

# Labyrinthe à bille

## Créer une scène 3D interactive

# Tutoriel 4 : Interfacer avec Arduino avec pySerial



Philippe Roy <[philippe.roy@ac-grenoble.fr](mailto:philippe.roy@ac-grenoble.fr)>

<https://forge.aeif.fr/blender-edutech/blender-edutech-tuto>

# Objectif



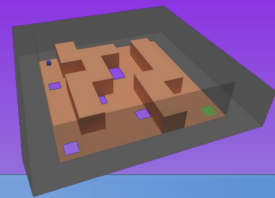
L'objectif de ce tutoriel est faire interagir les objets de la scène 3D (des objets virtuelles) à partir d'actions physiques mesurées par des capteurs. Les **capteurs** sont ici reliés à un **micro-contrôleur Arduino** par la **connectique Grove**. La **carte Arduino** communique avec l'ordinateur par 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 pySerial sous GNU/Linux](#)
- [3b. Installation de la bibliothèque pySerial 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. Inclure pySerial dans la distribution de l'exécutable](#)

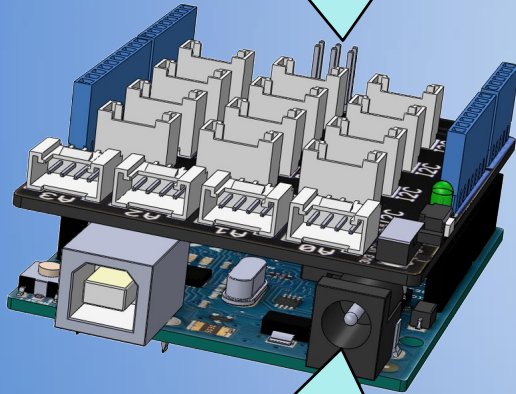


# 1. Préparer la carte Arduino



**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** »).

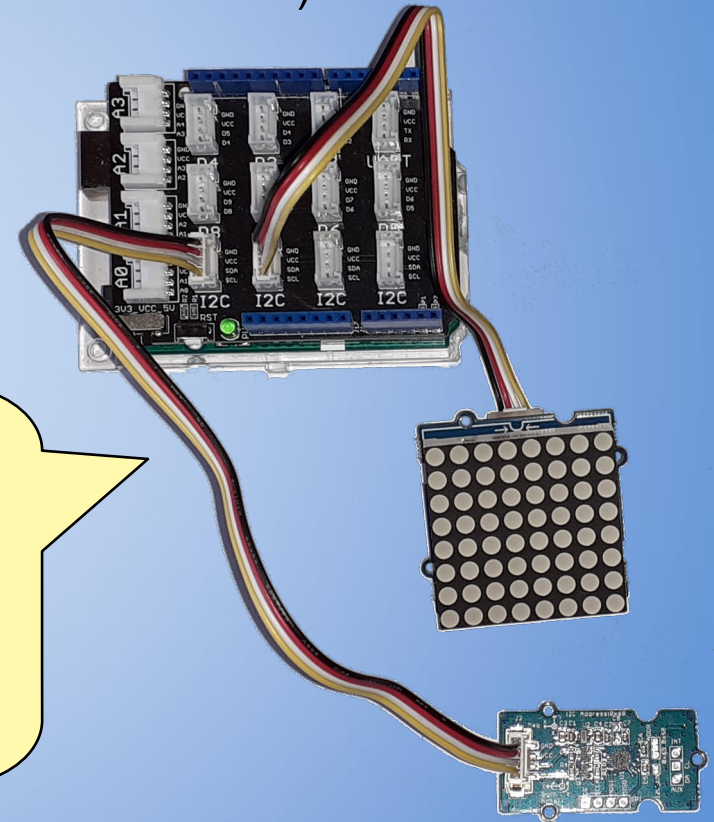
Platine  
(shield) Grove



Micro-contrôleur  
Arduino Uno

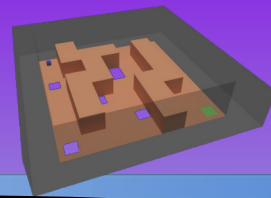
1 : Procéder  
au brochage

- 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](https://www.arduino.cc)

3 : Définir le type de carte sur **Arduino Uno**

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

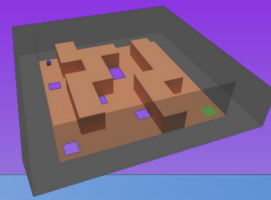
La **barre d'état** nous indique la carte et le port actifs

