Labyrinthe à bille
Créer une scène 3D interactive

Tutoriel 2
Passage au Python

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

Objectif

L'objectif de ce tutoriel est de programmer le comportement des objets par des règles codées en Python. Nous allons donc reprendre le fichier blender du tutoriel précédent et remplacer les règles définis par les **briques logiques** avec **un module Python**. La guidance de ce tutoriel a pour pré-requis la réalisation du tutoriel précédent (Tutoriel 1 - Ma première scène).

Le codage en Python permet :

- d'établir plus efficacement des règles plus complexes,
- d'accéder à des instructions plus précises,
- de séparer le fond (comportement) et la forme (objets 3D),
- de maintenir le code avec plus d'efficacité (gestion des versions).

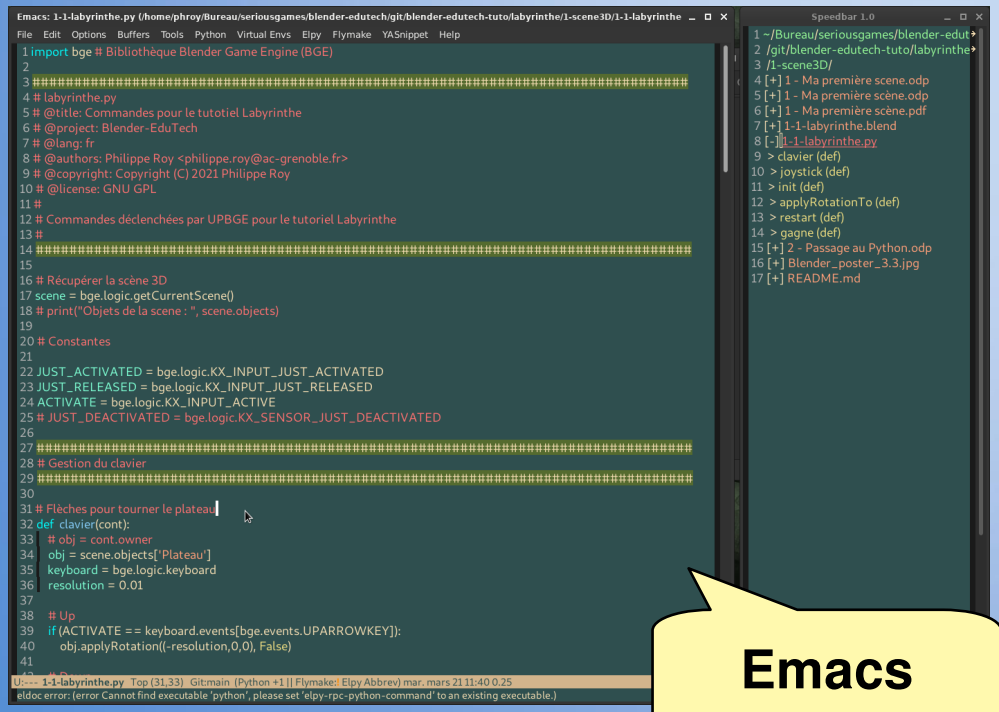
Le tutoriel se décompose en 5 étapes :

