User manual- October 12th 2020

miranda

Starting

Thanks to have choose miranda, the universal simulation software.

This manual gives an outline of the use of miranda, for more detailed explanations on certain points, consult the YouTube channel miranda:

https://www.youtube.com/watch?v=6g-dHlJrTP0&list=PLHlYmTo8fUg5V4-r3tcjzq-2wqwhRtZlo

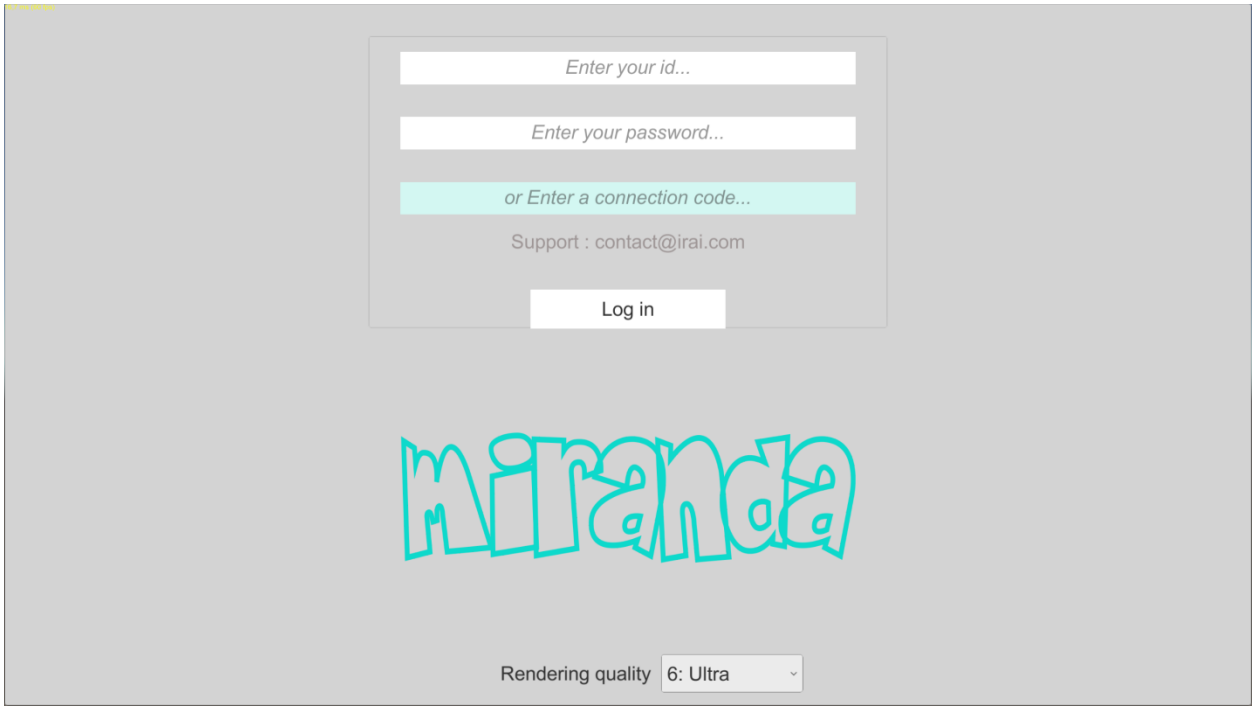
You can use miranda from a WebGL compatible browser or install an executable on Windows. For this, go to the website:

miranda.software

And choose "Connection" ...

From the home screen ...

Choose the language if necessary, by clicking at the bottom of the screen, then click in the middle of the screen ...

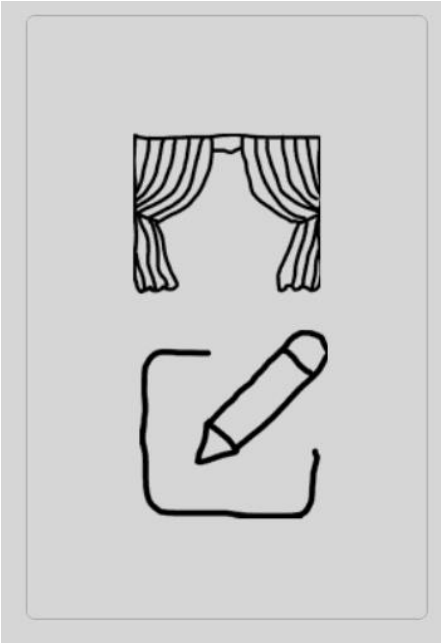


Choose the display quality, enter your username and password (sent by your supplier), then click on "Log in" ...

By moving the mouse over the different elements an explanatory text is displayed ...

Scenes

The scene editor ...



3d navigation

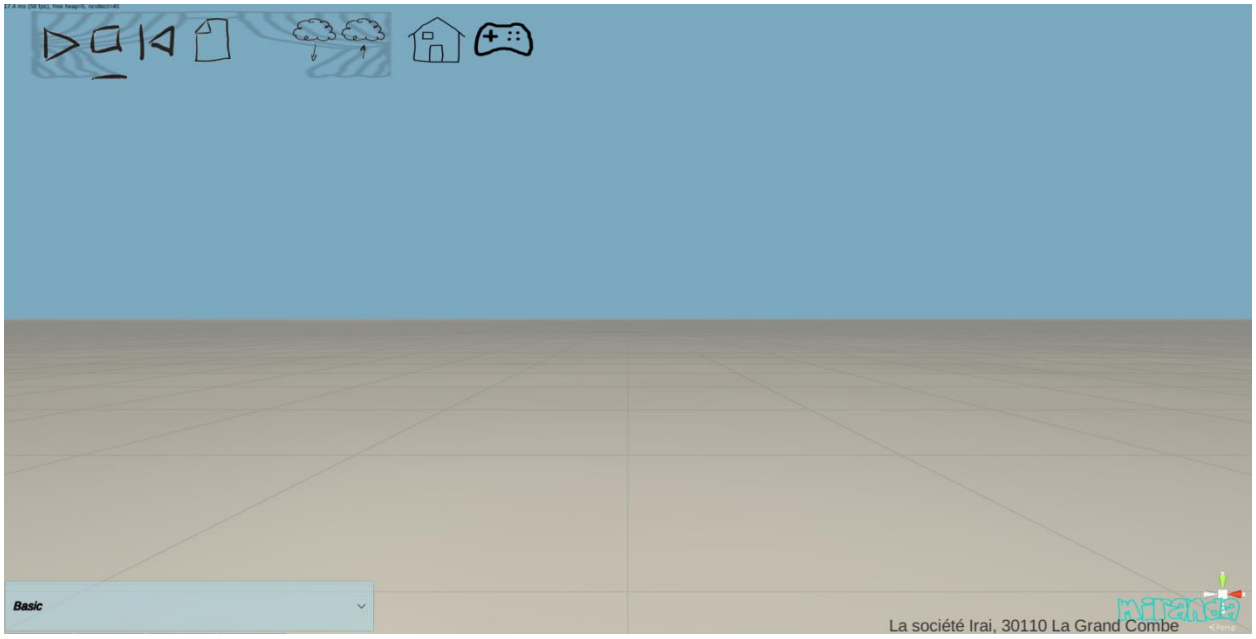
With the mouse:
-Turn the wheel = zoom,
-Right button pressed + move = rotation,
-Wheel pressed + move = translation.

With a touch screen:

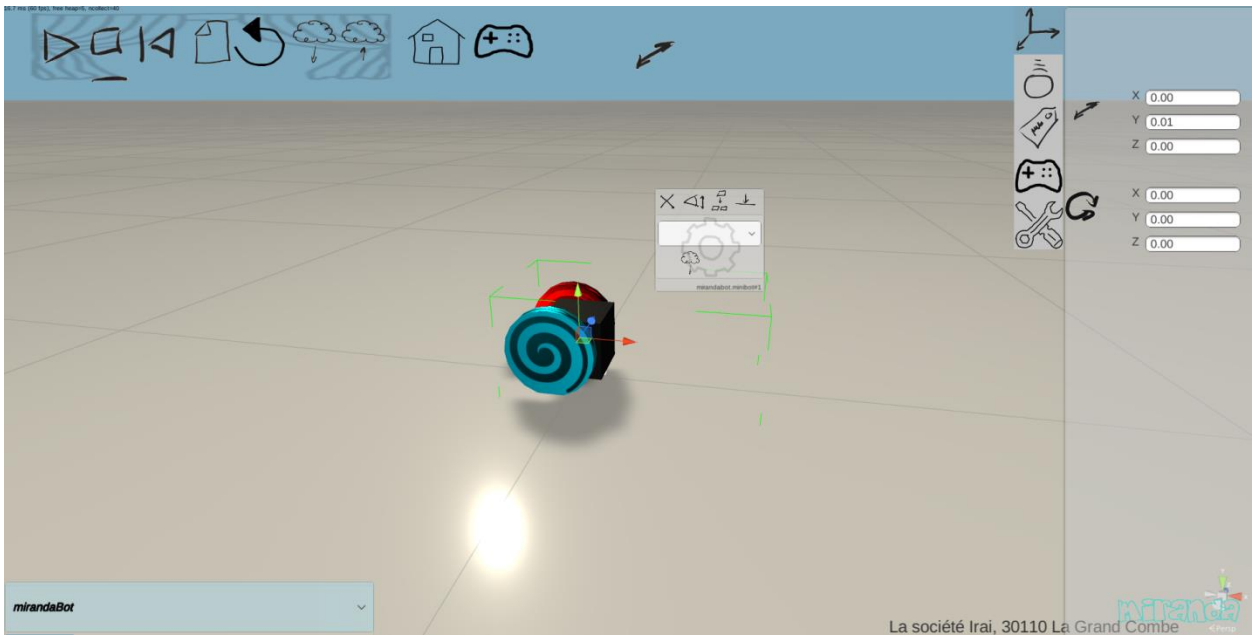
-Two fingers = rotation,
-Three fingers = translation.

Keyboard:

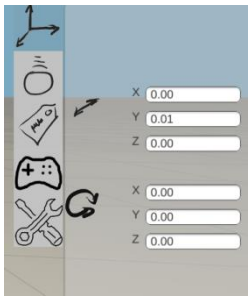
- Alt + arrow keys = move.



At the bottom left is the library of objects categorized. To add an object to the scene, choose it in this library.

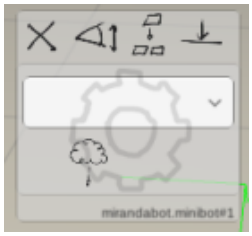


When an object is selected, its properties are displayed at the top left in the form of tabs:



Reminder: by moving the mouse over the different elements, an explanatory text is displayed.

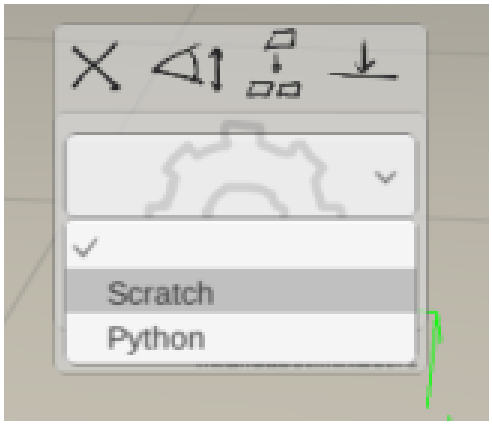
A window...



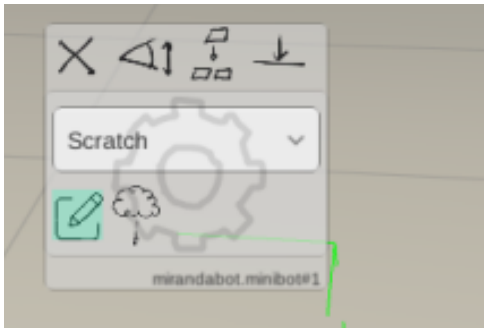
... is also displayed near the selected objects and allows you to carry out common operations: deletion, zoom, duplication, ...

This window also gives access to the program associated with each object.

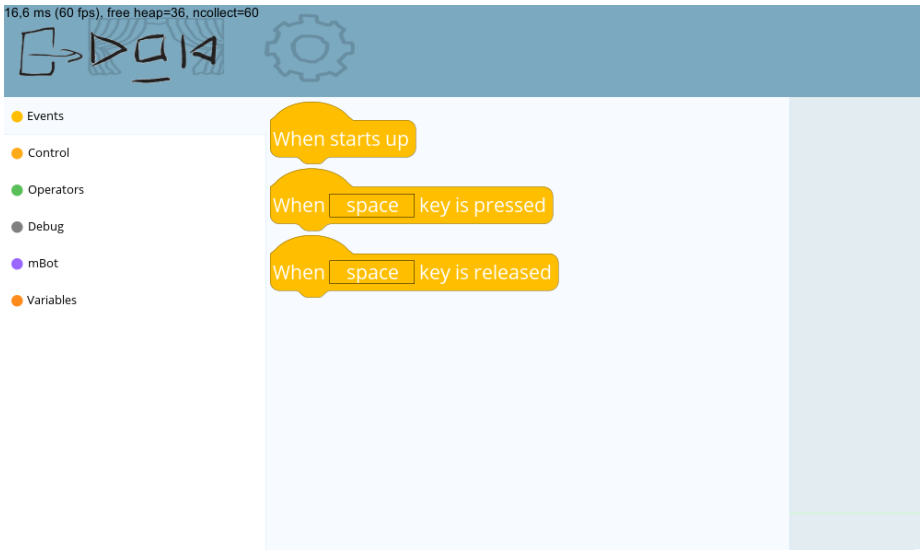
For example, to associate and edit a program in Scratch, click on:



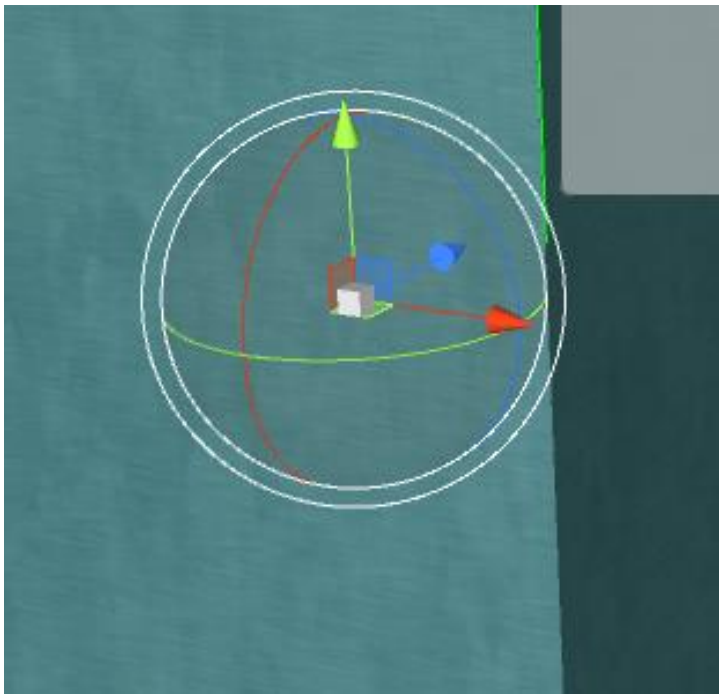
then on



...



The position, rotation or scale of the selected objects can also be modified with the "Gizmo" appearing next to the objects ...



... by grabbing the arrows, the object moves, grabbing the circles, the object rotates, grabbing the white square in the center of the Gizmo, the size is changed.

The Gizmo icon ...

... at the top of the miranda window allows you to change the type of Gizmo, going for example from a universal Gizmo type to a Gizmo allowing only the size change.

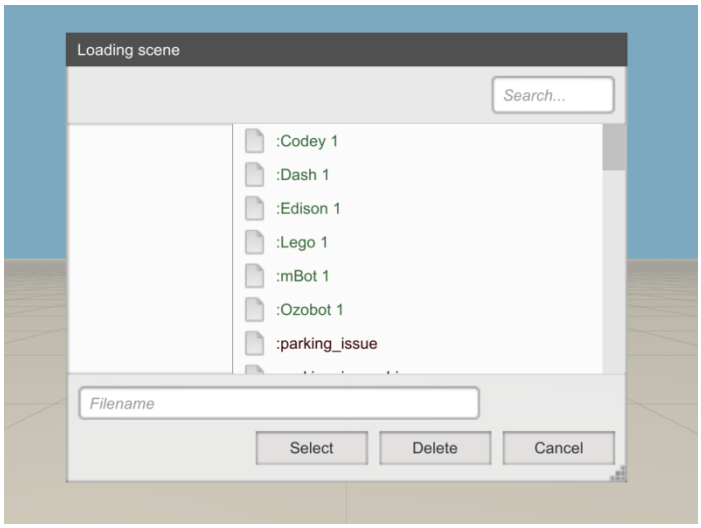
The simulation can be started, stopped and reset using these icons:

In simulation mode, if a system is selected, the "Properties, I/Os" tab ...

... provides access to dynamic visualization of system elements (motors, sensors, ...). By clicking on ...

... it is possible to manually control the output elements (manually activate the motors for example).

