Labyrinthe à bille
Créer une scène 3D interactive

Tutoriel 4 : Interfacer avec Arduino avec pySerial

Philippe Roy <philippe.roy@ac-grenoble.fr>
https://forge.aeif.fr/blender-edutech/blender-edutech-tuto

Objectif

L'objectif de ce tutoriel est faire interagir les objets de la scène 3D (des objets virtuelles) à partir d'actions physiques mesurées par des capteurs. Les **capteurs** sont ici reliés à un **micro-contrôleur Arduino** par la **connectique Grove** et la **liaison série**. La guidance de ce tutoriel a pour pré-requis la réalisation des deux tutoriels précédents (Tutoriel 1 : Ma première scène, Tutoriel 2 : Passage au Python).

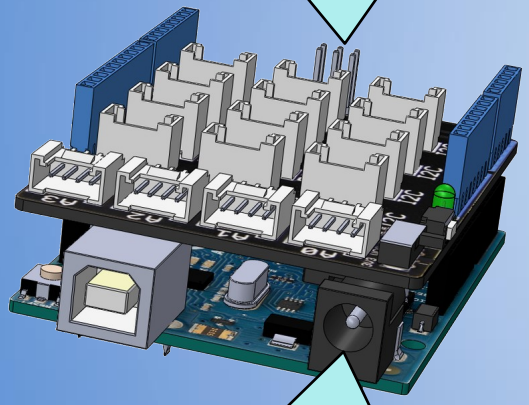
Le tutoriel se décompose en 6 étapes :

- 1. Préparer la carte Arduino
- 2. Programmer la carte Arduino pour le capteur IMU
- 3a. Installation de la bibliothèque pyFirmata sous GNU/Linux
- 3b. Installation de la bibliothèque pyFirmata sous Windows
- 4. Déplacer le plateau avec la centrale inertielle
- 5. Afficher la position de la bille sur la matrice de leds
- 6. Détecter automatiquement le micro-contrôleur
- 7. Distribuer l'exécutable

1. Préparer la carte Arduino

Arduino est une plateforme de conception et de fabrication d'objets électroniques interactifs. Le tutoriel utilise une **carte Uno** avec une **platine Grove** (voir le document joint « **DT - Grove pour Arduino** »).

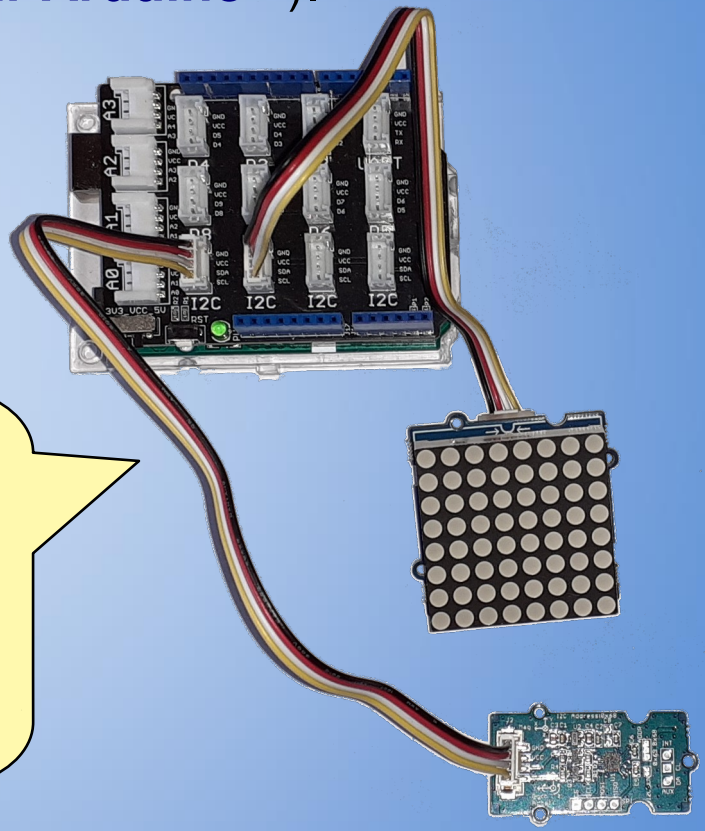
Platine (shield) Grove



Micro-contrôleur Arduino Uno

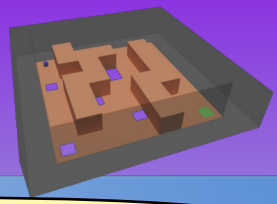
1 : Procéder au brochage

- `capteur IMU` sur un **port I2C**
- `matrice de leds` sur un **port I2C**

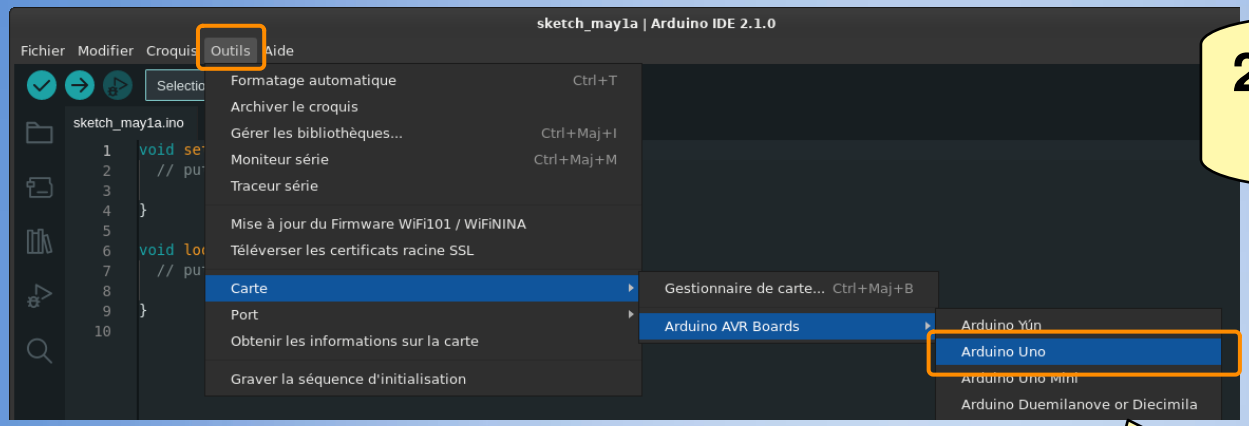


Le tutoriel vous propose la visualisation de la position de la bille sur la matrice de leds, mais il possible de faire le tutoriel sans la matrice de leds.