L 1, col 1

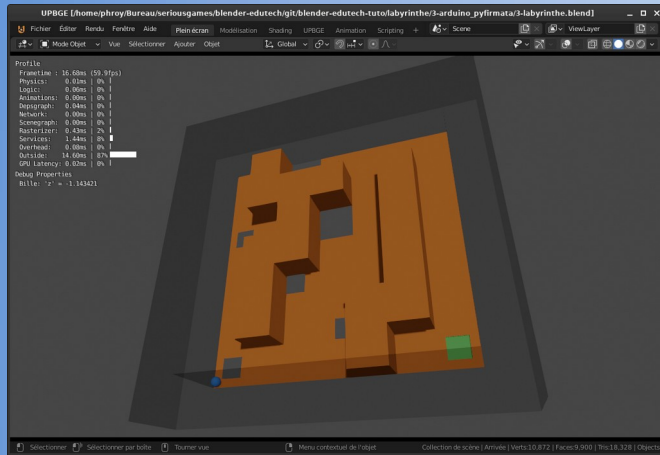
Arduino Uno sur /dev/ttyACM1

2

## 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.



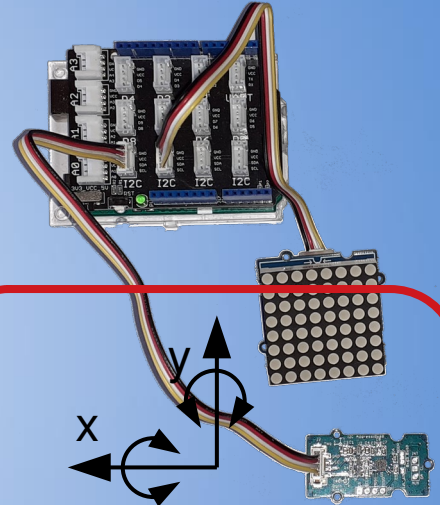
Programme  
UPBGE :  
4-labyrinthe.py



Format du  
message texte :  
"roulis,tangage"

Liaison série  
(USB)

Programme carte :  
4-labyrinthe-imu.ino

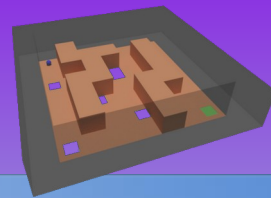


Roulis/Roll (x) et  
Tangage/Pitch (y)

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;

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

void setup() {

  // Liaison série
  Serial.begin(115200);

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

}
```

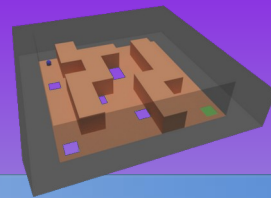
**5 : Importer les bibliothèques**

**6 : Déclarer les variables utilisées pour le capteur IMU**

**7 : 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();  
}
```

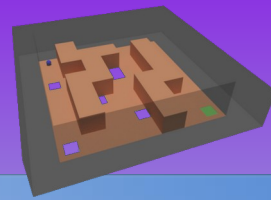
**8 : Lire les angles de roulis et de tangage**

**9 : Générer le message texte pour le port série**

**10 : Copier l'ensemble des fichiers de la bibliothèque « MPU6050 » dans le répertoire du programme : I2Cdev.h, I2Cdev.cpp, MPU6050.h MPU6050.cpp** (présents dans le répertoire du tutoriel)

**11 : Téléverser le programme vers la carte**

# 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.

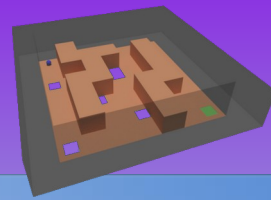
```
Terminal - phroy@debian: ~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
phroy@debian:~$ pip install pyserial
Collecting pyserial
  Using cached pyserial-3.5-py2.py3-none-any.whl (90 kB)
Installing collected packages: pyserial
Successfully installed pyserial-3.5
phroy@debian:~$
```

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

Si la bibliothèque pySerial n'est pas reconnue par UPBGE, il faut alors l'installer dans l'environnement local de UPBGE, pour cela voir l'**étape 7**.