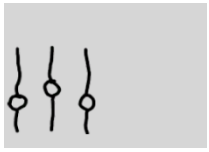
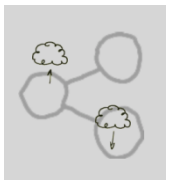
Scenes can be saved and reloaded ...



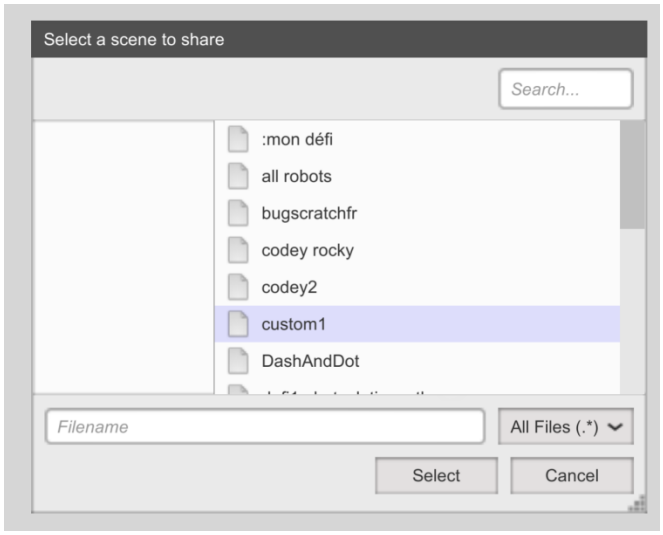
... backups are associated with your customer account in the cloud.

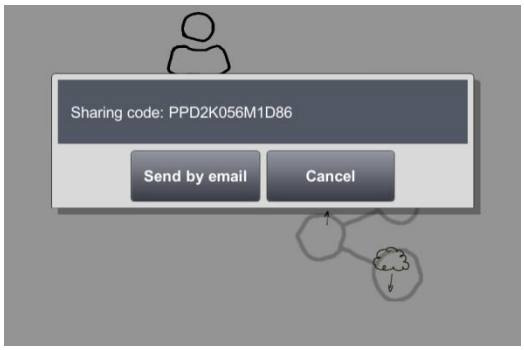
Share a scene

The scene sharing...

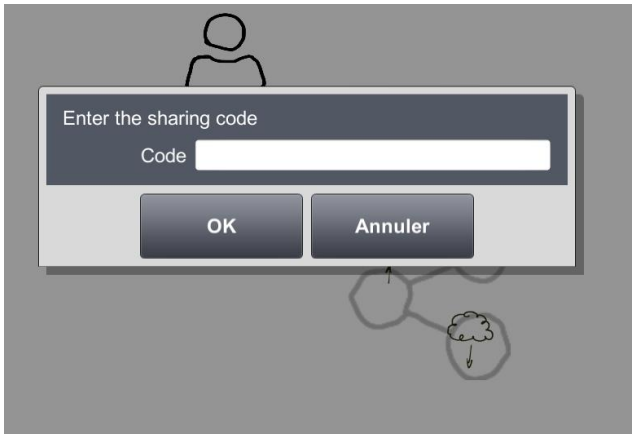
 ...  ... permits to an user to share a scene (Eg. a challenge) with another miranda user.

To send a sharing to another user...  ...





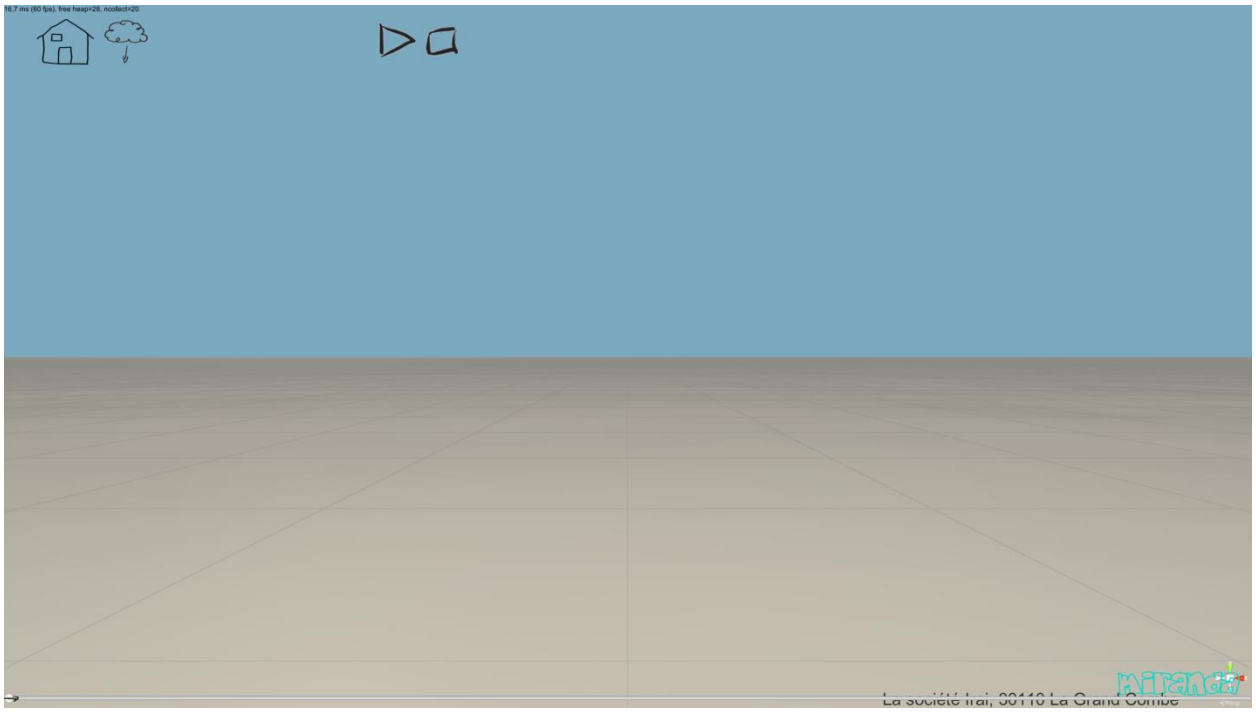
A same code can be shared with several users.

To receive a sharing...  ... 

Player mode

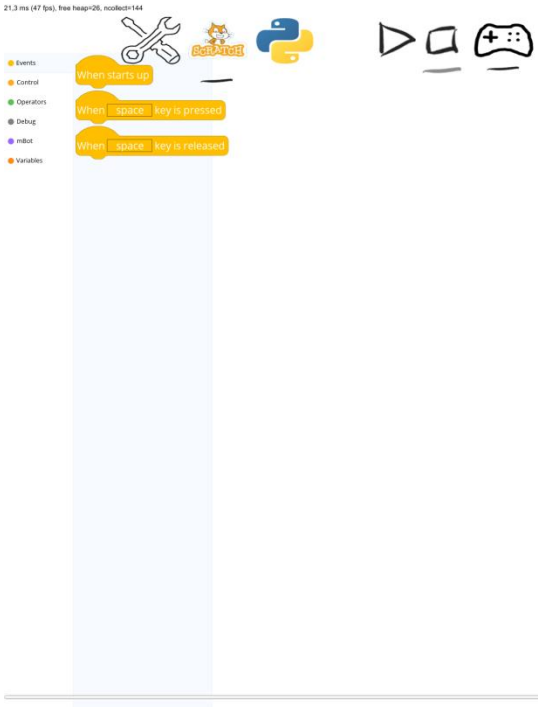
Player mode ...

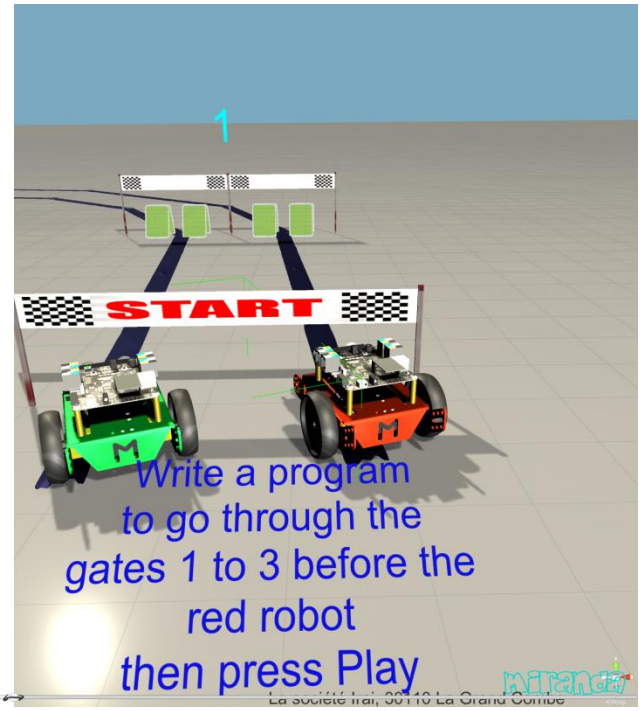
... is a mode limited to simulating scenes ...



This mode is typically intended for using challenges without the possibility of modifying them.

Example of challenge ...



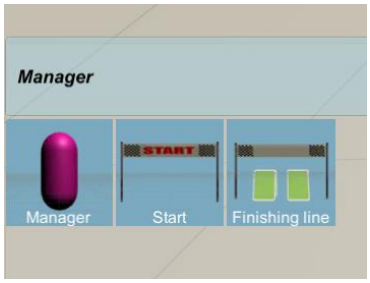


Challenges

For a scene to appear as a challenge usable in player mode, its name must start with the character ":".

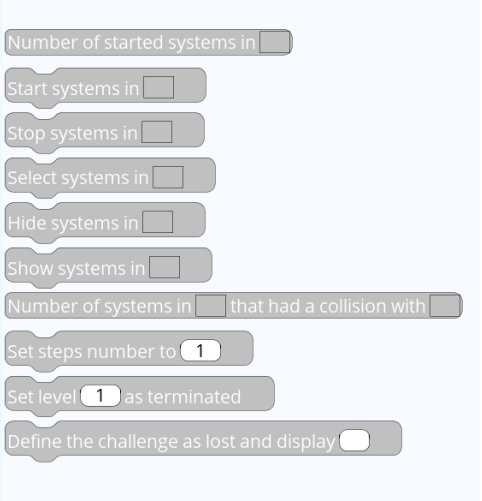
Miranda allows you to use the challenges provided with the software and also to create your own challenges.

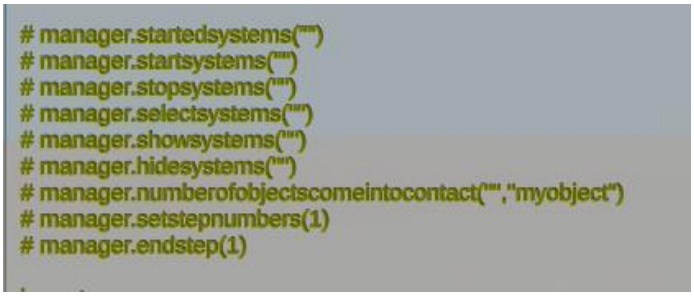
The "manager" object ...



... from the library is the central part of a challenge. This programmable object in Scratch or Python allows you to define the progress of the challenge. The "Finish Line" object can also be used to check for the presence of an object in a certain location. A challenge can consist of several stages.

List of programming elements of the "Manager" object ...

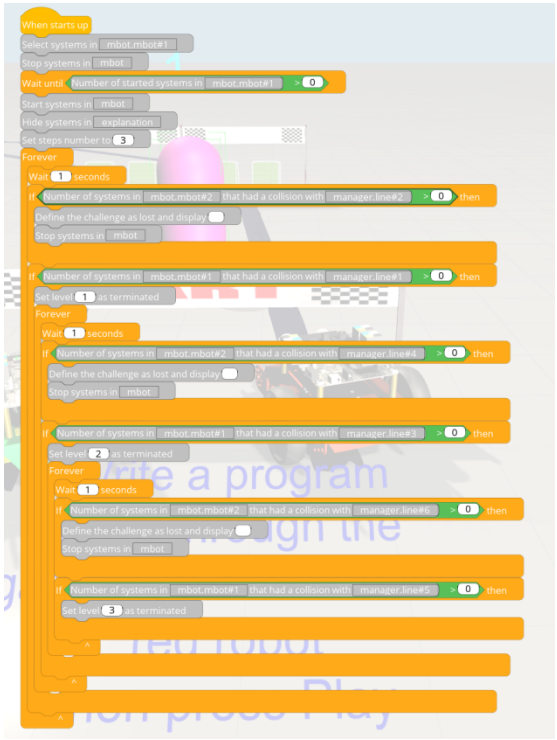




Example of use of the Manager object programmed in Python (challenge 1 mBot, for this challenge, the object mbot.mbot # 1 is the green robot programmed by the user of the challenge, mbot.mbot # 2 is the one computer-controlled red robot):

```
import time
import manager

manager.selectsystems("mbot.mbot#1")  Select the green robot
manager.stopsystems("mbot")  Stop all robots
while manager.startedsystems("mbot.mbot#1")==0:  Wait that the user starts the green robot
    continue
manager.startsystems("mbot")  Start all robots (donc aussi le robot piloté par l'ordinateur)
manager.hidesystems("explanation")  Hide explanations
manager.setstepnumbers(3)  Challenge has 3
while True:  The monitor loop for step #1
    time.sleep(1)  Monitor each second
    if manager.numberofobjectscomeintocontact("mbot.mbot#2","manager.line#2")>0:  The red robot has reached the finish line for step #1?
        manager.abort("")  Lost : stop robots
        manager.stopsystems("mbot")
    if manager.numberofobjectscomeintocontact("mbot.mbot#1","manager.line#1")>0:  The green robot has reached finish line for step#1 ?
        manager.endstep(1)  Step 1 ended, then monitor step #2 then step#3
        while True:
            time.sleep(1)
            if manager.numberofobjectscomeintocontact("mbot.mbot#2","manager.line#4")>0:
                manager.abort("")
                manager.stopsystems("mbot")
            if manager.numberofobjectscomeintocontact("mbot.mbot#1","manager.line#3")>0:
                manager.endstep(2)
                while True:
                    time.sleep(1)
                    if manager.numberofobjectscomeintocontact("mbot.mbot#2","manager.line#6")>0:
                        manager.abort("")
                        manager.stopsystems("mbot")
                    if manager.numberofobjectscomeintocontact("mbot.mbot#1","manager.line#5")>0:
                manager.endstep(3)
```

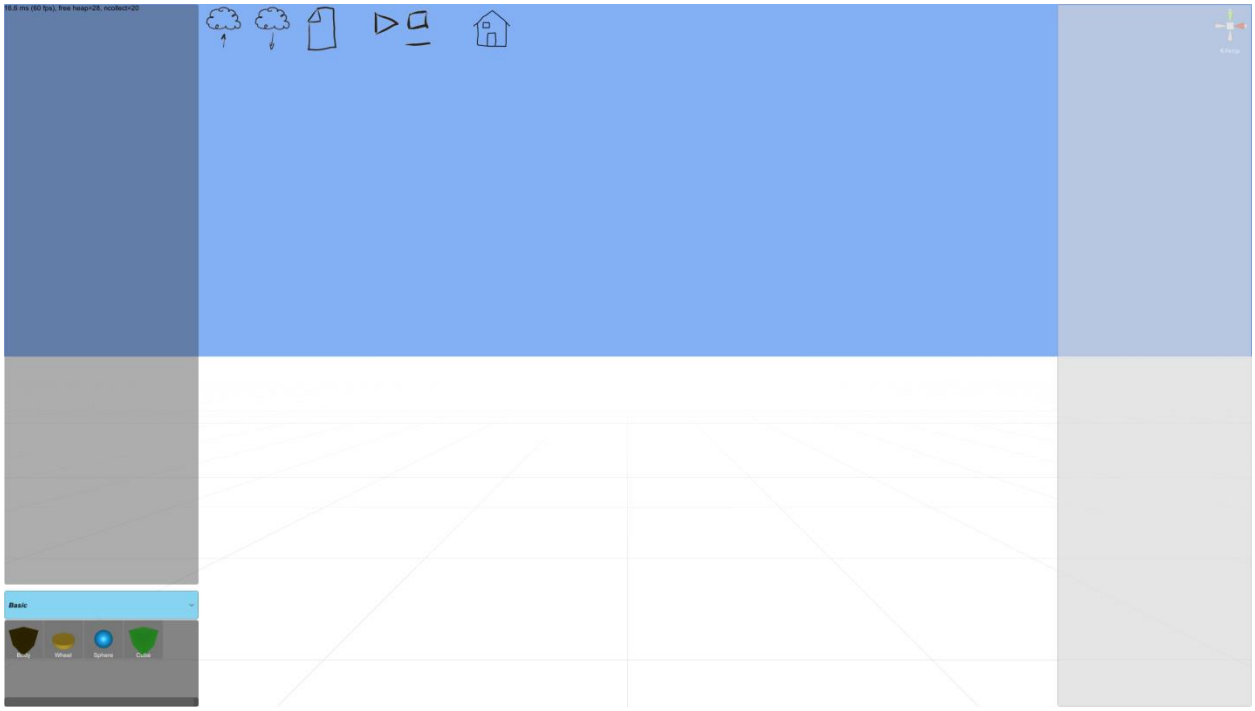
Same with Scratch :



Systems editor

The systems editor ...