2. Programmer la carte Arduino avec la centrale inertielle (capteur IMU)

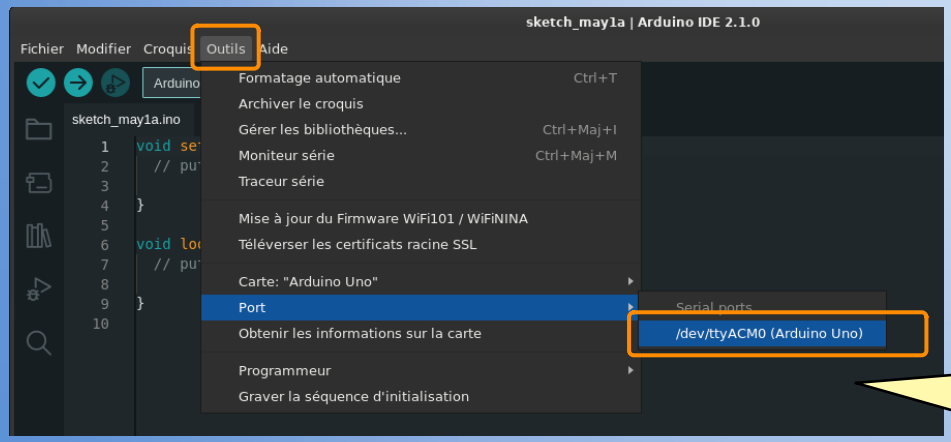


1 : Brancher la carte Arduino sur l'ordinateur avec le cordon USB



2 : Lancer le programme Arduino IDE

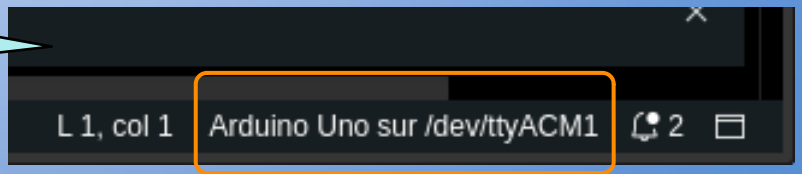
Arduino IDE est un éditeur libre disponible sur le site d' **Arduino**



3 : Définir le type de carte sur Arduino Uno

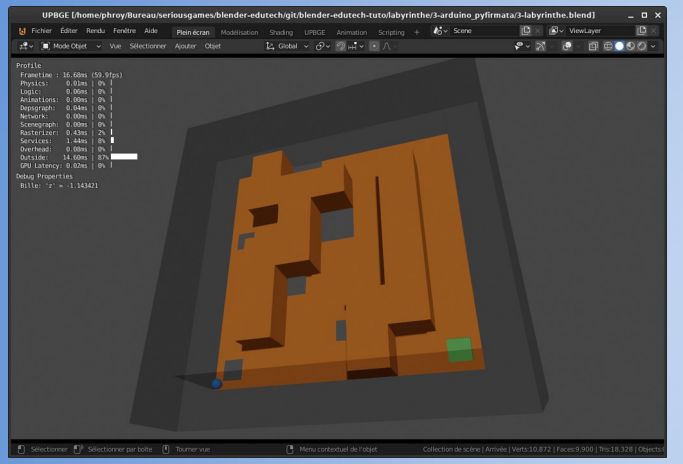
4 : Définir le port de communication sur celui qui a été détecté avec la carte.

La **barre d'état** nous indique la carte et le port actifs



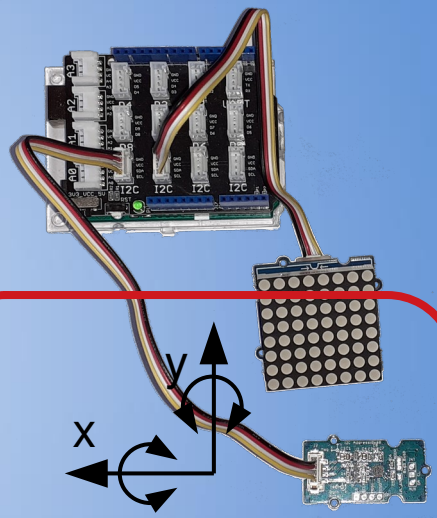
2. Programmer la carte Arduino avec la centrale inertielle (capteur IMU)

Nous allons utiliser la **liaison série** pour échanger des chaînes de caractère (message texte) entre la carte Arduino et le module Python.



Format du message texte : ‘’roulis,tangage‘‘

Liaison série (USB)



Roulis/Roll (x) et Tangage/Pitch (y)

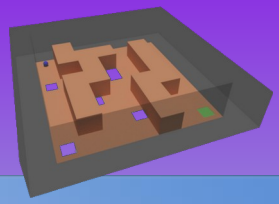
Programme UPBGE : 4-labyrinthe.py

Programme carte : 4-labyrinthe-imu.ino

Le **programme UPBGE** va être à l’écoute du port série et lire les messages. Il appliquera les angles lues à l’inclinaison du plateau.

Le **programme de la carte** va acquérir les angles de roulis et de tangage du capteur et générer le message sur le port série.

