Labyrinthe à bille
Créer une scène 3D interactive

Tutoriel 3 : Interfacer avec Arduino avec pyFirmata

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 le **protocole Firmata**. 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).

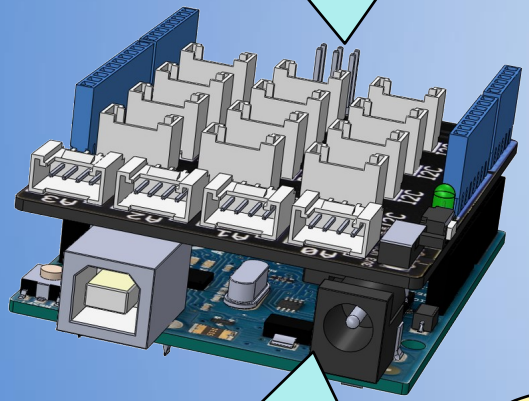
Le tutoriel se décompose en 6 étapes :

- 1. Préparer la carte Arduino
- 2a. Installation de la bibliothèque pyFirmata sous GNU/Linux
- 2b. Installation de la bibliothèque pyFirmata sous Windows
- 3. Déplacer le plateau avec 4 boutons binaires
- 4. Déplacer le plateau avec un joystick analogique
- 5. Détecter automatiquement le micro-contrôleur
- 6. Inclure pyFirmata dans la distribution de l'éxécutable

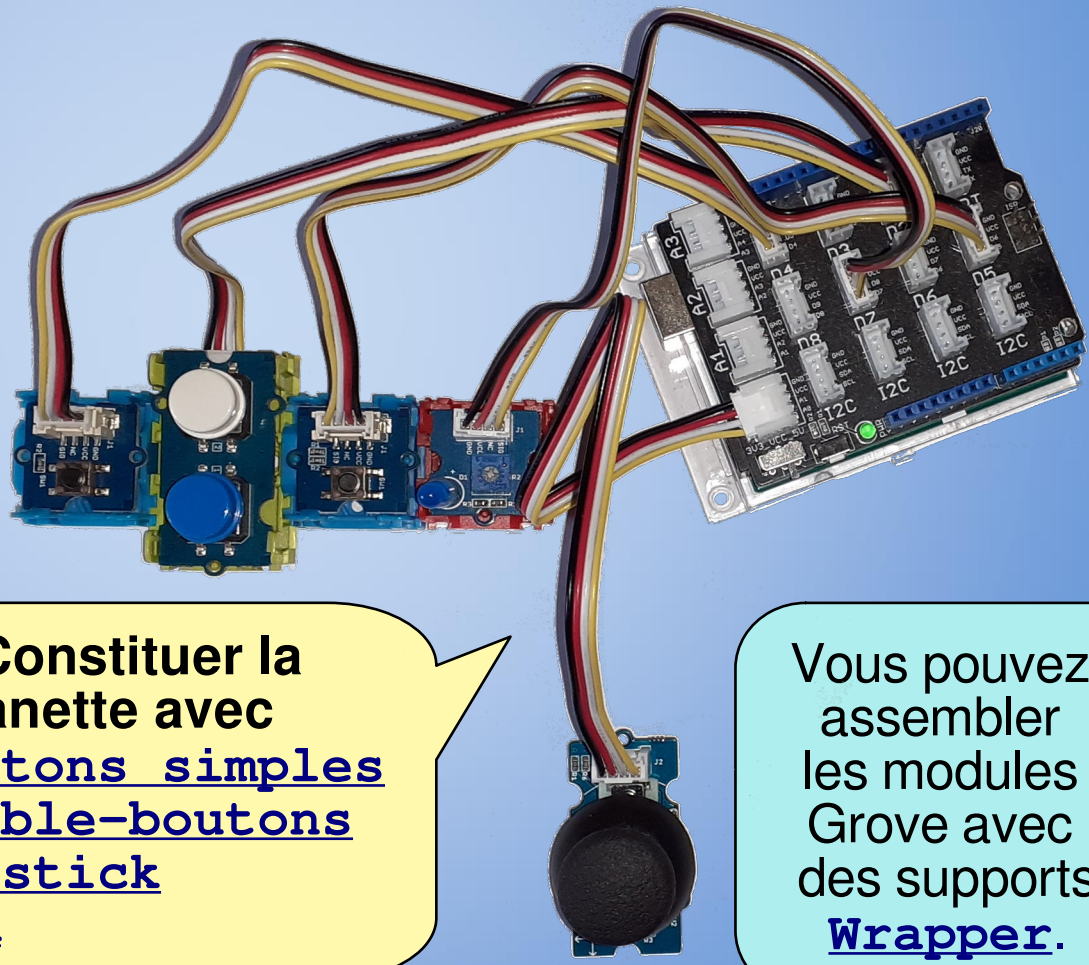
1. Préparer la carte Arduino

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 : Constituer la manette avec

- 2 `boutons simples`
- 1 `double-boutons`
- 1 `joystick`
- 1 `led`

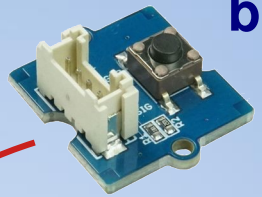
Vous pouvez assembler les modules Grove avec des supports `Wrapper`.