... allows you to create your own systems (robots for example) usable in simulation scenes.



Systems are created by defining the physical structure, for example the body of a robot and then the wheels.

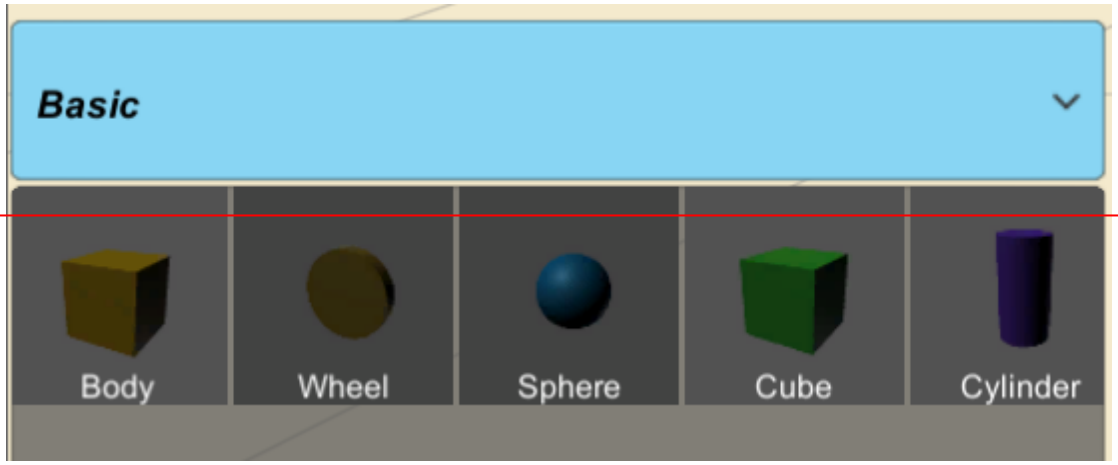
Other elements can also be added: sensors, leds.

Personalized geometries can finally be added as decorations to finalize the visual aspect of the robot.

Finally, the Scratch blocs associated to a device can be customized.

Some of the robots in the miranda library (Edison and Thymio for example) were created with the system editor and can be reopened in the editor to observe their internal structure and serve as a model for your own creations.

Items that can be used for the device creation are arranged in categories…



… one click on an item add it as child of the selected item.

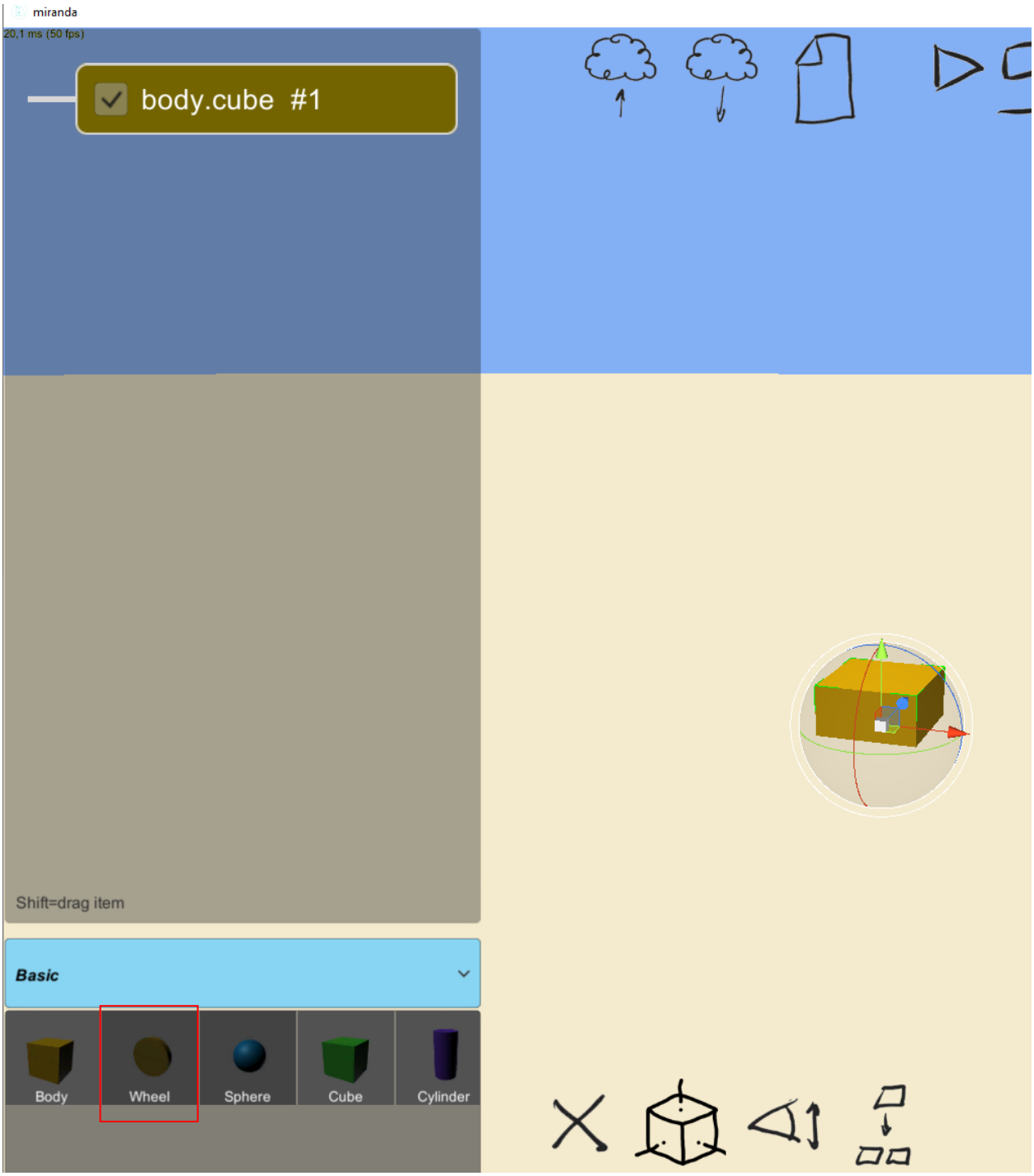
Categories are as follow:

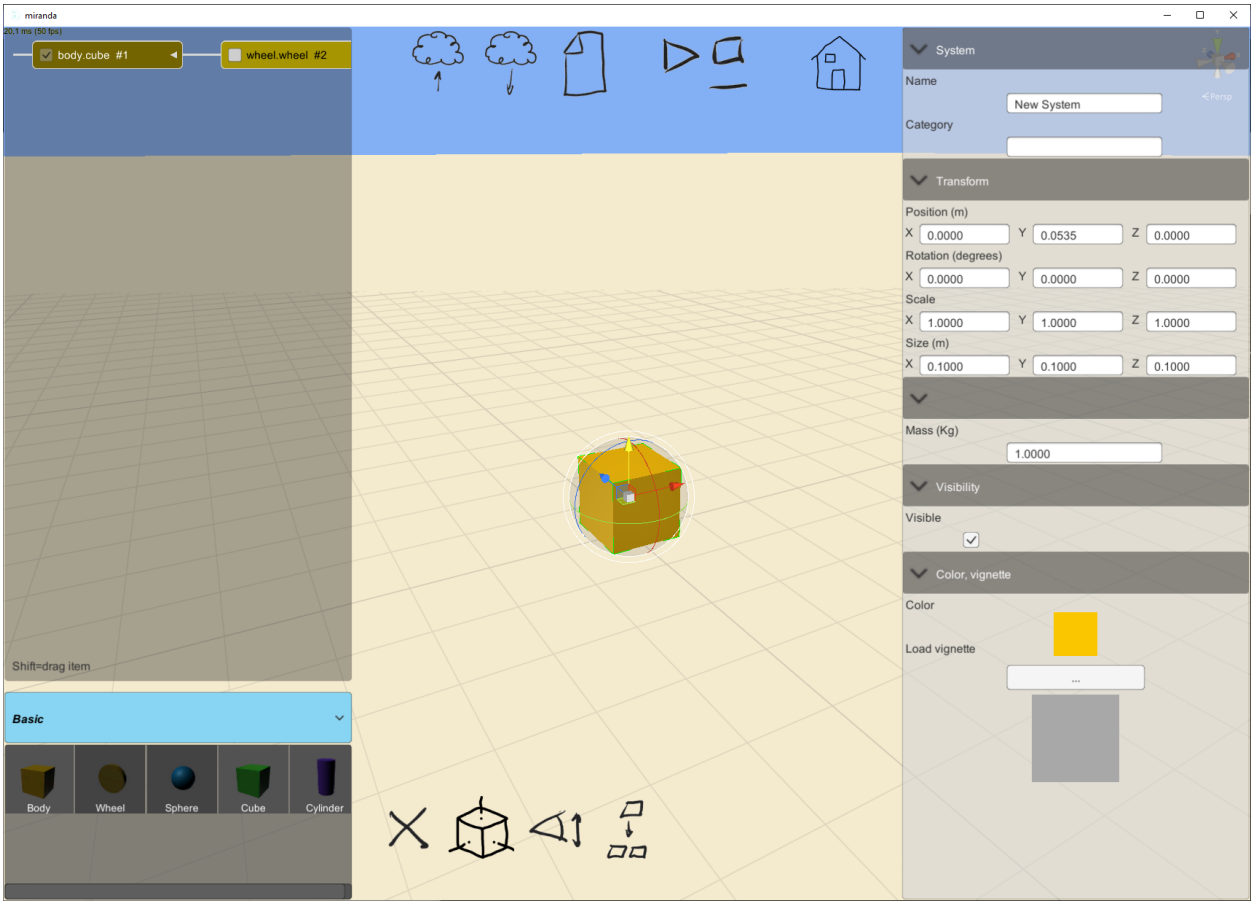
- Basic: body of a device and basic shapes,
- Custom geometries: to be used to import your own geometries from a ".glb" 3d file,
- Sensors: the different sensors,
- Leds: lighting items,
- Links: physic links,
- Sounds: for emitting a sound,
- Lights: for emitting a light,
- Other: all that do not belong to the other categories.

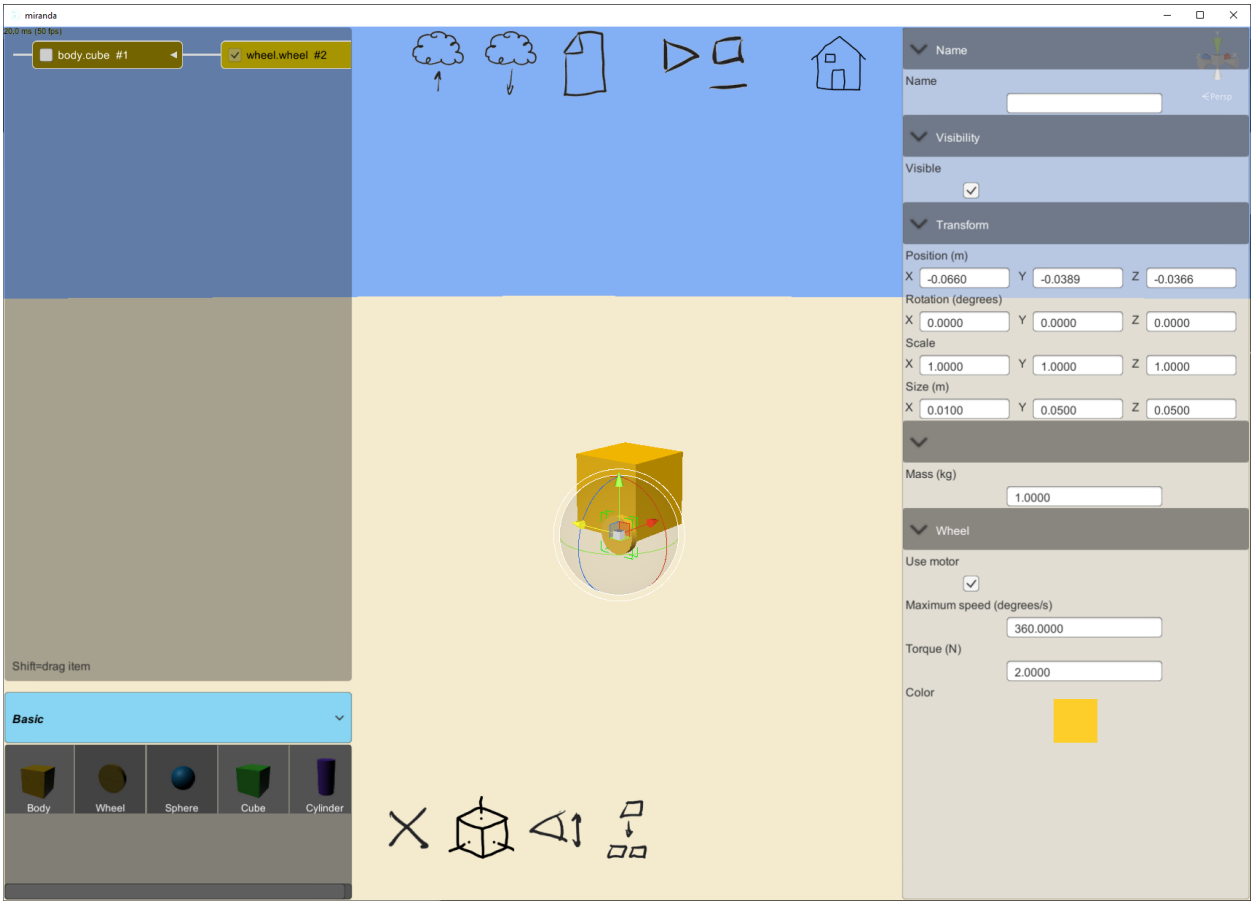
The following illustrates the creation of a device.

The different steps are only an example, some of them are optional and to be used only according to your needs.