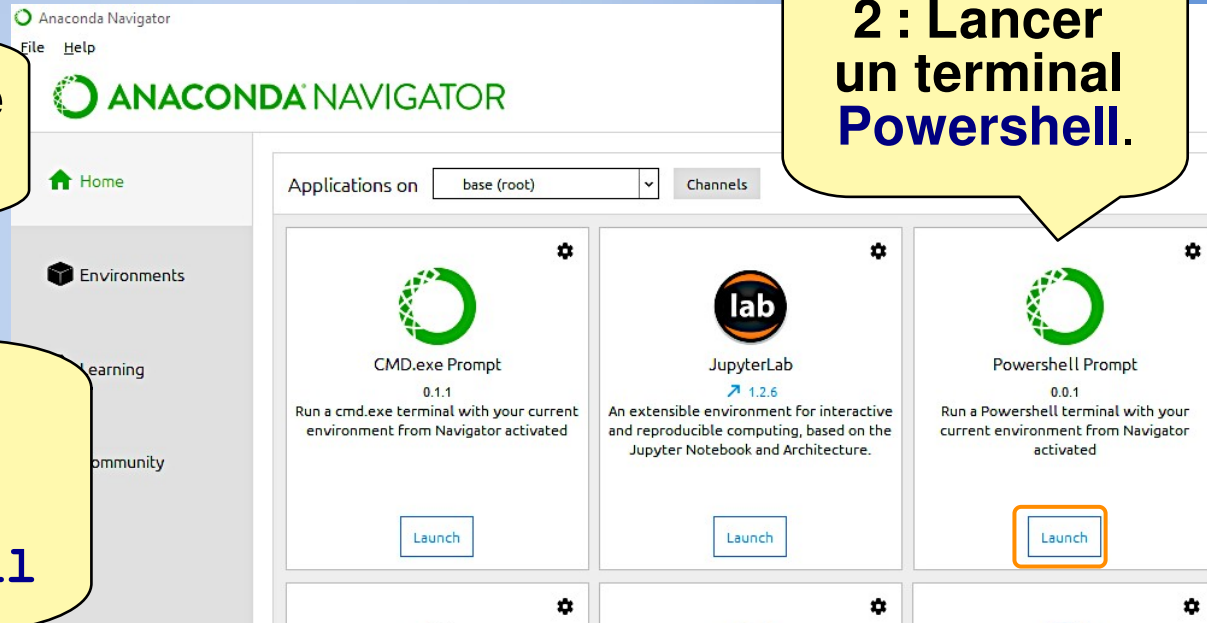


# 3b. Installation de la bibliothèque pySerial sous Windows



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**

```
C:\windows\System32\WindowsPowerShell\v1.0\powershell.exe
(base) PS C:\Users\philippe.roy> pip install pyserial
Collecting pyserial
  Downloading pyserial_5-py2.py3-none-any.whl (90 kB)
    |#####| 90 kB 1.9 MB/s
Installing collected packages: pyserial
Successfully installed pyserial-3.5
(base) PS C:\Users\philippe.roy>
```

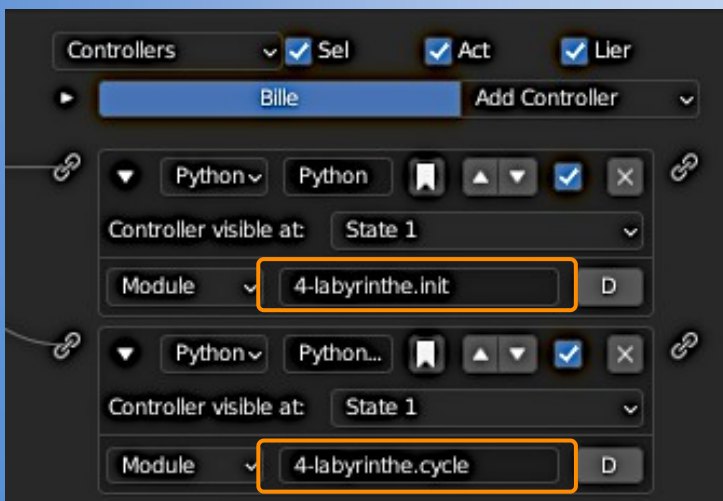
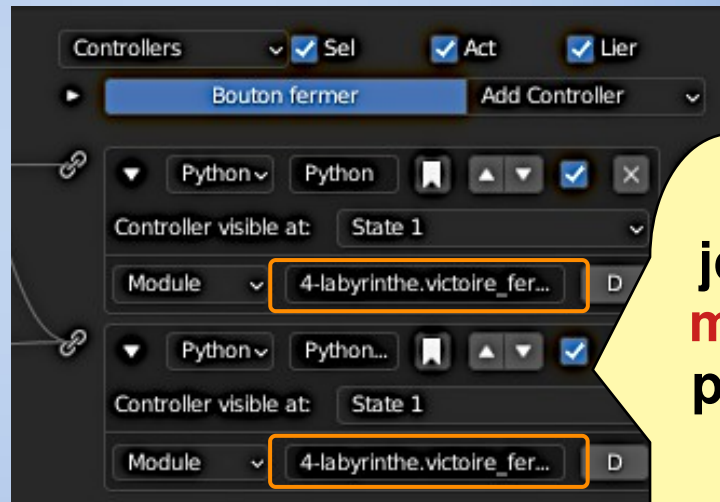
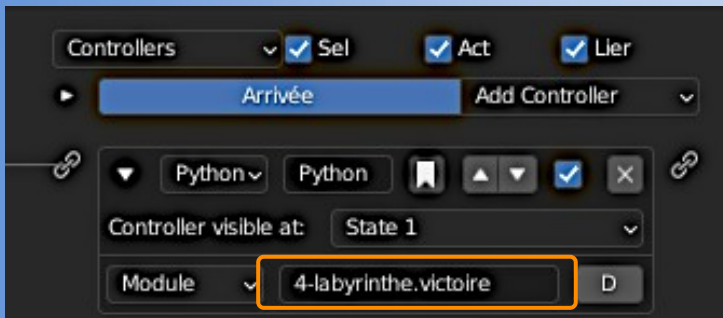
Si la bibliothèque pySerial n'est pas reconnue par UPBGE, il faut alors l'installer dans l'environnement local de UPBGE, pour cela voir l'**étape 7**.

