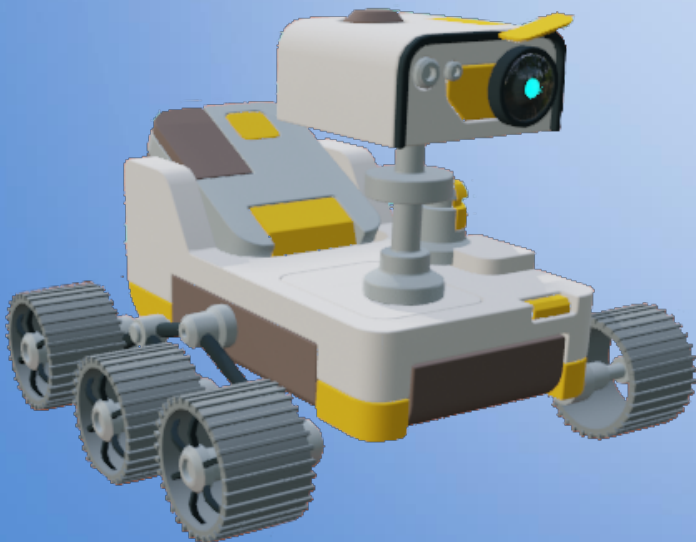


Séquence 3

Algorithme et programmation



Introduction à la programmation Python avec Ropy



Mission 1 – Les premiers pas de Ropy

Instruction et structure linéaire



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

Objectif 1 : Il faut aider **Ropy** à sortir de son emplacement et atteindre la case à l'est de la station. Afin de visualiser le trajet, il faudra marquer les cases.

Vous avez à disposition plusieurs **commandes élémentaires** pour diriger **Ropy** :

- Avancer : **rp_avancer ()**
- Tourner à gauche : **rp_gauche ()**
- Tourner à droite : **rp_droite ()**
- Marquer la case : **rp_marquer ()**

Le « **rp_** » dans le nom des fonctions permet d'identifier les fonctions de **Ropy**.

```
#####  
# Commandes  
#####  
  
rp_marquer ()  
rp_avancer ()  
rp_marquer ()  
rp_droite ()  
rp_avancer ()  
rp_marquer ()  
rp_avancer ()  
rp_marquer ()  
rp_avancer ()  
rp_marquer ()  
rp_gauche ()  
rp_avancer ()  
rp_marquer ()
```

Mission 2 - Ma première fonction

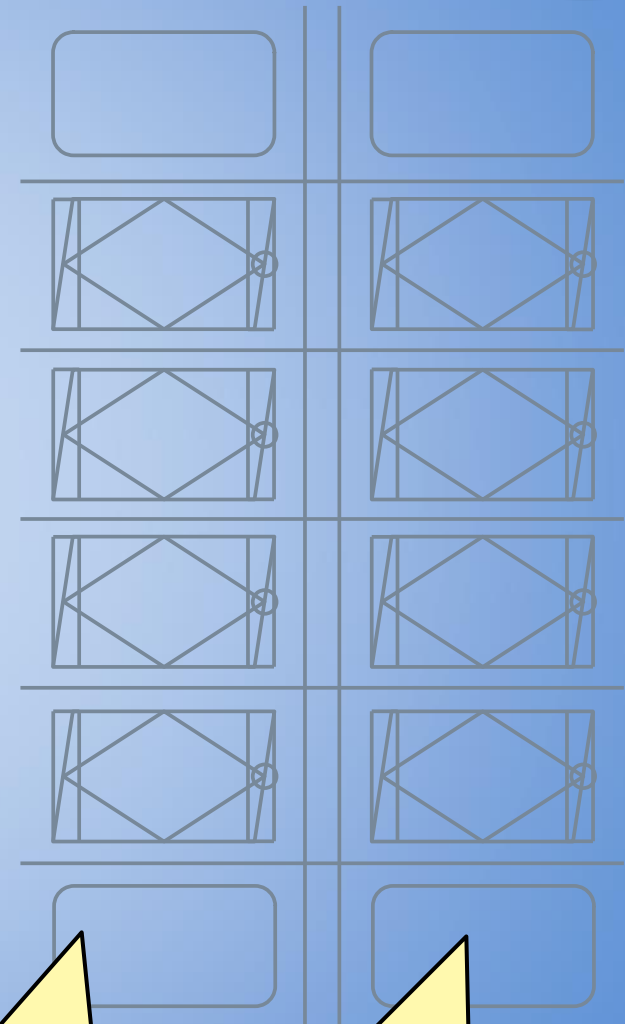
Création d'une fonction



Objectif 2 : Créer la fonction `mrp_avancer()` regroupant `avancer` et `marquer`.

