CupCarbon® User Guide

Version U-One 5.1

www.cupcarbon.com
contact@cupcarbon.com

Document version 7.1

Introduction

CupCarbon is a Smart City and Internet of Things Wireless Sensor Network (SCI-WSN) simulator. Its objective is to design, visualize, debug and validate distributed algorithms for monitoring, environmental data collection, etc., and to create environmental scenarios such as fires, gas, mobiles, and generally within educational and scientific projects. Not only it can help to visually explain the basic concepts of sensor networks and how they work; it may also support scientists to test their wireless topologies, protocols, etc.

CupCarbon offers two simulation environments. The first simulation environment enables the design of mobility scenarios and the generation of natural events such as fires and gas as well as the simulation of mobiles such as vehicles and flying objects (e.g. UAVs, insects, etc.). The second simulation environment represents a discrete event simulation of wireless sensor networks which takes into account the scenario designed on the basis of the first environment.

Networks can be designed and prototyped by an ergonomic and easy to use interface using the OpenStreetMap (OSM) framework to deploy sensors directly on the map. It includes a script called SenScript, which allows to program and to configure each sensor node individually. From this script, it is also possible to generate codes for hardware platforms such as Arduino/XBee. This part is not fully implemented in CupCarbon, it allows to generate codes for simple networks and algorithms.

CupCarbon simulation is based on the application layer of the nodes. This makes it a real complement to existing simulators. It does not simulate all protocol layers due to the complex nature of urban networks which need to incorporate other complex and resource consuming information such as buildings, roads, mobility, signals, etc.

CupCarbon offers the possibility to simulate algorithms and scenarios in several steps. For example, there could be a step for determining the nodes of interest, followed by a step related to the nature of the communication between these nodes to perform a given task such as the detection of an event, and finally, a step describing the nature of the routing to the base station in case that an event is detected. The current version of CupCarbon allows to configure the nodes dynamically in order to be able to split nodes into separate networks or to join different networks, a task which is based on the network addresses and the channel. The energy consumption can be calculated and displayed as a function of the simulated time. This allows to clarify the structure, feasibility and realistic implementation of a network before its real deployment. The propagation visibility and the interference models are integrated and includes the ZigBee, LoRa and WiFi protocols.

CupCarbon represents the main kernel of the ANR project PERSEPTEUR that aims to develop algorithms for an accurate simulation of the propagation and interference of signals in a 3D urban environment.

What's new?

1. **Simple MAC Layer based on a statistical behaviour**
2. Choose different energy consumption models for the radio modules. The models can be written by the user and can consider the weather temperature.
3. **Car mobility** and intelligent mobility which can be based on the sensed information,
4. New ergonomic and professional design and graphical user interface based on the new framework JavaFX. Any action can be cancelled and reconsidered (Undo/Redo),
5. Studied and fine drawing quality with pleasant visual objects for an easy use of the environment. Many options are added to the user in order to choose the preferred visual (dark, light, marking nodes and edges, modifying dynamically the simulation parameters, etc.) and to obtain the best and clean visualization of the network and the working environment,
6. Very easy script language based on the SenScript, which includes intuitive commands,
7. Possibility to consider the topology of a city (buildings) as well as the radio propagation and visibility in this environment,

8. Possibility to include in the simulation the interferences of signals at PHY layer for the Acknowledgment messages. The interferences use very accurate models based on the alpha-stable distribution,
9. Possibility to consider many radio modules and standards (802.15.4, WiFi and LoRa) in the same sensor node. Radio modules can be changed either in the environment or during simulation (in the Script),
10. Possibility to consider the clock drift.
11. Arduino Code generation for basic networks.

This Document

This document is composed of 5 parts:

1. Presentation of the environment CupCarbon: this environment is composed mainly on the graphical user interface and the map,
2. Presentation of the objects of CupCarbon: this section presents all the objects that can be added on the map of CupCarbon and how to manipulate and configure them,
3. Presentation of SenScript: the script used to program the sensor nodes. All the commands are presented with some examples to facilitate their use,
4. Presentation of 17 basic examples: these examples allow to familiarize with the CupCarbon environment and its objects in order to understand how it works and how to design complex objects,
5. Presentation of 4 advanced examples: this section presents some complex projects in order to give an idea on how to use CupCarbon.

Acknowledgment and contact

We want to thank all the participants of this project CupCarbon that is a part of the research project PERSEPTEUR supported by the French research agency ANR (*Agence Nationale de la Recherche*) under the reference ANR-14-CE24-0017-01. The partners of this project are:

- MMU (https://www2.mmu.ac.uk): Prof Mohammad Hammoudeh
- Virtualys company (http://www.virtualys.fr): Dr Olivier Marc
- IRCICA research institutes (http://www.ircica.univ-lille1.fr): Prof Laurent Clavier
- Xlim laboratory (http://www.xlim.fr): Dr Pierre Combeau
- Lab-STICC laboratory (http://lab-sticc.fr/en/index): Prof Ahcène Bounceur

To contact us please, use the one of the following emails:

- contact@cupcarbon.fr
- cupcarbon@univ-brest.fr

This work was a part of the research project PERSEPTEUR supported by the French Agence Nationale de la Recherche ANR.

New Book

We recommend the following book as it contains detailed explanations of new algorithms to determine the boundary nodes and the leader in distributed systems in general and IoT and WSNs in particular. The title of this book is: “Boundaries and hulls of Euclidean graphs: from theory to practice”.

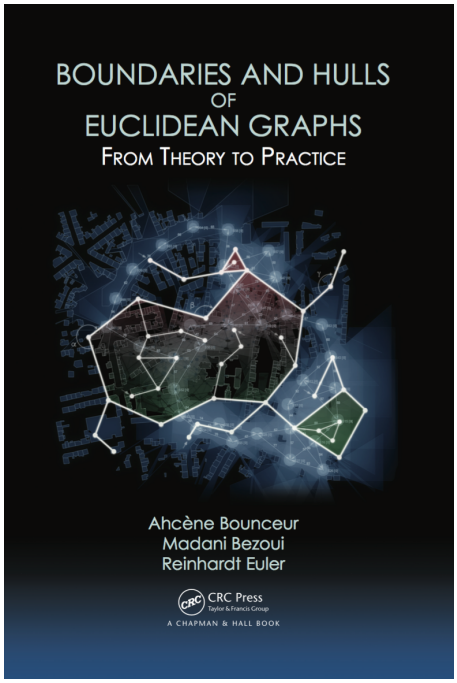


Table of content

CupCarbon Environment

To execute CupCarbon (jar file), use command window and go to the directory where the jar file is located. Then execute the following command:

```
java -jar CupCarbon.jar
```

In the case of existence of a proxy, use the following command:

```
java -jar CupCarbon.jar proxy_host_name proxy_number_of_port
```

As shown by Figure 1, the CupCarbon Graphical User Interface (GUI) is composed of the following five main parts:

1. The map (in the center)
2. The menu bar (on the top)
3. The Toolbar (below the menu)
4. The parameter menu (on the left)
5. The state bar (at the bottom)
6. The console (in the version 5 the console is separated from the main interface)

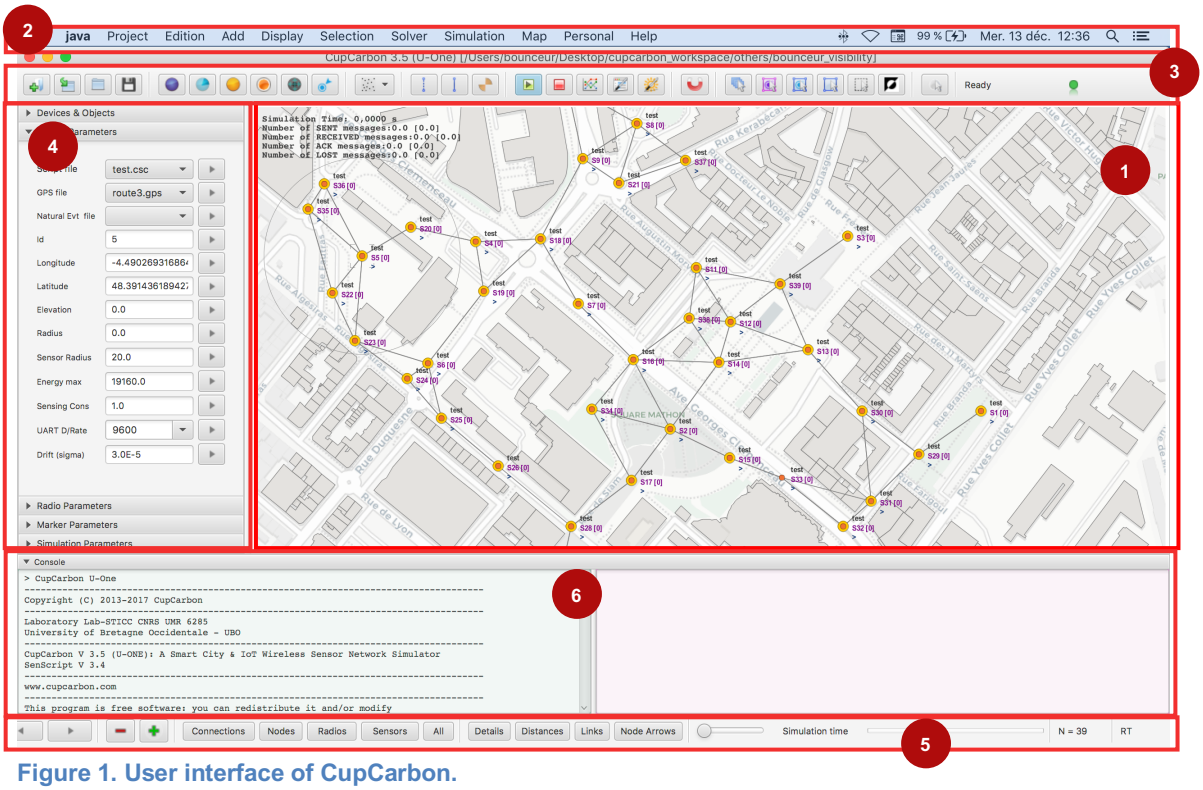


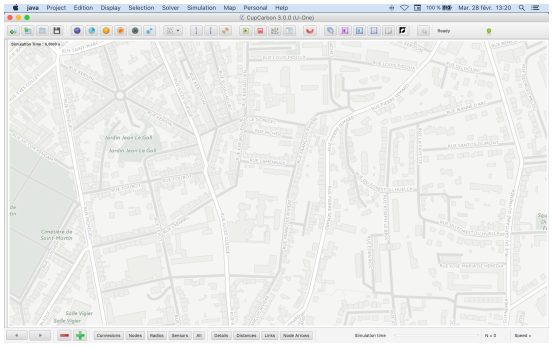
Figure 1. User interface of CupCarbon.

These parts will be detailed in the following.

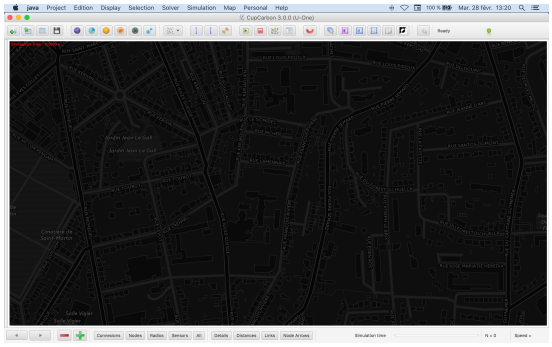
The Map

- The map is the main object of the simulator CupCarbon. It is the part where the network and the objects of the project can be designed. The map can be changed according to the preference of the user or the way the information must be presented. Figure 2 shows the 12 possible maps or backgrounds, that are:
    - (a) Light OpenStreetMap
    - (b) Dark OpenStreetMap
    - (c) Dashed background
    - (d) Small cell white grid background
    - (e) Small cell black grid background
    - (f) Black background
    - (g) White background
    - (h) Mean gray cell background
    - (i) Notebook background
    - (j) Mean blue cell background
    - (k) Google map
    - (l) Google map (Satellite)

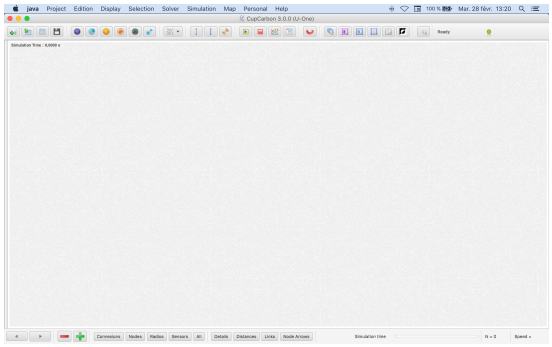
and B&W (Black and White) OpenStreetMap, which is not presented bellow.



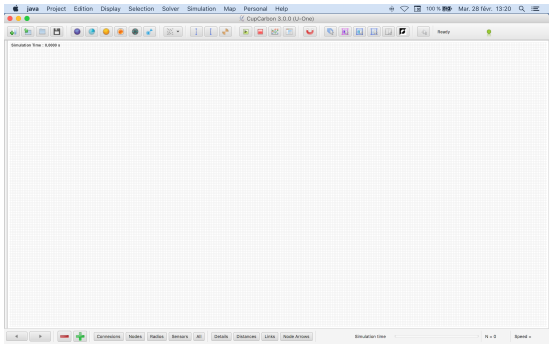
(a)



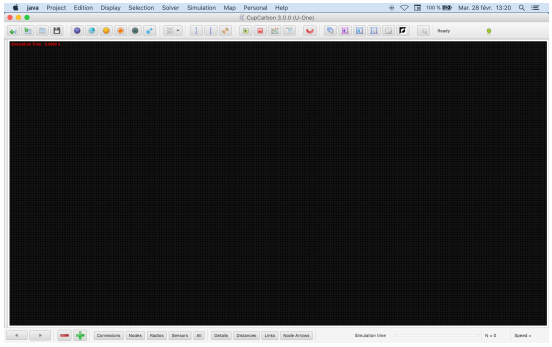
(b)



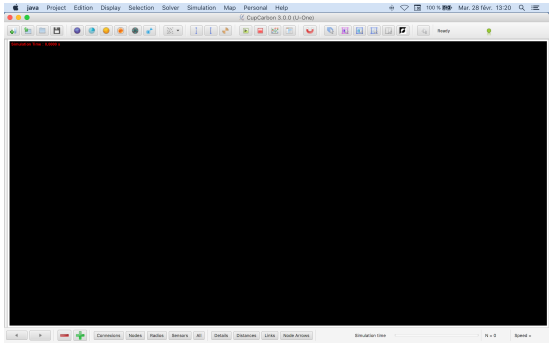
(c)



(d)



(e)



(f)

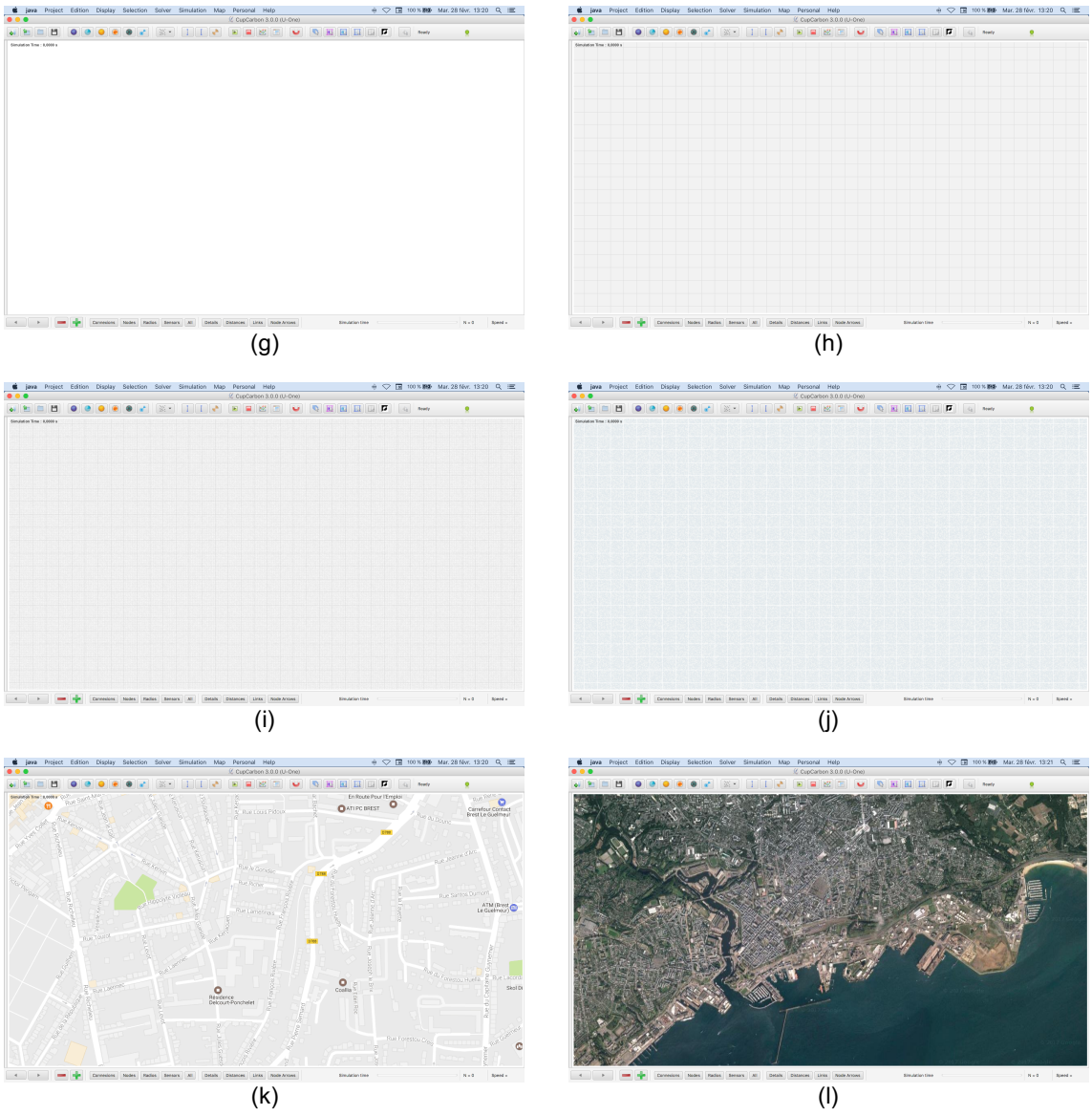


Figure 2 Maps of CupCarbon.

The simulation time is displayed on the top left part of the map. During simulation this time is displayed in red color and an additional red rectangle is drawn around the map in order to detect the simulation process (cf. Figure 3 (a)). In this part other information about the messages are also displayed, that are the number of sent, received, ACK and lost messages. This part can be hidden and displayed using ALT+D keys.

(a) (b)

Figure 3 Displaying information on the map during the simulation time.

However, when the simulation is finished, this rectangle is removed and the simulation time is displayed in green color (cf. Figure 3 (b)).

Since in the environment of CupCarbon we have both objects and the map, any mouse dragging is used to move either the objects or the map. Then it is impossible to make any standard mouse selection of objects (by mouse dragging). To do this, just use the right button of the mouse instead of the left one.

The Menu bar

As shown by Figure 4, the menu bar is composed of 10 items that are:

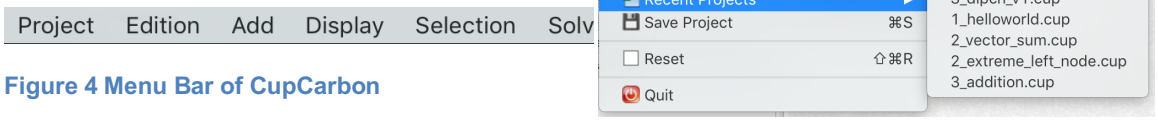
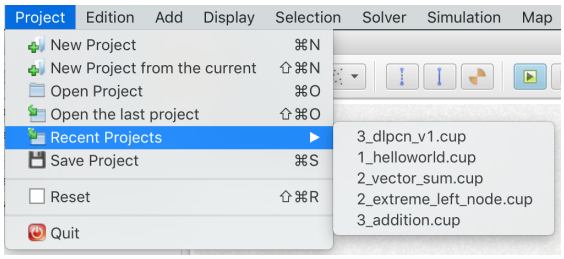


Figure 4 Menu Bar of CupCarbon



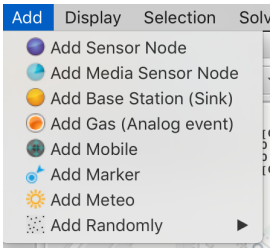
1. Project
2. Edition
3. Add
4. Display
5. Selection
6. Solver
7. Simulation
8. Map
9. Personal
10. Help

Each item is detailed in the following:

Project menu

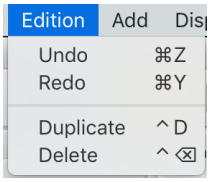
The **Project** menu contains the following sub items:

1. **New Project:** it allows to create a new project starting from a completely empty map. Be careful, if any project has been already started and designed before, this option will delete all the existing work. In order to avoid this, you have to choose the item “New Project from the current”.



2. **New Project from the current:** it allows to create a new project by considering an existing work. This option can be used in order to duplicate projects.
3. **Open project:** it is used to open existing projects.
4. **Open the last project:** it allows to open the last opened project.
5. **Recent Projects:** it allows to open one of the last 5 opened projects.
6. **Reset:** it reset the entire environment. If an project already exists and not saved, this option will cause a total loss of this project.
7. **Quit:** it allows to close and exit the project.

Edition menu



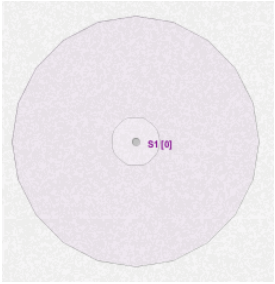
The **Edition** menu contains the following sub items:

1. Undo: to cancel an action
2. Redo: to reconsider a canceled action
3. Duplicate: to duplicate select objects on the map
4. Delete: to delete any selected object on the map

Add menu

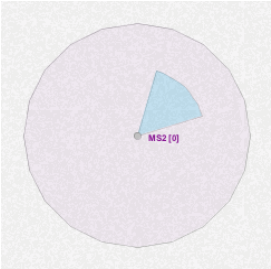
The Add menu allows to add objects on the map. A detail description of each object is given in Section CupCarbon objects. In the following, we will just show how to add these objects from the menu bar.

1. **Add Sensor Nodes:** A sensor node is an object that can detect any digital event (motion event like mobiles), send and receive data. It can be also mobile. The visible parameters of a sensor node are: the radio range, the radio of the sensor unit and the name.

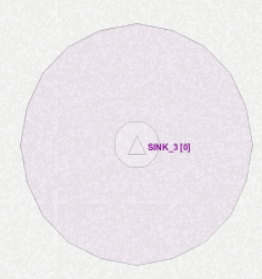


A sensor node has many parameters; it can contain many radio modules, a battery and a sensing unit.

2. **Add Directional Sensor Nodes:**

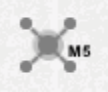


3. **Add Base stations (Sinks):**

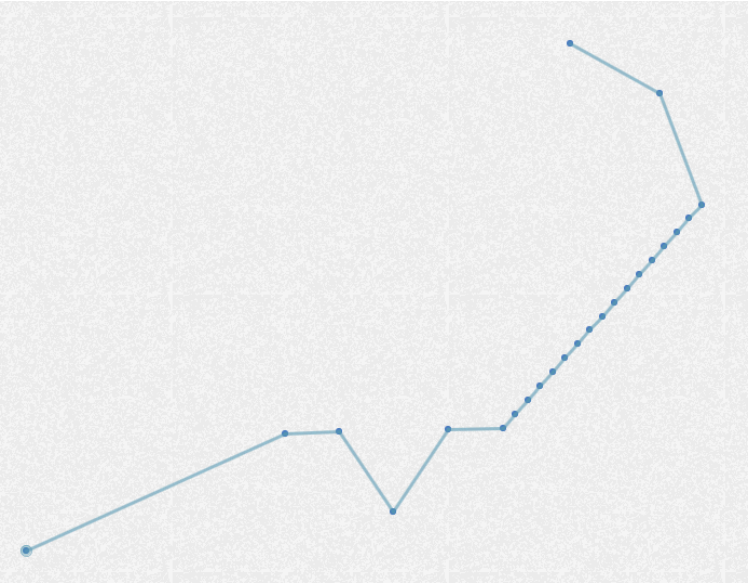


4. **Add Gasses:** used to generate analog events in order to simulate environment parameters like the temperature, the humidity, gasses, etc. This object requires the use of the Natural Event Generator window in order to generate files with the desired values. Mainly, the generated values are based on the Gaussian distribution. However, it is possible to add an existing, real or not, data file.

