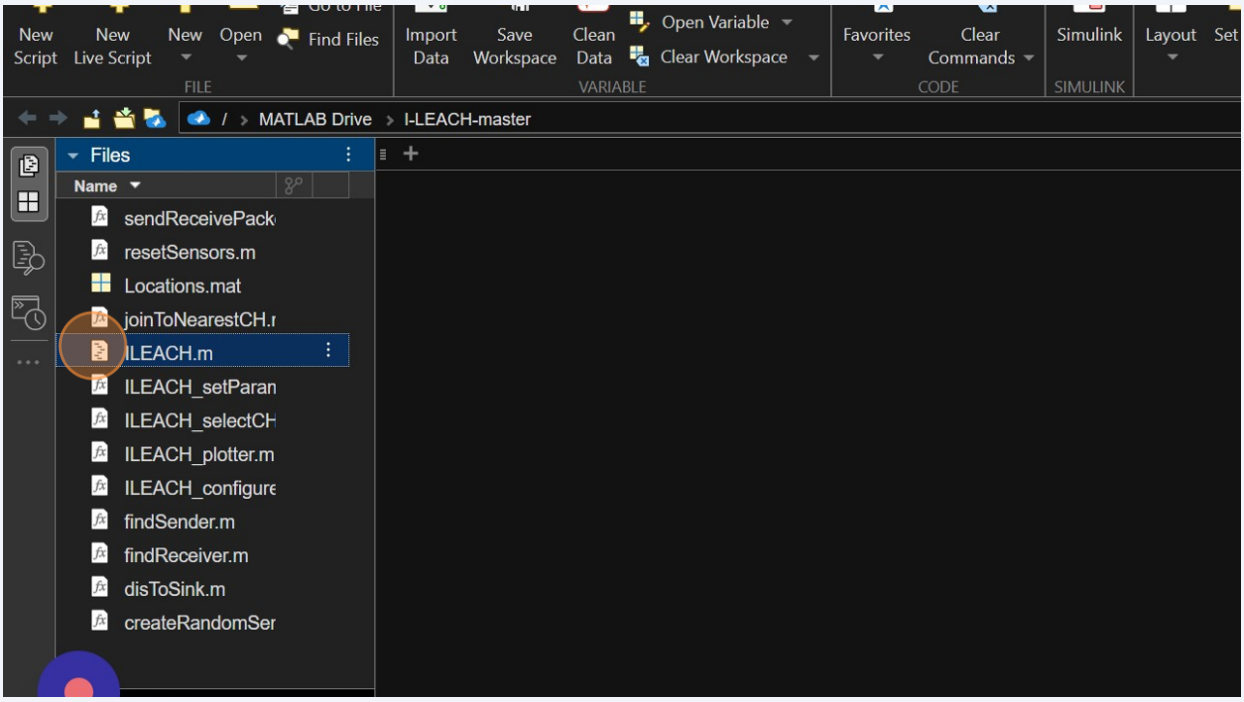
Step-by-Step Explanation of Implementing LEACH PROTOCOL in WSNs in MATLAB

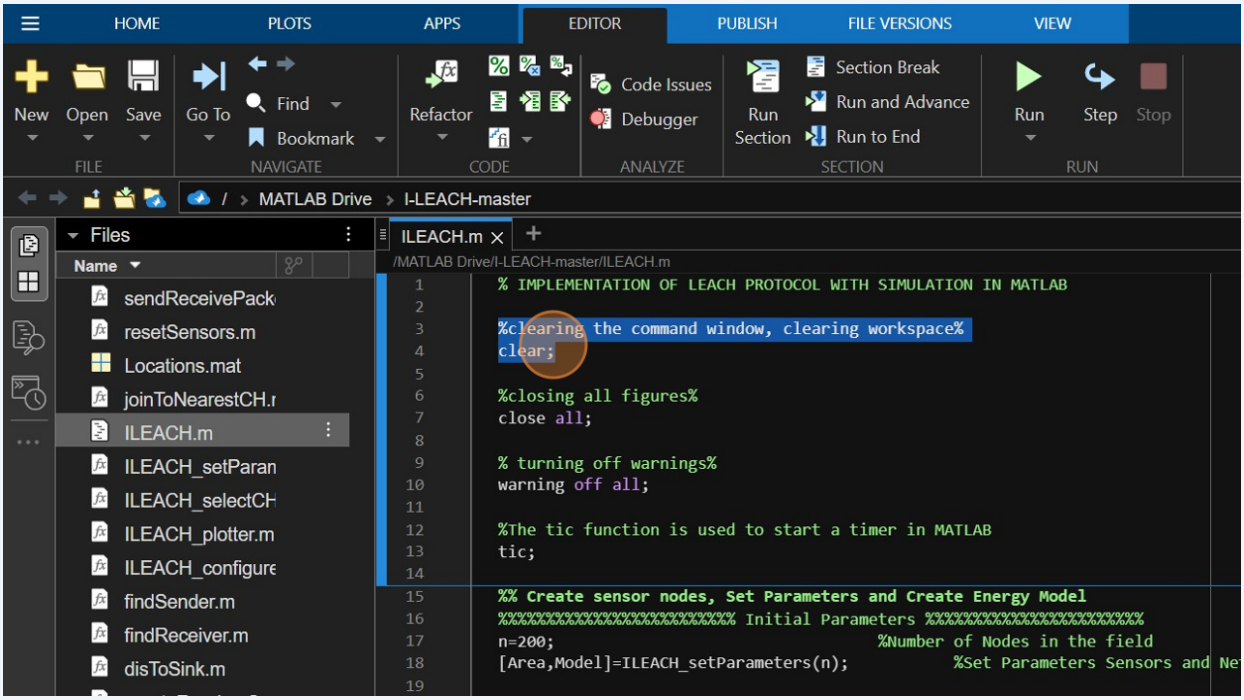
1 Navigate to https://matlab.mathworks.com/

2 Click "ILEACH.m" This is the Main file to Run the Leach Protocol

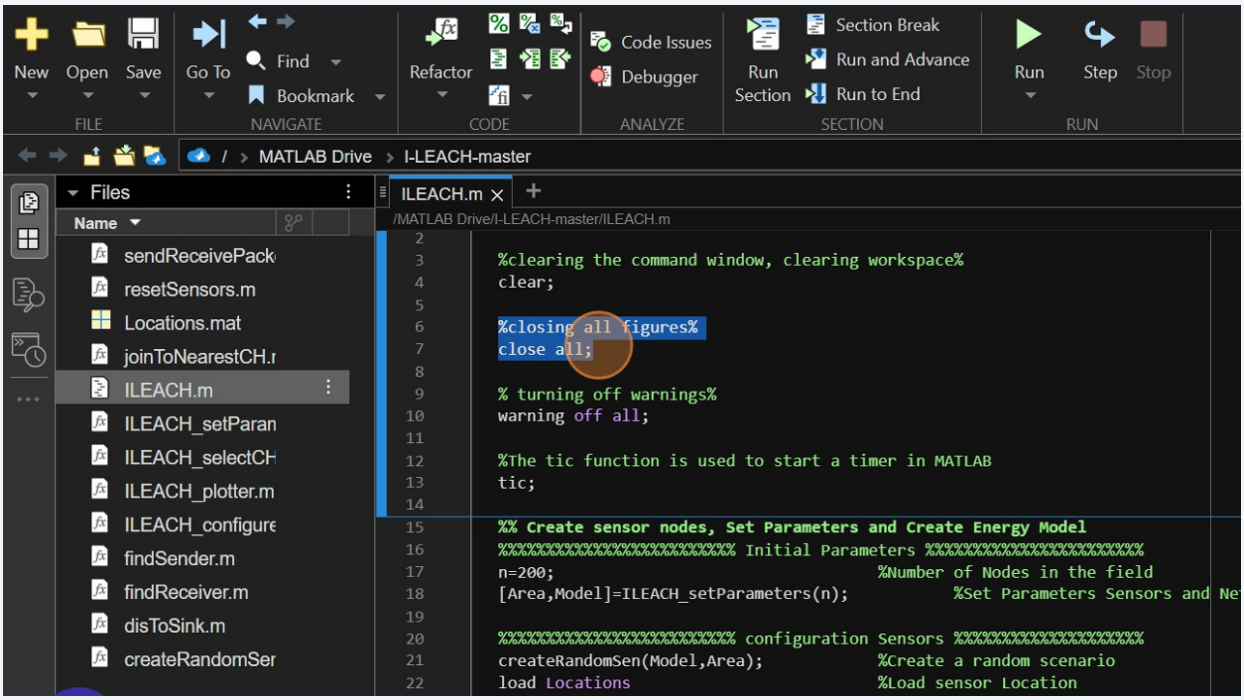


LEACH (Low-Energy Adaptive Clustering Hierarchy) is a widely used hierarchical clustering protocol designed for wireless sensor networks to prolong network lifetime by efficiently managing energy consumption. In LEACH, nodes self-organize into clusters, and each cluster has a rotating cluster head (CH) selected periodically to distribute the energy consumption across the network. Nodes take turns serving as CHs to reduce the load on individual nodes and mitigate energy depletion. LEACH employs a probabilistic approach for CH selection, where nodes decide to become CHs based on a pre-defined probability. Cluster heads collect data from their member nodes and transmit aggregated data to a sink, facilitating energy-efficient communication. The periodic rotation of CHs and distributed data aggregation contribute to balancing energy usage and extending the overall network lifespan in resource-constrained sensor environments.

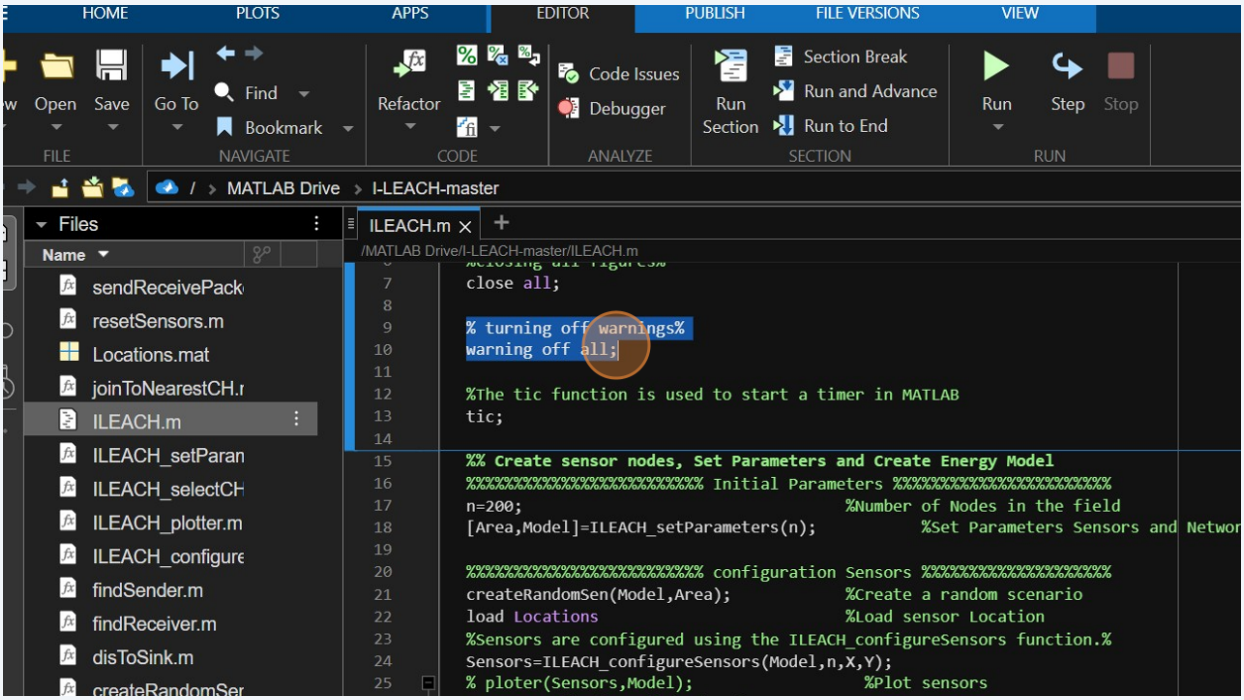
3 Firstly, the clear command written here is to clear the workspace in MATLAB



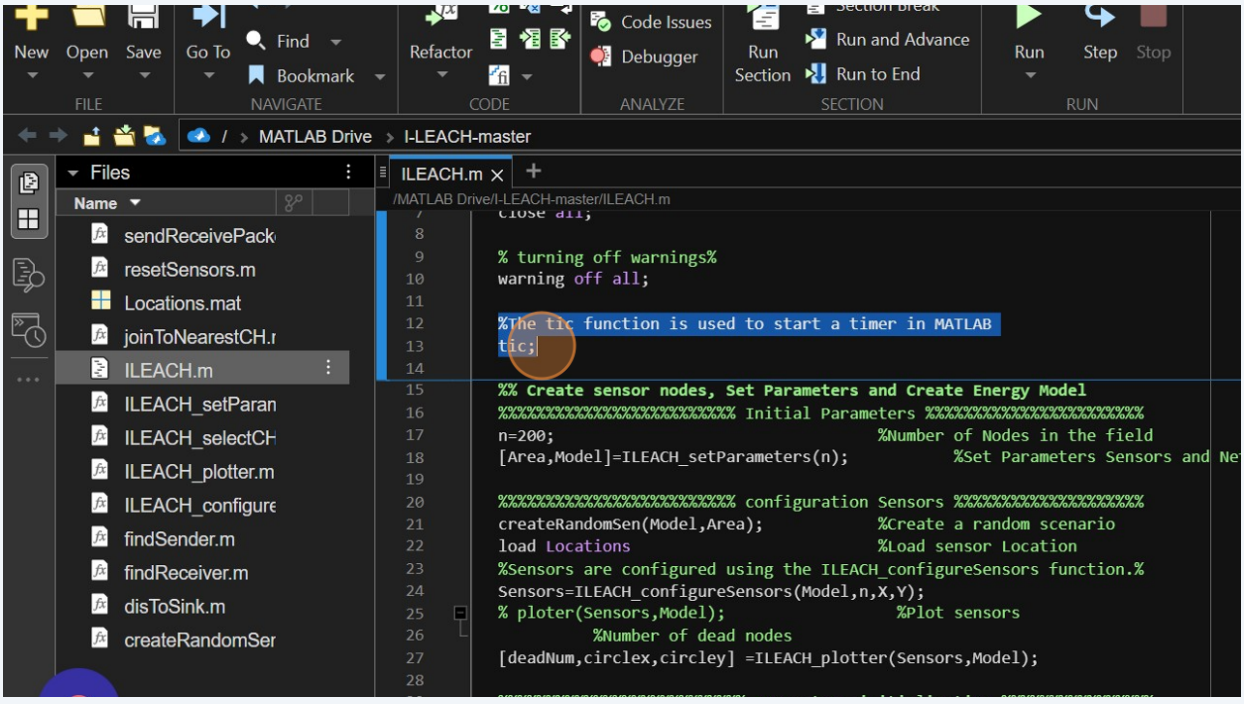
4 close all command here is used to close the figures of output simulation if previously opened.



5 By using warning off all;, the code is instructing MATLAB to turn off all warning messages, meaning that any warning that would typically be displayed during the execution of the script will be suppressed.



6 In MATLAB, the tic function is used to start a timer, marking the beginning of a time interval



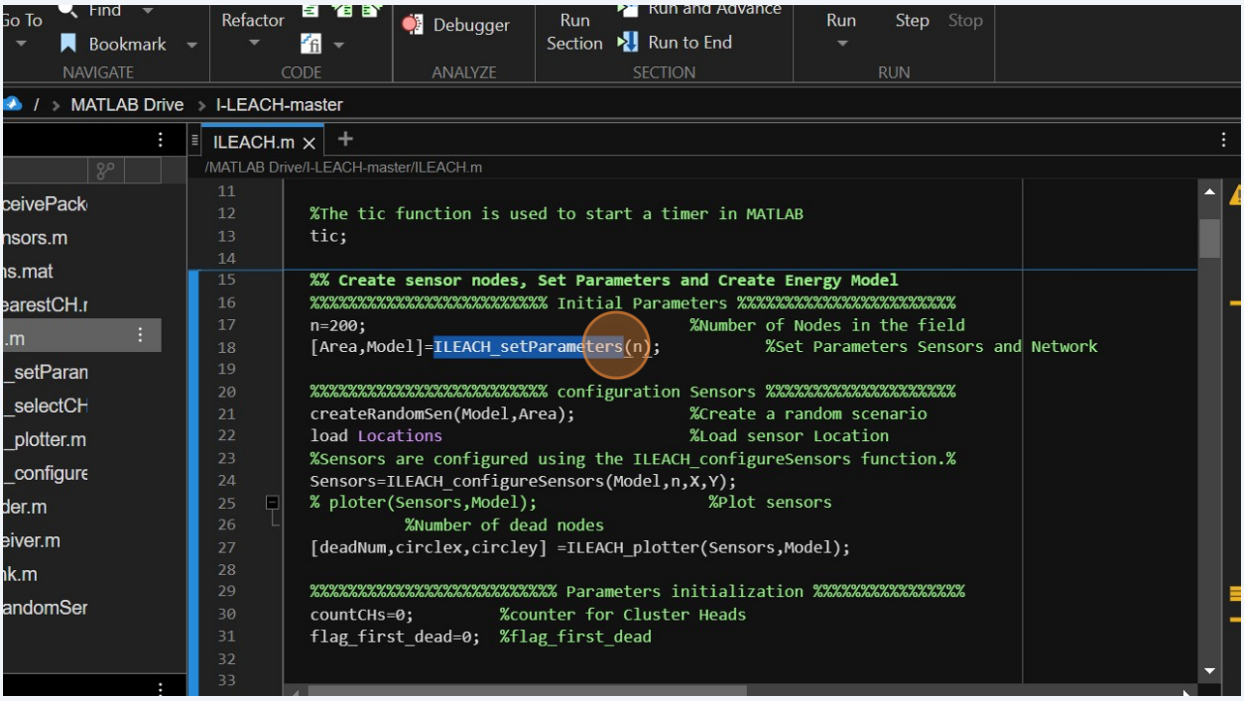
7 Here's a breakdown of the code:

1. n=200;: This line sets the number of nodes (n) in the field to 200.

2. [Area,Model]=ILEACH_setParameters(n);: This line calls the ILEACH_setParameters function with the argument n to set up initial parameters for the LEACH (Low-Energy Adaptive Clustering Hierarchy) protocol simulation. The function returns two outputs, Area and Model.

• Area is a structure containing the field dimensions (x and y).

• Model is a structure containing various parameters and settings for the simulation, including sink position, energy model parameters, maximum number of rounds, packet sizes, radio range, and I-LEACH specific parameters like the number of radio regions (numRx) and region width (dr).



8 Now navigate to ILEACH_setParemeters.m and Lets see what is the process of setting paremeters here:

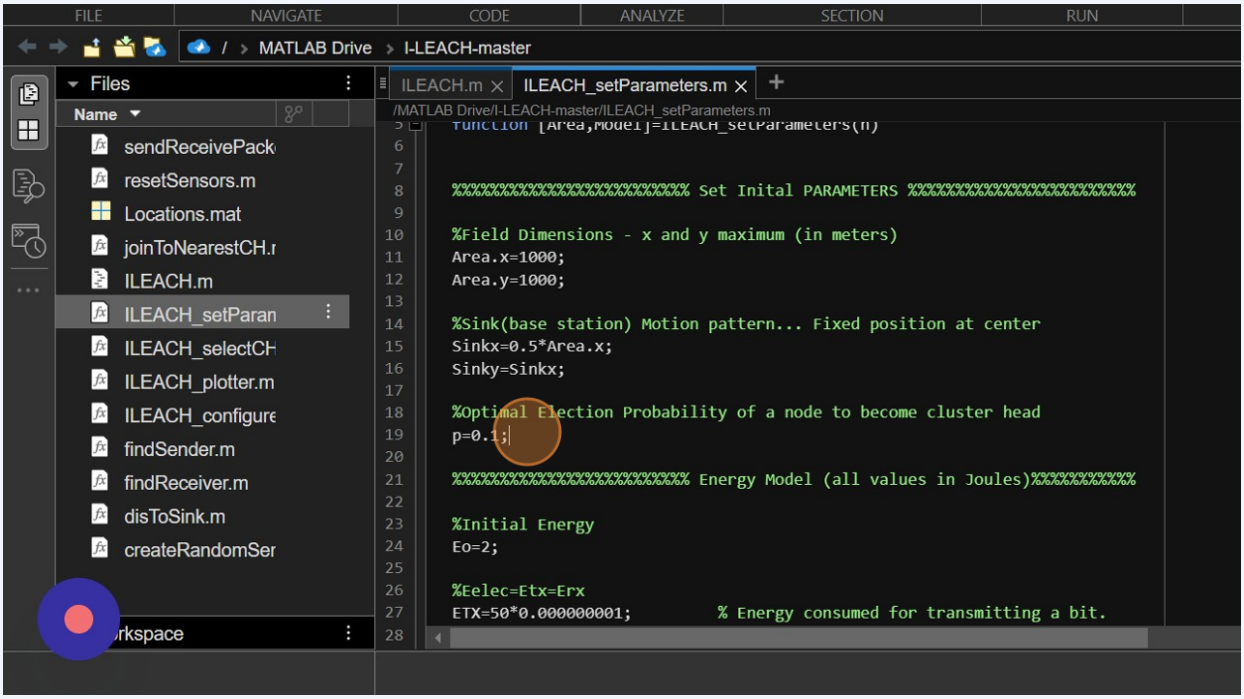
This part of the code is defining the initial parameters for the LEACH (Low-Energy Adaptive Clustering Hierarchy) simulation. Let's break down the key parameters:

1. Area.x=1000; and Area.y=1000;: These lines set the field dimensions. The field is a square with a maximum size of 1000x1000 meters.

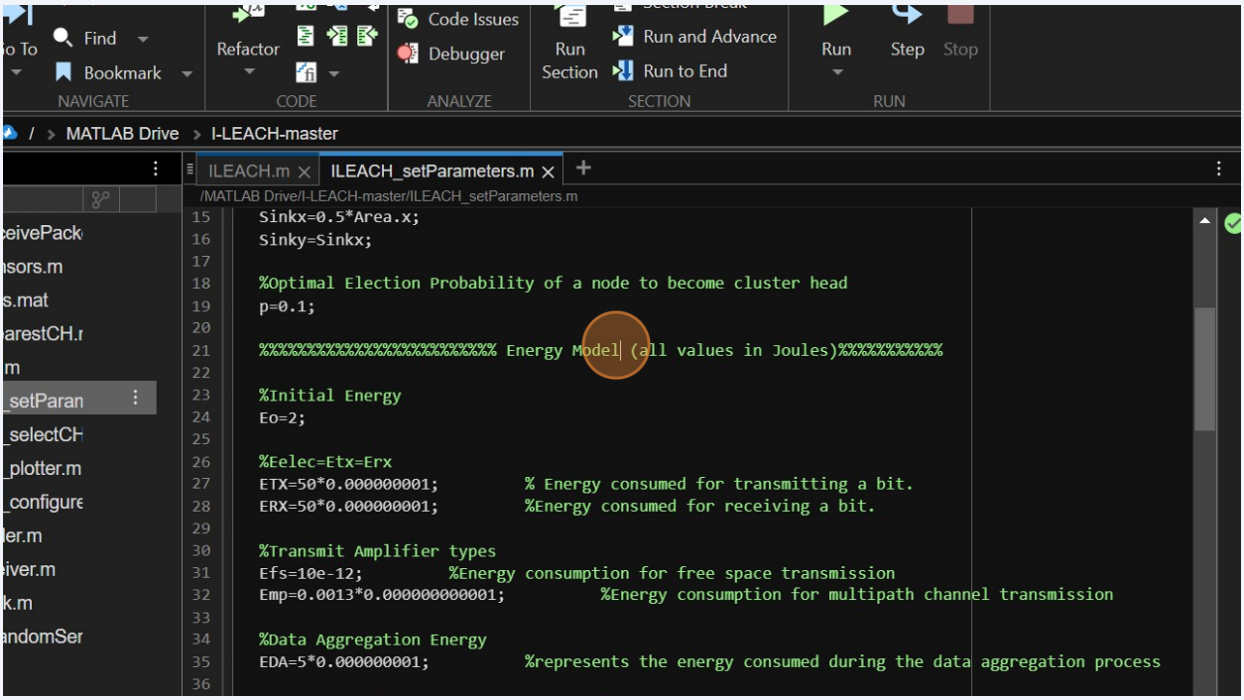
2. Sinkx=0.5*Area.x; and Sinky=Sinkx;: These lines set the fixed position of the sink (base station) at the center of the field. The sink's coordinates are determined as half of the maximum field dimensions.

3. p=0.1;: This line sets the optimal election probability of a node to become a cluster head. In LEACH, nodes probabilistically become cluster heads in each round, and p represents the probability.

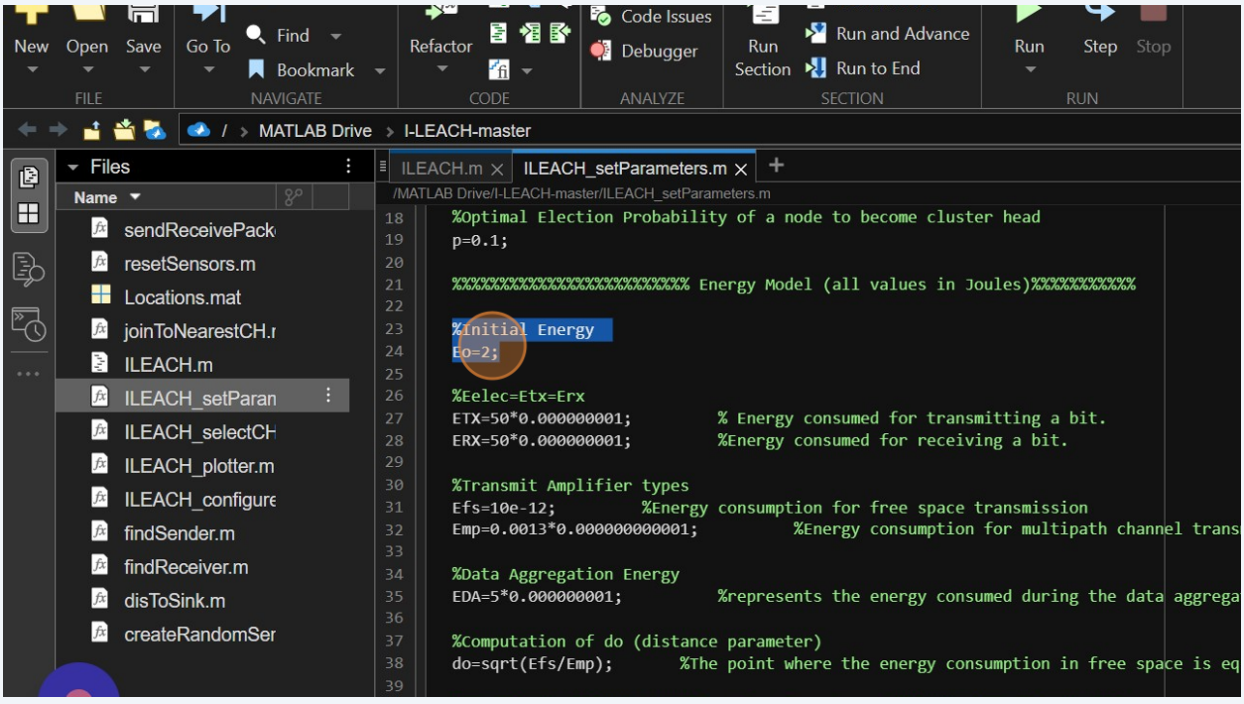
These parameters are crucial for the initial setup of the simulation and define the physical characteristics of the sensor field, the sink's location, and the probability for cluster head election



9 Now Lets Discuss the Energy Model:
This part of the code defines the energy model parameters for the sensor nodes in the LEACH simulation. Let's go through each parameter:



10 1. Eo=2;: This line sets the initial energy level of the sensor nodes to 2 Joules.



11

1. ETX=50*0.000000001; and ERX=50*0.000000001;: These lines set the energy consumed for transmitting (ETX) and receiving (ERX) a single bit, respectively. Both values are set to 50 nanojoules/bit.

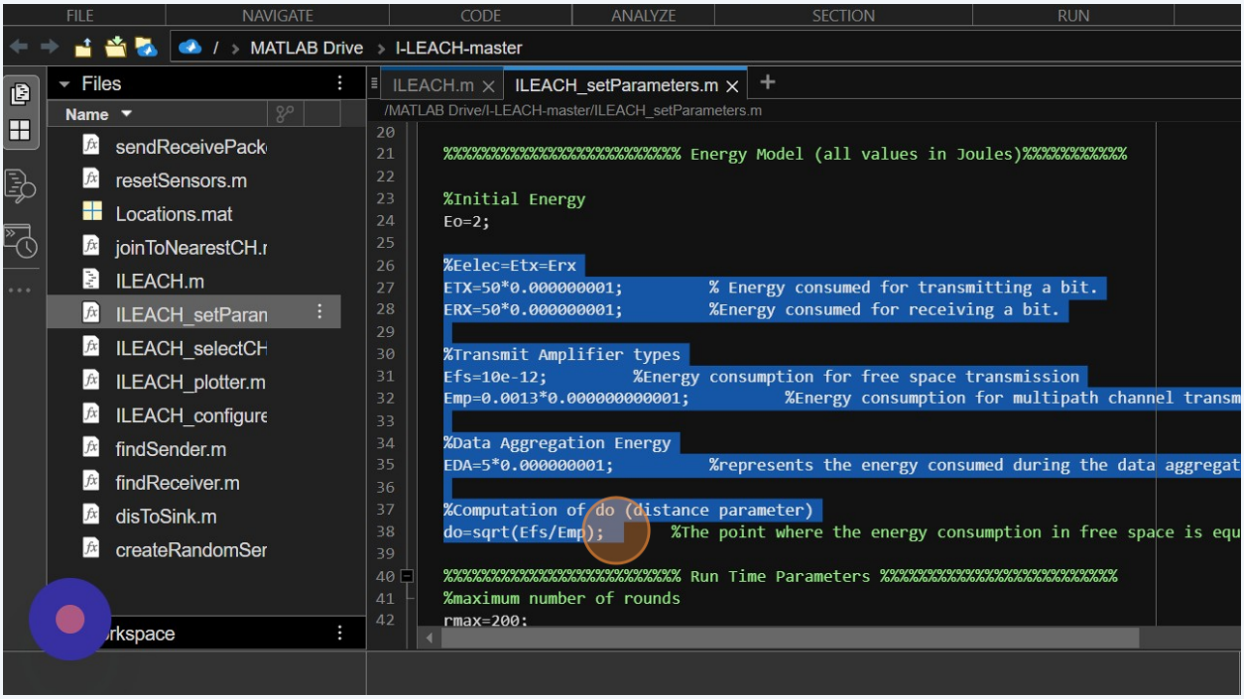
2. Efs=10e-12;: This line sets the energy consumption for free space transmission. It represents the energy consumed per bit when transmitting over a free space channel. The value is set to **1×10−121×10−12** Joules/bit.

3. Emp=0.0013*0.000000000001;: This line sets the energy consumption for multipath channel transmission. It represents the energy consumed per bit when transmitting over a multipath channel. The value is calculated based on a specific formula and set to **1.3×10−151.3×10−15** Joules/bit.

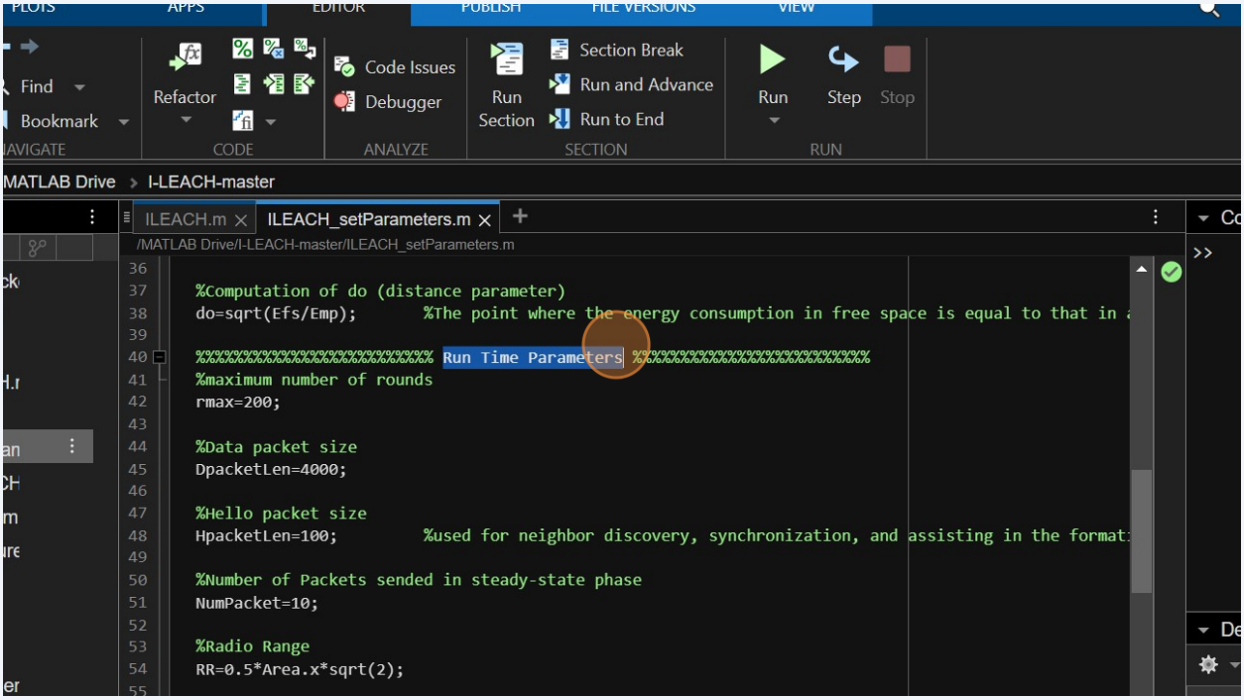
4. EDA=5*0.000000001;: This line sets the energy consumed during the data aggregation process. The value is set to **5×10−95×10−9** Joules.

5. do=sqrt(Efs/Emp);: This line computes the distance parameter do, which represents the distance at which the energy consumption in free space is equal to that in a multipath channel. It is calculated based on the defined values of Efs and Emp.

These parameters collectively define the energy consumption characteristics of the sensor nodes during communication and data aggregation in the simulation.



12 Now here are the Run Time Parameters lets discuss about these



13 Let's go through each parameter:

1. rmax=200;: This line sets the maximum number of rounds for the simulation to 200.

2. DpacketLen=4000;: This line sets the data packet size to 4000 bits.

3. HpacketLen=100;: This line sets the hello packet size to 100 bits. Hello packets are used for neighbor discovery, synchronization, and assisting in the formation and maintenance of clusters.

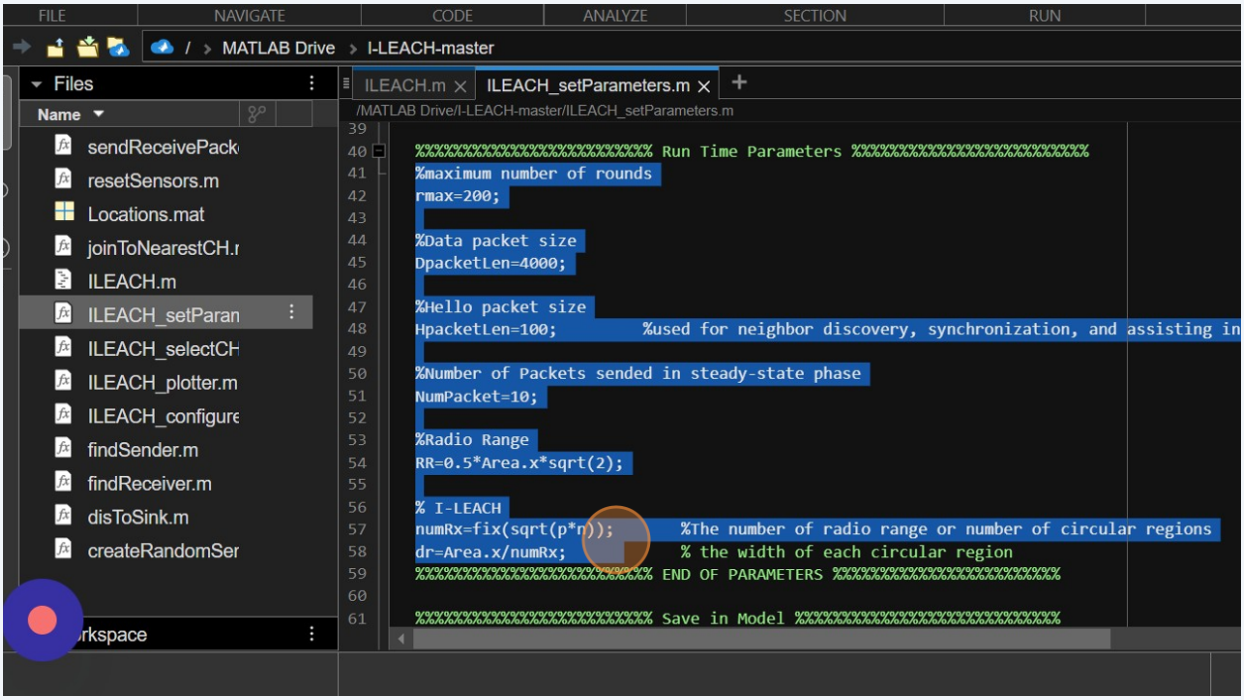
4. NumPacket=10;: This line sets the number of packets sent in the steady-state phase to 10.

5. RR=0.5*Area.x*sqrt(2);: This line calculates the radio range (RR) based on the field dimensions (Area.x and Area.y). The radio range is set to half of the diagonal of the field.

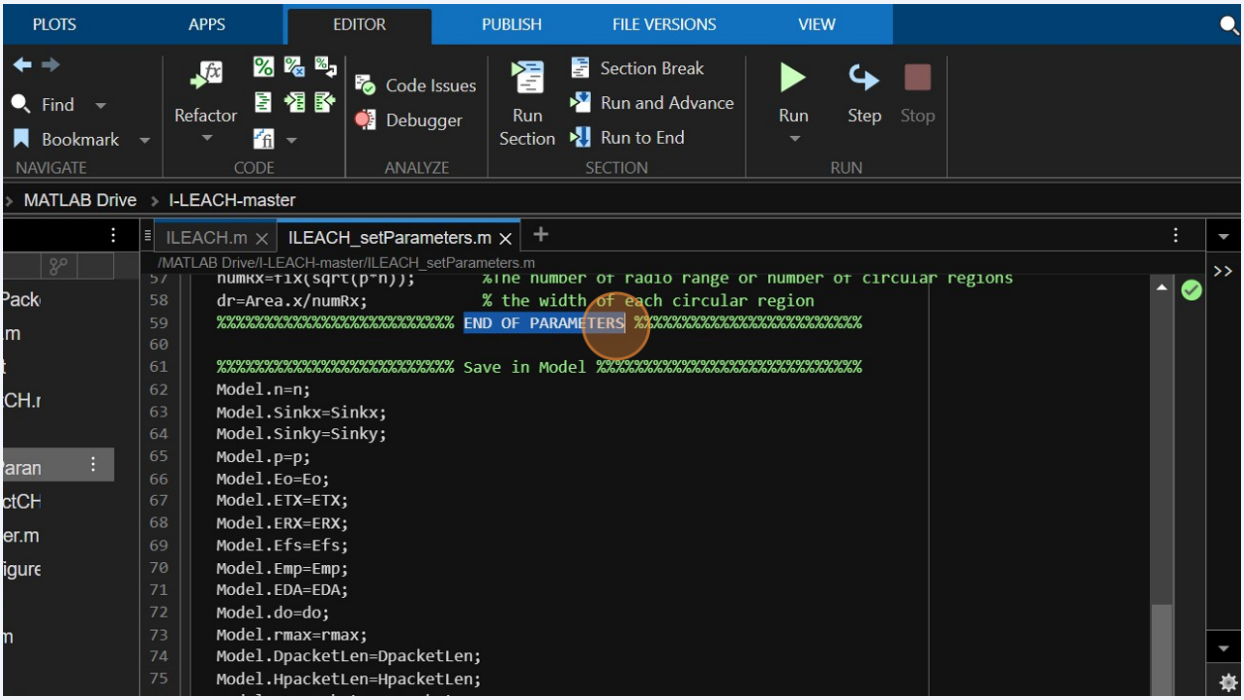
6. numRx=fix(sqrt(p*n));: This line calculates the number of radio ranges or the number of circular regions (numRx) based on the optimal election probability (p) and the number of nodes (n). The fix function rounds down the result to the nearest integer.

7. dr=Area.x/numRx;: This line calculates the width of each circular region (dr) based on the field dimensions (Area.x) and the number of circular regions (numRx).

These parameters control aspects such as the duration of the simulation, packet sizes, radio range, and the configuration of circular regions in the I-LEACH protocol.



14 Now Parameters in this function are ended



15 In this section, the values of the parameters and calculated variables are assigned to the Model structure. Here's a breakdown of what each line does:

1. Model.n=n;: Assigns the number of nodes (n) to the n field in the Model structure.

2. Model.Sinkx=Sinkx;: Assigns the x-coordinate of the sink (Sinkx) to the Sinkx field in the Model structure.

3. Model.Sinky=Sinky;: Assigns the y-coordinate of the sink (Sinky) to the Sinky field in the Model structure.

4. Model.p=p;: Assigns the optimal election probability (p) to the p field in the Model structure.

5. Model.Eo=Eo;: Assigns the initial energy (Eo) to the Eo field in the Model structure.

6. Model.ETX=ETX;: Assigns the energy consumed for transmitting a bit (ETX) to the ETX field in the Model structure.

7. Model.ERX=ERX;: Assigns the energy consumed for receiving a bit (ERX) to the ERX field in the Model structure.

8. Model.Efs=Efs;: Assigns the energy consumption for free space transmission (Efs) to the Efs field in the Model structure.