2. Programmer la carte Arduino avec la centrale inertielle (capteur IMU)



Le programme carte est le fichier « **4-labyrinthe-imu.ino** ».

```
#include "Wire.h"
#include "MPU6050.h"

/**************************************************************************
 * 4-labyrinthe-imu.ino
 **************************************************************************/

/**************************************************************************
 * IMU - I2C
 **************************************************************************/

MPU6050 accelgyro;

int16_t ax, ay, az;
int16_t gx, gy, gz;
int16_t mx, my, mz;
float Axyz[3];
float roll;// Roulis
float pitch; // Tangage
float roll_deg;
float pitch_deg;
String roll_txt;
String pitch_txt;
String serial_msg_int_txt;

/**************************************************************************
 * Initialisation
 **************************************************************************/

void setup() {

  // Liaison série
  Serial.begin(115200);

  // IMU
  Wire.begin();
  accelgyro.initialize();

}
```

5 : Déclarer les variables utilisées pour le capteur IMU

6 : Initialisation

- Ouvrir le port série
- Initialiser le capteur

2. Programmer la carte Arduino avec la centrale inertielle (capteur IMU)

```
/**************************************************************************
 * Boucle principale
 **************************************************************************/

void loop() {

    /*****
     * Lecture des accélérations
     * position capteur (X vers la gauche, Y vers l'arrière, Z vers le haut)
     *****/

    accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);
    Axyz[0] = (double) ax / 16384;
    Axyz[1] = (double) ay / 16384;
    Axyz[2] = (double) az / 16384;
    roll = asin(-Axyz[0]);
    roll_deg = roll*57.3;
    roll_txt = String(roll_deg);
    pitch = -asin(Axyz[1]/cos(roll));
    pitch_deg = pitch*57.3;
    pitch_txt = String(pitch_deg);

    /*****
     * IMU : Arduino -> UPBGE
     *****/

    Serial.print(roll_txt);
    Serial.print(",");
    Serial.print(pitch_txt);
    Serial.println();

}
```

7 : Lire les angles de roulis et de tangage

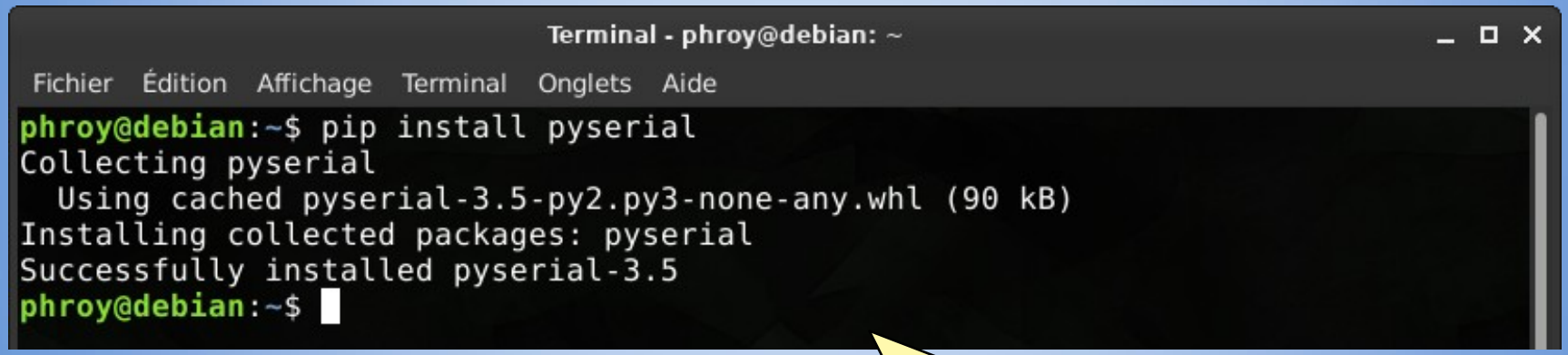
8 : Générer le message texte pour le port série

9 : Copier l'ensemble des fichiers de la bibliothèque « MPU6050 » dans le répertoire du programme : I2Cdev.h, I2Cdev.cpp, MPU6050.h MPU6050.cpp. Ils sont présents dans le répertoire du tutoriel.

3a. Installation de la bibliothèque pySerial sous GNU/Linux

La bibliothèque **pySerial** permet d'utiliser la **liaison série** dans un programme Python. Il faut donc installer la bibliothèque **pySerial**.

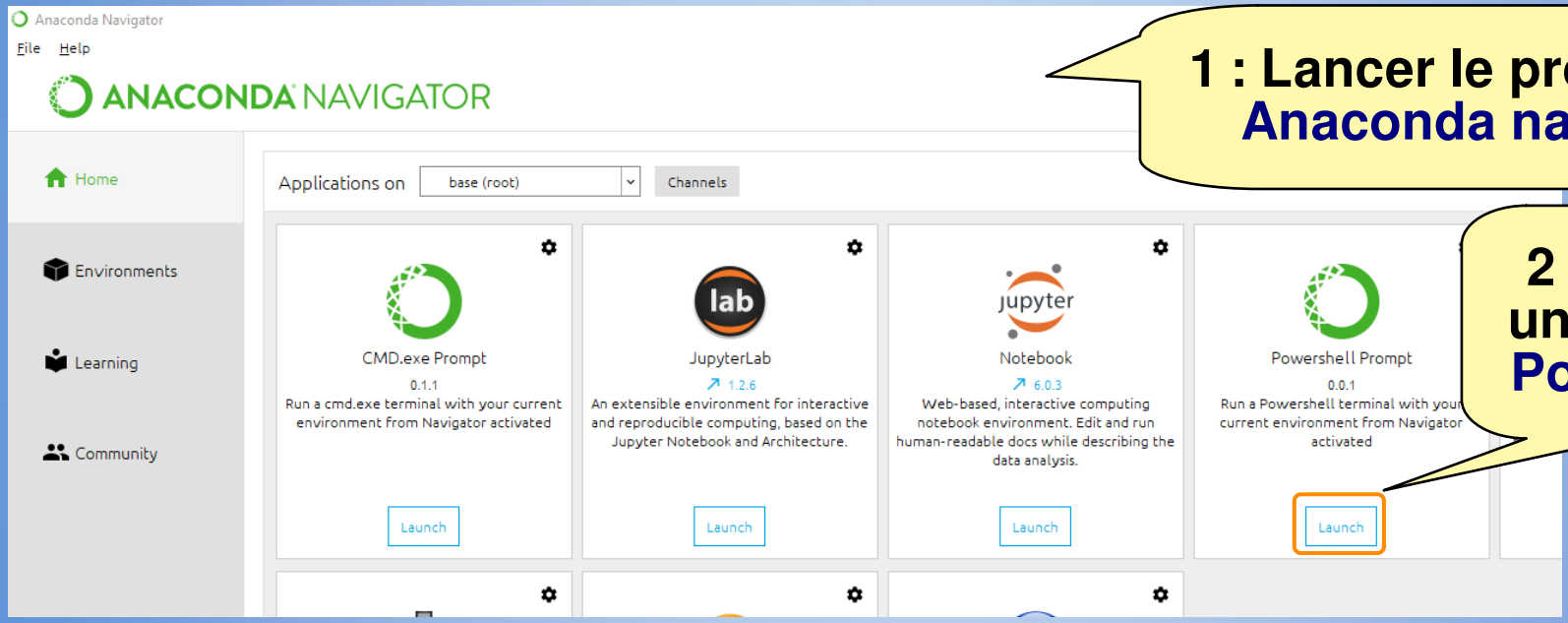
Généralement l'installateur de **bibliothèques Python** `pip` est déjà installé, sinon il faut utiliser le gestionnaire de paquet de la distribution pour l'installer.