5. **Add Mobiles:** used to simulate mobiles. Markers are also used to create routes followed by mobiles. Each mobile must have its own route. They are used also to generate digital events.

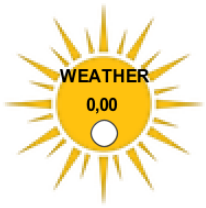


6. **Add Markers:** used mainly to generate routes for mobiles (or mobile sensors). They are used also to generate sensor nodes, create new buildings, and to indicate the area of generating buildings or random sensor nodes.

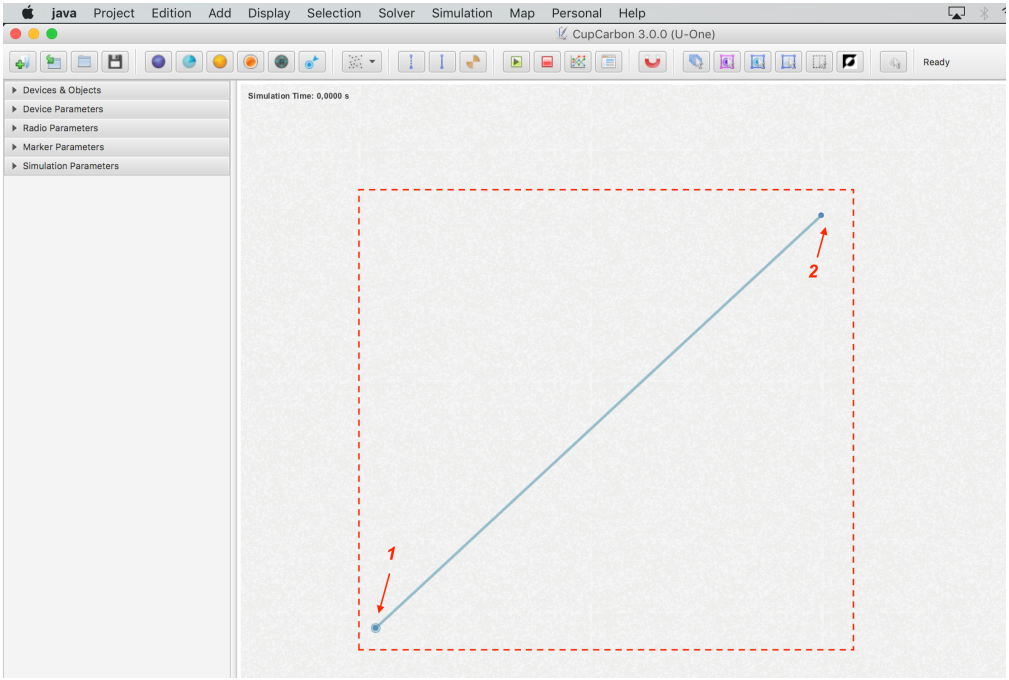


7. **Add Meteo node:** A Meteo (weather) node is used to add a variating temperature related to the environment. This is useful for the battery consumption models that are dependent to the weather. To generate temperatures during a day, many days or many hours one use the Natural event generator (cf. the Simulation menu presented bellow) using the following button of the menu Simulation or the tool bar:

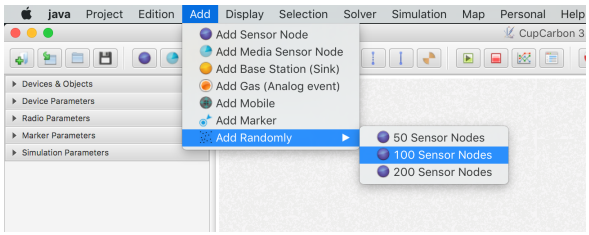
It is possible to add only one weather node in the project.

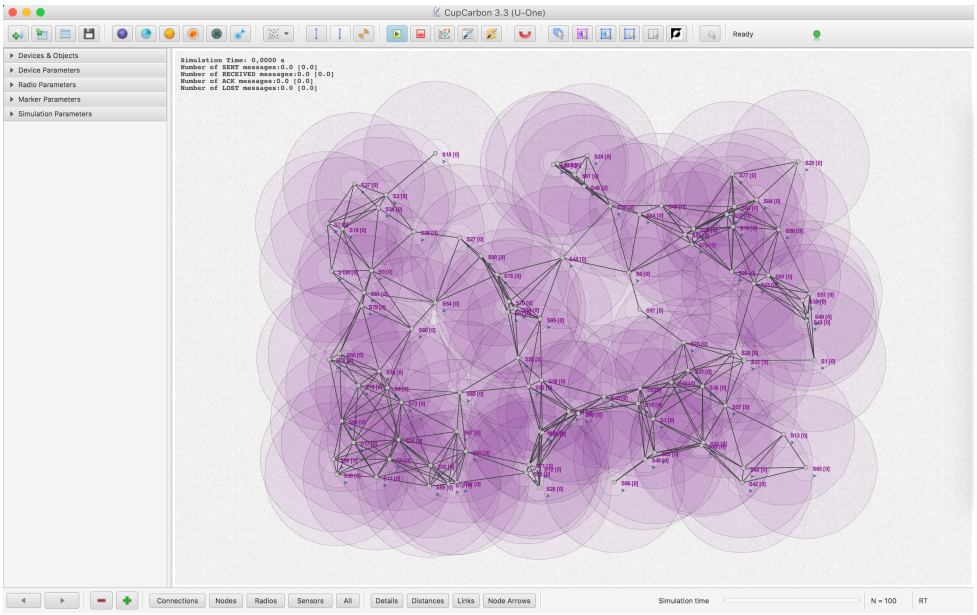


8. **Add randomly sensor nodes:** it allows to add randomly a certain number of sensor nodes in a selected area. To use this option, first, two markers must be added. The first marker must be in the bottom left of the map and the second one in the top left of the map, like it is shown by the following Figure. The red rectangle on this figure shows the area where the sensor nodes will be added randomly.

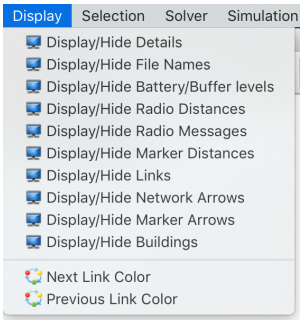


Once the two markers are added, click on the number of sensor nodes to add. In the following figure, an example of adding randomly 100 sensor nodes is shown.



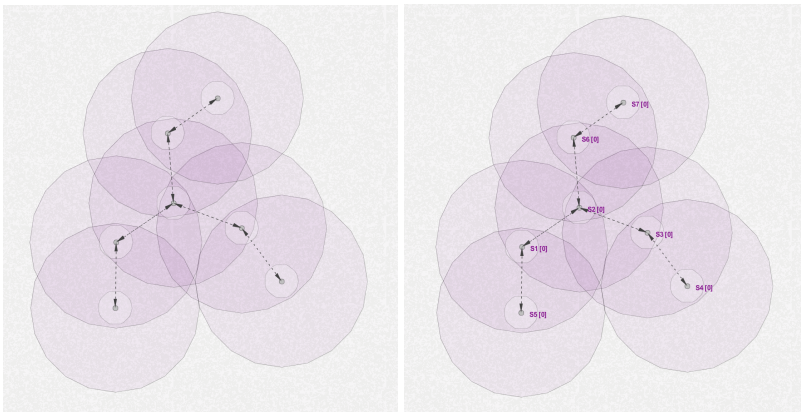


Display menu

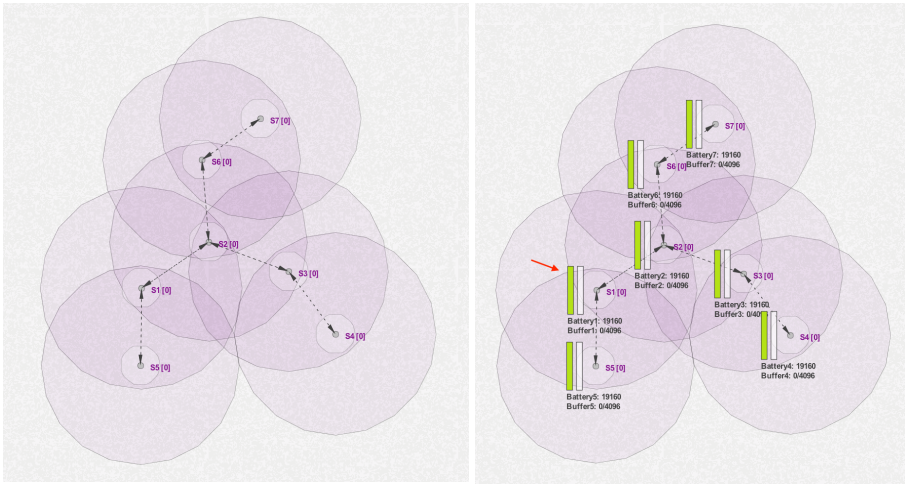


The **Display** item contains the following sub items:

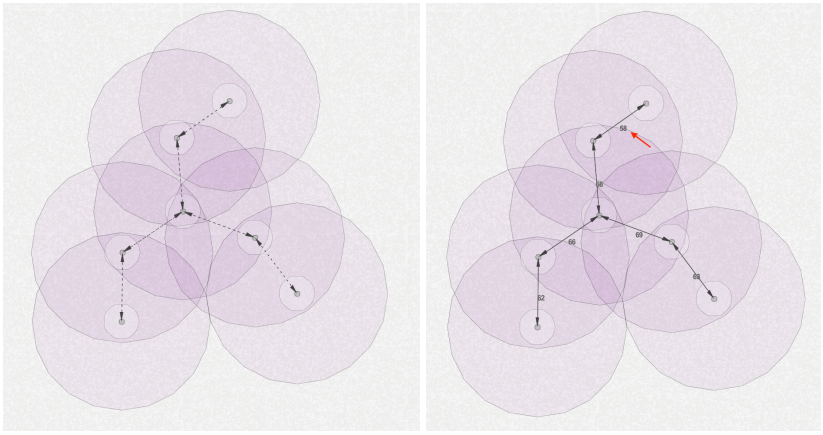
1. **Display/Hide Details:** show/hide the names of the selected sensor nodes
2. **Display/Hide File Name:** show/hide the assigned SenScript files name of the selected sensor nodes



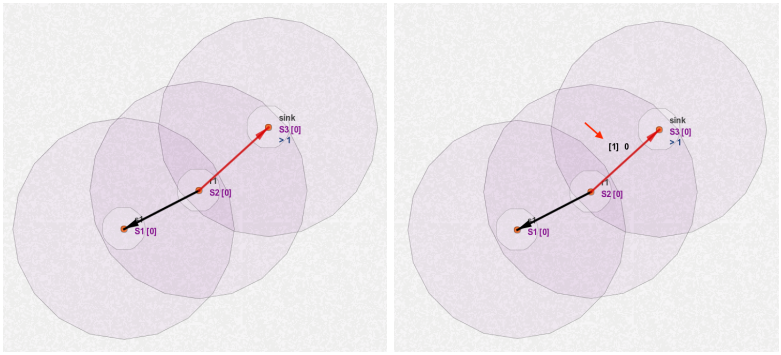
3. **Display/Hide Battery/Buffer levels:** show/hide the battery and the buffer levels of the selected sensor nodes.



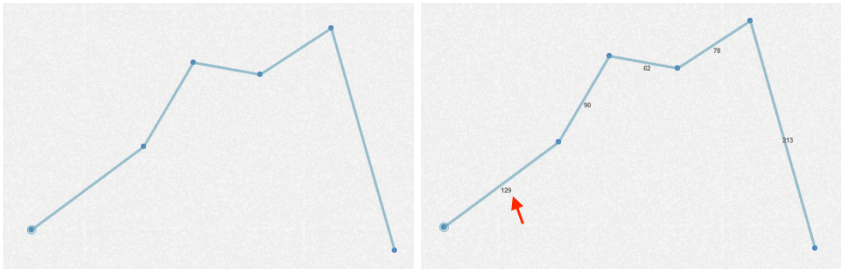
4. **Display/Hide Radio Distances:** show/hide the distances in meter between communicating sensor nodes.



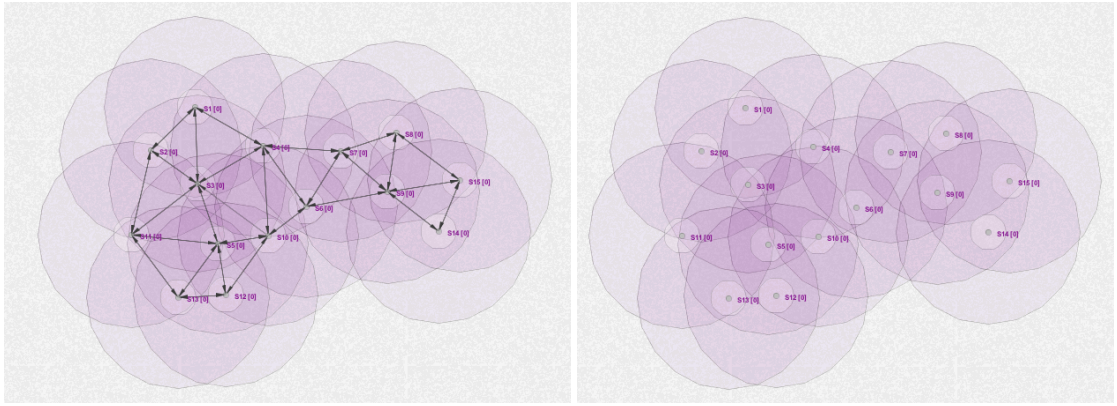
5. Display/Hide Radio Messages: show/hide messages being sent. If the ACK and Show ACK are activated, these messages will be preceded by the number of sending attempts.



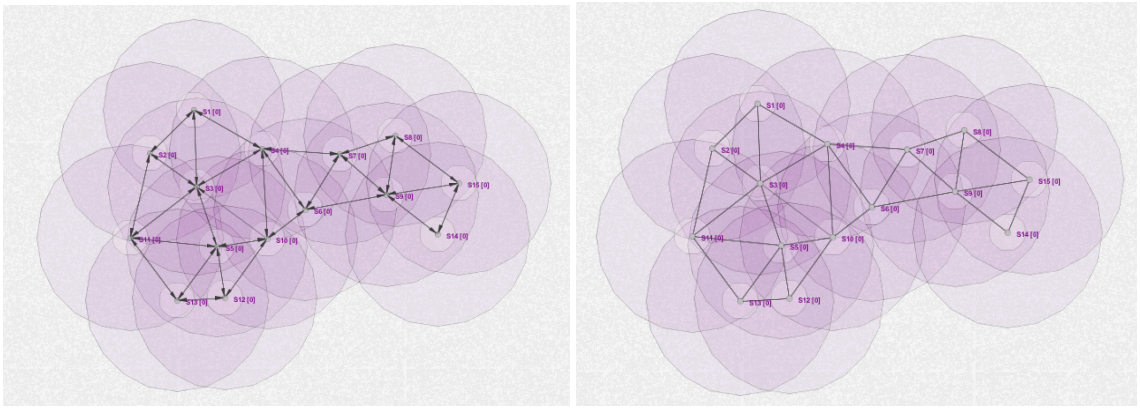
6. Display/Hide Marker Distances: show/hide the distances between markers.



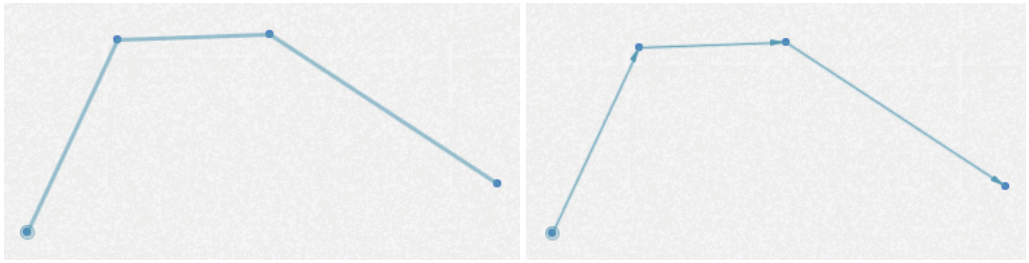
7. Display/Hide Links: show/hide the links between communicating sensor nodes.



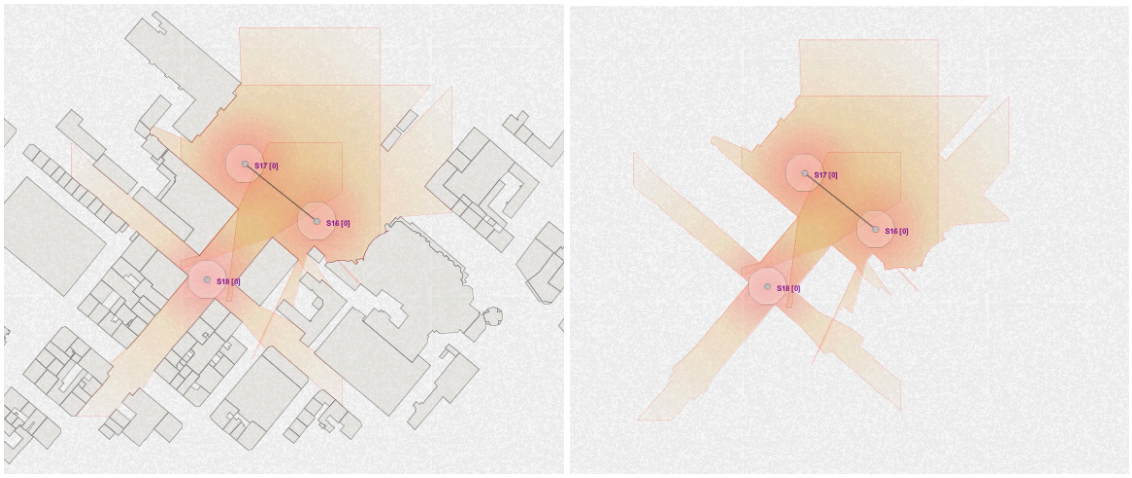
8. Display/Hide Network Arrows: show/hide the arrow of the links between communicating sensor nodes.



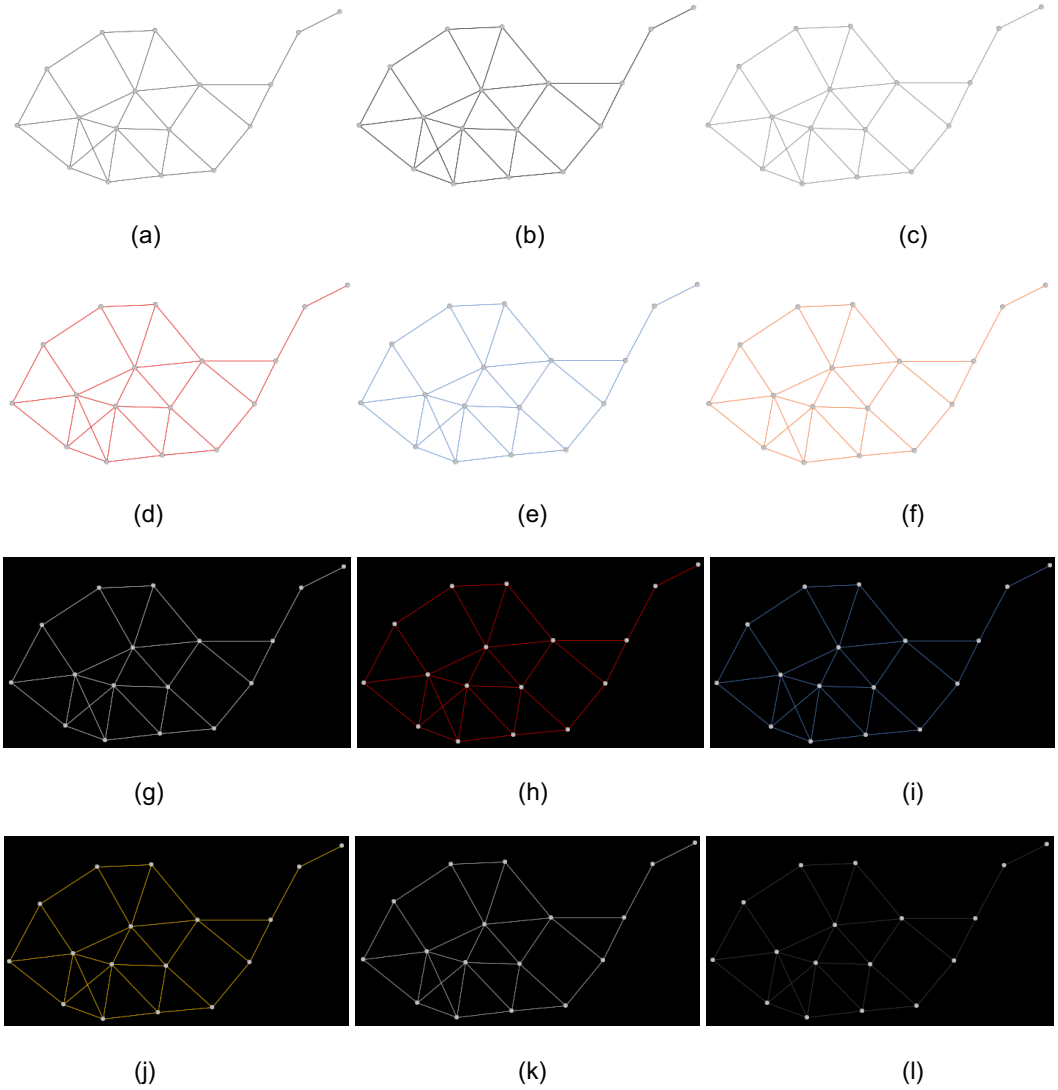
9. Display/Hide Marker Arrows: show/hide arrows between markers.



10. Display/Hide Buildings: show/hide the loaded/created buildings.



11. Next Link Color and Previous Link Color: change the color of the links between communicating sensor nodes.



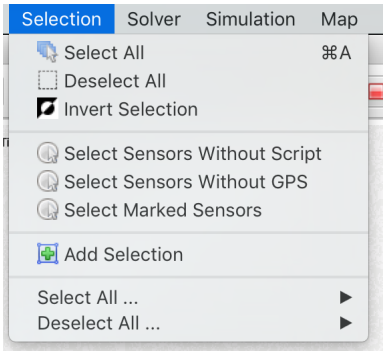
(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Selection menu

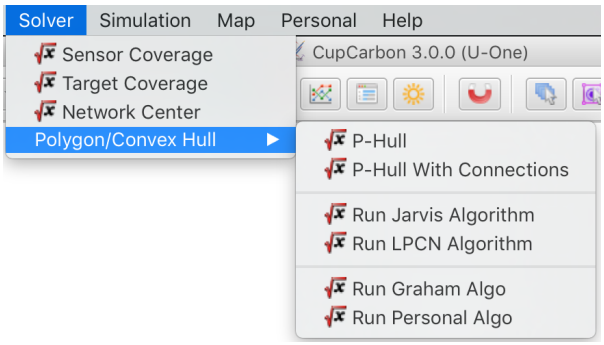


The **Selection** menu contains the following sub items:

1. Select All
2. Deselect All
3. Invert Selection
4. Select Sensors Without Script
5. Select Sensors Without GPS
6. Select Marker Sensors
7. Add Selection
8. Select All … and Deselect All …

More selection options are available in the left menu in the part Devices & Objects (tab: Selection). The button “Add Selection” allows to add a selection to the previously selected objects.