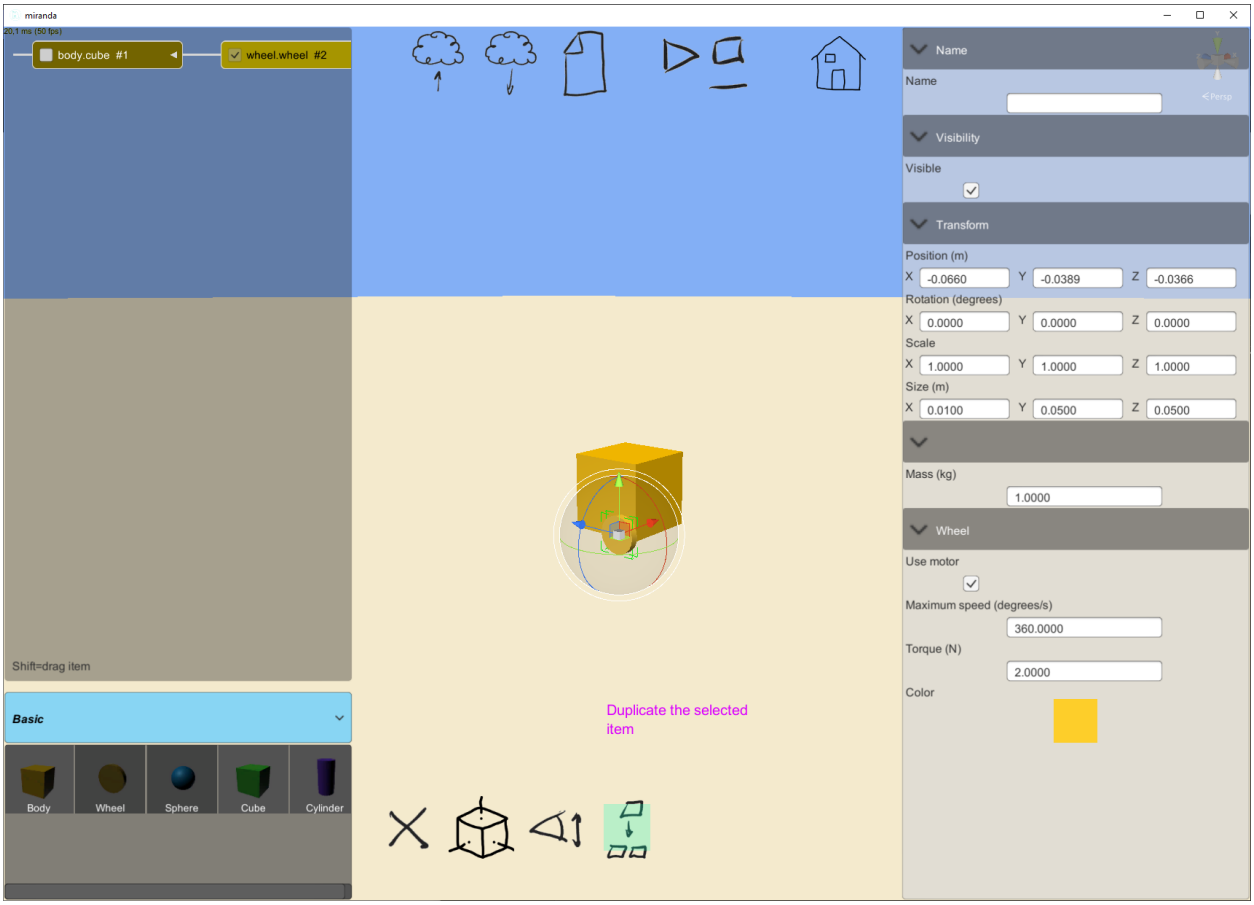
Creation of a minimalist robot…

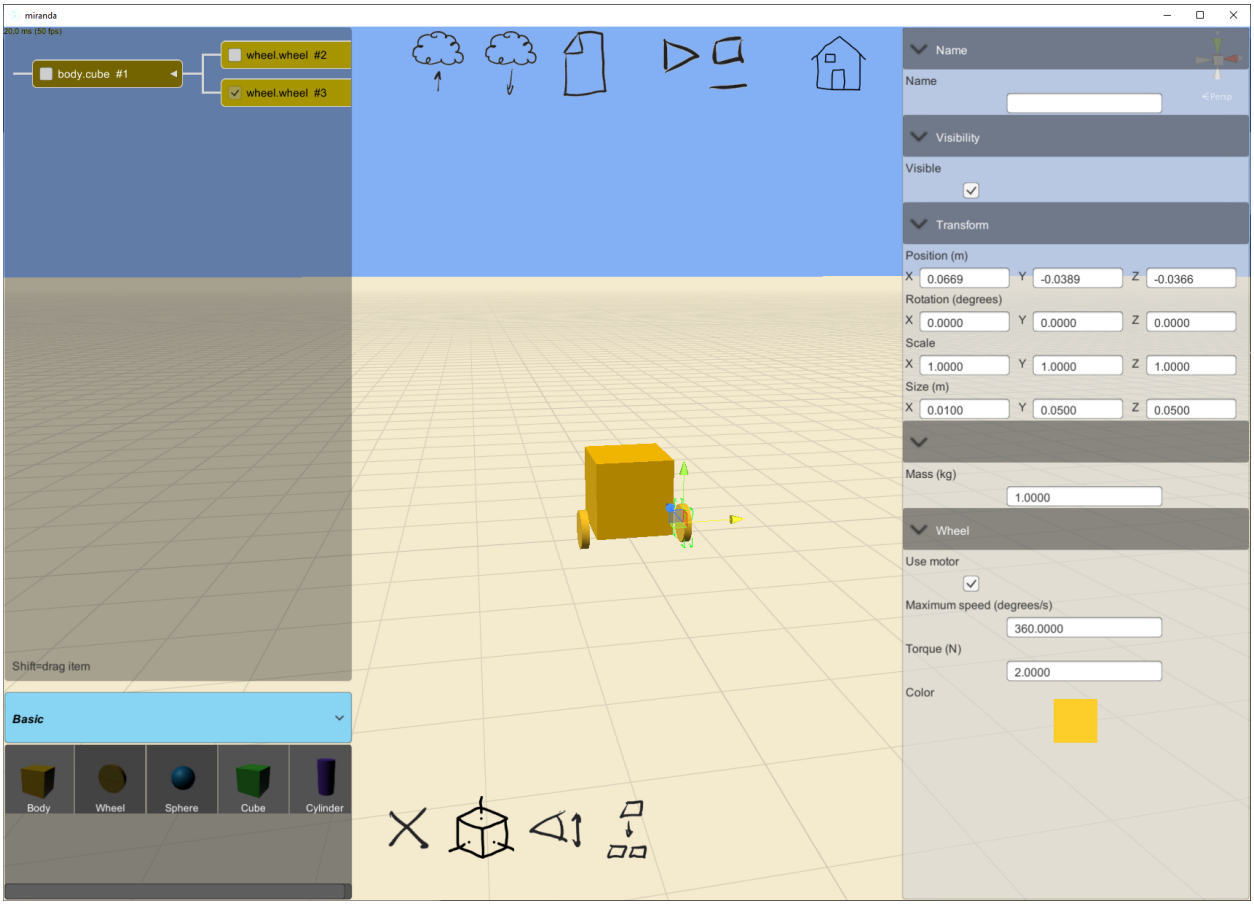

miranda
20,1 ms (50 fps)
body.cube #1
Shift=drag item
Basic
Body
Wheel
Sphere
Cube
Cylinder


miranda
body.cube #1
wheel.wheel #2
System
Name
New System
Category
Transform
Position (m)
X 0.0000 Y 0.0535 Z 0.0000
Rotation (degrees)
X 0.0000 Y 0.0000 Z 0.0000
Scale
X 1.0000 Y 1.0000 Z 1.0000
Size (m)
X 0.1000 Y 0.1000 Z 0.1000
Mass (Kg)
1.0000
Visibility
Visible
Color, vignette
Color
Load vignette
Shift=drag item
Basic
Body
Wheel
Sphere
Cube
Cylinder

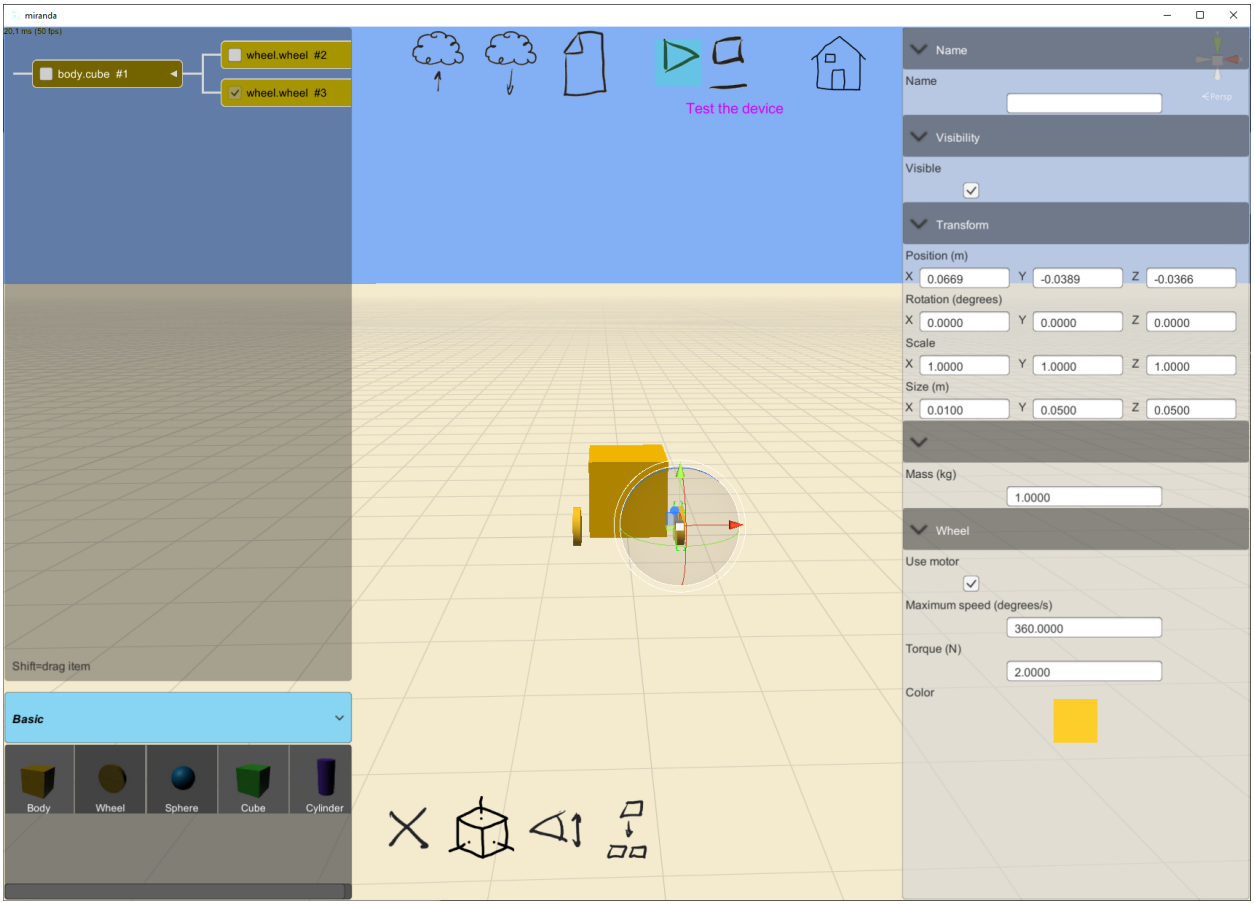

miranda
body.cube #1
wheel.wheel #2
Shift=drag item
Basic
Body
Wheel
Sphere
Cube
Cylinder
Name
Name
Visibility
Visible
Transform
Position (m)
X -0.0660 Y -0.0389 Z -0.0366
Rotation (degrees)
X 0.0000 Y 0.0000 Z 0.0000
Scale
X 1.0000 Y 1.0000 Z 1.0000
Size (m)
X 0.0100 Y 0.0500 Z 0.0500
Mass (kg)
1.0000
Wheel
Use motor
Maximum speed (degrees/s)
360.0000
Torque (N)
2.0000
Color

miranda


miranda
body.cube #1
wheel.wheel #2
Name
Visibility
Visible
Transform
Position (m)
Rotation (degrees)
Scale
Size (m)
Mass (kg)
1.0000
Wheel
Use motor
Maximum speed (degrees/s)
360.0000
Torque (N)
2.0000
Color
Shift=drag item
Basic
Body
Wheel
Sphere
Cube
Cylinder
Duplicate the selected item

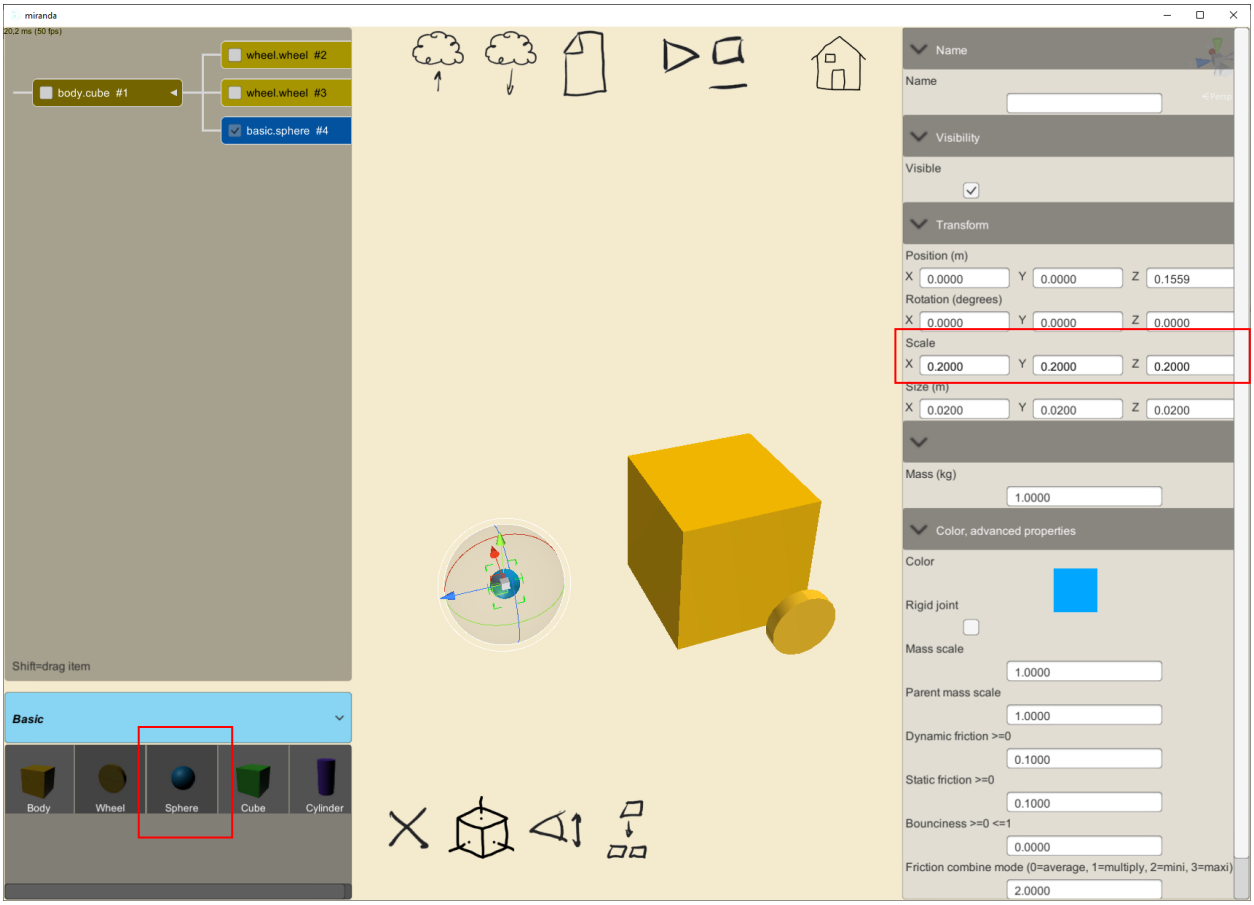

miranda
body.cube #1
wheel.wheel #2
wheel.wheel #3
Shift=drag item
Basic
Body
Wheel
Sphere
Cube
Cylinder
Name
Name
Visibility
Visible
Transform
Position (m)
X 0.0669 Y -0.0389 Z -0.0366
Rotation (degrees)
X 0.0000 Y 0.0000 Z 0.0000
Scale
X 1.0000 Y 1.0000 Z 1.0000
Size (m)
X 0.0100 Y 0.0500 Z 0.0500
Mass (kg)
1.0000
Wheel
Use motor
Maximum speed (degrees/s)
360.0000
Torque (N)
2.0000
Color

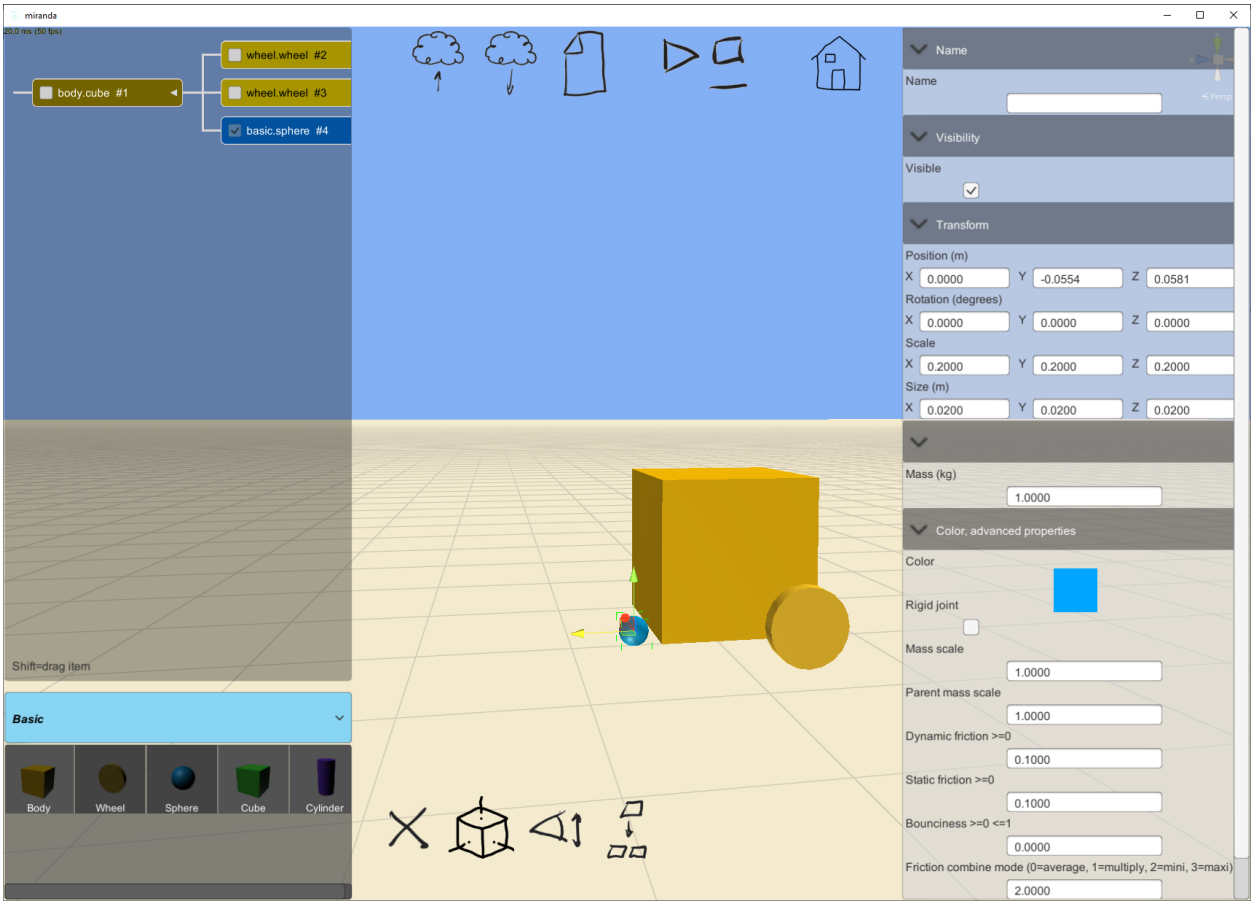
miranda



… by default, the wheels are motorized, so the system is functional and the robot moves forward.