1 : Installer pySerial
Dans un **terminal**
exécuter la commande :
`pip install pyserial`

3b. Installation de la bibliothèque pySerial sous Windows

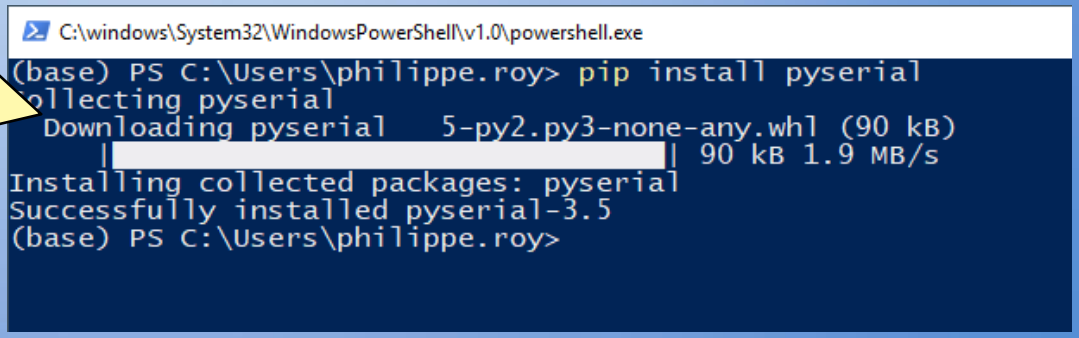
La bibliothèque **pySerial** permet d'utiliser la **liaison série** dans un programme Python. Il faut donc installer la bibliothèque **pySerial**. Sous Windows, l'installation de la distribution **Anaconda** est une solution simple et efficace de mettre en place un **environnement Python**.



1 : Lancer le programme Anaconda navigator

2 : Lancer un terminal Powershell.

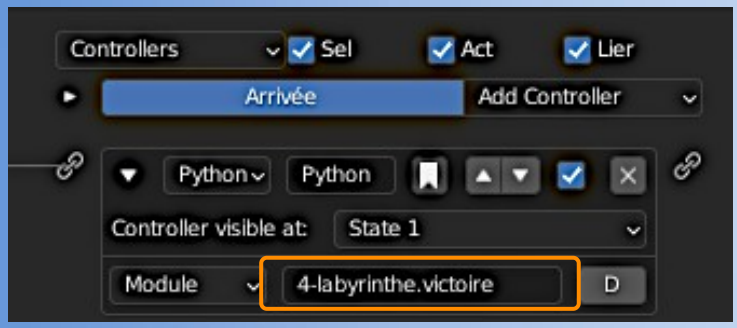
3 : Installer pySerial
Dans le **terminal**
exécuter la commande :
`pip install pyserial`

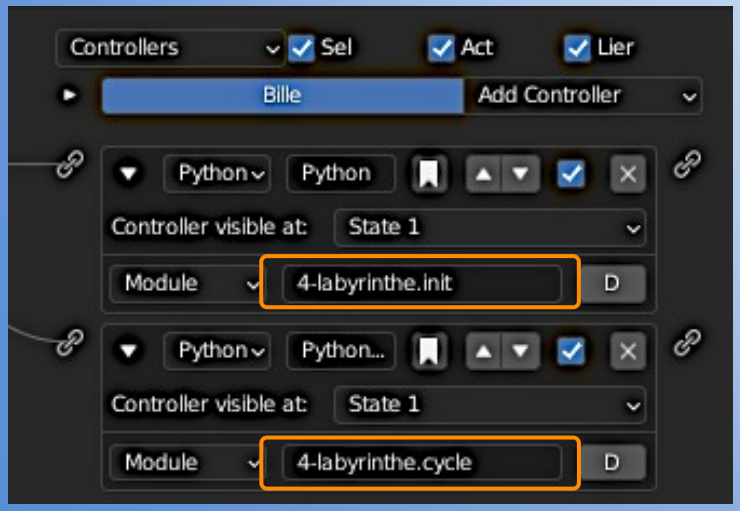


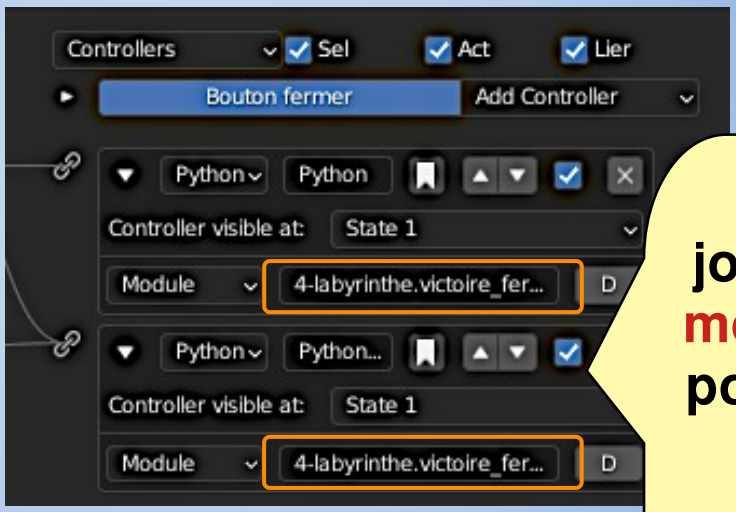
3. Déplacer le plateau avec la centrale inertielle

Les fichiers de départ de ce tutoriel sont les fichiers résultats du tutoriel 2. Il faut donc :

- copier et renommer « **2-labyrinthe.blend** » en « **4-labyrinthe.blend** »,
- copier et renommer « **2-labyrinthe.py**» en « **4-labyrinthe.py** ».