9. Model.Emp=Emp;: Assigns the energy consumption for multipath channel transmission (Emp) to the Emp field in the Model structure.

10. Model.EDA=EDA;: Assigns the energy consumed during the data aggregation process (EDA) to the EDA field in the Model structure.

11. Model.do=do;: Assigns the distance parameter (do) to the do field in the Model structure.

12. Model.rmax=rmax;: Assigns the maximum number of rounds (rmax) to the rmax field in the Model structure.

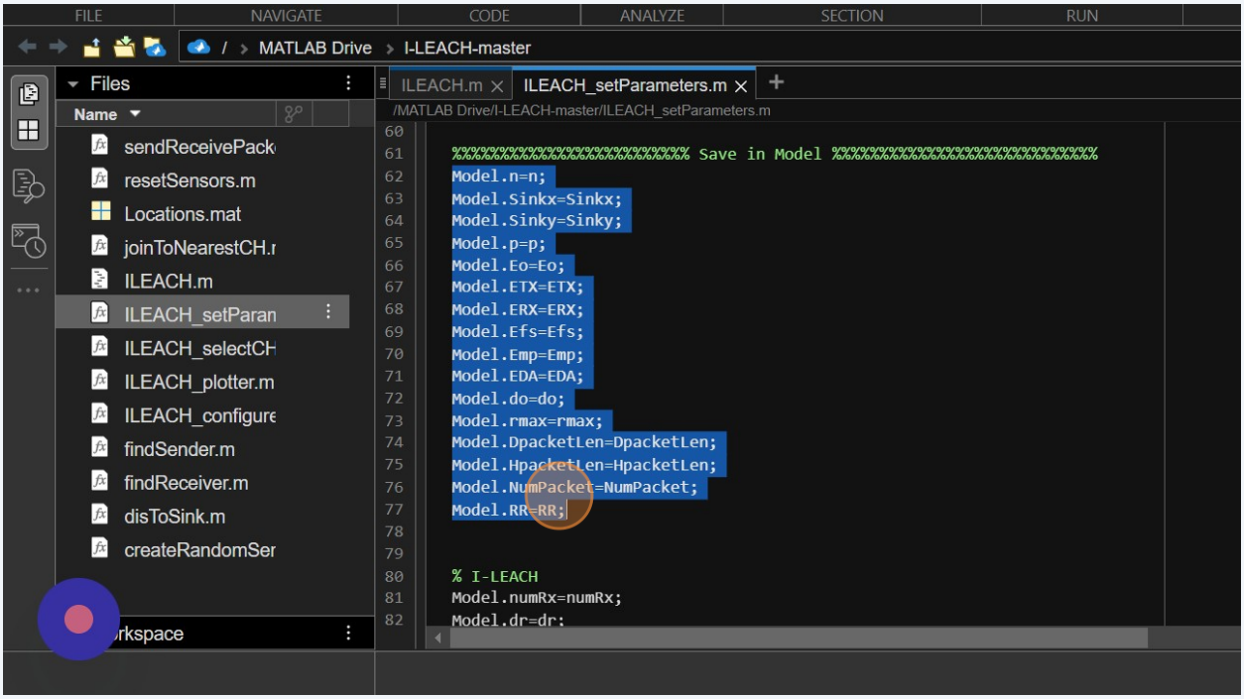
13. Model.DpacketLen=DpacketLen;: Assigns the data packet size (DpacketLen) to the DpacketLen field in the Model structure.

14. Model.HpacketLen=HpacketLen;: Assigns the hello packet size (HpacketLen) to the HpacketLen field in the Model structure.

15. Model.NumPacket=NumPacket;: Assigns the number of packets sent in the steady-state phase (NumPacket) to the NumPacket field in the Model structure.

16. Model.RR=RR;: Assigns the radio range (RR) to the RR field in the Model structure.

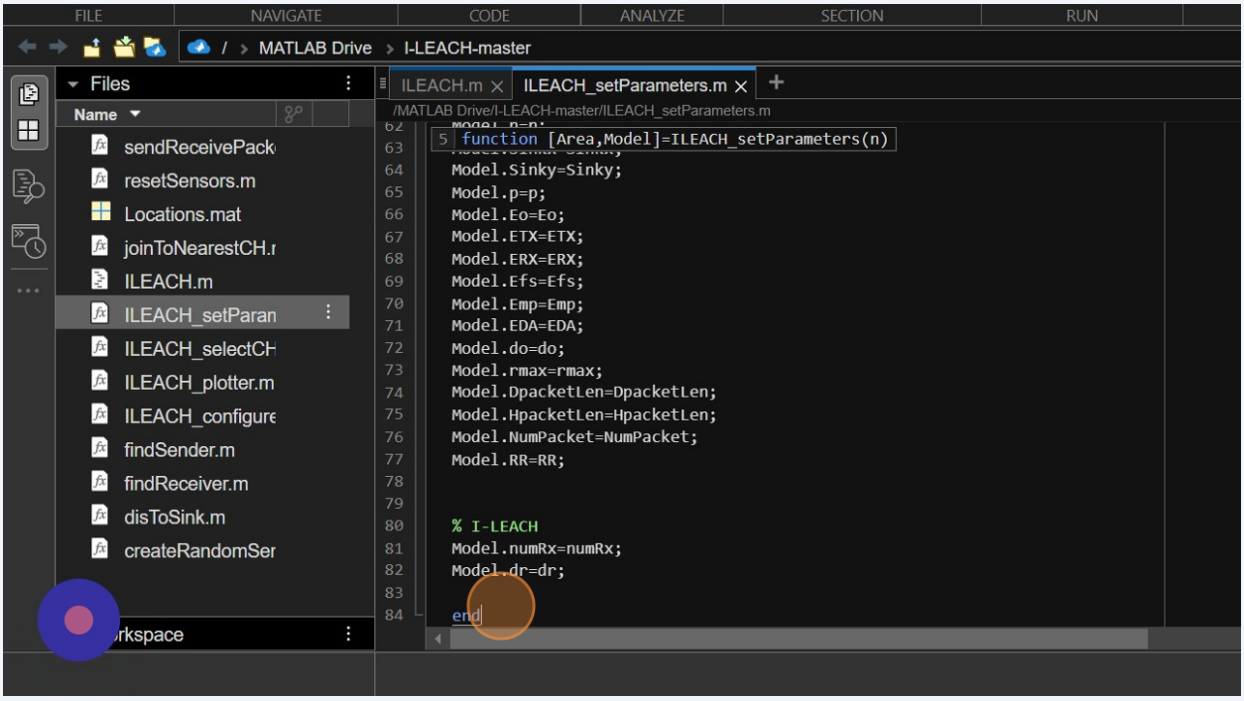
All these values are crucial for defining the simulation parameters and the I-LEACH protocol's behavior.



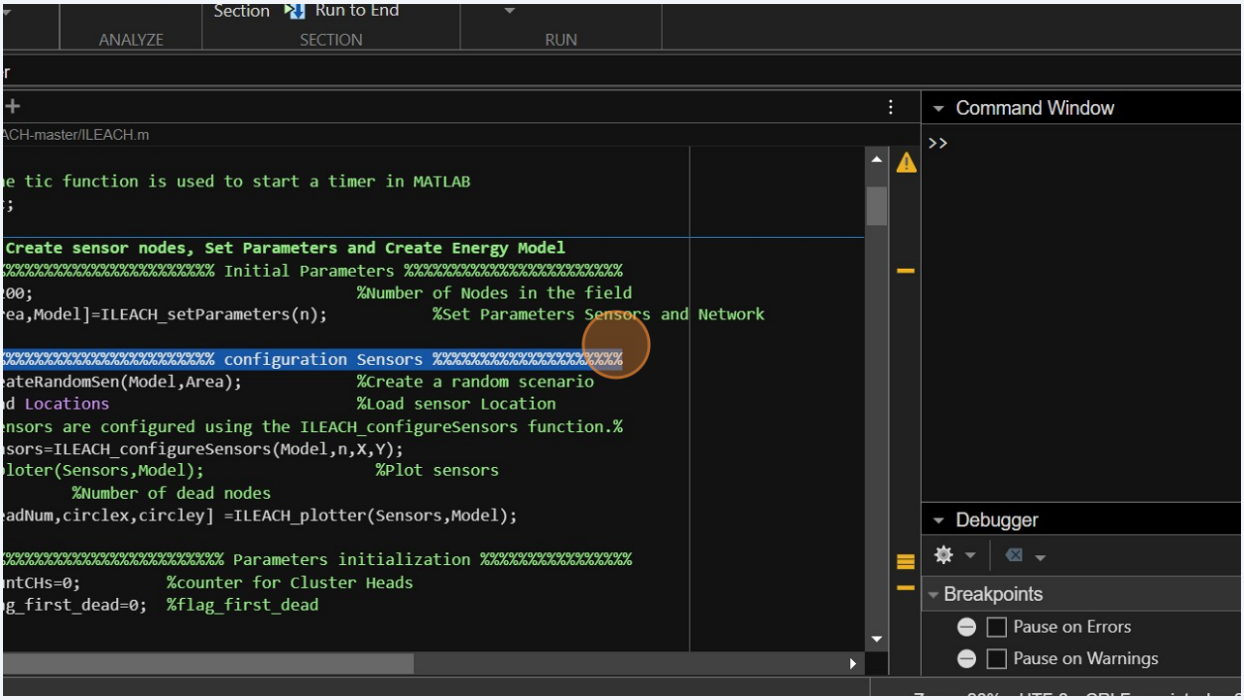
16 In this last part of this function,
the last two lines are additional assignments for the I-LEACH extension:

1. Model.numRx=numRx;: Assigns the number of radio range or the number of circular regions (numRx) to the numRx field in the Model structure.

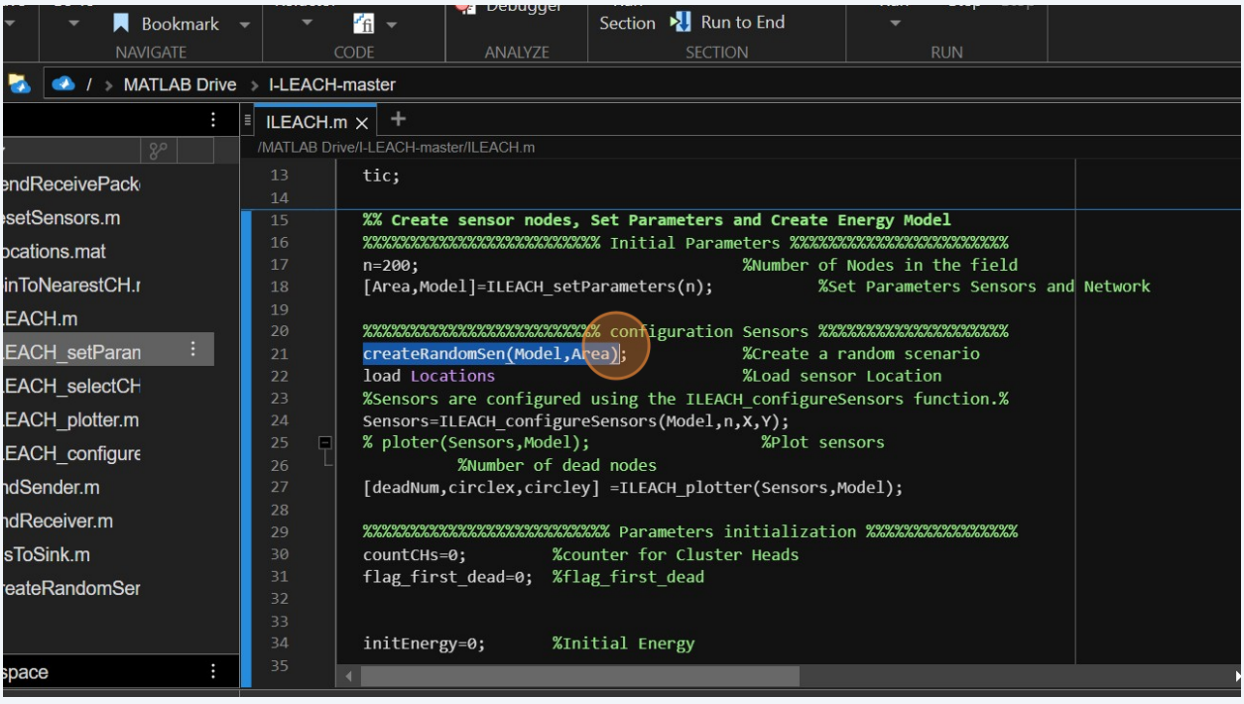
2. Model.dr=dr;: Assigns the width of each circular region (dr) to the dr field in the Model structure.



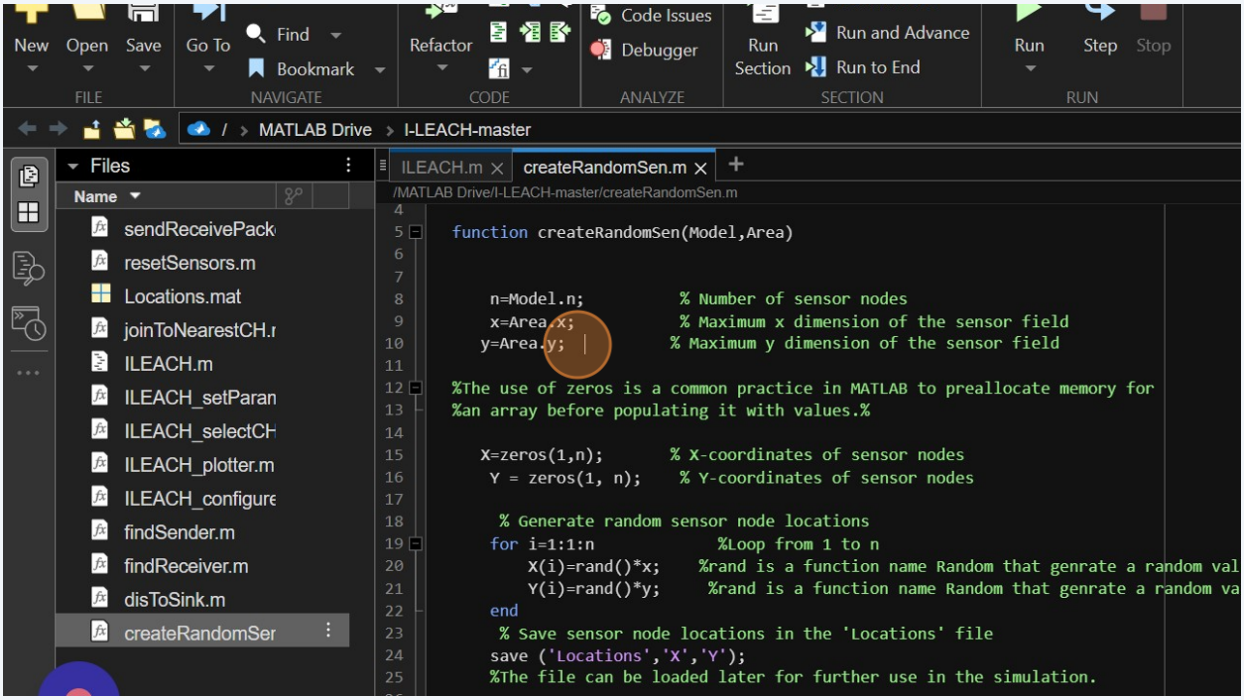
17 After setting perimeters Sensors are configured



18 In first there is a function createRandomSen that takes two paremeters and now lets observe the functionality of this funxtion from the file of the function name createRandomSen



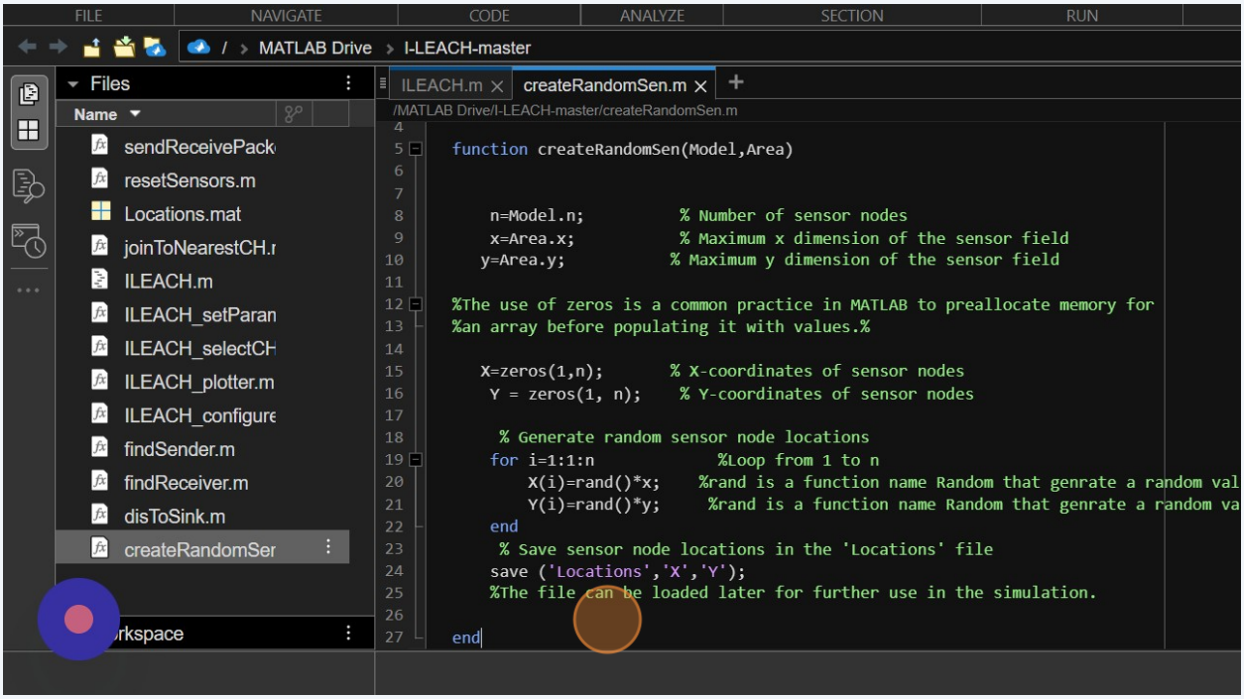
19 Click "y=Area.y;"



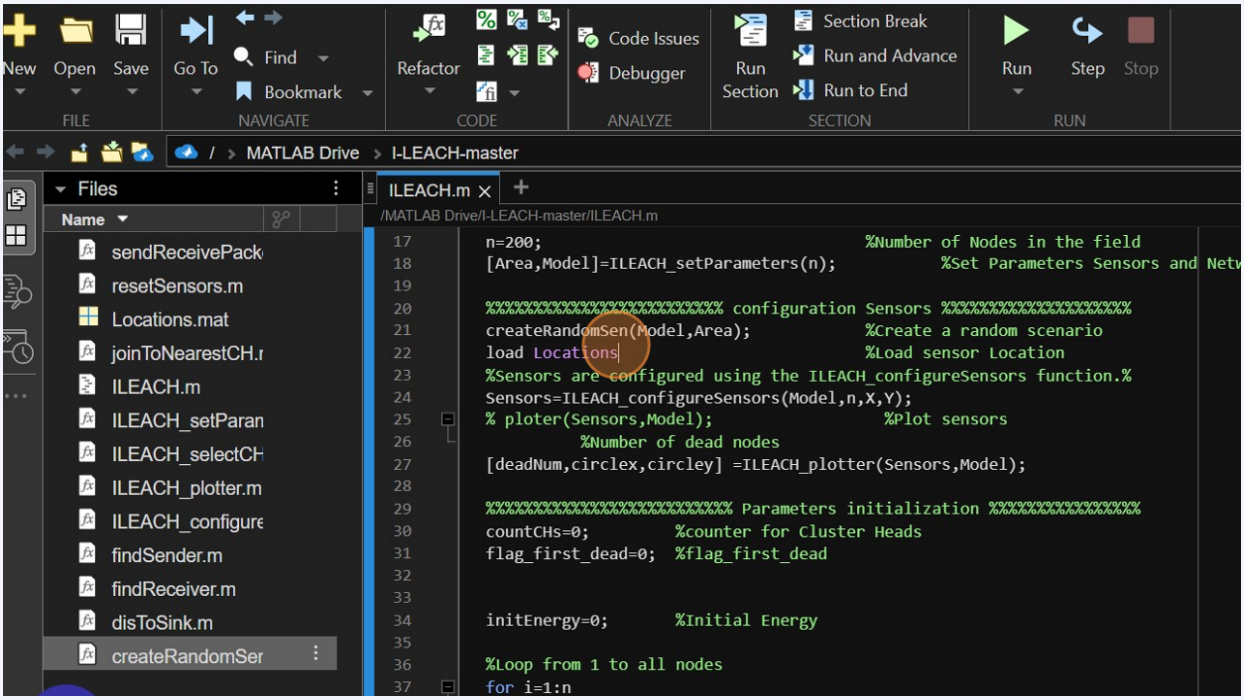
20 The remaining part of the this function code generates random sensor node locations within the specified field dimensions and saves them in a file named 'Locations'. Here's a breakdown of the code:

- X=zeros(1,n);: Preallocates an array X of size 1 x n to store x-coordinates of sensor nodes.
- Y=zeros(1,n);: Preallocates an array Y of size 1 x n to store y-coordinates of sensor nodes.
- for i=1:1:n: Iterates from 1 to n.
- X(i)=rand()*x;: Generates a random x-coordinate for the i-th sensor node and assigns it to X(i).
- Y(i)=rand()*y;: Generates a random y-coordinate for the i-th sensor node and assigns it to Y(i).
- save ('Locations','X','Y');: Saves the generated sensor node locations in the 'Locations' file. The file will contain the X and Y coordinates of each sensor node.

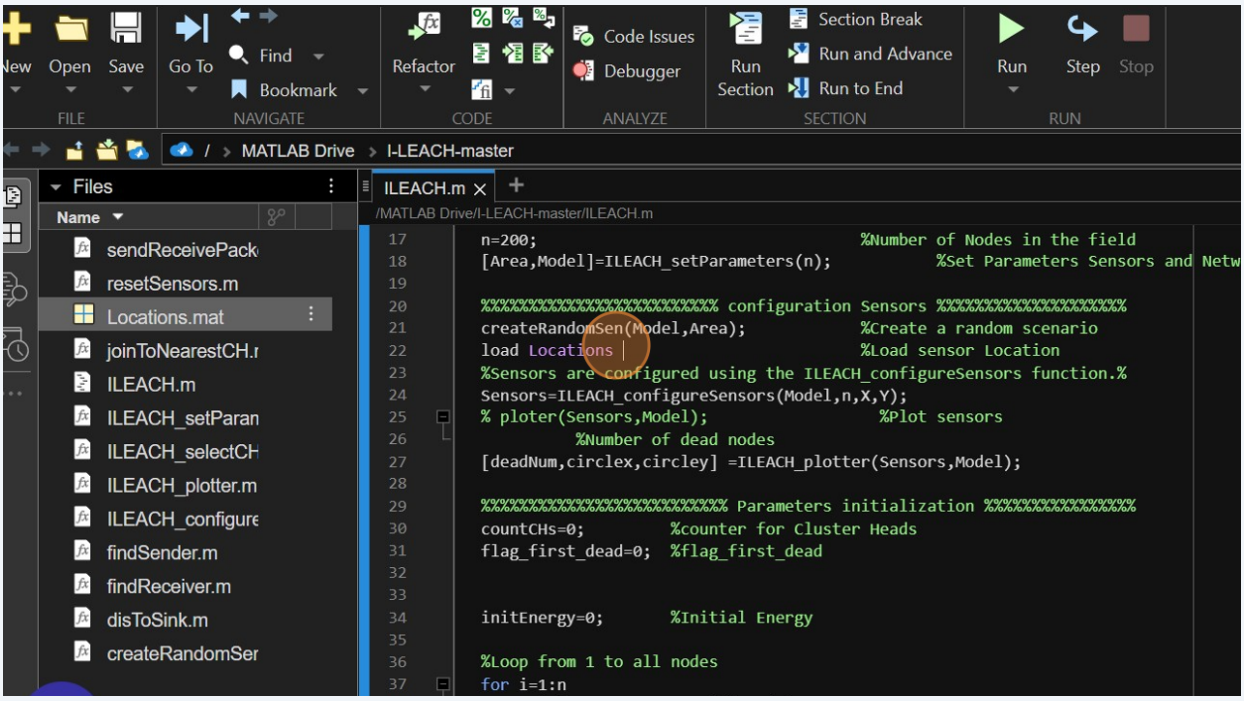
This file can be used later in the simulation for various purposes, such as initializing the sensor nodes with specific locations.



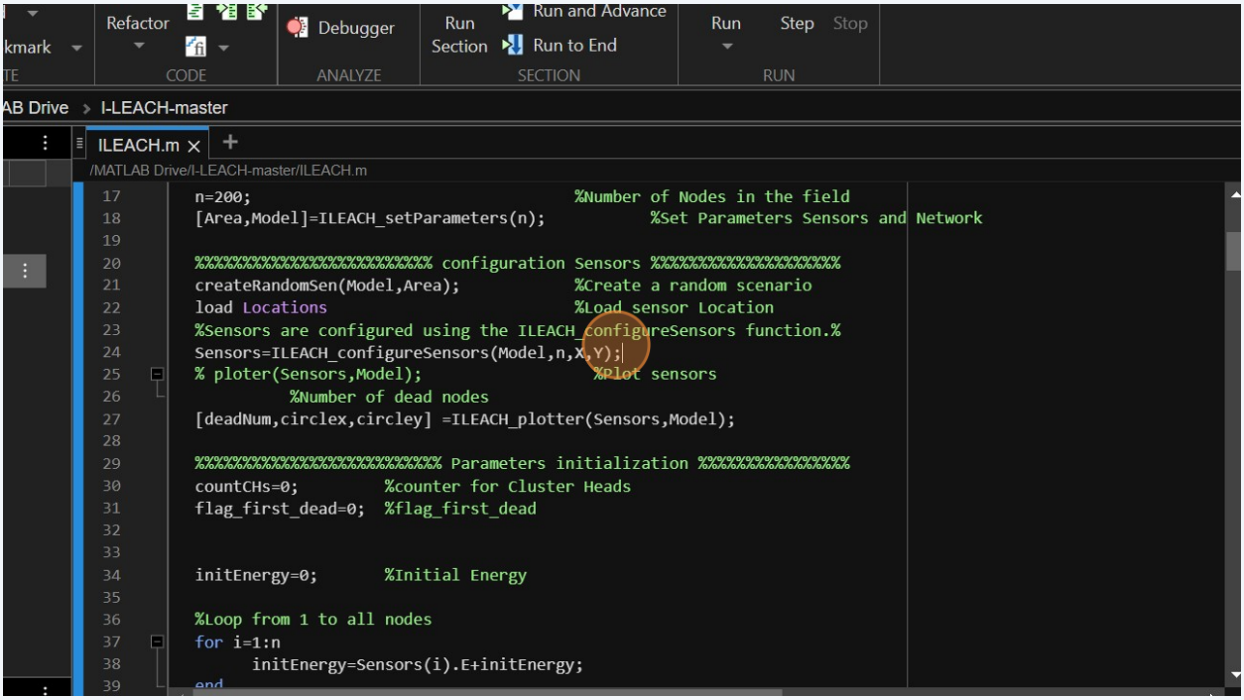
21 Now moving back to Main ILEACH file Here load Locations command is to load the Location of sensor



22 This command also has a file Locations.mat as you can see in Files , when it run it just loads the locations



23 Now in the next step ILEACH_configureSensors function is used that is also in another file with the same name as function



24 Now lets observe this function file code
This function, ILEACH_configureSensors, configures the sensor nodes in the LEACH protocol based on the specified model parameters. Here's a breakdown of the code:

• EmptySensor: Defines a structure representing the configuration of an empty sensor node with default values for various parameters.

• EmptySensor.xd=0;: Initializes the x-dimension of the sensor node to 0.

• EmptySensor.yd=0;: Initializes the y-dimension of the sensor node to 0.

• EmptySensor.G=0;: Initializes the 'G' attribute, representing whether the sensor node has been a cluster head in previous periods, to 0.

• EmptySensor.df=0;: Initializes the 'df' attribute, representing the dead flag (0 for alive, 1 for dead), to 0.

• EmptySensor.type='N';: Initializes the type of the sensor node to 'N', indicating it's a normal node.

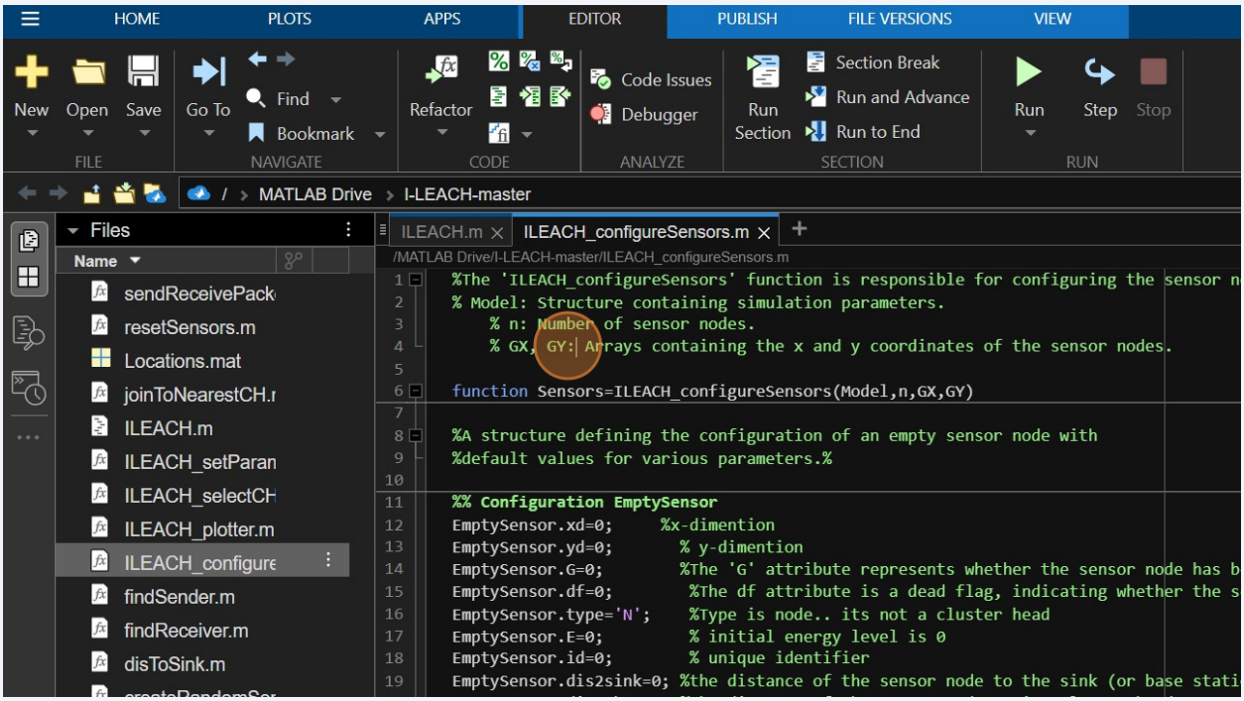
• EmptySensor.E=0;: Initializes the initial energy level of the sensor node to 0.

• EmptySensor.id=0;: Initializes the unique identifier of the sensor node to 0.

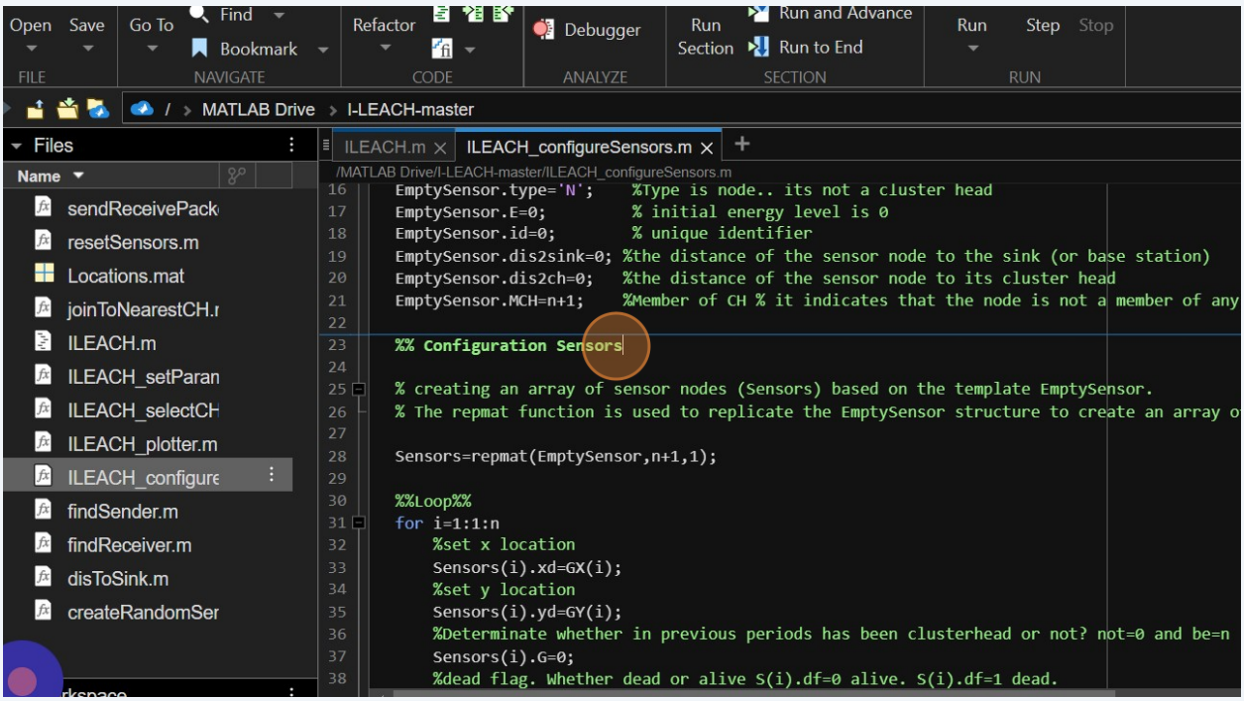
• EmptySensor.dis2sink=0;: Initializes the distance of the sensor node to the sink (or base station) to 0.

• EmptySensor.dis2ch=0;: Initializes the distance of the sensor node to its cluster head to 0.

• EmptySensor.MCH=n+1;: Initializes the member of CH (Cluster Head) to n+1, indicating that the node is not a member of any cluster initially.



25 • Sensors=repmat(EmptySensor,n+1,1);: Creates an array of sensor nodes (Sensors) based on the template EmptySensor. The repmat function replicates the EmptySensor structure to create an array of size (n+1) x 1.



26 • for i=1:1:n: Iterates over all nodes.

• Sensors(i).xd=GX(i);: Sets the x-location of the i-th sensor node.

• Sensors(i).yd=GY(i);: Sets the y-location of the i-th sensor node.

• Sensors(i).G=0;: Initializes whether the sensor node has been a cluster head in previous periods to 0.

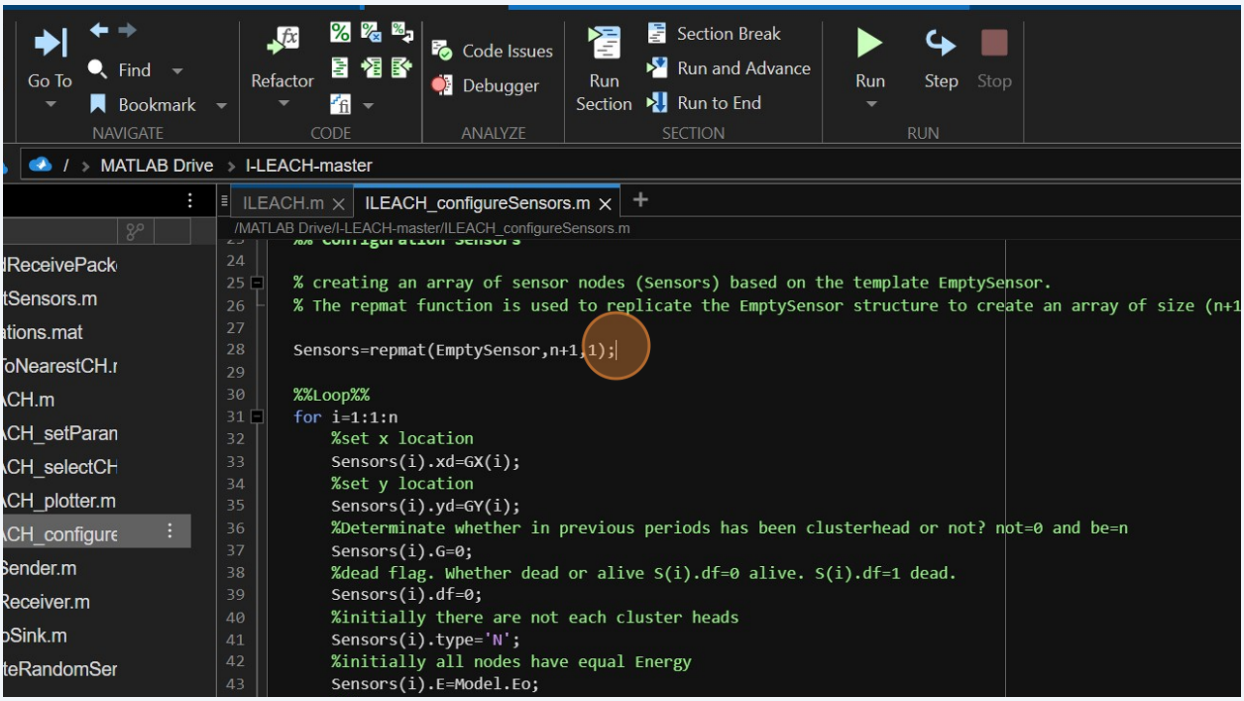
• Sensors(i).df=0;: Initializes the dead flag to 0, indicating the node is alive.

• Sensors(i).type='N';: Initializes the type of the sensor node to 'N', indicating it's a normal node.

• Sensors(i).E=Model.Eo;: Initializes the energy of the sensor node with the initial energy specified in the model parameters.

• Sensors(i).id=i;: Sets the unique identifier of the sensor node.

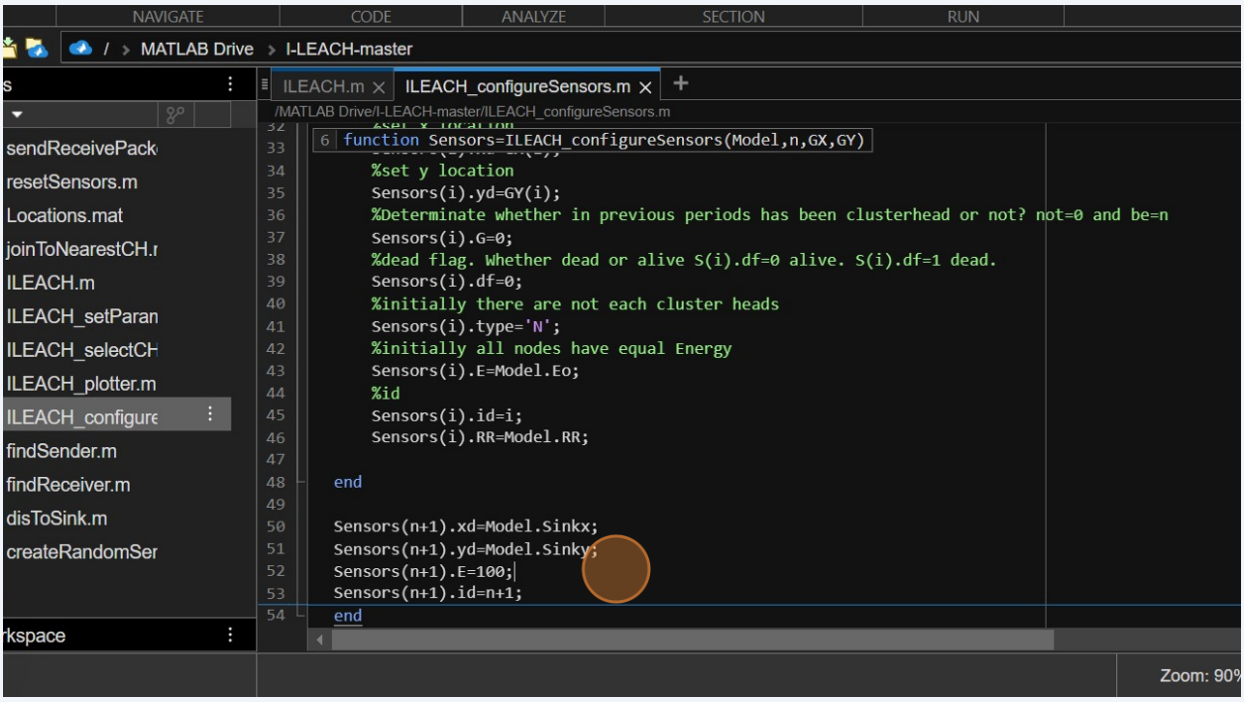
• Sensors(i).RR=Model.RR;: Sets the radio range of the sensor node based on the model parameters.