# 4. 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** ».



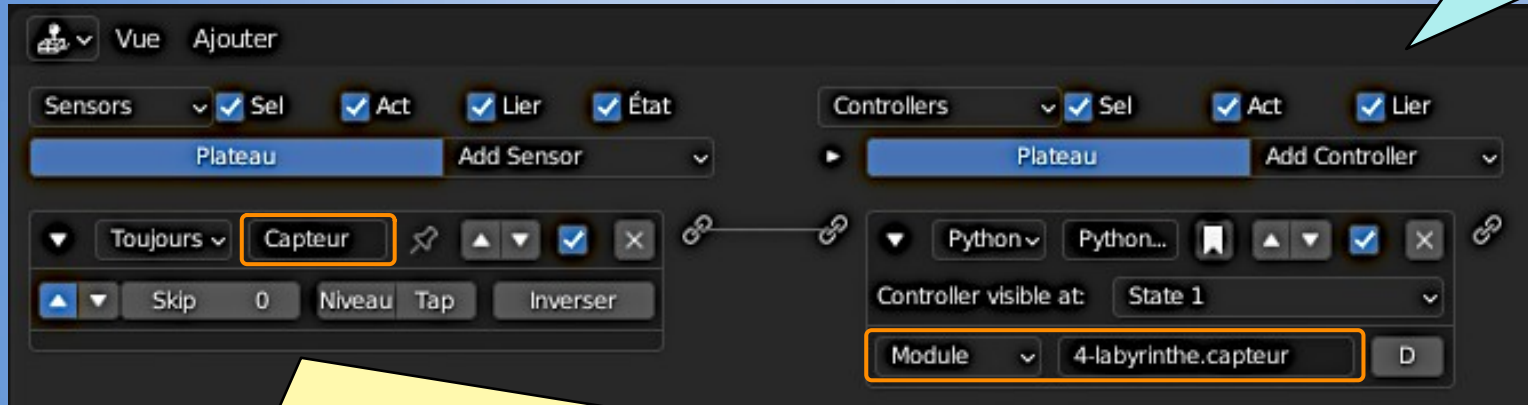
1 : Mettre à jour le nom des **modules Python** pour l'ensemble des **bricks logiques**

Renommer les noms « **2-labyrinthe.\*** » en « **4-labyrinthe.\*** »

# 4. Déplacer le plateau avec la centrale inertielle



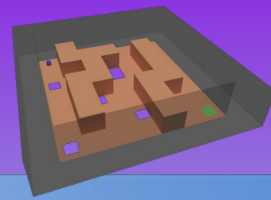
Briques logiques de **Plateau**



## 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.captteur**  
(ancien nom : « 2-labyrinthe.clavier »)

# 4. Déplacer le plateau avec la centrale inertielle



## 3 : Ajout l'importation de la bibliothèque

```
import serial
```

```
import bge # Bibliothèque Blender Game Engine (BGE)
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

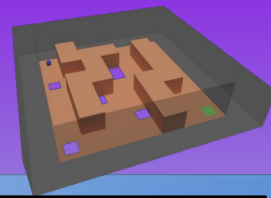
# Communication avec la carte Arduino
serial_baud = 115200
# serial_comm = serial.Serial('COM4',serial_baud, timeout=0.016) # Windows
serial_comm = serial.Serial('/dev/ttyACM1',serial_baud, timeout=0.016) # GNU/Linux
print (serial_comm)
```

4-labyrinthe.py

## 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)**

# 4. Déplacer le plateau avec la centrale inertielle



```
#####
# Gestion de la centrale inertielle (capteur IMU (inertial measurement unit))
#####

# Lecture du capteur IMU
def capteur(cont):
    obj = cont.owner # obj est l'objet associé au contrôleur donc 'Plateau'
    obj_bille = scene.objects['Bille']
    echelle = 0.2 # Facteur d'échelle entre la capteur et la scène 3D
    ecart=0.001 # Écart maxi sur la rotation

# Touche ESC -> Quitter
keyboard = bge.logic.keyboard
if keyboard.inputs[bge.events.ESCKEY].status[0] == ACTIVATE:
    serial_comm.close()
    bge.logic.endGame()

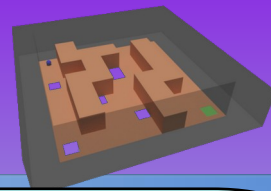
# Lecture de la liaison série : programme Arduino : 4-labyrinthe-imu.ino
serial_msg_in = str(serial_comm.readline())

# Roulis/Roll(x) et Tangage/Pitch(y)
if serial_msg_in.find(",") > 0:
    txt = serial_msg_in.split(',',2)
    x_txt = txt[0][2:]
    y_txt = txt[1][:5]
    if x_txt != " NAN" and y_txt != " NAN": # NAN : Not A Number
        x=- (float(x_txt)/57.3)*echelle # 1/ 360 / (2 * pi)
        y=- (float(y_txt)/57.3)*echelle # 1/ 360 / (2 * pi)
        while abs(x-obj.worldOrientation.to_euler().x) > ecart :
            obj.applyRotation((x-obj.worldOrientation.to_euler().x, 0,
                                -obj.worldOrientation.to_euler().z), False)
        while abs(y-obj.worldOrientation.to_euler().y) > ecart :
            obj.applyRotation((0, y-obj.worldOrientation.to_euler().y,
                                -obj.worldOrientation.to_euler().z), False)
```