Solver menu

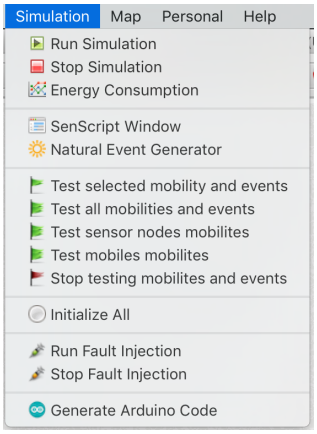


The **Solver** menu contains the following sub items:

1. Sensor Coverage algorithm
2. Target Coverage algorithm
3. Target Coverage (Th) algorithm
4. Scheduling
5. Network Center
6. Hull

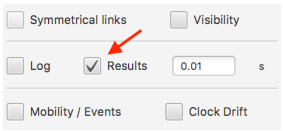
This item contains centralized algorithms, which are out of scope for this document.

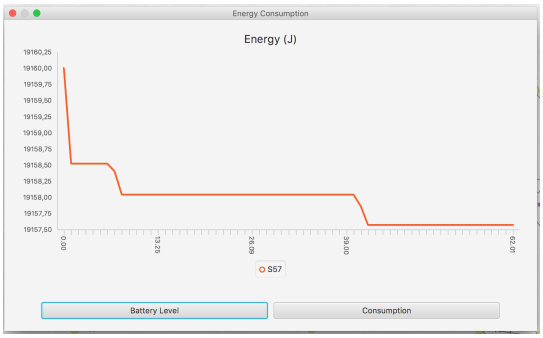
Simulation menu



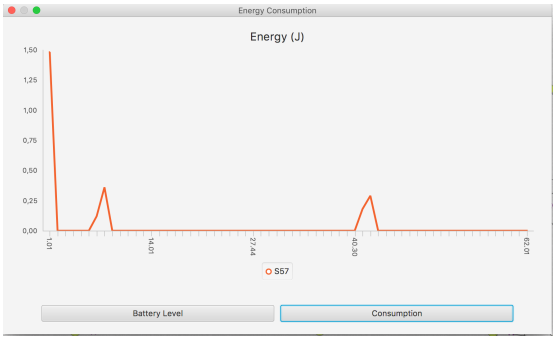
The **Simulation** menu contains the following sub items:

1. Run Simulation: to start the simulation
2. Stop Simulation: to stop the simulation
3. Energy Consumption: to display the graph of the energy consumption for the selected sensor nodes once the simulation is finished. You must first check the box *Results* before running the simulation. Two kinds of graphs are possible. The first one shows the state of the battery with respect to the simulation time and the other one shows the consumption of a sensor with respect to the simulation time.

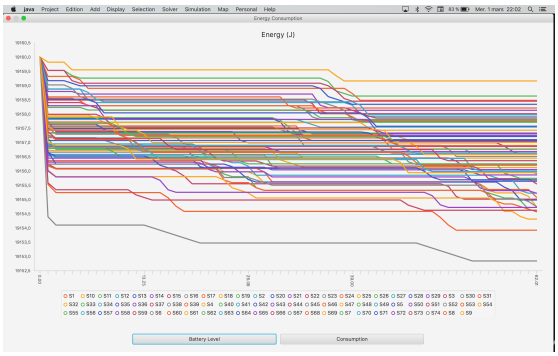




(a)



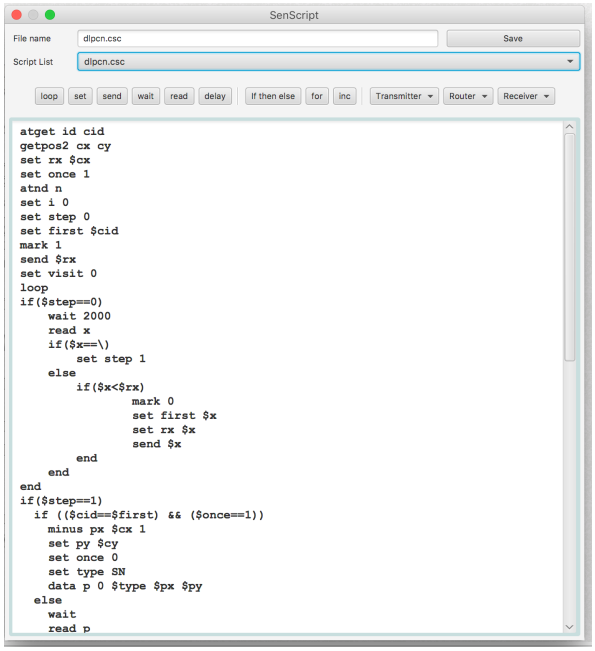
(b)



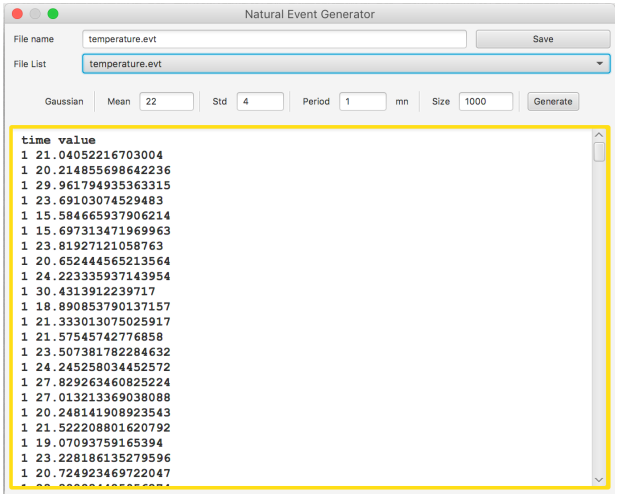
(a)

(b)

4. SenScript Window: to open the SenScript window



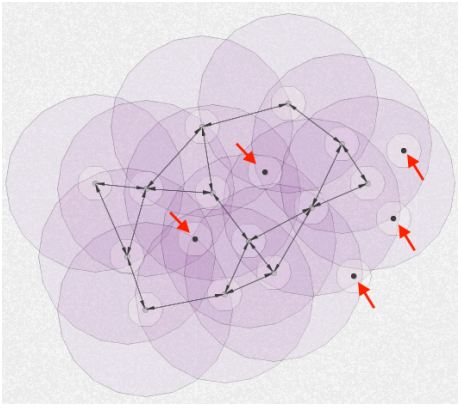
5. Natural Event Generator: to generate natural event values. This generator allows to generate values that can be detected by the sensor nodes, for example, the temperature, the humidity, the gas/pollution, etc. It can be used also to generate the weather temperatures.



It is possible to copy directly on the text area a real data file or which is generated with another tool. Note that the first column shows the time in seconds required to generate the next value.

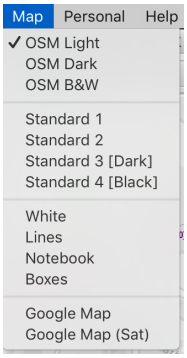
6. Test selected mobility and events: to simulate and check the mobility of a selected mobile, sensor node and events (Gas)
7. Test all mobilities and events: to simulate the mobility and the event of all devices (mobile, sensor node, gasses)
8. Test sensor nodes mobilities: to simulate the mobility of all sensors
9. Test mobiles mobilities: to simulate the mobility of all mobiles
10. Stop testing mobilities and events: to stop the simulation of mobility and events
11. Initialize All: to initialize the simulation environment and the information displayed during simulation like the sending messages arrows

12. Run Fault injection: to generate faulty sensor nodes randomly (during simulation or not). Faulty sensors can also be activated by selecting a sensor and clicking on the key ‘k’.



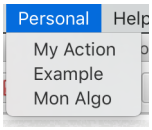
13. Stop Fault Injection: to stop the process of generating faulty sensor nodes
14. Generate Arduino Code: to generate automatically the corresponding Arduino code of each sensor node from its SenScript code. The Xbee parameters file must be added in the directory xbee.

Map menu



The **Map** menu allows to choose the preferred map and environment. All the available maps are presented in the beginning of this section.

Personal menu



The **Personal** menu is used to create personal SenScript functions and to use the VIZOR programming. This topic is presented in detail in the SenScrip reference guide.

Help menu

The **Help** menu contains the help link and the about box.

The Toolbar

The toolbar of CupCarbon is shown by Figure 5 and it is used to access to the main actions of CupCarbon.

Figure 5 Toolbar of CupCarbon.

It is composed of 7 parts that are:

Project part

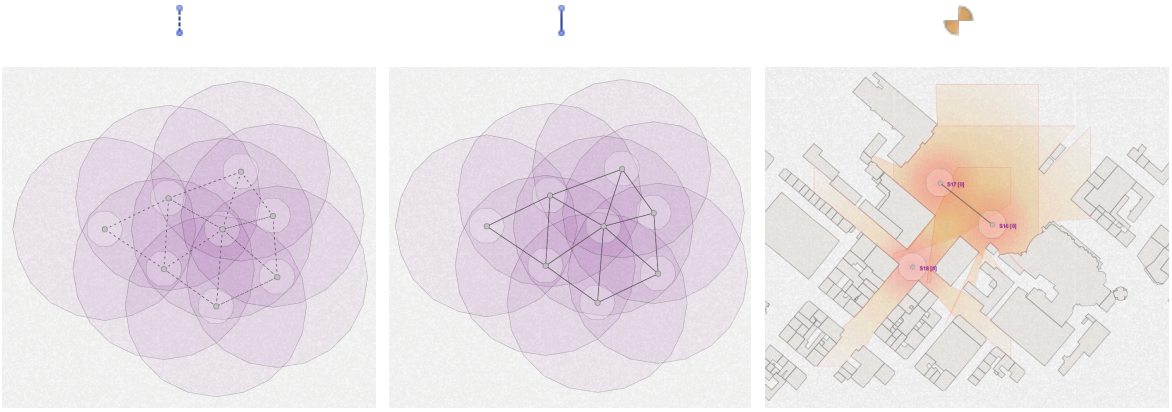
It allows to create new project, open the last project, to open a project and to save a project (cf. File menu).

Add object part

It allows to add objects (Sensor node, Directional Sensor node, Base station, Gas, Weather, Mobile, Marker, random sensor nodes) on the map (cf. Add menu).

Connections part

It allows to draw normal or radio propagation based connections between sensor nodes. It allows also to calculate the visiblity of the radio of the sensor node by considering the buildings of a city.

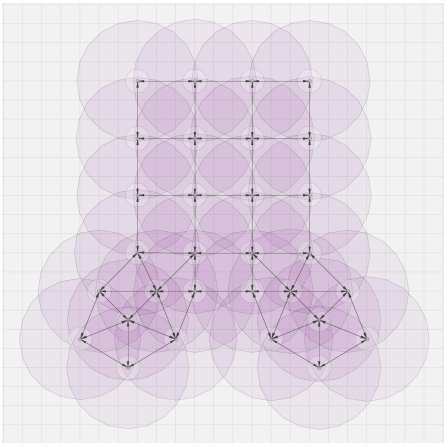


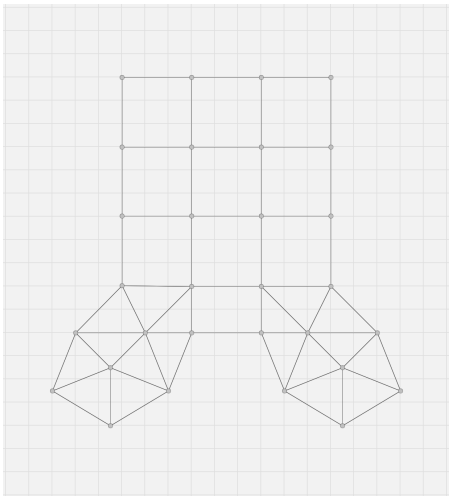
Simulation part

It allows to run the simulation, stop the simulation, draw the energy consumption function, open the SenScript window and to open the Natural Event generator. These options are explained above in the menu bar section.

Magnetism part

It allows to add objects in an (invisible) grid. It is recommanded to use the map (Mean gray cell background), shown by Figure 1 (h), when this option is used.



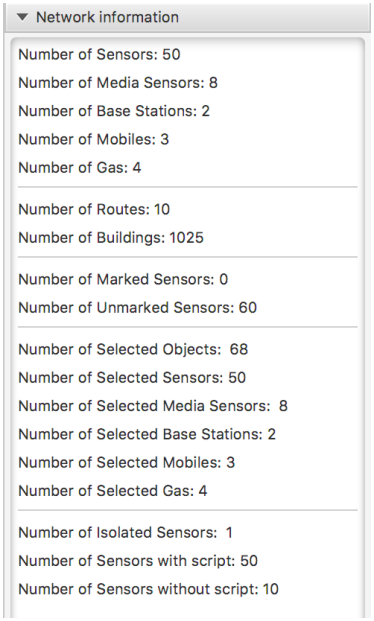


Selection part

It allows to select all, select all sensor nodes, select all markers, select sensor nodes/markers, deselect all and to invert selection (cf. Selection menu).

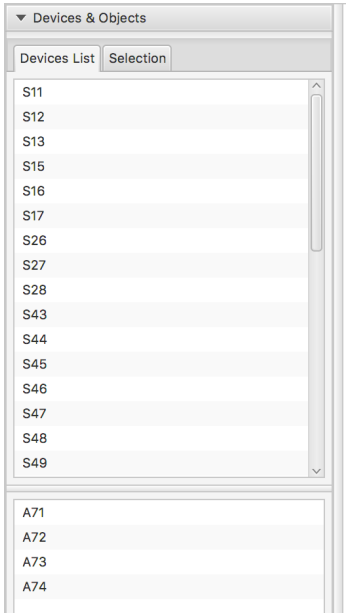
The parameter panel

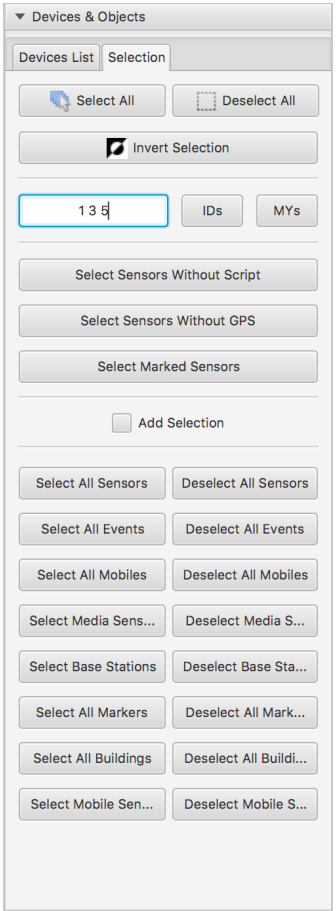
Network information panel



This panel shows some information about the network like the number of sensors, the number of marked sensors, the number of isolated sensors, etc.

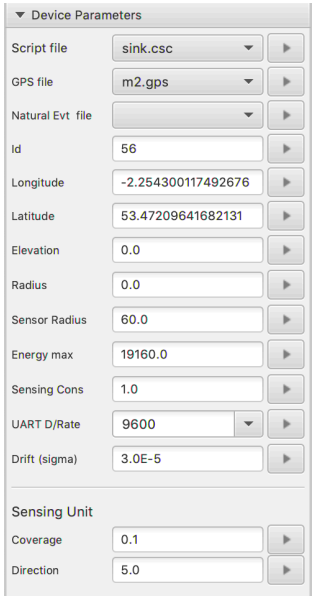
Devices & Objects [Device List] [Selection] panel





This panel has two tabs. The first one is Device List which allows to select an object on the map by its name. The second one is Selection tab that allows to select/deselect objects by their type. It is also possible to select objects by their MY addresses or their IDs. These addresses/IDs can be entred as a list of numbers in the corresponding text field.

Device Parameters panel

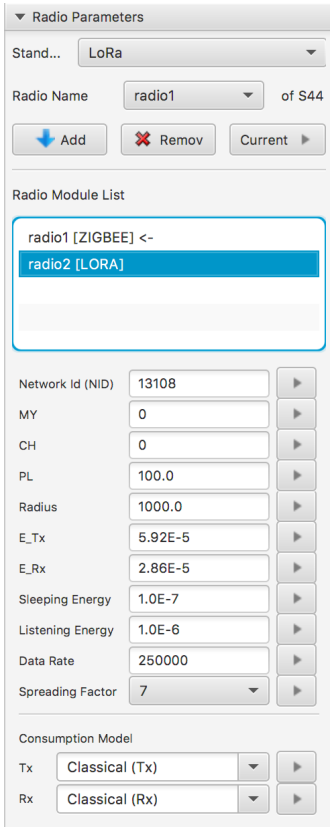


This panel allows to modify the parameters of the selected objects like:

1. Script file: to assign the SenScript file
2. GPS File: to assign the route file
3. Natural Event File: to assign the natural event file generated from the Natural Event generator
4. Id: to assign an Id
5. Longitude: to assign a longitude
6. Latitude: to assign a latitude
7. Elevation: to assign an elevation
8. Radius: to assign a radius for the sensor node (this is not the radio radius)
9. Sensor Radius: to assign a radius for the sensing unit
10. Energy max: the initial energy of the battery
11. Sensing consumption: the sensing consumption in units (it is not considered in this version of CupCarbon)
12. UART Datarate: the UART datarate which represents the necessary time to send data (bytes) to the buffer of the radio module.
13. Drift (sigma): the clock drift.
14. The coverage of a sensing unit (case of directional sensor node)
15. The direction (rotation) of a sensing unit (case of directional sensor node)

Any modification is considered only if it is followed by a click on the apply button with right gray arrow situated in the right part of the corresponding field.

Radio Parameters panel



This panel allows to modify the parameters of the radio module of the selected sensor nodes, like:

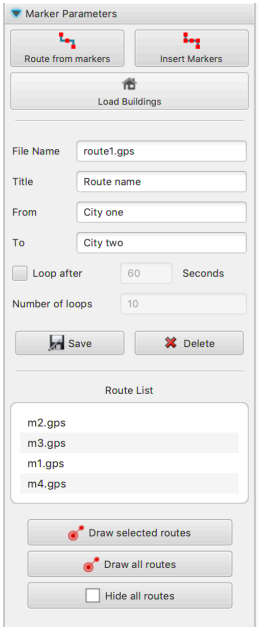
1. Standard: The standard of the radio module (ZigBee 802.15.4, Wifi and Lora)
2. Radio name: The name of the radio module to add to the sensor node. By default, one radio module ZigBee (802.15.4) is added automatically for each sensor node with the name radio1. This radio module can be changed and removed.
3. Add: to add another radio module.
4. Remove: to remove the selected radio module.
5. Current: to determine which radio module of the list is the current one, used by the sensor node to communicate. This part is used by the SenScript during simulation. The current radio module can be changed in the SenScript using the commande radio.
6. Radio Module List: the list of added radio modules
7. Network ID: the network ID of the selected radio module
8. MY: the MY address of the selected radio module
9. CH: the channel of the selected radio module
10. Radius: the radius range of the selected radio module. In the propagation mode (when clicking on the icon of the state bar) is activated than this radius is calculated automatically depending on the signal propagation and the environment.
11. E_Tx: the sending energy consumption [12]
12. E_Rx: the receiving energy consumption [12]
13. Sleeping Energy: the sleeping energy consumption (not available in the current version)
14. Listening Energy: the listening energy consumtion (not available in the current version)
15. Data Rate: the sending/receiving datarate (default: 250k bits/s)
16. Spreading Factor: for LoRa radio modules only
17. The consumption model where we can choose the model of the consumption for the transmissions (Tx) and the receptions (Rx) for each radio module. Two models are integrated, the classical one ant he model of Heinzelman. It is possible to write a personal model as any equation with the following predifined variables:

a. n → the number of transmitted/received bits
b. r → the radio range during the transmission
c. p → the pecentage of the power of the transmission (equal to 0 for 0% and 1 for 100%). It is fixed in the field PL of the same menu and it can be modified dynamically durint the simulation using the SenScript command **atpl**.
d. t → the temperature of the weather
e. etx → the energy consumption for each sent bit. It is fixed in the field E_Tx of the same menu.
f. erx → the energy consumption for each received bit. It is fixed in the field E_Rx of the same menu.

The classical model can be written as follows :

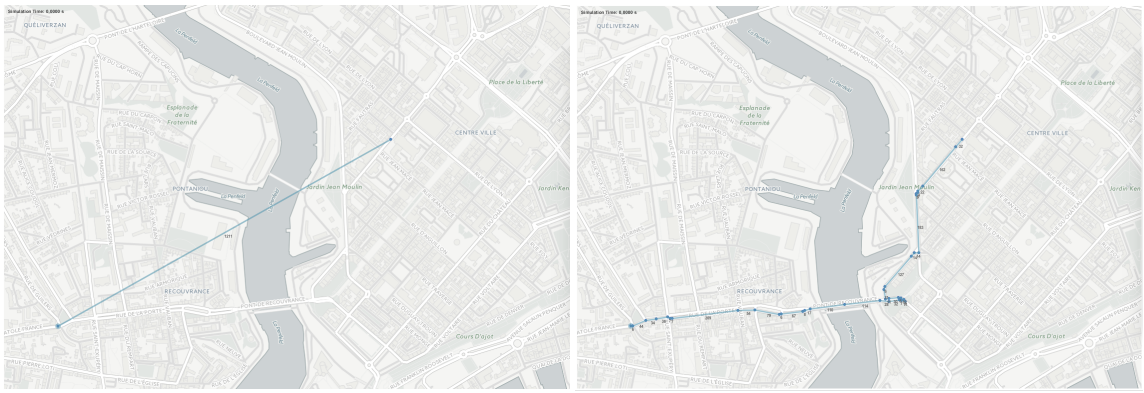
→ Tx: (n/8.0)*etx*p
→ Rx: (n/8.0)*erx

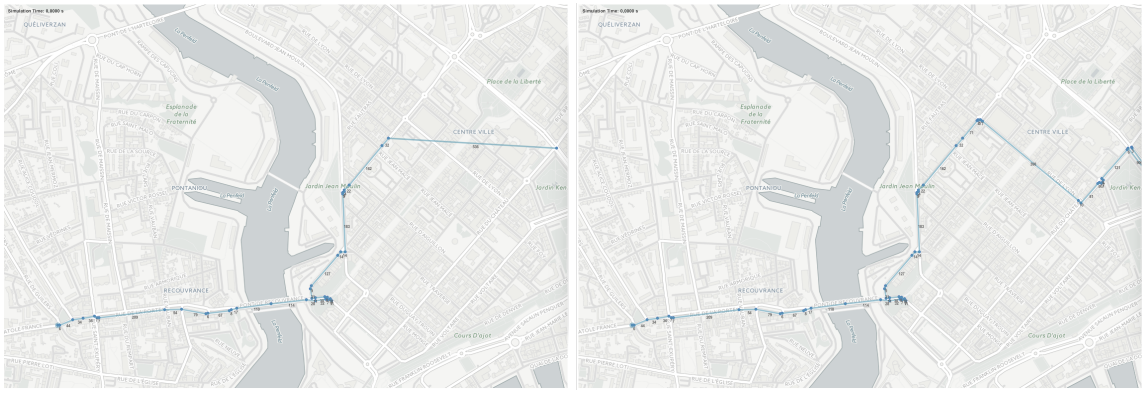
Marker Parameters panel



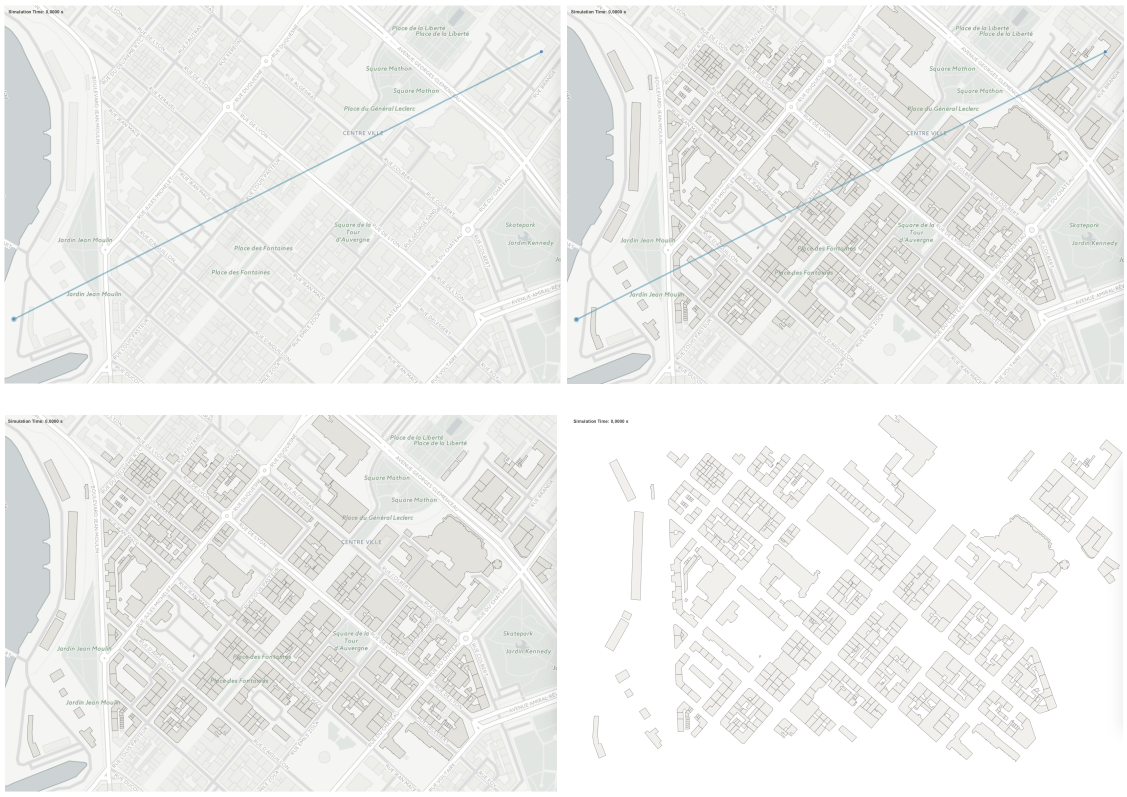
This panel allows to work with the markers as follows:

1. Route from markers: this button allows to generate a route situated between two points of the map that are determined by two markers. If there is more than 2 markers only the last two ones will be considered. This allows to draw new routes as a continuing route of the existing one.