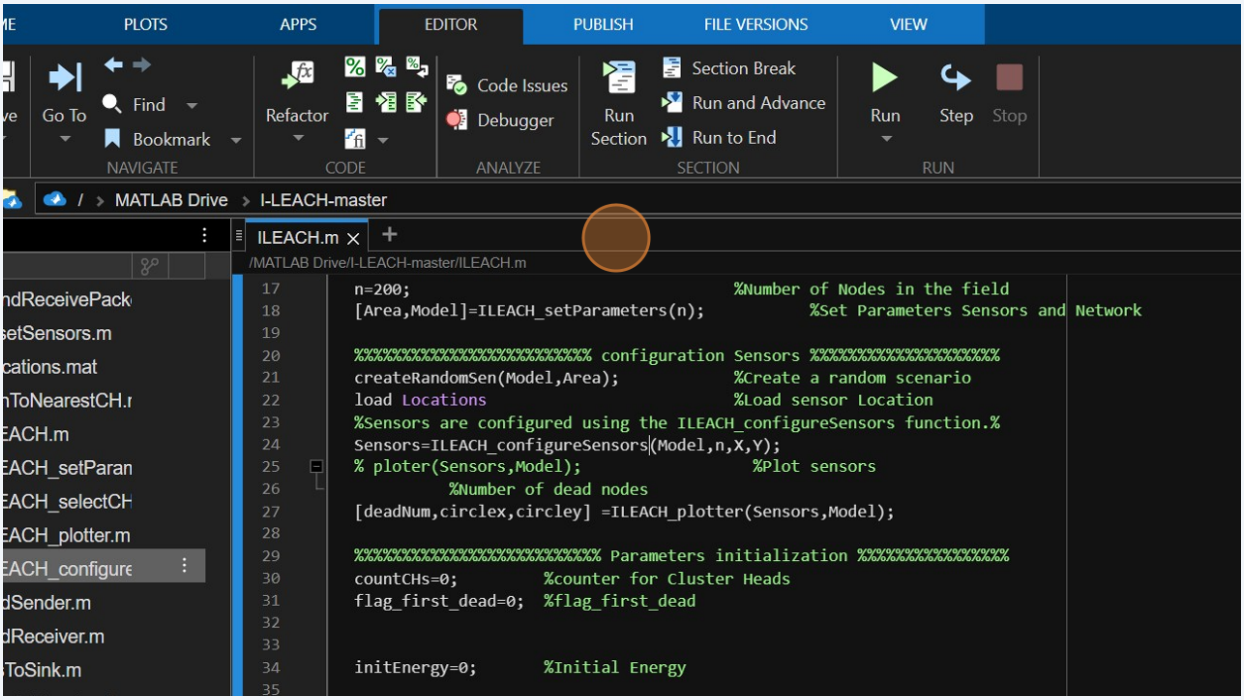
27

- Sensors(n+1).xd=Model.Sinkx;: Sets the x-location of the sink (base station).
- Sensors(n+1).yd=Model.Sinky;: Sets the y-location of the sink (base station).
- Sensors(n+1).E=100;: Initializes the energy of the sink (base station) to 100.
- Sensors(n+1).id=n+1;: Sets the unique identifier of the sink (base station).

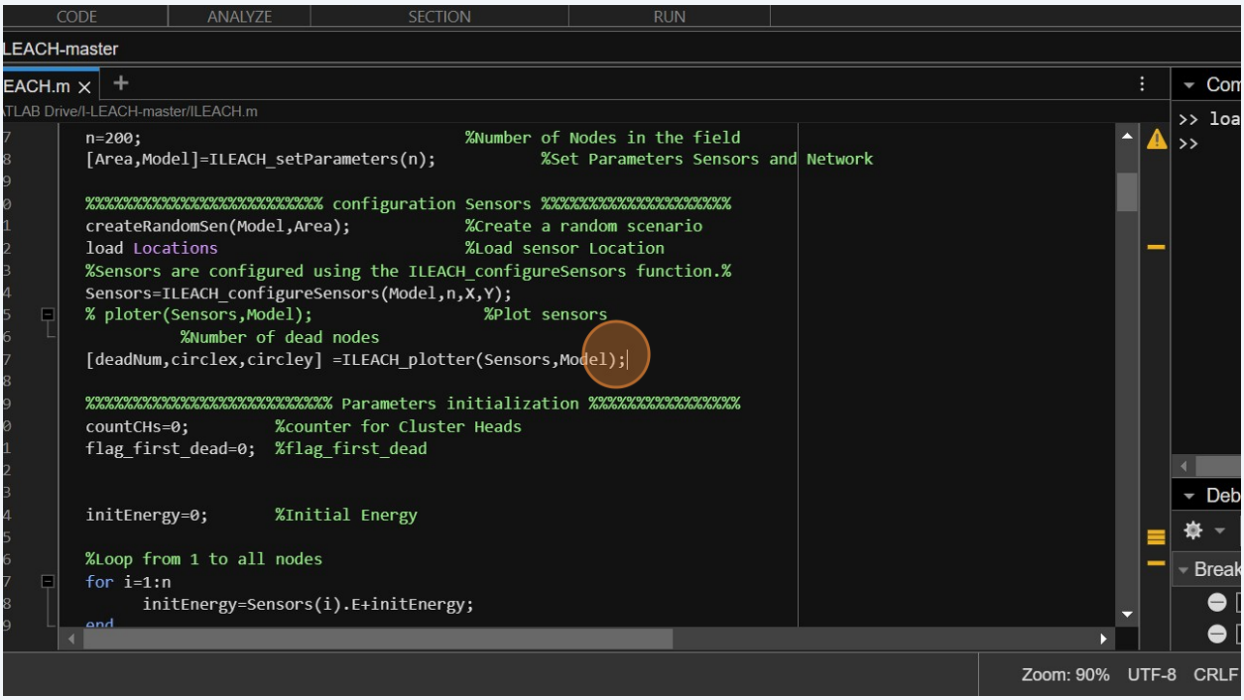
The function returns the configured array of sensor nodes (Sensors).



28 Now lets move back to Main file to observe next step



29 After the configureSensors plotter function is used that takes two parameters



30 Now lets observe this function code and working

This function, ILEACH_plotter, is responsible for visualizing the state of the sensor network in the LEACH simulation. Let's break down the code:

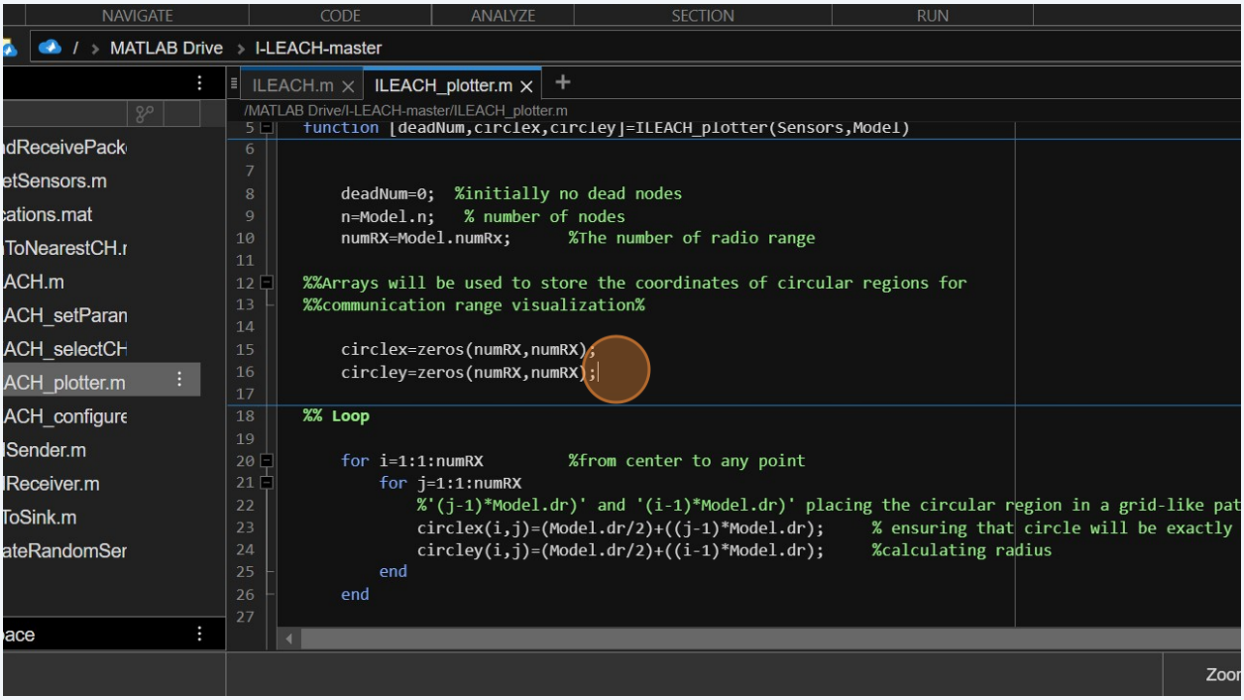
• deadNum=0;: Initializes a variable deadNum to 0, which will be used to count the number of dead nodes.

• n=Model.n;: Retrieves the total number of nodes from the model parameters.

• numRX=Model.numRx;: Retrieves the number of circular regions (radio range) from the model parameters.

• circlex=zeros(numRX,numRX);: Initializes a 2D array circlex with zeros to store the x-coordinates of circular regions for communication range visualization.

• circley=zeros(numRX,numRX);: Initializes a 2D array circley with zeros to store the y-coordinates of circular regions for communication range visualization



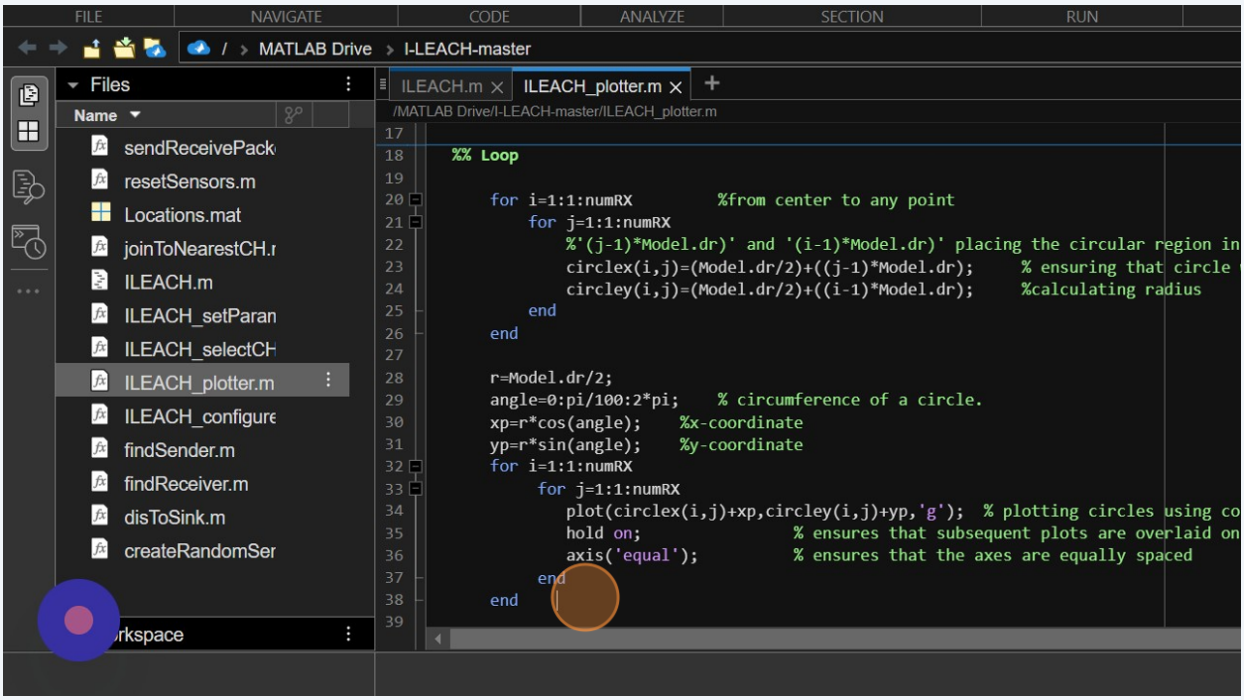
31

Inside the %Loop

This section of code is responsible for generating and plotting circular regions (communication ranges) in the sensor network simulation. Let's break it down:

- for i=1:1:numRX: This loop iterates through the rows of the circular regions.
- for j=1:1:numRX: This nested loop iterates through the columns of the circular regions.
- circlex(i,j)=(Model.dr/2)+((j-1)*Model.dr);: Calculates the x-coordinate of the center of each circular region based on the grid-like pattern.
- circley(i,j)=(Model.dr/2)+((i-1)*Model.dr);: Calculates the y-coordinate of the center of each circular region based on the grid-like pattern.

After executing these loops, the code generates a grid of circular regions based on the specified communication range (Model.dr) and the number of circular regions (numRX).



32

• The loop iterates through all nodes (for i=1:n).

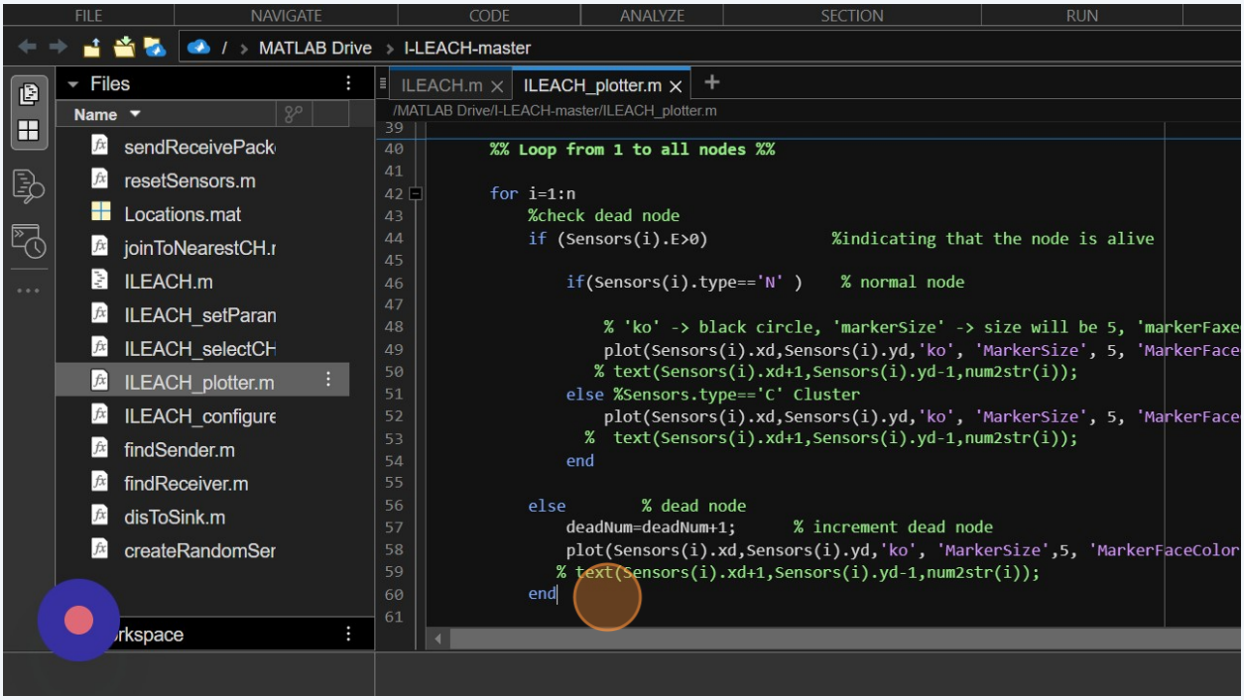
• It checks if the node is alive (if (Sensors(i).E > 0)). If yes, it further checks whether the node is a normal node or a cluster head.

• If it's a normal node (if (Sensors(i).type == 'N')), it plots a black circle ('ko') with a size of 5 and a black filled circle ('MarkerFaceColor', 'k').

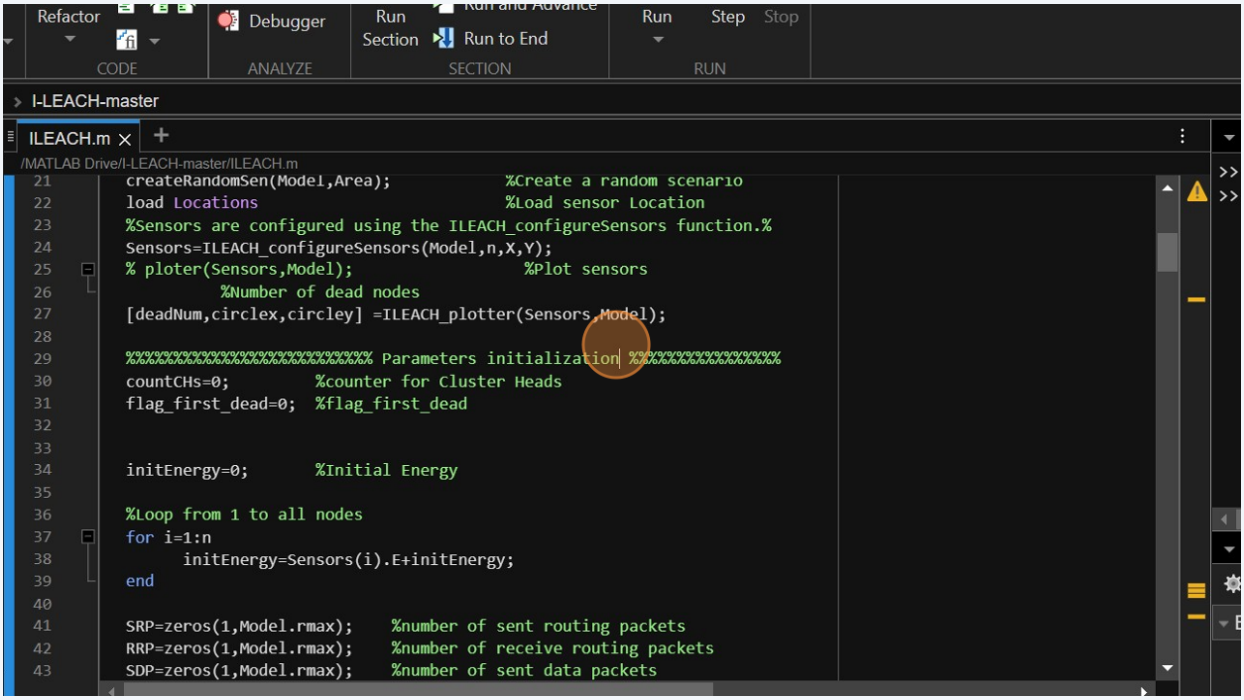
• If it's a cluster head (else), it plots a black circle ('ko') with a size of 5 and a red filled circle ('MarkerFaceColor', 'r').

• If the node is dead (else), it increments the deadNum counter and plots a black circle ('ko') with a size of 5 and a white filled circle ('MarkerFaceColor', 'w').

This section visually represents the state of each node in the simulation based on its status and type. The color and style of the circles indicate whether the node is alive, dead, a normal node, or a cluster head.



33 Now back to the Main ILEACH file code
We have a Parameters Initiaization part



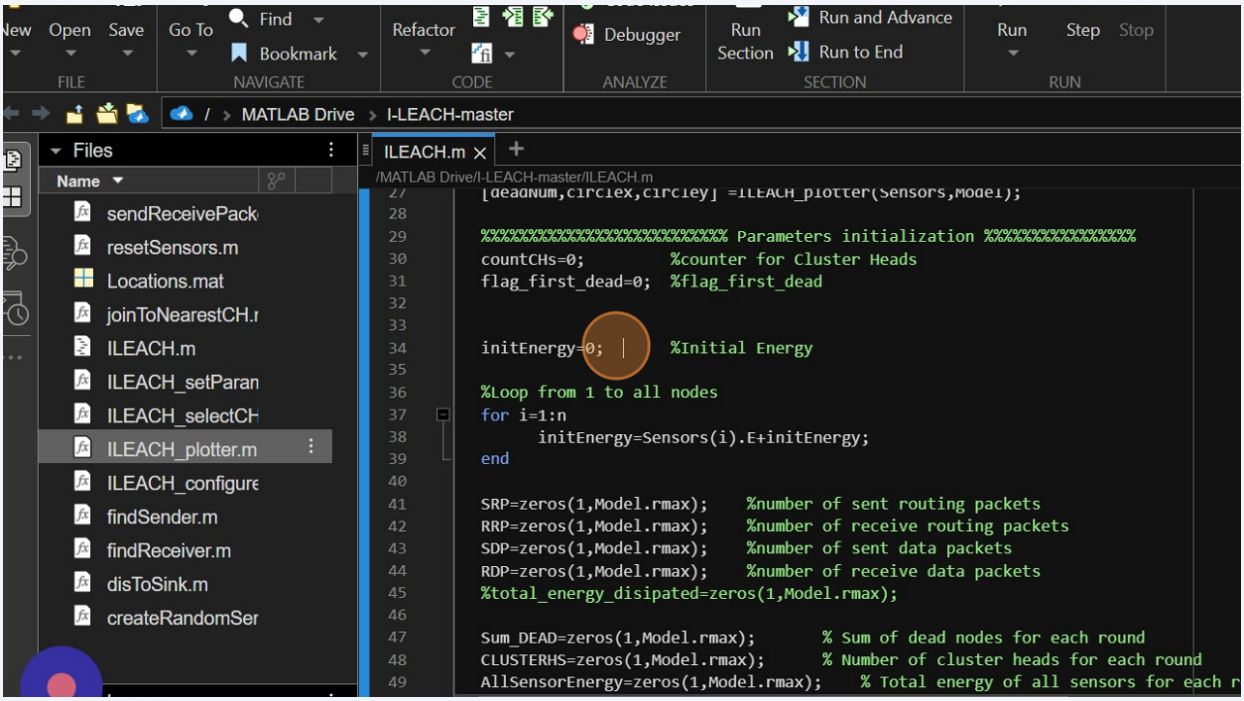
34

• countCHs: It is a counter that keeps track of the number of cluster heads in the network.

• flag_first_dead: This is a flag used to indicate whether the first node in the network has died. It's initialized to 0.

• initEnergy: This variable is set to 0 and is labeled as "Initial Energy." It might be used for some calculations or comparisons later in the code.

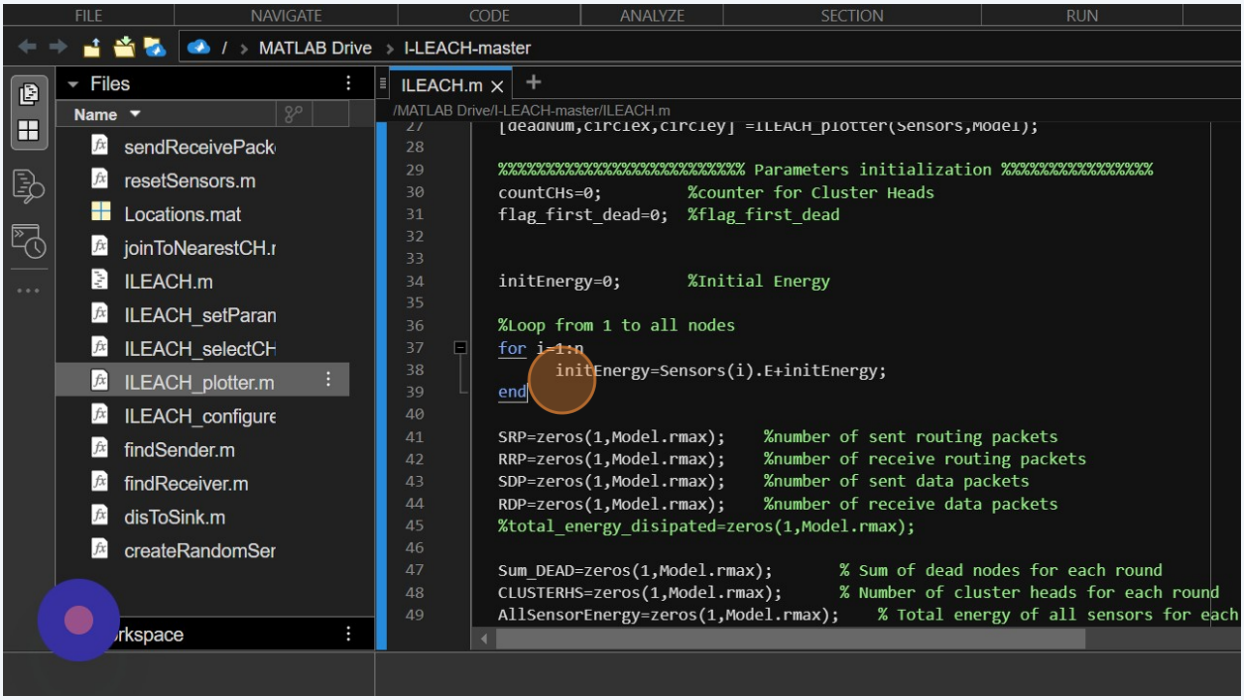
These variables are initialized at the beginning of the simulation and are likely to be updated and used throughout the execution of the code. The exact purpose of these variables will become clearer as we analyze more parts of the code.



35

- It uses a loop to iterate over all sensor nodes in the network (for i=1:n).
- For each node, it adds its initial energy level (Sensors(i).E) to the variable initEnergy.
- The result is the sum of the initial energy levels of all sensor nodes in the network.

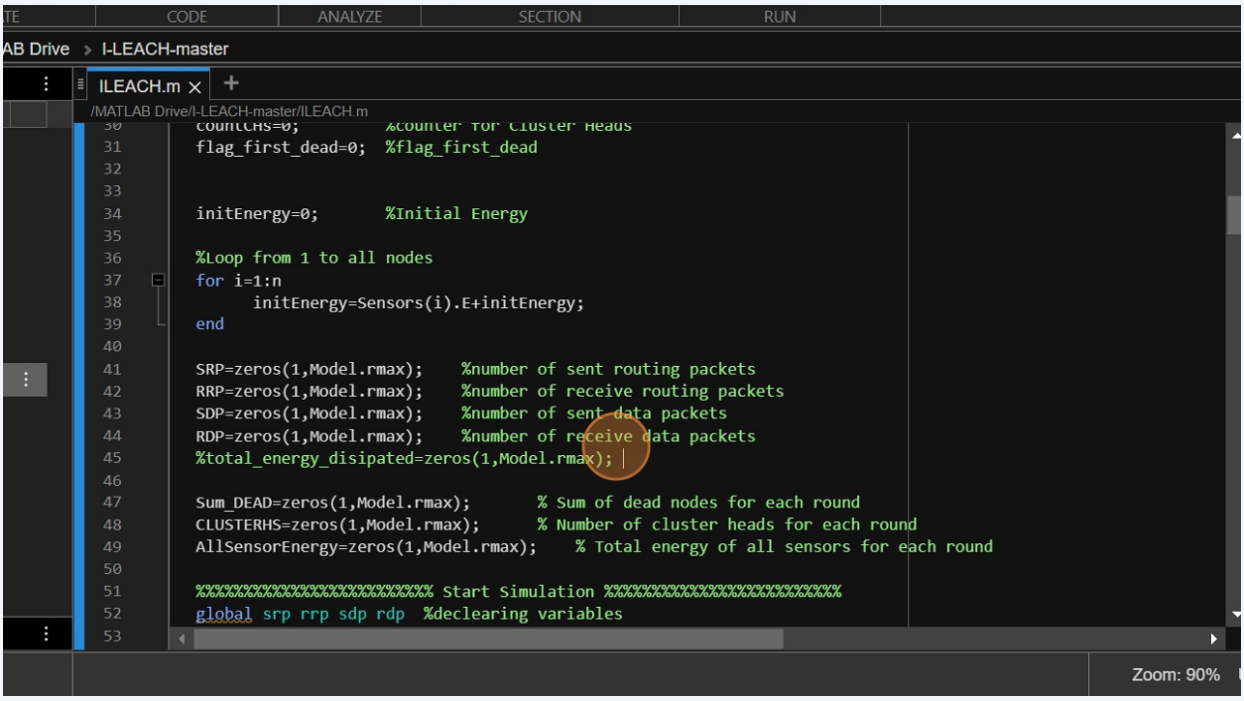
This sum can be used for various purposes in the simulation, such as calculating the total initial energy of the network or analyzing energy distribution among nodes.



36 This section initializes several arrays to keep track of various metrics over the rounds of the simulation:

- SRP: Number of sent routing packets per round.
- RRP: Number of received routing packets per round.
- SDP: Number of sent data packets per round.
- RDP: Number of received data packets per round.

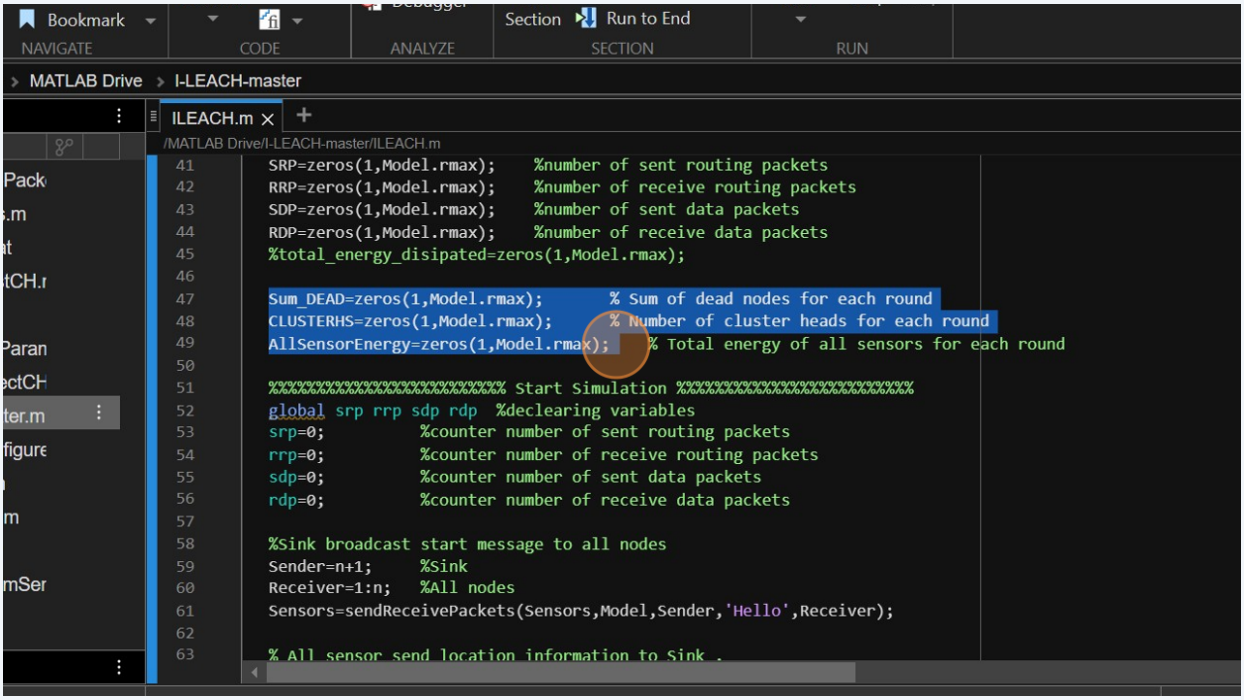
These arrays are preallocated with zeros and will be updated during the simulation to record the corresponding metrics for each round (Model.rmax rounds). These metrics are essential for evaluating the performance of the LEACH protocol and understanding the communication dynamics in the sensor network. The commented line at the end (%total_energy_disipated=zeros(1,Model.rmax);) suggests that there might be an additional metric related to total energy dissipation that is currently commented out.



37 This section initializes additional arrays to keep track of certain statistics over the rounds of the simulation:

- Sum_DEAD: Sum of dead nodes for each round.
- CLUSTERHS: Number of cluster heads for each round.
- AllSensorEnergy: Total energy of all sensors for each round.

Similar to the previous arrays, these are preallocated with zeros and will be updated during the simulation to record the corresponding statistics for each round (Model.rmax rounds). These metrics provide insights into the network's overall status, such as the number of dead nodes, the number of cluster heads, and the total energy of all sensors at each round.

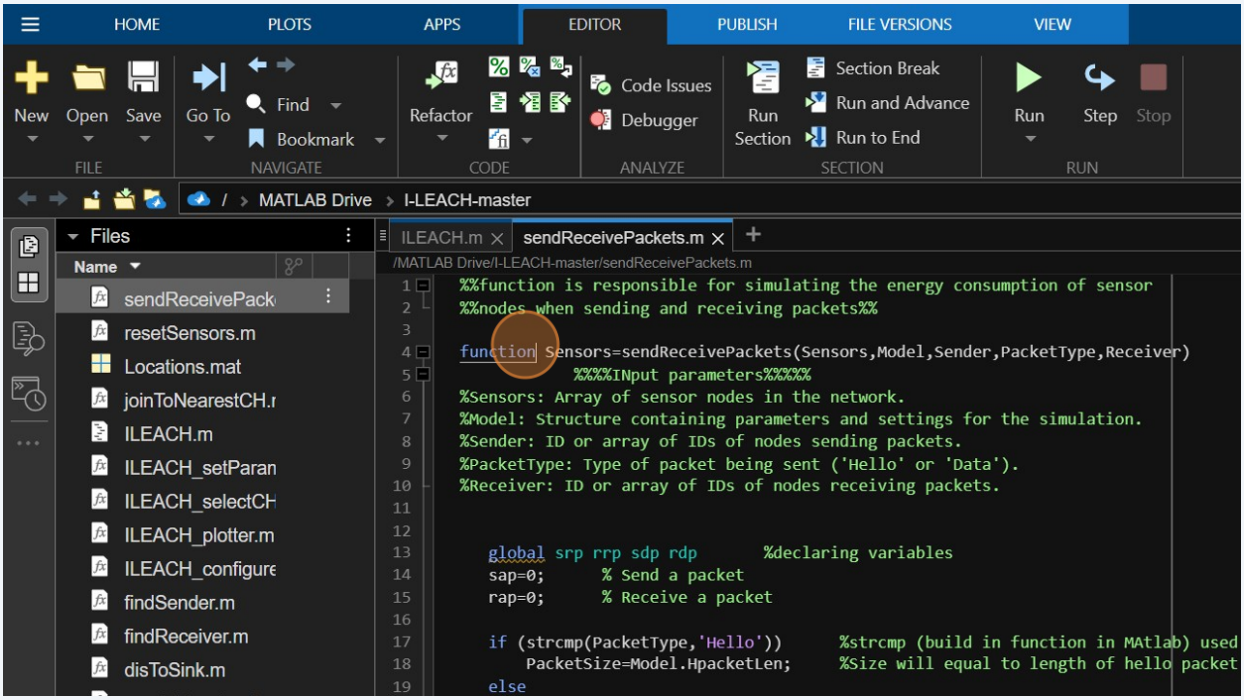


38 Now when sink broadcast start message to all nodes sendReceivePackets function is called and lets observe that function code file named sendReceivePackets

The sendReceivePackets function simulates the energy consumption of sensor nodes when sending and receiving packets. It takes the following input parameters:

- Sensors: Array of sensor nodes in the network.
- Model: Structure containing parameters and settings for the simulation.
- Sender: ID or array of IDs of nodes sending packets.
- PacketType: Type of packet being sent ('Hello' or 'Data').
- Receiver: ID or array of IDs of nodes receiving packets.

This function is crucial for modeling the energy consumption behavior of nodes during communication. Depending on the type of packet being sent ('Hello' or 'Data'), it adjusts the energy levels of sender and receiver nodes accordingly. The energy consumption is based on the transmission and reception costs specified in the energy model parameters.



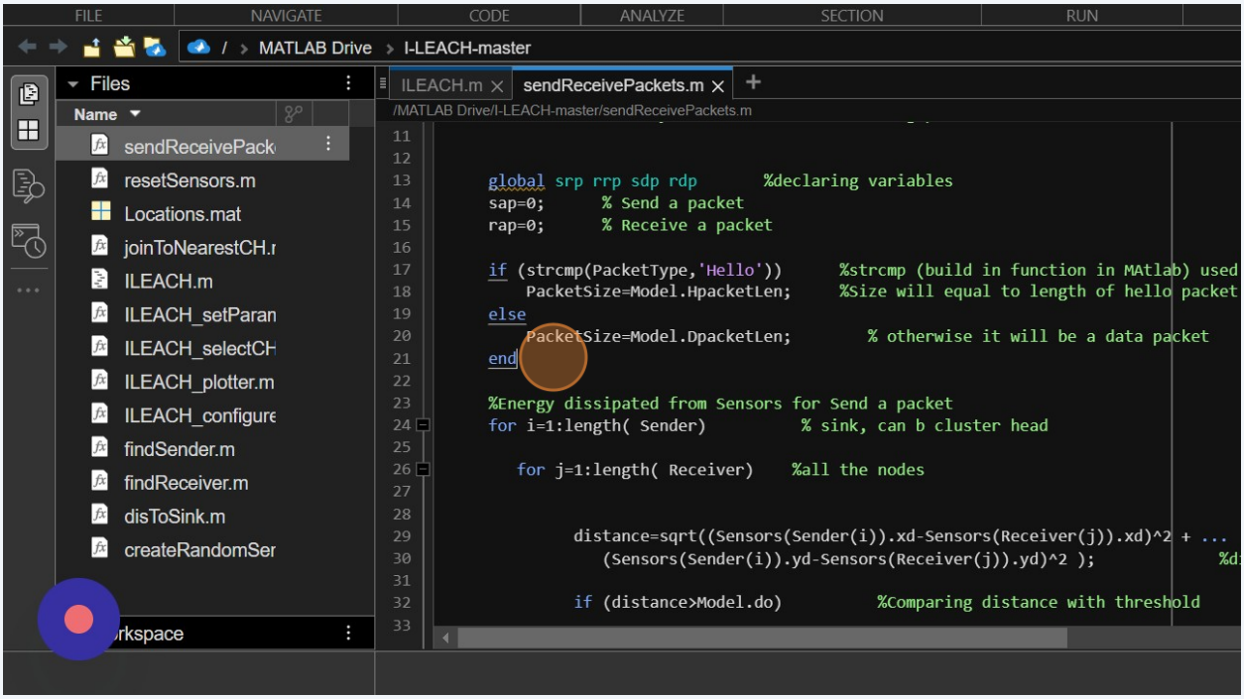
39 In this part of the code, the function is setting up some variables and conditions before proceeding with the simulation of sending and receiving packets.

• global srp rrp sdp rdp: This line declares global variables srp, rrp, sdp, and rdp, which are likely used to keep track of the number of sent routing packets, received routing packets, sent data packets, and received data packets, respectively. The use of global variables allows these counts to be accessed and modified across different functions or parts of the code.

• sap=0;: Initializes the variable sap (send a packet) to 0.

• rap=0;: Initializes the variable rap (receive a packet) to 0.

• The conditional statement if (strcmp(PacketType,'Hello')) checks whether the PacketType is equal to 'Hello'. strcmp is a built-in MATLAB function that compares two strings. If the condition is true, it sets PacketSize to Model.HpacketLen, which represents the size of a hello packet. Otherwise, it sets PacketSize to Model.DpacketLen, representing the size of a data packet



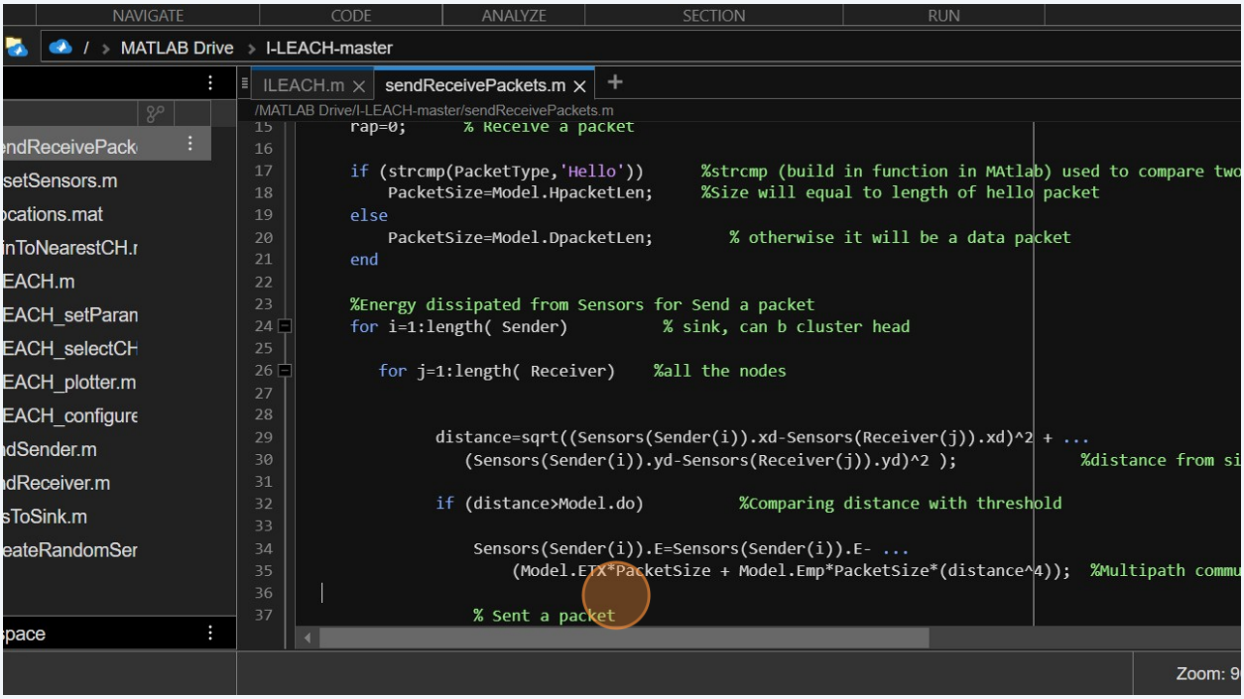
40 This part of the code calculates the energy dissipated from sensors when sending a packet. Let's break down the key components:

• The code is using a nested loop where the outer loop (for i=1:length(Sender)) iterates over all elements in the Sender array, and the inner loop (for j=1:length(Receiver)) iterates over all elements in the Receiver array.

• For each combination of Sender(i) and Receiver(j), it calculates the distance (distance) between the sender and receiver using the Euclidean distance formula.

• The conditional statement if (distance>Model.do) checks whether the calculated distance is greater than a threshold value Model.do. If the distance is greater, it means that the communication is subject to multipath conditions.