4-labyrinthe.py

5 : Créer  
la fonction  
capteur

# 4. Déplacer le plateau avec la centrale inertielle



```
#####  
# Gestion du clavier  
#####  
  
# Flèches pour tourner le plateau sur les axes Y et X avec une mise à 0 sur l'axe Z  
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, -obj.worldOrientation.to_euler().z), False)  
  
    # Flèche bas - Down arrow  
    if keyboard.inputs[bge.events.DOWNARROWKEY].status[0] == ACTIVATE:  
        obj.applyRotation((resolution, 0, -obj.worldOrientation.to_euler().z), False)  
  
    # Flèche gauche - Left arrow  
    if keyboard.inputs[bge.events.LEFTARROWKEY].status[0] == ACTIVATE:  
        obj.applyRotation((0, -resolution, -obj.worldOrientation.to_euler().z), False)  
  
    # Flèche droit - Right arrow  
    if keyboard.inputs[bge.events.RIGHTARROWKEY].status[0] == ACTIVATE:  
        obj.applyRotation((0, resolution, -obj.worldOrientation.to_euler().z), False)
```

4-labyrinthe.py

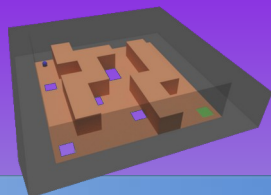
6 : Supprimer  
la fonction  
clavier

7 : Tester le capteur  
[P]

```
Terminal - phroy@debian: /mnt/fe77fe04-4eb8-4354-af...  
Fichier Édition Affichage Terminal Onglets Aide  
Blender Game Engine Started  
Serial<id=0x7fae538b9a30, open=True>(port='/dev/ttyACM0', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.016, xonxoff=False, rtscts=False, dsrdtr=False)  
Blender Game Engine Finished  
Debug: 1920, 957  
rcti: : xmin 0, xmax 1919, ymin 20, ymax 976 (1919x956)  
  
Blender Game Engine Started  
Serial<id=0x7fae53817ac0, open=True>(port='/dev/ttyACM0', baudrate=115200, bytesize=8, parity='N', stopbits=1, timeout=0.016, xonxoff=False, rtscts=False, dsrdtr=False)  
Chuuuu....te  
Chuuuu....te  
Blender Game Engine Finished
```



# 5. Afficher la position de la bille sur la matrice de leds



La **matrice de leds** va nous permettre d'**afficher la position de la bille** avec les coordonnées en x et en y. Avec **une matrice 8x8**, la valeur de x et de y est de **0 à 7**. La liaison série va être utilisée dans **les deux sens** :

- Arduino → UPBGE : angle de roulis et de tangage (ça c'est déjà fait !),
- UPBGE → Arduino : position en x et en y.

```
#include "Wire.h"
#include "MPU6050.h"
#include "Grove_LED_Matrix_Driver_HT16K33.h" }

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

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

...

/*****
 * Led Matrix - I2C
 *****/

Matrix_8x8 matrix; }

/*****
 * Communication série
 *****/

String serial_msg = ""; // Message
bool serial_msg_complet = false; // Flag de message complet }

/*****
 * Initialisation
 *****/
```

**4-labyrinthe-imu.ino**

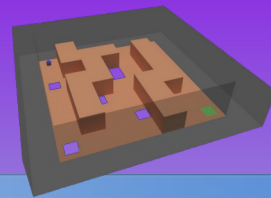
**1 : Ajout de l'importation de la bibliothèque de la matrice de leds**

**2 : Déclarer la matrice de leds**

**3 : Déclarer les variables pour le message entrant (UPBGE → Arduino)**



# 5. Afficher la position de la bille sur la matrice de leds



```
/* *****  
 * Initialisation  
***** */  
  
void setup() {  
  
  // Liaison série  
  Serial.begin(115200);  
  
  // IMU  
  Wire.begin();  
  accelgyro.initialize();  
  
  // Led Matrix  
  matrix.init();  
  matrix.setBrightness(0);  
  matrix.setBlinkRate(BLINK_OFF);  
  matrix.clear();  
  matrix.display();  
  
}
```

4-labyrinthe-imu.ino  
dans le setup()

4 : Initialiser la  
matrice de leds

```
/* *****  
 * Évènement provoqué par UPBGE (via la liaison série)  
***** */  
  
void serialEvent() {  
  while (Serial.available()) {  
    char inChar = (char)Serial.read();  
    serial_msg += inChar;  
    if (inChar == '\n') {  
      serial_msg_complet = true;  
    }  
  }  
}
```

4-labyrinthe-imu.ino  
à la fin du programme

5 : Lors d'un élèvement sur le port  
série ( fonction `serialEvent()` )