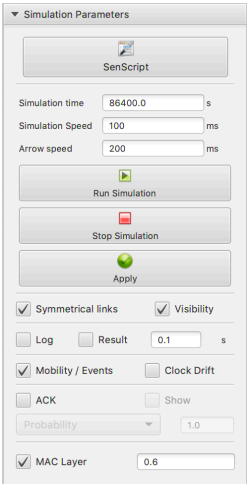




2. Insert Markers: allwos to insert markers after the selected ones (cf. Section Markers). The same result can be obtained by pressing on the key ‘u’.
3. Load buildings: allows to draw the polygons corresponding to the buildings that are situated on the rectangle delimited by two markes:



4. File name: the name given to the saved file of the corresponding route. Other informations (Title, From, To) are just for information. This route is used for the mobiles. It will be selected in the GPS file name of the view Device Parameters presented above. If the mobile must do many rounds, the check box **Loop after** must be activated. The number of loops must be given as well as the waiting time before starting the new round.
5. Save: this button allows to save the current route drawn on the map.
6. Delete : this button allows to delete the file corresponding the the selected route.
7. Selecting any route in the list of routes allows to draw the corresponding route on the map.
8. Draw all routes or a selected ones: to display all or a selected routes.
9. Hide all routes: to hide all the created routes.

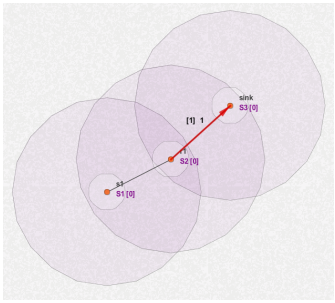


Simulation Parameters and SenScript Panel

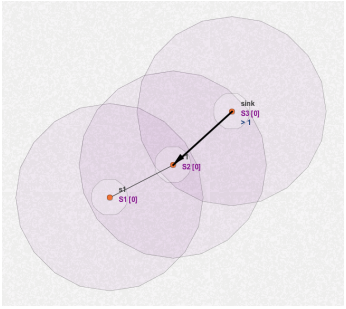
This panel is used for the simulation. It contains the following buttons and options:

1. SenScript: to open the SenScript window
2. Simulation time: the duration of the simulation
3. Simulation Speed: is the speed of the simulation. The objective of this button is to be able to follow and to visualize the simulation at the human speed. It is useful for debugging.
4. Arrow speed: the same as the Speed function where the delay in this option is related to the sending/receiving message. It allows to visualize the messages.
5. Run simulation: start the simulation (can be done also by pressing on Entree)
6. Stop Simulation: stop the simulation
7. Symmetrical links: this option forces the symmetrical links between sensor nodes event on the map this link is not symmetrical.
8. Visibility: it allows to recalculate the visibility of the mobile sensor nodes.
9. Log: generating the log file of the simulation

10. Results: if the objective of the simulation is the visualization then this option must be deactivated. Otherwise, it must be activated if the energy consumption must be saved and displayed. The period of saving results must be entered in seconds in the right field
11. Mobility/Events: this box must be activated if in the simulation the events and the mobility must be considered. Event here means the analog (Gas) events.
12. Clock Drift: activate this box if the clock drift must be taken into account in the simulation.
13. ACK (type): if ACK messages must be taken into account, activate this box. Then, three proposed algorithms to calculate the interferences are proposed. The first one is based on the simple calculation of the probability of passing or failing sending messages. The other ones are base on calculating the Gaussian of the alpha-stable distribution.
14. Show: if activated, the ACK messages will be shown during the simulation.
15. Activate or not the MAC Layer (it is a simplified version of the MAC Layer based on a probabilistic behavior)



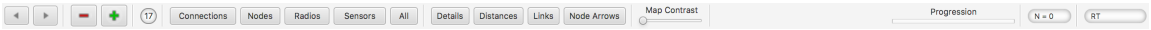
Sending a message



Sending an ACK message (Acknowledgment)

The state bar

The state bar is situated in the bottom of the main window of CupCarbon. It allows to display some information and it contains some button to do some option as in the case of the toolbar.

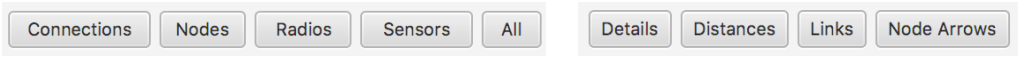


The different part of this bar will be presented in the following :

1. Showing/hiding the parameter panel in the left:

2. Zooming of the map: the zoom number is given inside the circle in the right.

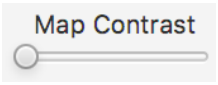
3. Nework visualisation options:

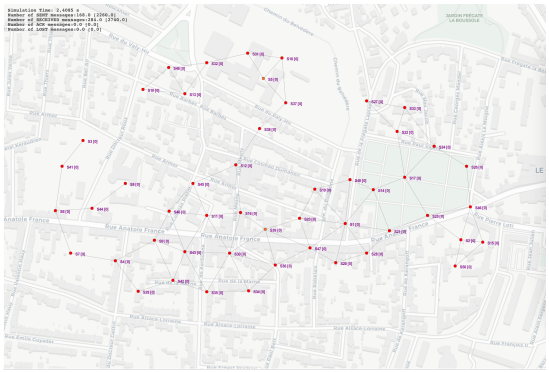


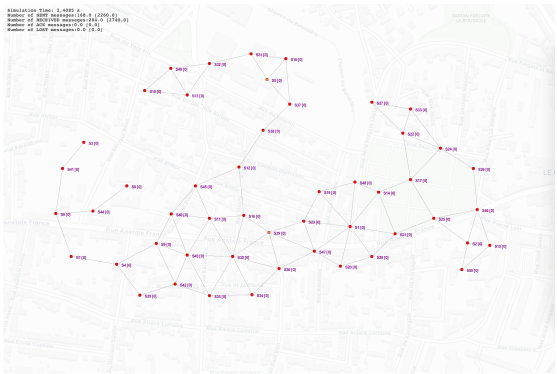
   a. Connections: shows only the sensor nodes with their connections
   b. Nodes: shows only the sensor nodes without connections
   c. Radios: shows only the sensor nodes with their radio range and without connections
   d. Sensors: shows only the sensor nodes with their sensor units and without connections
   e. All: shows all the parts of the sensor nodes
   f. Details: cf. Display/Hide Details
   g. Distances: cf. Display/Hide Radio Distances
   h. Links: cf. Display/Hide Radio Distances
   i. Nodes Arrows: cf. Display/Hide Network Arrows

These options can be combined between them. For example, in the case where we don't want to show the sensing units, Which means that we want to display sensor nodes with only their radio ranges and connections, then we click on the **Radios** button and then on the **Links** button.

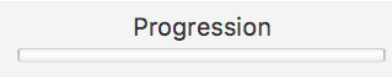
4. The contrast of the map: this slider is used to change the contrast of the map to improve the visibility of the network.



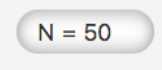




5. Simulation time progression: the following progression bar allows to visualize the progression simulation time or the progression of other operations.



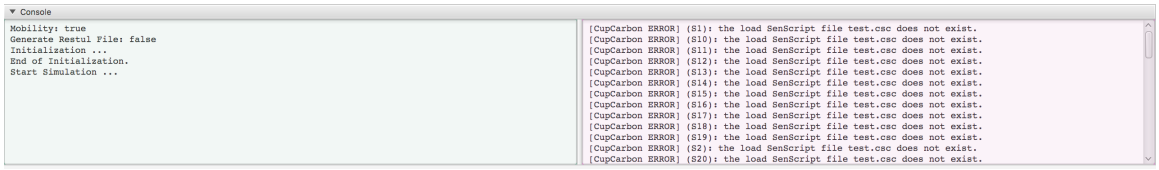
6. Number of sensor nodes on the map: the follwoing text view the number of the sensor nodes added on the map.



7. The real time of the simulation (RT): once the simulation finished, the following text view shows its real duration in seconds.

The Console

The console is used by the simulator to display some messages useful for the user during the simulation. It has two parts. The first one in the left is used to display messages about the simulation. It is possible also to display messages of the sensors using the SenScript command **`cprint`**. The second one in the right is used to display errors during the simulation. In the new versions of CupCarbon, this window of the console is separated from the main interface.



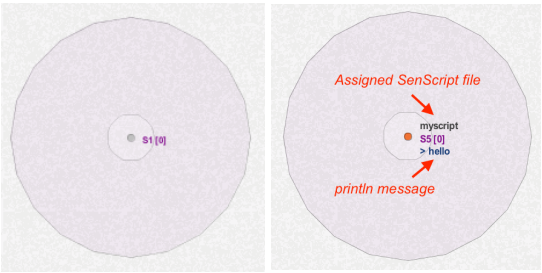
CupCarbon Objects

This section presents all the possible objects that can be added on the map and how to manipulate it.

Sensor Node

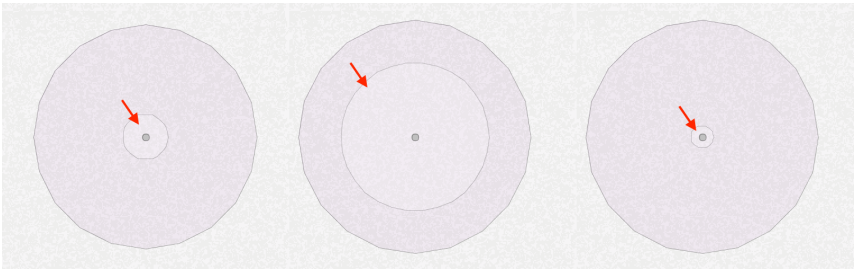
The sensor node is the main object of CupCarbon ! It contains four main parts : radio modules, a sensing unit and a battery. A sensor node can be added by clicking on the menu Add → Add Sensor Node and then by clicking on the map, in the place where the sensor node must be added. Another consecutive click of the mouse will add another sensor node, and so on. To stop adding sensor nodes, just click on the right button of the mouse, or by typing the escape [esc] button of the keyboard. We can also click on the button of the toolbar. Another manner to add sensor nodes is by typing on the key 1 of the keyboard and then click on the mouse as explained before. We can also click on the button of the toolbar or use the markers (cf. Marker section).

In the center of the sensor node, we find the name S followed by its id. For example, if its id is equal to 4, then its name will be S4. In the right part of the name, we find a number situated between brackets which is equal to [0] by default. This number represents the MY address of this sensor node. If a SenScript is assigned to it then it will be displayed in a gray color above its name. The *print* messages will be displayed in blue bellow its name.



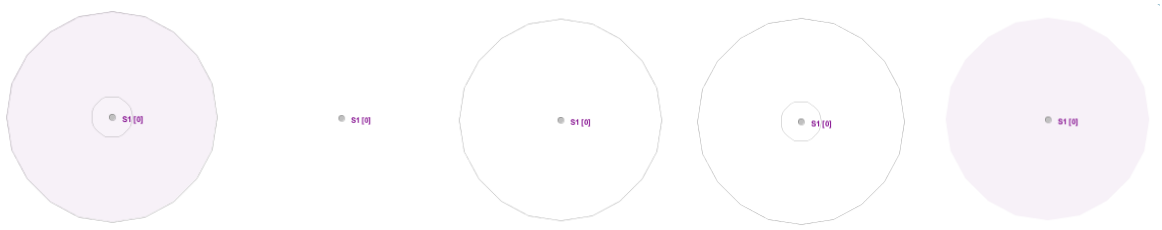
A sensor node can contain many radio modules with different or the same standards (802.15.4, WiFi or LoRa). However, for the communication, only one radio module is considered. To change the considered radio module, we select the name of this radio on the Radio Module List of the **Radio Parameters** view situated in the left of the CupCarbon environment (4). Then, one click on Current. The radio range circle displayed in the purple color represents the radio range of the current radio. This color changes to blue for the LoRa standard and to green for the WiFi standard. One can change the radio range using the keys '+' and '–' of the keyboard. If the propagation mode is activated by the button then the radio range is calculated according to the radio propagation and the environment. The current radio can be changed during the simulation by using the command **radio** in the SenScript file (cf. SenScript user reference).

A sensor node contains a sensing unit represented by transparent white circle. The radius of the area can be changed using the buttons ')' for increasing the radius and '(' for decreasing the radius.



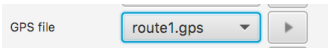
Other keys of the keyboard can be used to modify some parameters of the sensor node, like, ‘d’ to show/hide the name of the sensor node, ‘D’ to show/hide the message displayed by the sensor node, ‘n’ to show/hide the assigned SenScript file name. The key ‘b’ show/hide the battery/buffer levels. The key ‘h’ allows to hide some parts of the sensor node. By typing many times we hide in each time the following parts :

1. First time: all the parts except the center
2. Second time: shows only the radio range circle of the current radio module
3. Third time: shows only the radio range of the current radio module and the sensing unit
4. Fourth time: shows only the sensing unit
5. Fifth time: shows only the radio range area of the current radio module
6. Sixth time: coming back to the initial drawing of the sensor node (with all the parts)

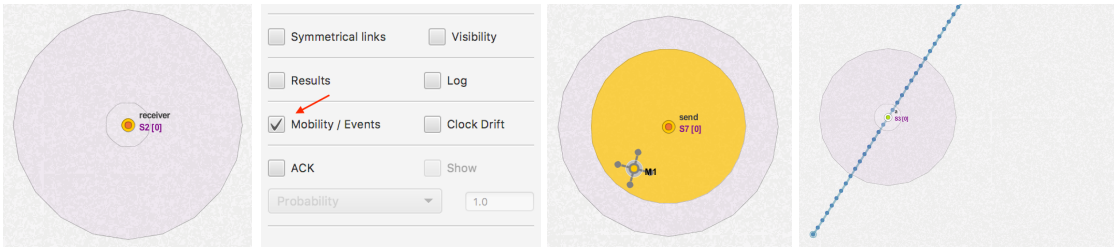


1 2 3 4 5

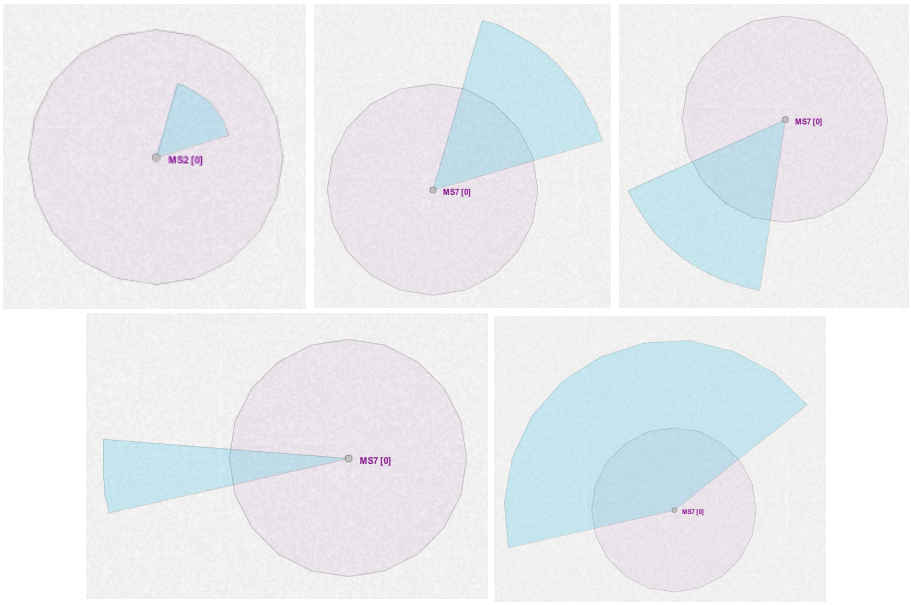
The key ‘j’ allows to come back directly to the initial drawing of the sensor node. The key ‘r’ allows to show directly on the sensor node some parameters (name, id, my address, network address, channel, SenScript file name, route file name, and the battery level). If the sensor node is mobile, it is possible to test its mobility by clicking on the key ‘s’. To stop the mobility, click on ‘q’. To duplicate a sensor node, use the key ‘c’. Finally, to kill a sensor node, use the key ‘k’. Clicking a second time on ‘k’ will lead to the initial sensor node. A sensor node can be mobile. The route which is followed by a mobile sensor node is chosen in the list of created routes from the field GPS file in the Device Parameters view (CupCarbon environment 4).



Once the route is assigned, the center of the sensor node will be surrounded in yellow. During the simulation process, the center of the mobile sensor nodes will be colored green. The section **Markers** presented bellow explains how to create routes. To take into accound the mobility during the simulation process, the (Mobility/Event) box must be activated in the Simulation Parameters view. If any event is detected then the sensing unig area will be colored yellow. If a script is assigned then the center will be colored orange. It can also be colored red if it is waiting for a message (using the command **wait**). A sensor node can be marked. In this case, a flashy green disc will be added around the center.



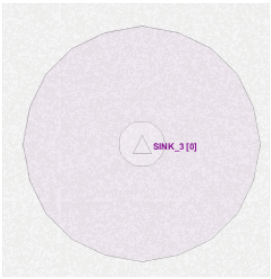
Directional Sensor Node



The Directional Sensor Node is the same as the classical sensor node with another type of sensing unit, which is directional. This last is not circular, it has a form of a cone that can be modified with the SenScript and manually using the following keys of the keyboard:

1. ‘)’ and ‘(‘: increase or decrease the sensing unit range (distance). Use ‘]’ and ‘[‘ for more precision.
2. ‘p’ and ‘o’: rotate (left or right) the sensing unit range. Click in addition on ALT for more precision.
3. ‘P’ and ‘O’ : increase or decrease the areas (angle) of the sensing unit. Click in addition on ALT for more precision.

Base Station (Sink)



The base station is exactly the same object than the sensor node with the exception that it has an infinite battery.

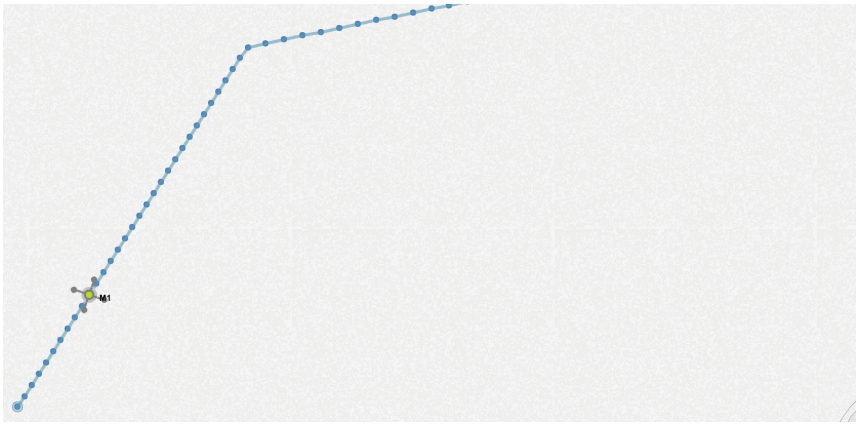
Gas (Analog events)

The Gas or the natural event allows to generate analog values (the velue is displayed in the red text). It can be mobile. Its objective is to simulate random or given values coming from the environment. It is possible to use the Natural Event Generator in order to generate random values based on the Gaussian distribution. To take into account the event during the simulation process, the (Mobility/Event) box must be activated in the Simulation Parameters panel. A red center means that the event is not mobile, otherwise it will be in orange color. The second white circle around becomes yellow when a natural event file is assigned. During the simulation, if the event is mobile, then the center becomes green.

Mobile

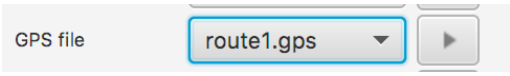
Route not assigned (gray center)

Route assigned (orange center)



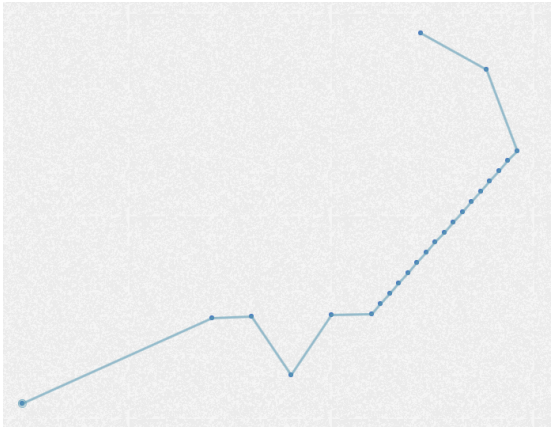
Mobile Simulation (green center)

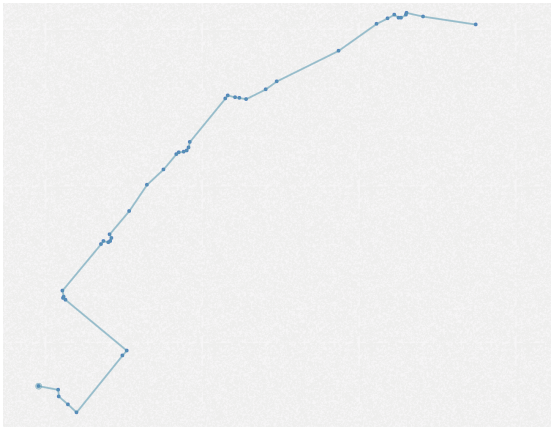
A mobile depends on its route (trajectory). This last one is created using markers as it is explained in the next section, bellow, one needs, just to select the route which will be followed by the mobile in the list of created routes from the field GPS file in the Device Parameters view (CupCarbon environment 4).



During the simulation, the mobile goes to each next point (destination) in each 1 second. To take into account the mobility during the simulation process, the (Mobility/Event) box must be activated in the Simulation Parameters view.

Marker





The markers can be used for many objectives, which are:

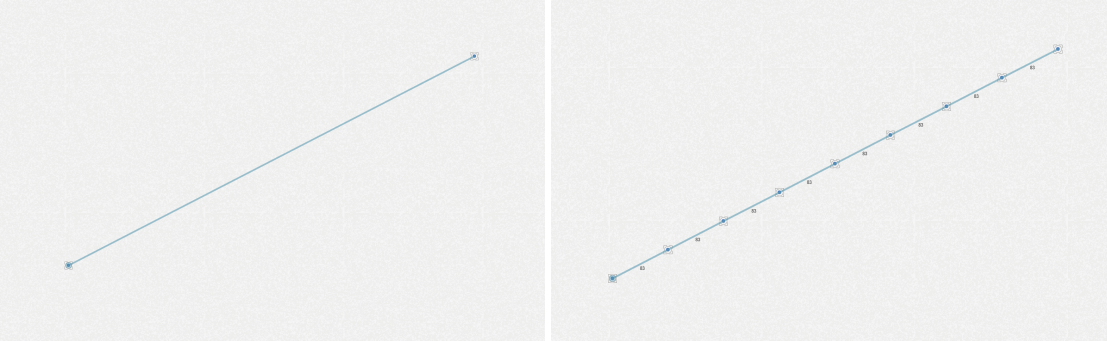
Add randomly sensor nodes

This part is explained above in Section “Add Menu” (point 7).