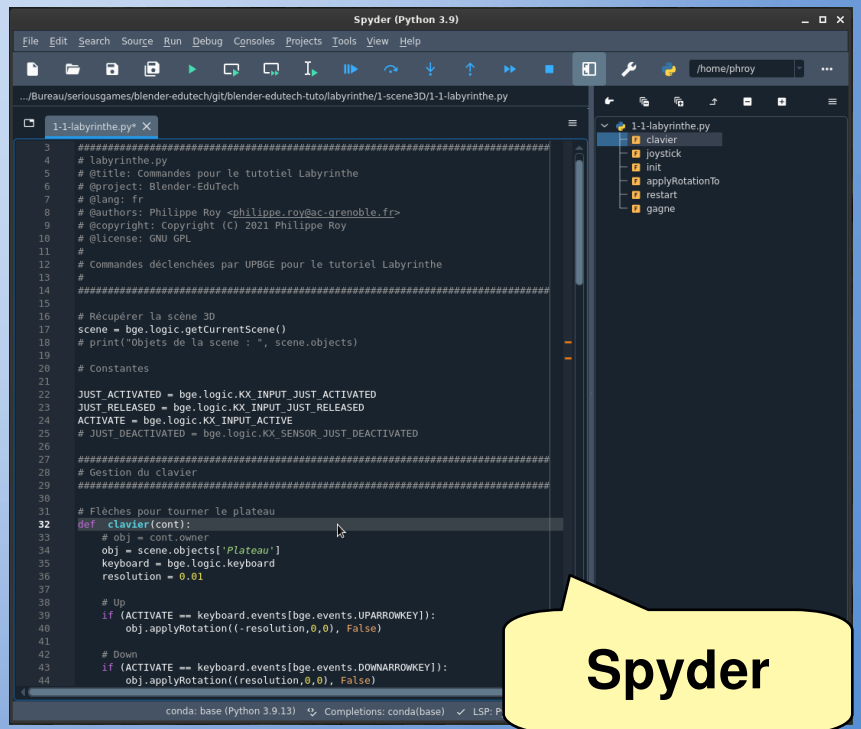
- 1. Installer l'**éditeur de texte**
- 2. **Déplacer** le plateau
- 3. **Définir le game play** (règles d'échec et de réussite)
- 4. **Animer** la fenêtre de fin
- 5. Fermer la fenêtre de fin par un **bouton cliquable**

1. Installation de l'éditeur de texte

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

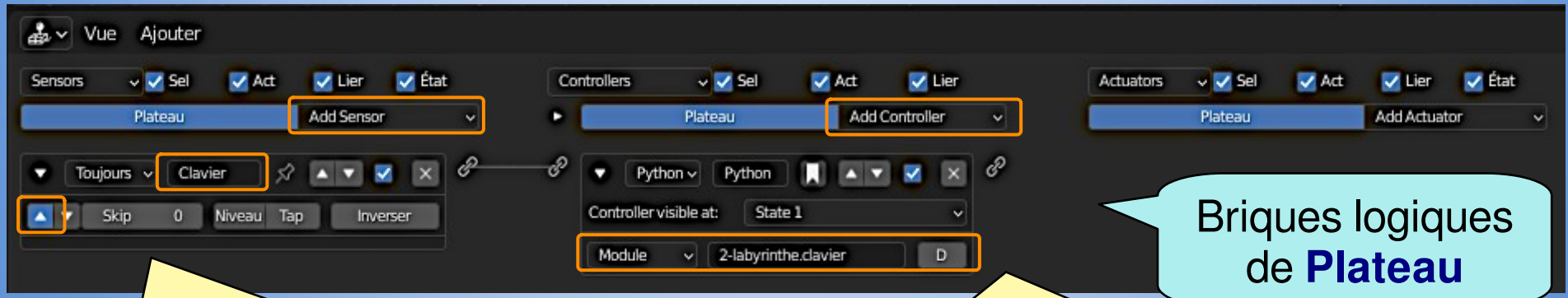




2. Déplacer le plateau

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]

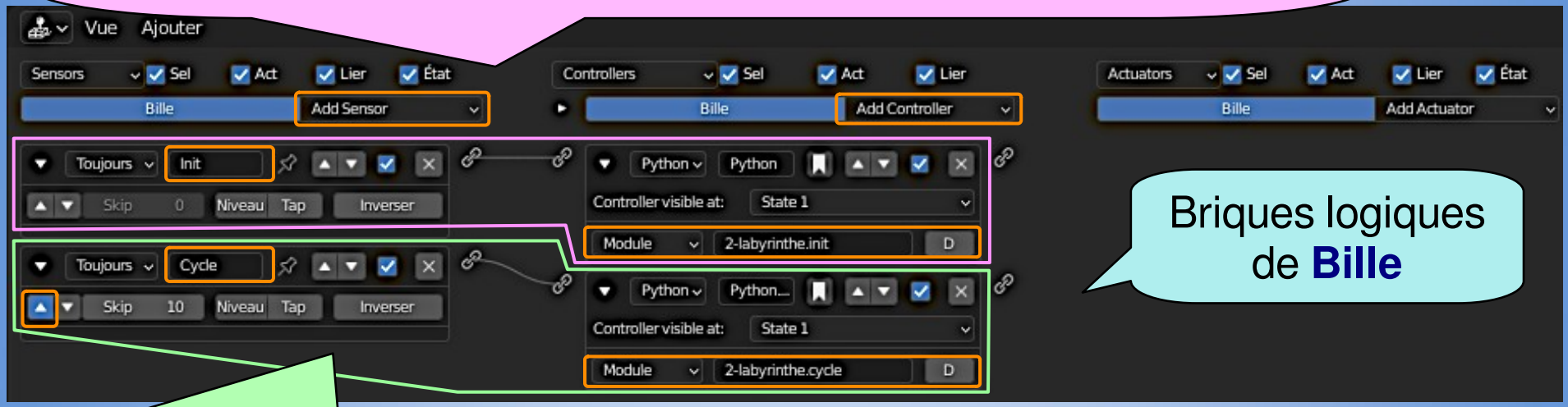
Les pages de l'
API Python de UPBGE
les plus utilisées sont