- Récupérer le message texte
- mettre à 1 le **flag** du message complet





# 5. Afficher la position de la bille sur la matrice de leds



```
/* *****  
 * Boucle principale  
 * *****  
 */  
  
void loop() {  
  ...  
  
  /* *****  
   * Led Matrix : UPBGE -> Arduino  
   * *****/  
  
  if (serial_msg_complet) {  
    matrix.clear();  
    int xy = serial_msg.toInt(); // Message par chiffre : XY  
  
    // Position de la bille en x et y  
    if (xy < 90) {  
      int x = xy/10;  
      int y = xy-(x*10);  
      matrix.writePixel(x, y, true);  
    }  
  
    // Chute  
    if (xy == 91) {  
      matrix.writeOnePicture(0x81423c0000666600);  
      matrix.display();  
      delay(1000);  
      Serial.println("start"); // Relance le jeu  
    }  
  
    // Victoire  
    if (xy == 92) matrix.writeOnePicture(0x003c428100666600);  
  
    matrix.display();  
    serial_msg = "";  
    serial_msg_complet = false;  
  }  
}
```

4-labyrinthe-imu.ino  
à la fin de la  
boucle principale

6 : Exécuter l'ordre  
associé au message  
entrant (venant de UPBGE)

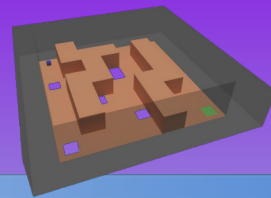
- Lire le message entrant
- si le message <90 afficher la position de la bille
- si le message = 91 afficher l'icône de la chute et envoie le message « start »
- si le message = 92 afficher l'icône de la victoire

Les images de leds ont été codées avec l'[outil en ligne de Xantorohara](#).

7 : Téléverser le programme vers la carte



# 5. Afficher la position de la bille sur la matrice de leds



Au niveau du **programme Python**, il y a trois modifications :

- **envoyer le message XY** (une valeur de 0 à 77) de la **position de la bille** par la liaison série,
- envoyer les messages « **91** » ou « **92** » en cas respectivement de **chute** ou de **victoire**,
- repositionner la bille au départ si nous recevons le message « **start** » de la carte Arduino.

## 4-labyrinthe.py

```
#####
# Gestion de la centrale inertielle (capteur IMU (inertial measurement unit))
#####

# Lecture du capteur IMU
def capteur(cont) :
    obj = cont.owner # obj est l'objet associé au contrôleur donc 'Plateau'
    obj_bille = scene.objects['Bille']
    echelle = 0.2 # Facteur d'échelle entre la capteur et la scène 3D
    ecart=0.001 # Écart maxi sur la rotation

    # Touche ESC -> Quitter
    keyboard = bge.logic.keyboard
    if keyboard.inputs[bge.events.ESCKEY].status[0] == ACTIVATE:
        serial_comm.close()
        bge.logic.endGame()

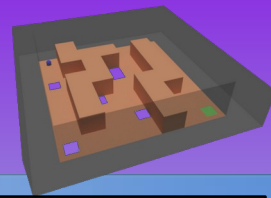
    # Lecture de la liaison série : programme Arduino : 4-labyrinthe-imu.ino
    serial_msg_in = str(serial_comm.readline())

    # Mettre la bille à la position de départ avec une vitesse nulle
    if serial_msg_in.find("start")>0:
        if obj_bille['victoire'] or obj_bille['chute']:
            depart()

    # Roulis/Roll(x) et Tangage/Pitch(y)
    ...
```

**8 : Remettre à la bille au départ si le message entrant est « start » (dans la fonction capteur)**

# 5. Afficher la position de la bille sur la matrice de leds



4-labyrinthe.py

```
#####
# Gameplay
#####
...

# Cycle (boucle de contrôle de la bille)
def cycle(cont):
    obj = cont.owner # obj est l'objet associé au contrôleur donc 'Bille'
    obj['z']=obj.worldPosition.z # la propriété z est mis à jour avec la position globale en z de la bille

# Écriture de la position de la bille sur la liaison série
if obj['victoire'] == False and obj['chute'] == False:
    # obj.worldPosition.x varie de -3.5 à 3.5 -> les leds pour X vont de 7 à 0
    # obj.worldPosition.y varie de 3.5 à -3.5 -> les leds pour Y vont de 0 à 7
    obj['Lx']=-1*round(obj.worldPosition.x-3.5) # de 7 à 0
    if obj['Lx']<0: obj['Lx']=0
    if obj['Lx']>7: obj['Lx']=7
    obj['Ly']=-1*round(obj.worldPosition.y-3.5) # de 0 à 7
    if obj['Ly']<0: obj['Ly']=0
    if obj['Ly']>7: obj['Ly']=7
    serial_msg_out = str(obj['Lx'])+str(obj['Ly'])+"\n"
    serial_comm.write(serial_msg_out.encode())

# Si l'altitude de bille < -10 et pas de victoire -> chute
if obj['z'] < -10 and obj['victoire'] == False:
    print ("Chuuuu...te")
    depart () # Replacer la bille au départ

# Afficher image de chute sur la matrice de leds
serial_msg_out = "91\n"
serial_comm.write(serial_msg_out.encode())
obj['chute'] = True
```

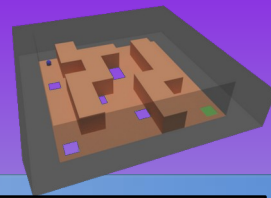
9 : Envoyer le message  
texte des coordonnées  
en x et y de la bille  
(dans la fonction **cycle**)

10 : Envoyer le message « 91 » lors de  
la chute et supprimer les 2 lignes  
**print** et **depart ()**  
(toujours dans la fonction **cycle**)

Le départ de la bille est  
maintenant provoqué  
par le message « **start** ».