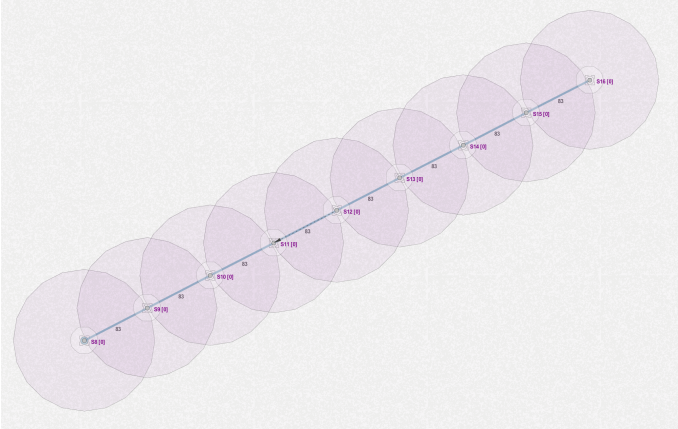
Add sensor nodes

It is possible to transform markers to sensor nodes. This can help to add sensor nodes separated with equivalent distances. This process is explained in the following:

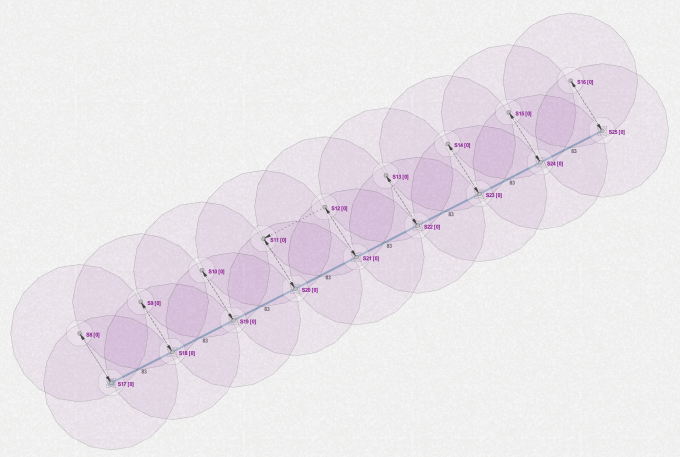
1. Add 2 markers. Then, select all these markers (Select all markers) and press on the key ‘u’ to insert intermediate markers. This action can be repeated many times until one obtains the desired number of markers. The markers can be inserted only after the selected ones. The key ‘U’ can be used to select the markers situated after the selected ones. To display the distances between markers, press the key ‘X’ (capital X)



2. Select all markers, again and type on the key ‘t’ to transform all the selected markers to sensor nodes:



3. It is possible to move directly the markers, already selected, to add other sensor nodes.



Generating routes

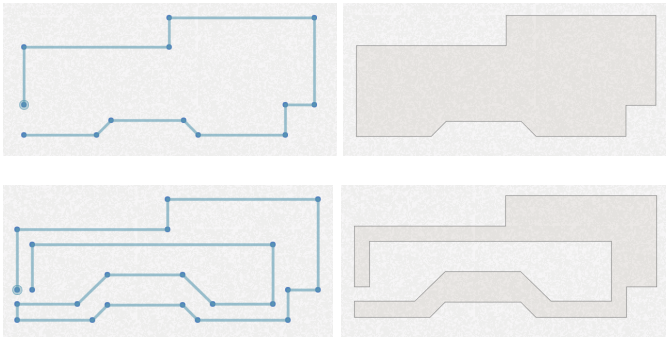
It is possible to generate routes either by drawing the route manually using the markers or by creating just two markers and click on the button Route From Markers in the Marker Parameters view (CupCarbon environment 4). Then, each created route can then be saved and added to the list of the routes of the project. This procedure is explained above in the Marker Parameters view section.

Adding buildings

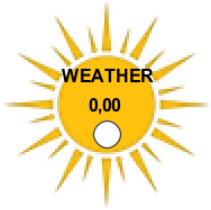
A list of buildings can be added in the area delimited by two markers as is explained above in the Marker Parameters view section.

Drawing buildings

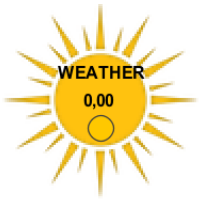
It is possible to draw a building by drawing the form using markers and then by typing on the key ':'.



Weather



Weather File non-assigned



Weather File assigned

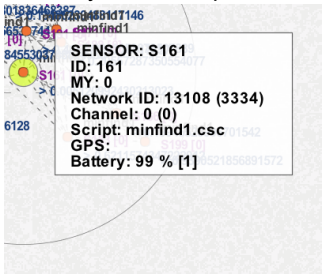
The weather module is used to generate any weather parameter and especially the temperature. Only one weather module can be added to the project. In this version, the weather is used to simulate the variations of the environment temperature, which can be considered in the models of the energy consumption of the radio modules. The temperatures can be generated using the Natural Event generator presented above and using the following button:

Network keyboard and mouse events

Keyboard events

1. Key 'a': show/hide the arrows of the links between communicating sensor nodes
2. Key 'A': show/hide the arrows of the links between markers
3. Key 'x': show/hide distances in the links between communicating sensor nodes
4. Key 'X': show/hide distances in the links between markers
5. Key 'w': select all sensor nodes → select all markers → deselect all
6. Key 'l': show/hide links between communicating sensor nodes
7. Key 'v': change the colors of the radio links
8. Key 'd': show/hide the name of the objects and the messages displayed by the **print** command
9. Key 'n': show/hide the file name of the objects

10. Key ‘r’: show/hide some parameters directly on the map



11. Key ‘R’: show/hide all the routes
12. Direction keys: move the map
13. Deleted key: delete objects
14. esc key: deselect all.

Mouse events

1. Double click in a selected sensor node: open the Devices Parameters view → open the Radio Parameters view
2. Right click on the map: deselect all and stop the process of adding objects
3. Click on the map: deselect all or add objects
4. Drag with the right button on the map: draw the rectangle of selecting many objects
5. Combining the shift key with dragging the mouse will do the same action as dragging the right button (multiple selection)
6. Combining the CTRL key with clicking on different objects will lead to multiple selections of objects.

SenScript

SenScript is the script used to program sensor nodes of the CupCarbon simulator. It is a script where variables are not declared and without types, but they can be initialized (**set** command). It is possible to use the instruction **function** to add complex and additional functions programmed in Java (in a source code mode only).

Example:

SenScript: **set x "abcd"** → Java: **x = "abcd";**
SenScript: **set y x** → Java: **y = x;**
SenScript: **set z x+y** → Java: **z = x+y;**

List of Commands

Language	Sensor	Radio Module	Embedded Card	Messages Data	Mobile	Visualization
inc / dec	areadsensor	atch	loop	data	move	Simulation
conc	dreadsensor	atid	stop	rdata	rmove	cprint
hash	getpos	atpl	buffer	vdata	route	edge
int	getpos2	atmy	cbuffer	send		
max / min	getinfo	atnd	delay	receive		
smax / smin	rotate	atnid	led	wait		
and / or		atget	mark	read		
xor / not		drssi	print	pick		
band / bor		distance	printfile			
bxor / bnot			battery			
for end			time			
if [else] end			kill			
while end			script			
goto			rscript			
function						
math						
rand / randb						
rgauss						
set						
tab / vec						
tget / tset						
vget / vset						
angle / angle2						
sadd / spop / nth						
charat / length						

and (logic)

and x a b

→ x = a & b

angle / angle2

angle a x1#y1 x2#y2 x3#y3

→ a = angle formed by the edge {(x2, y2),(x1,y1)} and {(x2, y2),(x3,y3)}

angle2 a x1 y1 x2 y2 x3 y3

→ a = angle formed by the edge {(x2, y2),(x1,y1)} and {(x2, y2),(x3,y3)}

areadsensor (Analog Read Sensor)

areadsensor s

→ s=X or s=S#y#v, where X means that there is no sensed value (or event), S means that a sensed value has been read from the event having id=y, where the read value is equal to v (use the function rdata function to read these values). If there is many events then s=S#s_1#v1#s_2#v2#S_3#v3… (use rdata function to read these values). Note that, s_2, s_3, … can be null if there is no many events.

atch

atch 12

→ Change the channel of the current sensor to 12

atid

```
atid 1111
```

→ Change the identifier of the sensor to 1111

atpl

```
atpl 60
```

→ Change the signal power (power level) of the radio of the sensor. In this example, the sensor will use only 60% of its radio range.

atmy

```
atmy 4
```

→ Change the my of the sensor to 4.

atnd

```
atnd n
```

→ n = the number of the neighbors of the sensor.

```
atnd n v
```

→ n = the number of the neighbors of the sensor and v the vector of the identifiers of each neighbor (use vget to get the values of v)

atnid

```
atnid 2222
```

→ Change the network address

atget

```
atget id x
atget my x
atget ch x
```

→ x = id, my or ch of the sensor node

band

```
band x 1010 1100
```

→ x = 1000

battery

```
battery x
```

→ x = the level of the battery in Joules

```
battery set x
```

→ The level of the battery will be set to x

bnot

```
band x 1010
```

→ x = 0101

bor

```
bor x 1010 1100
```

→ x = 1110

buffer

```
buffer x
```

→ x = the size of the buffer in bits

bxor

```
band x 1010 1100
```

→ x = 0110

cbuffer

```
cbuffer
```

→ clear the buffer

charat
```
charat c hello 1
```
→ c = ‘e’
the character situated in the index 1 of hello

conc
```
conc x a b
```
→ concatenate a and b and assign it to x (=ab). In java this command is written as `x = “a”+”b”;`

```
conc x a b
```
→ concatenate the value of the variable a with the value of the variable b

cprint
```
cprint "hello" x
```
→ The same function as print where the result is displayed in the console

data
```
data p 1 4 6
```
→ p = 1#4#6

dec
```
dec x
```
→ x = x - 1

delay
```
delay 1000
```
→ wait 1 second (1000 ms) before the next instruction

distance
```
distance x 2
```
→ x = the Euclidean distance between the current sensor and a communicating sensor node having an id=2.

dreadsensor
```
dreadsensor x
```
→ x=1 if the sensor detects an event (mobile), x=0, otherwise

drssi
```
drssi x
```
→ x = the rssi value transformed to a Euclidean distance between the current sensor the last sending sensor node.

egde
```
edge a x
```
→ to mark (if a=1) or unmark (if a=0) the edge (communication link) between the current sensor node and the node having the id x.

for end
```
for i 0 10
print "i = " i
delay 1000
end
```
→ the same as the following for on C: for(int i=0; i<10; i+=1)

```
for i 0 10 2
print "i = " i
delay 1000
end
```
→ the same as the following for on C: for(int i=0; i<10; i+=2)

function
The function command can be used only with the source code of CupCarbon.

```
function x myf 1,2,v
```
→ execute the function myf with the arguments 1, 2 and the value of v. The obtained result is returned in x.

To create your own function (example: myf(args)):

1. Add the following script in the method function of the class ScriptFunction (package script_functions):

```
if(function.equals("myf")) {
return Function_Calc.myf(args);
}
```

2. Add the methode myf the class Functions as follows:

```
public static String myf(String [] args) {
    String valToReturn = "";
    // TODO
    // Your program here
    return valToReturn;
}
```

getinfo

```
getinfo p
```
→ if a mobile is detected by the sensing unit, it returns in p information formed by the id and the coordinates of the mobile in a format: id#longitude#latitude.

getpos

```
getpos pos
```
→ pos = the position of the sensor longitude#latitude

→ pos = "-5.489438533782959#48.390415333407574"; // example

```
getpos pos
rdata pos x y
```
→ x = longitude; // example: -5.489438533782959

→ y = latitude; // example: 48.390415333407574

getpos2

```
getpos2 x y
```
→ x = longitude and y = latitude
→ x = -5.489438533782959; and y = 48.390415333407574; // example

goto

```
goto 5
```
→ Go to to the line of the code number 5

```
goto B
```
→ Go to to the line labeled by B

To label a line by B:

```
B:set x 12
```

hash

```
hash x hello
```
→ assign to x the hash code of hello

if [else] end

```
if (x==1)
    mark 1
else
    mark 0
end

if ((x==1) && ((y>0) || (y<5)))
    mark 1
```

```
else                                    end
    mark 0
```

- In the current version of CupCarbon, it is not possible to write: `if (x==1 && y>0) ...`
- One must write: `if ((x==1) && (y>0)) ...`

inc

```
inc x
```

→ x = x + 1

int

```
int x a
```

→ assign to x the integer value of a (if a=3.2 then x=3)

kill

```
kill 0.3
```

→ the current node will be killed (i.e., with empty battery) with a probability of 30%

led

```
led 1 2
```

→ set the color number 2 to the sensor (1 is not important here, it represents the pin number in the read card)

length

```
length v hello
```

→ v = 5
the length of the string “hello”

loop

```
loop
```

→ Starting the loop section

mark

```
mark 1
```

→ Mark or unmark (`mark 0`) the sensor

math

```
math f y x
```

→ y = f(x) where f = "sqrt, sin, cos, tan, asin, acos, atan, abs, pow, log, log10, exp".

Example:

```
math sin y 3.14
```

→ y=sin(3.14)

max

```
max x 4 3
```

→ assign to x the maximum value between 4 and 3 (4 and 3 can be replaced with variables)

min

```
min x 4 3
```

→ assign to x the minimum value between 4 and 3 (4 and 3 can be replaced with variables)

move

```
move x y z t
```

→ The sensor node will go to the position latitude (x), longitude (y), and elevation (z) with the speed t meters/second.

not (logic)

```
not x a
```

→ x = ~a

nth
```
nth v i t
```
→ v will be equal to the i^{th} value of t (t is in a format of a string with the separator &)
→ Example: if t=4&9&2&7 then nth v 1 t will put in v the value 9 (v=9) and t remains the same.

or (logic)
```
or x a b
```
→ x = a | b

pick
```
pick x
```
→ same as read x without removing the read data from the buffer

printfile
```
printfile "hello"
```
→ add the string “hello” in a file generated during the simulation in the directory *results* with the name of the sensor node

print
```
print "hello"
```
→ display hello

```
print x
```
→ display the value of x

```
print "i = " i
```
→ display "i = (value of i)"

```
print ""
```
→ display nothing

radio
```
radio [name of a radio module]
radio radio2
```
→ Select the radio module having the name radio2. This means that any **send** command that will be used after this instruction will consider the radio2 as the radio module which will send the message. The considered standard is the one of the considered radio module.

rand
```
rand [variable]
rand x
```
→ x = rand

randb
```
reandb [varialbe][inf][sup]
reandb x 2 6
```
→ x = random values between 2 (included) and 6 (included)

rdata
```
rdata p a b
```
→ We assume that p is a message formed using the command **data**. It is possible to form manually p which is a string with # separator (example: p=hello#4). In this example, **rdata p a b** will lead to an a="hello” and b=4

read
```
read x
```
→ Assign the value of the buffer to x

receive
```
receive x
```

→ Wait until receiving data in the buffer and assign it to x. This is a blocking function, if there is not data in the buffer then it remains blocked on this instruction.

```
receive x t
```
→ Wait until receiving data in the buffer. If there is no data received after t milliseconds then x will receive an empty message (x="") and the execution of the script will be continued.

rgauss
```
rgauss x
```
→ x = Standard Gaussian random value

rmove
```
rmove t
```
→ Go to the next point of the route created by the markers. A route must be assigned before simulation to the sensor node. The next instruction of the script will be executed after t milliseconds. Note that the mobility check box in the simulation parameters view must be activated.

rotate
```
rotate x t
```
→ Rotate with x (double) units the Directional sensor node. The next instruction of the script will be executed after t milliseconds.

route
```
route route1
```
→ The sensor node will change its route to route having the name route1 after intersecting with it.

rscript
```
rscript flooding
```
→ Load the script floodint.csc while reinitializing the variables of the environment (see script function)

sadd
```
sadd x t
```
→ add the value of x to the stack t which is in a format of string separated by the symbol &. You can use the command spop to get the first element of the stack t.

→ Example: if t = 4&6&7, sadd 56 t will lead to: t = 56&4&6&7

script
```
script flooding
```
→ Load the script flooding.csc without reinitializing the variables of the environment (see rscript function)

send
```
send hello 2
```
→ Sends hello to the sensor having an id=2

```
send p 2
```
→ Sends the value of p to the sensor having an id=2

```
send p
```
→ Sends the value of p in a broadcast mode

```
send p *
```
→ Sends the value of p in a broadcast mode(the same as send p)

```
send p * 3
```
→ Sends the value of p in broadcast except the sensor having an id=3

```
send p 0 4
```
→ Sends the value of p to sensors having a MY address equal to 4

```
send p 0 0 5
```

→ Direct sending of the value p to sensors having an id equal to 5 (like in a GPRS/3G/4G mode)

```
send !color 1
```
→ Change the color of the send link on the IHM. It is practical to differentiate the type of the communication links(of the send instruction) between sensors.

set

```
set x 5
```
→ x = 5

```
set s hello
```
→ x = "hello";

```
set x ""
```
→ x = ""

```
set z x*5
set z (x*y)/3
set z x%y
set z (x+y)%2
```

simulation

```
simulation 50 200
simulation sspeed 50
simulation aspeed 200
```
→ just for the visualization and it is not considered in the simulation process. It allows to change the simulation speed during the simulation process.

smin

```
smin m x y
```
→ x and y must be in a format a#b. If x=a#b and y=c#d then m=x if b<d or m=y if d<=b

smax

```
smax m x y
```
→ x and y must be in a format a#b. If x=a#b and y=c#d then m=x if b>d or m=y if d>=b

spop

```
spop v t
```
→ retrieve the first value from the stack t (in a format of a string with the separator &) and put it in the variable v.
→ Example: if t = 4&6&7, sad v t will put in v the value 4 and t becomes t = 6&7

stop

```
stop
```
→ Stop the execution of the script

tab

```
tab t 2 5
```
→ create a table t with 2 rows and 5 columns (t[2][5])

tget

```
tget x t 0 2
```
→ x = t[0][2]

time

```
time x
```
→ assign to x the current simulation time

tset

```
tset 55 t 0 2
```
→ t[0][2] = 55

vdata

```
vdata v 14#54#2
```
Transform tokens that are separated with # to a vector, with the name v, of these tokens.
→ v[0]=14, v[1]=54 and v[2]=2

vec

```
vec v 5
```

→ create a vector v with 5 elements (t[5])

vget / vset

```
vget x v 2
```

→ x = v[2]

```
vset 55 v 1
```

→ v[1] = 55

wait

```
wait (deprecated → use receive)
```

→ Wait until receiving data in the buffer

```
wait 1000 (deprecated → use receive)
```

→ Wait until receiving data in the buffer. If there is no data received after 1 second (=1000 milliseconds) then the execution of the script will be continued.

while end

```
while (1)
```

→ The same as "while (true)" in C.

Example:

```
set i 0
while ((i<5) && (i>=0))
    print "i = " i
    delay 1000
    set i i+1              // → inc i
end
```

xor (logic)

```
xor x a b      →  x = a ^ b
```

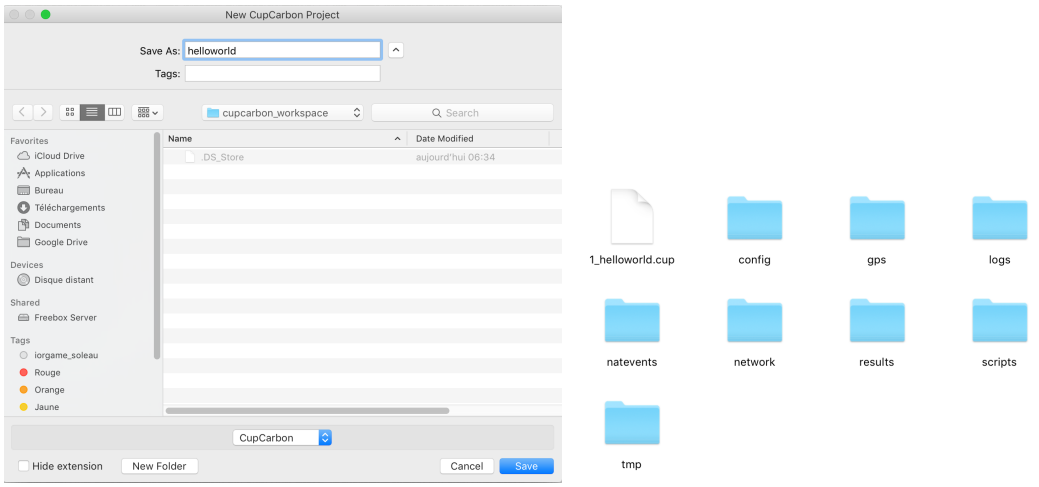
Basic examples

In this section we will present some basic examples that can help to understand how to use the environment of CupCarbon and also how to simulate different scenarios. The different examples can be downloaded from the official web site of CupCarbon (www.cupcarbon.com) and they are included in the source code and the executable.

Example 1: Hello World

This example shows how a sensor can display a message “Hello World”. The following steps show how to do this:

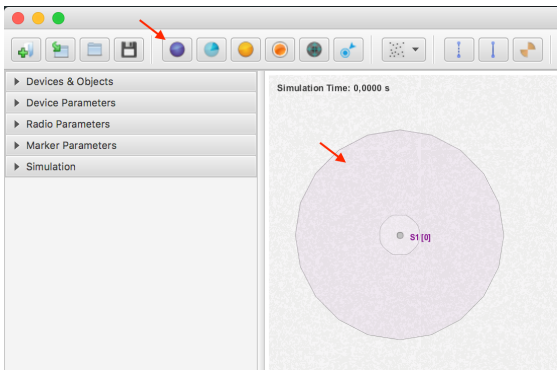
Step 1. Create a new project: this can be done either by clicking on the “New project” icon of the toolbar or on the menu Project → New project. Choose the name (example: helloworld) and the place where you want to save your project. The project will create a new folder with the given project name.

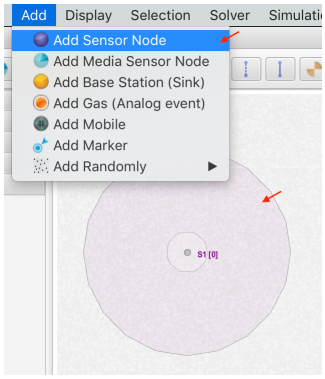


Inside this folder, 1 file (helloworld.cup) and 8 other directories will be created. The content of each directory is given in the following:

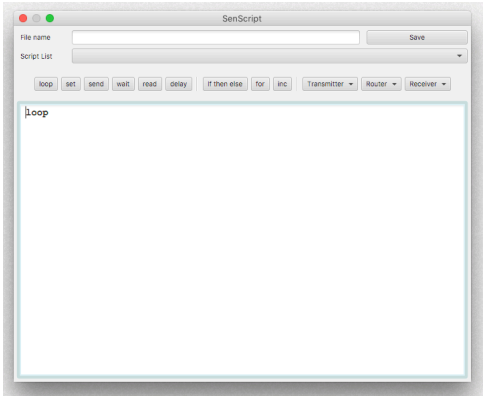
a. config: it contains the simulation parameters file, the building list file, the marker list file and two other directories (sensor and sensor_radios) that contains the list of sensor nodes (one file by sensor node) and the list of the radio modules of each sensor.
b. gps: it contains the list of routes
c. logs: it contains the log file
d. results: it contains the simulation results (a csv file)
e. scripts: it contains the SenScript files of the project
f. netevents: it contains the natural event files
g. tmp, network: are used by the simulator

Step 2. Add a new sensor node on the map: click either on the Add Sensor icon of the toolbar or from the menu bar (Add → Add Sensor Node). Then, click on the map where you want to add the sensor node. Another click will lead to another new sensor node and so on. To stop adding sensor nodes, just click on the right button of the mouse. You can also click on the icon of the toolbar, or by typing on the escape [esc] button of the keyboard.