1. Préparer la carte Arduino

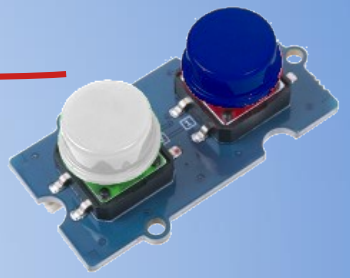
2 : Procéder au brochage des modules Grove

Led plateau en mouvement : broche 7

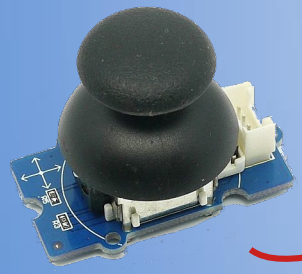
Flèche gauche : broche 4



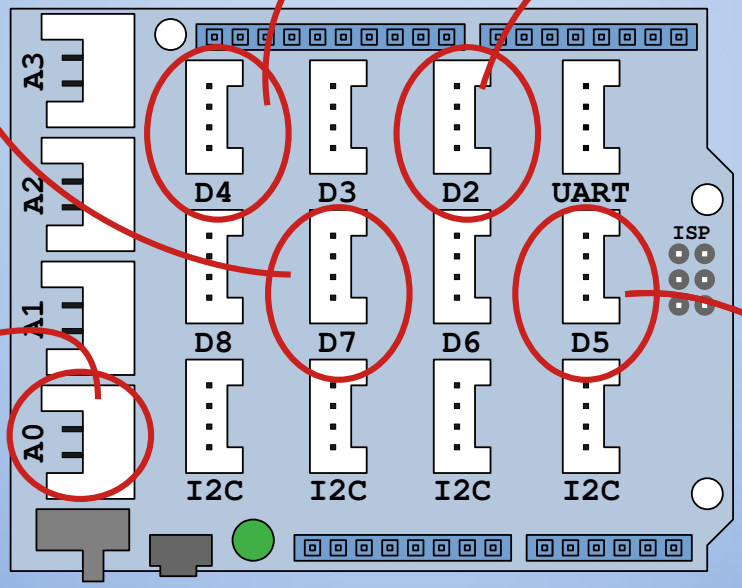
Flèche haut : broche 2



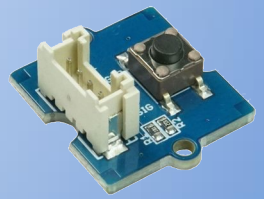
Flèche bas : broche 3



Joystick :
X : broche A0
Y : broche A1

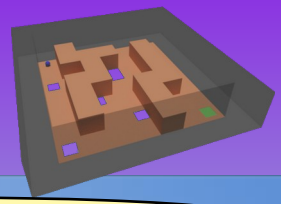


Flèche droit : broche 5

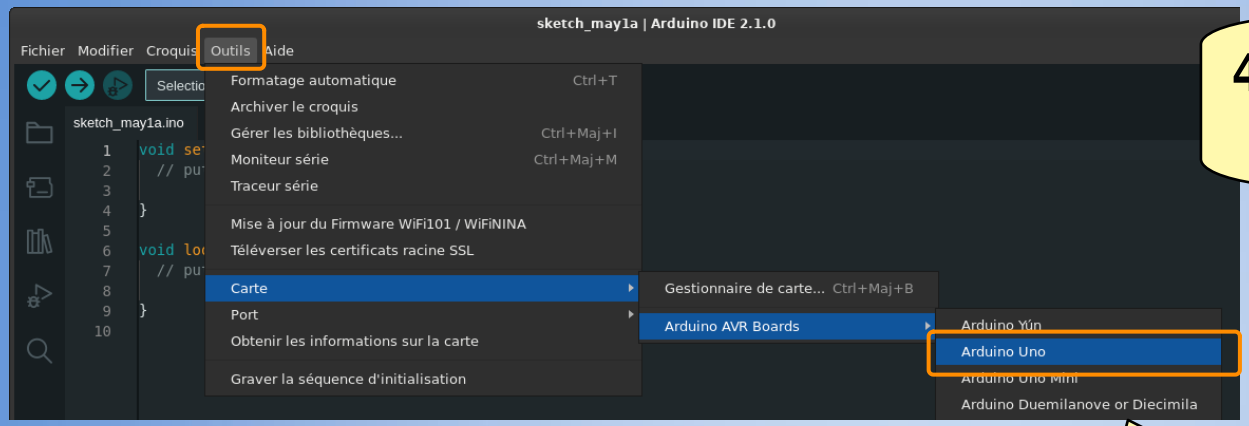


Le tutoriel vous propose la commande du plateau par les boutons et le joystick, mais il possible de ne faire qu'un seul type de commande sur les deux.