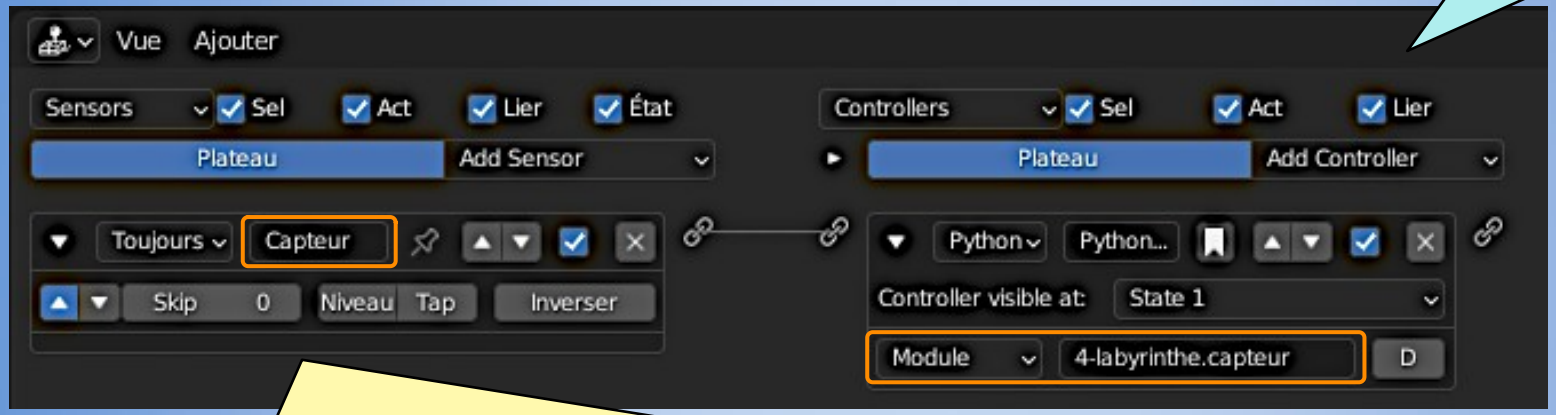




1 : Mettre à jour le nom des modules Python pour l'ensemble des briques logiques
Renommer les noms « **2-labyrinthe.*** » en « **4-labyrinthe.*** »

3. Déplacer le plateau avec la centrale inertielle

Briques logiques de **Plateau**



2 : Mettre UPBGE à l'écoute du capteur IMU

- **Renommer** le **Capteur Toujours** avec **Capteur**
  (ancien nom : « Clavier »)
- **renommer** le **Module Python** avec **4-labyrinthe.capteur**
  (ancien nom : « 2-labyrinthe.clavier »)

3. Déplacer le plateau avec la centrale inertielle

3 : Ajout l'importation de la bibliothèque

```
import serial
```

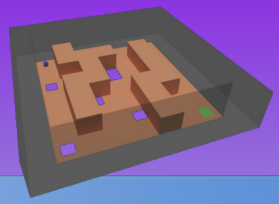
4-labyrinthe.py

```
import bge # Bibliothèque Blender Game Engi
import serial # Liaison série

################################################################################
# 4-labyrinthe.py
################################################################################

# Récupérer la scène 3D
scene = bge.logic.getCurrentScene()

# Constantes
JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
JUST_RELEASED = bge.logic.KX_INPUT_JUST_RELEASED
ACTIVATE = bge.logic.KX_INPUT_ACTIVE

################################################################################
# Communication avec la carte Arduino
################################################################################

serial_baud=115200
# serial_comm = serial.Serial('COM4',serial_baud, timeout=0.016) # Windows
serial_comm = serial.Serial('/dev/ttyACM1',serial_baud, timeout=0.016) # GNU/Linux
print (serial_comm)
```

4 : Créer la communication par la liaison série

- **Port** : COM4, /dev/ttyACM0, ... (indiqué par **IDE Arduino**)
- **Vitesse** : 115200 bauds
- **Timeout** : **0.016** s , c'est le temps de cycle de **UPBGE** (60 fps)

3. Déplacer le plateau avec la centrale inertielle



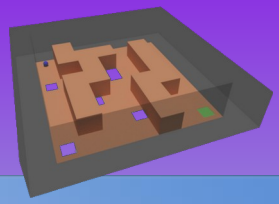
4-labyrinthe.py

```
import bge # Bibliothèque Blender Game Engine (UPBGE)
import serial # Liaison série

################################################################################
# 4-labyrinthe.py
################################################################################

# Récupérer la scène 3D
scene = bge.logic.getCurrentScene()

# Constantes
JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
JUST_RELEASED = bge.logic.KX_INPUT_JUST_RELEASED
ACTIVATE = bge.logic.KX_INPUT_ACTIVE

################################################################################
# Communication avec la carte Arduino
################################################################################

serial_baud=115200
# serial_comm = serial.Serial('COM4',serial_baud, timeout=0.016) # Windows
serial_comm = serial.Serial('/dev/ttyACM1',serial_baud, timeout=0.016) # GNU/Linux
print (serial_comm)



carte = pyfirmata.Arduino('COM4') # Windows
carte = pyfirmata.Arduino('/dev/ttyACM0') # GNU/Linux
print("Communication Carte Arduino établie")

# Itérateur pour les entrées
it = pyfirmata.util.Iterator(carte)
it.start()

# Définition des 4 boutons
bt_haut = carte.get_pin('d:2:i')
bt_bas = carte.get_pin('d:3:i')
bt_gauche = carte.get_pin('d:4:i')
bt_droit = carte.get_pin('d:5:i')

# Définition de la led
```

3 : Ajout l'importation de la bibliothèque
`import serial`

4 : Créer la communication par la liaison série
Le nom du port (COM4, /dev/ttyACM0, ...) est indiqué par **IDE Arduino**