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

3. Définir le game play (règles d'échec et de réussite)

1 : Initialisation par un appel unique lors du lancement

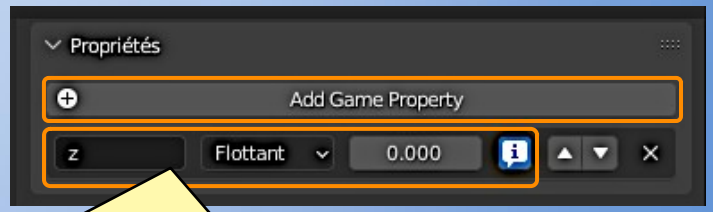
- **Ajouter** un **Capteur Toujours**, **renommer-le** avec **Init**
- **ajouter** le **Module Python** avec la fonction **2-labyrinthe.init**



Briques logiques de **Bille**

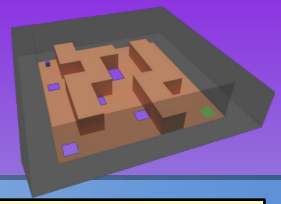
2 : Créer la boucle infinie pour le contrôle continu de la bille

- Ajouter un **Capteur Toujours**, **renommer-le** avec **Cycle**
- **activer** le **Pulse True Level (▲)**
- **ajouter** le **Module Python** avec la fonction **2-labyrinthe.cycle**



3 : Créer la **propriété** '**z**' de **Bille** de **type Flottant** et **visible** comme **propriété de débuggage**

3. Définir le game play (règles d'échec et de réussite)



```
##############################################################################
# Gameplay
##############################################################################

# Initialisation de la scène
def init(cont):
    obj = cont.owner # obj est l'objet associé au contrôleur donc 'Bille'

    # Mémorisation de la position de départ de la bille
    obj['init_x']= obj.worldPosition.x
    obj['init_y']= obj.worldPosition.y
    obj['init_z']= obj.worldPosition.z

    # Cacher le panneau de la victoire et suspendre la physique du panneau cliquable
    scene.objects['Panneau victoire'].setVisible(False,True)
    scene.objects['Panneau victoire - plan'].suspendPhysics(True)

# 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 # propriété 'z' = altitude de la bille
    obj_plateau = scene.objects['Plateau'] # obj_plateau est l'objet 'Plateau'

    # Si l'altitude de bille < -20 et pas de victoire -> chute
    if obj['z'] < -20 and scene.objects['Panneau victoire'].visible == False:
        print ("Chuuuu.....te") # Message pour la sortie standard

        # Replacement du plateau (tous les angles à 0 en plusieurs fois)
        while obj_plateau.worldOrientation.to_euler().x != 0 and
              obj_plateau.worldOrientation.to_euler().y != 0 and
              obj_plateau.worldOrientation.to_euler().z != 0 :
            obj_plateau.applyRotation((-obj_plateau.worldOrientation.to_euler().x,
                                       -obj_plateau.worldOrientation.to_euler().y,
                                       -obj_plateau.worldOrientation.to_euler().z),
                                      False)

        # Mettre la bille à la position de départ avec une vitesse nulle
        obj.worldLinearVelocity = (0, 0, 0)
        obj.worldAngularVelocity = (0, 0, 0)
        obj.worldPosition.x = obj['init_x']
        obj.worldPosition.y = obj['init_y']
        obj.worldPosition.z = obj['init_z'] + 0,5 # On repose la bille
```