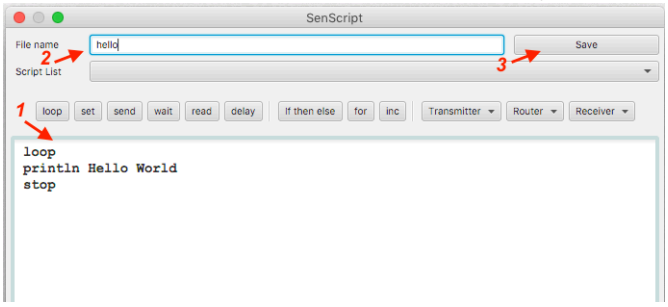
Step 3. Open the SenScript Window: the SenScript window can be opened by clicking on the icon  of the toolbar or from the menu Simulation → SenScript Window.



Step 4. Write the script: add the following script (1) in the text area part of the SenScript window:

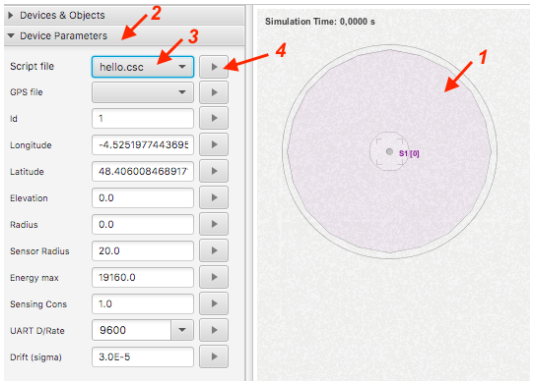
```
loop
print "Hello World"
stop
```

Add the name **hello** of this script in the File name field (2), then click on the Save button (3) just in the left part of this field. This will create a file **hello.csc** in the directory **scripts**.

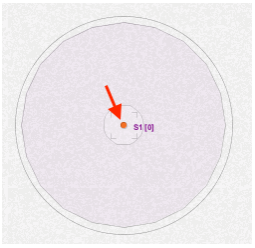


Finally, close the SenScript Window.

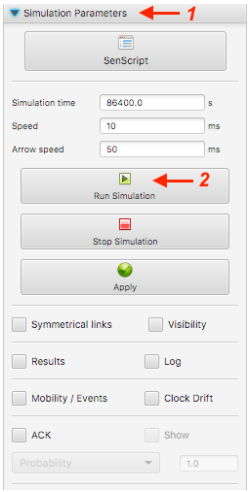
Step 5. Assign the SenScript file to the sensor node: Select the sensor node on the map (1). Go to Device Parameters in the left part of the main window (2). Then, select the hello.csc file in the field Script file (3). And then, click on the apply button just in the right (4).



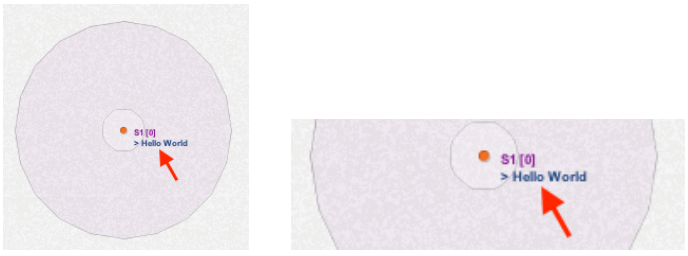
Note that once the script is assigned to a sensor, the center will be colored in orange. This can help to detect graphically sensors without scripts.



Step 6. Run the simulation: For this example, there is no need to parameterize the simulation, just click on the run simulation button in the toolbar or in the simulation parameters menu in the left.



Step 7. Simulation results: in this example the simulation results shows a *Hello World* message displayed by the sensor.



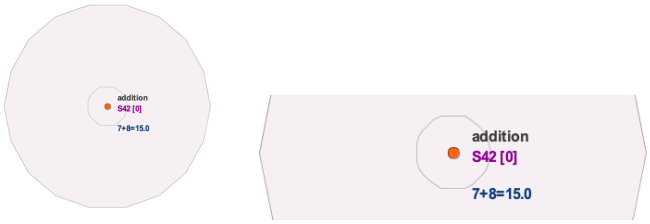
Step 8. Save the project: Click on the icon to save the project.

Example 2: Calculate a+b

This example shows how to calculate the sum of two variables a and b. Repeat all the steps of Example 1 where only the script must be changed as follows:

```
loop
set a 7
set b 8
set x a+b
print a "+" b "=" x
stop
```

The simulation result will display 7 + 8 = 15.



Test with the following script (with a simulation speed of 1000):


```
set a 7
loop
for b 0 11
    set x a*b
    print a " X " b " = " x
    delay 1000
end
```

Example 3: Calculate the sum of a vector

This example shows how to create tables and how to access their elements. As an example, we will try to create a table with 5 rows and 1 column. Then, we will display the sum of its elements.

Repeat all the steps of the Example 1 where only the script must be changed as follows:

```
tab t 5 1
tset 3 t 0 0
tset 5 t 1 0
tset 2 t 2 0
tset 6 t 3 0
tset 4 t 4 0
set s 0
```

```
loop
for i 0 5
  tget x t i 0
  set s s+x
end
print s
stop
```

The simulation result will display 20.

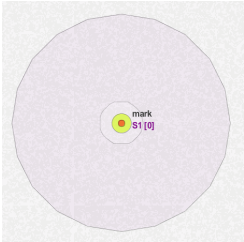
Example 4: Marking nodes

A marker node represents in the reality a sensor node with a switched on led. It helps to do a visible action rather than displaying messages. Marking a node is done by the SenScript command **mark 1**. Unmarking sensor node is done by the command **mark 0**.

To mark a node, use the same steps of Example 1 with the following script:

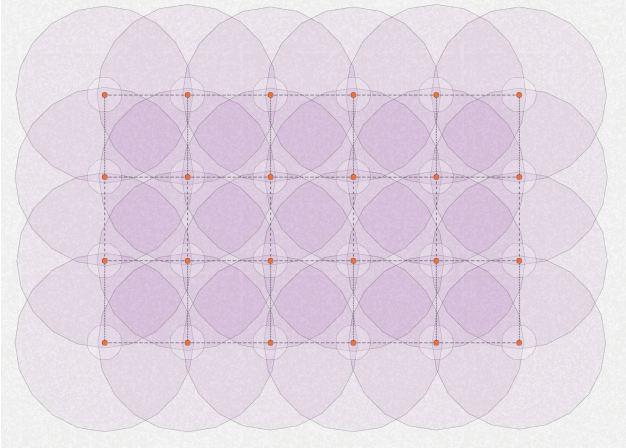
```
loop
mark 1
stop
```

The simulation result will shows a sensor marked with a yellow green center.



Example 5: Marking randomly (game of light)

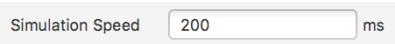
Add many sensor nodes on the map or generate a randomly network.



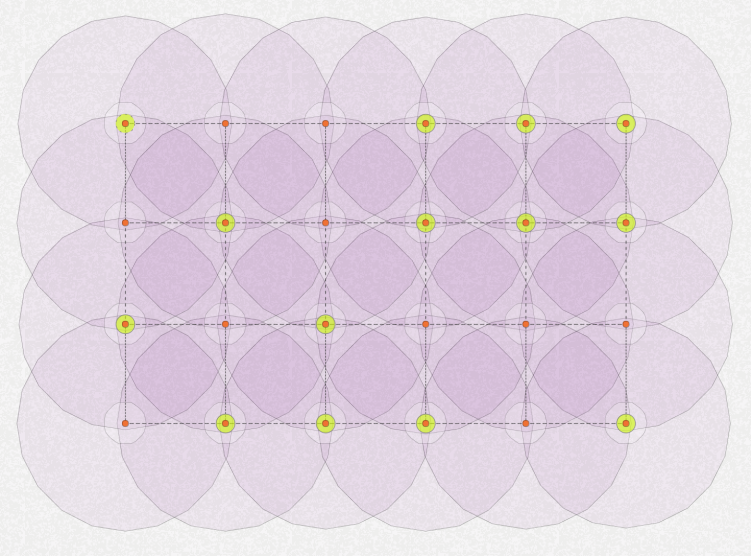
Then, use the same steps of Example 1, with the following script:

```
loop
rand x
if(x<0.5)
    mark 1
else
    mark 0
end
delay 1000
```

The simulation speed must be equal to 200:



The simulation results show marked and unmarked sensor nodes that change the marking state during the simulation.



In this example, the simulation will stop only if it reaches the simulation time. To stop manually the simulation, just click on the button .

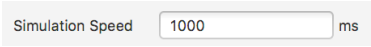
Example 6: Blinking and LEDs

Blinking

Use Example 3 with the following script:

```
loop
mark 1
delay 1000
mark 0
delay 1000
```

Simulate with the simulation speed equal to 1000.



LEDs

Use Example 3 with the following script:

```
loop
for i 0 15
    led 13 i
    delay 1000
end
```

Simulate with the simulation speed equal to 1000.

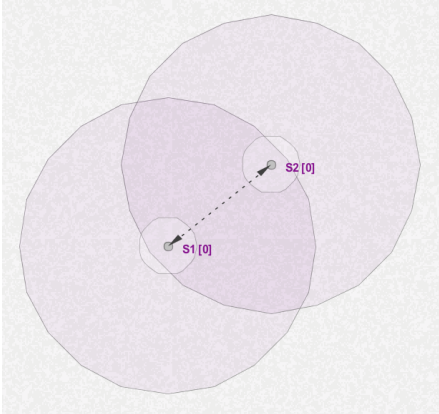
Note that the command **led 13 0** is the same as **mark 0** and the **command led 13 1** is the same as **mark 1**. The other colors are between the values 2 and 14. The number 13 means that this LED will be connected to the pin 13 of the considered embedded card. It is not considered in the simulation.

Example 7: Sending and Receiving messages

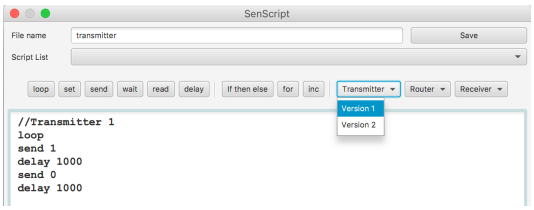
The example of sending and receiving messages is very important since the main actions of the simulation under CupCarbon are based on sending/receiving messages. The following example will show a sensor node that will send each second 0 and 1 to another sensor node. The receiver will be marked if it receives 1 (mark 1) and unmarked if it receives 0 (mark 0).

Step 1. Create a new project: this can be done either by clicking on the "New project" icon of the toolbar or on the menu Project → New project.

Step 2. Add two sensor nodes on the map: click either on the Add Sensor icon of the toolbar or from the menu bar (Add → Add Sensor Node). Then, click on the map where you want to add the sensor nodes so that they can communicate (i.e., There exists a link between the two sensor nodes).



Step 3. Write the SenScript of the transmitter (sensor node 1): The SenScript of the transmitter can be obtained directly from the SenScript window by clicking on the menu button Transmitter (Version 1) which must be completed by adding the id of the receiver (i.e., 2) in the command **send**. Save the file with the name **transmitter**.

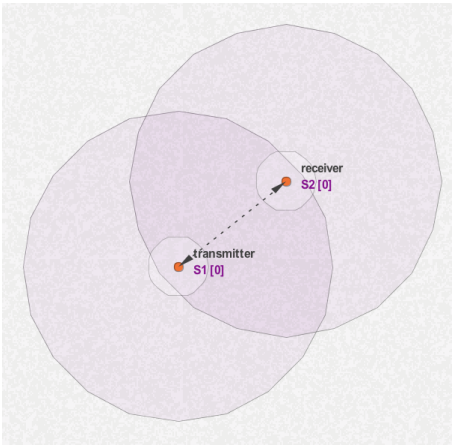



```
loop
send 1 2
delay 1000
send 0 2
delay 1000
```

Step 4. Write the SenScript of the receiver (sensor node 2): As for the first script, The SenScript of the receiver can be obtained directly from the SenScript window by clicking on the menu button Receiver (Version 1). Save the file with the name **receiver**.


```
loop
receive v
mark v
```

Step 5. Assign the SenScript file to the sensor nodes: Select the sensor node 1 on the map (1). Go to Device Parameters in the left part of the main window (2). Then, select the transmitter.csc file in the field Script file (3). And then, click on the apply button just in the right (4). Do the same procedure for the second sensor node by choosing the SenScript file receiver.csc. After doing this, the center of each sensor node will be colored in orange and the name of the assigned SenScript file will be displayed on the sensor node in gray color.



Step 6. Configure the simulation parameters: To visualize the result of the simulation, in this example one assign to the simulation speed the value 500 ms (1/2 second) and to the arrow speed the value 1000 ms (1 second). The arrows correspond to the send messages.

Step 7. Run the simulation: Just click on Run Simulation button

The red arrow shows the sent messages. The value in the middle of the arrow represents the sent message. If the transmitter is sending A and B instead of 1 and 0, then the scripts must be rewritten as follows:

Transmitter 2:

```
loop
send "A" 2
delay 1000
send "B" 2
delay 1000
```

Receiver 2:

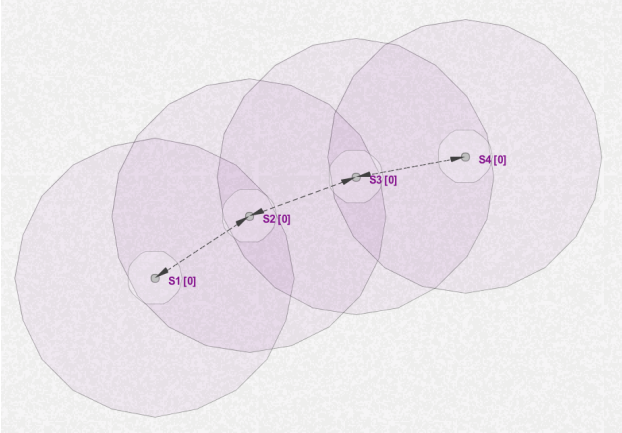
```
loop
receive v
if(v=="A")
    mark 1
else
    mark 0
end
```

To stop manually the simulation, just click on the button .

Remark: Be sure that the id of the transmitter is 1 and the one of the receiver is 2. It is possible to change the value 2 by * for the broadcast sending (send "A" *).

Example 8: Routing messages

The previous Example 7 can be completed by adding routers between the transmitter and the receiver. Let create 4 sensor nodes as follows.



In the same way as in the Example 7, we will create the same SenScript codes for the transmitter (sensor node 1) and the receiver (here, the sensor node 4). We will add two routers, the first one, the sensor node 2, will route the received messages to the sensor node 3, and the second router is the sensor node 3, which will route the received messages to the sensor node 4 (the receiver). The codes of the routers are given as follows:

Router 1 (routing to sensor node 3)

```
loop
receive v
send v 3
```

Router 2 (routing to sensor node 4)

```
loop
receive v
send v 4
```

Simulate …

Using this method requires to create a new script for each router. It is possible to use another method based on the command `send m * x` which allows to send the message **m** to all the neighbors except the neighbor **x**. The advantage of this command is to use the same script file for all the routers. Then, the scripts of the previous example can be rewritten as follows:

The transmitter:

```
atget id id          → id = the identifier of the sensor node (in the example id=1)
loop
data p id "A"        → p = "1#A"
send p               → send "1#A" in a broadcast mode
delay 1000           → wait for 1 second
data p id "B"        → p = "1#B"
send p               → send "1#B"
delay 1000           → wait 1 second
```

The router, this script will be assigned to the sensor node 2 and 3:

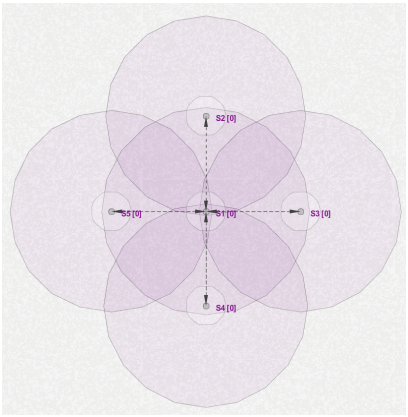
```
atget id id          → id = the identifier of the sensor node (id of the router 2 or 3)
loop
receive rp           → the received message will be assigned to rp (eg. rp = 1#A or 2#A)
rdata rp rid v       → rid=1 and v=A (sensor node 2), rid=2 and v=A (sensor node 3)
data p id v          → p = 2#A
send p * rid         → send 2#A to all the neighbors except the neighbor rid (ie. 1 or 2)
```

The receiver:

```
loop
receive rp          → read the received message and assign it to rp
rdata rp rid v      → rid = 3 and v=A
if(v=="A")          → if v==A
  mark 1            → the sensor node will be marked
else                → otherwise,
  mark 0            → it will be unmarked
end
```

Example 9: Sending Broadcast messages

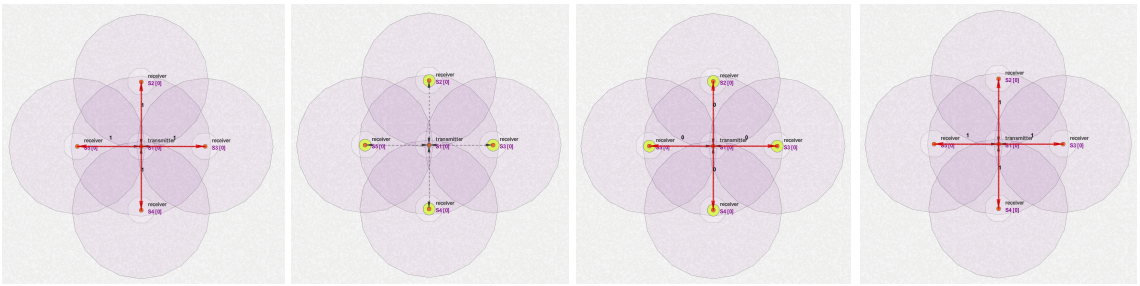
To send a message in a broadcast mode, one needs just to remove the id of the receiver in the **send** command. One can also replace it by *. Let consider the same Example 7 with one transmitter and 4 receivers, as follows:



The codes of the receivers are same. That is to say, one will assign the same SenScript file for each receiver. No need to create different scripts for each sensor node since they have the same script. We need just to remove in the script of the transmitter the id of the receiver. The script will be as follows:

```
loop
send "A" 2
delay 1000
send "B" 2
delay 1000
```

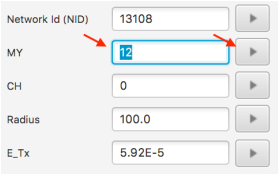
Simulate …



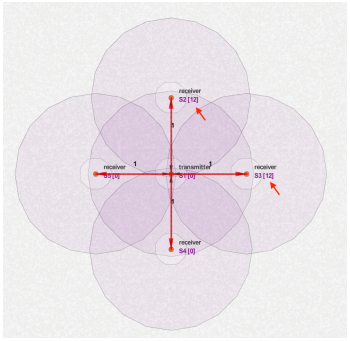
Example 10: Sending messages to a group

This example is the same as the Example 9 with the current changes:

1. Change the MY addresses of the sensor nodes 2 and 3 to the value 12. To do this, select each sensor node and open the Radio Parameters view in the left of the CupCarbon environment. Modify the value of the field MY to 12 and then click on the apply button in the right of this filed.



Once the button apply is pressed, one remark, for the corresponding sensor nodes, that between the brackets situated in the right of the name in the center, the value is changed from 0 to 12. This value represents the MY address.

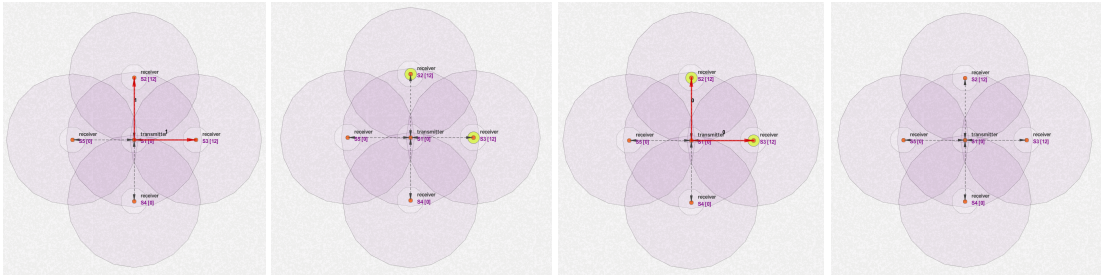


2. To send messages to a group of sensor nodes having the same MY address, we modify slightly the code of the transmitter, of the previous example, as follows:

```
loop
send "A" 0 12
delay 1000
send "B" 0 12
delay 1000
```

The value 0 of this script means that the message will be sent for sensor nodes having the MY address given in the followed number. In this example, this number is equal to 12.

The simulation will lead to the following result:

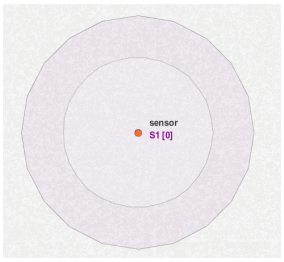


Example 11: Reading digital sensor values

In this example, we will add two sensor nodes and one mobile. The first sensor node will send a message to its neighbor (the second sensor node) about the state of its sensing unit each 100 milliseconds (0 if no

detection and 1 if an event is detected). The second sensor node will print the received message and will be marked if it receives 1 and unmarked if it receives 0. A mobile will be added to pass near to the first sensor node, exactly near to its sensing unit, in order to be detected.

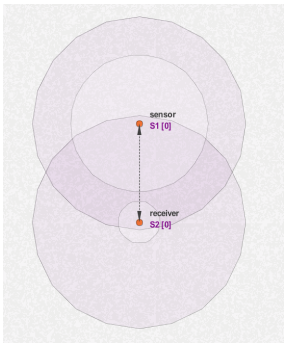
Let first create the first sensor node that sends each 100 milliseconds the value of its sensing unit. Let call its script sensor. Let increase its sensing unit so that to obtain the following result:



Script of the sensor node 1

```
loop
dreadsensor s
send s
delay 100
```

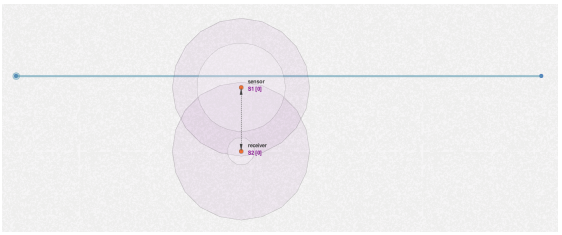
Then, the second sensor node will just await for messages received from the first one and be marked or not according to the received message.

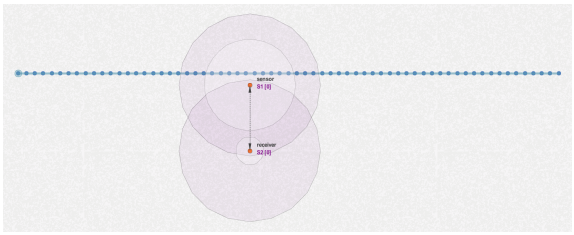


Script of the sensor node 2

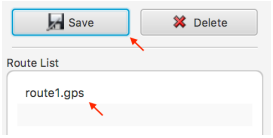
```
loop
receive v
mark v
```

Before adding a mobile, first, we start with adding a route. To do this, we will add two markers and then select both of them. Then we will type many times on the key ‘u’ of the keyboard until to obtain the following result:

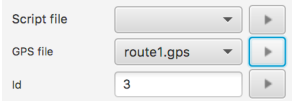




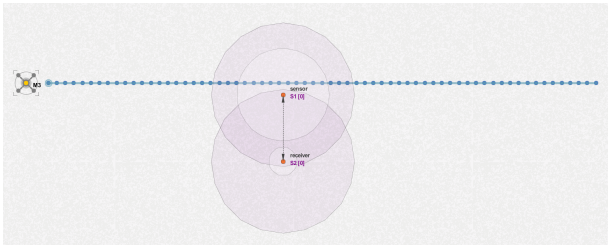
The route must be saved using the button **Save** in the Marker Parameters view.



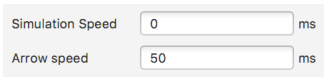
Now, we can add a mobile and assign it the previously created route. This is done using the field **GPS file** in the Device Parameters view.