1. Préparer la carte Arduino

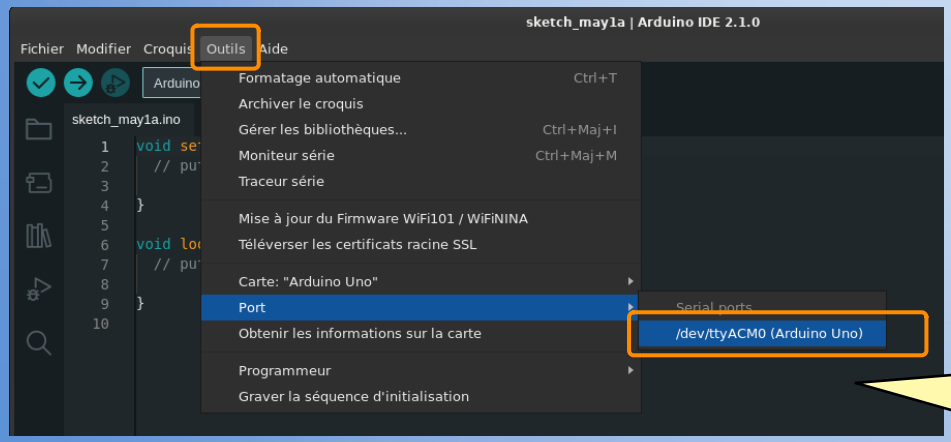


3 : Brancher la carte Arduino sur l'ordinateur avec le cordon USB



4 : Lancer le programme Arduino IDE

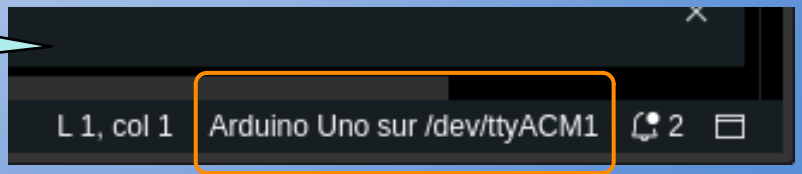
Arduino IDE est un éditeur libre disponible sur le site d' **Arduino**



5 : Définir le type de carte sur Arduino Uno

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

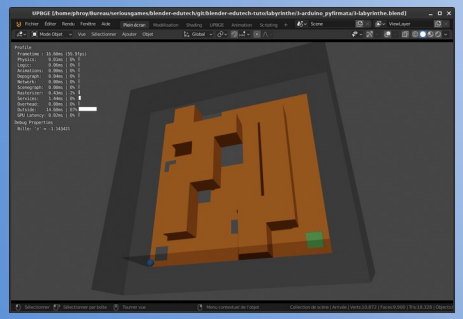
La **barre d'état** nous indique la carte et le port actifs



1. Préparer la carte Arduino

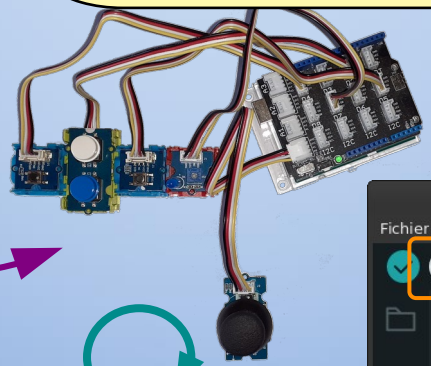
Firmata est un **protocole de communication** entre un ordinateur et un microcontrôleur par le port série. Le microcontrôleur en mis en « mode écoute » afin de pouvoir lire et écrire sur les broches binaires ou analogiques à partir de l'ordinateur.

L'intérêt de Firmata est l'absence de programmation spécifique de la carte.



