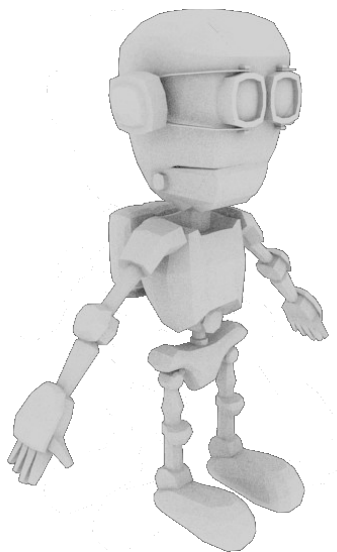




Introduction à la programmation avec Ropy

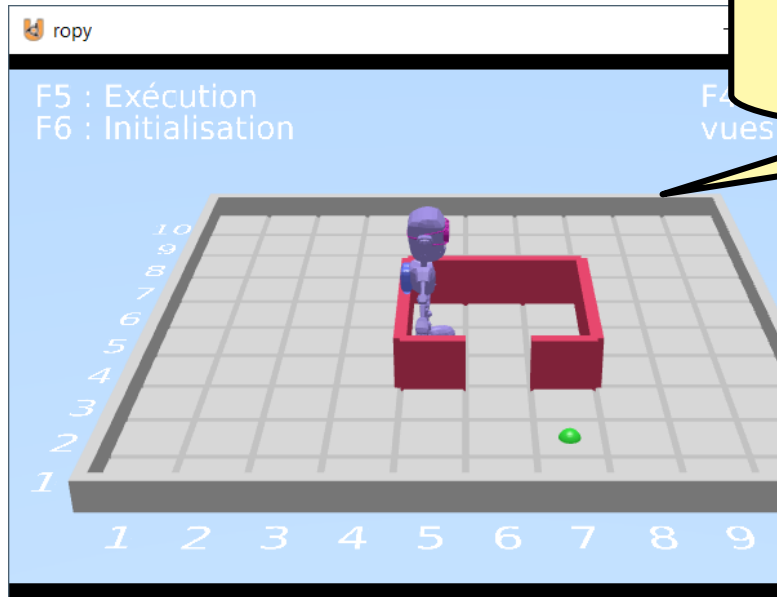


Présentation de Ropy et de son environnement de programmation



Ropy est un robot qui se commande grâce au langage **Python**. L'interface de programmation se décompose en 2 fenêtres : un éditeur de texte et le simulateur.

Le **simulateur** permet de **visualiser l'évolution du robot**.

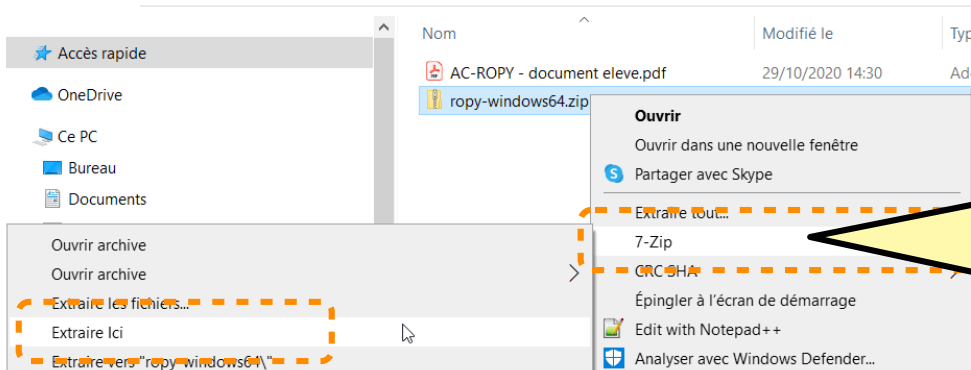


Un **éditeur de texte** (Notepad++, Spyder, Jupyter, Emacs, ...) pour **écrire le programme** en **Python**.

```
18 scene = bge.logic.getCurrentScene() # Récupérer la scène 3D
19 cont = bge.logic.getCurrentController() # Récupérer le contrôleur BGE
20 obj = cont.owner # Récupérer le robot de la scène 3D
21 obj.visible=False
22
23 #####
24 # Initialisation du niveau :
25 # Niveau 0 : Vide
26 # Niveau 1 : Les premiers pas de Ropy
27 # Niveau 2 : Sécuriser Ropy
28 # Niveau 3 : Partir au bout du monde
29 # Niveau 4 : Faire face à l'inconnu
30 # Niveau 5 : Se rendre utile
31 #####
32
33 rp_niveau (0) # Saisir le niveau (de 0 à 5)
34 ropy_init.main() # Initialisation de la scène 3D
35
36 #####
37 # Fonctions
38 #####
39
40 #####
41 # Commandes
42 #####
```