3. Déplacer le plateau avec la centrale inertielle



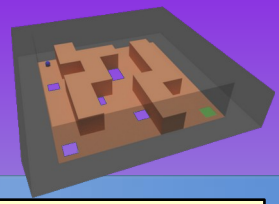
4-labyrinthe.py

```
import bge # Bibliothèque Blender Game Engine (UPBGE)
import serial # Liaison série

################################################################################
# 4-labyrinthe.py
################################################################################

# Récupérer la scène 3D
scene = bge.logic.getCurrentScene()

# Constantes
JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
JUST_RELEASED = bge.logic.KX_INPUT_JUST_RELEASED
ACTIVATE = bge.logic.KX_INPUT_ACTIVE

################################################################################
# Communication avec la carte Arduino
################################################################################

serial_baud=115200
# serial_comm = serial.Serial('COM4',serial_baud, timeout=0.016) # Windows
serial_comm = serial.Serial('/dev/ttyACM1',serial_baud, timeout=0.016) # GNU/Linux
print (serial_comm)



carte = pyfirmata.Arduino('COM4') # Windows
carte = pyfirmata.Arduino('/dev/ttyACM0') # GNU/Linux
print("Communication Carte Arduino établie")

# Itérateur pour les entrées
it = pyfirmata.util.Iterator(carte)
it.start()

# Définition des 4 boutons
bt_haut = carte.get_pin('d:2:i')
bt_bas = carte.get_pin('d:3:i')
bt_gauche = carte.get_pin('d:4:i')
bt_droit = carte.get_pin('d:5:i')

# Définition de la led
```

3 : Ajout l'importation de la bibliothèque
`import serial`

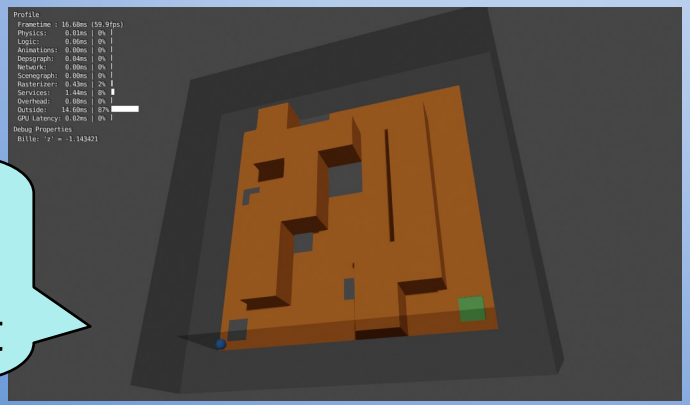
4 : Créer la communication par la liaison série
Le nom du port (COM4, /dev/ttyACM0, ...) est indiqué par **IDE Arduino**

3. Déplacer le plateau avec la centrale inertielle



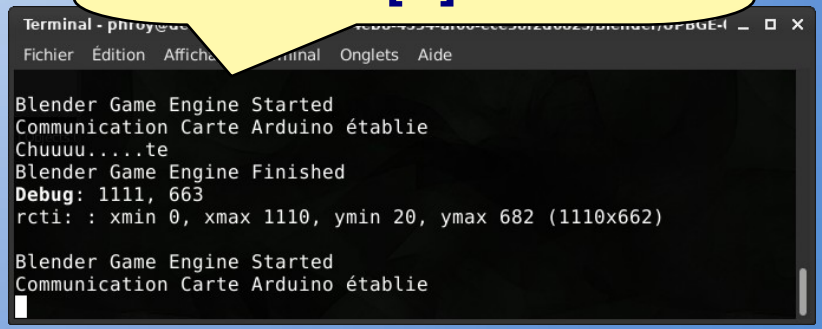
```
##############################################################################
# Gestion de la manette Arduino
##############################################################################

def manette(cont):
    obj = cont.owner # obj est l'objet associé au contrôleur donc 'Plateau'
    resolution = 0.01
    led.write(False) # Éteindre led

    # Bouton haut - Broche 2
    if bt_haut.read() == True and bt_bas.read() == False:
        obj.applyRotation((-resolution, 0, -obj.worldOrientation.to_euler().z), False)
        led.write(True)

    # Bouton bas - Broche 3
    if bt_haut.read() == False and bt_bas.read() == True:
        obj.applyRotation((resolution, 0, -obj.worldOrientation.to_euler().z), False)
        led.write(True)

    # Bouton gauche - Broche 4
    if bt_gauche.read() == True and bt_droit.read() == False:
        obj.applyRotation((0, -resolution, -obj.worldOrientation.to_euler().z), False)
        led.write(True)

    # Bouton droit - Broche 5
    if bt_gauche.read() == False and bt_droit.read() == True:
        obj.applyRotation((0, resolution, -obj.worldOrientation.to_euler().z), False)
        led.write(True)
```

6 : Créer la fonction manette



La led indique le plateau en mouvement

7 : Tester les boutons [P]



5. Détecter automatiquement le micro-contrôleur

Au début du programme il faut saisir le port sur lequel est branché la carte. Par exemple si le port est « COM4 » le code est : `carte = pyfirmata.Arduino('COM4')`. Or le port change souvent et afin d'éviter de retoucher le code on souhaite détecter automatiquement le port.

Nous allons créer un module Python uniquement pour la détection du port : « **labyrinthe_carte.py** ». Il sera aussi utilisé pour les tutoriels 4 et 5.