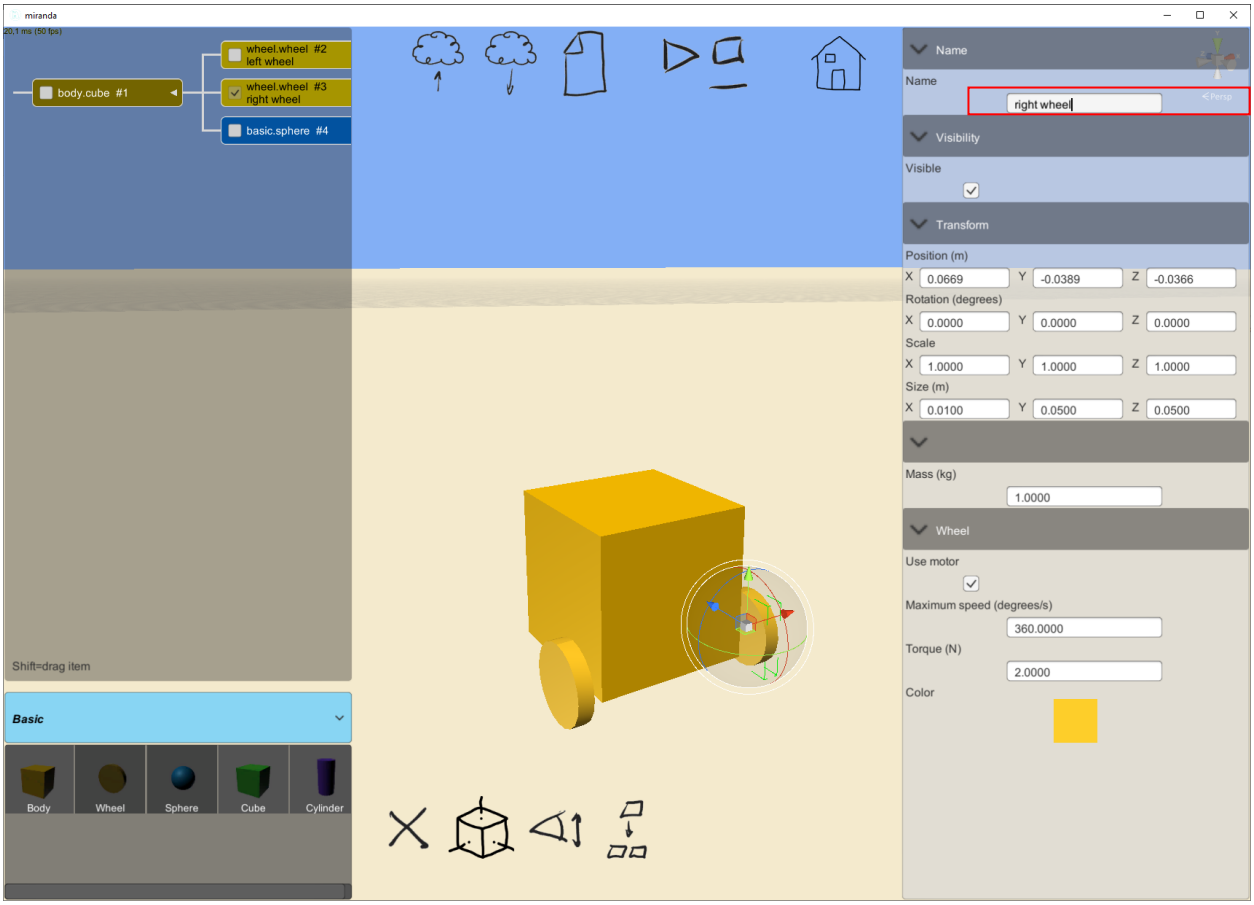
Let’s enrich our model, add a sphere as a third point of support ...


miranda
20,2 ms (50 fps)
wheel.wheel #2
body.cube #1
wheel.wheel #3
basic.sphere #4
Shift=drag item
Basic
Body
Wheel
Sphere
Cube
Cylinder
Name
Name
Visibility
Visible
Transform
Position (m)
X 0.0000 Y 0.0000 Z 0.1559
Rotation (degrees)
X 0.0000 Y 0.0000 Z 0.0000
Scale
X 0.2000 Y 0.2000 Z 0.2000
Size (m)
X 0.0200 Y 0.0200 Z 0.0200
Mass (kg)
1.0000
Color, advanced properties
Color
Rigid joint
Mass scale
1.0000
Parent mass scale
1.0000
Dynamic friction >=0
0.1000
Static friction >=0
0.1000
Bounciness >=0 <=1
0.0000
Friction combine mode (0=average, 1=multiply, 2=mini, 3=maxi)
2.0000

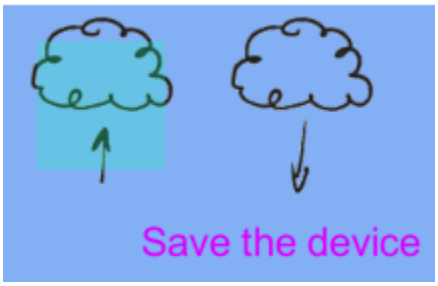


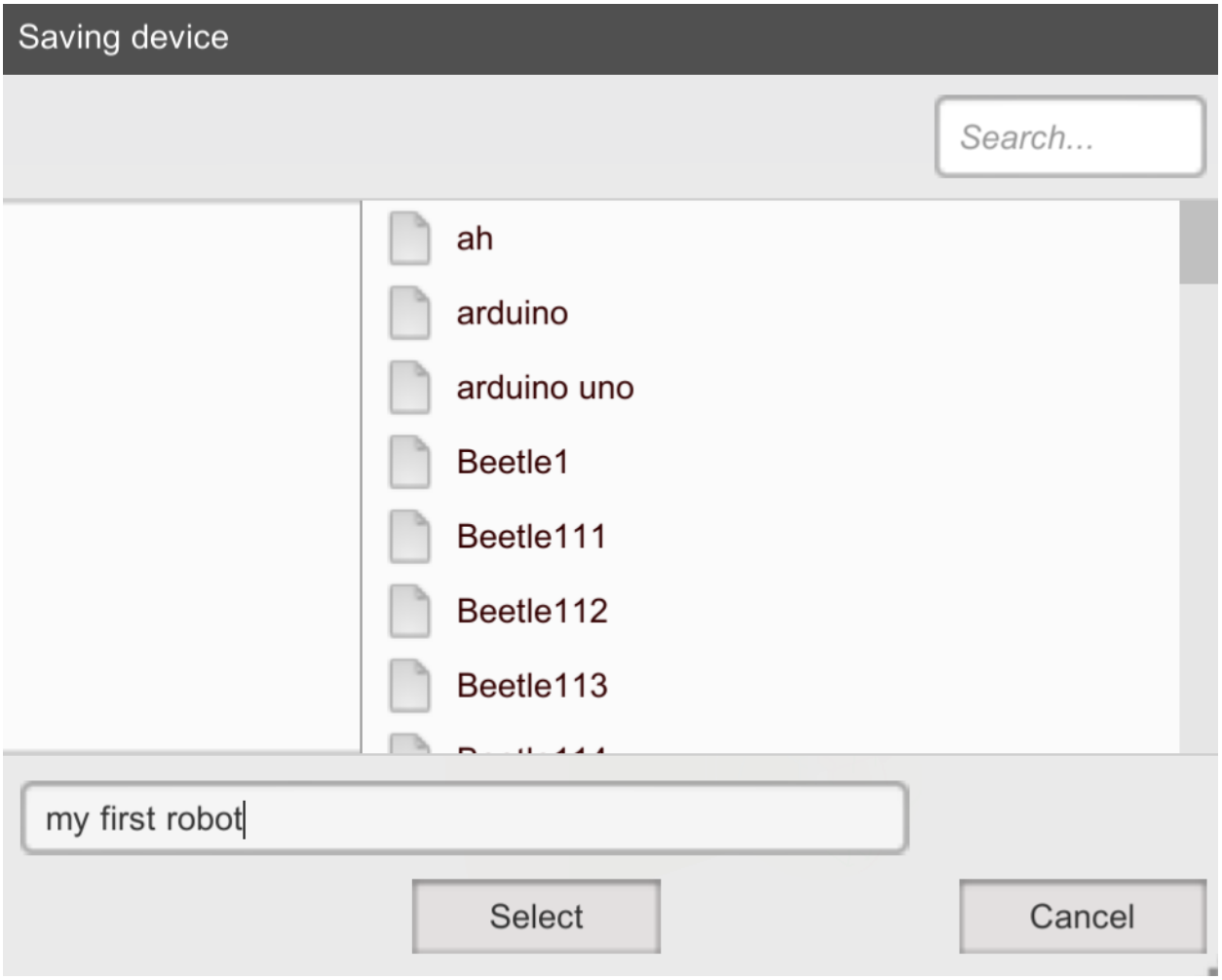
… By default, the items automatically have a physical joint with their parent.

Let's see how to use our system in the scenes editor. To begin with, let's name the items so that we can identify them when it comes to driving them ...