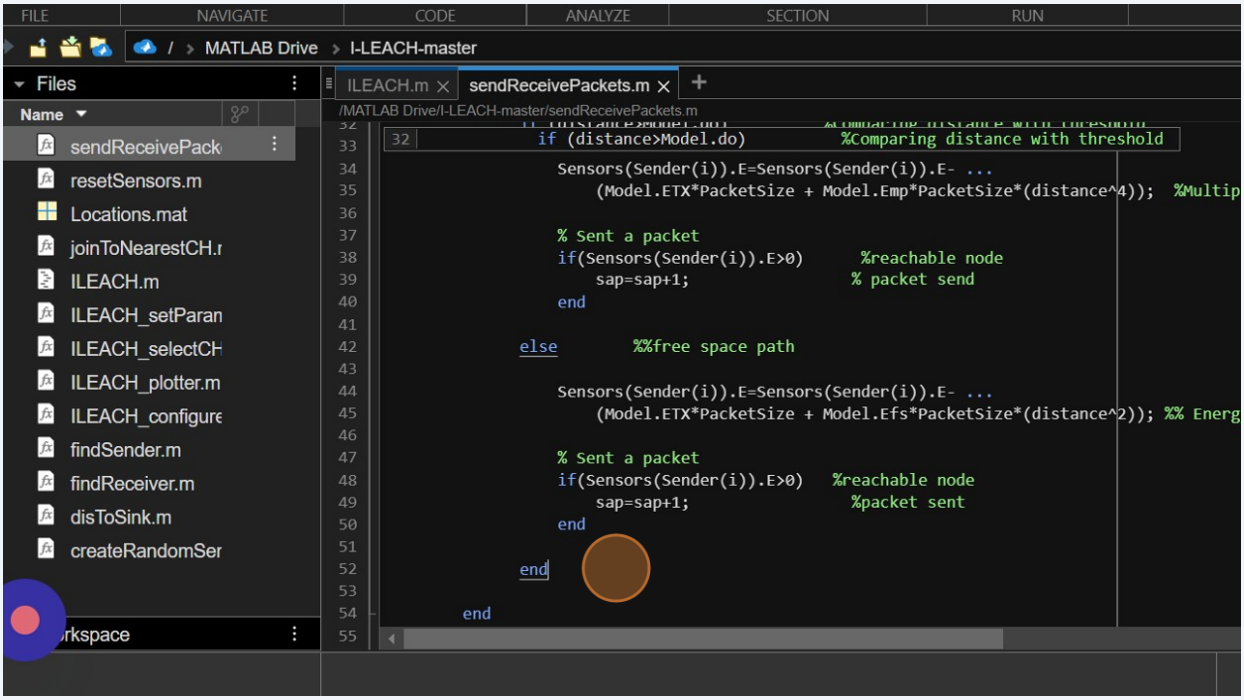
• Inside the conditional block, it calculates the energy dissipated from the sender node (Sensors(Sender(i)).E) based on the energy model. The energy dissipation is calculated as a combination of energy consumed for transmitting a bit (Model.ETX), energy consumed for multipath channel transmission (Model.Emp), and the distance between the sender and receiver. This energy dissipation model reflects the impact of distance and channel conditions on energy consumption during communication.



41 In this part of the code, it handles the scenario where the distance (distance) is less than or equal to the threshold (Model.do). This corresponds to the free-space path scenario.

• Inside the else block, it calculates the energy dissipated from the sender node (Sensors(Sender(i)).E) based on the energy model. This calculation considers the energy consumed for transmitting a bit (Model.ETX) and the energy consumption for free-space transmission (Model.Efs). The energy dissipation is proportional to the square of the distance between the sender and receiver.

• After calculating the energy dissipation, it checks if the sender node's energy level (Sensors(Sender(i)).E) is still greater than zero. If the energy level is greater than zero, it increments the sap counter, indicating that a packet has been successfully sent.

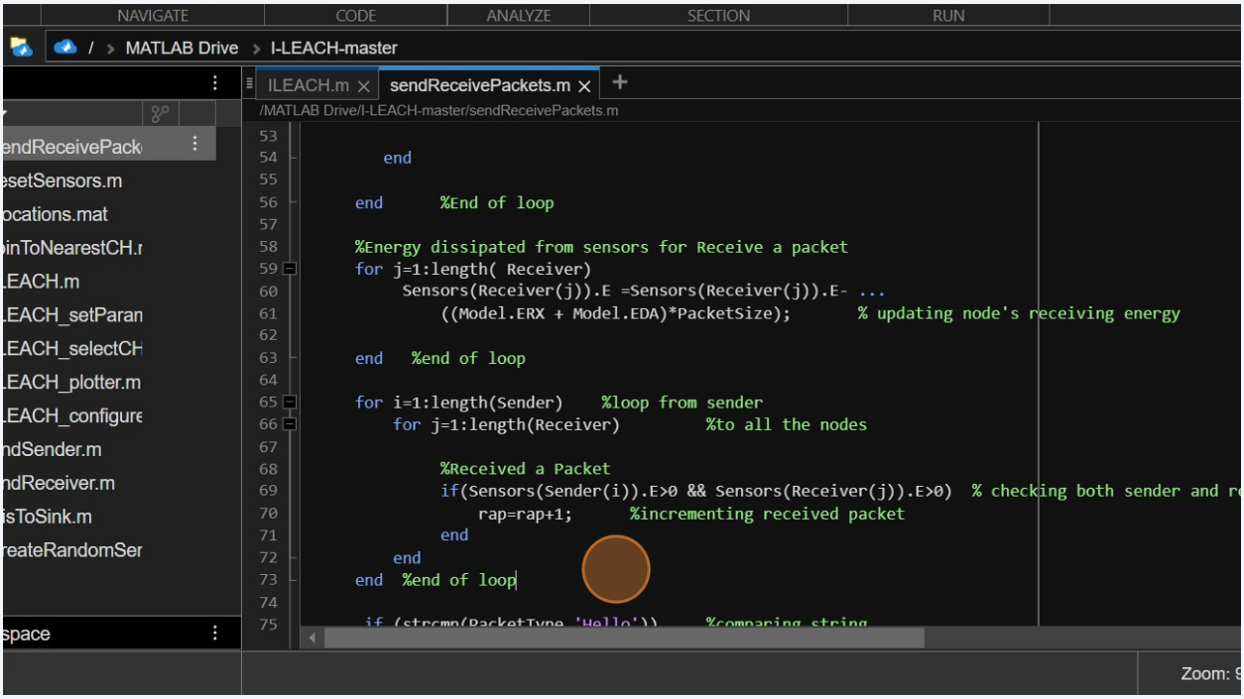


42 In this part of the code, it models the energy consumption when receiving a packet.

• Inside the first loop (for j=1:length(Receiver)), it iterates through all the receivers (Receiver) to update the energy levels of the receiver nodes. The energy dissipation is based on the energy consumed for receiving a bit (Model.ERX) and the data aggregation energy (Model.EDA). This energy dissipation is applied to each receiver in the loop.

• In the second loop (for i=1:length(Sender) nested inside another loop for j=1:length(Receiver)), it iterates through each pair of sender (Sender(i)) and receiver (Receiver(j)) nodes. If both the sender and receiver have remaining energy (Sensors(Sender(i)).E and Sensors(Receiver(j)).E are greater than zero), it increments the rap counter, indicating that a packet has been successfully received.

These loops collectively model the energy consumption associated with receiving a packet by updating the energy levels of the receiver nodes and tracking the number of received packets (rap).

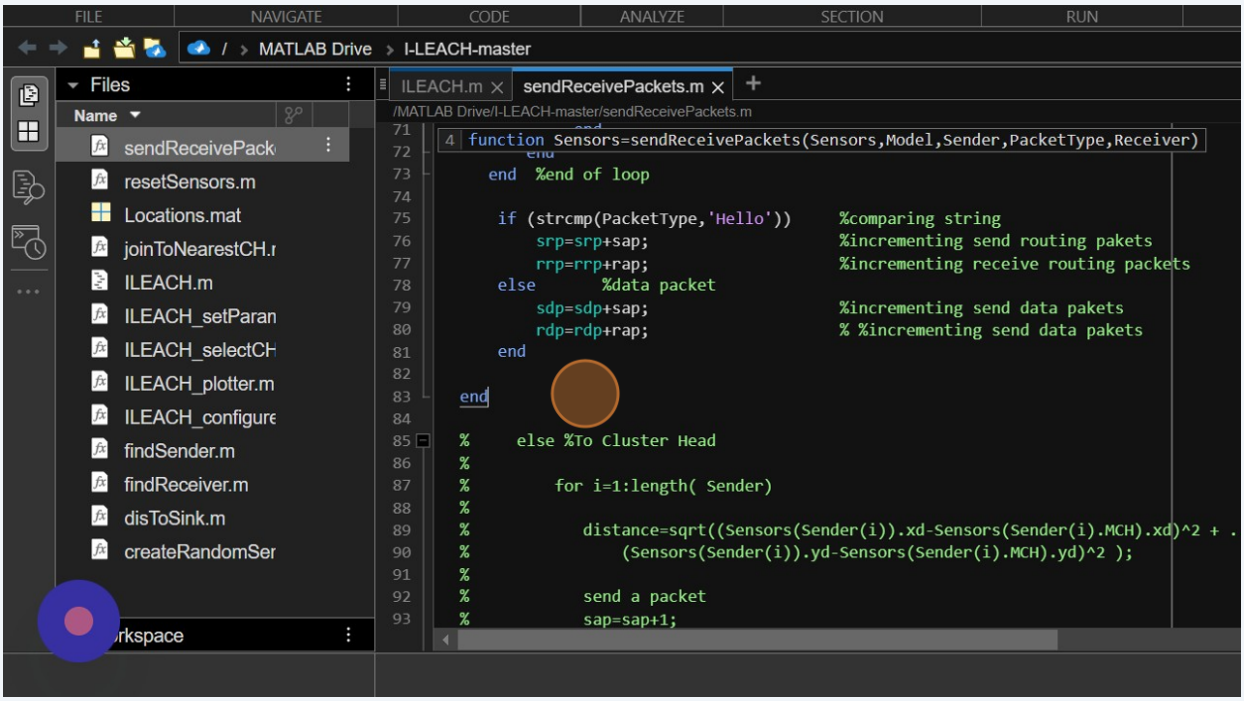


43 This part of the code distinguishes between 'Hello' packets and 'Data' packets. Depending on the type of packet (PacketType), it updates the counters for sent and received packets accordingly.

• If the packet type is 'Hello', it increments the counters for sent routing packets (srp) by adding the value of sap (send a packet) and increments the counters for received routing packets (rrp) by adding the value of rap (receive a packet).

• If the packet type is not 'Hello' (i.e., it's a 'Data' packet), it increments the counters for sent data packets (sdp) by adding the value of sap and increments the counters for received data packets (rdp) by adding the value of rap.

This way, it keeps track of the total number of sent and received packets for both routing and data packets in the simulation.



44 This commented-out code is an alternative implementation or an additional scenario for sending and receiving packets. The code here is to handle the case when a packet is sent to a cluster head (CH). However, it is currently commented out, indicating that it is not being executed in the current version of the code.

Here is a breakdown of the code inside the commented block:

1. **For each sender (**Sender(i)), calculate the distance to its associated cluster head (Sender(i).MCH).

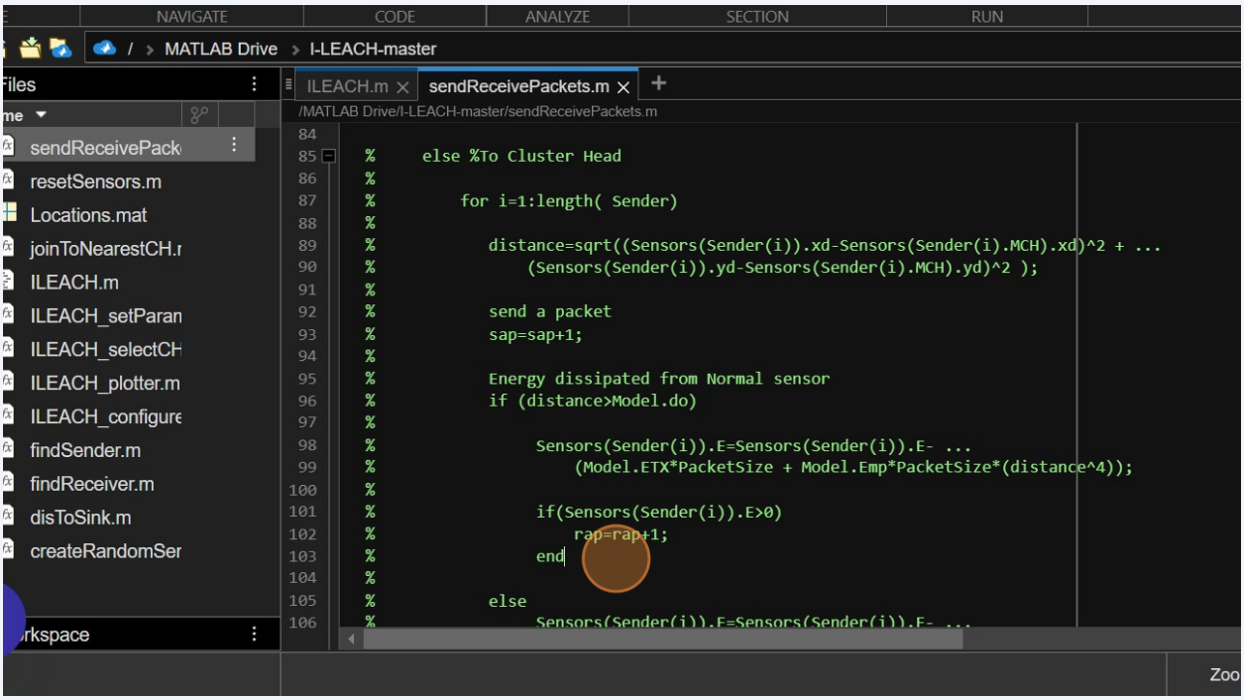
2. **Send a packet (**sap=sap+1).

3. **Calculate energy dissipated from the normal sensor based on the distance to the cluster head.**

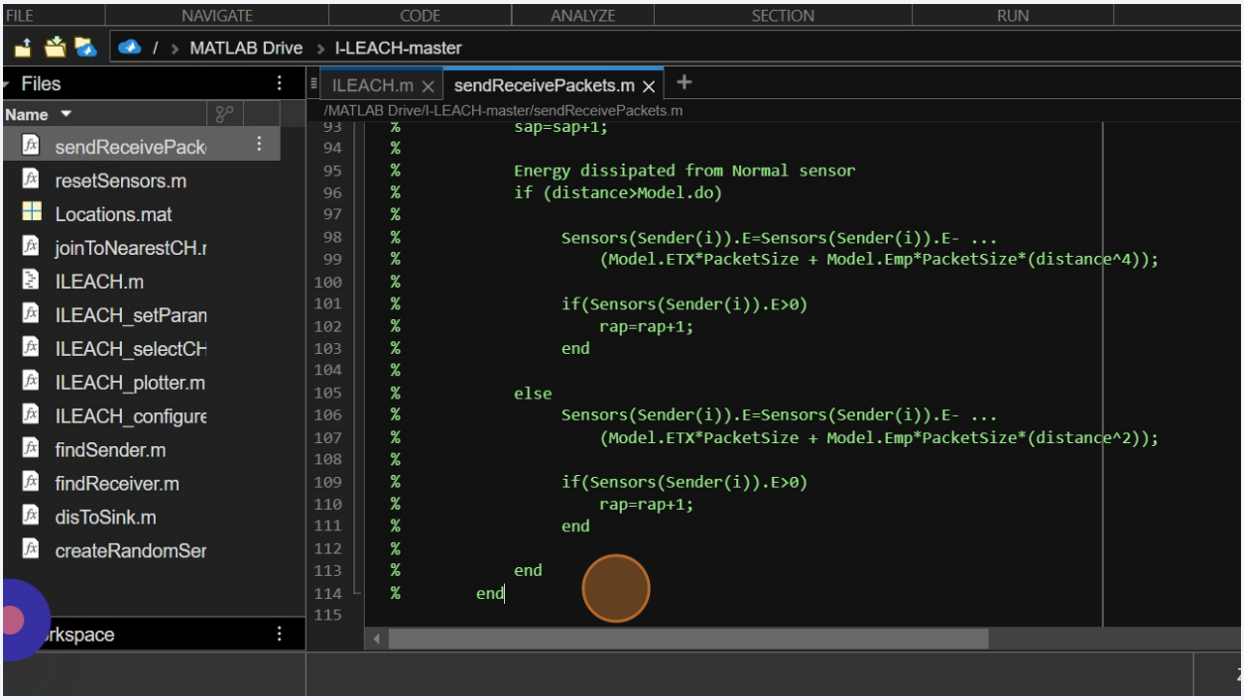
• If the distance is greater than Model.do, it uses multipath communication energy dissipation.

• If the distance is less than or equal to Model.do, it uses free space path communication energy dissipation.

4. **If the sender's energy is still positive after sending the packet, increment the received packet counter (**rap=rap+1).



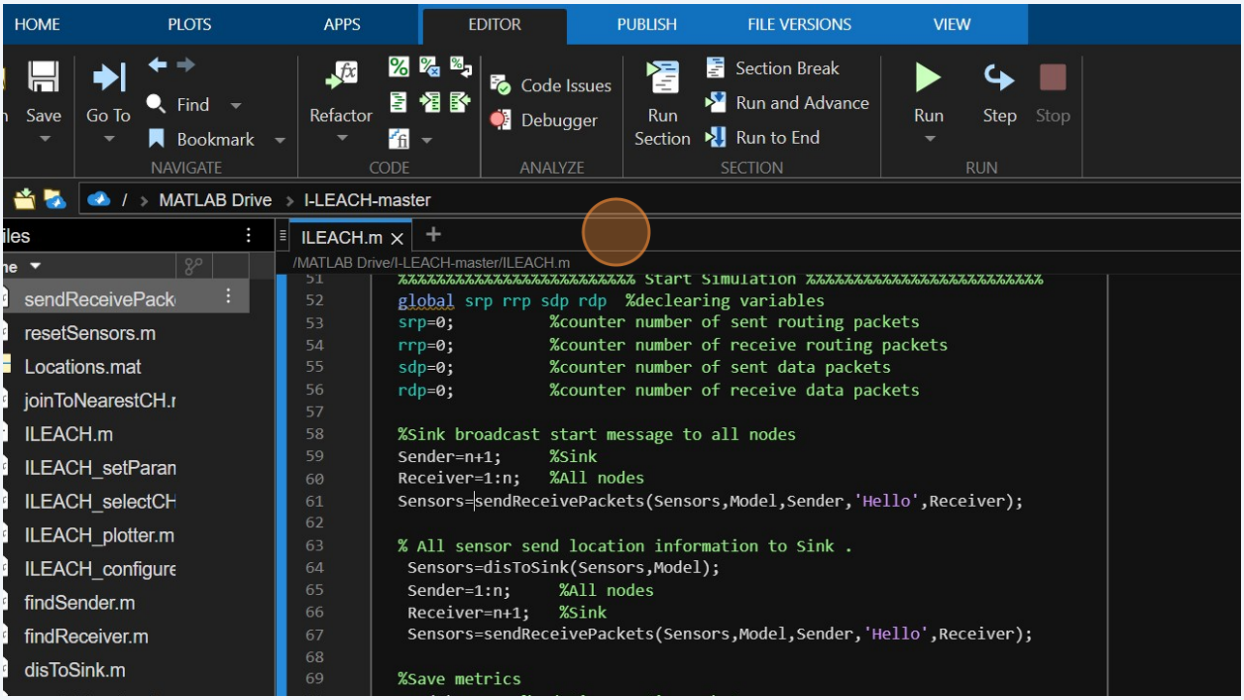
45 Continued



46 Now back to Main File ILEACH where simulation code starts
This section is setting the initial values for the counters that will keep track of the number of sent and received packets during the simulation. The counters are initialized to zero:

- srp: Counter for the number of sent routing packets.
- rrp: Counter for the number of received routing packets.
- sdp: Counter for the number of sent data packets.
- rdp: Counter for the number of received data packets.

These counters will be updated during the simulation as packets are sent and received. They provide a way to monitor and analyze the network's behavior and performance.



47 This part of the code simulates the initial broadcast of "Hello" messages from the sink to all nodes in the network, and then each node sends its location information back to the sink. Additionally, the code updates the counters for the number of sent and received routing packets (SRP and RRP) and the number of sent and received data packets (SDP and RDP) for the first round.

Here's a breakdown of the code:

1. **Sink broadcast "Hello" messages to all nodes:**

   • Sender: The sink node.

   • Receiver: An array containing all nodes in the network.

   • Sensors=sendReceivePackets(Sensors,Model,Sender,'Hello',Receiver);: This function simulates the transmission of "Hello" packets from the sink to all nodes. It updates the energy levels of the sink and nodes accordingly.

2. **All nodes send their location information to the Sink:**

   • Sender: An array containing all nodes in the network.

   • Receiver: The sink node.

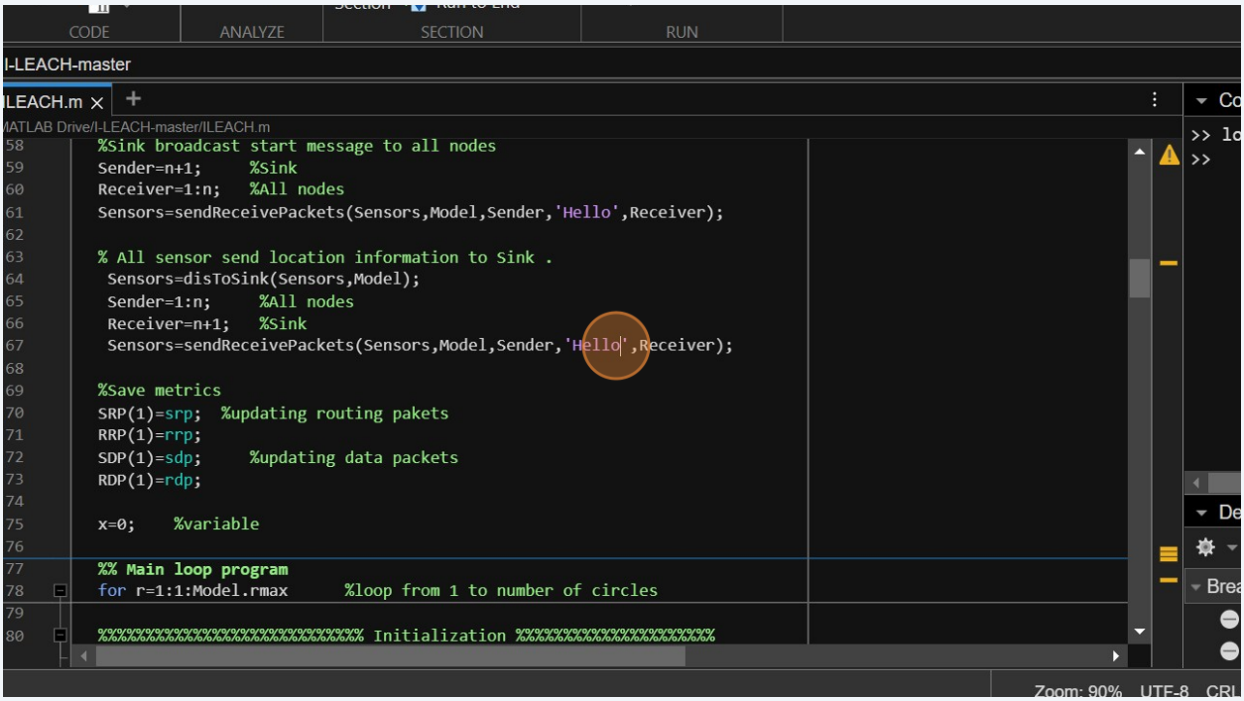
   • Sensors=sendReceivePackets(Sensors,Model,Sender,'Hello',Receiver);: This function simulates each node sending its location information to the sink. It updates the energy levels of the nodes and the sink.

3. **Save metrics for the first round:**

   • SRP(1), RRP(1), SDP(1), RDP(1): Update the counters with the number of sent routing packets, received routing packets, sent data packets, and received data packets for the first round.

4. **Reset the variable** x **to 0:**

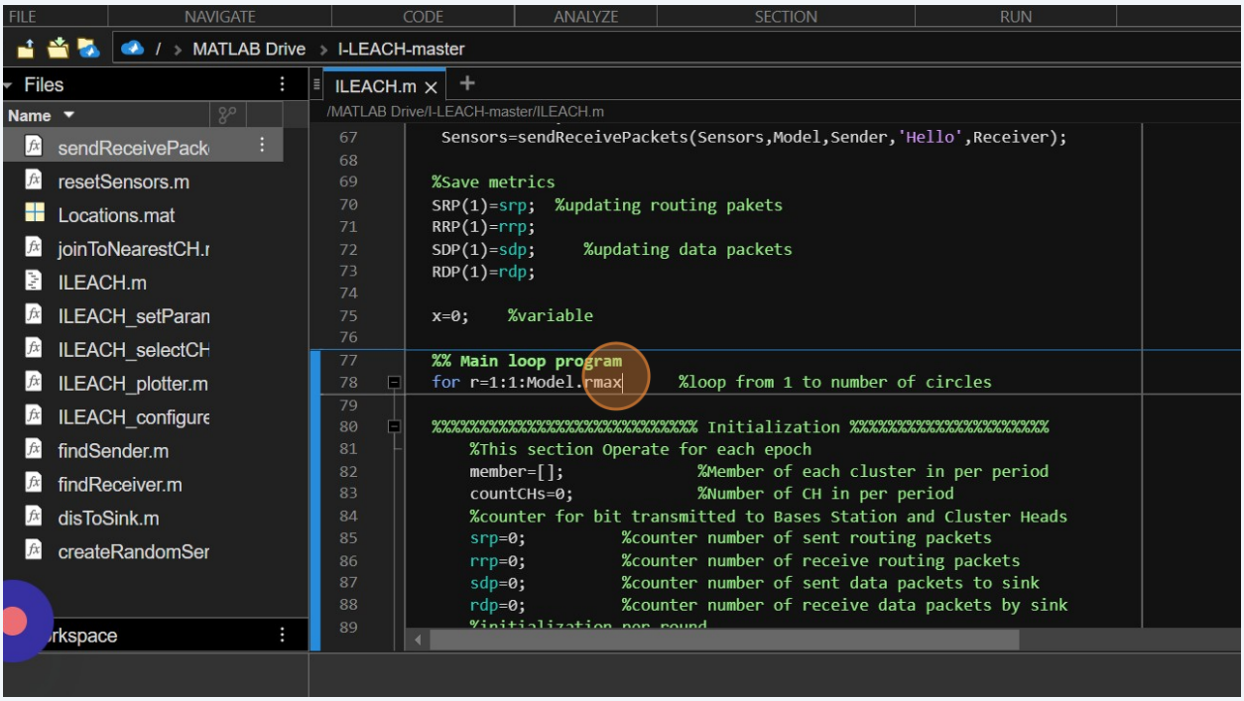
   • x=0;: This variable is declared but not used in this part of the code



48

• for r=1:1:Model.rmax: This line initiates a for loop that runs from r=1 to r=Model.rmax with a step size of 1. The loop variable r represents the current round of the simulation.

The code inside this loop will be responsible for executing the simulation logic for each round.



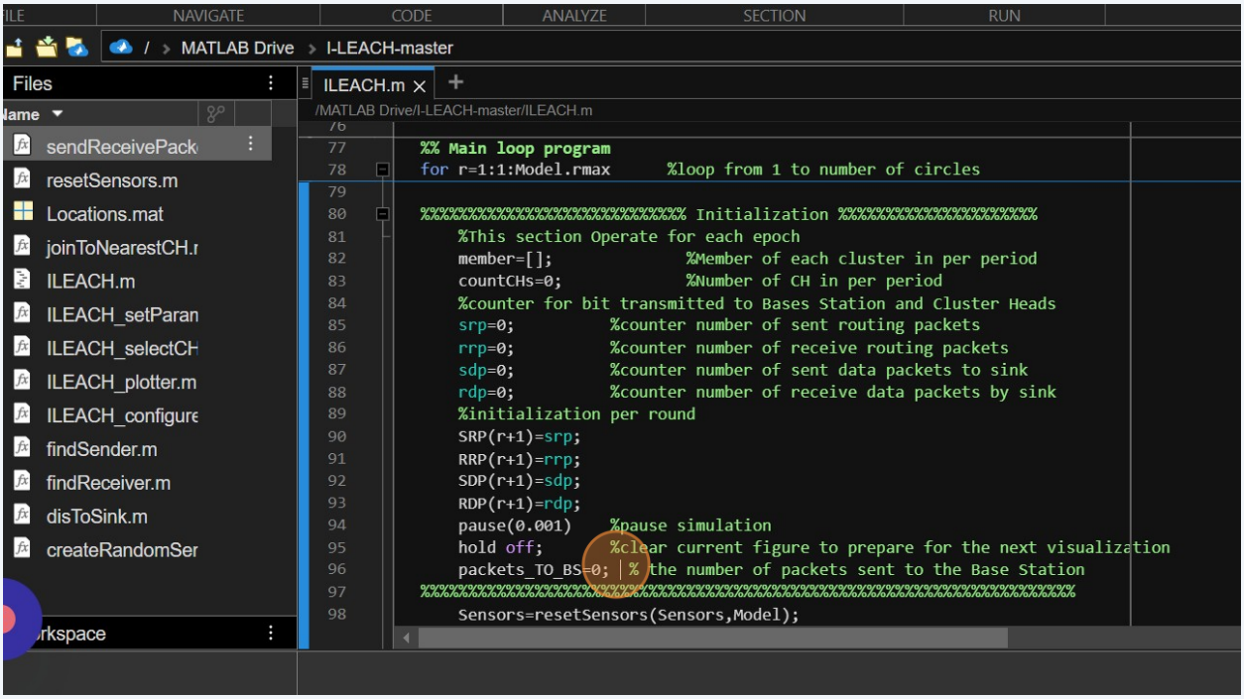
49 **This section of the code appears to be related to the initialization and setup for each epoch (simulation round).**

```
% This section Operates for each epoch (round)
member=[]; % Member of each cluster in each period
countCHs=0; % Number of Cluster Heads in each period

% Counter for bits transmitted to Base Station and Cluster Heads
srp=0; % Counter for the number of sent routing packets
rrp=0; % Counter for the number of received routing packets
sdp=0; % Counter for the number of sent data packets to the sink
rdp=0; % Counter for the number of received data packets by the sink

% Initialization per round
SRP(r+1)=srp; % Update the counters for sent routing packets
RRP(r+1)=rrp; % Update the counters for received routing packets
SDP(r+1)=sdp; % Update the counters for sent data packets
RDP(r+1)=rdp; % Update the counters for received data packets

pause(0.001) % Pause the simulation for a short duration
hold off; % Clear the current figure to prepare for the next visualization
packets_TO_BS=0; % Variable for counting packets sent to the Base Station
```



50

• Sensors=resetSensors(Sensors,Model);: This function resets the sensors to their initial state. It's likely that the initial state includes information about whether a node was a cluster head in the previous round (G attribute).

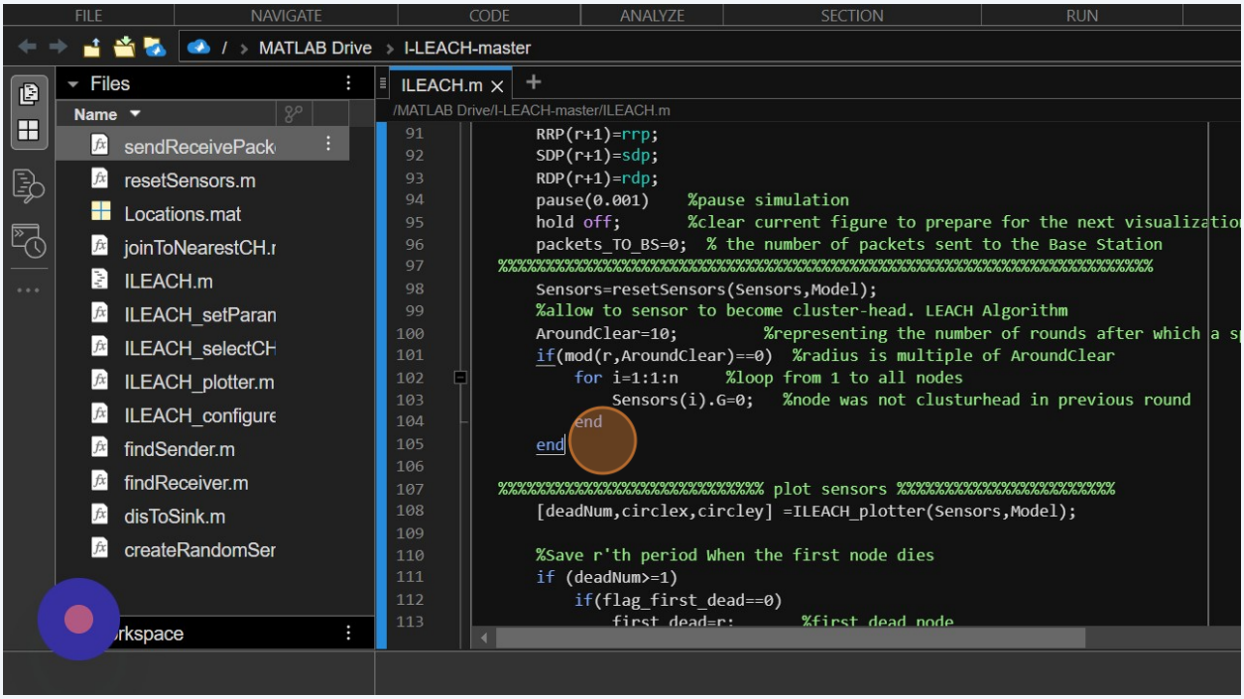
• AroundClear=10;: This variable represents the number of rounds after which a specific action will be taken. This action, as we can see in the next block, is resetting the 'G' attribute for all nodes.

• if(mod(r,AroundClear)==0): Checks if the current round r is a multiple of AroundClear. If it is, the following block of code is executed.

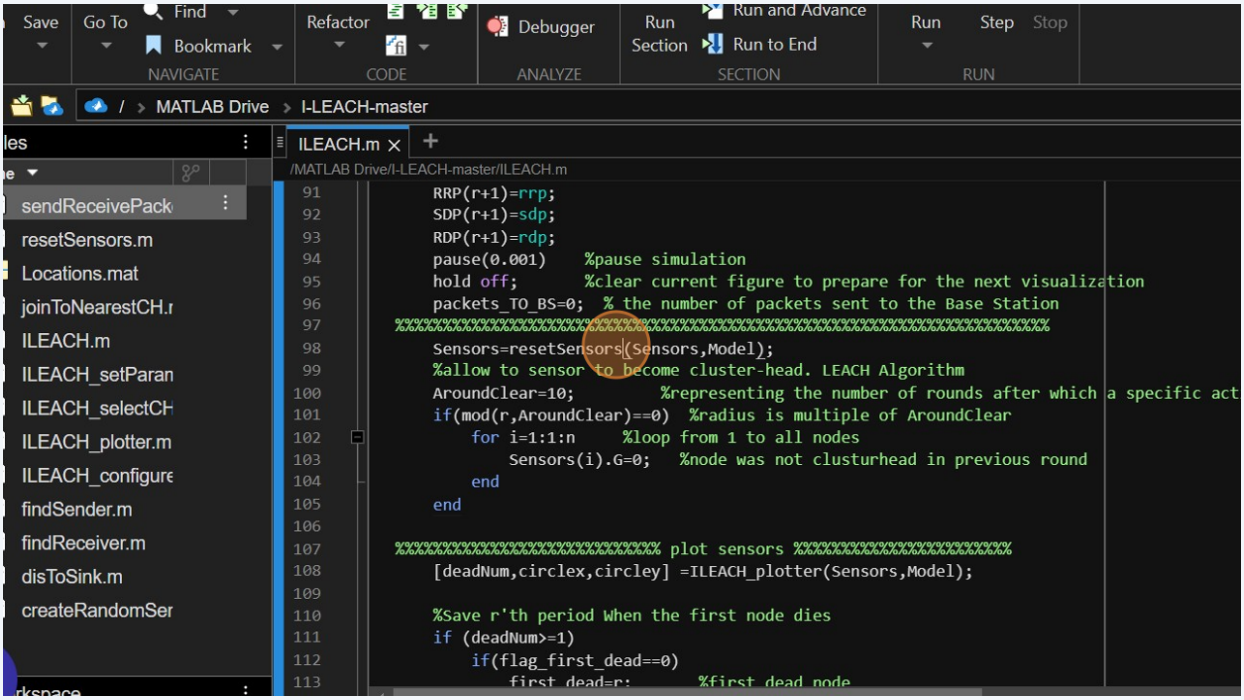
• for i=1:1:n: Loops through all nodes.

• Sensors(i).G=0;: Resets the 'G' attribute for each node, indicating that the node was not a cluster head in the previous round.

This process seems to be part of the LEACH protocol where cluster heads are selected periodically based on the defined parameter AroundClear.



51 Now lets observe the resetSensors function



52 Here's what each part does:

• function Sensors=resetSensors(Sensors,Model): This function takes as input the Sensors array and the Model structure and returns the modified Sensors array.

• n=Model.n;: This line assigns the number of nodes (n) from the Model structure.

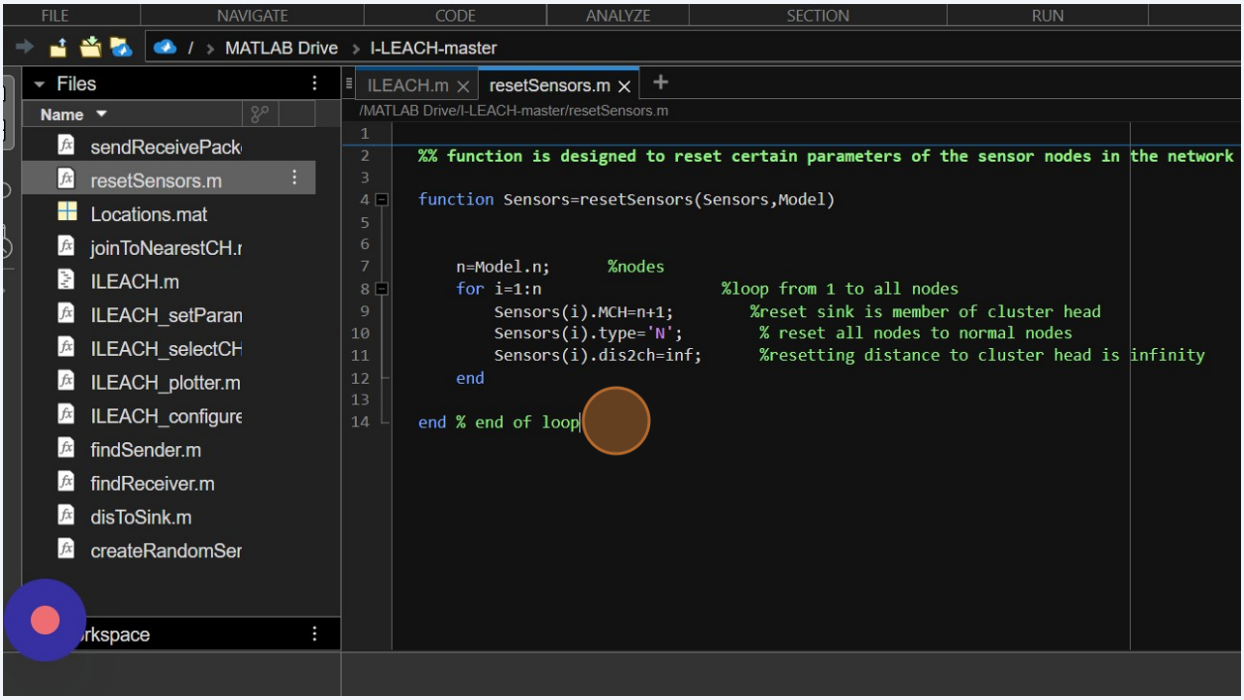
• for i=1:n: The loop iterates from 1 to the total number of nodes (n).

• Sensors(i).MCH=n+1;: It resets the MCH (Member of CH) attribute of each node to be equal to n+1, indicating that the node is a member of the sink or base station. This is typical in LEACH-like protocols where the sink plays a special role.

• Sensors(i).type='N';: It resets the type attribute of each node to be 'N', indicating that the node is a normal node, not a cluster head.

• Sensors(i).dis2ch=inf;: It resets the dis2ch attribute (distance to the cluster head) to infinity. This indicates that the distance to the cluster head is not known or undefined.