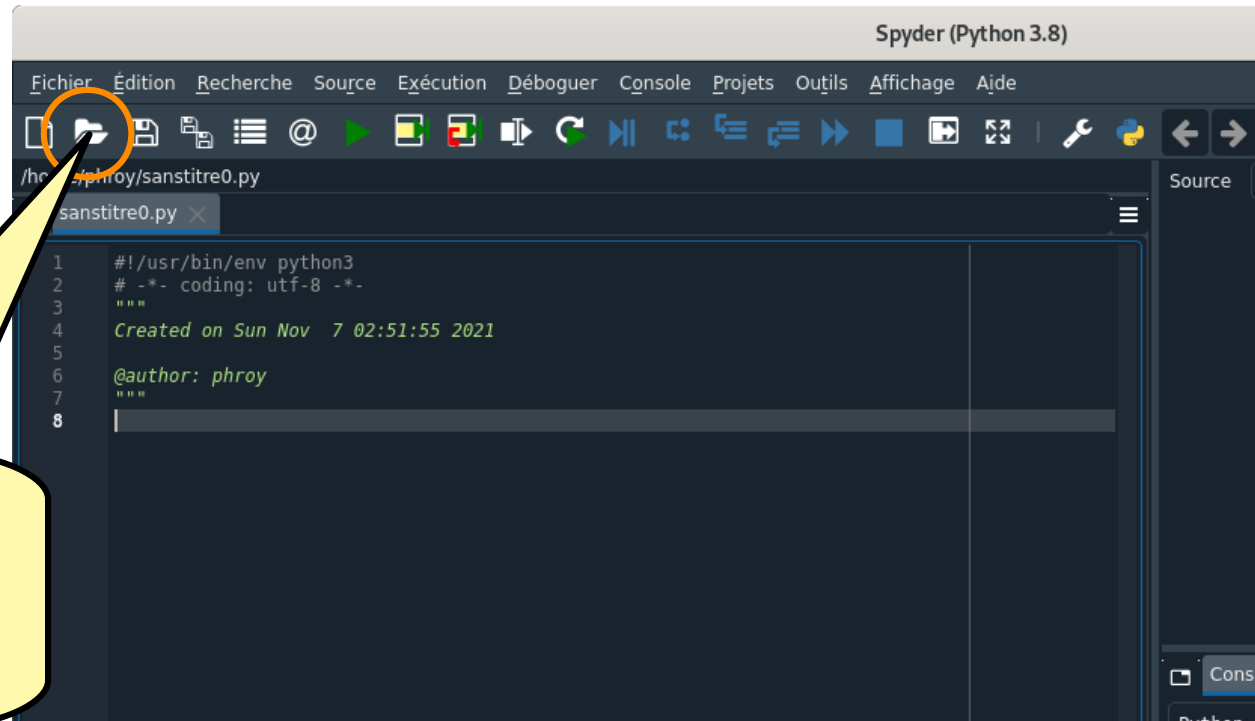
Éditer le programme avec Spyder

Ouvrir le fichier ropy_cmd.py



1 : Récupérer l'archive **ropy-windows64.zip** et la décompresser avec **7-Zip** dans votre répertoire.

2 : Lancer le Logiciel **Spyder**.



3: Ouvrir le fichier Python à éditer **ropy_cmd.py** (Ropy commandes).

Éditer le programme avec Spyder

Exécution du programme



5 : **Sauvegarder** le fichier

Attention !

Toujours sauvegarder le fichier avant son exécution avec le simulateur.

Le simulateur est le programme **ropy.exe**

6 : **Exécuter** le programme

F6 :

Revenir à l'état initial

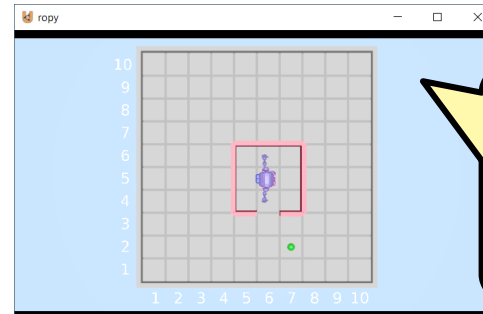
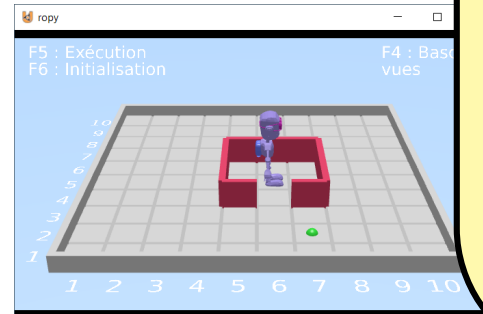
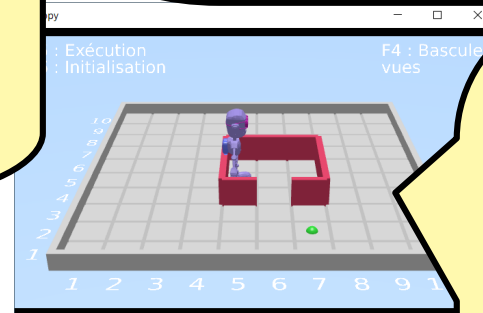
F5 :