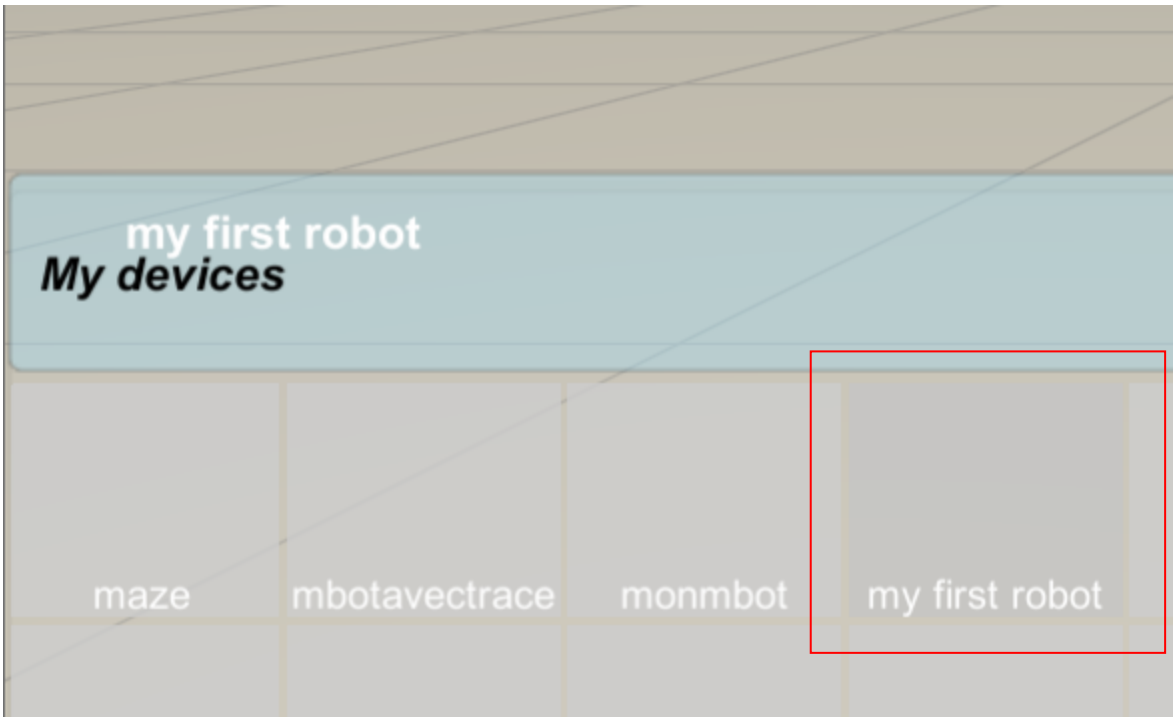


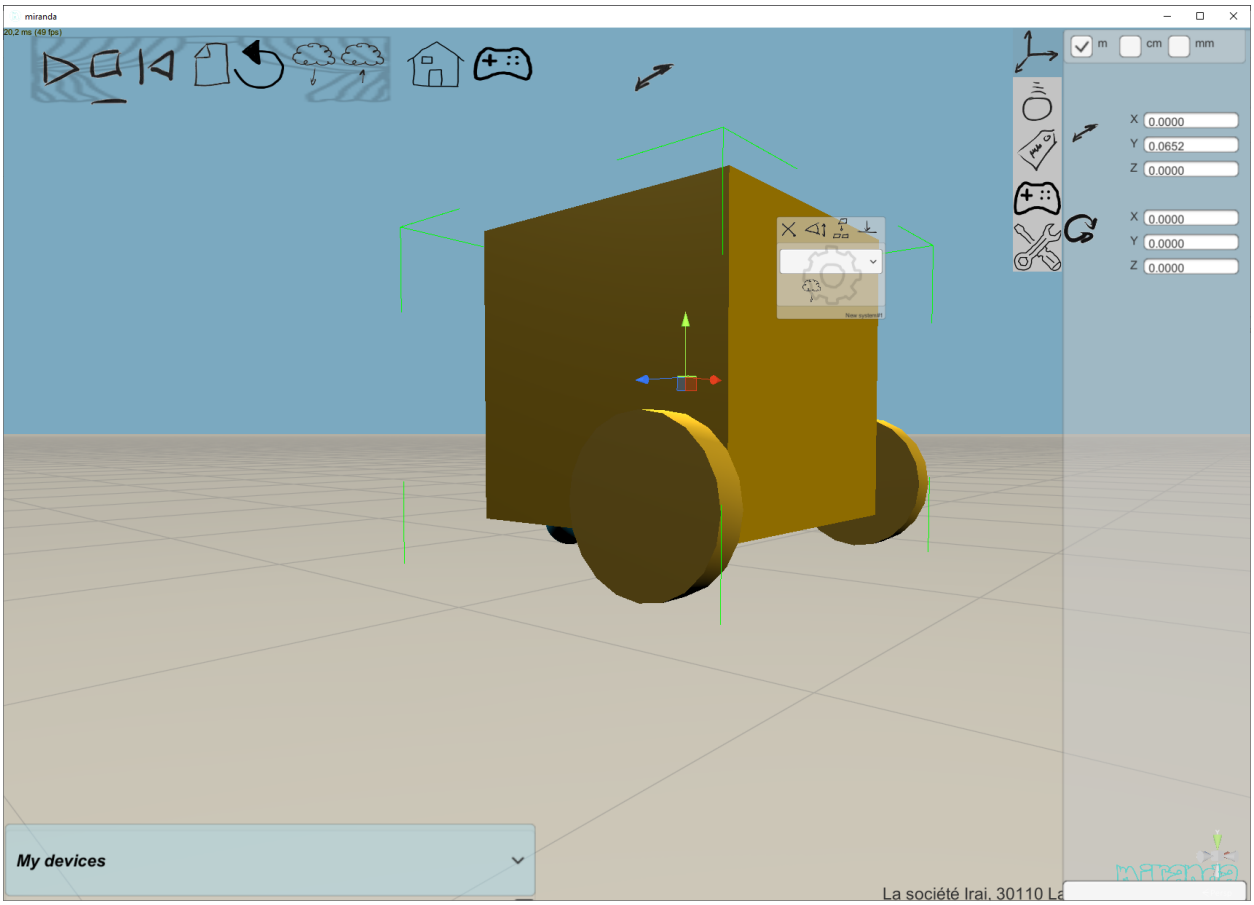
… then let's save our device…



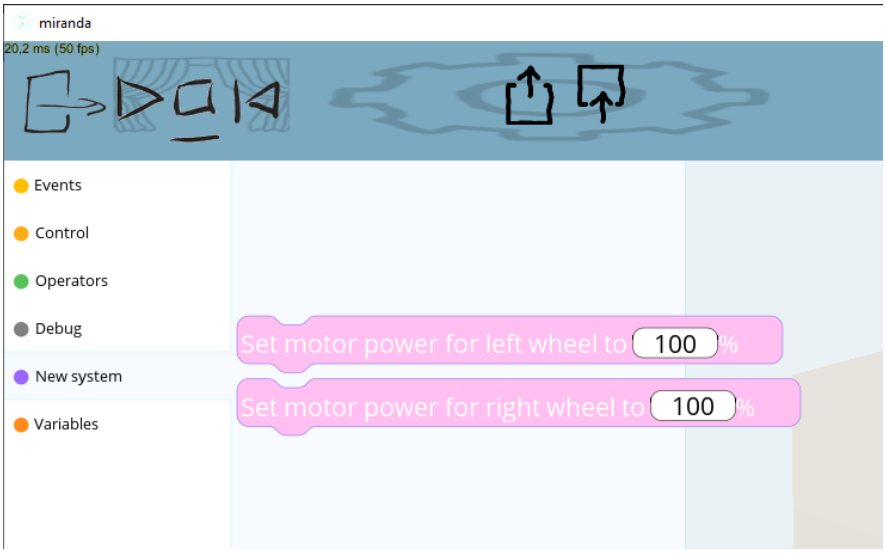


… now from the scenes editor…

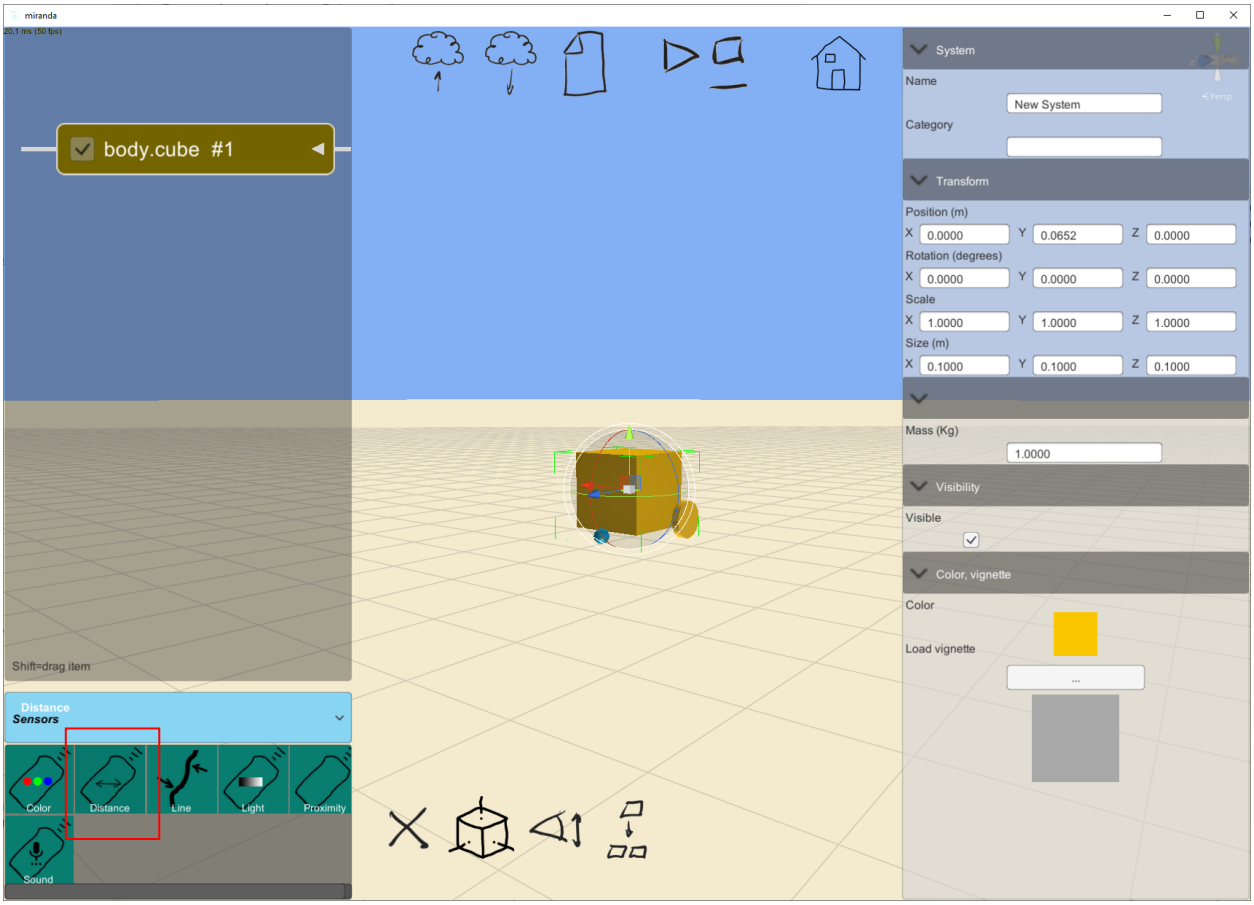

my first robot
My devices
maze
mbotavectrace
monmbot
my first robot


miranda
m
cm
mm
X 0.0000
Y 0.0652
Z 0.0000
X 0.0000
Y 0.0000
Z 0.0000
My devices

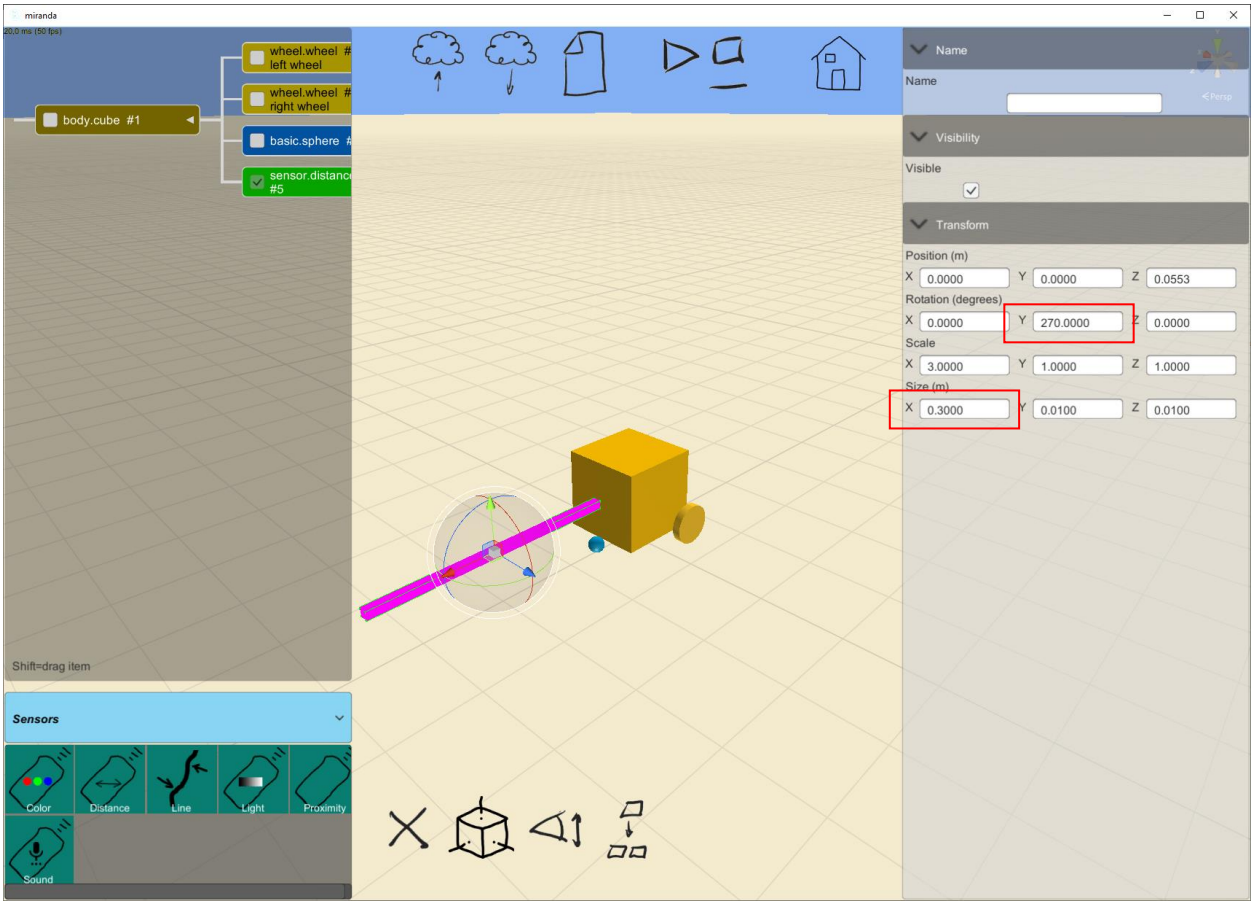
… the functional items, here only the wheels, are usable from the Scratch or Python programing…



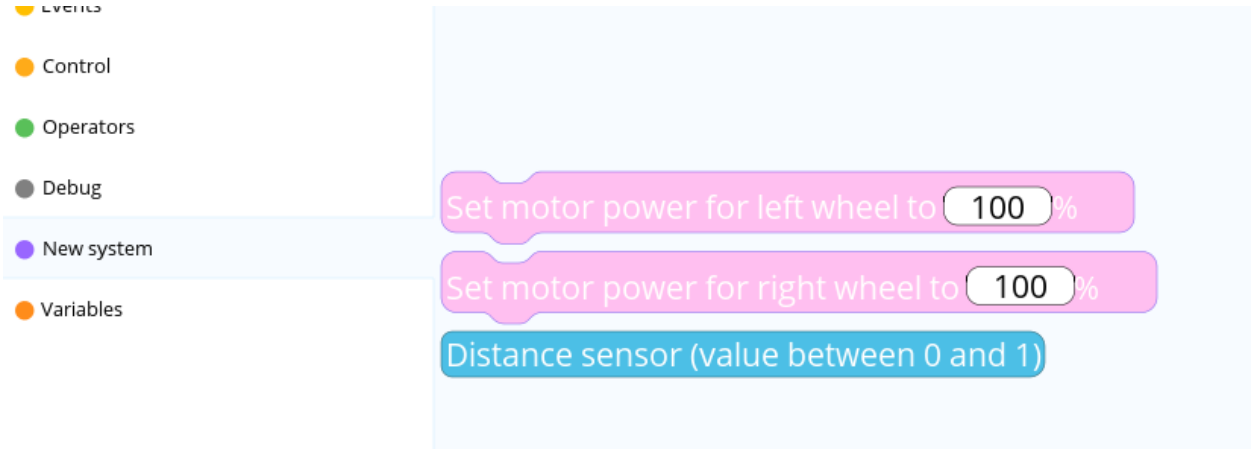
… Let's go back to complete our model by adding a distance sensor…



... let's say we wanted to get a distance sensor operating at a maximum distance of 30 cm in front of the robot ...

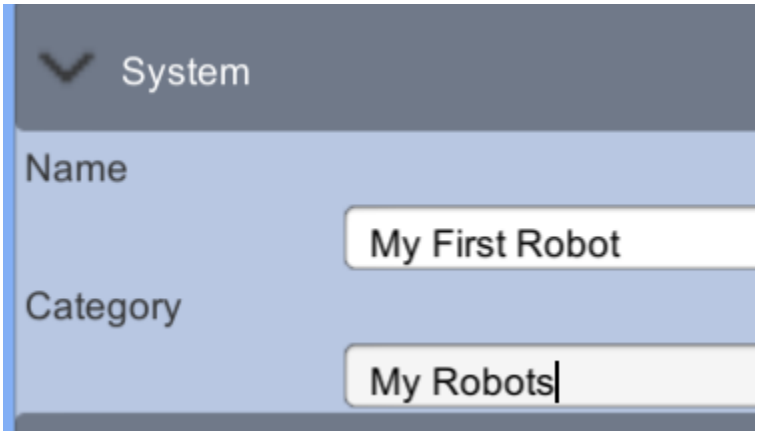


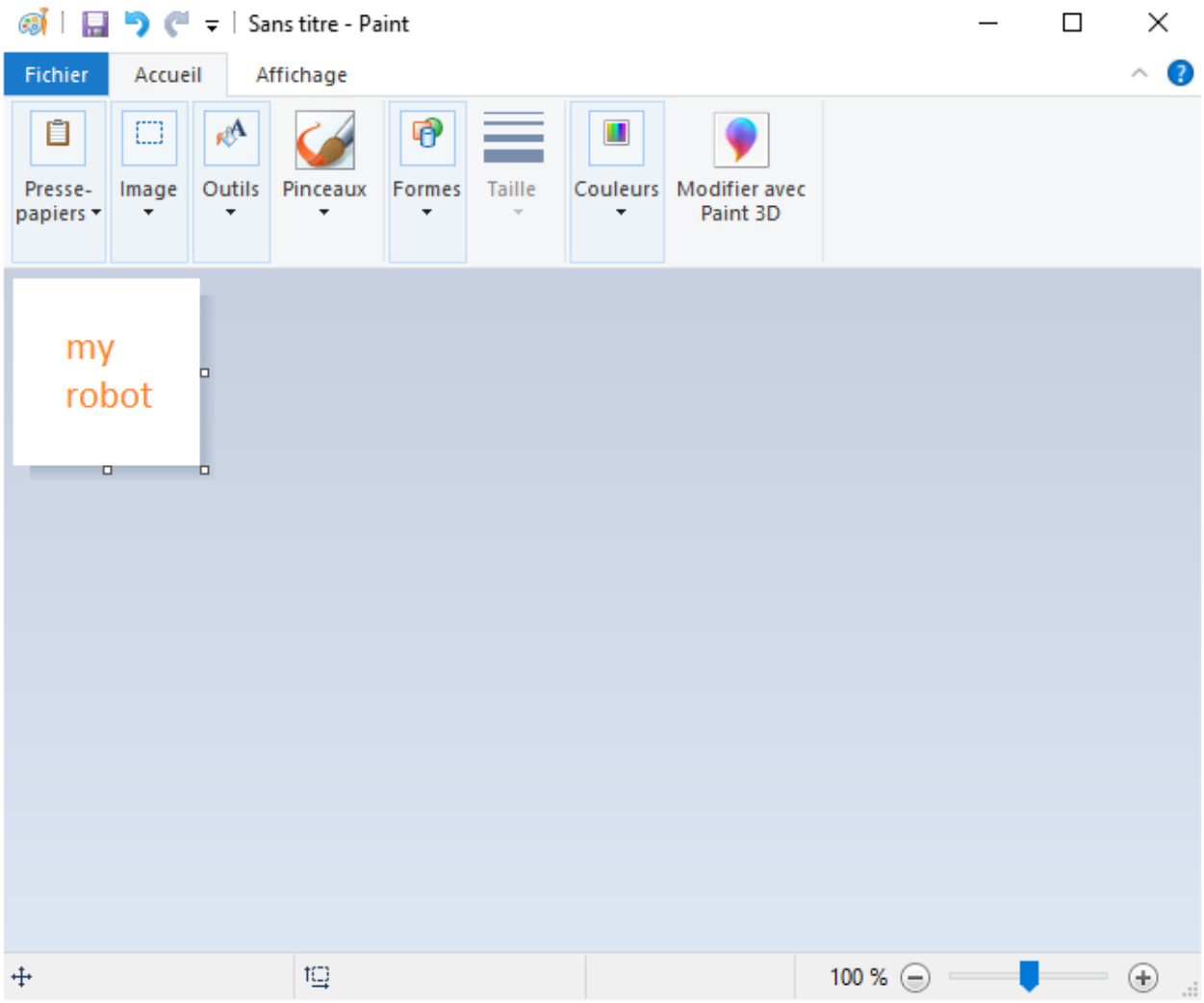
... The "Visible" checkbox allows you to hide the geometry associated with the detection zone. Scratch blocks and "default" Python functions are automatically accessible in programming mode ...



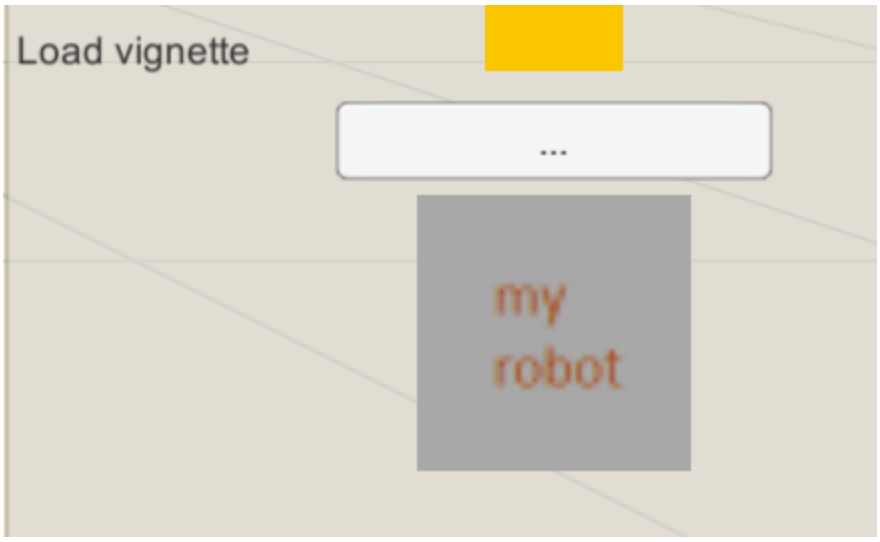
```
#Python script for New system

# newsystem.leftwheel.setpower(<power (from -100 to 100)>)
# newsystem.rightwheel.setpower(<power (from -100 to 100)>)
# newsystem.distancesensor()

import newsystem
```

… Let's continue the customization of our system by defining the category in which our robot must appear as well as its name and a personalized icon…