The purpose of this function is to prepare the sensor nodes for the next round of the simulation by resetting certain parameter

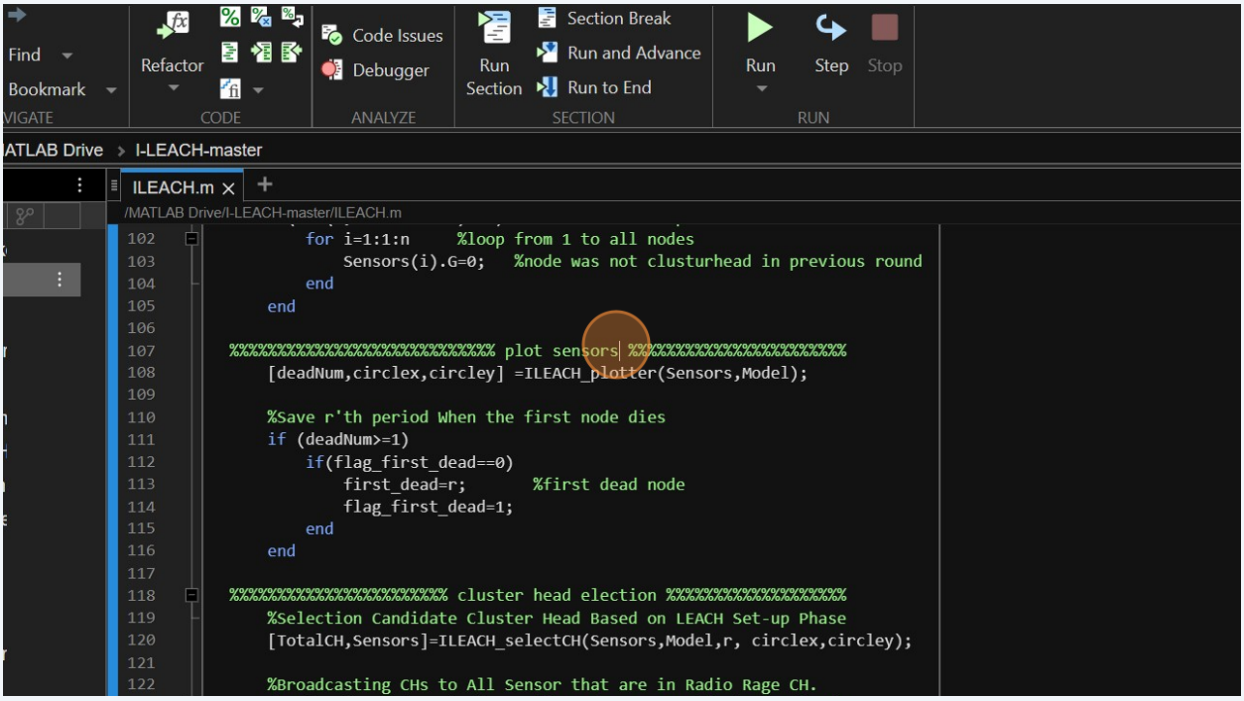


53

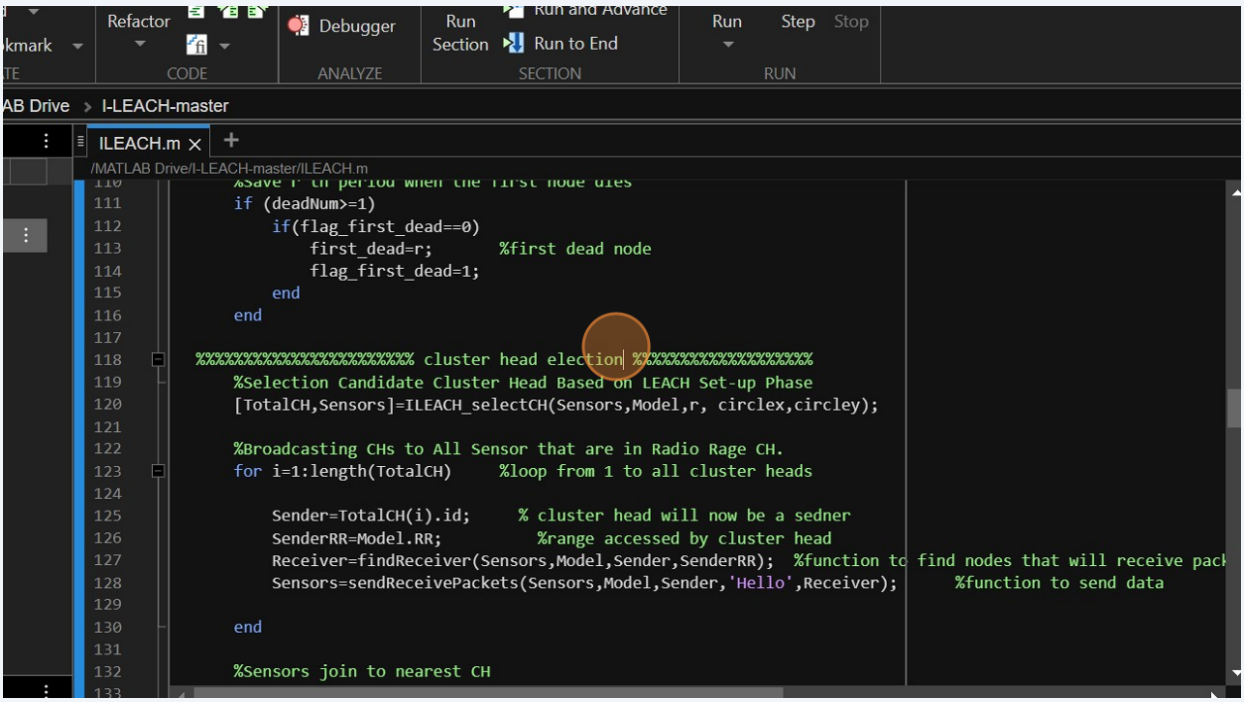
• ILEACH_plotter: This function is responsible for visualizing the state of the sensor network in the LEACH simulation. It takes the Sensors array and the Model structure as input and returns three outputs: deadNum (the number of dead nodes), circlex, and circley (coordinates of circular regions for communication range visualization).

• If there is at least one dead node (deadNum >= 1), it checks whether the flag_first_dead is 0. If it is, it means this is the first time a node has died in the simulation. It records the current round r as the first_dead round and sets flag_first_dead to 1 to indicate that the first dead node is recorded.

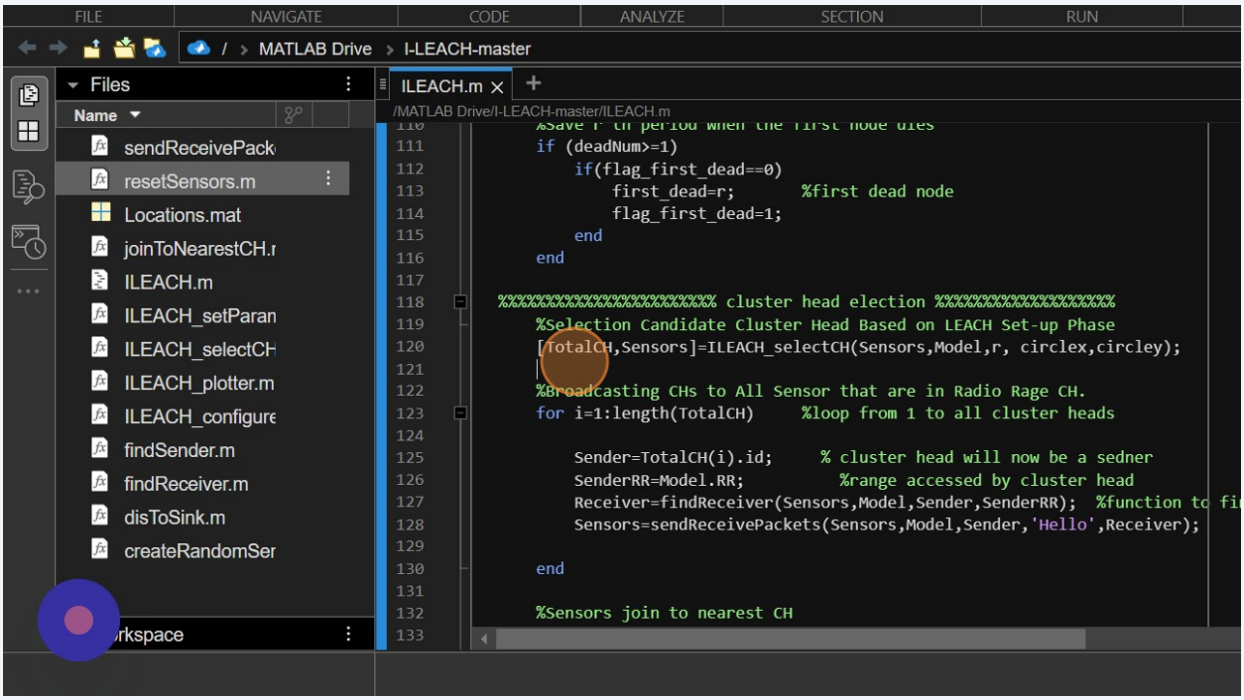
This part of the code seems to be monitoring the rounds during which nodes die in the simulation. The first_dead variable is used to keep track of the round when the first node dies



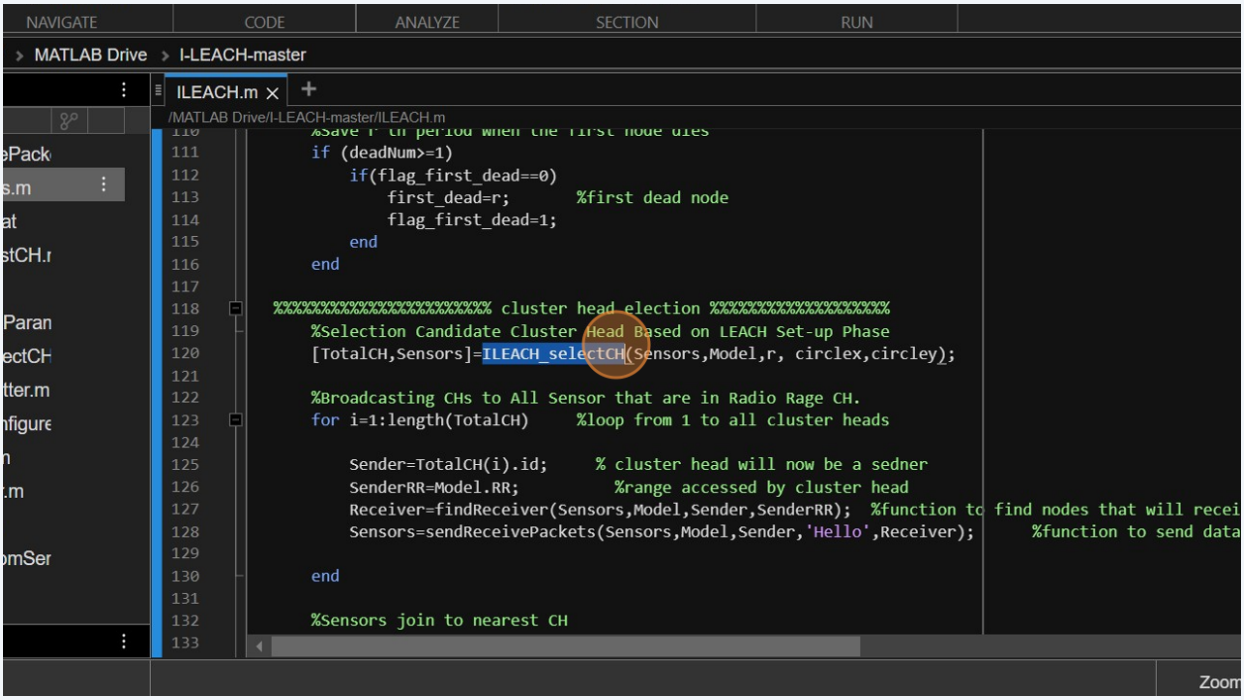
54 Now comes the Cluster Head Election for each cluster



55 A function named ILEACH_selectCH is called and passed some paremeters



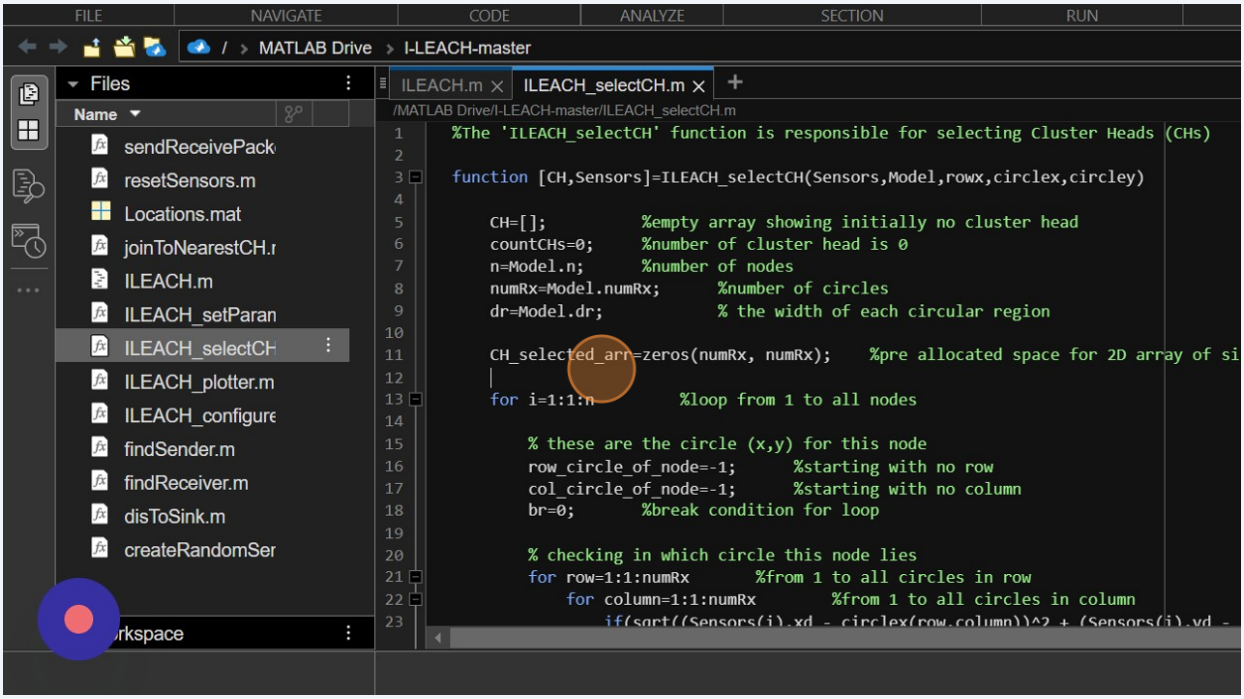
56 Now lets observe this function



57 This function ILEACH_selectCH seems to be responsible for selecting Cluster Heads (CHs) in the LEACH simulation. Let's break down the code:

CH: It is an empty array that will be used to store the IDs of selected Cluster Heads.

- Sensors: This is the array of sensor nodes.
- Model: The structure containing simulation parameters.
- rowx: It seems to be the current row index in the circular grid pattern.
- circlex and circley: These are arrays storing the coordinates of circular regions for communication range visualization.
- CH: Initialized as an empty array to store the selected Cluster Heads.
- countCHs: Initialized to 0, representing the count of selected Cluster Heads.
- n: Number of sensor nodes.
- numRx: The number of circular regions.
- dr: The width of each circular region.
- CH_selected_arr: A 2D array of zeros with a size equal to the number of circles. It seems to be used for tracking the selected CHs in each circular region.



58 Here's what's happening:

• The outer loop (for i=1:1:n) iterates over all sensor nodes in the network.

• row_circle_of_node and col_circle_of_node are initialized as -1, indicating that the node has not been assigned to any circular region.

• br is used as a flag to break out of the loop once the circular region is found for the current node.

Now, let's look at the inner loop:

• The inner loop (for row=1:1:numRx) iterates over the rows of the circular regions.

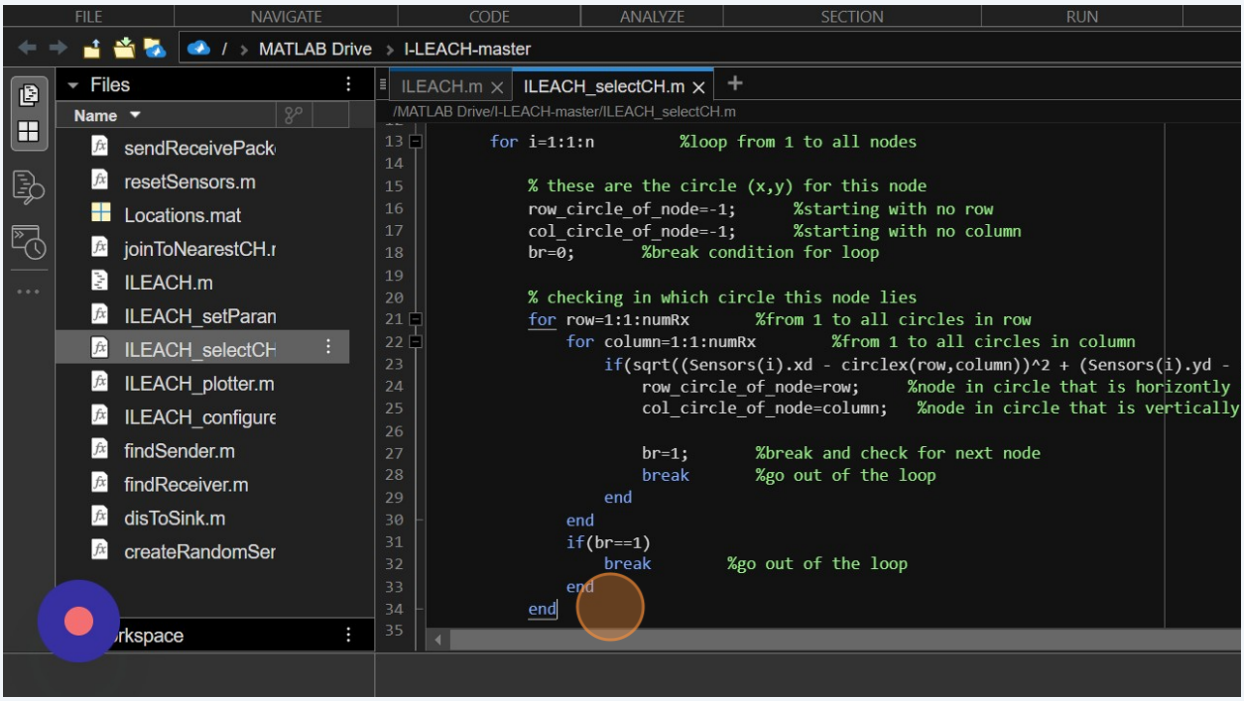
• The nested loop (for column=1:1:numRx) iterates over the columns of the circular regions.

• The condition checks whether the Euclidean distance between the sensor node's coordinates and the center of the circular region is less than or equal to half of the circular region's width (dr/2).

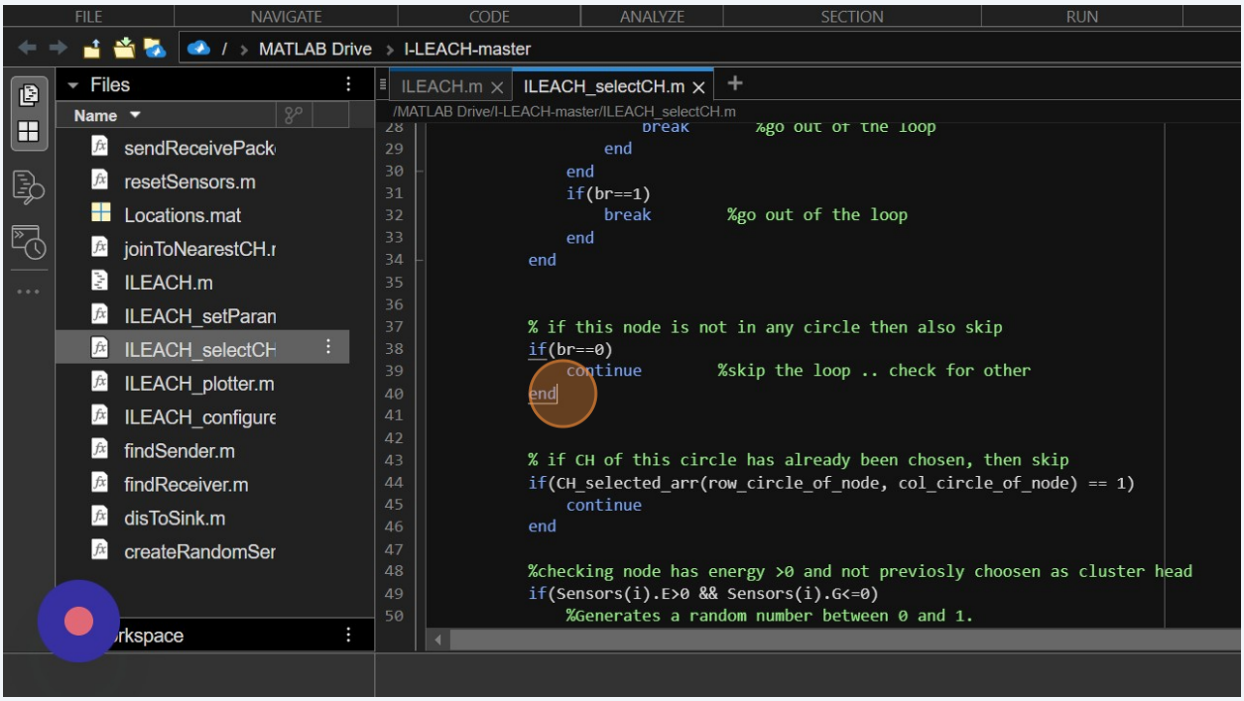
• If the condition is true, it means the sensor node lies within the circular region, and row_circle_of_node and col_circle_of_node are set to the corresponding row and column indices.

• The flags br and break are used to break out of both loops once the circular region is found.

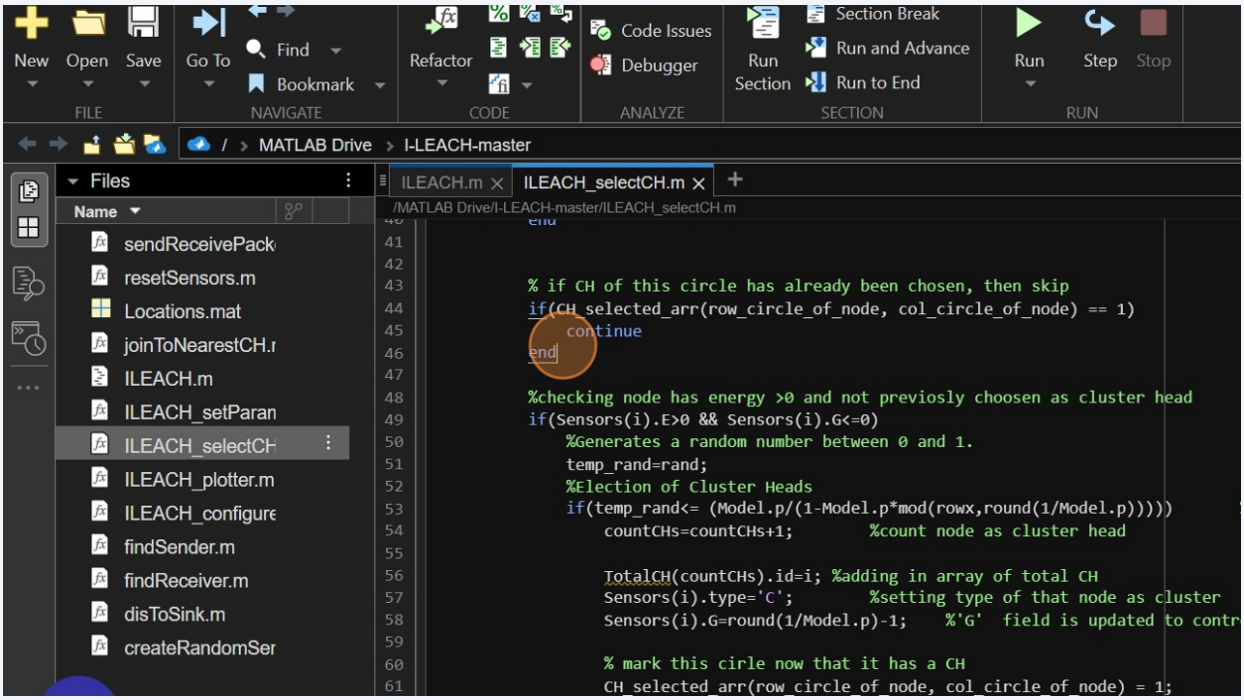
This logic is useful for determining the position of a sensor node within the grid of circular regions.



59 If br is equal to 0 (indicating that the sensor node was not found within any circular region), the continue statement is executed, causing the code to skip the remaining part of the loop for the current sensor node and move on to the next iteration of the loop, where the next sensor node is considered

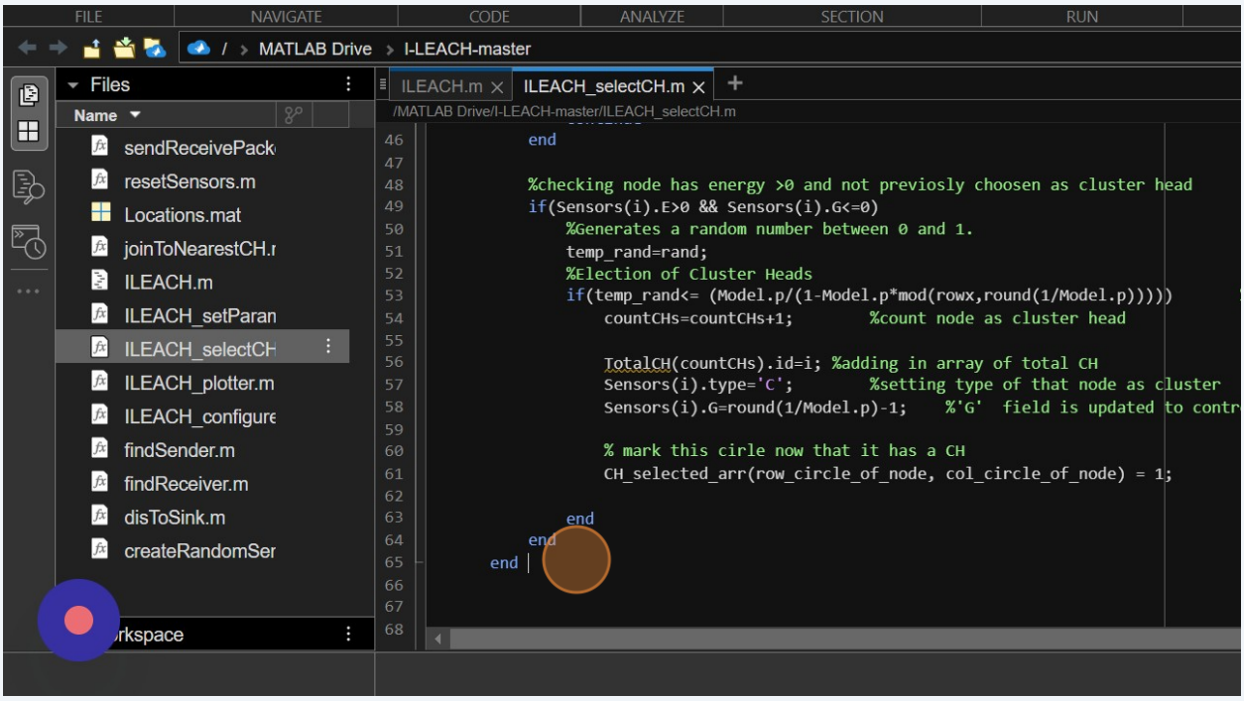


60 This part of the code checks whether a Cluster Head (CH) has already been chosen for the circular region where the current sensor node is located. If a CH has already been selected for that specific circle (as indicated by CH_selected_arr(row_circle_of_node, col_circle_of_node) == 1), the continue statement is executed. This means that the code skips the remaining part of the loop for the current sensor node and moves on to the next iteration of the loop.

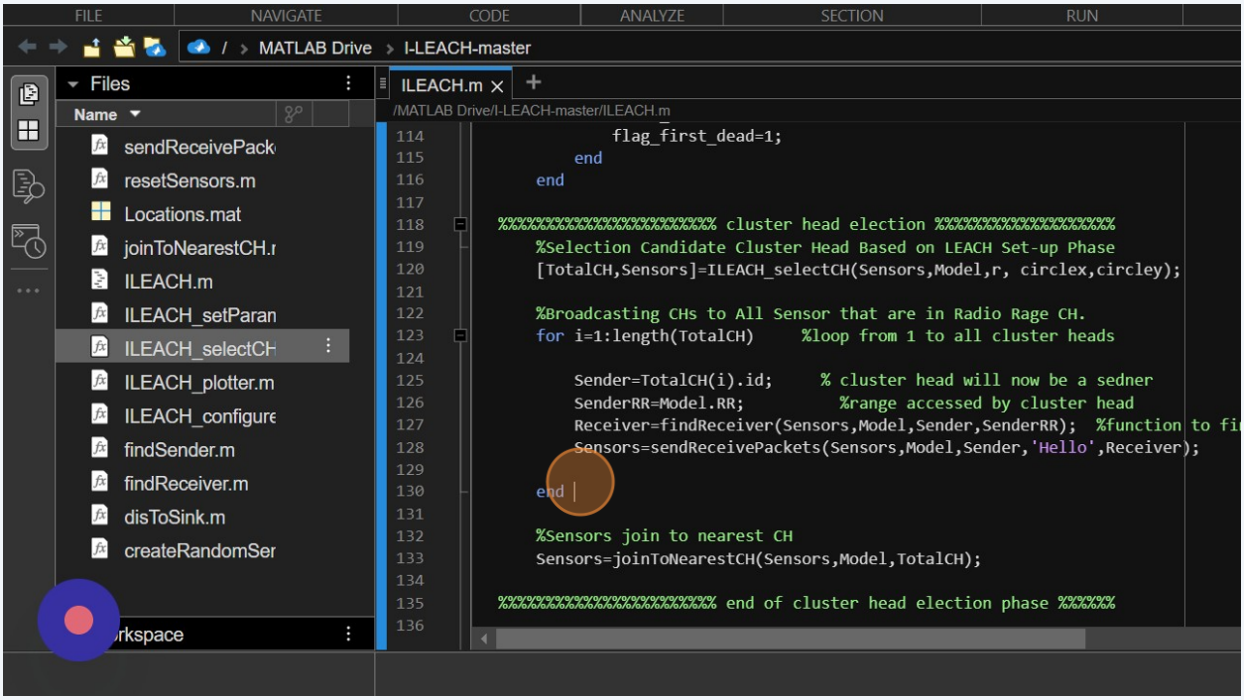


61 This part of the code checks whether a sensor node has energy greater than zero (Sensors(i).E > 0) and has not been previously chosen as a cluster head (Sensors(i).G <= 0). If these conditions are met, a random number (temp_rand) is generated between 0 and 1. If this random number is less than or equal to a certain probability, the node is elected as a cluster head. The probability of being elected as a cluster head is determined by the LEACH algorithm.

The countCHs variable keeps track of the total number of cluster heads. The information about the chosen cluster heads is stored in the TotalCH array. Additionally, the type of the selected node is set to 'C' to indicate that it is a cluster head. The Sensors(i).G field is updated to control the periodicity of cluster head selection. Finally, the circular region where a cluster head has been chosen is marked in the CH_selected_arr array to ensure that only one cluster head is selected per circular region.



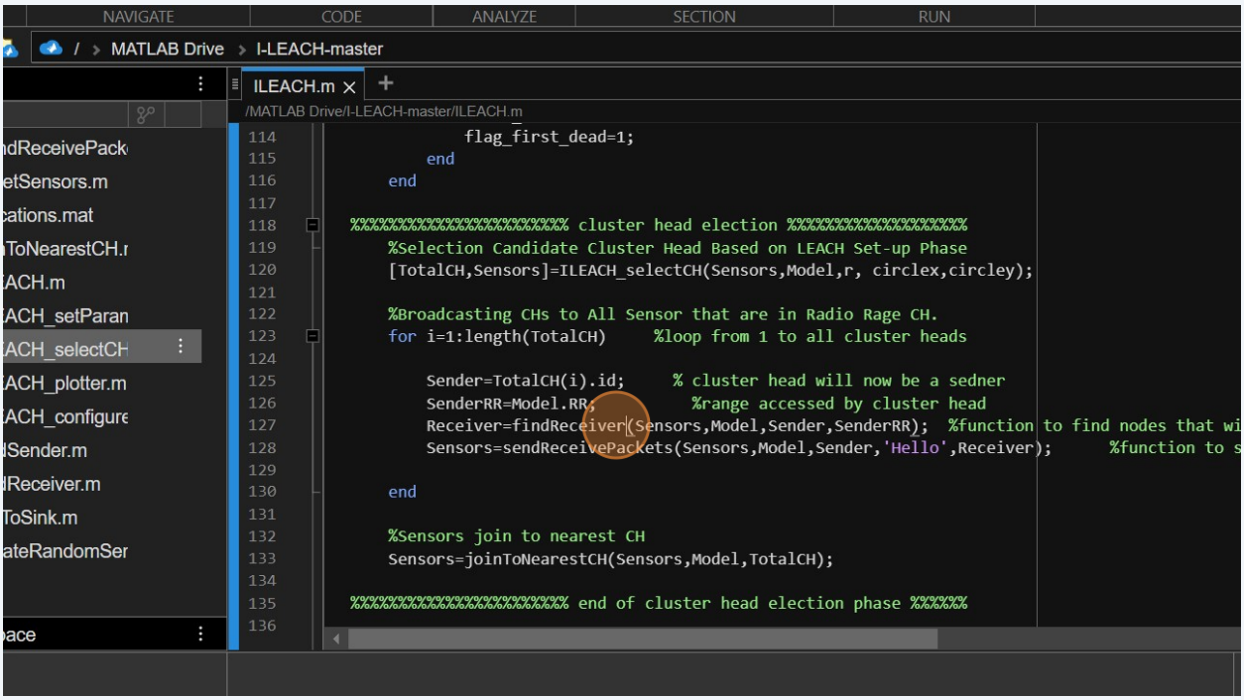
62 Click here.



63 Here's what each line does:

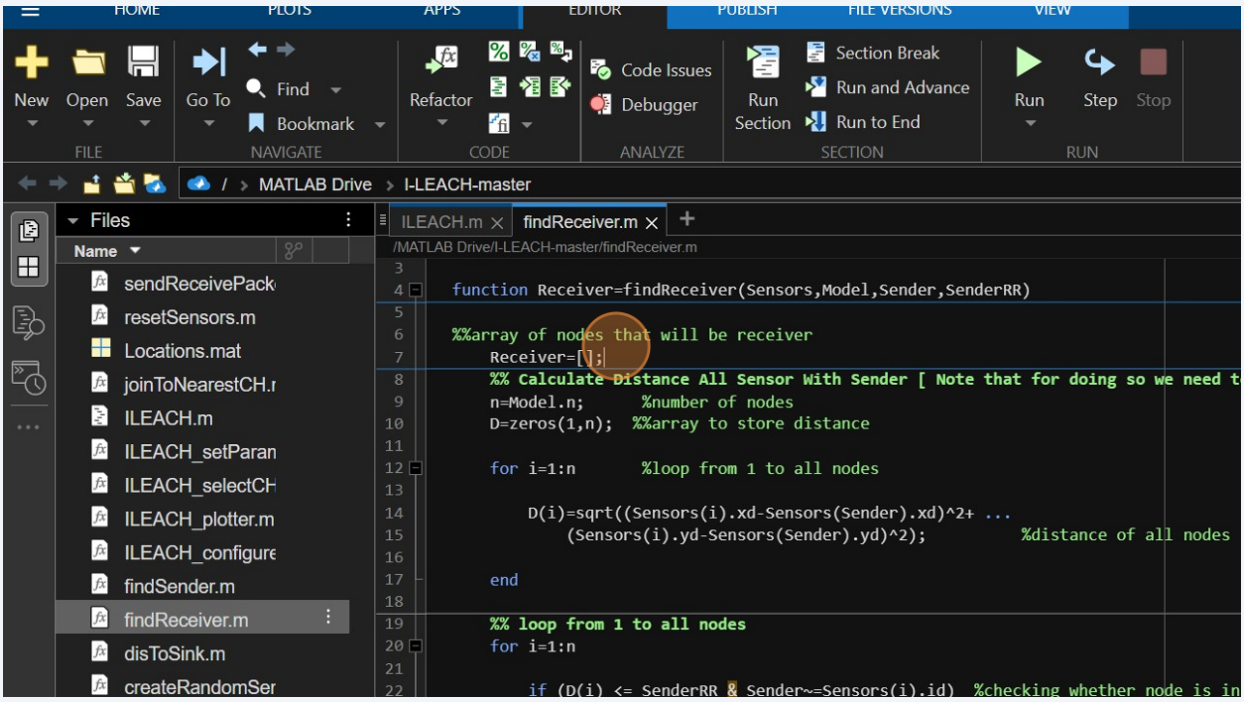
- for i = 1:length(TotalCH): Loop through all cluster heads stored in the TotalCH array.
- Sender = TotalCH(i).id;: Set the current cluster head as the sender.
- SenderRR = Model.RR;: Set the radio range of the sender (cluster head).
- Receiver = findReceiver(Sensors, Model, Sender, SenderRR);: Find nodes that are in the radio range of the current cluster head.
- Sensors = sendReceivePackets(Sensors, Model, Sender, 'Hello', Receiver);: Use the sendReceivePackets function to simulate the sending of "Hello" packets from the cluster head to the identified receivers.

This part of the code ensures that each cluster head broadcasts "Hello" packets to all sensor nodes within its radio range.



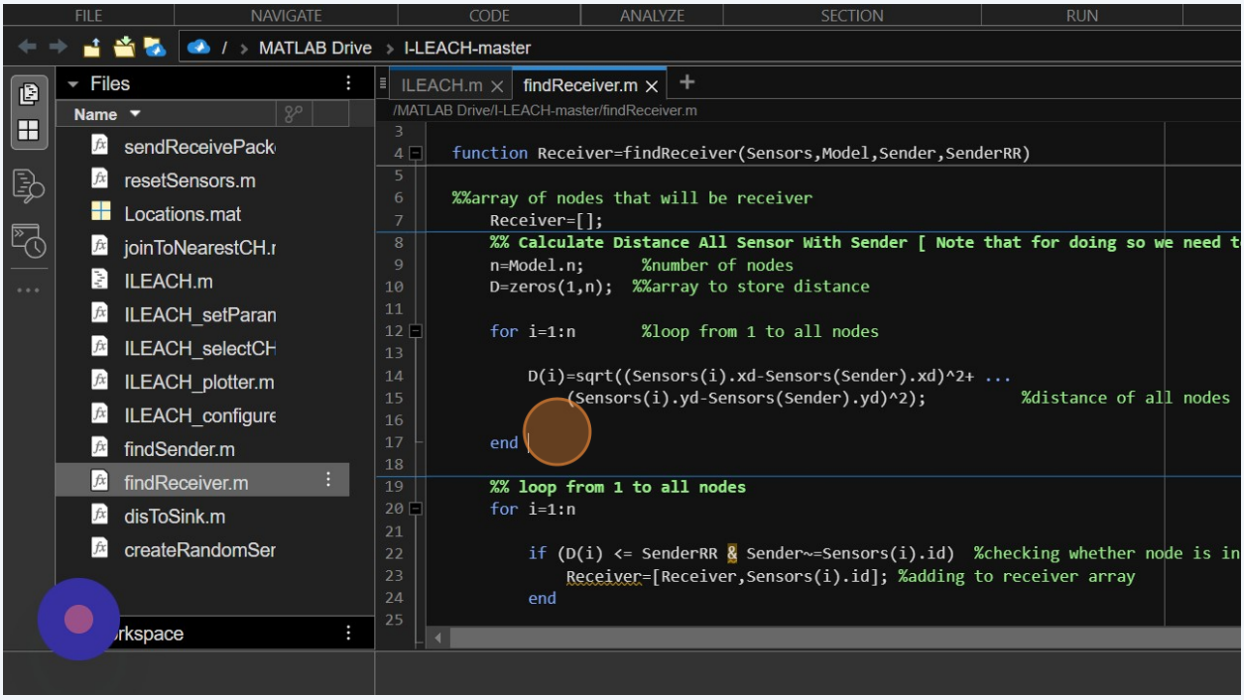
64 The function findReciever has a file of code lets observe that here:
Receiver: An array that will store the IDs of potential receiver nodes.

• Calculate Distance of All Sensors with Sender: It calculates the Euclidean distance between the specified sender and all other sensor nodes.



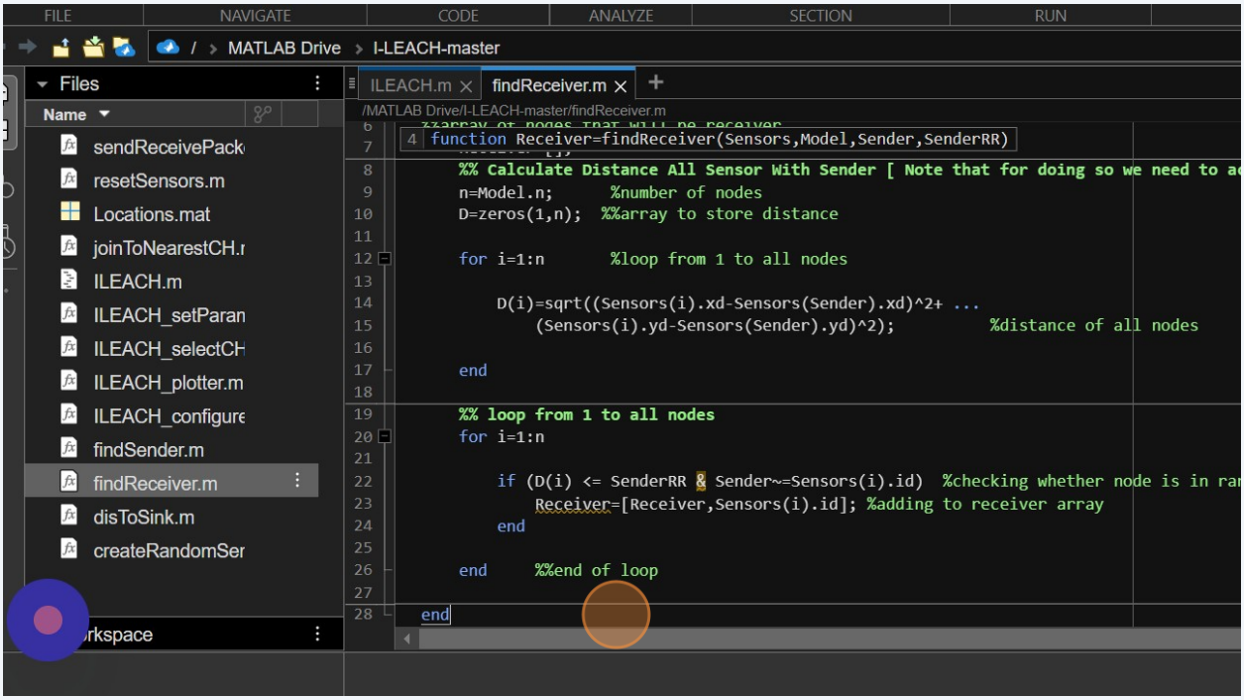
65 • for i = 1:n: Loop through all nodes to calculate distances.

• if (D(i) <= SenderRR & Sender ~= Sensors(i).id): Check if the distance between the sender and a node is within the communication range (SenderRR) and the node is not the sender itself.