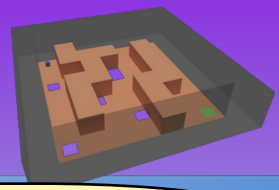
1 : Créer le fichier labyrinthe_carte.py

```
import pyfirmata # Protocole Firmata
import serial # Liaison série
from serial.tools.list_ports import comports # Détection du port automatique

##############################################################################
# labyrinthe_carte.py
##############################################################################

# Recherche automatique du port (microbit, Arduino Uno et Arduino Mega)
def autoget_port():
    # USB Vendor ID,  USB Product ID
    carte_dict={'microbit' :[3368, 516],
                'uno' :[9025, 67],
                'mega' :[9025, 66]}
    for com in comports(): # micro:bit
        if com.vid == carte_dict["microbit"][0] and com.pid == carte_dict["microbit"][1]:
            return [com.device,"micro:bit"]
    for com in comports(): # Arduino Uno
        if com.vid == carte_dict["uno"][0] and com.pid == carte_dict["uno"][1]:
            return [com.device,"Arduino Uno"]
    for com in comports(): # Arduino Mega
        if com.vid == carte_dict["mega"][0] and com.pid == carte_dict["mega"][1]:
            return [com.device,"Arduino Mega"]
    return [None,""]
```

2 : Importation des bibliothèques

3 : Fonction de détection de la carte

5. Détecter automatiquement le micro-contrôleur



labyrinthe_carte.py

4 : Fonction d'initialisation de la communication avec la carte avec le protocole Firmata

```
# Établir la communication avec la carte avec le protocol Firmata
def init_firmata():
    [port, carte_name] = autoget_port()
    if port is None:
        print("Communication avec Carte Arduino impossible")
        return None
    else:
        try:
            carte = pyfirmata.Arduino(port)
            print("Communication avec Carte Arduino établie sur "+port+" avec le protocole Firmata")
            return carte
        except:
            print("Communication avec Carte Arduino impossible")
            return None
```

Nous allons maintenant utiliser la fonction **`init_firmata()`**.

7 : Supprimer la création manuelle de l'objet carte

```
##################################################################
# Communication avec la carte Arduino
##################################################################

# carte = pyfirmata.Arduino('COM4') # Windows
carte = pyfirmata.Arduino('/dev/ttyACM0') # GNU/Linux
print("Communication Carte Arduino établie")

# Détection de la carte avec la protocol Firmata
carte = labyrinthe_carte.init_firmata()
if carte is None:
    bge.logic.endGame()
```

8 : Ajouter la création automatique de l'objet carte

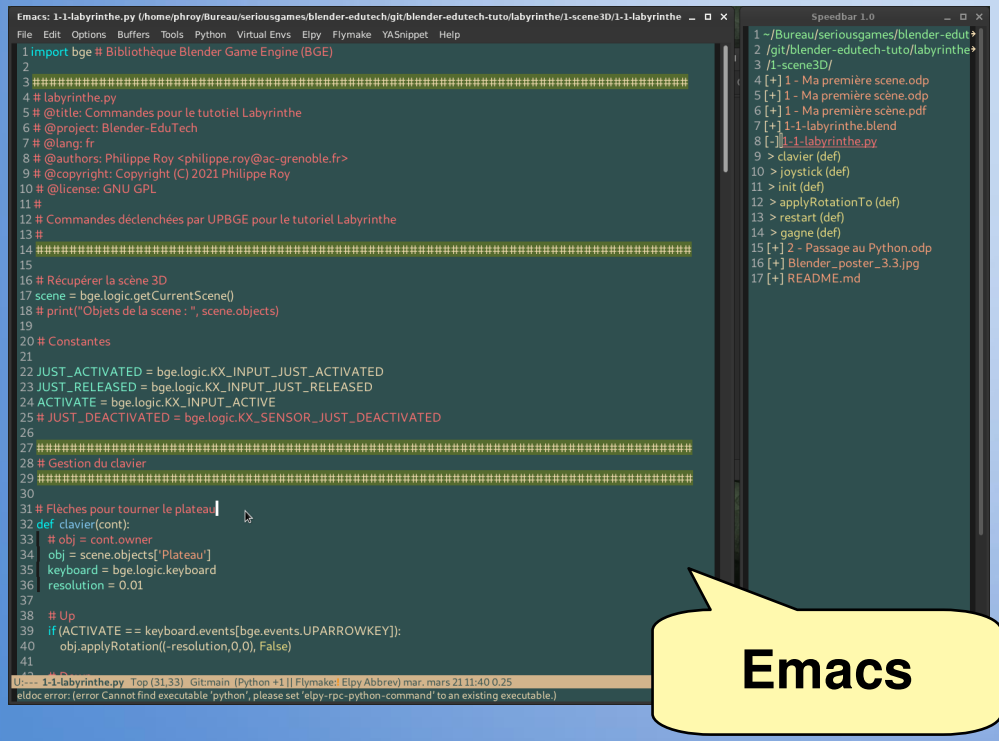
3-labyrinthe.py

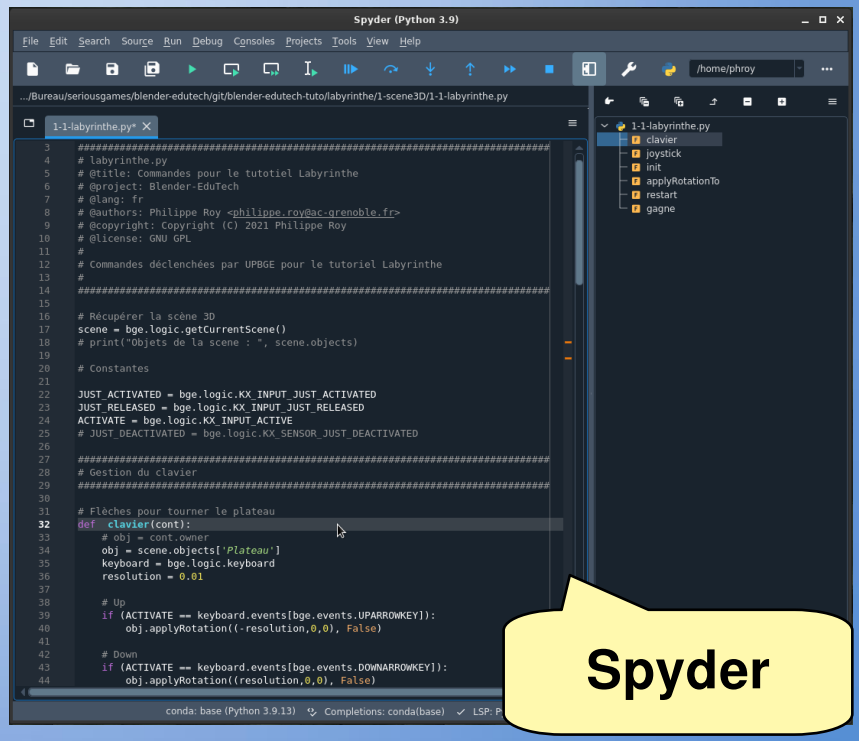
9 : Tester le détection automatique de la carte [P]

5. Inclure pySerial dans la distribution de l'exécutable

Le codage se fait par un éditeur de texte. Il en existe beaucoup et tout éditeur peut convenir. Pour ce tutoriel j'utilise **Emacs**, il est open source, très efficace et polyvalent mais peu intuitif. Le choix de l'éditeur est souvent très personnel, sans préférence je vous conseille **Spyder**, il est open source et complet.