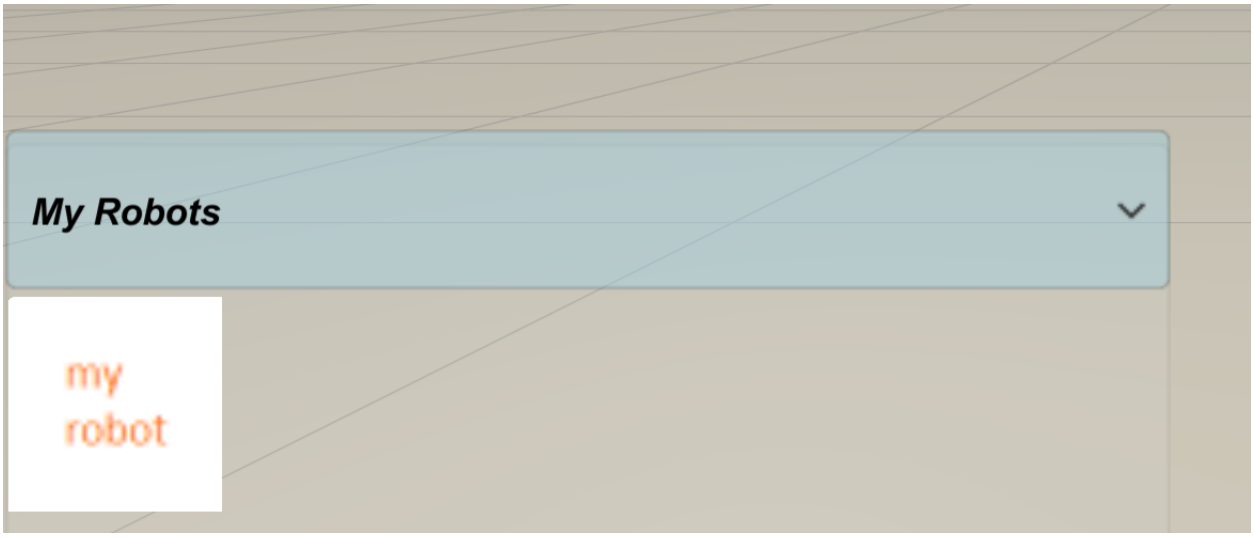



Sans titre - Paint
Fichier
Accueil
Affichage
Presse-papiers
Image
Outils
Pinceaux
Formes
Taille
Couleurs
Modifier avec Paint 3D
my robot
100 %

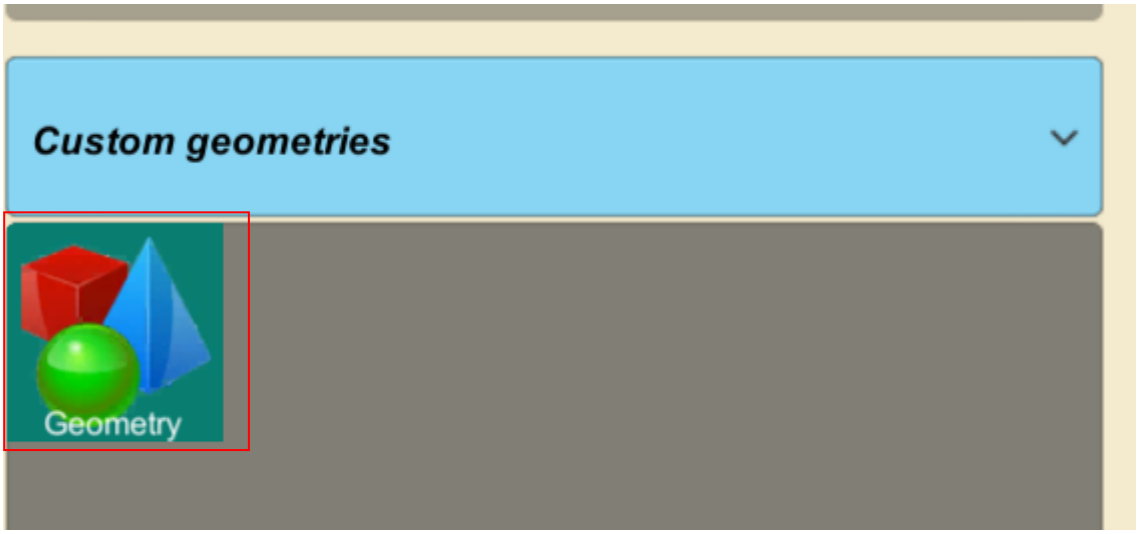

Color, vignette
Color
Load vignette
...



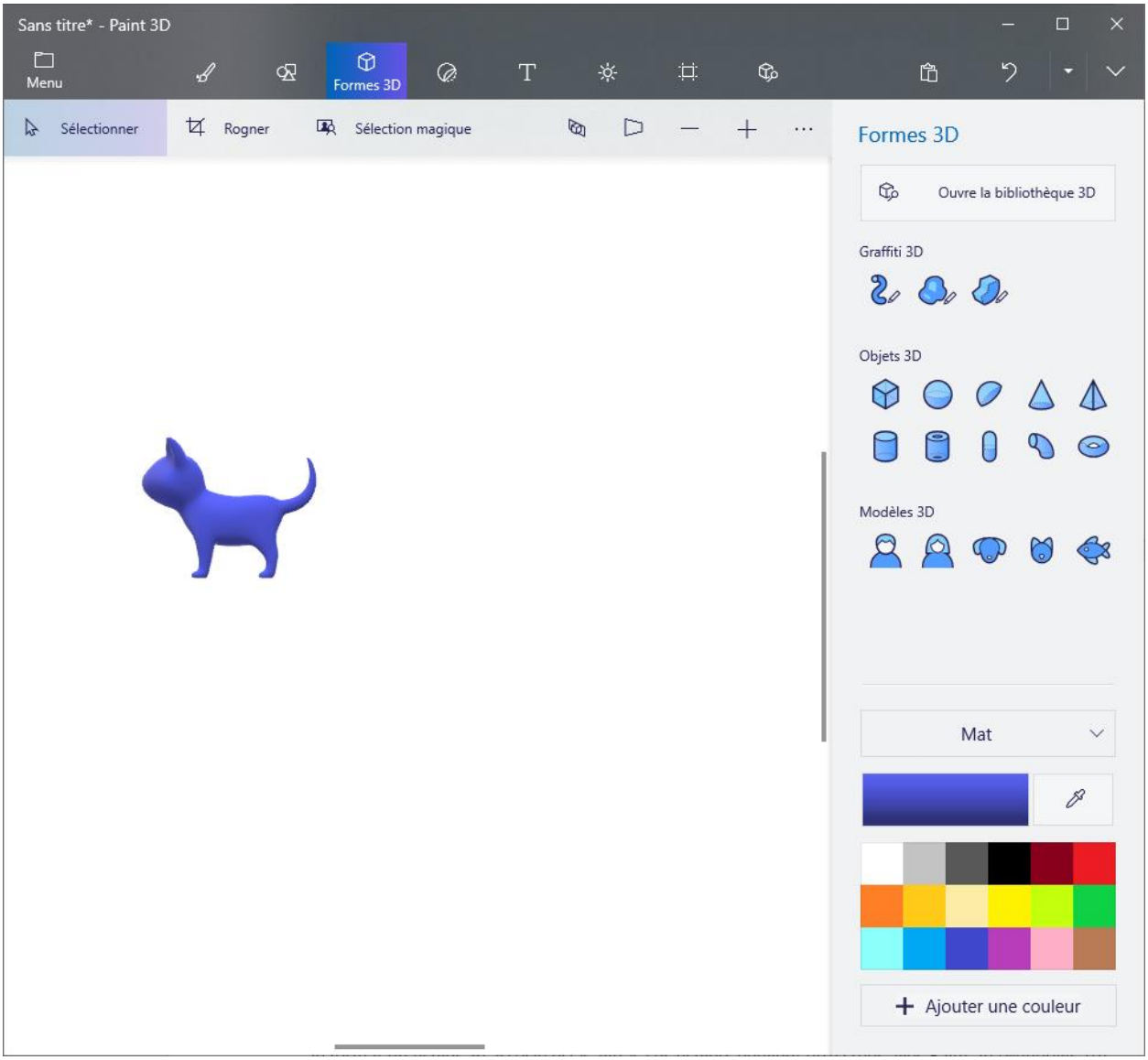
... after saving, your robot will appear like this in the scene editor ...



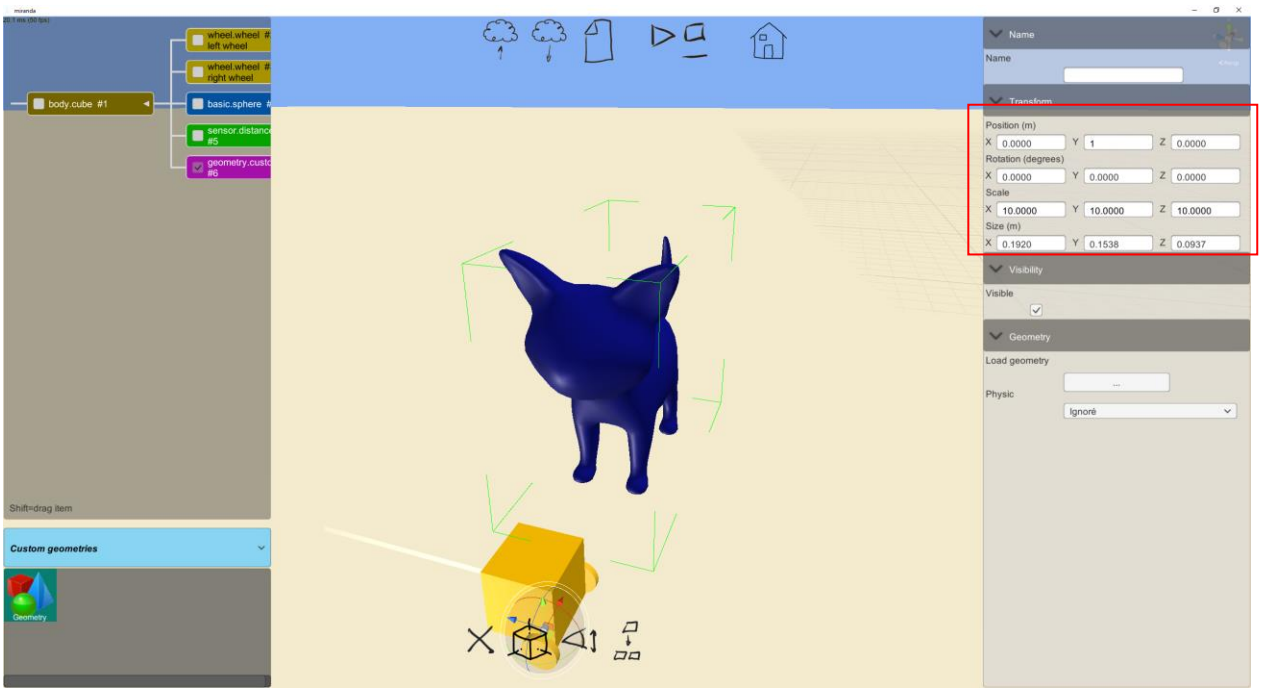
... Let's add custom geometry...

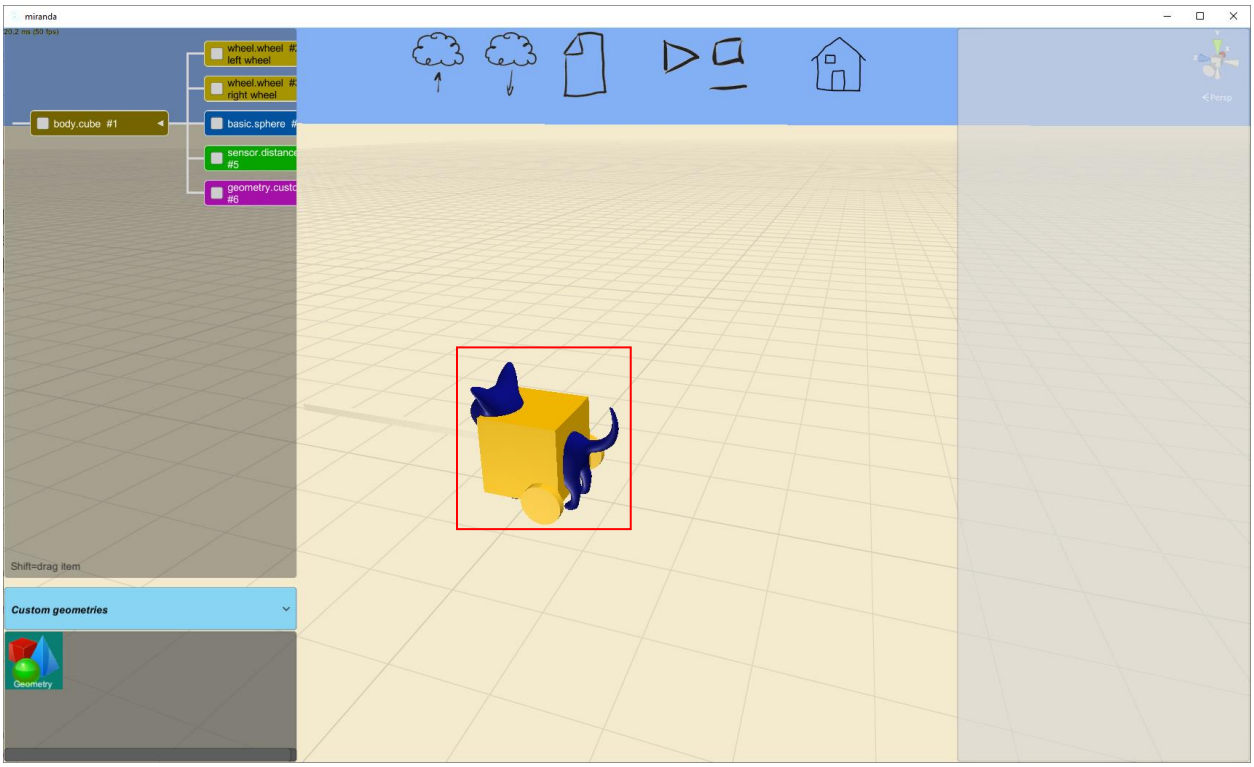


… The format of the associated 3d file is “.glb”, these files can be created with Paint 3d (Windows 10) or Blender…

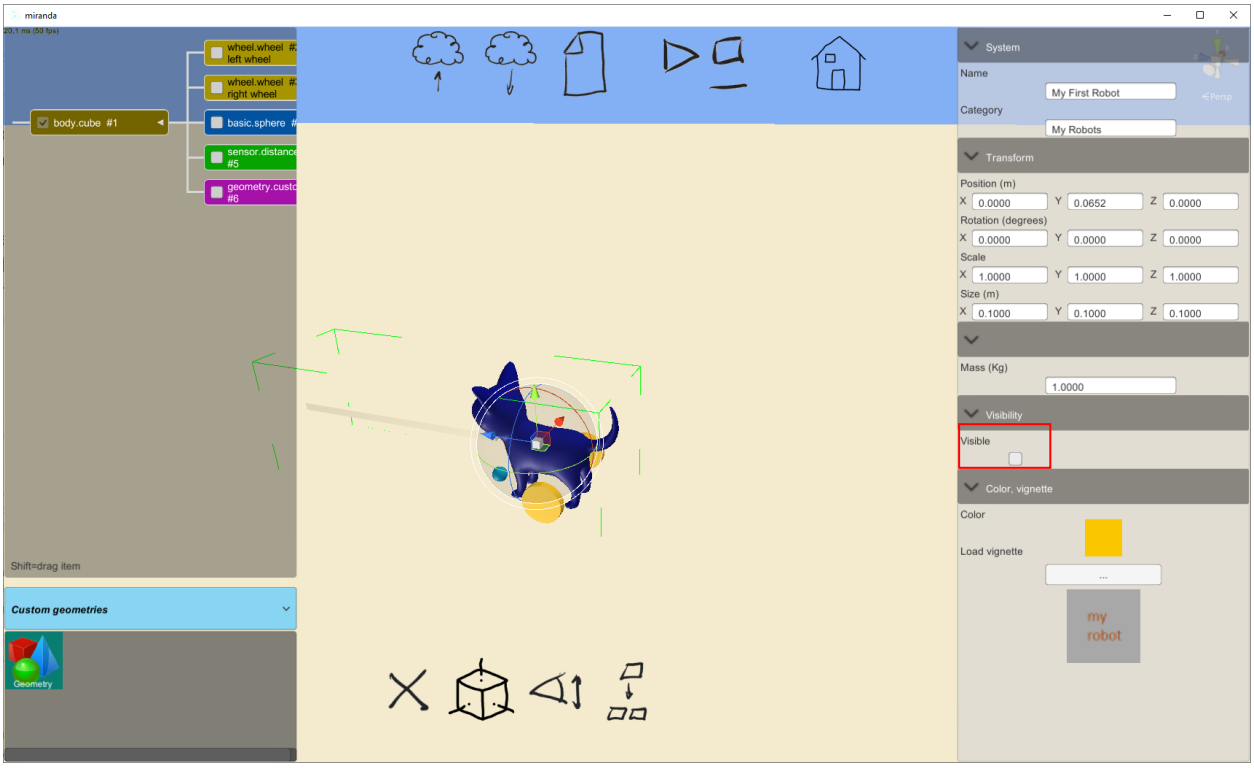


… save the .glb file, then open it from miranda…



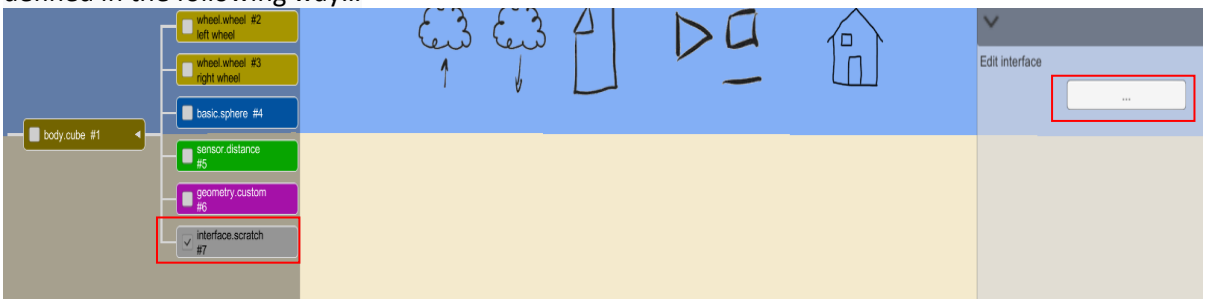


… The physical shapes can be hidden so that only custom geometries appear…



… personnalisons maintenant les blocs Scratchs associés à notre robot…

… When this item is added to a system, the default Scratch blocks are entirely replaced by the blocks defined in the following way…



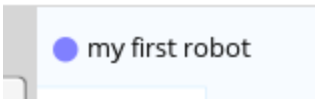
... this part is the most complex and requires writing a few lines in Python language to define the appearance of the blocks as well as their internal workings. The first Python function to use allows you to create a category, here is the syntax ...

adb.addcat(<category name>,<color>)

… and an example …

catid=adb.addcat("my first robot", "ff8080")

... will display the category in the Scratch editor like this ...