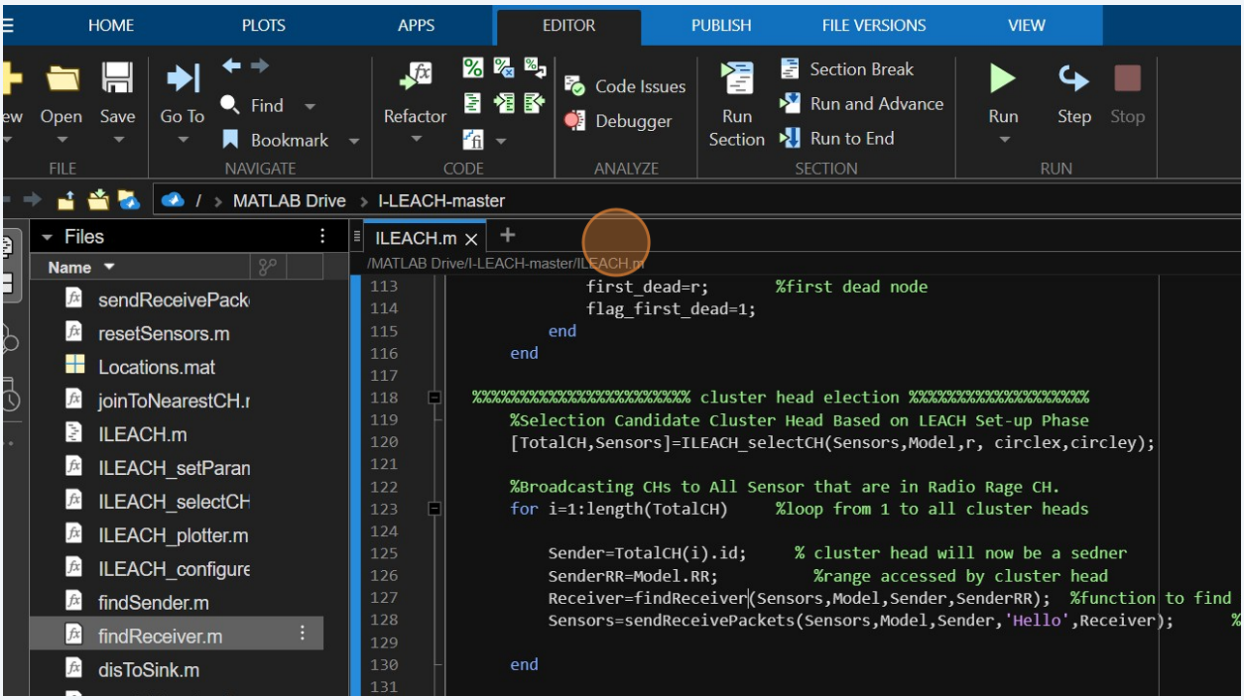


66 • Receiver = [Receiver, Sensors(i).id];: If the conditions are met, add the node to the receiver array.

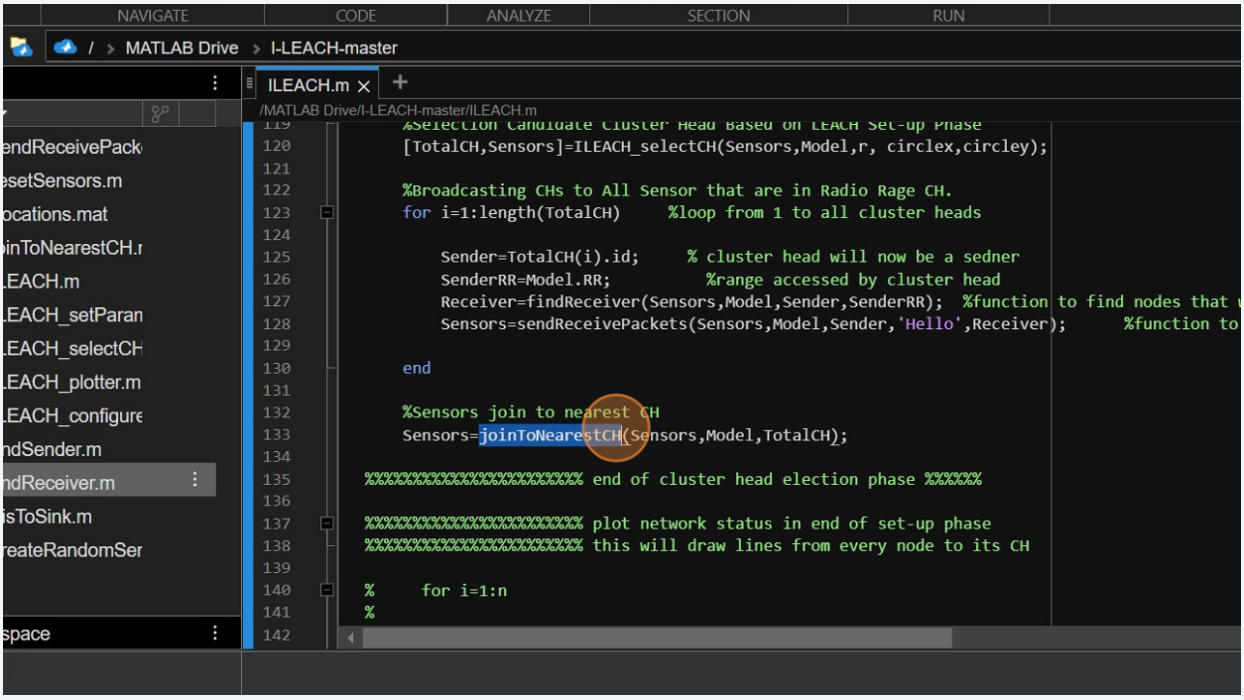
The function essentially finds and returns the set of nodes that are potential receivers within the communication range of the specified sender



67 Now lets move back to main FIle ILEACH



68 Now lets dicuss the next function used here for the sensors join to nearest Cluster Head



69 Here's a step-by-step explanation:

• n=Model.n: Assign the total number of nodes in the network to the variable n.

• m=length(TotalCH): Calculate the total number of clusters by getting the length of the TotalCH array, which contains information about cluster heads.

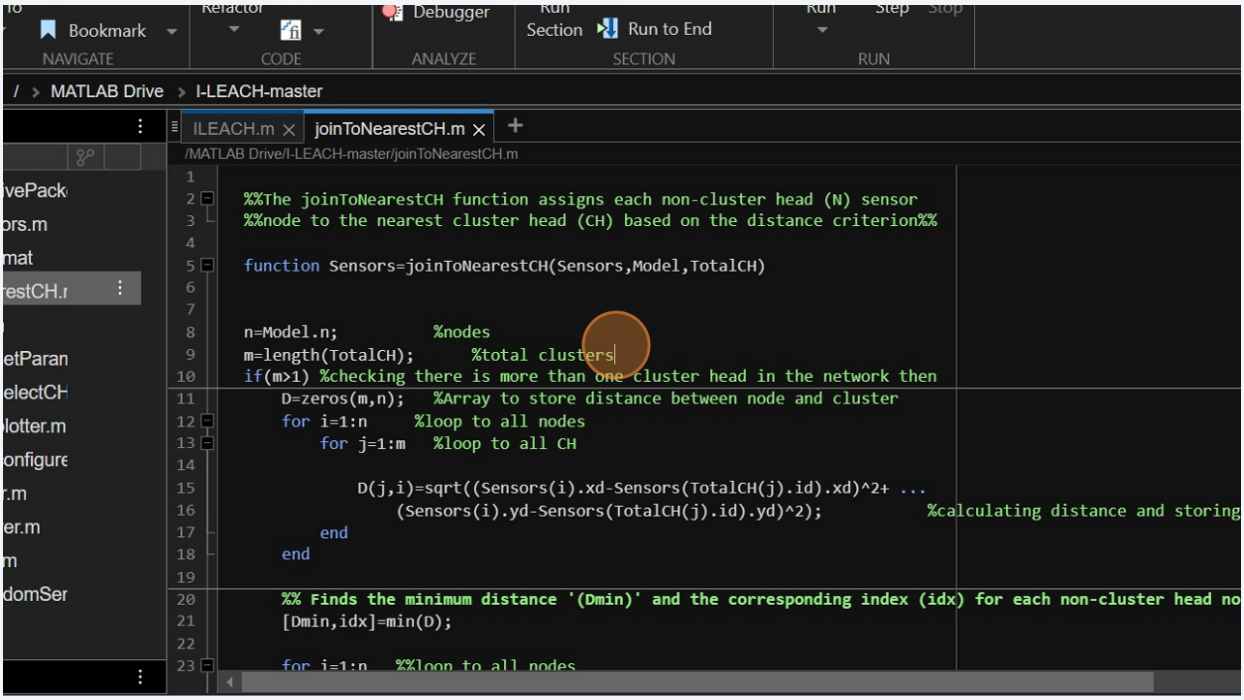
• if(m>1): Check if there is more than one cluster head (if there is at least one cluster).

• D=zeros(m,n): Initialize a matrix D with dimensions m (number of clusters) by n (number of nodes) to store the distances.

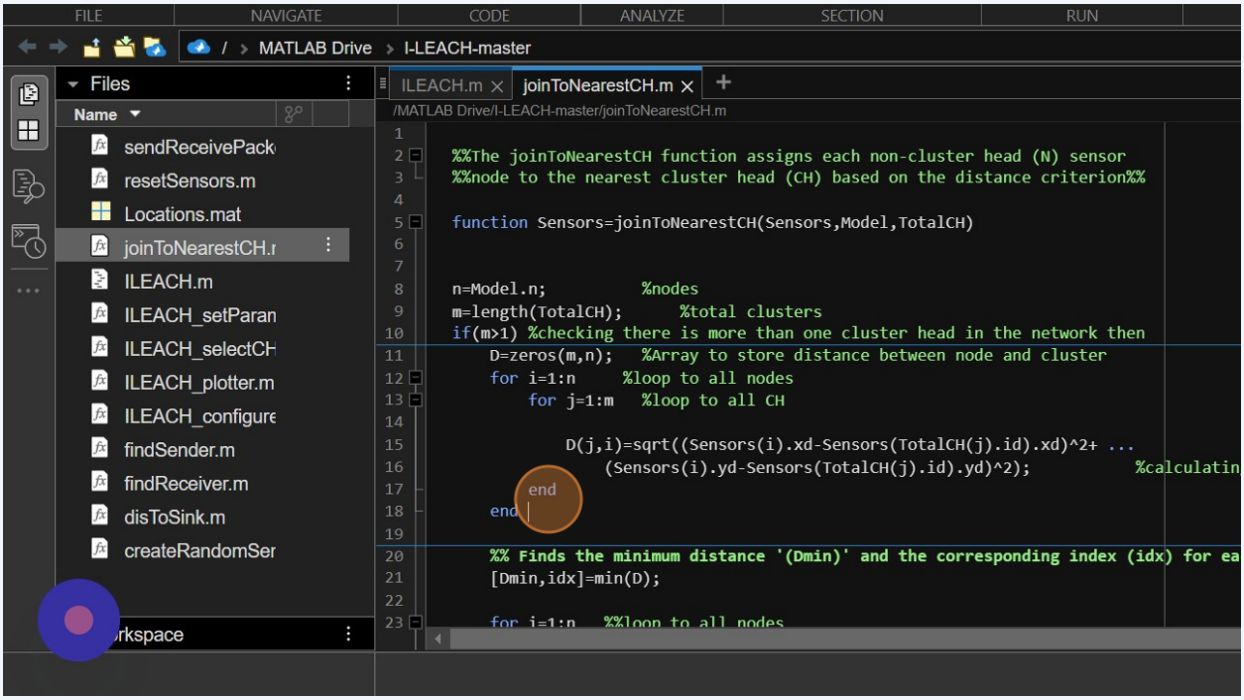
• The nested loops (for i=1:n and for j=1:m) iterate over all nodes and cluster heads, respectively.

• Inside the nested loops, it calculates the Euclidean distance between each node (Sensors(i)) and each cluster head (Sensors(TotalCH(j).id)), then stores the result in the matrix D.

The resulting matrix D contains distances between each non-cluster head node and each cluster head. Each row corresponds to a cluster head, and each column corresponds to a non-cluster head node.



70 Continued



71

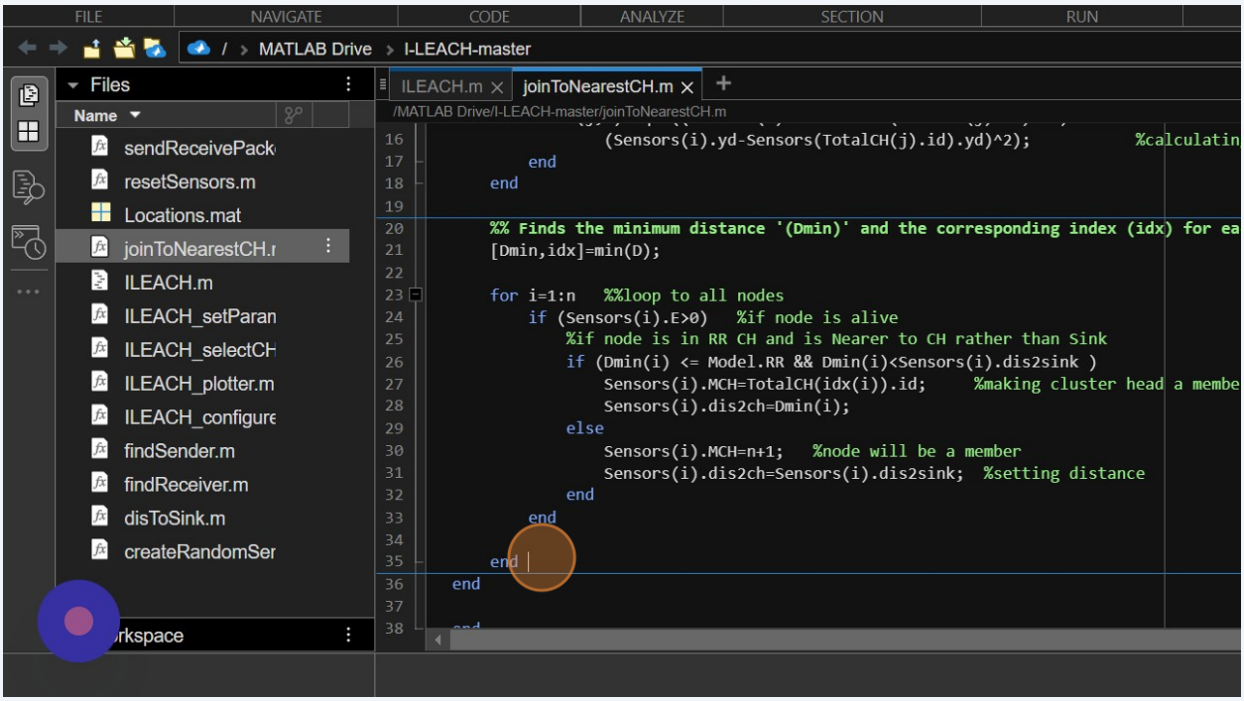
• [Dmin, idx] = min(D): This line finds the minimum distance (Dmin) and the corresponding index (idx) for each column (non-cluster head node) in the distance matrix D. Dmin is an array containing the minimum distance for each node, and idx is an array containing the index of the cluster head associated with the minimum distance.

• The subsequent loop (for i = 1:n) iterates through all nodes.

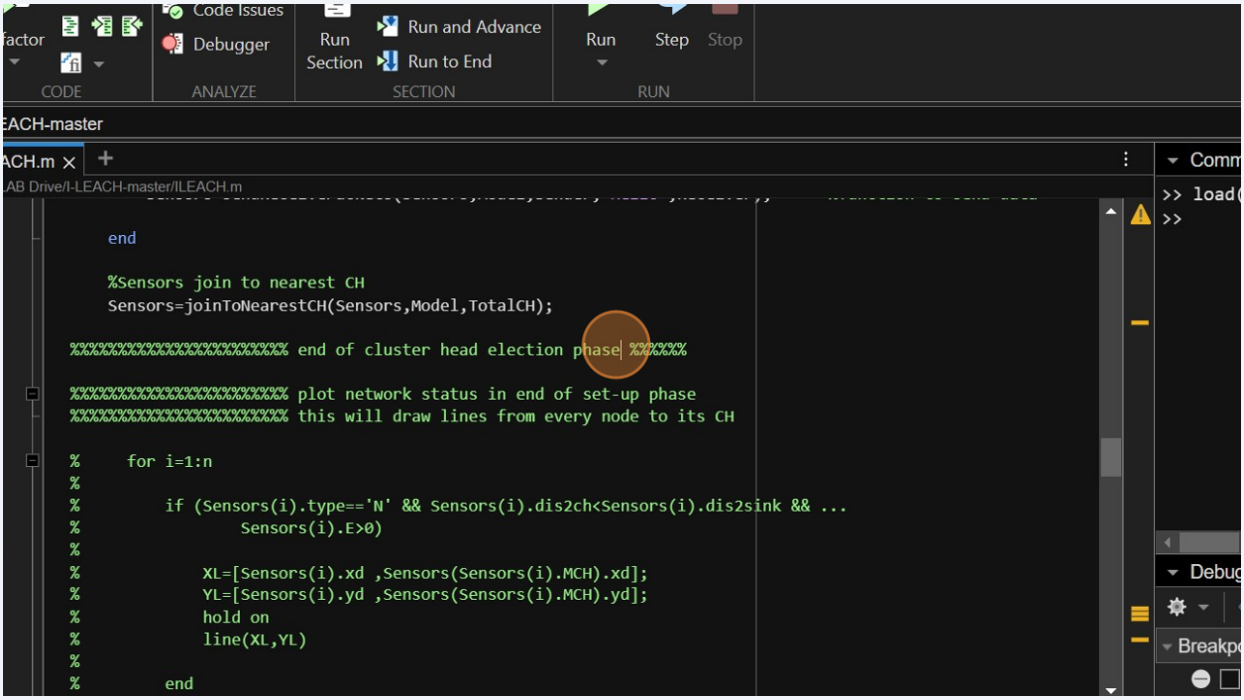
• Inside the loop, it checks if the node is alive (Sensors(i).E > 0).

• If the node is in the radio range of a cluster head and is closer to that cluster head than the sink, it assigns the cluster head as the member (Sensors(i).MCH) and updates the distance (Sensors(i).dis2ch).

• Otherwise, it assigns the sink as the member and sets the distance to the sink.



72 Now moving back to main file ILEACH and now Here cluster head election phase ends



73

• The loop iterates through all nodes.

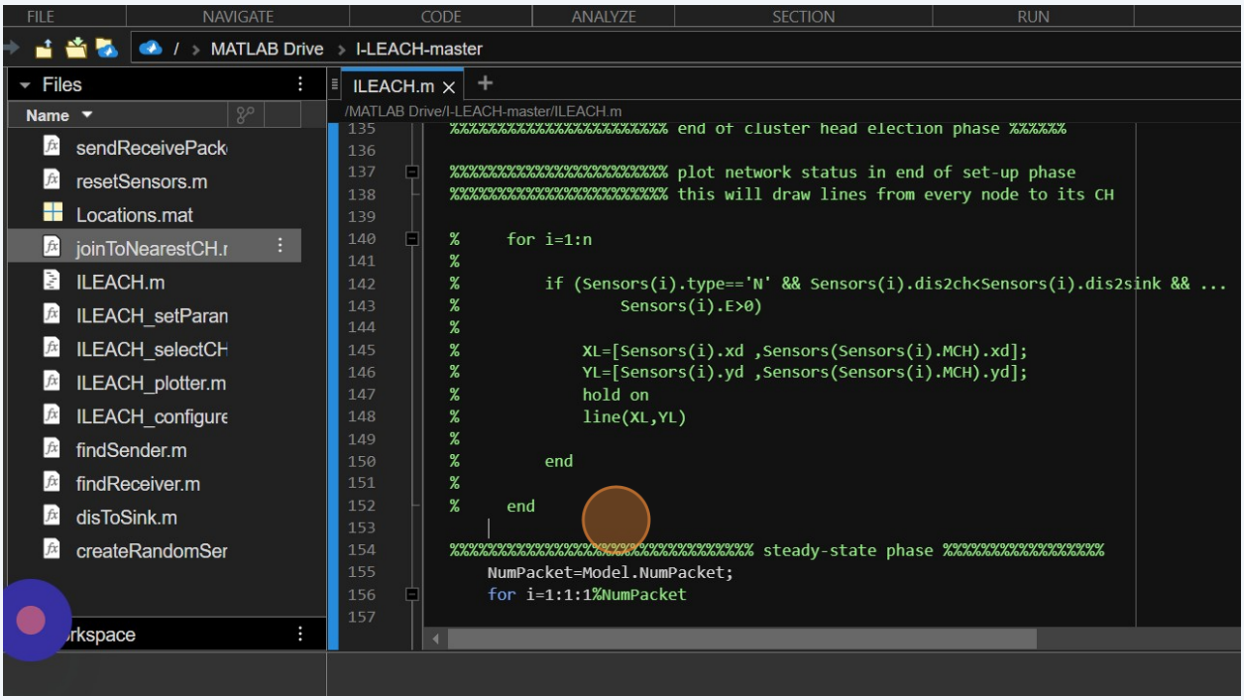
• It checks if the node is a normal node (Sensors(i).type == 'N'), closer to its cluster head (Sensors(i).dis2ch < Sensors(i).dis2sink), and still has energy (Sensors(i).E > 0).

• If the conditions are met, it defines the X and Y coordinates (XL and YL) for the line connecting the node to its cluster head.

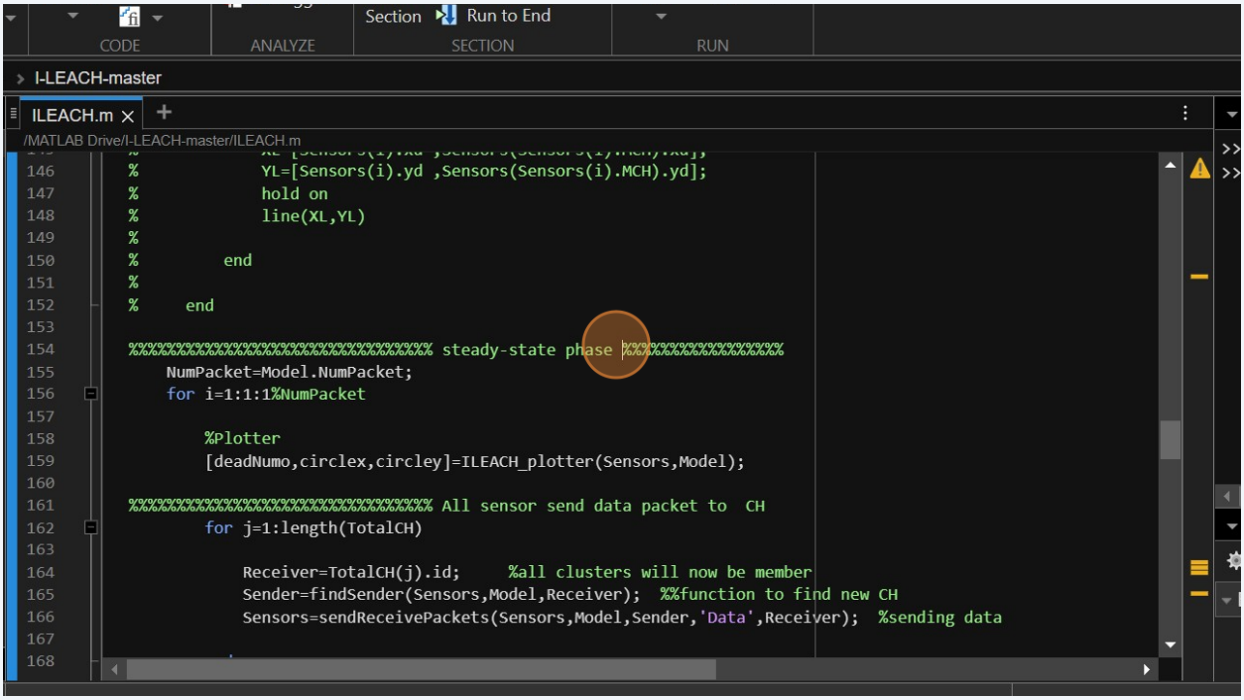
• hold on is used to ensure that subsequent plots are overlaid on the current figure.

• Finally, line(XL, YL) draws a line from the node to its cluster head.

This visualization helps illustrate the connectivity between normal nodes and their respective cluster heads in the network.



74 Now here steady state phase is implemented Lets have an explanation of how it works

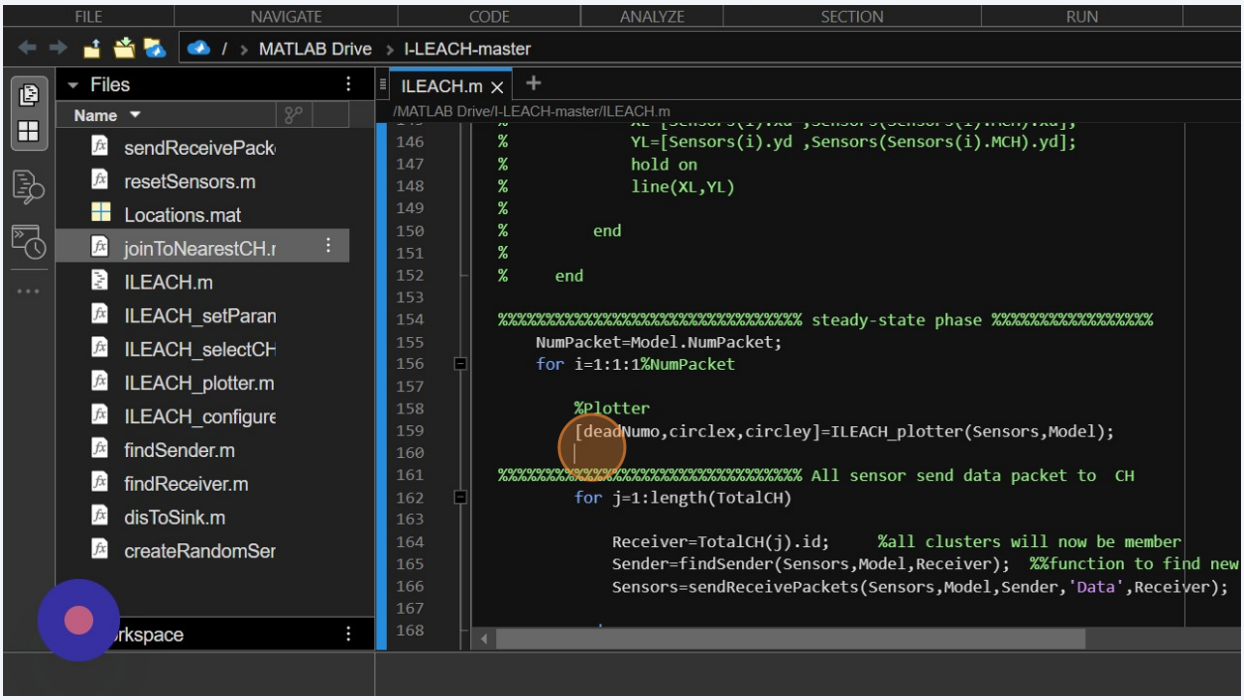


75 Here we're starting a loop to simulate packet transmissions (presumably data packets).

NumPacket = Model.NumPacket; retrieves the number of packets (NumPacket) from the simulation model (Model).

• The loop for i = 1:1:1 seems to iterate only once, indicating that this block of code is specifically for the first packet.

• Inside the loop, the ILEACH_plotter function is called to visualize the network status. The function returns the number of dead nodes (deadNumo) and circular regions (circlex and circley).



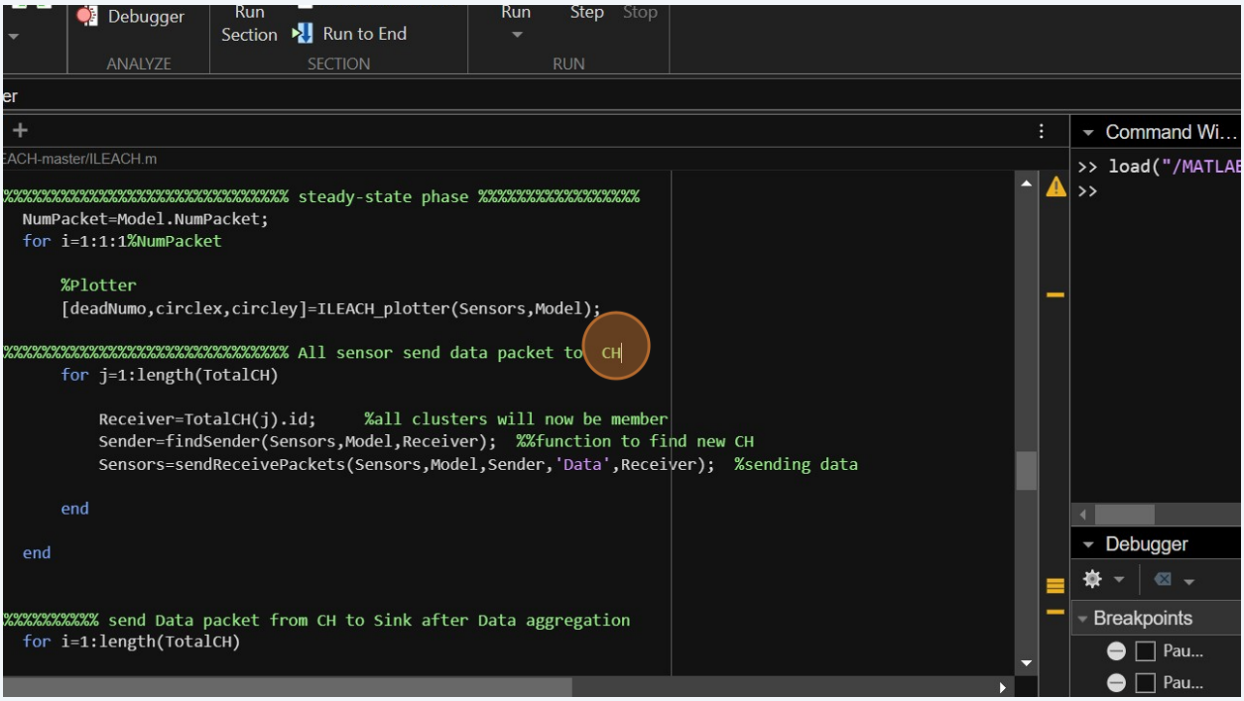
76

• The loop iterates through each cluster head (TotalCH(j).id).

• Receiver is set to the ID of the cluster head, as each cluster head is intended to receive data packets.

• Sender is determined using the findSender function. This function likely identifies the normal nodes in the vicinity of the cluster head that will send data packets.

• The sendReceivePackets function is then called to simulate the process of sending data packets from normal nodes to their respective cluster heads.



77

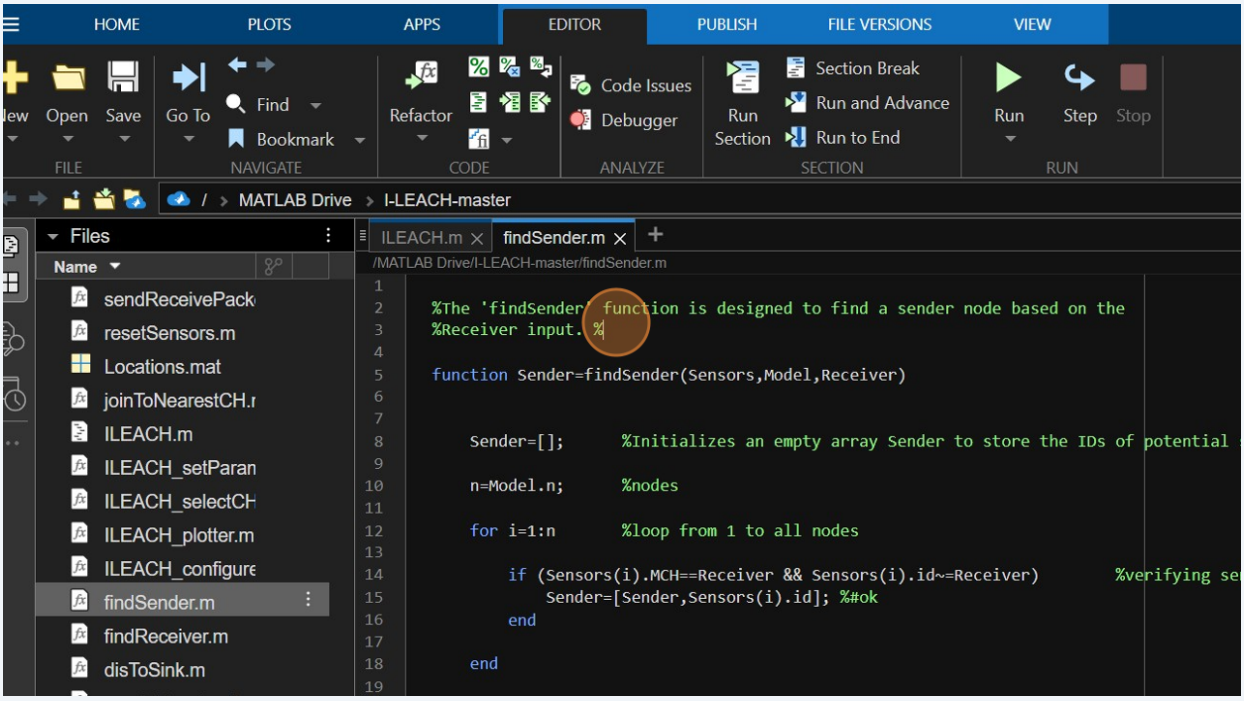
• The function loops through all nodes (Sensors) in the network.

• For each node, it checks if the node is a potential sender to the specified Receiver based on the conditions Sensors(i).MCH == Receiver and Sensors(i).id ~= Receiver.

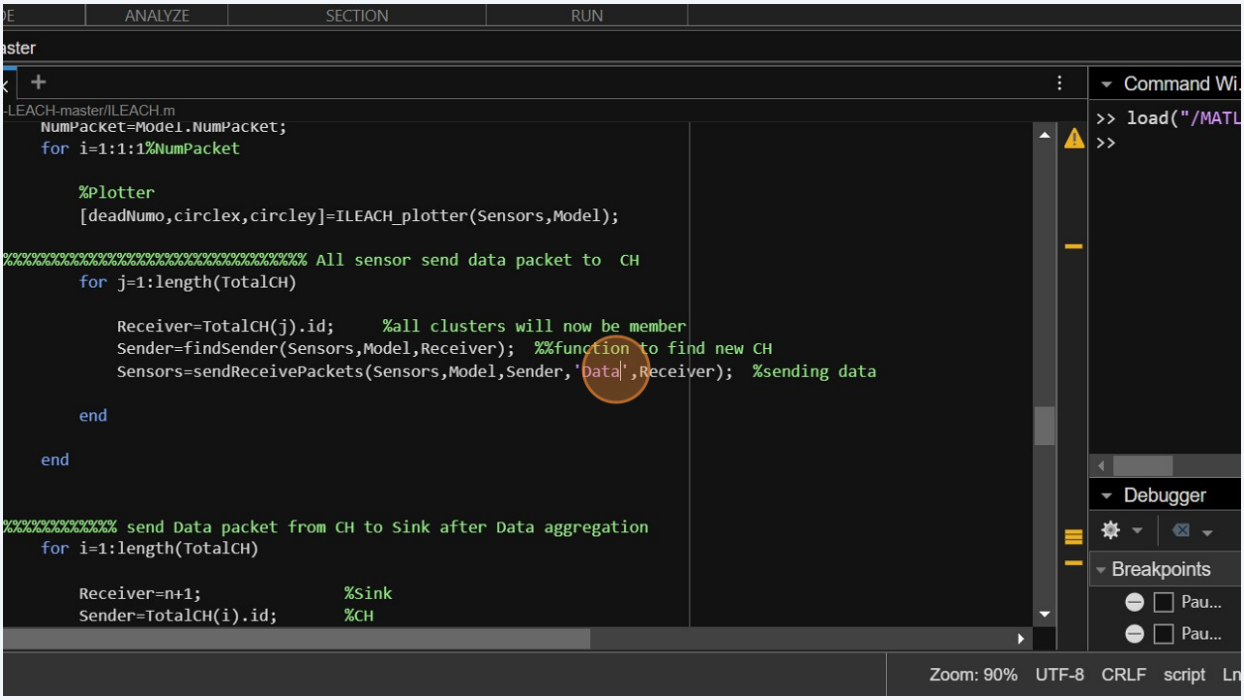
• If a node satisfies the conditions, its ID is added to the Sender array.

• The final Sender array contains the IDs of potential sender nodes.

This function is used to identify nodes that should send data packets to a specific receiver, such as a cluster head.



78 Now here Data is send instead of Hello Messages



79

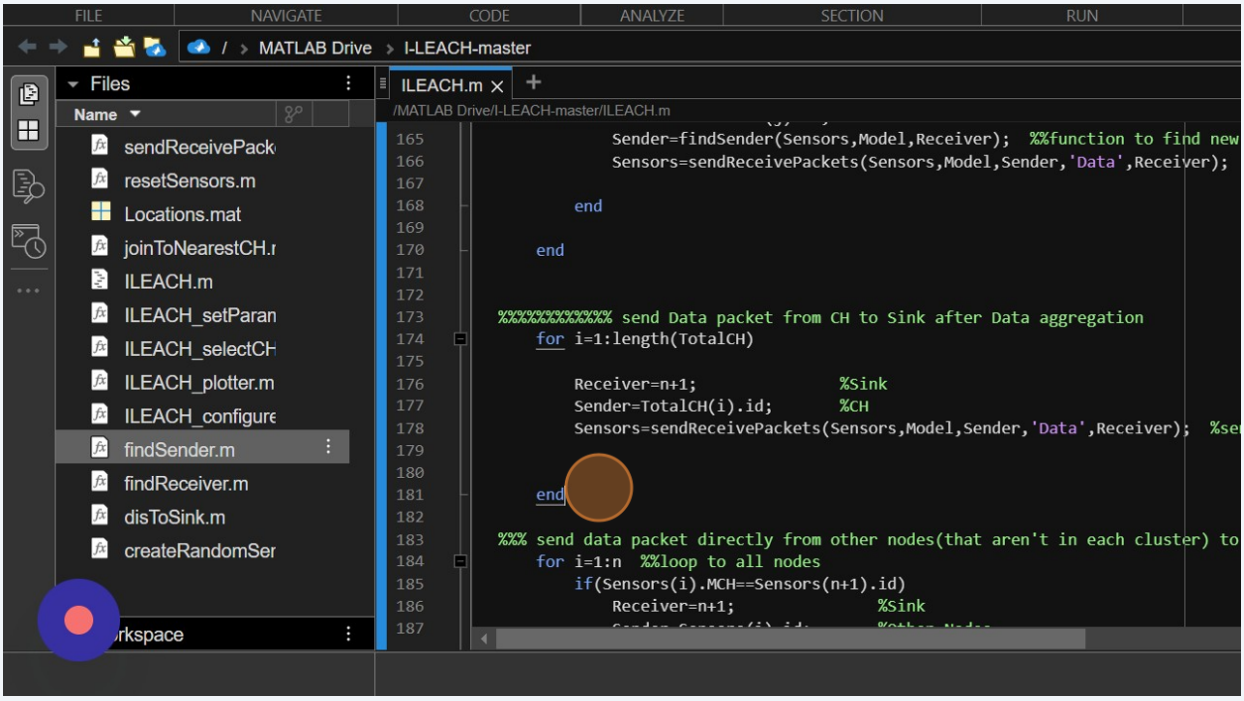
• The loop iterates over each Cluster Head (TotalCH is an array containing information about all cluster heads).

• For each Cluster Head, it sets the Receiver variable to the Sink (n + 1).

• It sets the Sender variable to the ID of the current Cluster Head (TotalCH(i).id).

• Calls the sendReceivePackets function to simulate the transmission of a data packet from the CH to the Sink. The function updates the energy levels of the involved nodes.

This part of the code represents the data aggregation phase where each Cluster Head forwards aggregated data to the Sink.



80

• The loop iterates over all nodes (n represents the total number of nodes).

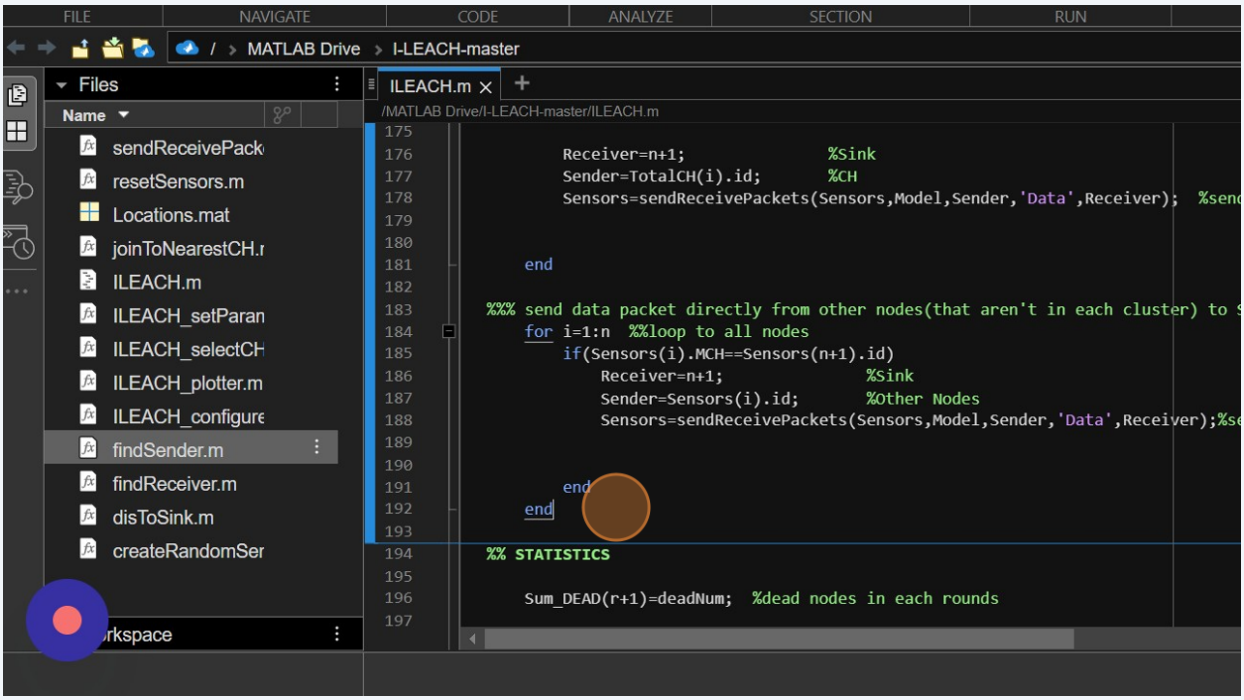
• It checks whether the current node (Sensors(i)) is a member of any cluster by verifying if its MCH (Member Cluster Head) ID is equal to the Sink ID (Sensors(n + 1).id).

