.
      - Otherwise it will gather the queue sizes using agent.getVal over all neighbors and then search for the minimum queue size and return the agent object with this minimum queue size.
   iii. Returns: either the baseStation or the agent object with the least queue size.

d. Identification functions: isUAV, isBase, isIot, isBaseStation, isBlock
   i. Parameters: self
   ii. Logic: Used to identify the type of node.
   iii. Returns: true for IOT, false for everything else.

e. run (very important)
   i. Parameters: None
   ii. Logic:
      - Run is called at every time step in the program.
      - It calls generatePacket() to generate self.rate number of packets and put them in the queue.
      - Then for each of those packets, it finds a neighbor. If there are no neighbors, the loop continues without doing anything. Otherwise, if the queue is not empty (which is a given since generatePacket() was called just above it) then it uses findNeighbor to locate the Neighbor with the least queue size and transmits the packet using acceptPacket() with the packet at the head of the queue passed into this (self.queue.pop(0)).
   iii. Returns: None

f. getPosition
   i. Parameters: Self

ii. Logic: Specifies the coordinates of the given IotNode.
iii. Returns: It returns the position tuple (x, y) of the IotNode.

g. addNeighbor
   i. Parameters: self, neighbor to be added
   ii. Logic: Takes the neighbor parameter and appends it to the list of neighbors.
   iii. Returns: None.

h. getVal
   i. Parameters: self
   ii. Logic: Returns the total number of packets generated (total_packets instance variable).
   iii. Returns: total number of packets generated.

i. getQueueSize
   i. Parameters: self
   ii. Logic: returns the number of packets in queue for the IoT node.
   iii. Returns: number of packets in the queue (len of self.queue)

j. Reset
   i. Parameters: self
   ii. Logic: Resets the queue of packets generated and total packets generated to start testing mode.
   iii. Returns: None.

5. CLASS packet
   a. Constructor method (__init__)
      i. Parameters: time to live
      ii. Logic: Creates the following instance variables for each packet:
         - Path: an empty array that documents the path of each packet as it is transmitted using acceptPacket between node objects.
         - Time to live: keeps track of how many times packet has been transmitted. If ttl < 0 packet expires.
      iii. Returns: None

   b. decrease_ttl():
      i. Parameters: self

ii. Logic: decreases the ttl by one every time packet is transmitted (in which case agent calls this method, refer to the documentation of agent above.)
iii. Returns: none

c. get_ttl:
   i. Parameters: self
   ii. Logic: returns the ttl of the packet.
   iii. Returns: ttl of the packet.

d. getPath:
   i. Parameters: self
   ii. Logic: Returns the array containing the path of the packet.
   iii. Returns: self.path.

e. addtoPath:
   i. Parameters: self, tuple (x, y) of agent that received the packet.
   ii. Logic: adds the tuple (x, y) to the packet's path instance variable.
   iii. Returns: None

6. CLASS Map
   a. Constructor method (__init__):
      i. Parameters: self, n (rows), m (cols), p (probability of finding UAV in a cell)
      ii. Logic: Creates the following instance variables for the Map Object:
         - self.n : number of rows (from params)
         - self.m : number of columns (from params)
         - self.p : prob of finding UAV (from params)
         - self.map: map matrix read from file (set to empty 2d array)
         - self.agents: list of all agents (UAVs) in the map
         - self.Iot_Nodes: list of all IoT Nodes in the map.
         - self.BaseStation: A null object. Will be set later.
      iii. Returns: None

   b. read:
      i. Parameters: none
      ii. Logic:
         - Creates a temporary 2D array map_.
         - Opens the serialized pickle object containing the map itself.
         - Reads the file sequentially, and depending on what character it encounters, it will create a node of that class (refer to the previous classes defined) and stored it in map_ AND append it to one of self.BaseStation, self.agents, or self.Iot_Nodes.

- Stops the operation when the dimensions have been reached.
- Then it populates the neighbors of all the IoTNodes and UAVs (Base stations and blocks don't have neighbors.)
- Then, finally, it will store the map_ temporary array in self.map.

iii. Returns: none.

c. generate:
   i. Parameters: self
   ii. Logic: This function is used when a random map has to be generated and not read from a file (for which the previous function was used).
      - First, it creates a temporary map_ array of size (self.m, self.n)
      - Next, it runs a loop over the dimensions, and for every iteration, it obtains a probability sampled from a uniform distribution between 0 and 1.
      - If this probability is less than or equal to p, then an agent must be placed at the coordinates. Map_ and self.agents are populated.
      - If the probability is more than p, then an IoT node must be placed at the coordinates. Map_ and self.Iot_nodes are populated.
      - Then it uses the same process to populate the neighbors of all the IoT Nodes and UAVs in the map.
      - Then it sets self.map to the map_ temp array.
   iii. Returns: None.

d. getBaseStation, getAgents, getIotNodes
   i. Parameters: self
   ii. Logic: Specifies the numbers of each object in the map.
   iii. Returns: self.BaseStation, self.agents, self.Iot_Nodes respectively.

e. renderMap
   i. Parameters: self
   ii. Logic: Creates a visual representation of the map and uses map[i][j].getVal() to also print the number of packets pertaining to each type of node (number generated at IoTs, number received at Base station, etc.). prints an X for the block.
   iii. Returns: None.

f. dummyMap:
   i. Parameters: self
   ii. Logic: Creates a dummy map of dimensions (1, 3) NOT (3,) and populates it in the following order: Base station, agent, ioT node.

g. resetAll
   i. Parameters: self

ii. Logic: calls the reset function on all the node objects. Refer to the above documentation to determine what the reset method does for each class of objects.
iii. Returns: None.

h. loadModel
   i. Parameters: folder name containing the model.
   ii. Logic: For all the UAVs, it calls loadModel (refer above documentation) using the folder name passed into the method.
   iii. Returns: None.

i. initModels
   i. Parameters: self, device (cpu or gpu)
   ii. Logic: For all the UAVs, it calls the initModel method (refer above documentation) using the device passed into the method.
   iii. Returns: None

7. Utilities
   a. getManhattanDistance
      i. Parameters: position of the sending node (x, y) and position of the base station (x, y).
      ii. Logic: Uses the formula for manhattan distance on the parameters.
      iii. Returns the calculated manhattan distance. Notice that it is used in the getReward method in the Agent class while obtaining a reward on packet transmission.