- **Emacs** ce trouve à cette adresse : https://www.gnu.org/software/emacs
- **Spyder** ce trouve à cette adresse : https://www.spyder-ide.org

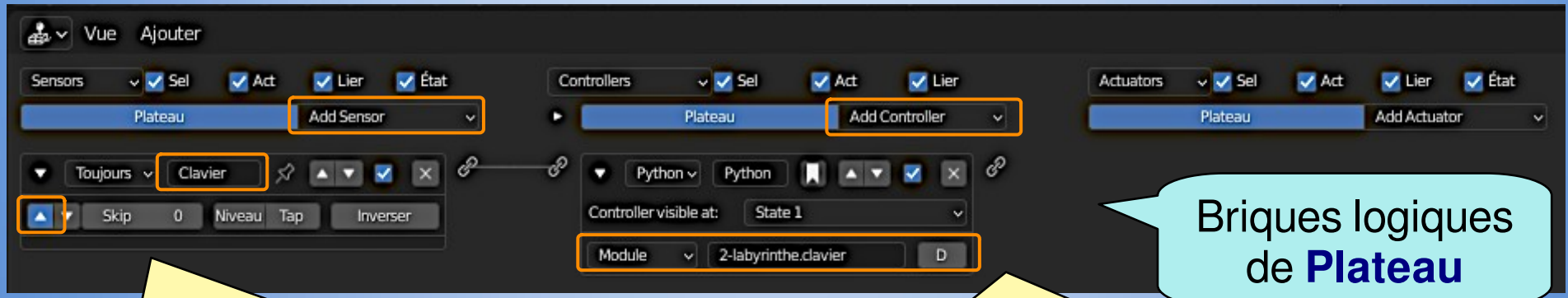




2. Déplacer le plateau

Le fichier Blender de départ est le fichier résultat du tutoriel 1 sans les briques logiques ni les propriétés. Il est disponible dans le répertoire du tutoriel sous le nom « **2-labyrinthe-debut.blend** ».

Pour la gestion du clavier, le principe est de créer un **boucle infinie** qui exécute la fonction **clavier** à chaque **tic logique (logic tick)**.



Briques logiques de **Plateau**

1 : Créer la boucle infinie

- **Ajouter** un **Capteur Toujours**
- **activer** le **Pulse True Level (▲)**
- **renommer** le capteur avec **Clavier**

2 : Appeler la fonction

- **Ajouter** un **Contrôleur Python**
- **définir** le **Module** avec la fonction **2-labyrinthe.clavier**

2. Déplacer le plateau

Le module Python est le fichier « **2-labyrinthe.py** ».

```
import bge # Bibliothèque Blender Game Engine (UPBGE

################################################################################
# 2-labyrinthe.py
################################################################################

# Récupérer la scène 3D
scene = bge.logic.getCurrentScene()

# Constantes
JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
JUST_RELEASED = bge.logic.KX_INPUT_JUST_RELEASED
ACTIVATE = bge.logic.KX_INPUT_ACTIVE

################################################################################
# Gestion du clavier
################################################################################

# Flèches pour tourner le plateau
def clavier(cont):
    obj = cont.owner # obj est l'objet associé au contrôleur donc 'Plateau'
    keyboard = bge.logic.keyboard
    resolution = 0.01

    # Flèche haut - Up arrow
    if keyboard.inputs[bge.events.UPARROWKEY].status[0] == ACTIVATE:
        obj.applyRotation((-resolution,0,0), False)

    # Flèche bas - Down arrow
    if keyboard.inputs[bge.events.DOWNARROWKEY].status[0] == ACTIVATE:
        obj.applyRotation((resolution,0,0), False)

    # Flèche gauche - Left arrow
    if keyboard.inputs[bge.events.LEFTARROWKEY].status[0] == ACTIVATE:
        obj.applyRotation((0, -resolution,0), False)

    # Flèche droit - Right arrow
    if keyboard.inputs[bge.events.RIGHTARROWKEY].status[0] == ACTIVATE:
        obj.applyRotation((0, resolution,0), False)
```

3 : Créer le fichier Python
Ouvrir votre éditeur et créer le fichier **2-labyrinthe.py**

4 : Créer le fonction clavier
Copier-coller le code

5 : Tester la scène [P]

Les pages de l'
API Python de UPBGE
les plus utilisées sont

- **GameObject**
- **Game Logic**

labyrinthe_carte.py

4 : Fonction d'initialisation de la communication avec la carte avec le protocole Firmata (tutoriel 3)

```
# Établir la communication avec la carte avec le protocol Firmata
def init_firmata():
    [port, carte_name] = autoget_port()
    if port is None:
        print("Communication avec Carte Arduino impossible")
        return None
    else:
        try:
            carte = pyfirmata.Arduino(port)
            print("Communication avec Carte Arduino établie sur "+port+" avec le protocole Firmata")
            return carte
        except:
            print("Communication avec Carte Arduino impossible")
            return None

# Établir la communication avec la carte avec la liaison série avec une vitesse
def init_serial(speed=115200):
    [port, carte_name] = autoget_port()
    if port is None:
        print("Communication avec Carte Arduino/microbit impossible")
        return None
    else:
        try:
            carte = serial.Serial(port,speed)
            print("Communication avec Carte Arduino/microbit établie sur "+port+" à la vitesse "+speed+"bauds")
            return carte
        except:
            print("Communication avec Carte Arduino/microbit impossible")
            return None
```

5 : Fonction d'initialisation de la communication avec la carte par la liaison série (tutoriel 4 et 5)