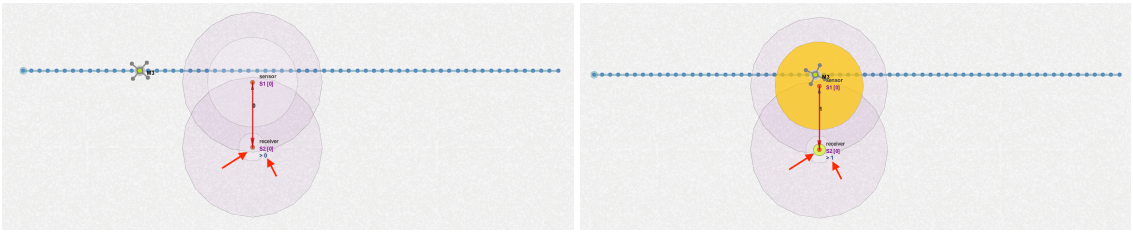
Once done, the mobile will have a center with the orange color. Then, the project is ready for simulation.



For the simulation, change the value of the simulation speed to 0 ms and the Arrow Speed to 50 ms.

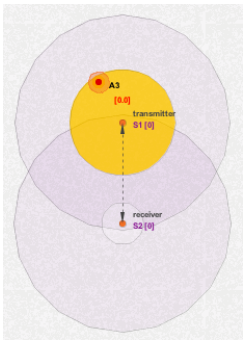


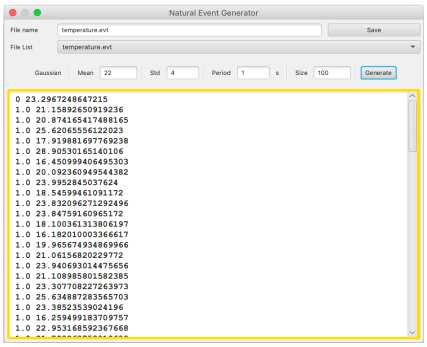
The simulation results must be as follows:



Example 12: Reading analog sensor values

This example is the same as the previous one (Example 11) except that there is no mobility and the mobile is replaced by a natural event (a gas) that will generate random values following the Gaussian distribution (mean = 22 and std = 4). The transmitter will read each 100 milliseconds the value of its sensor unit and then send it to the receiver. The receiver will be marked if it receives a value that is greater than 20. The interface will look like the following one:





To create a file with 100 natural events generated each second from a Gaussian distribution (mean=22 and std=4), we will use the Natural Event Generator. The script of the transmitter and the receiver are given as follows:

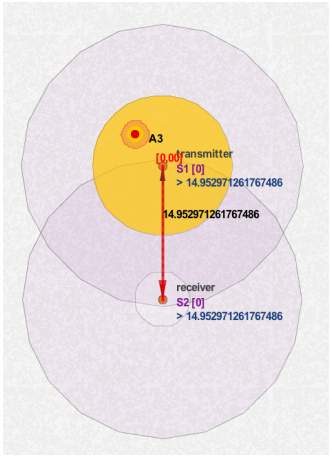
Transmitter:

```
loop
areadsensor v
if(v!="X")
    print v
    rdata v a b c
    send c 2
end
delay 1000
```

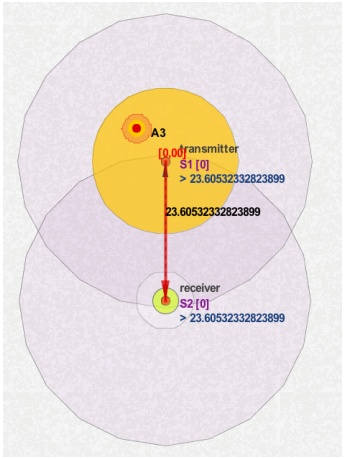
Receiver:

```
loop
receive y
print y
if(y>20)
    mark 1
else
    mark 0
end
```

The simulation results will show something like the following figures where the sensor node is unmarked after receiving a value less than 20 and it is marker after receiving the value 23 which is greater than 20.



Unmarked sensor node after receiving the value 14<20



Marked sensor node after receiving the value 23>20

It is also possible to write the received values in a file during the simulation using the command **printfile** as follows:

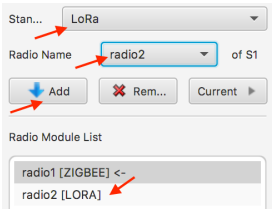
```
loop
receive y
print y
time t          → get the current simulation time
printfile t y   → print in a file the current simulation time and the received value x
if(y>20)
    mark 1
else
    mark 0
end
```

The obtained file has the same name as the executing sensor node and it is located in the directory results.

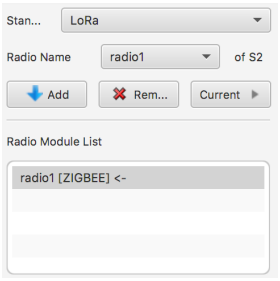
Example 13: Using many radio modules and standards

This example shows how to used many radio modules in the same sensor node and switch between them. As an example, we will consider 3 sensor nodes. One transmitter with two radio modules (ZigBee and WiFi), and two receivers having each one of them one radio module. The standard of the first one is ZigBee and the standard of the second one is WiFi. The transmitter will send 1 and 0 each second using its first radio module and will switch to the second radio module while sending 1 and 0 each second. The scenario is created as follows:

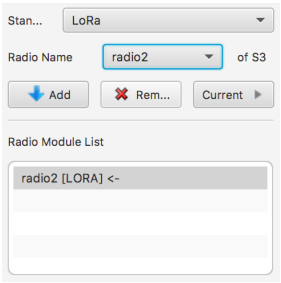
1. Create a new project,
2. Add a new sensor node,
3. Open the radio parameters view: we can see that one radio module with the standard ZigBee is already added automatically,
4. Add a second radio module by changing the standard to **LoRa** and the name to **radio2** and then by clicking on **Add** button. In the radio module list we will find two radio modules radio1 [ZIGBEE] and radio2 [LoRa]. It is possible to change the current radio module to the second one (LoRa) by selecting it and by clicking on the button **Current**. It is not possible to delete all the radio modules as well as the current radio module.



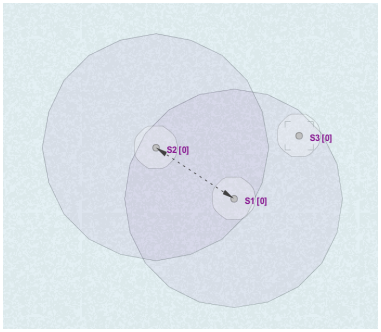
5. Add another sensor node without making any changes.

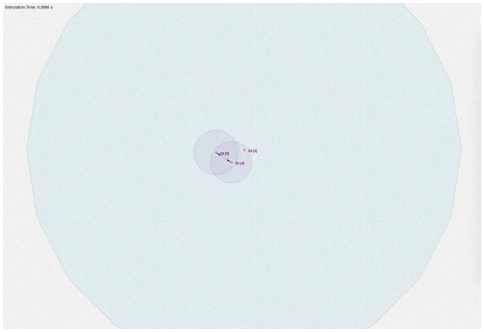


6. Add a third sensor node by adding a radio module LoRa, choose it as the current radio module and then delete the first (ZigBee) radio module.



7. The scene will look like in the following Figure:





8. Add the following scripts for each sensor node:
   a. Sensor Node S1 (with two radio modules):

```
loop
radio radio1
send 1
delay 1000
send 0
delay 1000
radio radio2
send 1
delay 1000
send 0
delay 1000
```

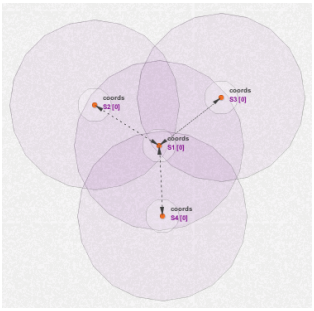
   b. The same script for the other sensor odes (S2 and S3):

```
loop
receive v
mark v
```

9. Simulate …

Example 14: My coordinates and my neighbors

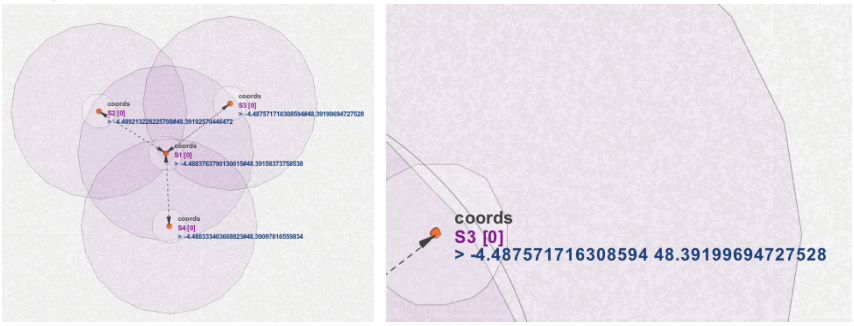
In this example, we will add 3 sensor nodes as follows and we will display they GPS coordinates (longitude and latitude).



1. **Display the Coordinates of each sensor node:** the script that allows to display the coordinates of a sensor node is given by:

```
loop
getpos x
print x
stop
```

Assign this script to each sensor node and simulate. The result will be as follows:



To recuperate each coordinate separately, use the command **rdata**. One can use also the command getpos2 that allows to recuperate the coordinates into 2 separated variable as follows:

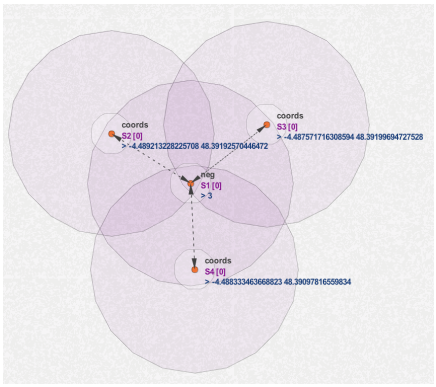
```
loop
getpos2 x y
print x y
stop
```

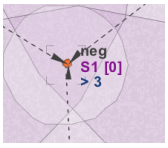
2. **Obtaining the list of the neighbors:** To obtain the list of the neighbors of the sensor S1 for example, we can use two commands. The first one is **atnd x** and the second one is **atnd x y**. The first command allows to assign to x the number of neighbors and the second one allows to assign to x the number of neighbors and to y the list of the identifiers of the neighbors separated by #. In the following we present two scripts using each of them each of these commands and the results of their simulation.

Script 1 (**atnd x**): this will display the number of the neighbors of S1.

```
loop
atnd x
print x
stop
```

Simulation result:

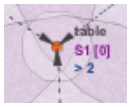




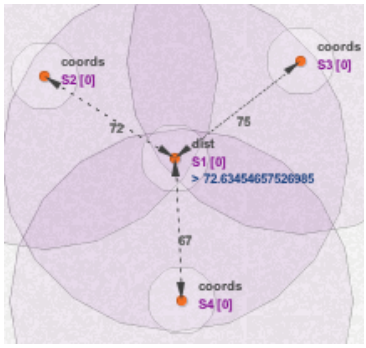
Script 2 (**atnd x y**): this will display each second the identifier of each neighbor (for good visualization, you must set the value of the simulation speed to 1000):

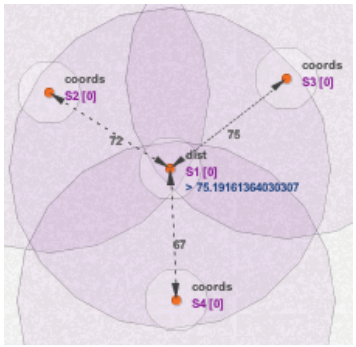
```
loop
atnd n v
for i 0 n
    vget x v i
    print x
    delay 1000
end
stop
```

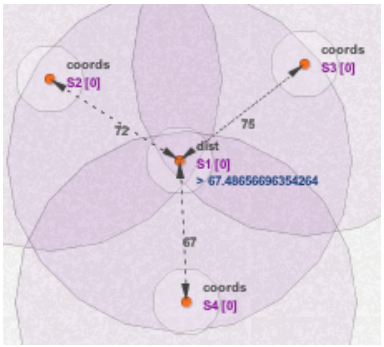
Simulation result:



In the following, we will display each second the distance with each neighbor:







Example 15: Working with radio parameters

In this example, we will use the same procedure as Example 1. Instead of displaying Hellow World we will display each second:

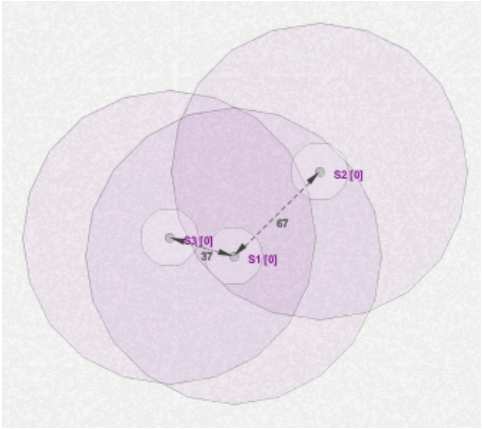
1. The Network ID
2. The Channel
3. The MY address

```
loop
atget nid v1
atget ch v2
atget my v3
print v1
delay 1000
print v2
delay 1000
print v3
stop
```

Change the simulation speed to 1000 and simulate …

Example 16: Transmission power

In the following, in order to show how to change the power of transmitted signal, let consider the following example with three sensor nodes:



The first sensor node will send A and B each second and the other sensor nodes will display the received message (i.e., A and B). After each transmission of the couple A and B we will change the percentage of the power of the sending message. In the beginning we will put 100% and then we switch to 60%. The script of this situation is written as follows:

The script of the transmitter:

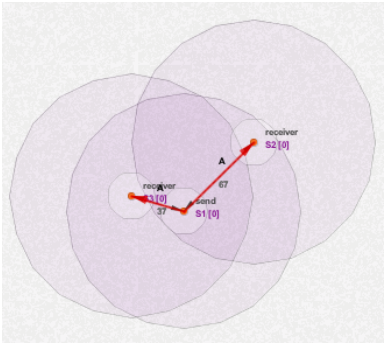
```
loop
atpl 100
send "A"
delay 1000
send "B"
delay 1000
atpl 60
send "A"
delay 1000
send "B"
delay 1000
```

The script of the receiver:

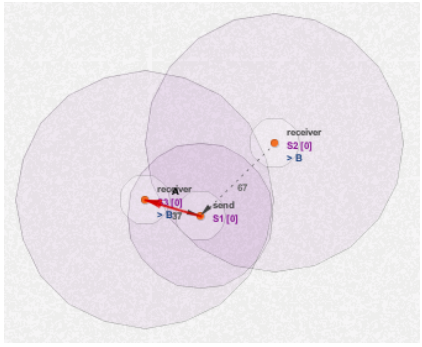
```
loop
receive x
print x
```

Simulate with the following simulation parameters:

The results are close to:



100%



60%

Example 17: Interferences and Acknowledgments

Use Example 8 with the following scripts:

Router 1 (routing to sensor node 3)

```
loop
receive v
send v 3
```

Router 2 (routing to sensor node 4)

```
loop
receive v
send v 4
```

Before simulating, activate the ACK and the Show boxes. Run simulation …

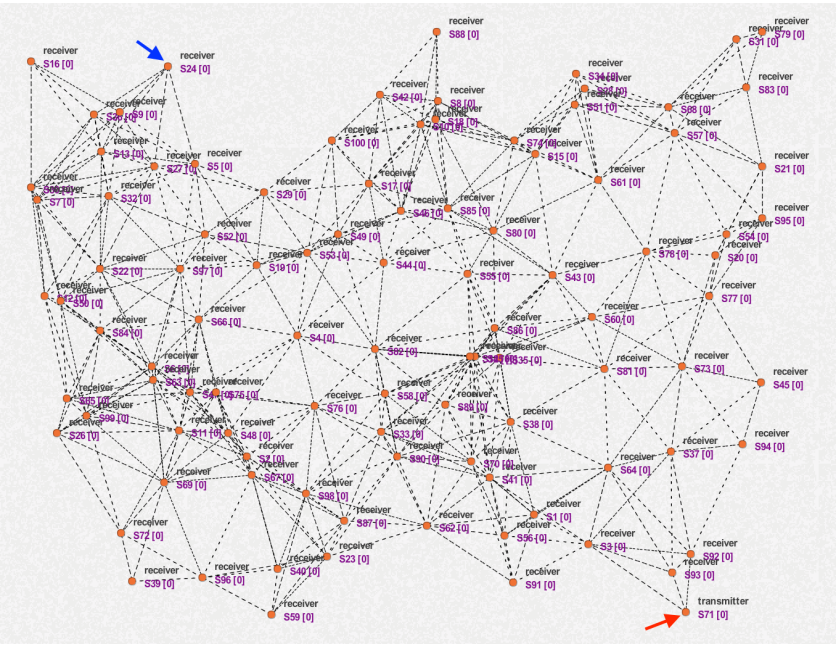
Note that there are no acknowledgment messages when sending in a broadcast mode (i.e., in the example, there is no ACK message for the message sent by S1).

Advanced Examples

Tutorial 1: Send me your coordinates please

In this example a specified sensor node will send a message to ask a specific sensor node in the network for its coordinates. We will see, in this example, that there is no need of neighboring detection process and saving the table of the neighbors locally. This is just an example and not a recommended routing protocol. Because, in a real network, doing this process for each action (asking for coordinates) will be, indeed, very energy consuming. Note that the CupCarbon simulation and the SenScript is executed at the application level.

Create a new project and add 100 sensor nodes randomly.



To obtain this visualization, click on the button **Connections** in the state bar. Let take any sensor node, for example S71, the starting node situated in the extreme bottom right of the network, as shown in red arrow in the Figure above. This sensor node will ask for the coordinate of the sensor node S24 situated in the top left of the network, as shown by the blue arrow of the figure above. These ids (24 and 71) must be adapted and changed in your example. Then, replace these values in your script by the ones of your example.

First, we will start with a script that allows just to detect and mark the concerned sensor node (i.e., S24). To do this, S71 will send in a broadcast the message formed by A and 24 which means “I am searching for the sensor node having the id 24”. The other sensor nodes, which will receive this message, will do the same thing **once** if they are not the sensor node S24. Otherwise, they will be marked.

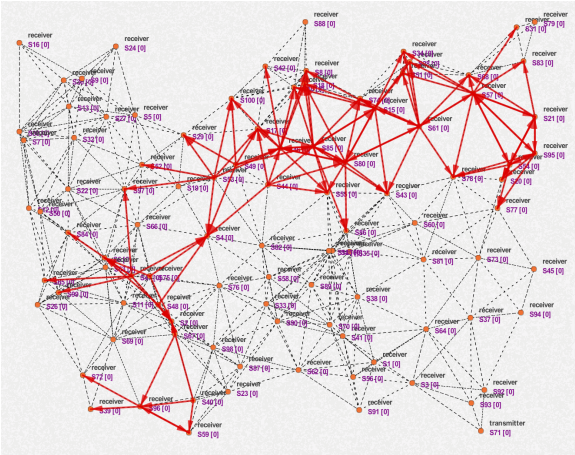
The script of the starting node (S71) is given as follows:

```
atget id id
data d id "A" 24
send d
loop
stop
```

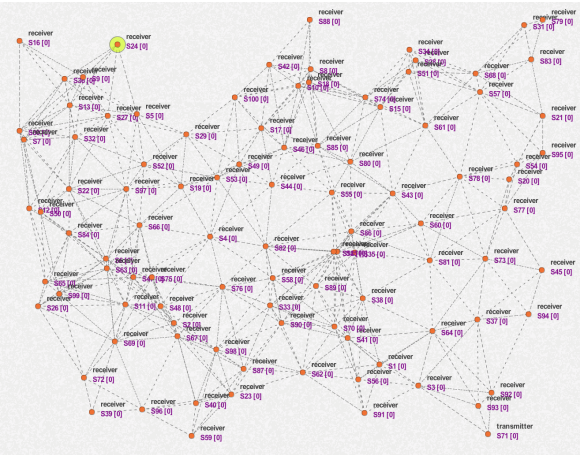
The script of the receivers (other sensor nodes) is given as follows:

```
atget id id
set recA 0
loop
receive m
rdata m rid type info
if((type=="A") && (recA==0))
    set recA 1
    if(info==id)
        mark 1
    else
        data d id "A" info
        send d
    end
end
```

The variable id represents the current id of the sensor. The variable recA means “already received the message A” in order to send the message A once or to be marked once if its id is the one which is researched (here, 24). The info represents here 24, which is the researched id. We call it info because in the following next step this info will become the coordinates of the sensor node 24 which will be sent to the sensor node S71. Simulate with Simulation speed = 0 and Simulation Arrow speed = 500. The simulation will give the following result:



During simulation



End of simulation

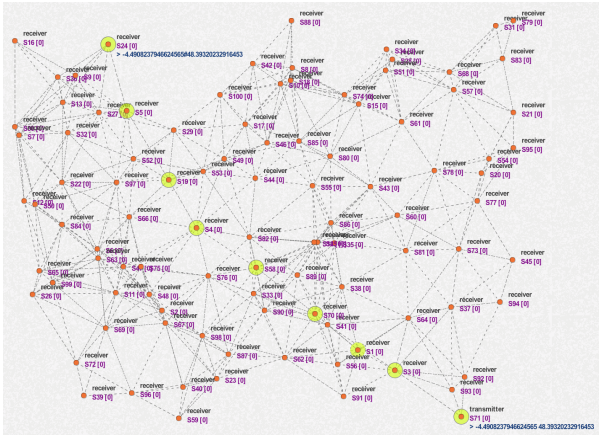
Now, the marked sensor node will display its coordinates and will send them as a message B to the previous sensor node (here, S5), which sent to it the message A. This previous sensor node (S5) will send again the B to its previous sensor node, and so on until reaching the starting sensor node S71. This last one will be marked and display the received coordinates! The previous scripts will be completed as follows. The script of the starting node (S71) will be as follows:

```
atget id id
data d id "A" 24
send d
loop
receive m
rdata m rid type x y
if (type=="B")
    mark 1
    print x " " y
    stop
end
```

The script of the other sensor nodes will be as follows:

```
getpos p
atget id id
set recA 0
loop
receive m
rdata m rid type info info2
if((type=="A") && (recA==0))
    edge 1 rid
    set recA 1
    if(info==id)
        mark 1
        print p
        data d id "B" p
        send d rid
    else
        set prev rid
        data d id "A" info
        send d
    end
end
if(type=="B")
    mark 1
    data d id "B" info info2
    send d prev
end
```

The simulation will give the following result:



Simulate with Simulation speed = 0 and Simulation Arrow speed = 500.

Tutorial 2: Find the extreme left node

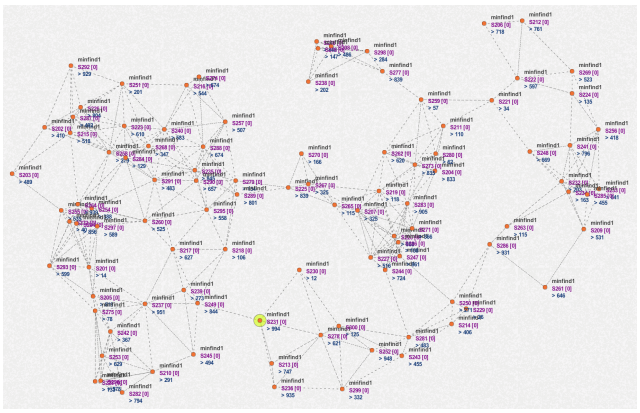
This example is inspired from the example of finding the leader in the network. First, each sensor node will generate a random value between 0 and 1000 and then marks the sensor node having the maximum generated value.

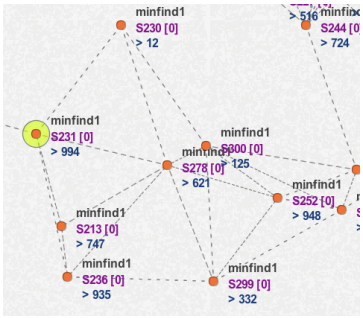
The script of all the sensor nodes is the same and it is given as follows:

```
randb x 0 1000
print x
mark 1
set vmax x
send vmax
loop
receive v
```

```
if (v > vmax)
    mark 0
    set vmax v
    send v
end
delay 1
```

The result of the simulation can be as follows. Test with Simulation Speed=0 and for Arrows Speed =0, 100 and 500. If there are many sensor nodes having the same maximum then they will all be marked.