Exécution de votre programme.

4 : **Écrire** le code Python

F4 (bascule) :
Permet de changer de vue

```
Fichier  Édition  P...
.../1000/gvts/sitp:host=192.1
ropy_cmd.py x
19  cont = bge.
20  obj = cont.
21  obj.visible=Fa
22
23  #####
24  # Initialisation du niveau :
25  # Niveau 0 : Vide
26  # Niveau 1 : Les premiers pas de Ropy
27  # Niveau 2 : Sécuriser Ropy
28  # Niveau 3 : Partir au bout du monde
29  # Niveau 4 : Faire face à l'inconnu
30  # Niveau 5 : Se rendre utile
31  #####
32
33  rp_niveau(0) # Saisir le niveau (de 0 à 5)
34  ropy_init.main() # Initialisation de la scène 3D
35
36  #####
37  # Fonctions
38  #####
39
40  #####
41  # Commandes
42  #####
43
44
45  rp_avancer(0)
46  rp_avancer()
47  rp_avancer()
48  rp_avancer()
49
50
51
```



Contenu du fichier rOPY_cmd.py



Le fichier **ropy_cmd.py** comporte 5 sections.

```
import bge # Bibliothèque Blender Game Engine (BGE)
import math # Bibliothèque Math
from ropy_lib import * # Bibliothèque Ropy
import ropy_init # Initialisation du robot Ropy

#####
# rOPY_cmd :
# @title: Commandes du Robot Ropy
# @project: Ropy (RobotProg pour Python)
#####

def main():

    #####
    # Récupérer les objets du moteur 3D (BGE) << NE PAS MODIFIER CETTE SECTION >>
    #####

    scene = bge.logic.getCurrentScene() # Récupérer la scène 3D
    cont = bge.logic.getCurrentController() # Récupérer le contrôleur BGE
    obj = cont.owner # Récupérer le robot de la scène 3D
    obj.visible=False

    #####
    # Initialisation du niveau :
    # Niveau 0 : Vide
    # Niveau 1 : Les premiers pas de Ropy
    # Niveau 2 : Sécuriser Ropy
    # Niveau 3 : Partir au bout du monde
    # Niveau 4 : Faire face à l'inconnu
    # Niveau 5 : Se rendre utile
    #####

    rp_niveau (0) # Saisir le niveau (de 0 à 5)
    ropy_init.main() # Initialisation de la scène 3D

    #####
    # Fonctions
    #####

    #####
    # Commandes
    #####

    rp_avancer()
    rp_avancer()
    rp_avancer()
    rp_avancer()
```

Import des bibliothèques

Récupération des objets du
moteur 3D BGE
(Blender Game Engine)
Ne pas modifier cette section

Initialisation du niveau

Vous devez entrer le niveau
avec la commande
rp_niveau(numero_niveau)

Fonctions : section pour le
codage de vos fonctions

Commandes : section pour le
codage des commandes du robot

Niveau 1 – Les premiers pas de Ropy

Création d'une fonction



Objectif 1.2 : Pour faciliter le codage, on va créer la fonction `mrp_avancer()` regroupant `avancer` et `marquer`.

La **définition d'une fonction** se fait de la manière suivante :

```
def fonction_1(arguments) :  
    instruction_1  
    instruction_2  
    ...  
↔ return valeurs_renvoyées
```

Cet espace est l'**indentation**, il se fait avec la touche tabulation (Tab).

Attention ! C'est l'**indentation** qui définit le **début et la fin d'un bloc**.

