

# Labyrinthe à bille

## Créer une scène 3D interactive

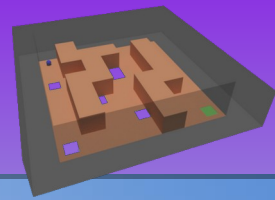
# Tutoriel 4 : Interfacer avec Arduino avec pySerial



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

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

# Objectif

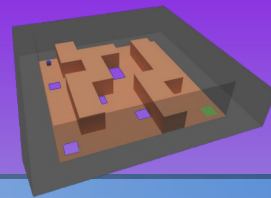


L'objectif de ce tutoriel est faire interagir les objets de la scène 3D (des objets virtuelles) à partir d'actions physiques mesurées par des capteurs. Les **capteurs** sont ici reliés à un **micro-contrôleur Arduino** par la **connectique Grove** et la **liaison série**. La guidance de ce tutoriel a 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 :

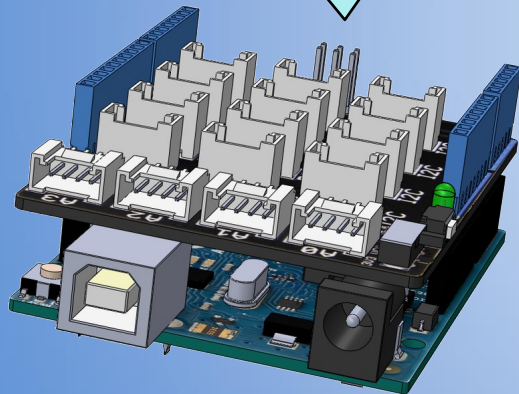
- [1. Préparer la carte Arduino](#)
- [2. Programmer la carte Arduino pour le capteur IMU](#)
- [3a. Installation de la bibliothèque pyFirmata sous GNU/Linux](#)
- [3b. Installation de la bibliothèque pyFirmata sous Windows](#)
- [4. Déplacer le plateau avec la centrale inertielle](#)
- [5. Détecter automatiquement le micro-contrôleur](#)
- [6. Distribuer l'exécutable](#)

# 1. Préparer la carte Arduino



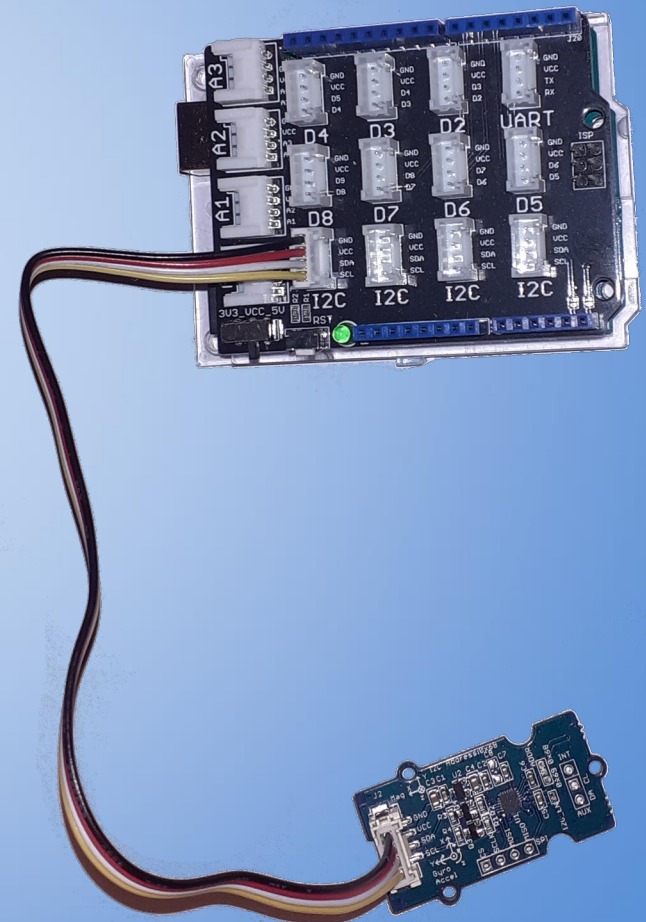
**Arduino** est une plateforme de conception et de fabrication d'objets électroniques interactifs. Le tutoriel utilise une **carte Uno** avec une **platine Grove** (voir le document joint « **DT - Grove pour Arduino** »).