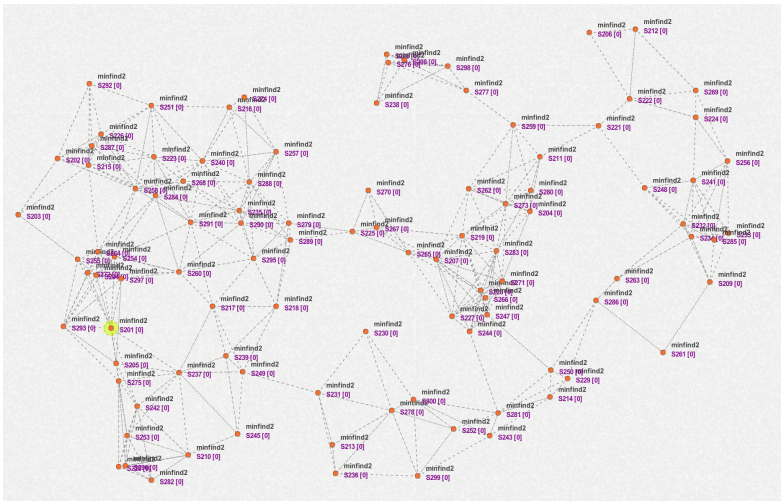




This algorithm can be used also to find the sensor node having the minimum id. In this case, only one sensor node will be marked. The script is almost like the previous one, only the variable **x** will be replaced by the id of the sensor node instead of a random value, and we don't need to print it as it represents the id itself, which is already displayed on the map for each sensor node. Also, as we are searching for the min instead of the max, we will change the if condition to **<**.

```
atget id vmin
mark 1
send vmin
loop
receive v
if (v < vmin)
    mark 0
    set vmin v
    send v
end
delay 1
```

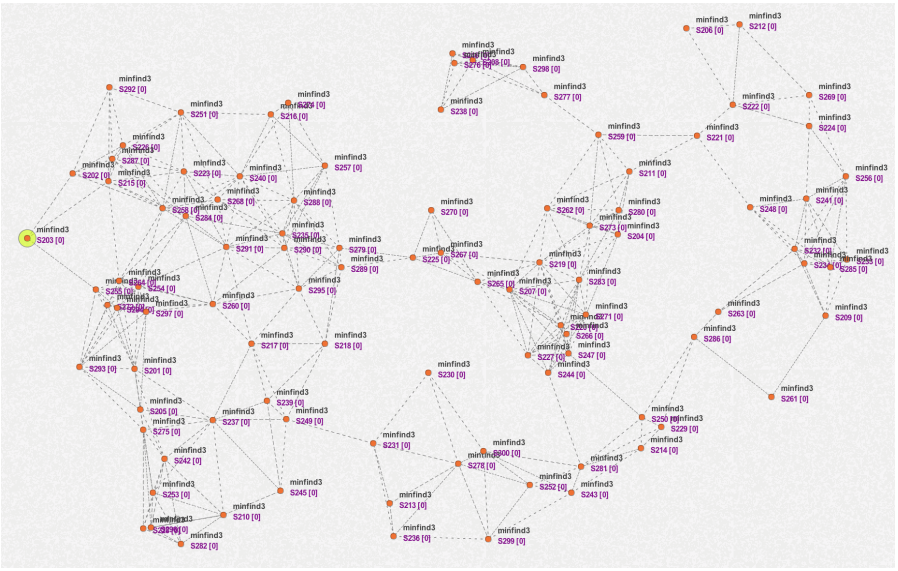
The simulation result will give (here the id max is 201):



This same script can be used now to find the sensor node, which is in the extreme left of the network. It is simply, the sensor node having the minimum x-coordinate.

```
getx vmin
mark 1
send vmin
loop
receive v
if (v < vmin)
    mark 0
    set vmin v
    send v
end
delay 1
```

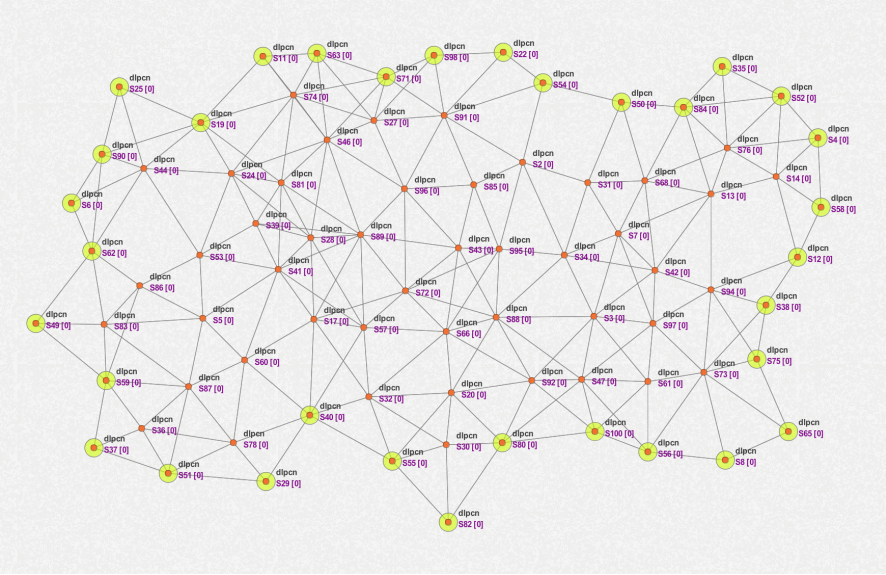
The simulation result gives:



Tutorial 3: Simulate the D-LPCN algorithm (version 1)

The D-LPCN algorithm allows to find the boundary sensor nodes of a network. It starts from any boundary sensor node (for example, the one in the extreme left). Then, each node calculates the angles formed by the previously found boundary node and each one of its neighbors; and then it chooses the node that forms the smallest angle as the next boundary node. To write this algorithm, we will use 3 types of messages. A message *AC* to ask the neighbors to send their coordinates, a message *CS* for sending the coordinates and a message *SN* to inform a sensor that it is a boundary node. The script of the DLPCN algorithm is given as bellow. First, (lines 1 to 3), each sensor node will assign to the variable *cid* its current *id* and it calculates its current coordinates *cx* and *cy*. Since this script is executed by all the sensor nodes of the network, we will add the variable *first* (here it is equal to 49, which represents the extreme left sensor node) to designate the starting node. Some parts of the script will be executed only by the first node. Then, (lines 5 to 10), only the first node will execute this part. It will execute it once. That is why in line 6, the variable *first* is set to 0. To calculate the angles formed by the previous boundary node and its neighbors. In lines 5 to 8, we will consider that there is a virtual sensor node, which is situated in the left part of the starting node, which have the coordinates (*px=cx-1* and *py=cy*). Then, a message *SN* will be considered since the starting node is also a boundary node. The line 10 (*SN* message) will be followed directly by line 22. The block 23 to 26 represents the stop condition, which represents visiting twice the starting node. In the first time, the variable *first* is equal to 0, then the stop condition is not verified, and the variable *first* will be set to -1 (line 26). However, if we come back to this block a second time, the condition *(first == –1)* is verified, then the *stop* command will be called. If the stop condition is not verified, then, as the considered block is executed after receiving an *SN* message, which means that the receiver is a boundary sensor node. The line 27 will mark this sensor, and then, from line 28 to 45 the process of asking and receiving the coordinates of the

neighbors will be started. For each received message (coordinates) the angle will be calculated and compared with the previously calculated angle in order to determine the smallest one. Line 28 allows to get the value of the coordinates *px* and *py* of the previous boundary node. Line 29 forms a message with an angle equal to 10 radian. Line 30 is used to get the number of neighbors and their identifiers. Lines 31 to 34 are used to send to each neighbor node an *AC* message to ask it to send its coordinates. Lines 35 to 45 are used to wait for the answer of the neighbors. If a *CS* message is received, then the coordinates are sent and can be used to calculate the angle (line 40) in order to select the minimum one (line 42). The function *smin* allows to find the minimum value between two values extracted from messages formed by the *id* of a sensor node and its corresponding variable. Once the minimum angle is found as well as its corresponding sensor node (*id*), then a message *SN* will be sent (lines 46 and 47) to inform the obtained sensor node that it is a boundary sensor node. Note that, the lines 17 to 20 are executed by the sensor nodes that receive the message *AC* in order to send their coordinates (with message *CS*: line 18). For simplicity, we consider as stop condition when the first starting node is visited twice, which is not the real stop condition. For more information about this algorithm, please refer to the paper [1]. The execution of the script given bellow will result to the following situation:



```
1:   atget id cid
2:   getpos2 cx cy
3:   set first 49
4:   loop
5:   if (cid==first)
6:      set first 0
7:      set px cx-1
8:      set py cy
9:      data p 0 "SN" px py
10:     set type "SN"
11:  else
12:
13:     receive p
14:     rdata p id type
15:  end
16:
17:  if (type=="AC")
18:     data p cid "CS" cx cy
19:     send p id
20:  end
21:
22:  if (type=="SN")
23:     if (first==-1)
24:        stop
25:     end
26:     set first -1
```

```
27:     mark 1
28:     rdata p id type px py
29:     data m cid 10
30:     atnd n tneg
31:     for i 0 n
32:         vget neg tneg i
33:         data p cid "AC"
34:         send p neg
35:
36:         receive p
37:         rdata p id type
38:         if (type=="CS")
39:             rdata p id type x y
40:             angle2 a px py cx cy x y
41:             data p id a
42:             smin m m p
43:             rdata m id a
44:         end
45:     end
46:     data p cid "SN" cx cy
47:     send p id
48: end
```

This script can be written as follows, where we consider broadcast messages to the neighbors.

```
1  : atget id cid
2  : getpos2 cx cy
3  : set first 49
4  : loop
5  : if (cid==first)
6  :    set first 0
7  :    set px cx-1
8  :    set py cy
9  :    data p 0 "SN" px py
10 :    set type "SN"
11 : else
12 :
13 :    receive p
14 :    rdata p id type
15 : end
16 :
17 : if (type=="AC")
18 :    data p cid "CS" cx cy
19 :    send p id
20 : end
21 : if (type=="CS")
22 :    rdata p id type x y
23 :    angle2 a px py cx cy x y
24 :    data p id a
25 :    smin m m p
26 :    rdata m id a
27 :    inc i
28 :    if(i==n)
29 :        data p cid "SN" cx cy
30 :        send p id
31 :    end
32 : end
33 : if (type=="SN")
34 :    if(first==-1)
35 :        stop
36 :    end
37 :    set first -1
38 :    rdata p id type px py
39 :    mark 1
40 :    data m cid 10
41 :    atnd n
```

```
42 :    set i 0
43 :    data p cid "AC"
44 :    send p
45 : end
```

Tutorial 4: Simulate the D-LPCN algorithm (version 2)

In this example we will show how we can merge many SenScripts in one script. For example, the script given previously in Example 3 starts from a given sensor node. It is the sensor node that is in the extreme left of the network. Example 2 shows another script allowing to find this extreme left sensor node. Therefore, in the following, we will show how to write a code that, first, finds the extreme left sensor node, using the script of Example 2, and then starts the D-LPCN algorithm form that one, using Example 3. To do this, we will create a new variable ***step***. This variable will be equal to 1 and then to 2 in order to determine which algorithm will be executed. Let call the code of Example 2 by **Code1** and the one of Example 3 by **Code2**. Then the final script will have almost the following structure:

```
set step 1
if(step == 1)
    if (condition)
         set step 2
    end
    Code1
end
if(step == 2)
    Code2
end
```

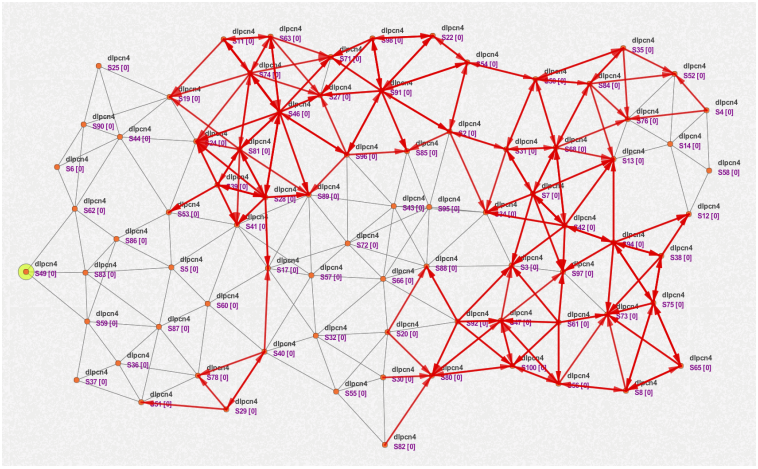
Code1 will allows to determine the sensor node which is in the extreme left. The identifier of this sensor node will be the value of the variable ***first*** which is used in Code2 for the starting node. In this new situation, in the script that finds the extreme left sensor node, the ***wait*** command will lead to a blocking situation. No sensor node can continue execution. Then, it is not possible to go to the second step. To overcome this limitation, we will use a command ***wait t***, which will wait for receiving messages during a certain time ***t*** and continue the execution after this time. In our case, this time represents the required time to finish the first process of finding the extreme left node. If after this time there are no read messages (empty message), then we start the second step of determining the boundary nodes using the D-LPCN algorithm.

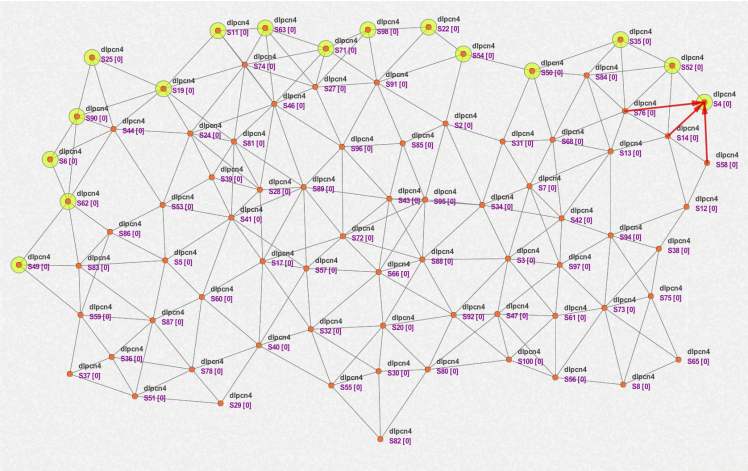
The final script is given as follows:

```
1  : set step 1
2  : set first -2
3  : getpos2 vmin y
4  : set marked 1
5  : send vmin
6  : loop
7  : if(step == 1)
8  :     delay 1
9  :     receive v 2000
10 :     if(v == "")
11 :         atget id cid
12 :         getpos2 cx cy
13 :         set step 2
14 :         if (marked == 1)
15 :             set first cid
16 :         end
17 :     else
18 :         if (v < vmin)
19 :             set marked 0
20 :             set vmin v
21 :             send v
22 :         end
23 :     end
24 : end
25 : if(step == 2)
26 :     if (cid==first)
27 :         set first 0
```

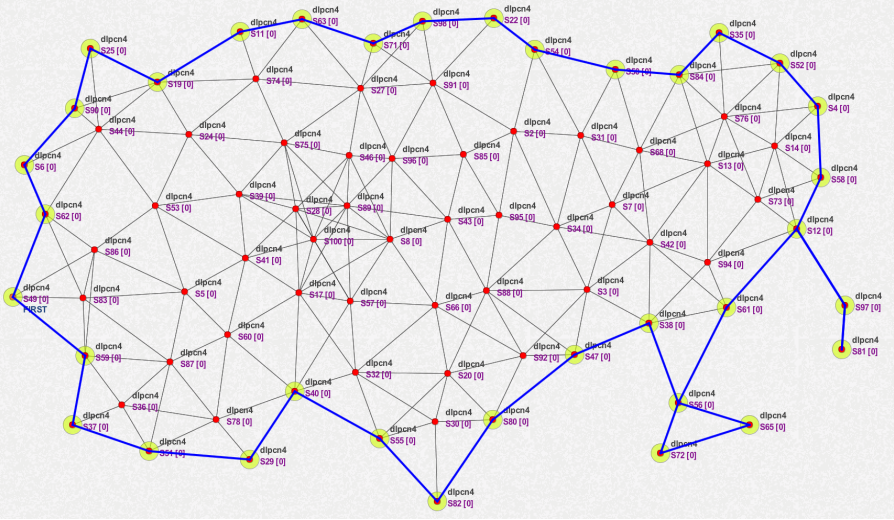
```
28 :         set px cx-1
29 :         set py cy
30 :         data p 0 "SN" px py
31 :         set type "SN"
32 :     else
33 :
34 :         receive p
35 :         rdata p id type
36 :     end
37 :     if (type=="AC")
38 :         data p cid "CS" cx cy
39 :         send p id
40 :     end
41 :     if (type=="CS")
42 :         rdata p id type x y
43 :         angle2 a px py cx cy x y
44 :         data p id a
45 :         smin m m p
46 :         rdata m id a
47 :         inc i
48 :         if(i==n)
49 :             data p cid "SN" cx cy
50 :             send p id
51 :         end
52 :     end
53 :     if (type=="SN")
54 :         if(first==-1)
55 :             stop
56 :         end
57 :         if(first==0)
58 :             set first -1
59 :         end
60 :         set first -1
61 :         rdata p id type px py
62 :         mark 1
63 :         data m cid 10
64 :         atnd n
65 :         set i 0
66 :         data p cid "AC"
67 :         send p
68 :     end
69 : end
```

Simulate with Simulation speed = 0 and Arrow Speed = 100. For the visualization, since the simulation speed is more quick for the first step than for the second one, it is recommended to change the arrow speed parameter for example from 5 for step 1 to 200 for step2. This procedure is difficult to do manually. That is why we recommend to use the command ***simulation aspeed*** to force different values for each step.





If we add the instruction **`edge 1 id`** between the lines 48 and 49 of the last script, we get the following result (with marked boundary edges):



Tutorial 5: Simulate the D-LPCN algorithm (version 3)

In this tutorial we will use the command **script**. First, we can consider the script of the D-LPCN algorithm without specifying the value of the variable **first** representing the id of the starting node. This script must be saved with the name dlpcn.csc. To call it from another script, we use the command **script dlpcn**.

```
1  : atget id cid
2  : getpos2 cx cy
3  : set first 49
4  : loop
5  : if (cid==first)
6  :    set first 0
7  :    set px cx-1
8  :    set py cy
9  :    data p 0 "SN" px py
10 :    set type "SN"
11 : else
12 :
13 :    receive p
14 :    rdata p id type
15 : end
16 :
17 : if (type=="AC")
18 :    data p cid "CS" cx cy
19 :    send p id
20 : end
21 : if (type=="CS")
22 :    rdata p id type x y
23 :    angle2 a px py cx cy x y
24 :    data p id a
25 :    smin m m p
26 :    rdata m id a
27 :    inc i
28 :    if(i==n)
29 :        data p cid "SN" cx cy
30 :        send p id
31 :    end
32 : end
33 : if (type=="SN")
34 :    if(first==-1)
35 :        stop
36 :    end
37 :    if(first==-1)
38 :        set first -1
39 :    end
40 :    rdata p id type px py
41 :    mark 1
42 :    data m cid 10
43 :    atnd n
44 :    set i 0
45 :    data p cid "AC"
46 :    send p
47 : end
```

Then, in the MinFind algorithm, we will call the previous script LPCN once finished using the command script (line 16). Note, that we will add a delay time once the first node is found (line 12). This will allow to wait for the other nodes to finish their MinFind algorithm and to be ready to start the LPCN algorithm. The MinFind algorithm can be rewritten in this case as follows. In this tutorial, only the MinFind algorithm is assigned to the sensor nodes. The LPCN algorithm will be called automatically from this one (line 16).

```
1  : atget id cid
2  : getpos2 vmin y
3  : set marked 1
4  : send vmin
5  : loop
```

```
6  : mark marked
7  :
8  : receive v 2000
9  : if(v == "")
10 :    if (marked == 1)
11 :        set first cid
12 :        delay 1000
13 :    else
14 :        set first -3
15 :    end
16 :    script "dlpcn"
17 : else
18 :    if (v < vmin)
19 :        set marked 0
20 :        set vmin v
21 :        send v
22 :    end
23 : end
24 : delay 1
```

References:

[1] Massinissa Saoudi, Ahcène Bounceur, Farid Lalem, Reinhardt Euler, M-Tahar Kechadi, Abdelkader Laouid, Madani Bezoui, Marc Sevaux, D-LPCN: A Distributed Least Polar-angle Connected Node Algorithm for Finding the Boundary of a Wireless Sensor Network, Ad Hoc Networks Journal, Elsevier, Volume 56, 1 March 2017, Pages 56–71, DOI: 10.1016/j.adhoc.2016.11.010.

[2] Taha Alwajeeh, Pierre Combeau, Rodolphe Vauzelle and Ahcène Bounceur, A High-Speed 2.5D Ray-Tracing Propagation Model for Microcellular Systems, Application: Smart Cities, In the 11th IEEE European Conference on Antennas and Propagation (EuCAP 2017), 19-24 March 2017, Paris, France.

[3] Umber Noreen, Ahcène Bounceur and Laurent Clavier, Modeling Interference for Wireless Sensor Network Simulators, In ACM International conference on Big Data and Advanced Wireless technologies (BDAW'2016). Blagoevgrad, Bulgaria, November 10-11, 2016.

[4] Farid Lalem, Ahcene Bounceur, Rahim Kacimi, Reinhardt Euler and Saoudi Massinissa, Faulty Data Detection in Wireless Sensor Networks Based on Copula Theory, In ACM International conference on Big Data and Advanced Wireless technologies (BDAW'2016). Blagoevgrad, Bulgaria, November 10-11, 2016.

[5] Umber Noreen, Ahcène Bounceur, Laurent Clavier, Rahim Kacimi, Performance Evaluation of IEEE 802.15.4 PHY with Impulsive Network Interference in CupCarbon Simulator, Invited paper, in the 3rd IEEE International Symposium on Networks, Computers and Communications (ISNCC), Hammamet, Tunisia, May 11-13, 2016.

[6] Farid Lalem, Rahim Kacimi, Ahcène Bounceur, Reinhardt Euler, Boundary Node Failure Detection in Wire-less Sensor Networks, The 3rd IEEE International Symposium on Networks, Computers and Communications (ISNCC), Hammamet, Tunisia, May 11-13, 2016.

[7] Umber Noreen, Ahcene Bounceur, Laurent Clavier, Integration of OFDM Based Communication System with Alpha-Stable Interference using CupCarbon Simulator, In the ACM International Conference on Internet of things and Cloud Computing (ICC'2016), University of Cambridge, United Kingdom, March, 22-23, 2016.

[8] Taha Alwajeeh, Pierre Combeau, AhcèneBounceur, Rodolphe Vauzelle, Efficient Method for Associating Radio Propagation Models with Spatial Partitioning for Smart City Applications, In the ACM International Conference on Internet of things and Cloud Computing (ICC'2016), University of Cambridge, United Kingdom, March, 22-23, 2016.

[9] Farid Lalem, Muath Alshaikh, Ahcène Bounceur, Reinhardt Euler, Lamri Laouamer, Laurent Nana, Anca Pascu, Data Authenticity and Integrity in Wireless Sensor Networks Based on a Watermarking Approach, The 29th In-ternational Florida Artificial Intelligence Research Society Conference (FLAIRS), May 16-18, 2016, Key Largo, Florida, USA.

[10] Massinissa Lounis, Ahcène Bounceur, Laga Arezki and Bernard Pottier. GPU-based Parallel Computing of Energy Consumption in Wireless Sensor Networks. European Conference on Networks and Communications (EuCNC), June 2015, Paris, France.

[11] K. Mehdi, M. Lounis, A. Bounceur, and T. Kechadi. Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. 7th International Conference on Simulation Tools and Techniques (SIMUTools'14), Lisbon, Portugal, March 17-19, 2014.

[12] A. S. Wander, N. Gura, H. Eberle, V. Gupta, S. C. Shantz, Energy analysis of public-key cryptography for wireless sensor networks, in: Third IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE, 2005, pp. 324–328.