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

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



Représenter que le premier appel

Représenter la fonction

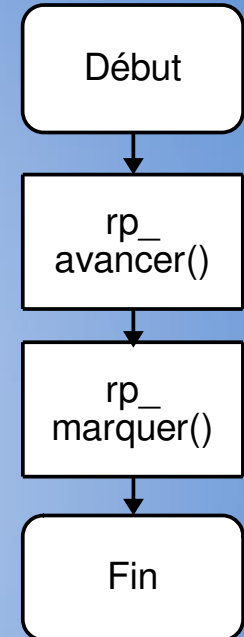
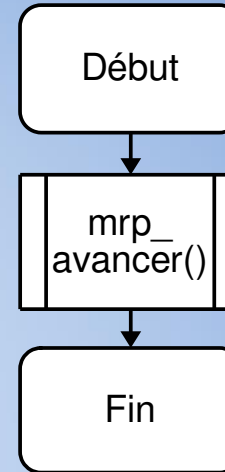
Mission 2 - Ma première fonction

Création d'une fonction



Objectif 2 : Créer la fonction `mrp_avancer()` regroupant `avancer` et `marquer`.

```
#####  
# Fonctions  
#####  
  
def mrp_avancer() :  
    rp_avancer()  
    rp_marquer()  
  
#####  
# Commandes  
#####  
  
mrp_avancer()
```