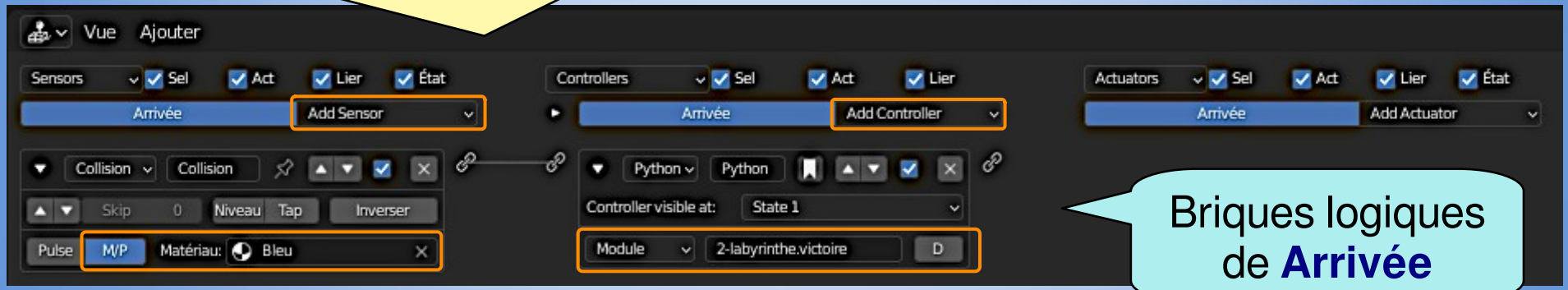
4 : Créer la fonction init

5 : Créer la fonction cycle

3. Définir le game play (règles d'échec et de réussite)

6 : Détecter la collision entre la bille et le plan d'arrivée

- **Ajouter** un **Capteur Collision**,
- **définir** le filtre de détection sur matériaux **Bleu** (la bille donc)
- **ajouter** le **Module Python** avec la fonction **2-labyrinthe.victoire**



Briques logiques de **Arrivée**

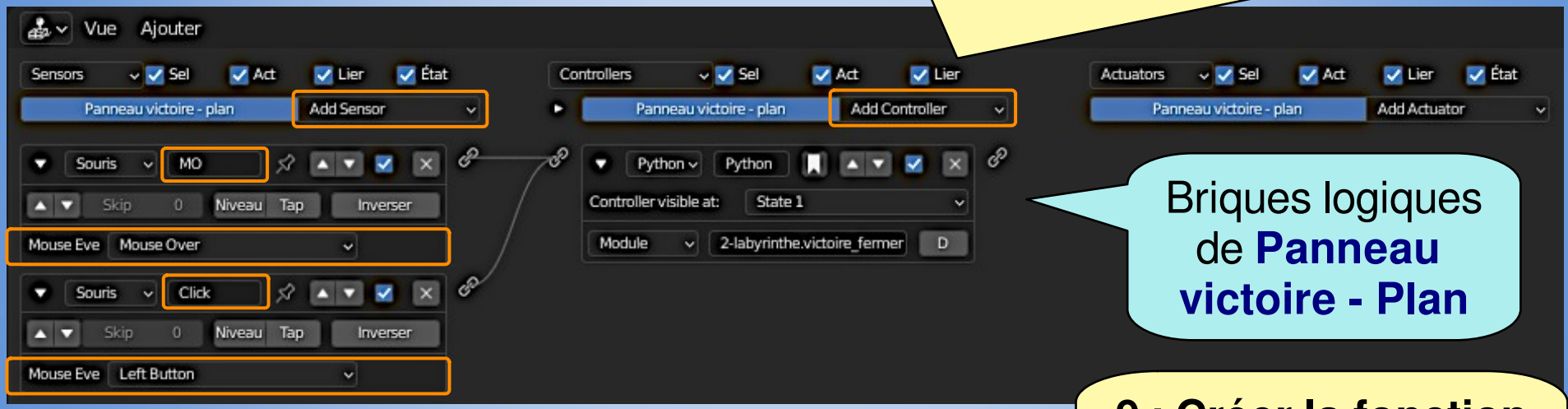
7 : Créer la fonction victoire (partie gameplay)

```
##############################################################################
# Gameplay
##############################################################################

...

# Victoire (collision de la bille avec l'arrivée)
def victoire(cont):
    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
```