# 5. Afficher la position de la bille sur la matrice de leds



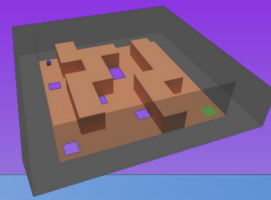
```
#####  
# Gameplay  
#####  
...  
  
# Victoire (collision de la bille avec l'arrivée)  
def victoire(cont):  
  
    # Afficher image de victoire sur la matrice de leds  
    serial_msg_out = "92\n"  
    serial_comm.write(serial_msg_out.encode())  
    scene.objects['Bille']['victoire'] = True  
  
    # Animation du Panneau victoire  
    scene.objects['Panneau victoire'].setVisible(True, True) # Afficher le panneau de la victoire  
    scene.objects['Panneau victoire - plan'].restorePhysics() # Restaurer la physique du panneau cliquable  
    start = 1  
    end = 100  
    layer = 0  
    priority = 1  
    blendin = 1.0  
    mode = bge.logic.KX_ACTION_MODE_PLAY  
    layerWeight = 0.0  
    ipoFlags = 0  
    speed = 1  
    scene.objects['Panneau victoire'].playAction('Panneau victoireAction', start, end, layer,  
                                                priority, blendin, mode, layerWeight, ipoFlags, speed)
```

**11 : Envoyer le message « 92 » lors de la victoire (dans la fonction **victoire**)**

**4-labyrinth.py**

**12 : Tester la matrice de leds [P]**

# 6. 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 : `serial_comm = serial.Serial('COM4', serial_baud, timeout=0.016)`. 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` ».

## 1 : Créer le fichier `labyrinthe_carte.py`

```
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

# 6. Détecter automatiquement le micro-contrôleur



**labyrinthe\_carte.py**

```
# Établir la communication avec la carte par la liaison série avec une vitesse
def init_serial(speed=115200):
    [port, carte_name] = autoget_port()
    if port is None:
        print("Carte Arduino/microbit introuvable")
        return None
    else:
        serial_comm = serial.Serial(port, speed, timeout=0.016)
        if serial_comm is None:
            print("Communication avec Carte Arduino/microbit impossible")
            return None
        else:
            print("Communication avec Carte Arduino/microbit établie sur "+port+" à la vitesse "
                  +str(speed)+" bauds")
            return serial_comm
```

**4 : Fonction d'initialisation de la communication avec la carte par la **liaison série****

# 6. Détecter automatiquement le micro-contrôleur



Nous allons maintenant utiliser la fonction `init_serial()` dans **4-labyrinthe.py**.

```
import bge # Bibliothèque Blender Game Engine (BGE)
import pserial # Liaison série
import labyrinthe_carte # Liaison avec la carte
```

```
#####
# 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)
```

```
# Détection de la carte avec la liaison série
serial_comm = labyrinthe_carte.init_serial()
if serial_comm is None:
    bge.logic.endGame()
```

5 : Ajout l'importation de notre module