Représenter que
le premier appel

Représenter
la fonction

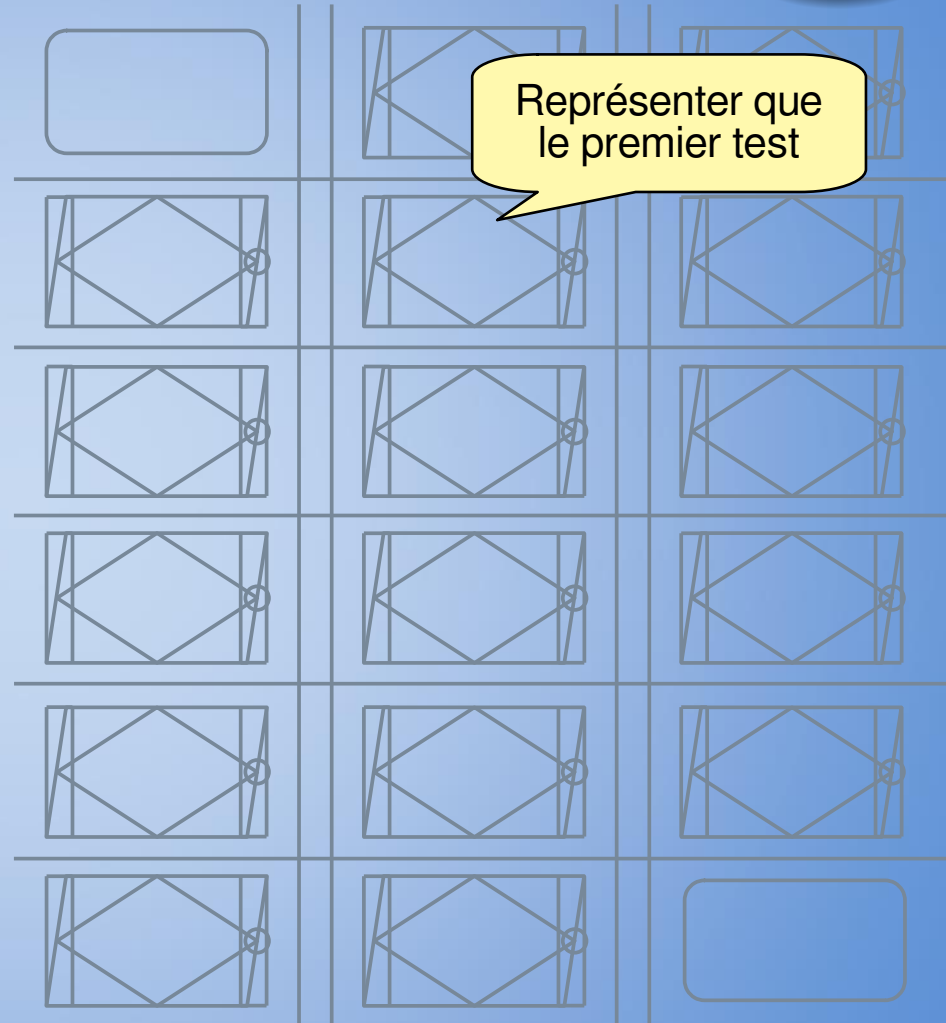
Mission 3 – Apprendre le danger

Structure conditionnelle (si, alors, sinon)



Objectif 3.1 : Aller à la mission 3, provoquer une collision avec un obstacle en avançant et observer ce qui se passe. Il vous faut donc sécuriser l'avance du robot.

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



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

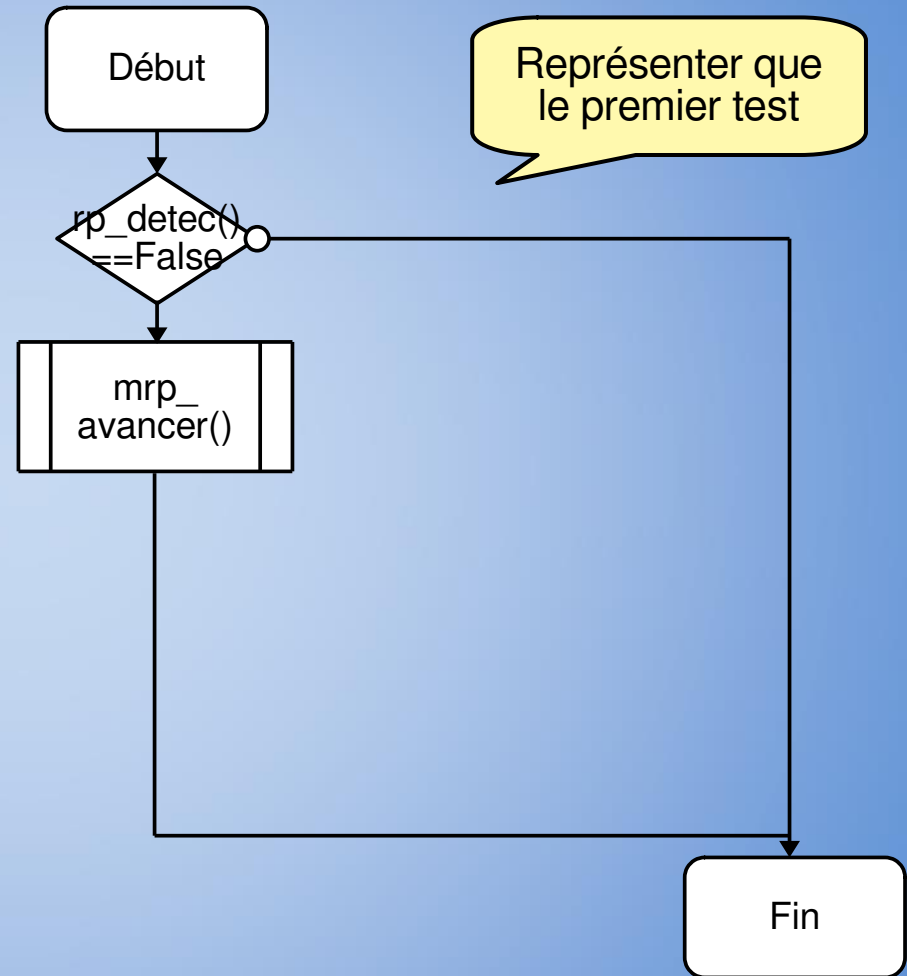
Mission 3 – Apprendre le danger

Structure conditionnelle (si, alors, sinon)



Objectif 3.1 : Aller à la mission 3, provoquer une collision avec un obstacle en avançant et observer ce qui se passe. Il vous faut donc sécuriser l'avance du robot.

```
#####  
# Commandes  
#####  
  
mrp_avancer()  
if rp_detect() == False:  
    mrp_avancer()
```