L'**appel de la fonction** est des plus simple :
`fonction_1(arguments)`

```
#####  
# Fonctions  
#####
```

```
#####  
# Commandes  
#####
```

Niveau 1 – Les premiers pas de Ropy

Création d'une fonction



Objectif 1.3 : **Ropy** ne sait pas reculer, vous allez donc créer la fonction `mrp_reculer()`. Pour tester cette nouvelle fonction en toute sécurité (sans les murs), vous pouvez vous mettre sur le **niveau 0** avec la fonction `rp_niveau(niveau)`.

```
#####  
# Initialisation du niveau :  
# Mission 1 : Les premiers pas de Ropy  
# Mission 2 : Sécuriser Ropy  
# Mission 3 : Partir au bout du monde  
# Mission 4 : Faire face à l'inconnu  
# Mission 5 : Se rendre utile  
#####  
  
rp_niveau(0) # Saisir la mission (de 0 à 5)  
ropy_init_scene() # Initialisation de la scène 3D
```

Garder votre fonction `mrp_avancer()`.
Ne marquer que la nouvelle fonction.

`rp_niveau(niveau)` se trouve
avant la définition de vos fonctions.

```
#####  
# Fonctions  
#####
```

```
#####  
# Commandes  
#####
```

Niveau 2 – Apprendre le danger

Structure conditionnelle (si, alors, sinon)



Objectif 2.1 : Aller au niveau 2, provoquer une collision avec un mur en avançant et observer ce qu'il se passe. Il semble assez clair qu'il faut sécuriser l'avance du robot.

Si le test de `condition` est vrai
alors exécuter `instruction_1`
sinon exécuter `instruction_2`

```
#####  
# Commandes  
#####  
  
_____  
  
_____  
  
_____  
  
_____
```

Une **structure conditionnelle** permet d'exécuter des instructions en fonction du résultat d'un test (condition).

```
if condition :  
    instructions_1  
else :  
    instructions_2
```

le **sinon**
n'est pas
obligatoire

Les conditions peuvent être

- `a == b` : a est égal à b
- `a != b` : a est différent de b
- `a < b` : a est strictement inférieur à b
- `a <= b` : a est inférieur ou égal à b
- `a == b and c == d` : les deux conditions doivent être vrai (fonction ET)
- `a == b or c == d` : une des deux conditions doit être vrai (fonction OU)

La fonction pour **détecter un mur** est : `rp_detect_mur()`. La fonction retourne **True** si il a un mur et **False** si il n'y a pas de mur.

Niveau 2 – Apprendre le danger

Affichage d'un message vers la console



Objectif 2.2 : Intégrer le test de sécurisation dans votre fonction `mrp_avancer()`. Avancer uniquement en cas d'absence de mur.

Nous pouvons générer des **message dans la console** afin de faciliter la vérification et la correction (débogage) du programme.

```
print("Texte à afficher")
```

Il est possible d'afficher la valeur d'une variable. Par exemple, je souhaite afficher la valeur de la variable `nb_pas` accompagnée d'un texte explicatif.

```
print("Nombre de pas:", nb_pas)
```

```
#####  
# Fonctions  
#####
```

Niveau 3 – Partir au bout du monde

Passage d'argument (dans une fonction)



Objectif 3.2 : Afin de faciliter le code nous allons créer une fonction pour avancer d'un nombre de pas : `mrp_avancer_pas (nb_pas)`.

Lors de la définition de fonction `mrp_avancer()`, nous n'avons pas utilisé les arguments. Un **argument** est une variable qui permet de **paramétrer la fonction**.

Par exemple : une fonction pour faire tourner le robot à partir de valeur angulaire.

```
def mrp_tourner(angle):  
    if angle == 90:  
        rp_droite()  
    if angle == -90:  
        rp_gauche()  
    if angle==180 or angle==-180:  
        rp_droite()  
        rp_droite()
```

angle est
l'argument

```
#####  
# Fonctions  
#####
```

```
#####  
# Commandes  
#####
```

Niveau 4 - Faire face à l'inconnu

Structure itérative - boucle indéfinie (tant que)



Objectif 4 : Aller au niveau 4, **Ropy** doit toujours atteindre la case (10,10), mais son lieu de départ change à chaque fois. Pour pallier à l'aléatoire, il faut créer une fonction qui permet d'atteindre un mur : `mrp_avancer_mur()`.

Une **boucle indéfinie** (nombre de répétitions inconnu à l'avance) se poursuit **tant qu'une condition est vraie**.

```
while condition :  
    bloc_instructions
```

Par exemple : une boucle pour activer le robot par la saisie d'un code de déverrouillage. On reste dans la boucle **tant que** la saisie n'est pas « okropy ».

```
saisie=""  
while saisie!="okropy" :  
    saisie = input()
```

`input()` permet de faire une saisie au clavier dans la console.

```
#####  
# Fonctions  
#####
```

```
#####  
# Commandes  
#####
```