```
import labyrinthe_carte
```

6 : Supprimer la **création manuelle** de l'objet `serial_comm`

7 : Ajouter la **création automatique** de l'objet `serial_comm`

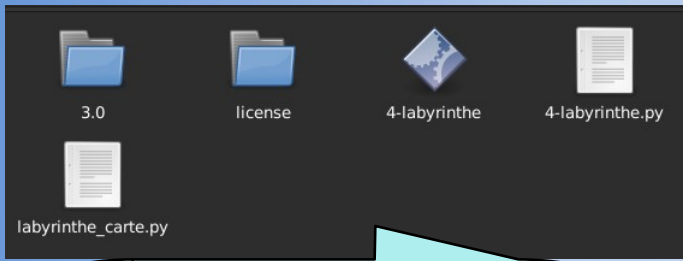
4-labyrinthe.py

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

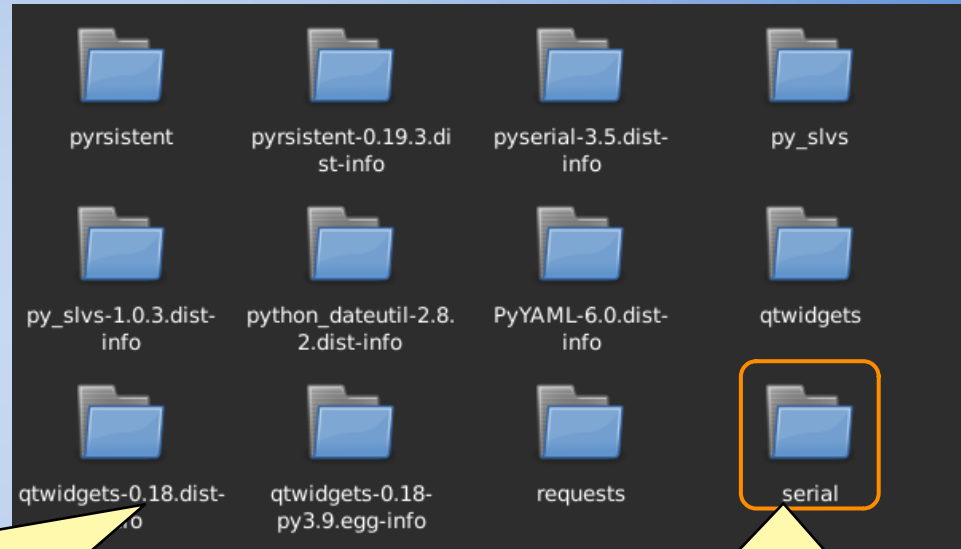
# 7. Inclure pySerial dans la distribution de l'exécutable



Pour pouvoir faire fonctionner l'**exécutable (game runtime)**, il faut que la bibliothèque **pySerial** soit présente dans l'**environnement local de l'exécutable**.



Répertoire de l'exécutable



1 : Aller dans le répertoire contenant les bibliothèques de l'environnement local de l'exécutable

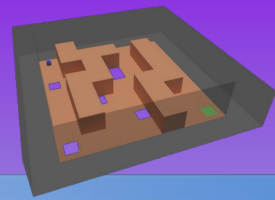
**Chemin** à partir du répertoire de l'exécutable :  
**3.0/python/lib/python3.9/site-packages/**

2 : Vérifier la présence du répertoire **serial**.

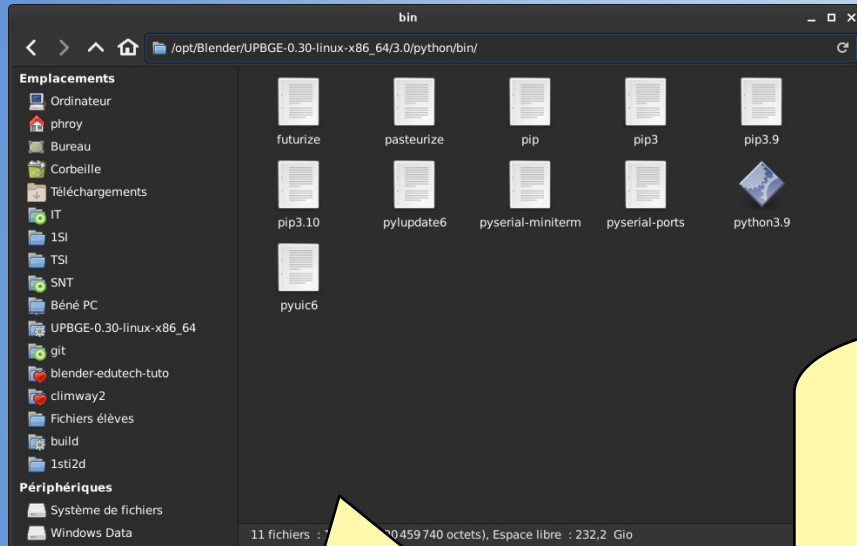
- **Si c'est le cas, c'est bon cela devrait fonctionner**
- **Sinon voir la suite**



# 7. Inclure pySerial dans la distribution de l'exécutable



Si la bibliothèque **serial** n'est pas présente, il faut l'installer dans l'**environnement Python de UPBGE**.



4 : Télécharger le script d'installation du gestionnaire de paquet **pip** [get-pip.py](#) et copier le dans le répertoire des binaires Python de UPBGE

5 : Toujours dans le répertoire des binaires Python de UPBGE, ouvrir une console et installer le gestionnaire de paquet Pip :

**GNU/Linux** : `$ ./python3.9 get-pip.py`

**Windows** : `python.exe get-pip.py`

3 : Aller dans le répertoire des binaires Python de UPBGE

**GNU/Linux** : `UPBGE-0.30-linux-x86_64/3.0/python/bin/`

**Windows** : `UPBGE-0.30-windows-x86_64\3.0\python\bin`

# 7. Inclure pySerial dans la distribution de l'exécutable



## 6 : Installer la bibliothèque pySerial

### GNU/Linux :

- avec la console ouverte
- toujours dans le répertoire des binaires Python de UPBGE,
- `$ ./pip install pyserial`

### Windows :

- avec la console ouverte
- aller dans le répertoire 'Scripts' :  
`cd C:\foo\UPBGE-0.30-windows-x86_64\3.0\python\Scripts`
- `pip.exe install pyserial`

**7 : Générer un nouveau exécutable (game runtime)**, la bibliothèque dit être maintenant incluse dans l'environnement local de l'exécutable.

**Bravo ! Vous êtes arrivé à l'issue de ce tutoriel.**