Platine  
(shield) Grove

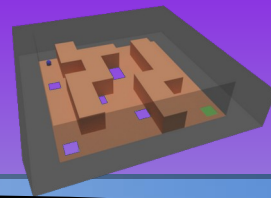


Micro-contrôleur  
Arduino Uno

1 : Procéder  
au brochage  
du capteur IMU  
sur un **port I2C**



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



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

#### 5 : Définir le type de carte sur **Arduino Uno**

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

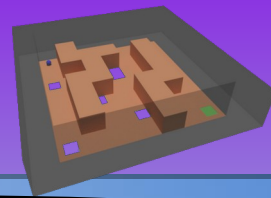
La **barre d'état** nous indique la carte et le port actifs

L 1, col 1

Arduino Uno sur /dev/ttyACM1

2

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



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

#### 5 : Définir le type de carte sur **Arduino Uno**

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

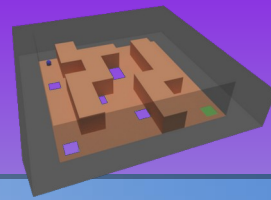
La **barre d'état** nous indique la carte et le port actifs

L 1, col 1

Arduino Uno sur /dev/ttyACM1

2

# 3a. Installation de la bibliothèque pyFirmata sous GNU/Linux



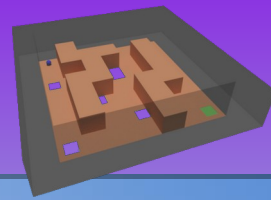
La bibliothèque **pyFirmata** permet d'utiliser le protocole **Firmata** dans un programme Python. Il faut donc installer la bibliothèque **pyFirmata**.

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

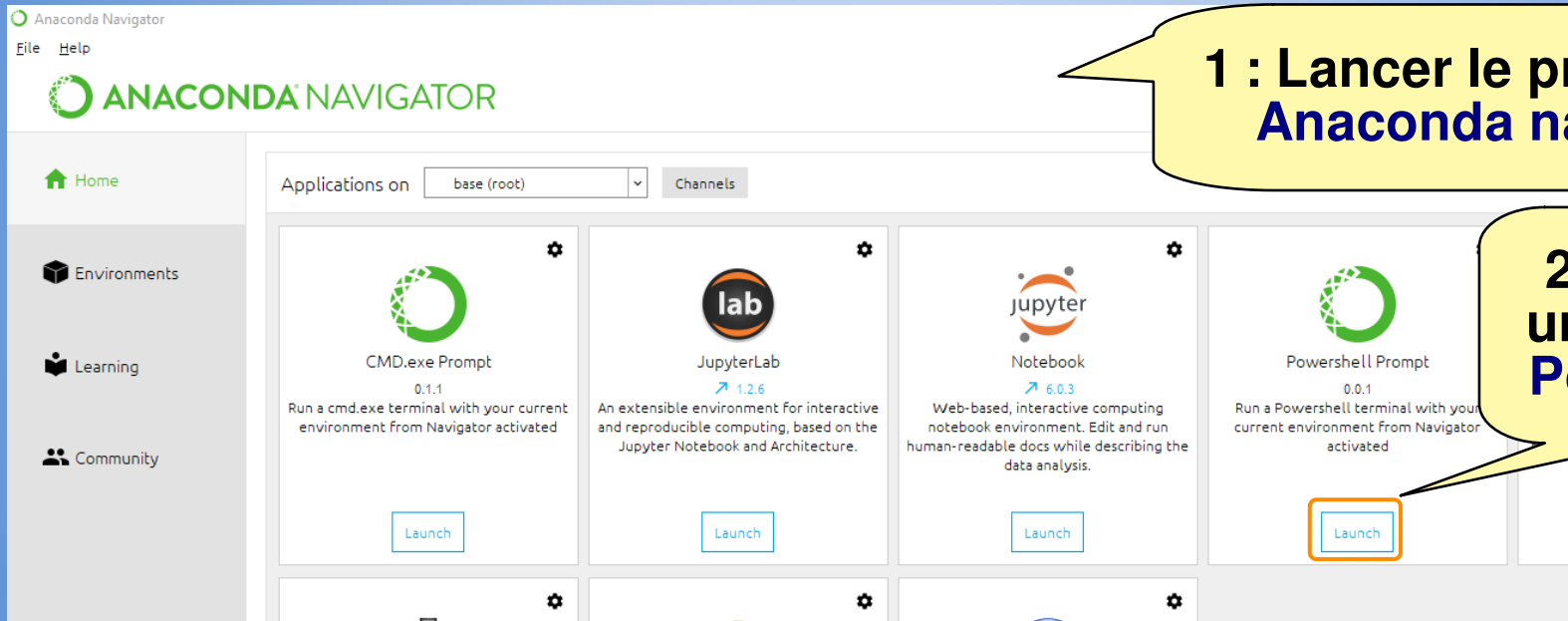
```
Terminal - phroy@debian: ~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
phroy@debian:~$ pip install pyfirmata
Collecting pyfirmata
  Using cached pyFirmata-1.1.0-py2.py3-none-any.whl (14 kB)
Requirement already satisfied: pyserial in ~/.local/lib/python3.9/site-packages (from pyfirmata) (3.5)
Installing collected packages: pyfirmata
Successfully installed pyfirmata-1.1.0
phroy@debian:~$
```