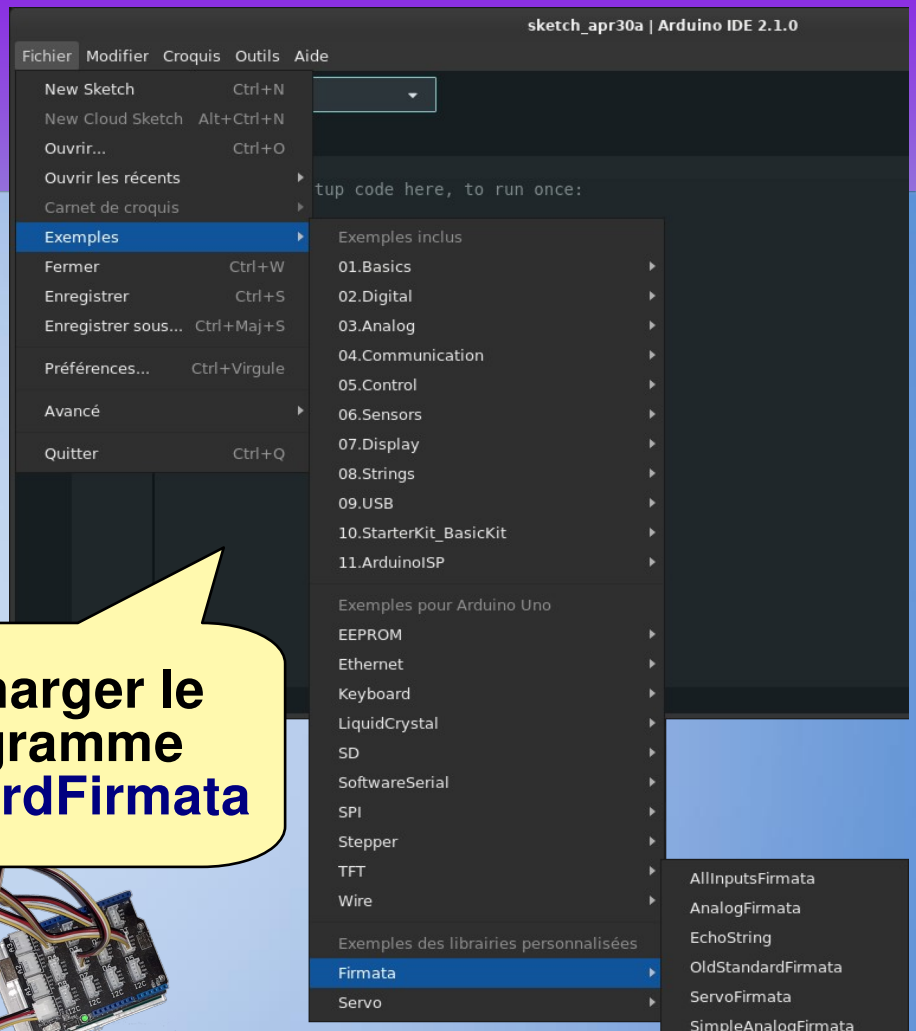
Programme ordinateur : 3-labyrinthe.py

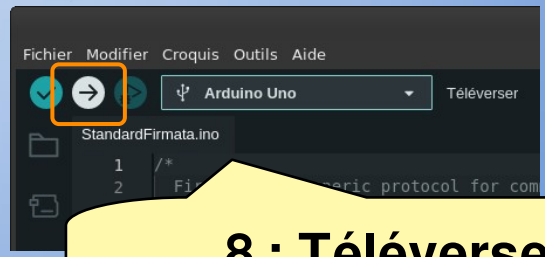
Protocole Firmata (USB)



Programme carte : StandardFirmata.ino

7 : Charger le programme StandardFirmata



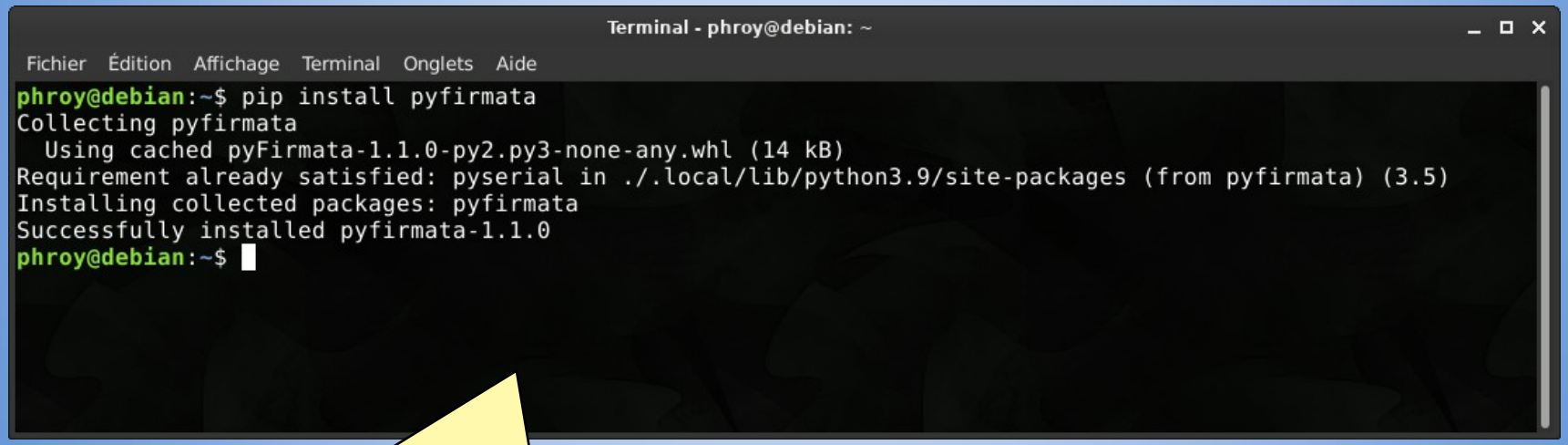


8 : Téléverser le programme vers la carte

2a. Installation de la bibliothèque pyFirmata sous GNU/Linux

La bibliothèque **pyFirmata** permet d'utiliser le protocole **Firmata** dans un programme Python. Il faut donc installer la bibliothèque **pyFirmata**.

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 pyFirmata
Dans un **terminal**
exécuter la commande :
`pip install pyfirmata`

Si la bibliothèque pyFirmata n'est pas reconnue par UPBGE, il faut alors l'installer dans l'environnement local de UPBGE, pour cela voir l'**étape 6**.