3. Définir le game play (règles d'échec et de réussite)

8 : Détecter le clic sur le panneau victoire

- **Ajouter** un **Capteur Souris** avec **Mouse Over**, renommer-le avec **MO**
- **ajouter** un **Capteur Souris** avec **Left Button**, renommer-le avec **Click**
- **ajouter** le **Module Python** avec la fonction **2-labyrinthe.victoire_fermer**



Briques logiques de **Panneau victoire - Plan**

9 : Créer la fonction victoire_fermer (partie gameplay)

```
##############################################################################
# Gameplay
##############################################################################

...

# Fermer le panneau de la victoire (Mouse Over positif et clic)
def victoire_fermer(cont):
    if cont.sensors['Click'].status == JUST_ACTIVATED and cont.sensors['MO'].positive :
        scene.objects['Panneau victoire'].setVisible(False,True) # Cacher le panneau de la victoire
        scene.objects['Panneau victoire - plan'].suspendPhysics(True) # Suspendre la physique du panneau
        scene.objects['Bille']['z']= -21 # On provoque le redémarrage si la bille est ressortie
```

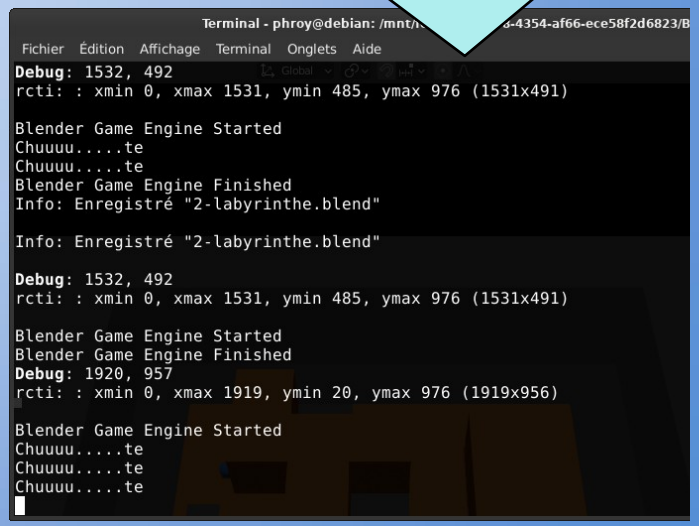
3. Définir le game play (règles d'échec et de réussite)

Avant de passer à la suite, nous pouvons tester notre programmation Python.

10 : Rendre le panneau victoire visible en se plaçant la frame 100

La valeur des **propriétés de débuggage**

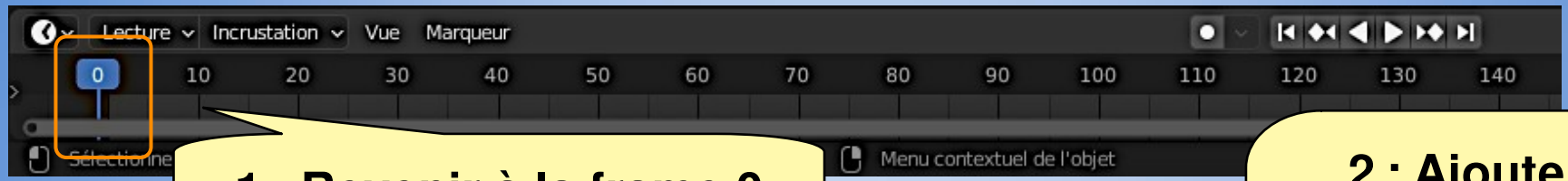
La sortie standard (il faut lancer Blender dans un terminal)



11 : Tester le jeu [P]

4. Animer la fenêtre de fin

On peut bien évidement déclencher les animations à partir de module Python.



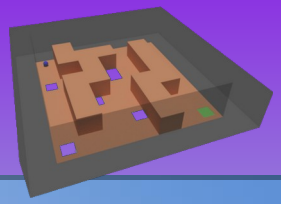
1 : Revenir à la frame 0

2 : Ajouter l'animation dans la fonction victoire (partie gameplay)

```
##############################################################################
# Gameplay
##############################################################################

...

# Victoire (collision de la bille avec l'arrivée)
def victoire(cont):
    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)
```

3 : Tester l'animation [P]

5. Fermer la fenêtre de fin par un bouton cliquable



Afin de fermer la panneau victoire avec un bouton en forme de croix (archétype visuel de la fermeture), je propose de partir du dessin 2D vectoriel.