… If several categories must be created, several calls to this function must be made. The function returns an identifier that will be used when adding Scratchs blocks to the category. There are several syntaxes for adding blocks depending on the type of block to add. For a block of action ...

adb.addregularblk(<category id>,<block perimeter color>,<block background color>)

… A block returning a numeric or text value…

adb.addvalueblk(<category id>,<block perimeter color>,<block background color>)

… A block returning a boolean value…

adb.addbvalueblk(<category id>,<block perimeter color>,<block background color>)

… These functions return a block identifier which will be used by the functions for creating block elements. For a text ...

adb.addstring(<block id>,<text>)

… for a text input or a numerical value input …

adb.addvalue(<block id >,<default value>)

… for a boolean value…

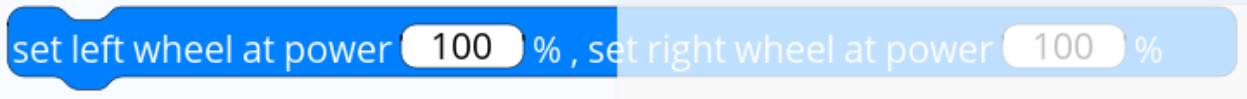
adb.addbvalue(<block id >)

… for a choice list…

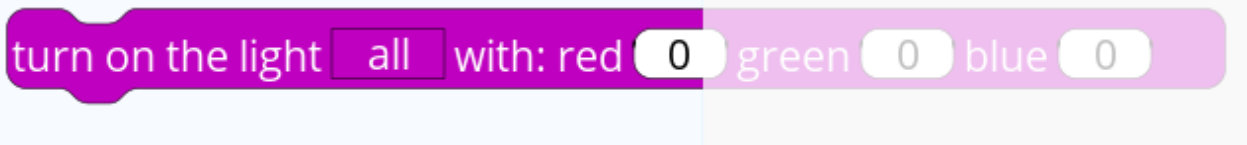
adb.addoption(<block id>,<options separated with ‘ ;’>,<default value>)

… some samples …

```
id=adb.addregularblk(catid,"000000","ff8000")
adb.addstring(id,"set left wheel at power ")
adb.addvalue(id,"100")
adb.addstring(id," % , set right wheel at power ")
adb.addvalue(id,"100")
adb.addstring(id," % ")
```




```
id=adb.addregularblk(catid,"000000","c000c0")
adb.addstring(id,"turn on the light ")
adb.addoption(id,"all;left;right","all")
adb.addstring(id," with: red ")
adb.addvalue(id,"0")
adb.addstring(id," green ")
adb.addvalue(id,"0")
adb.addstring(id," blue ")
adb.addvalue(id,"0")
```



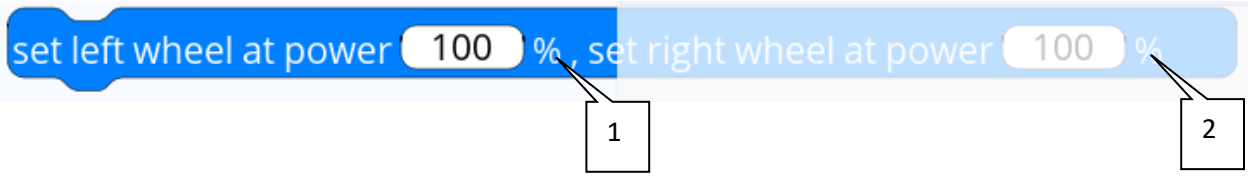
… As said previously, the python script defines both the blocks and their operation. A variable named "mode" is documented with a value determining whether the code must define the blocks (mode = 0) or if the code must manage the internal operation of each block (block = 1 for the operation of the first block, block = 2 for the second block, etc…). The blocks are numbered in the order of their definition: the first defined block is number 1, the second is number 2, etc. The Python code will be structured by if… tests on the mode variable. If mode = 0 then define the blocks, if mode = 1 then manage the execution of the first block, etc. Python code should use a function named "adb.readvar (" mode ")" to read the value of the "mode" variable. Let's come back to our example and create a block allowing to control the two motors of our robot ...

```
# block definition
if adb.readvar("mode")==0:
        idcat=adb.addcat("My first robot","ff8080")
        id=adb.addregularblk(idcat,"000000","ff8000")
        adb.addstring(id,"set left wheel at power ")
        adb.addvalue(id,"100")
        adb.addstring(id," %, set right wheel at power ")
        adb.addvalue(id,"100")
        adb.addstring(id,"%")

# block number 1 operation
if adb.readvar("mode")==1:
        import myfirstrobot
        myfirstrobot .leftwheel.setpower(adb.readvar("arg2"))
        myfirstrobot .rightwheel.setpower(adb.readvar("arg4"))
```

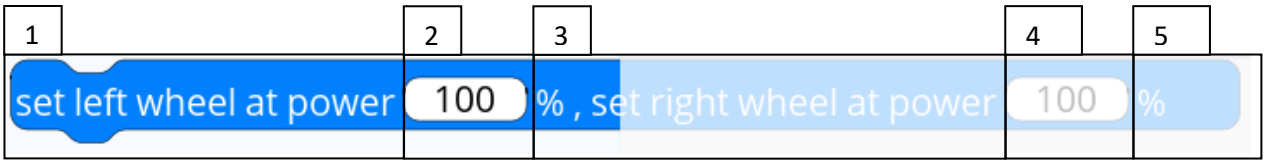
… Some explanations on the definition of the operation of the block are necessary. "Import myfirstrobot" is necessary to be able to call the functions of our robot. The usable functions, depending on the elements used to create a device, are automatically listed in the comments at the top of the Python script ...

```
# myfirstrobot.leftwheel.setpower(<puissance (de -100 à 100)>)
# myfirstrobot.rightwheel.setpower(<puissance (de -100 à 100)>)
# myfirstrobot.distancesensor()

# myfirstrobot.leftwheel.setpower(<power (from -100 to 100)>)
# myfirstrobot.rightwheel.setpower(<power (from -100 to 100)>)
# myfirstrobot.distancesensor()
```

… Another explanation is needed on the method used to reference a value associated with a block. Our block has two values ..



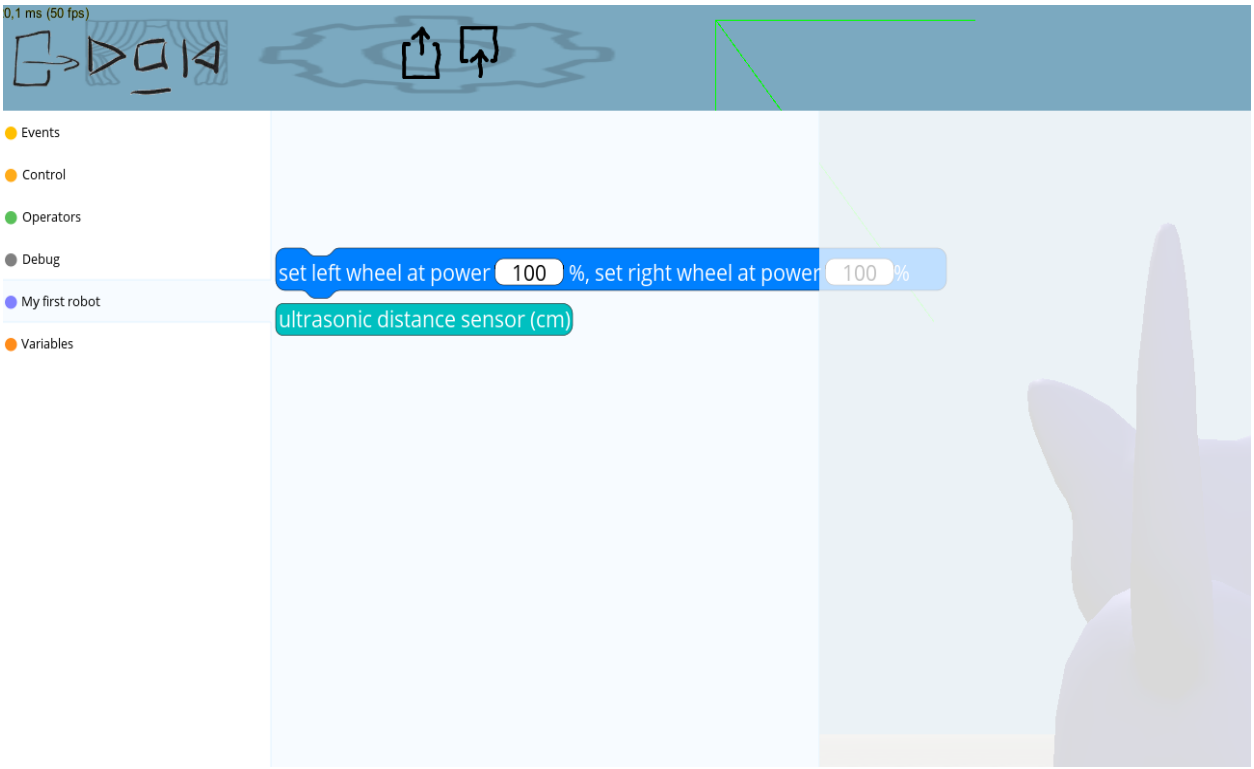
… The position of the values is determined by their order of creation on the block…



… The “arg <position>” variables are used to refer to the variables of the block in relation to their position. For our example, adb.readvar ("arg2") returns the value for the left wheel and adb.readvar ("arg4") returns the value for the left wheel. It remains to pass these values to the functions corresponding to each wheel: “myfirstrobot.leftwheel.setpower” and “myfirstrobot.rightwheel.setpower”. Let's continue our example by defining a block that will return the distance measured by the distance sensor in centimeters ...

```
# block definition
if adb.readvar("mode")==0:
        idcat=adb.addcat("My first robot","ff8080")
        id=adb.addregularblk(idcat,"000000","ff8000")
        adb.addstring(id,"set left wheel at power ")
        adb.addvalue(id,"100")
        adb.addstring(id," %, set right wheel at power ")
        adb.addvalue(id,"100")
        adb.addstring(id,"%")
        id=adb.addvalueblk(idcat,"000000","c0c000")
        adb.addstring(id,"ultrasonic distance sensor (cm)")


# block number 1 operation
if adb.readvar("mode")==1:
        import myfirstrobot
        myfirstrobot .leftwheel.setpower(adb.readvar("arg2"))
        myfirstrobot .rightwheel.setpower(adb.readvar("arg4"))


# block number 2 operation
if adb.readvar("mode")==2:
        import myfirstrobot
        adb.writevar("result" , myfirstrobot.distancesensor()*30)
```

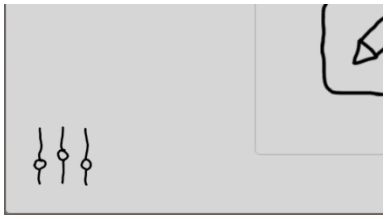
… Some additional explanations. The "distance sensor" block returns a value. This value return is handled by writing from the Python Script a variable named "result" using the "adb_writevar" function. The result is multiplied by 30, in fact, the “, myfirstrobot.distancesensor()” function returns a value between 0 and 1 depending on the distance measured and we have defined a detection zone of 30cm long. We are done with this example, after saving, the programming interface of our robot in the Scratch editor looks like this ...

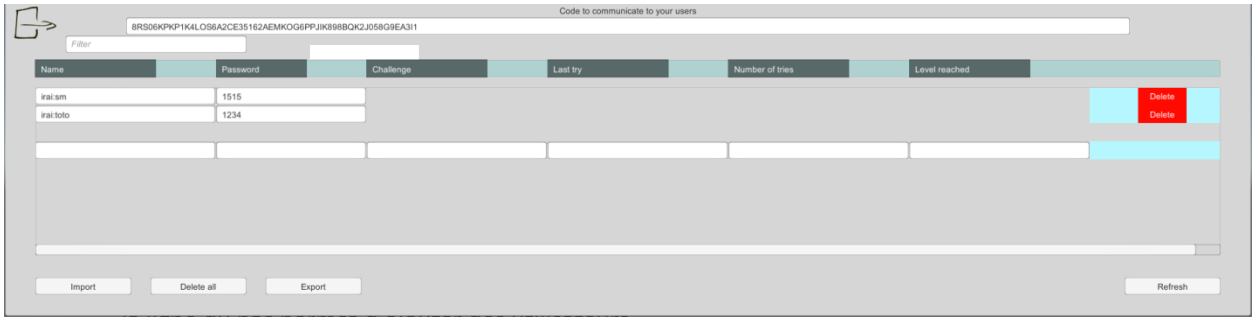


Users manager

The "school" versions of miranda allow you to manage users, generally associated with students or groups of students.

Access to the list of users ...





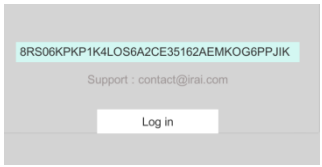
... the bottom line lets you add users.

The « filter » area...

... lets you display only the users for which the name includes the filter. After modifying, click on the « Refresh » button.

For each user, the challenges used appear with the level of advancement. The « Open » button lets open the last program created for each challenge.

The code displayed on the top line allows your users to log into miranda ...



The code can be passed when launching miranda. This code is always the same for a same miranda id (customer code).

For the Web browser version:

<url> ?<CODE>

Example:

http://irai2.com/mir?8RS06KPKP1K4LOS6A2CE35162AEZKOG6PPJIK898BQK2J058G9EA2I1

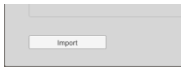
For the exe version:

<path to miranda exe> CODE=<CODE>

Example:

"C:\program files\miranda\miranda.exe" CODE=8RS06KPKP1K4LOS6A2CE35162AEMKOG6PPJIK8Z8BQ32J058G9EA3I1

Users list can be imported (miranda exe version only) ...



... from a text file.

The file may contain on, each line, an username, possibly followed by a password.

Files examples:

Elisa
John

Elisa,1234
John,9856

Elisa ;1234
John ;9856

If the password is not specified, then a random password is generated.

The displayed users can also be exported.

The users list can be deleted.

Users groups

Each user name may contain a header following by « : » defining the belonging to a group.

Example...

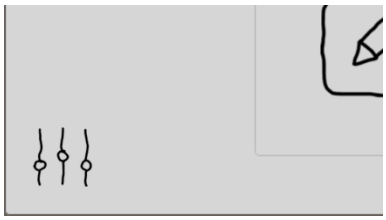
irai:sm

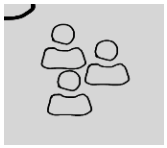
... user « sm » belonging to « irai » group.

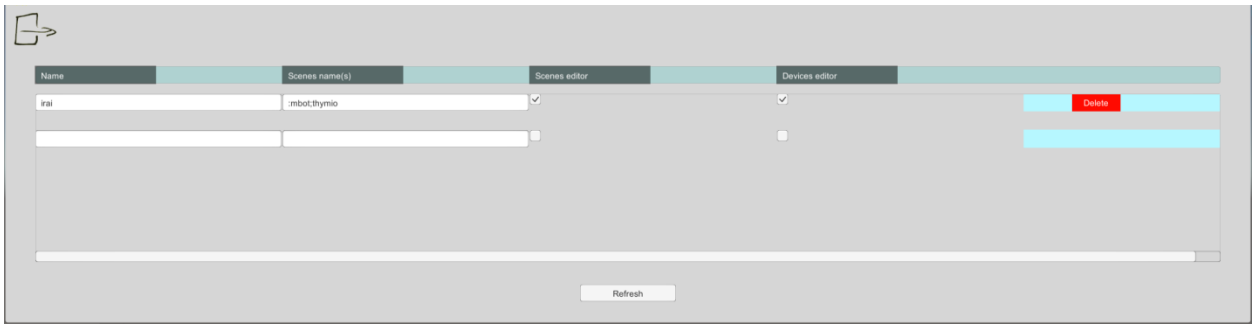
When login, users have to enter the full name. For instance:

irai:sm

Access to groups setup...







... for each group is defined : its name, a list of visible scenes, the possibility to use or not the scene editor and the device editor.

The list of visible scenes may contain one or more beginning of scene name (use « ; » as delimiter).

If this list is void, the default mode is apply : only the scenes with name beginning with « : » are visible.