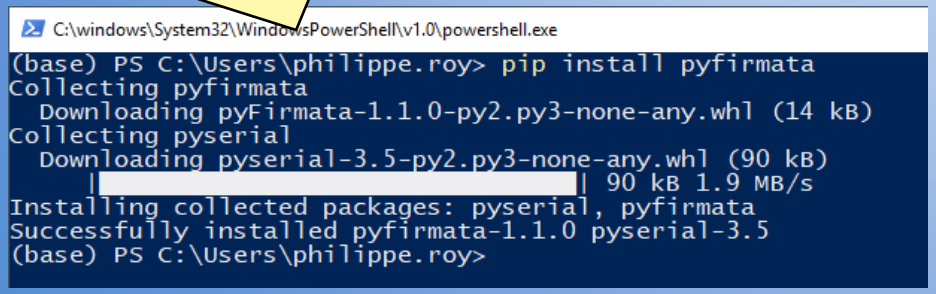
2b. Installation de la bibliothèque pyFirmata sous Windows

La bibliothèque **pyFirmata** permet d'utiliser le protocole **Firmata** dans un programme Python. Il faut donc installer la bibliothèque **pyFirmata**. 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 pyFirmata
Dans le **terminal**
exécuter la commande :
`pip install pyfirmata`

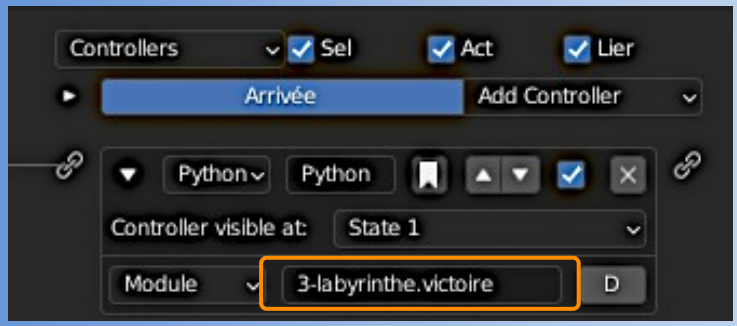


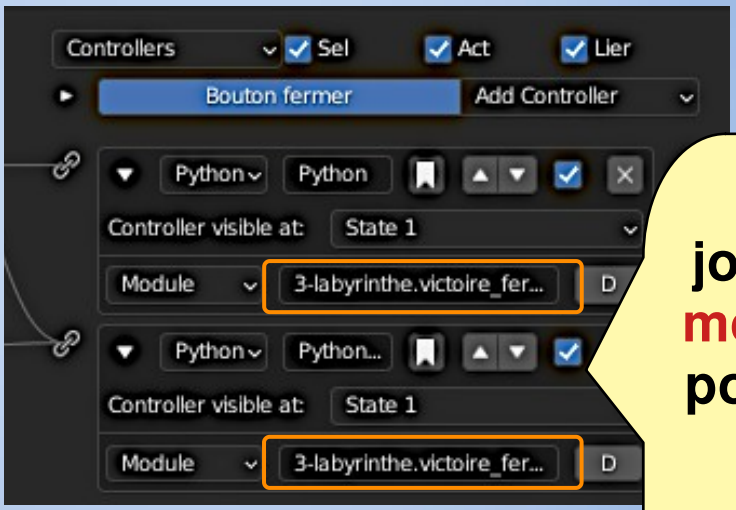
Si la bibliothèque pyFirmata n'est pas reconnue par UPBGE, il faut alors l'installer dans l'environnement local de UPBGE, pour cela voir l'**étape 6**.

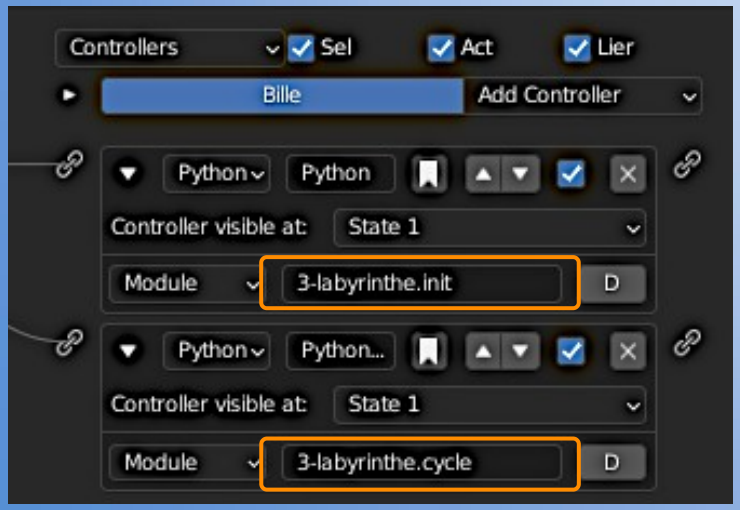
3. Déplacer le plateau avec 4 boutons binaires

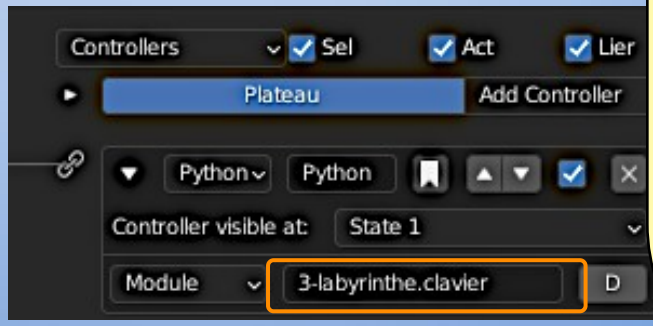
Les fichiers de départ de ce tutoriel sont les fichiers résultats du tutoriel 2. Il faut donc :

- copier et renommer « **2-labyrinthe.blend** » en « **3-labyrinthe.blend** »,
- copier et renommer « **2-labyrinthe.py**» en « **3-labyrinthe.py** ».