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

# 3b. 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**.



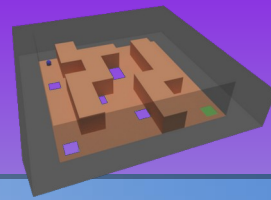
1 : Lancer le programme Anaconda navigator

2 : Lancer un terminal Powershell.

3 : Installer 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 pyfirmata
Collecting pyfirmata
  Downloading pyFirmata-1.1.0-py2.py3-none-any.whl (14 kB)
Collecting pyserial
  Downloading pyserial-3.5-py2.py3-none-any.whl (90 kB)
  |████████████████████████████████████████| 90 kB 1.9 MB/s
Installing collected packages: pyserial, pyfirmata
Successfully installed pyfirmata-1.1.0 pyserial-3.5
(base) PS C:\Users\philippe.roy>
```

# 3. Déplacer le plateau avec la centrale inertielle



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>

```
1 import bge # Bibliothèque Blender / Game Engine (BGE)
2
3 #####
4 # labyrinth.py
5 # @title: Commandes pour le tutotiel Labyrinthe
6 # @project: Blender-EduTech
7 # @lang: fr
8 # @authors: Philippe Roy <philippe.roy@ac-grenoble.fr>
9 # @copyright: Copyright (C) 2021 Philippe Roy
10 # @license: GNU GPL
11 #
12 # Commandes déclenchées par UPBGE pour le tutoriel Labyrinthe
13 #
14 #####
15
16 # Récupérer la scène 3D
17 scene = bge.logic.getCurrentScene()
18 # print("Objets de la scene : ", scene.objects)
19
20 # Constantes
21
22 JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
23 JUST_RELEASED = bge.logic.KX_INPUT_JUST_RELEASED
24 ACTIVATE = bge.logic.KX_INPUT_ACTIVE
25 JUST_DEACTIVATED = bge.logic.KX_SENSOR_JUST_DEACTIVATED
26
27 #####
28 # Gestion du clavier
29 #####
30
31 # Flèches pour tourner le plateau
32 def clavier(cont):
33     # obj = cont.owner
34     obj = scene.objects["Plateau"]
35     keyboard = bge.logic.keyboard
36     resolution = 0.01
37
38     # Up
39     if (ACTIVATE == keyboard.events[bge.events.UPARROWKEY]):
40         obj.applyRotation((-resolution,0,0), False)
41
42     # Down
43     if (ACTIVATE == keyboard.events[bge.events.DOWNARROWKEY]):
44         obj.applyRotation((resolution,0,0), False)
```

```
5 # labyrinth.py
6 # @title: Commandes pour le tutotiel Labyrinthe
7 # @project: Blender-EduTech
8 # @lang: fr
9 # @authors: Philippe Roy <philippe.roy@ac-grenoble.fr>
10 # @copyright: Copyright (C) 2021 Philippe Roy
11 # @license: GNU GPL
12 #
13 # Commandes déclenchées par UPBGE pour le tutoriel Labyrinthe
14 #
15 #####
16 # Récupérer la scène 3D
17 scene = bge.logic.getCurrentScene()
18 # print("Objets de la scene : ", scene.objects)
19
20 # Constantes
21
22 JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
23 JUST_RELEASED = bge.logic.KX_INPUT_JUST_RELEASED
24 ACTIVATE = bge.logic.KX_INPUT_ACTIVE
25 JUST_DEACTIVATED = bge.logic.KX_SENSOR_JUST_DEACTIVATED
26
27 #####
28 # Gestion du clavier
29 #####
30
31 # Flèches pour tourner le plateau
32 def clavier(cont):
33     # obj = cont.owner
34     obj = scene.objects["Plateau"]
35     keyboard = bge.logic.keyboard
36     resolution = 0.01
37
38     # Up
39     if (ACTIVATE == keyboard.events[bge.events.UPARROWKEY]):
40         obj.applyRotation((-resolution,0,0), False)
41
42     # Down
43     if (ACTIVATE == keyboard.events[bge.events.DOWNARROWKEY]):
44         obj.applyRotation((resolution,0,0), False)
```



# 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

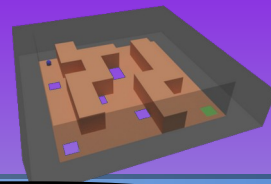
#####
# labyrinthe_carte.py
#####

# 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()`.