• If the node is a member of a cluster, it sets the Receiver variable to the Sink (n + 1).

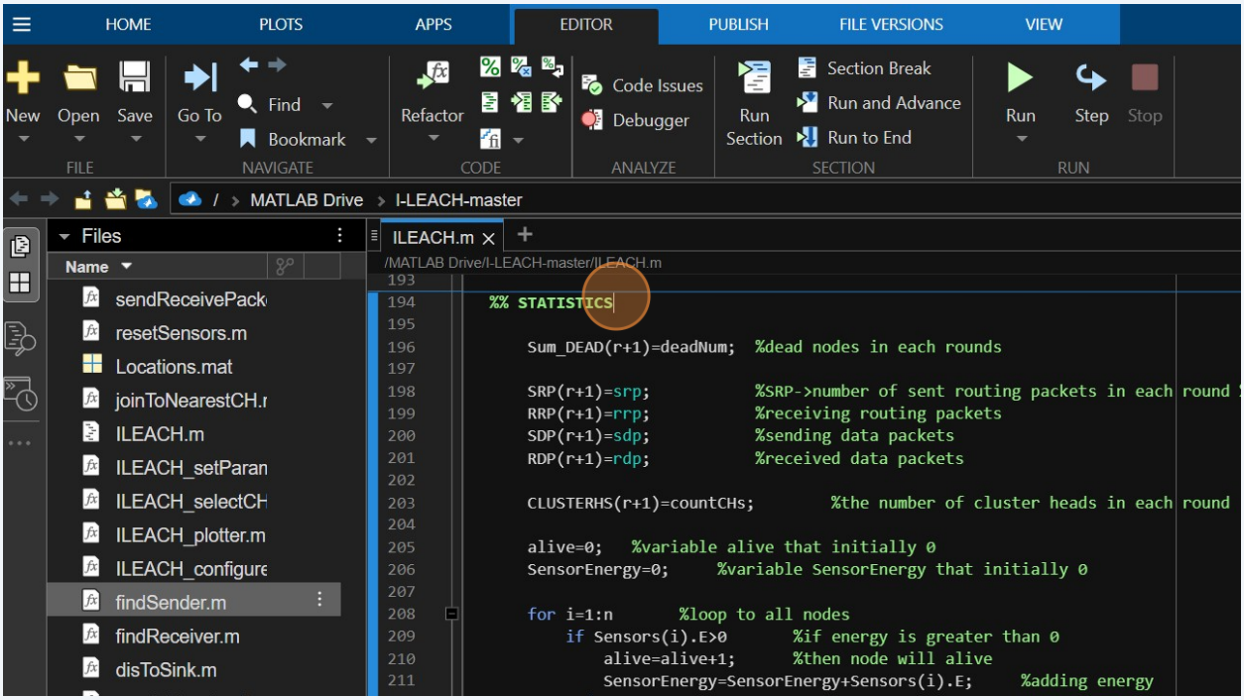
• It sets the Sender variable to the ID of the current node (Sensors(i).id).

• Calls the sendReceivePackets function to simulate the transmission of a data packet from the non-cluster head node to the Sink. The function updates the energy levels of the involved nodes.

This part of the code represents the direct transmission of data packets from nodes that are not cluster heads to the Sink.



81 Now comes the statistics



82

• Sum_DEAD(r + 1): Records the number of dead nodes in each round.

• SRP(r + 1): Records the number of sent routing packets in each round.

• RRP(r + 1): Records the number of received routing packets in each round.

• SDP(r + 1): Records the number of sent data packets in each round.

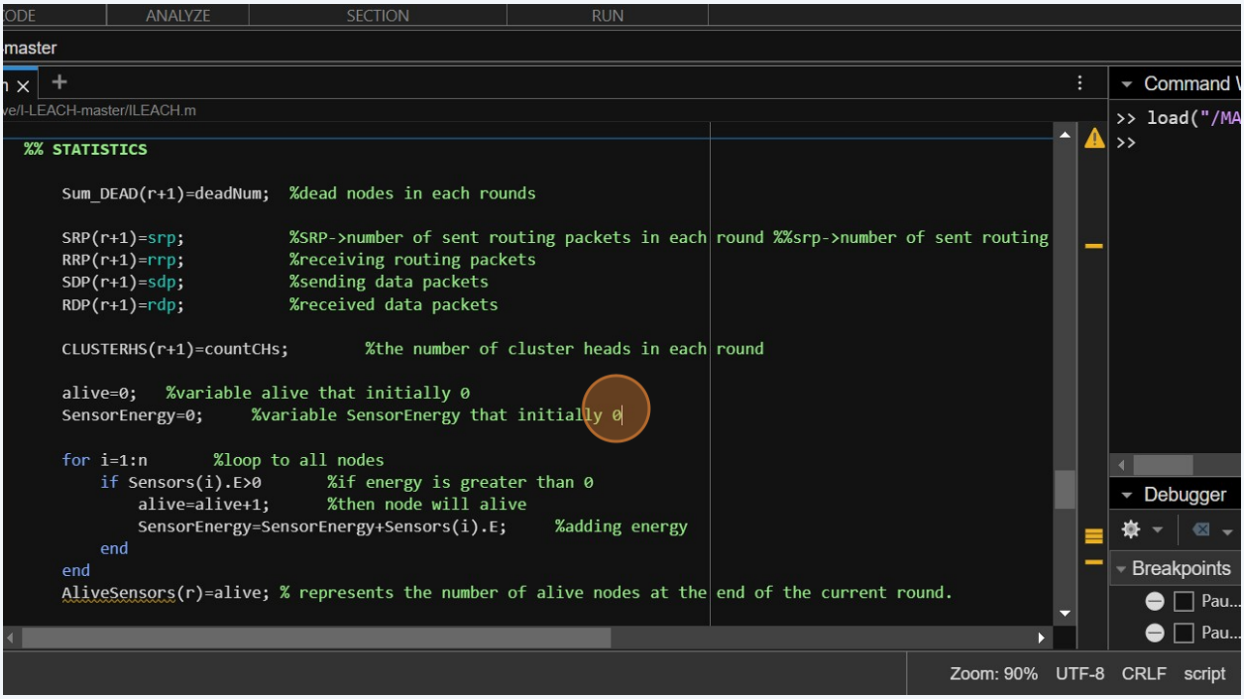
• RDP(r + 1): Records the number of received data packets in each round.

• CLUSTERHS(r + 1): Records the number of cluster heads in each round.

• alive: A variable initialized to 0. It appears to be used to track the number of alive nodes, but the specific implementation details are not provided in this segment.

• SensorEnergy: A variable initialized to 0. It seems to be related to tracking the total energy of the sensor nodes, but the specific details are not provided in this segment.

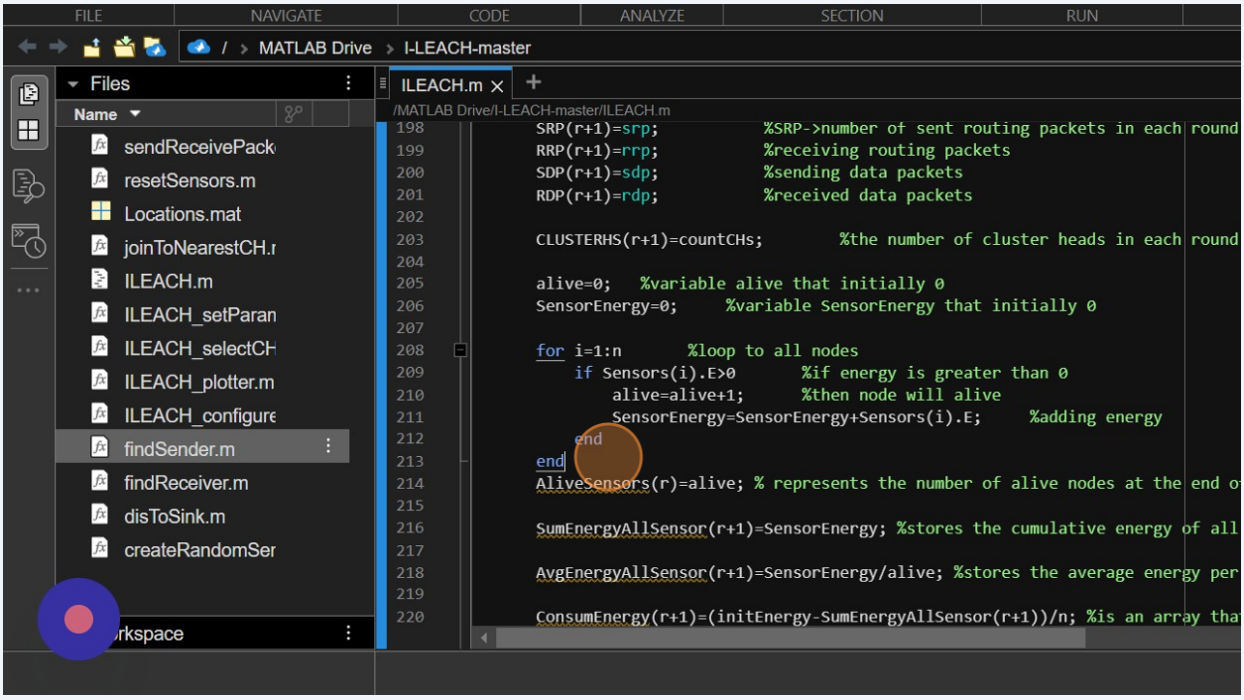
These variables are likely used for monitoring and analyzing the performance of the simulation over multiple rounds



83

• alive: This counter is incremented for each node with energy greater than 0, representing the number of alive nodes.

• SensorEnergy: This variable accumulates the total energy of all nodes with energy greater than 0. It provides a measure of the total energy in the network.



84

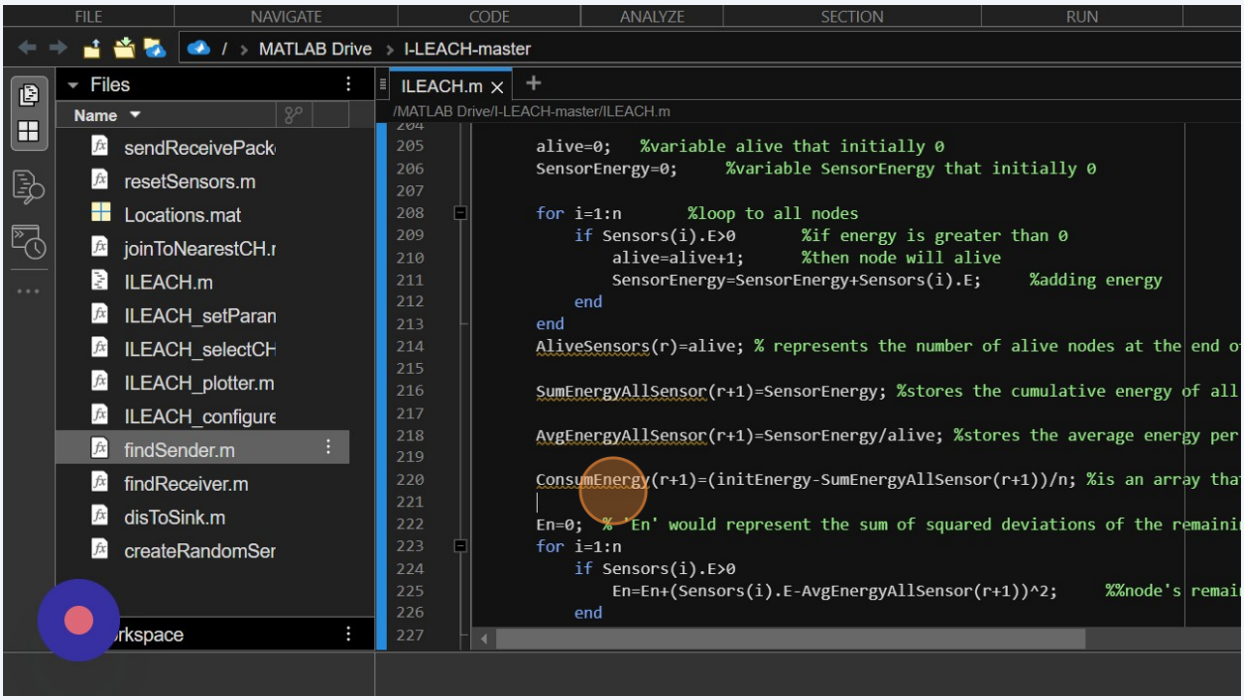
• AliveSensors(r): This array stores the number of alive nodes at the end of each round (denoted by r). The variable alive from the previous loop is used here.

• SumEnergyAllSensor(r+1): This array stores the cumulative energy of all nodes up to the current round (r+1). The SensorEnergy variable, which accumulates the energy of alive nodes, is used here.

• AvgEnergyAllSensor(r+1): This array stores the average energy per node up to the current round (r+1). It is calculated by dividing the cumulative energy (SensorEnergy) by the number of alive nodes (alive).

• ConsumEnergy(r+1): This array tracks the average energy consumption per node in each round. It is calculated as the difference between the initial energy (initEnergy) and the cumulative energy at the end of the round (SumEnergyAllSensor(r+1)), divided by the number of nodes (n).

• En: This variable is initialized to 0. However, it seems to be unused in the provided code segment.



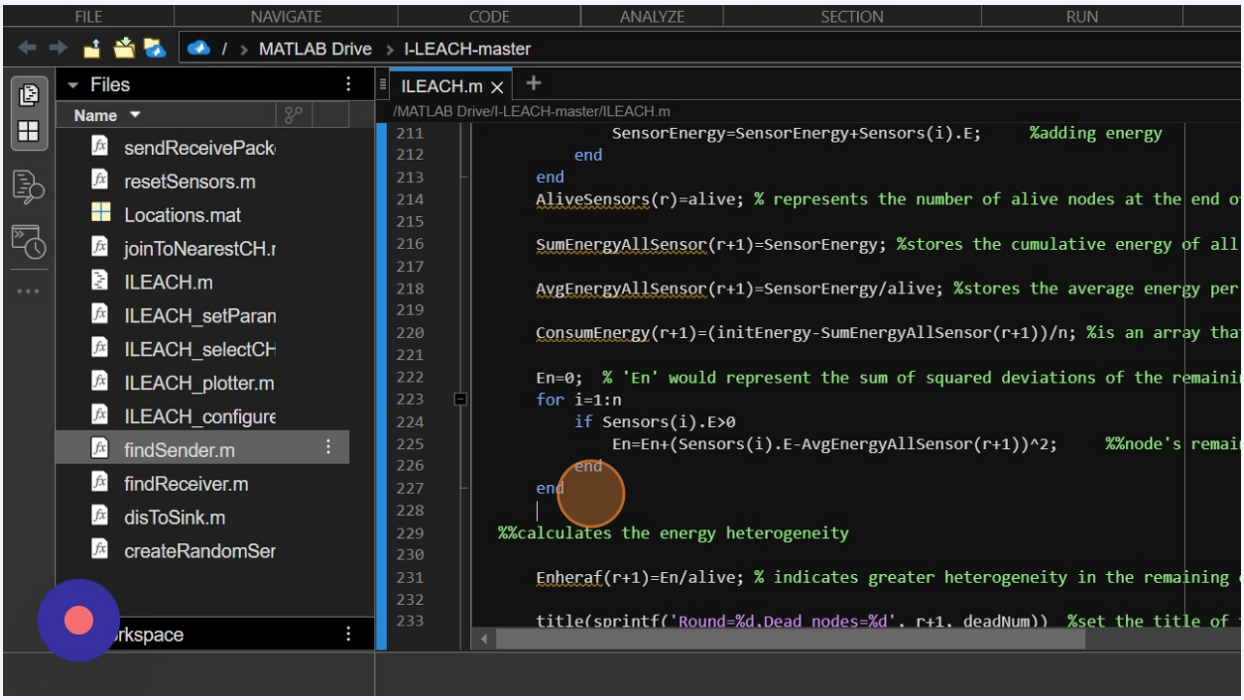
85

• This loop iterates through all nodes (1 to n).

• The if Sensors(i).E > 0 condition checks if the energy of the current node (i) is greater than 0, indicating that the node is alive.

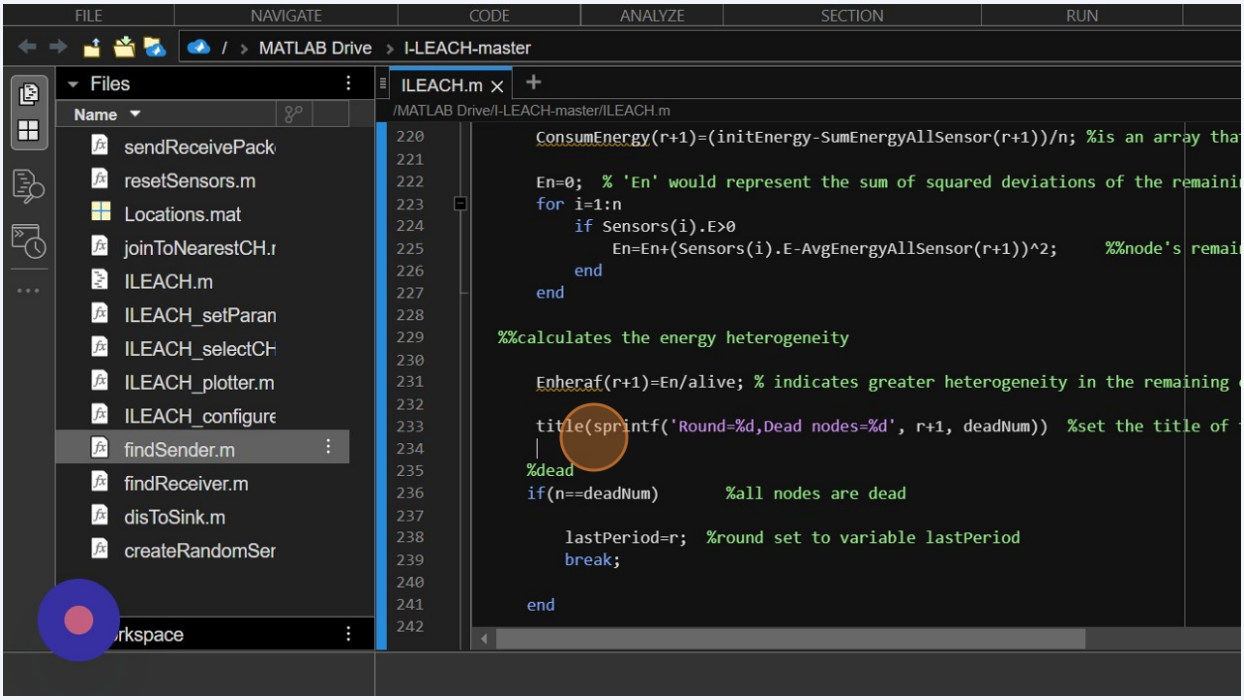
• The line En = En + (Sensors(i).E - AvgEnergyAllSensor(r+1))^2 accumulates the squared difference between the remaining energy of the current node (Sensors(i).E) and the average energy per node (AvgEnergyAllSensor(r+1)). This is done for all alive nodes in the network.

This operation is part of a calculation related to the variance of the remaining energy among nodes.

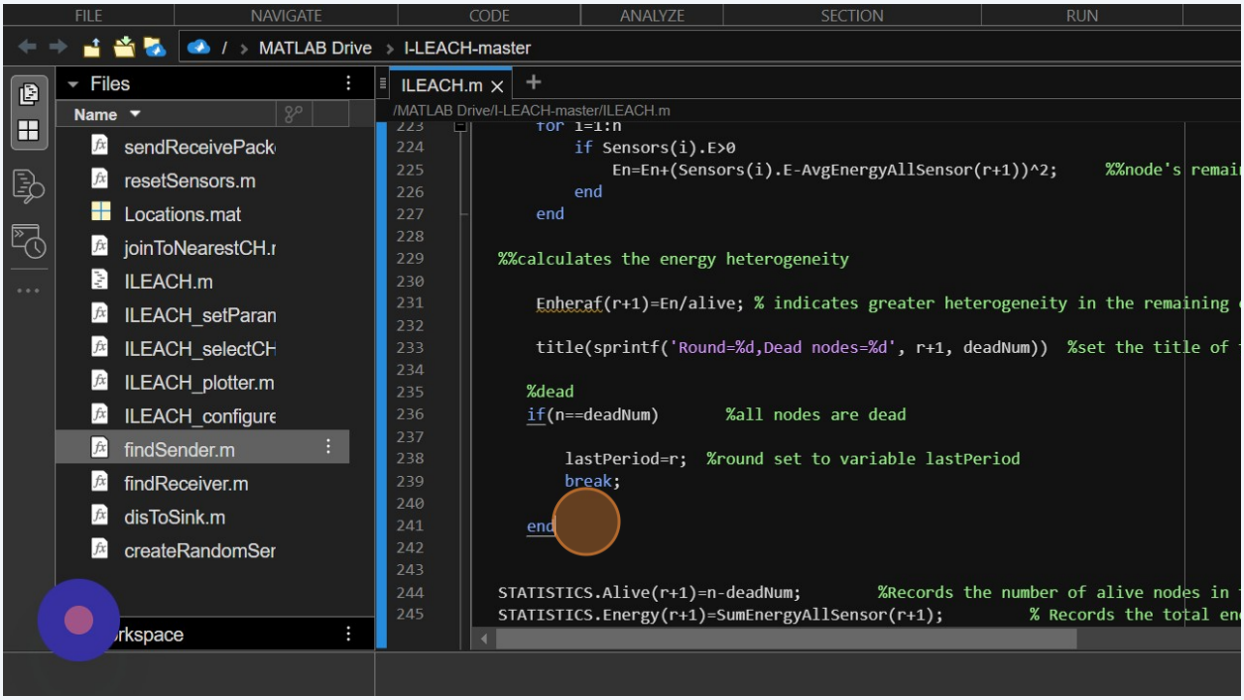


86 The line Enheraf(r+1) = En/alive; calculates the energy heterogeneity at the end of the current round (r+1). Energy heterogeneity is a measure of how non-uniformly energy is distributed among the nodes in the network. It is calculated by dividing the sum of squared differences between each node's remaining energy and the average energy by the number of alive nodes (alive).

The title(sprintf('Round=%d,Dead nodes=%d', r+1, deadNum)) sets the title of the plot. It displays information about the current round number (r+1) and the number of dead nodes (deadNum). The sprintf function is used to format the text that will be displayed in the title.



87 The condition if (n == deadNum) checks whether the number of dead nodes is equal to the total number of nodes (n). If this condition is true, it implies that all nodes in the network are dead, and the simulation is considered completed. The variable lastPeriod is then set to the current round (r), and the break statement is used to exit the loop, ending the simulation.

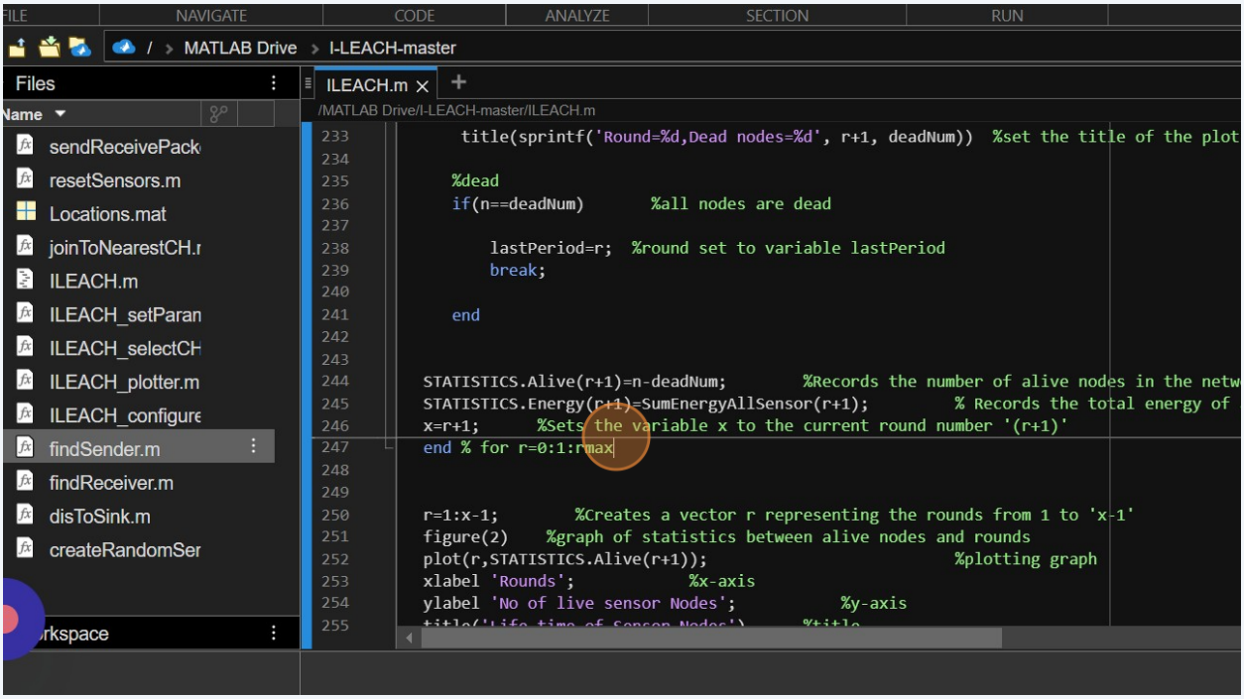


88

• STATISTICS.Alive(r+1) records the number of alive nodes in the network at the end of the current round (r+1). It subtracts the number of dead nodes (deadNum) from the total number of nodes (n).

• STATISTICS.Energy(r+1) records the total energy of all sensors at the end of the current round (r+1). It uses the cumulative energy recorded in SumEnergyAllSensor(r+1).

These statistics provide insights into the network's performance and energy consumption throughout the simulation.



89 The last part of the code generates a plot and saves simulation results to a MAT file. Here's a breakdown:

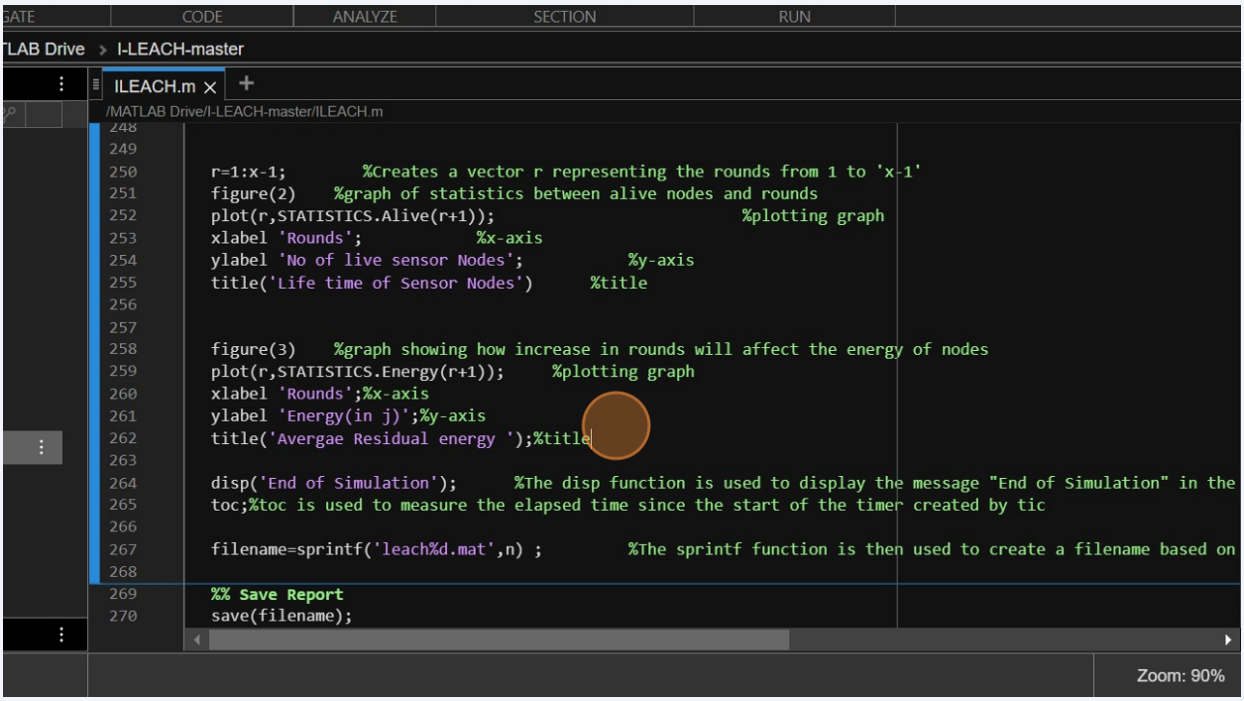
• figure(3): Creates a new figure with a handle of 3. This is necessary if you want to plot multiple graphs in the same script.

• plot(r, STATISTICS.Energy(r+1)): Plots the average residual energy against the number of rounds (r). The STATISTICS.Energy(r+1) array holds the total energy of all sensors at the end of each round. The x-axis is labeled "Rounds," the y-axis is labeled "Energy (in J)," and the title of the plot is set to "Average Residual Energy."

• xlabel 'Rounds': Sets the label for the x-axis.

• ylabel 'Energy(in J)': Sets the label for the y-axis.

• title('Average Residual Energy'): Sets the title of the plot.



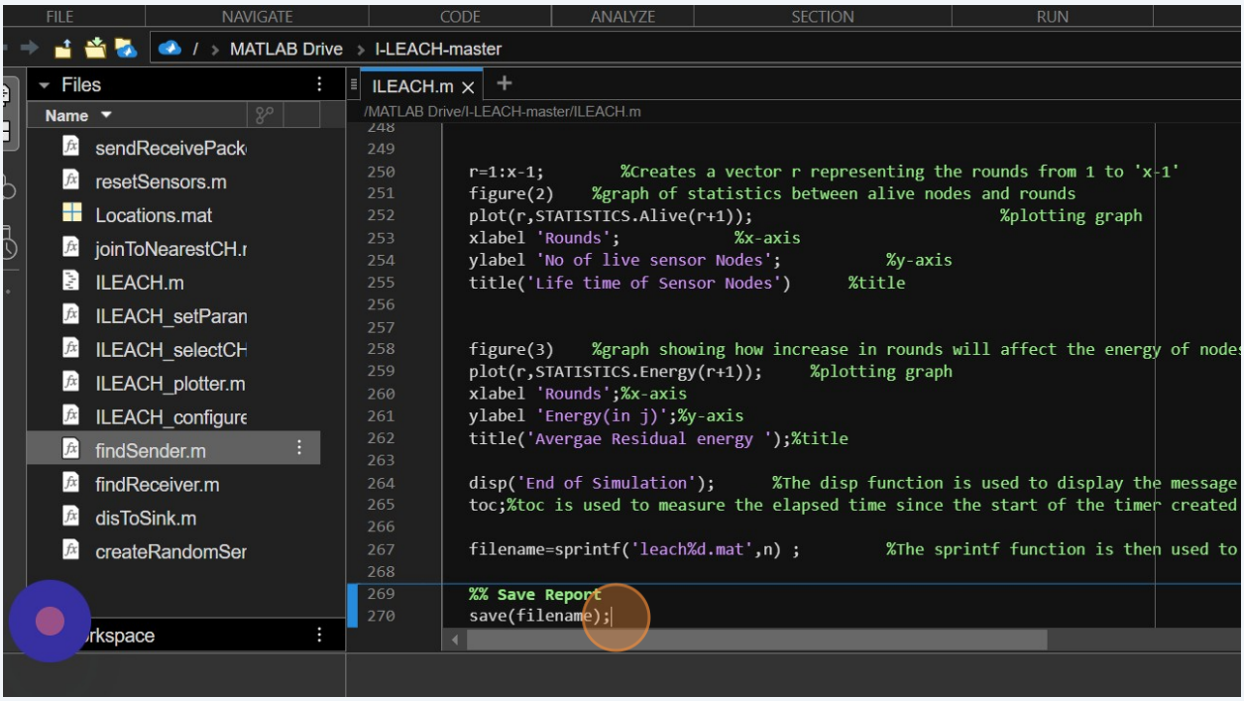
90

• disp('End of Simulation'): Displays the message "End of Simulation" in the command window.

• toc: Outputs the elapsed time since the timer was started by tic.

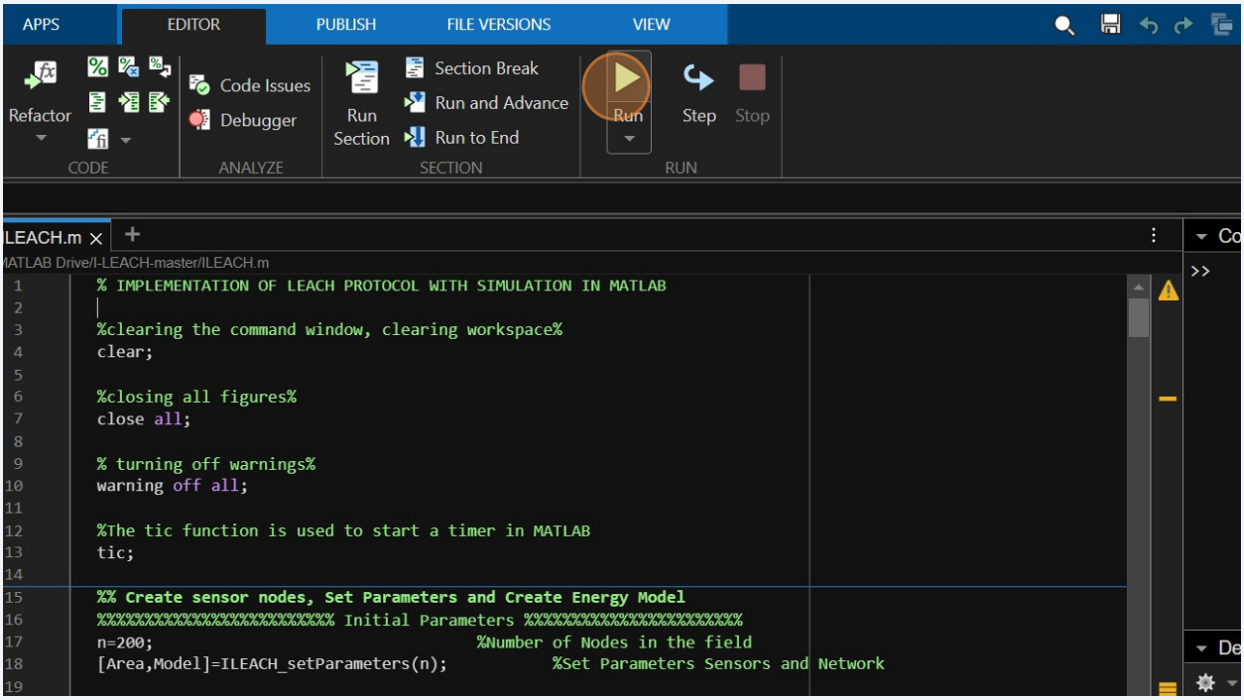
• filename = sprintf('leach%d.mat', n): Generates a filename for saving the simulation results as a MAT file. The sprintf function is used to create a filename based on the number of nodes (n).

• save(filename): Saves the simulation results in a MAT file with the generated filename.