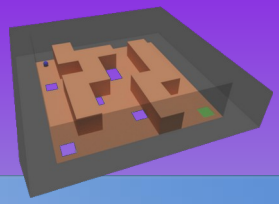


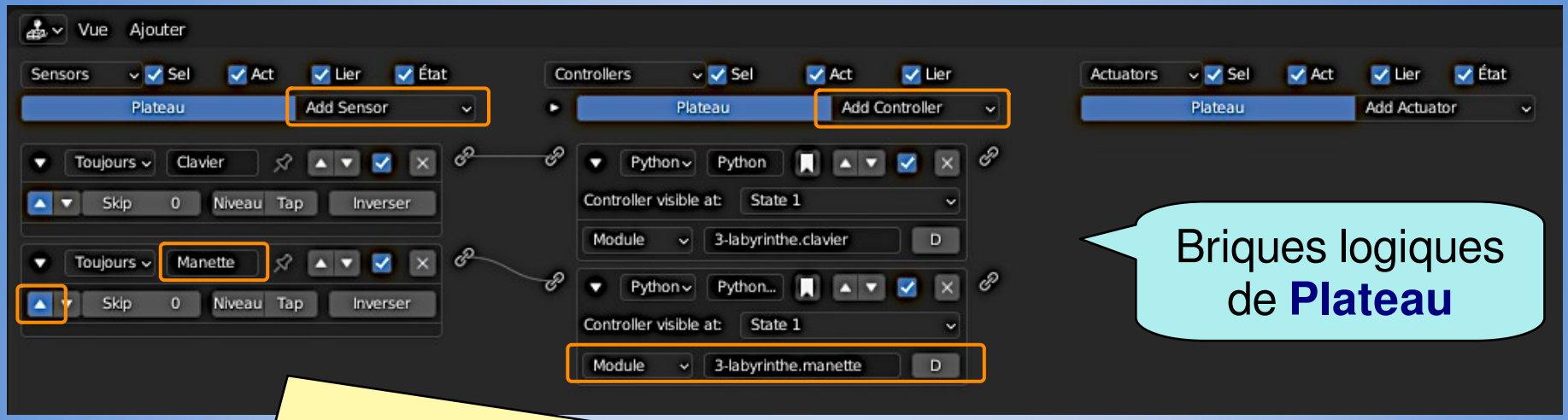




1 : Mettre à jour le nom des modules Python pour l'ensemble des briques logiques
Renommer les noms « **2-labyrinthe.*** » en « **3-labyrinthe.*** »

3. Déplacer le plateau avec 4 boutons binaires

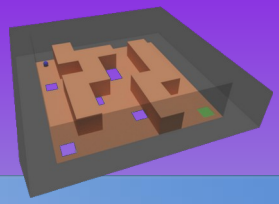




2 : Mettre UPBGE à l'écoute de la manette Arduino

- **Ajouter** un **Capteur Toujours** avec le **Pulse True Level (▲)**
- **renommer** le capteur avec **Manette**
- **ajouter** le **Module Python** avec la fonction **3-labyrinthe.manette**

3. Déplacer le plateau avec 4 boutons binaires



Une carte de référence de la bibliothèque **pyFirmata** est fournie avec le document joint « **DT - Carte de référence pyFirmata** ». Il détaille comment définir une broche, lire ou définir sa valeur.

3-labyrinthe.py

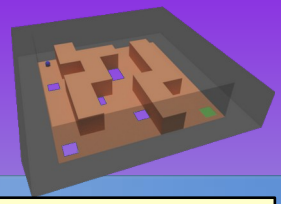
```
import bge # Bibliothèque Blender Game Engine (UPBGE)
import pyfirmata # Protocole Firmata

##########################################################################
# 3-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
##########################################################################

# 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
led = carte.get_pin('d:7:o')
```

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

4 : Créer la carte et son itérateur
Le nom du port (COM4, /dev/ttyACM0, ...) est indiqué par **IDE Arduino**