```
#####
# 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()
```

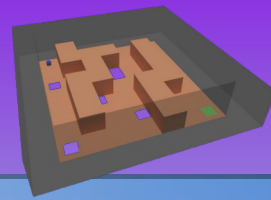
7 : Supprimer la **création manuelle** de l'objet carte

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>

```
1 import bge # Bibliothèque Blender / Game Engine (BGE)
2
3 #####
4 # labyrinth.py
5 # @title: Commandes pour le tutotiel Labyrinthe
6 # @project: Blender-EduTech
7 # @lang: fr
8 # @authors: Philippe Roy <philippe.roy@ac-grenoble.fr>
9 # @copyright: Copyright (C) 2021 Philippe Roy
10 # @license: GNU GPL
11 #
12 # Commandes déclenchées par UPBGE pour le tutoriel Labyrinthe
13 #
14 #####
15
16 # Récupérer la scène 3D
17 scene = bge.logic.getCurrentScene()
18 # print("Objets de la scene : ", scene.objects)
19
20 # Constantes
21
22 JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
23 JUST_RELEASED = bge.logic.KX_INPUT_JUST_RELEASED
24 ACTIVATE = bge.logic.KX_INPUT_ACTIVE
25 JUST_DEACTIVATED = bge.logic.KX_SENSOR_JUST_DEACTIVATED
26
27 #####
28 # Gestion du clavier
29 #####
30
31 # Flèches pour tourner le plateau
32 def clavier(cont):
33     # obj = cont.owner
34     obj = scene.objects["Plateau"]
35     keyboard = bge.logic.keyboard
36     resolution = 0.01
37
38     # Up
39     if (ACTIVATE == keyboard.events[bge.events.UPARROWKEY]):
40         obj.applyRotation((-resolution,0,0), False)
41
42     # Down
43     if (ACTIVATE == keyboard.events[bge.events.DOWNARROWKEY]):
44         obj.applyRotation((resolution,0,0), False)
```

```
5 # labyrinth.py
6 # @title: Commandes pour le tutotiel Labyrinthe
7 # @project: Blender-EduTech
8 # @lang: fr
9 # @authors: Philippe Roy <philippe.roy@ac-grenoble.fr>
10 # @copyright: Copyright (C) 2021 Philippe Roy
11 # @license: GNU GPL
12 #
13 # Commandes déclenchées par UPBGE pour le tutoriel Labyrinthe
14 #
15 #####
16 # Récupérer la scène 3D
17 scene = bge.logic.getCurrentScene()
18 # print("Objets de la scene : ", scene.objects)
19
20 # Constantes
21
22 JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
23 JUST_RELEASED = bge.logic.KX_INPUT_JUST_RELEASED
24 ACTIVATE = bge.logic.KX_INPUT_ACTIVE
25 JUST_DEACTIVATED = bge.logic.KX_SENSOR_JUST_DEACTIVATED
26
27 #####
28 # Gestion du clavier
29 #####
30
31 # Flèches pour tourner le plateau
32 def clavier(cont):
33     # obj = cont.owner
34     obj = scene.objects["Plateau"]
35     keyboard = bge.logic.keyboard
36     resolution = 0.01
37
38     # Up
39     if (ACTIVATE == keyboard.events[bge.events.UPARROWKEY]):
40         obj.applyRotation((-resolution,0,0), False)
41
42     # Down
43     if (ACTIVATE == keyboard.events[bge.events.DOWNARROWKEY]):
44         obj.applyRotation((resolution,0,0), False)
```

## 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)