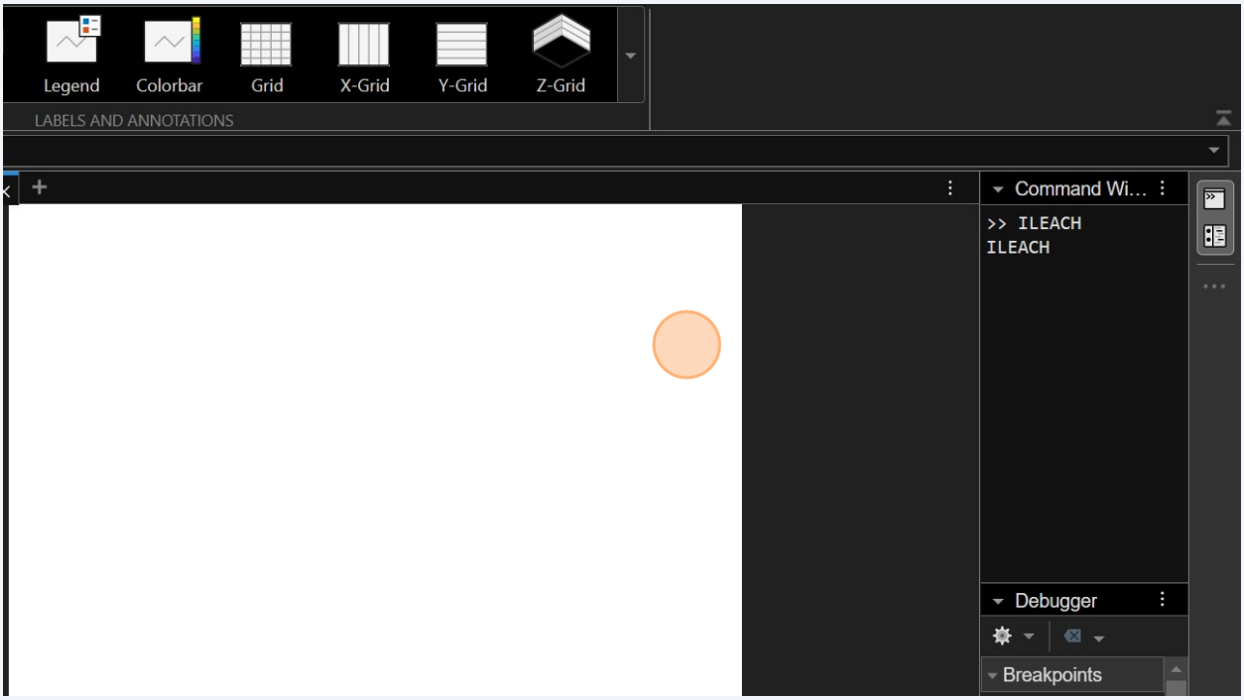


LEACH PROTOCOL OUTPUT IN MATLAB

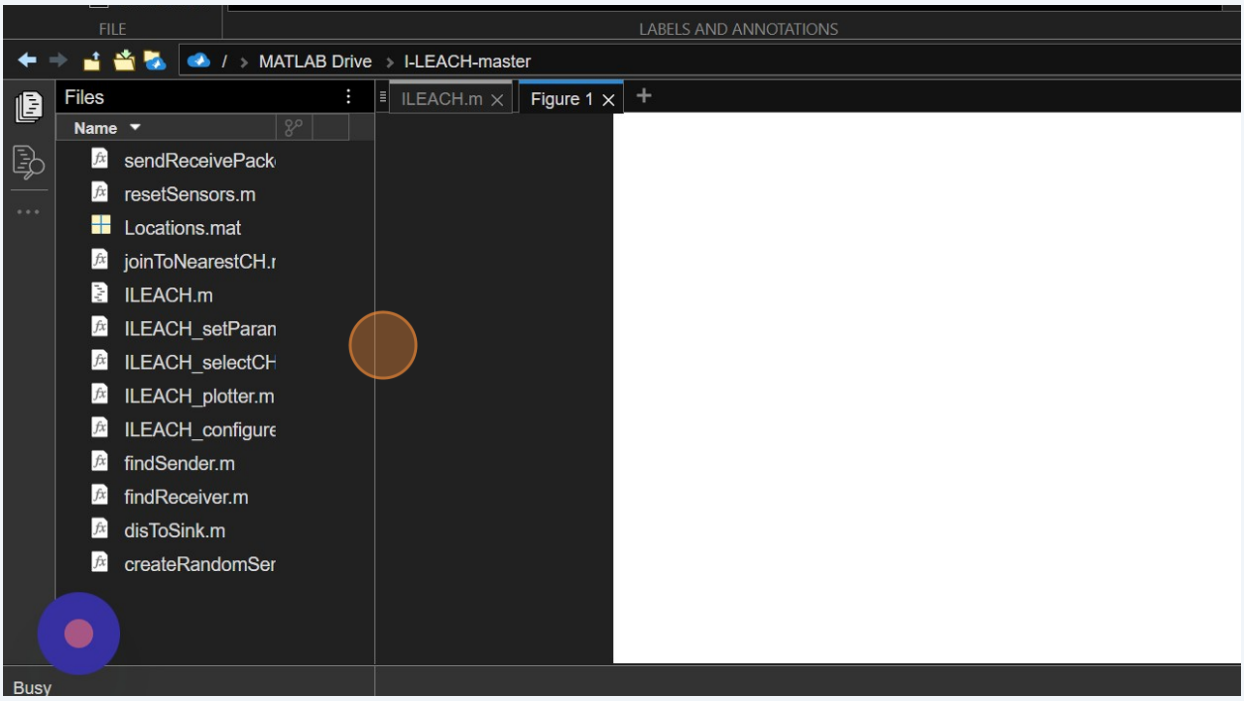
1 Click this icon. to run the simulation



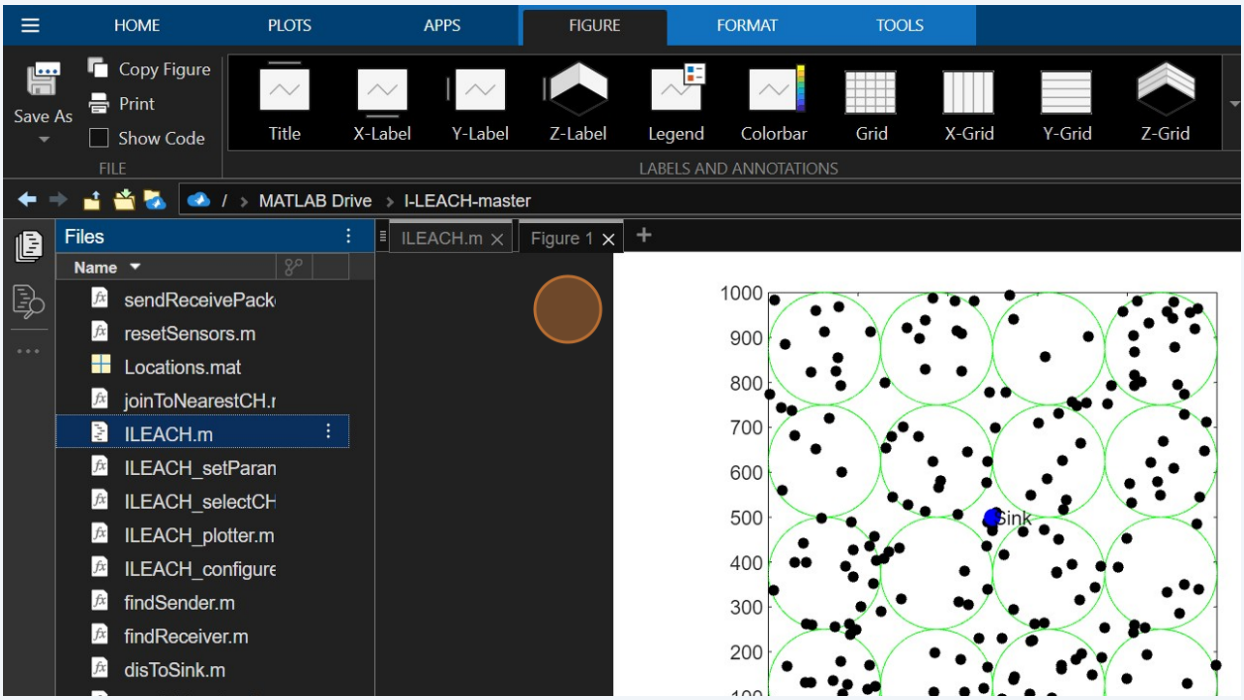
2 The protocol simulation starts It can also be run from command prompt by typing name of main file ILEACH



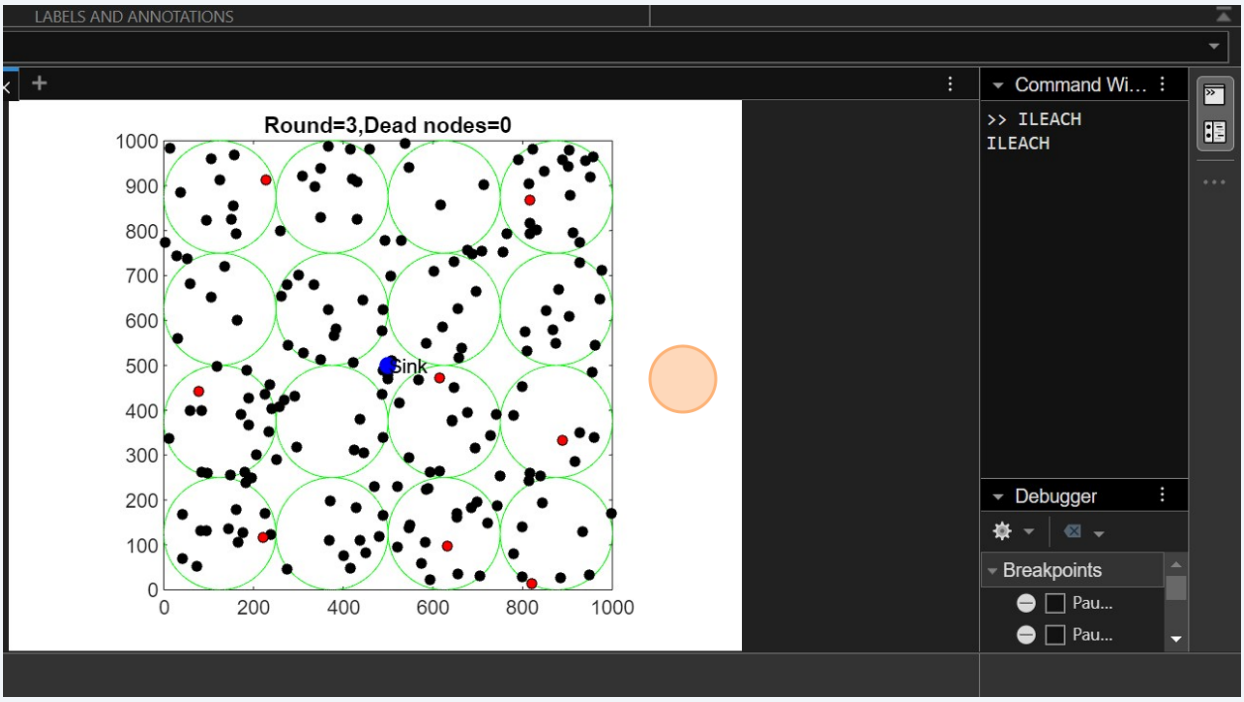
3 A Figure will be opened



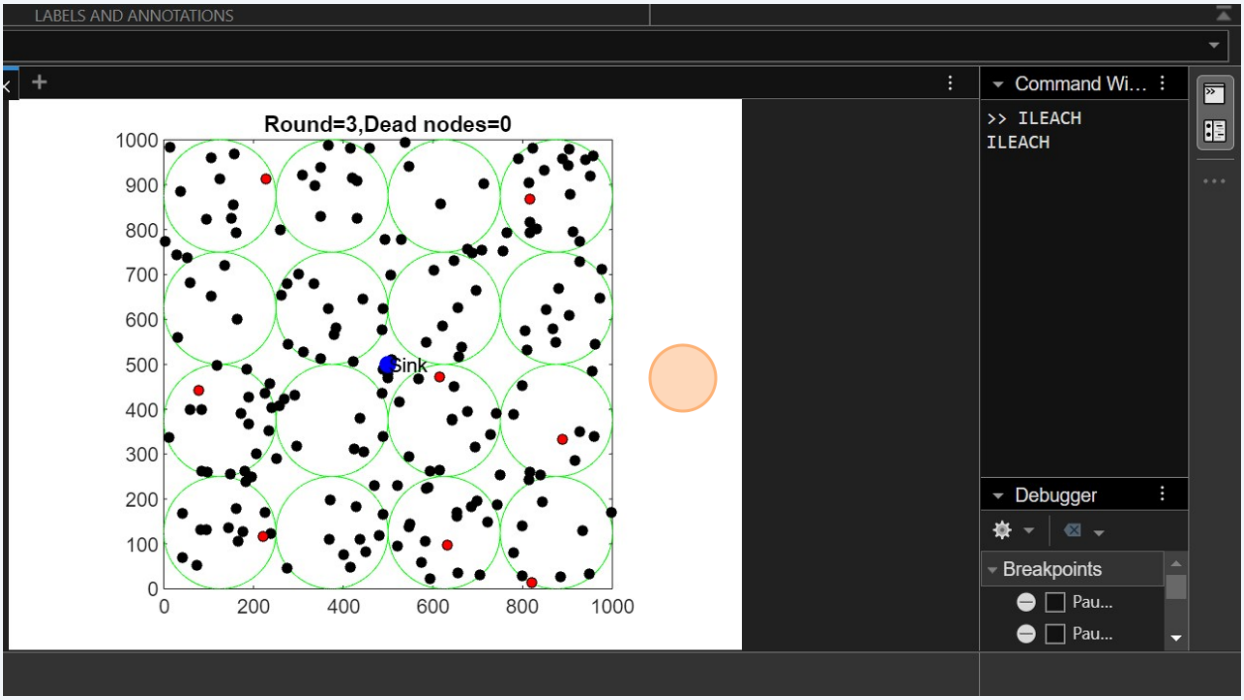
4 And Simulation is started, The green nodes represents the clusters black nodes are member nodes and cluster heads are represented by red color and the sink is displayed with blue color



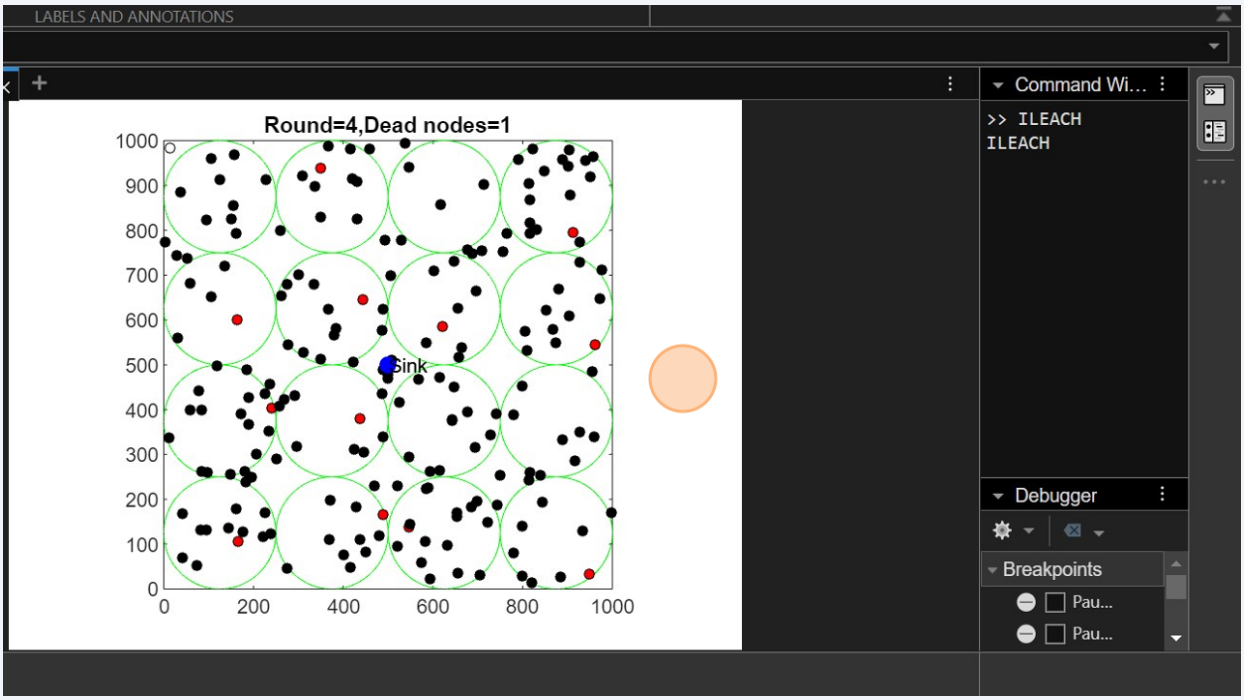
5 As LEACH works in Rounds , so in each round each cluster make its new cluster head to conserve the energy for longer time



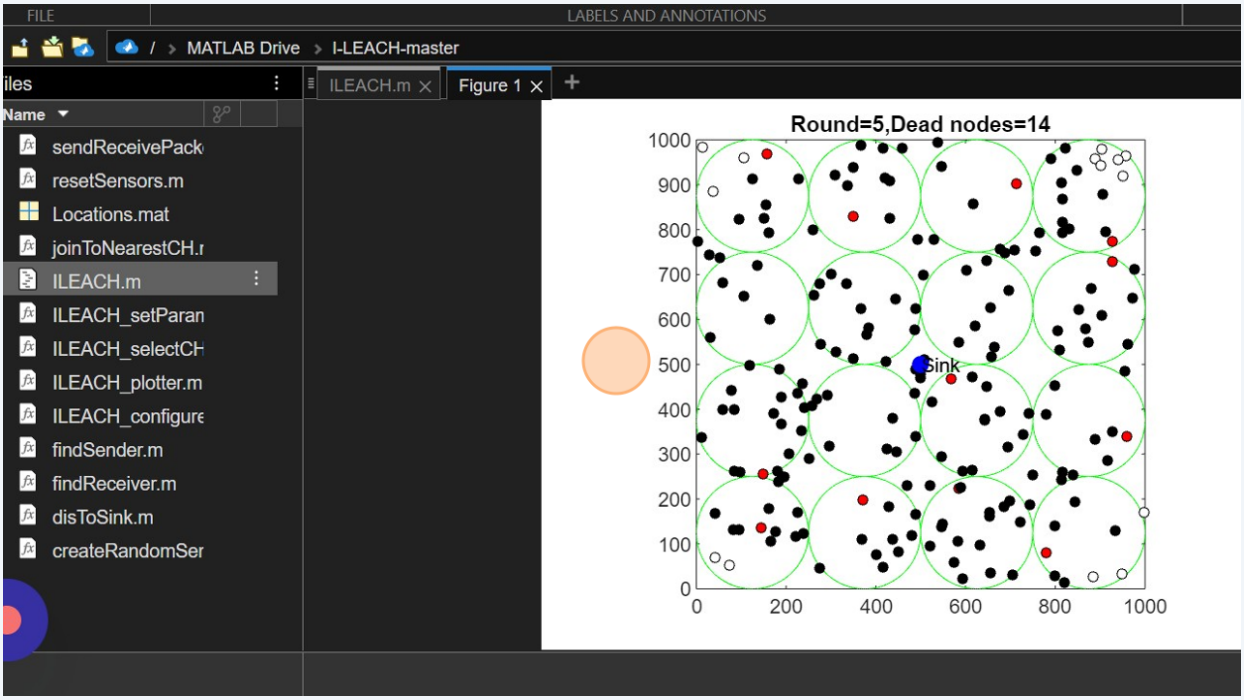
6 Nodes cannot send the data to one another only the members nodes can send data packets to cluster head and then cluster head send the packets to sink node



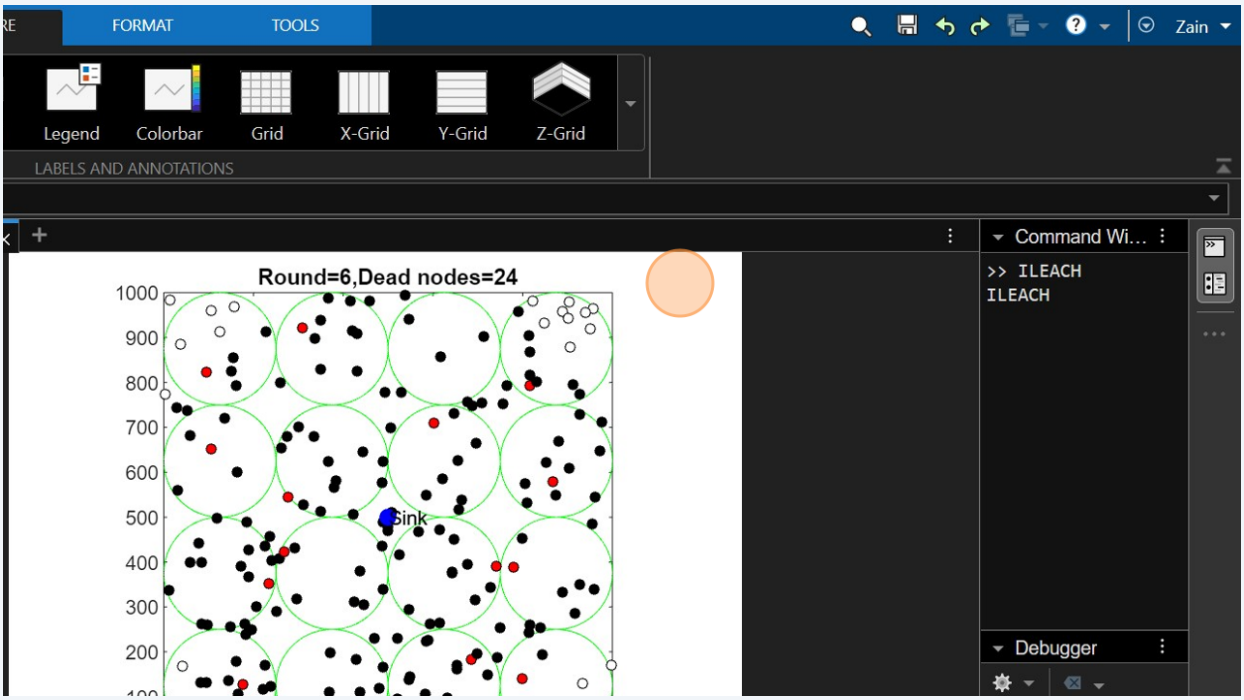
7 Each node has time slot to send data packets to Cluster Head



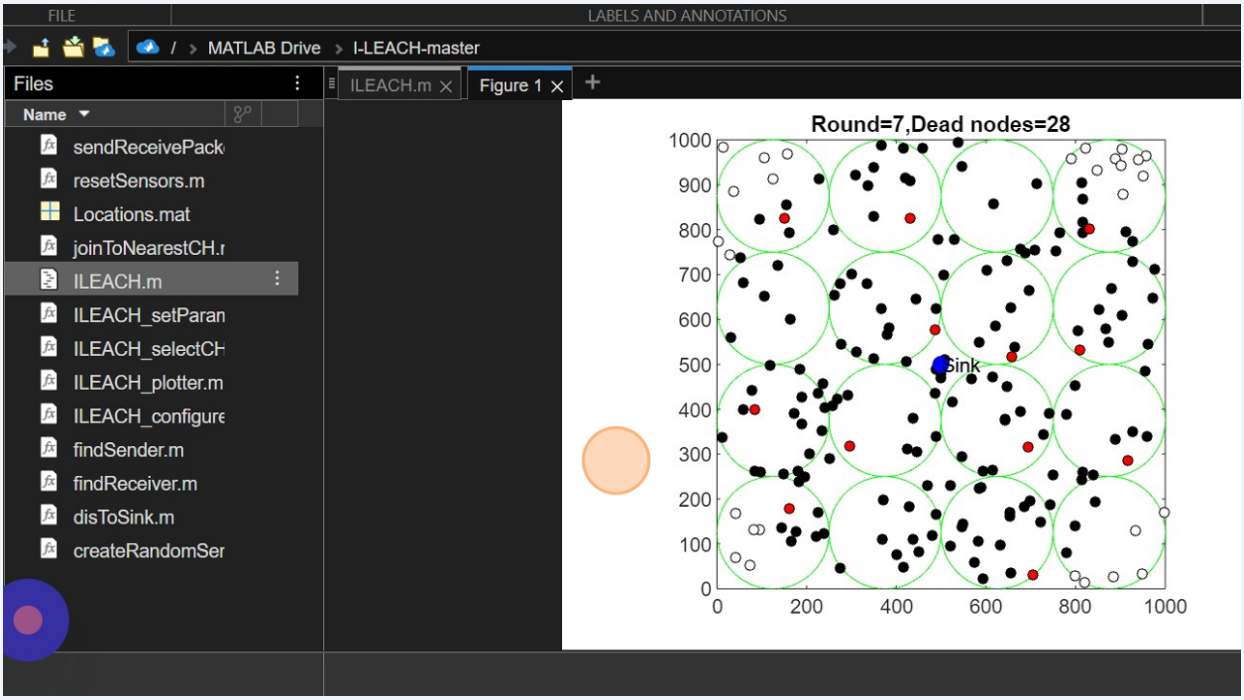
8 As soon as round changes the energy of member nodes become less and a time will come when the nodes will be dead



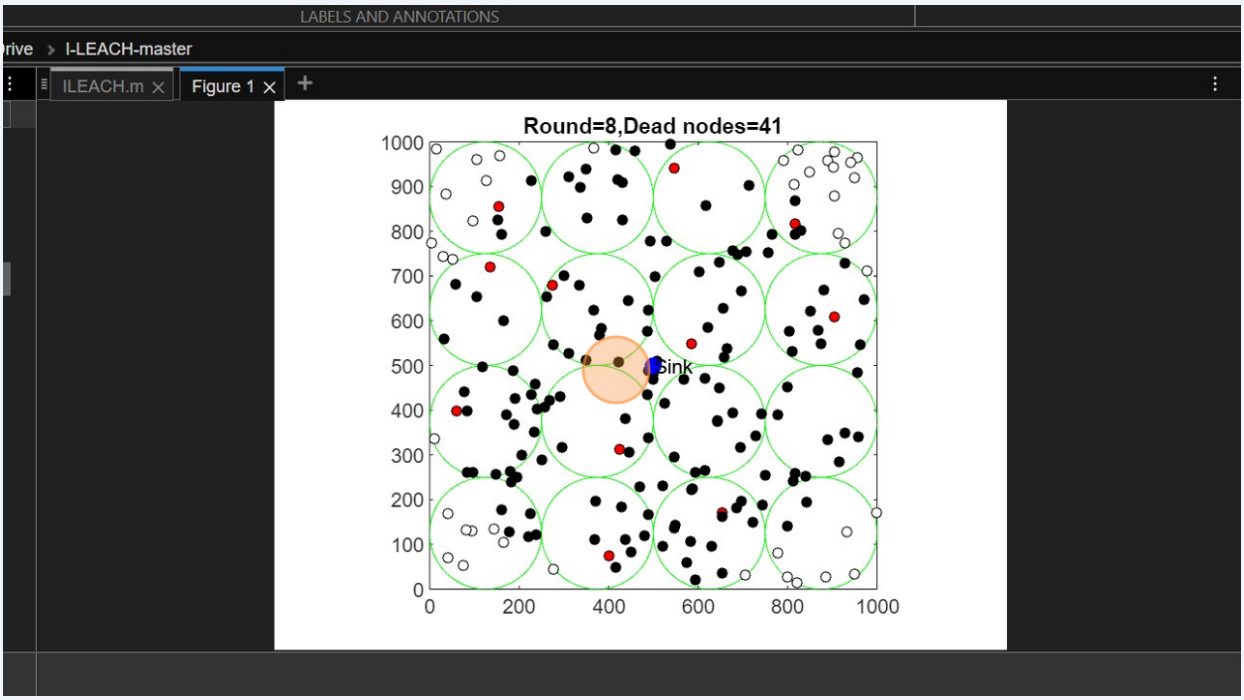
9 Number of dead nodes and Rounds can be observed from graph



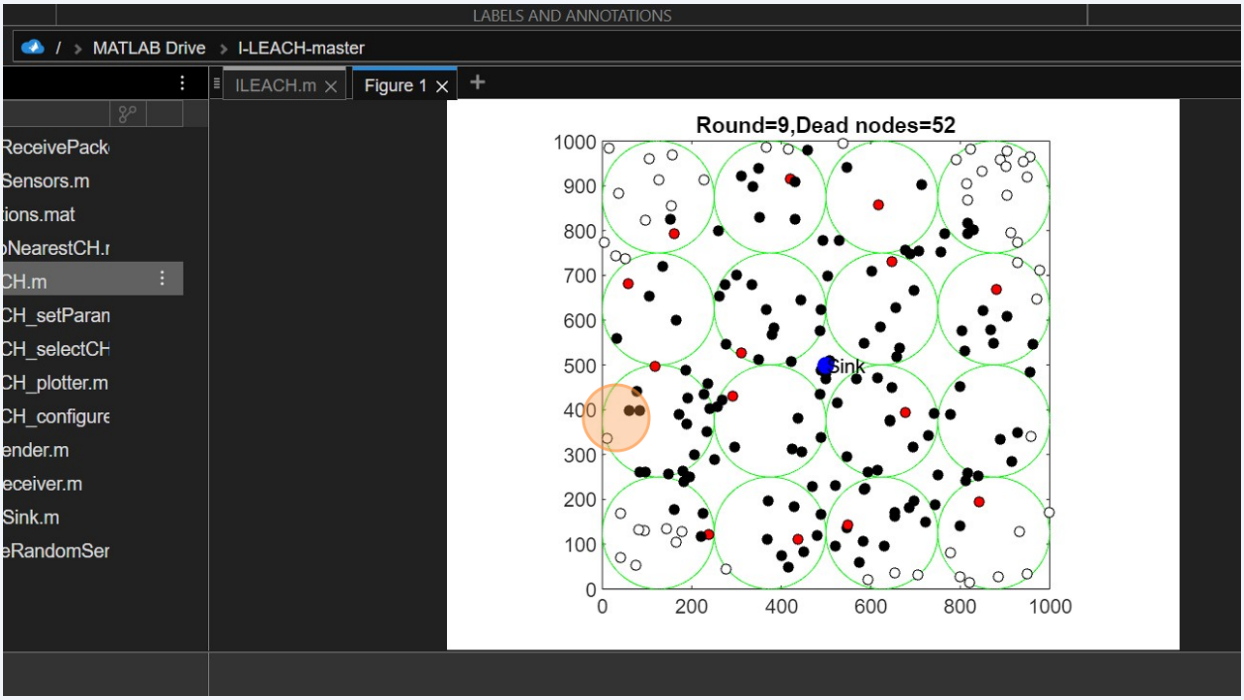
10 Dead notes will be empty circle of white filled circles



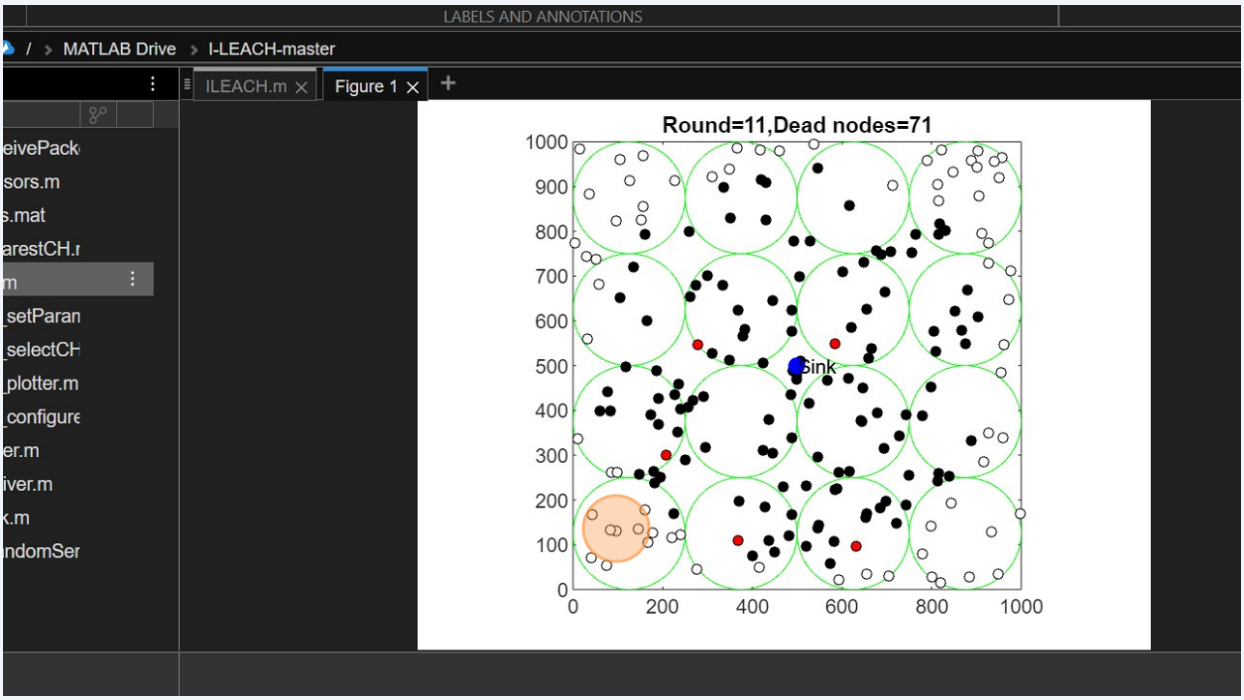
11 This data transmission will keep on going for the number of rounds that was fixed in the code



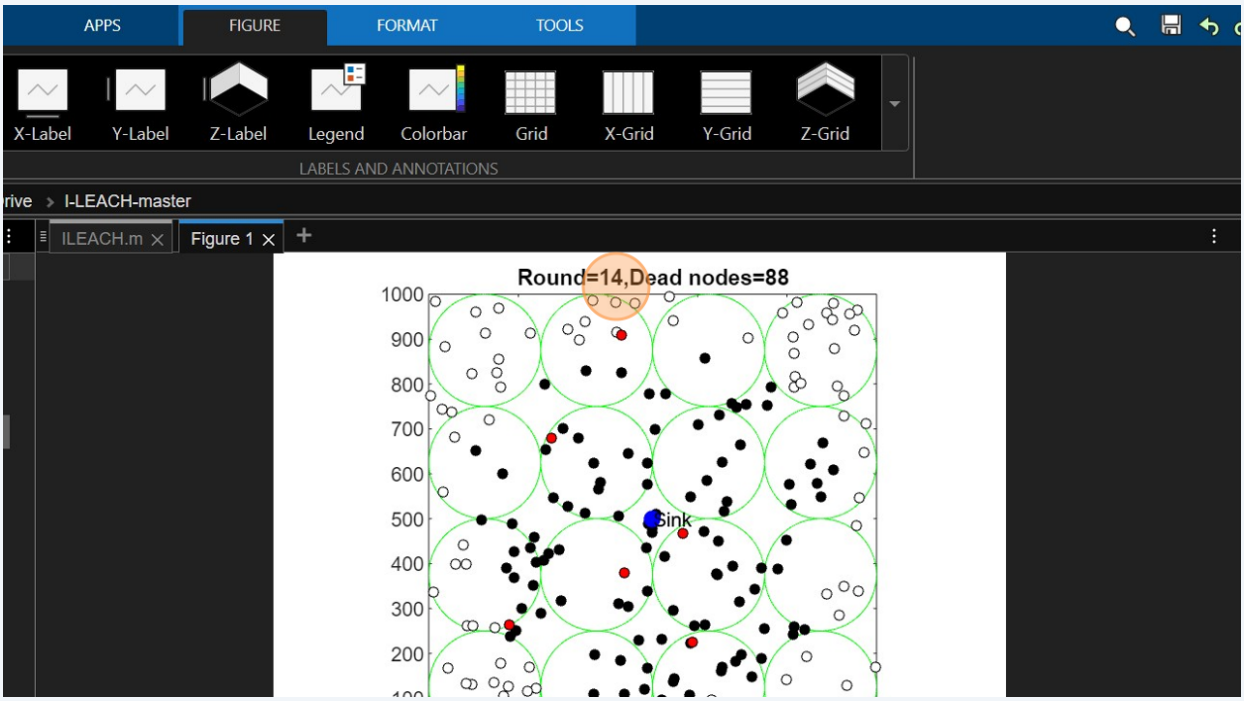
12 CLuster heads are selected on the bases of their energy level



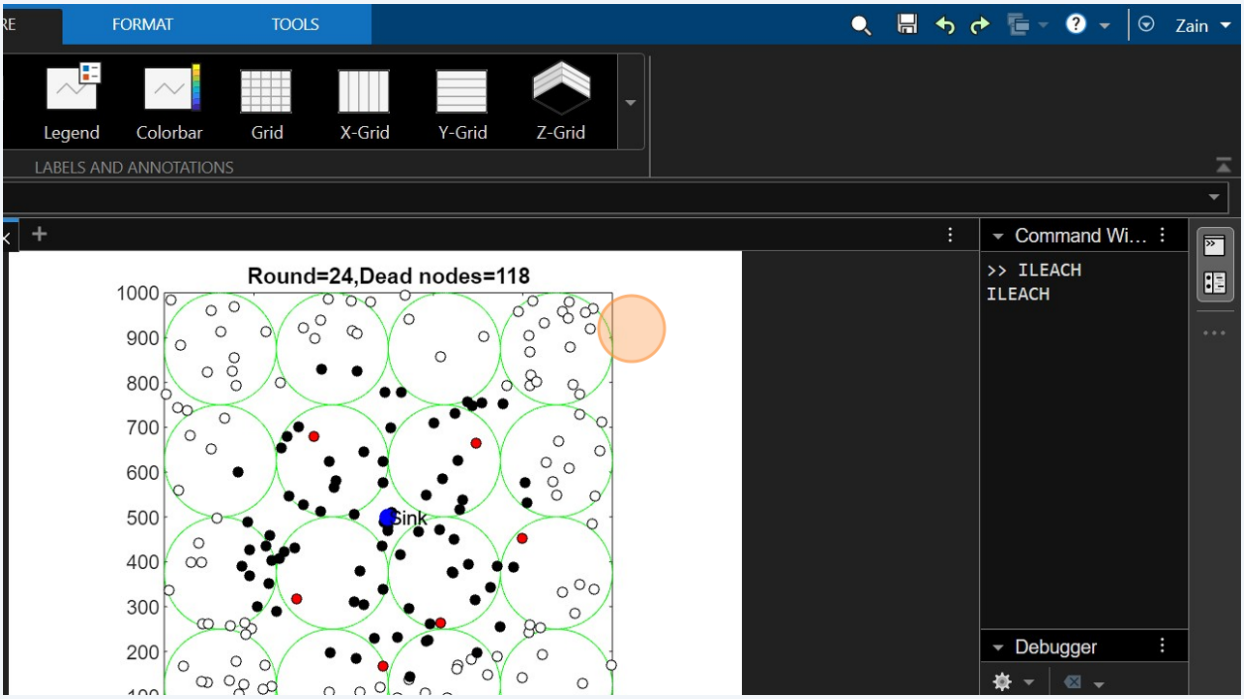
13 In Each round there is a new cluster head in order to conserve energy



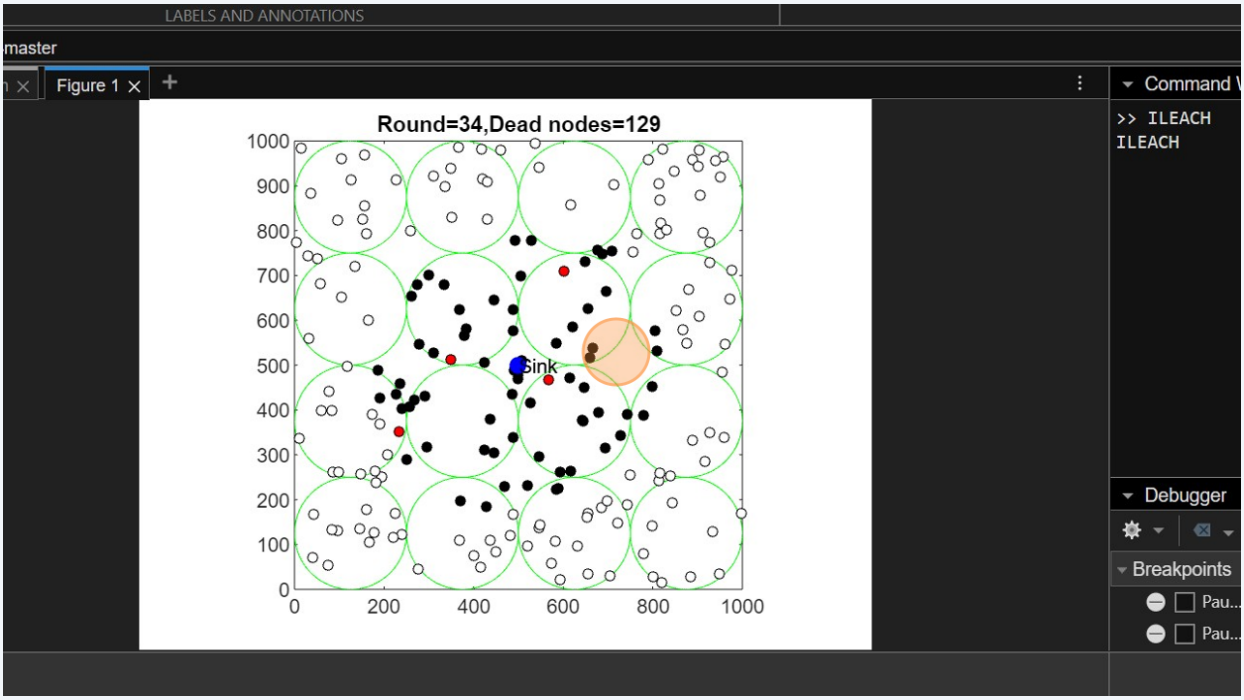
14 Similarly the member node itself do no send any data packet to sink as if each of the nodes sends on its own then sooo much energy will be required and lose so in order to conserve energy the member nodes sends tha data packet to CH and all CHs sends it to sink



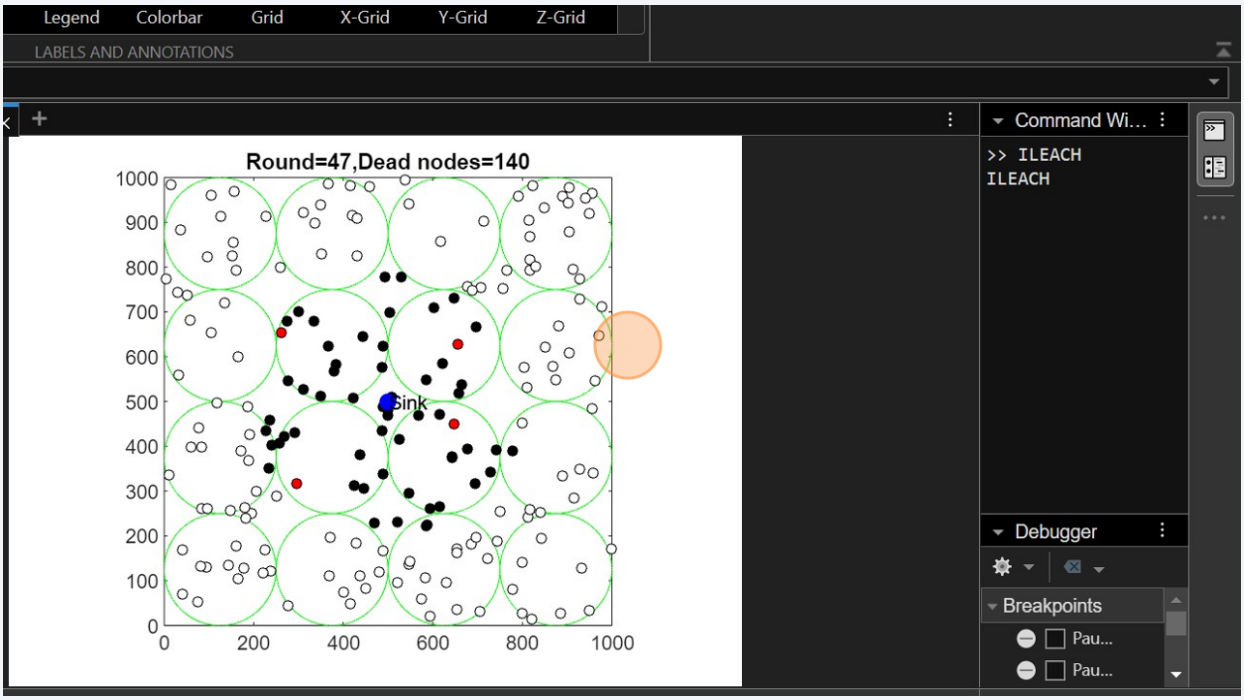
15 Here are some sample screenshots of different rounds



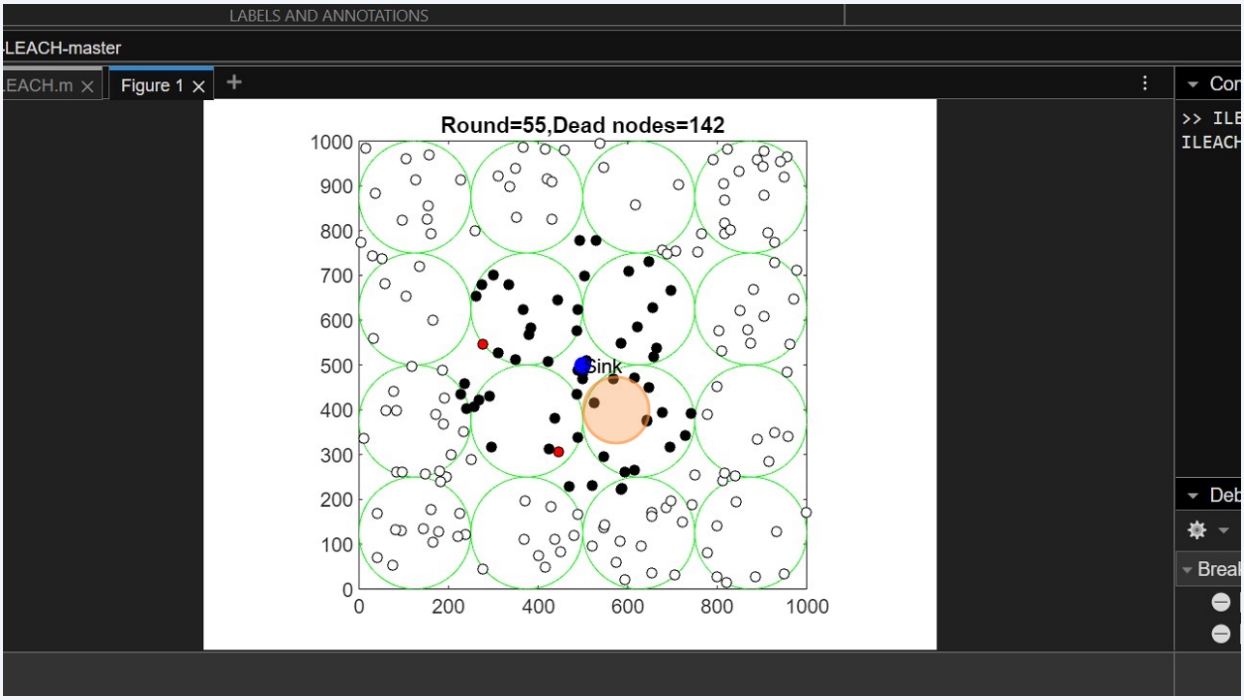
16 Sample 1



17 Sample 2



18 Sample 3



19 Click here.