5 : Créer les broches
Une broche par bouton et led

3. Déplacer le plateau avec 4 boutons binaires

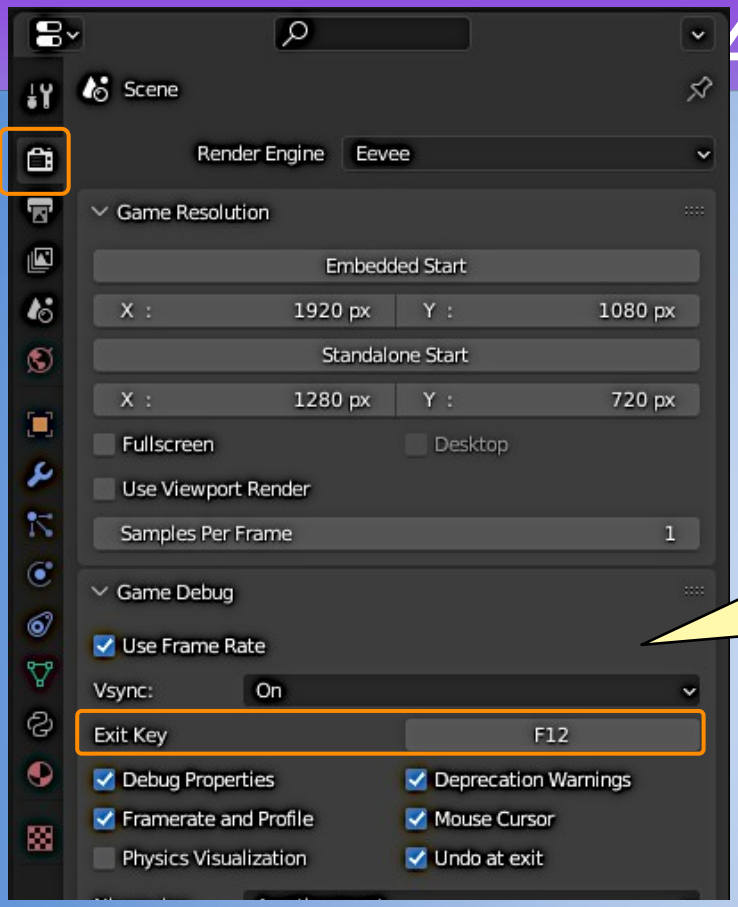


```
##############################################################################
# Gestion de la manette Arduino
##############################################################################

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)

##############################################################################
# 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

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

6 : Créer la fonction manette

7 : Ajouter la touche ESC à l'écoute du clavier (fonction clavier) pour quitter le programme

3. Déplacer le plateau
4 boutons binaires

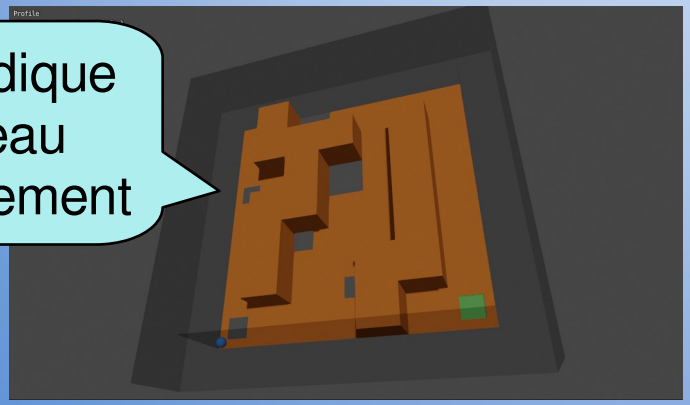


Afin de **fermer proprement le canal de communication** il faut qu'en fin de programme nous appelons la fonction `carte.exit()`. Sinon le canal risque d'être indisponible et donc il faudra alors relancer le noyau Python. **Touche ESC** a été programmée pour **fermer le canal de communication** et quitter le programme.

8 : Définir la nouvelle touche d'interruption du programme sur F12

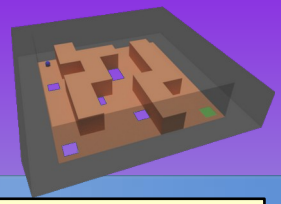
9 : Tester les boutons [P]

La led indique le plateau en mouvement



```
Terminal - phroy@debian: /mnt/fe
Fichier  Édition  Affichage  Terminal  Onglets  Aide

Blender Game Engine Started
Communication Carte Arduino établie
Chuuuu.....te
Blender Game Engine Finished
Debug: 1111, 663
rcti: : xmin 0, xmax 1110, ymin 20, ymax 682 (1110x662)

Blender Game Engine Started
Communication Carte Arduino établie
```

4. Déplacer le plateau avec un joystick analogique



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

...

# Définition du joystick

jstk_x = carte.get_pin('a:0:i')
jstk_y = carte.get_pin('a:1:i')
```

1 : Créer les broches du joystick
Une broche analogique par axe

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

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

    ...

    # Joystick : axe X
    # de 0,25 à 0,75 avec une zone morte entre 0,48 et 0,52
    # print (jstk_x.read(), jstk_y.read())
    if jstk_x.read() is not None:
        if jstk_x.read() < 0.48 or jstk_x.read() > 0.52:
            resolution_prop = (jstk_x.read()-0.5)*(resolution/0.25)
            obj.applyRotation((resolution_prop, 0, -obj.worldOrientation.to_euler().z), False)
            led.write(True)

    # Joystick : axe Y
    if jstk_y.read() is not None:
        if jstk_y.read() < 0.48 or jstk_y.read() > 0.52:
            resolution_prop = (jstk_y.read()-0.5)*(resolution/0.25)
            obj.applyRotation((0, resolution_prop, -obj.worldOrientation.to_euler().z), False)
            led.write(True)
```

2 : Ajouter à la fonction manette la commande proportionnelle du joystick

3 : Tester le joystick [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** ».

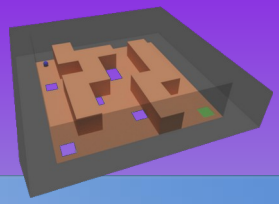
1 : Créer le fichier labyrinthe_carte.py

```
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

```
# 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
```

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

5. Détecter automatiquement le micro-contrôleur

Nous allons maintenant utiliser la fonction `init_firmata()` dans **3-labyrinthe.py**.

```
import bge # Bibliothèque Blender Game Engine (BGE)
import pyfirmata # Protocole Firmata
import labyrinthe_carte # Liaison avec la carte

################################################################################
# 3-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
# 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()
```

5 : Ajout l'importation de notre module `import labyrinthe_carte`

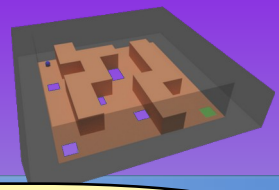
6 : Supprimer la création manuelle de l'objet carte

7 : Ajouter la création automatique de l'objet carte

3-labyrinthe.py

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

5. Détecter automatiquement le micro-contrôleur



labyrinthe_carte.py

4 : Fonction d'initialisation de la communication avec la carte par 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()
```

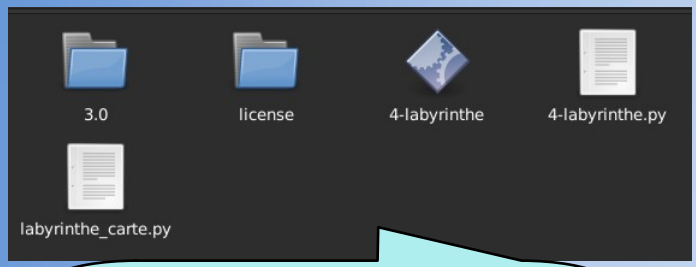
8 : Ajouter la création automatique de l'objet carte

3-labyrinthe.py

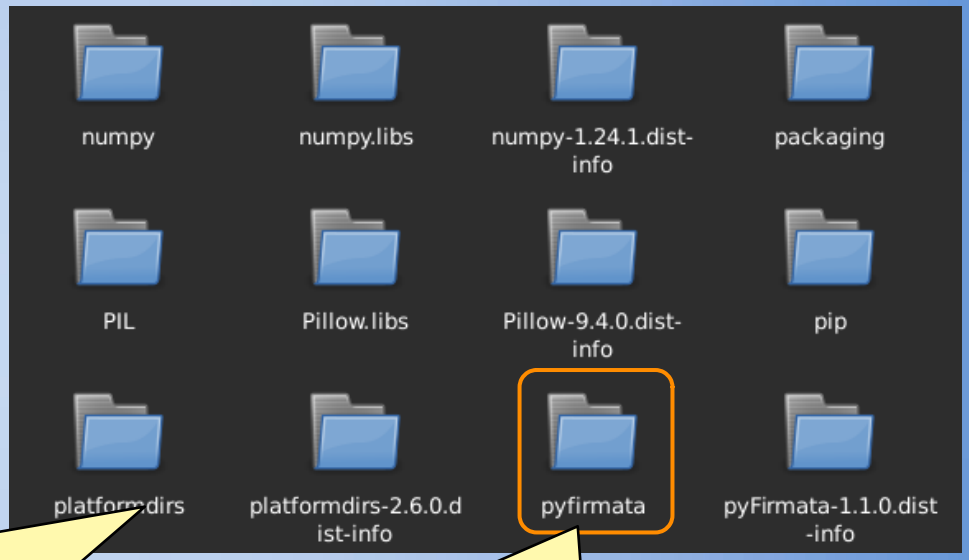
9 : Tester le détection automatique de la carte [P]

6. Inclure pyFirmata dans la distribution de l'exécutable

Pour pouvoir faire fonctionner l'**exécutable (game runtime)**, il faut que la bibliothèque **pyFirmata** 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

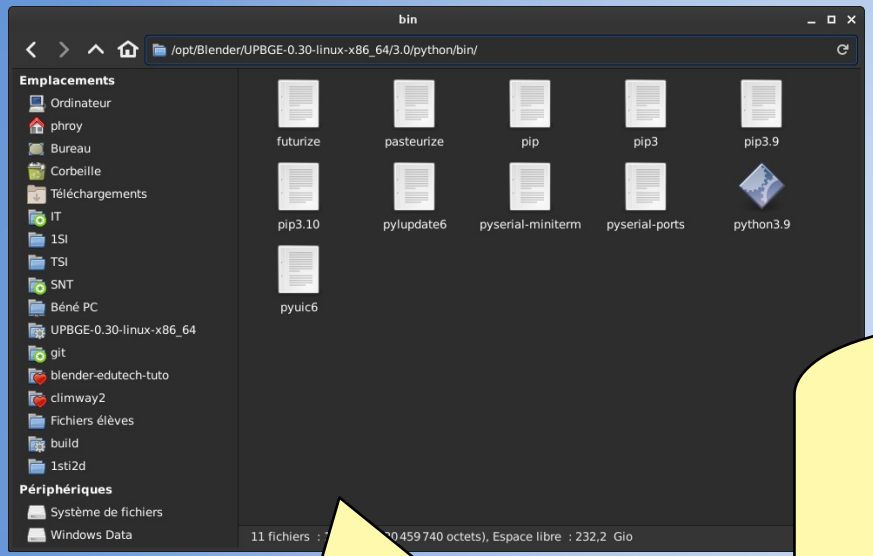
Chemin à partir du répertoire de l'exécutable :
3.0/python/lib/python3.9/site-packages/

2 : Vérifier la présence du répertoire pyfirmata

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

6. Inclure pyFirmata dans la distribution de l'exécutable

Si la bibliothèque **pyfirmata** 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

6. Inclure pyFirmata dans la distribution de l'exécutable

6 : Installer la bibliothèque pyFirmata

GNU/Linux :

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

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 pyfirmata`

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.