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

Mission 3 – Apprendre le danger

Structure conditionnelle (si, alors, sinon)



Objectif 3.2 : Intégrer le test de sécurisation dans votre fonction `mrp_avancer()`.

```
#####  
# Fonctions  
#####  
  
def mrp_avancer() :  
    if rp_detect() == False :  
        rp_avancer()  
        rp_marquer()
```

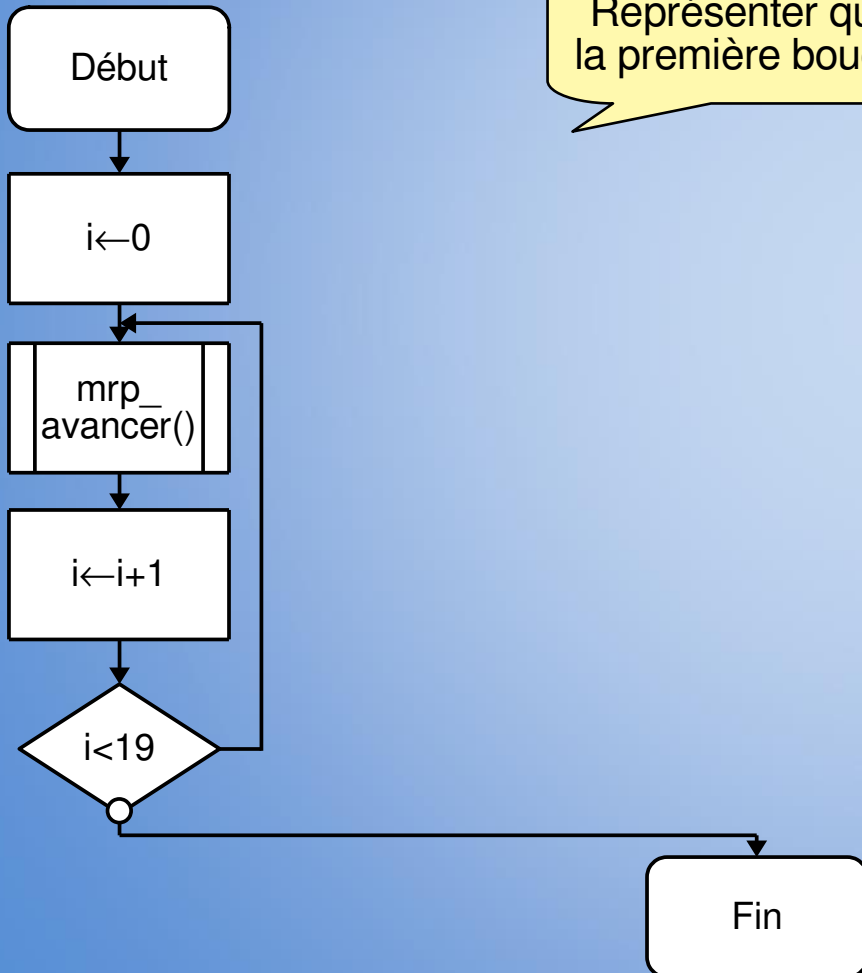
```
#####  
# Commandes  
#####  
  
mrp_avancer()  
mrp_avancer()
```


Mission 4 – Partir au bout du monde

Structure itérative - boucle définie



Objectif 4.1 : Aller à la mission 4, **Ropy** est maintenant prêt pour l'aventure et donc atteindre une case éloignée. Pour un tel voyage, l'utilisation d'une boucle s'impose.



Représenter que la première boucle

```
#####  
# Commandes  
#####  
  
for i in range(19):  
    mrp_avancer()  
    rp_gauche()  
for i in range(5):  
    mrp_avancer()
```


Mission 4 – Partir au bout du monde

Passage d'argument (dans une fonction)



Objectif 4.2 : Afin d'enrichir notre bibliothèque, nous allons créer une fonction pour avancer d'un nombre de pas : `mrp_avancer_nbpas (pas)` .

```
#####  
# Fonctions  
#####  
  
def mrp_avancer_nbpas (pas) :  
    for i in range (pas) :  
        mrp_avancer ()
```

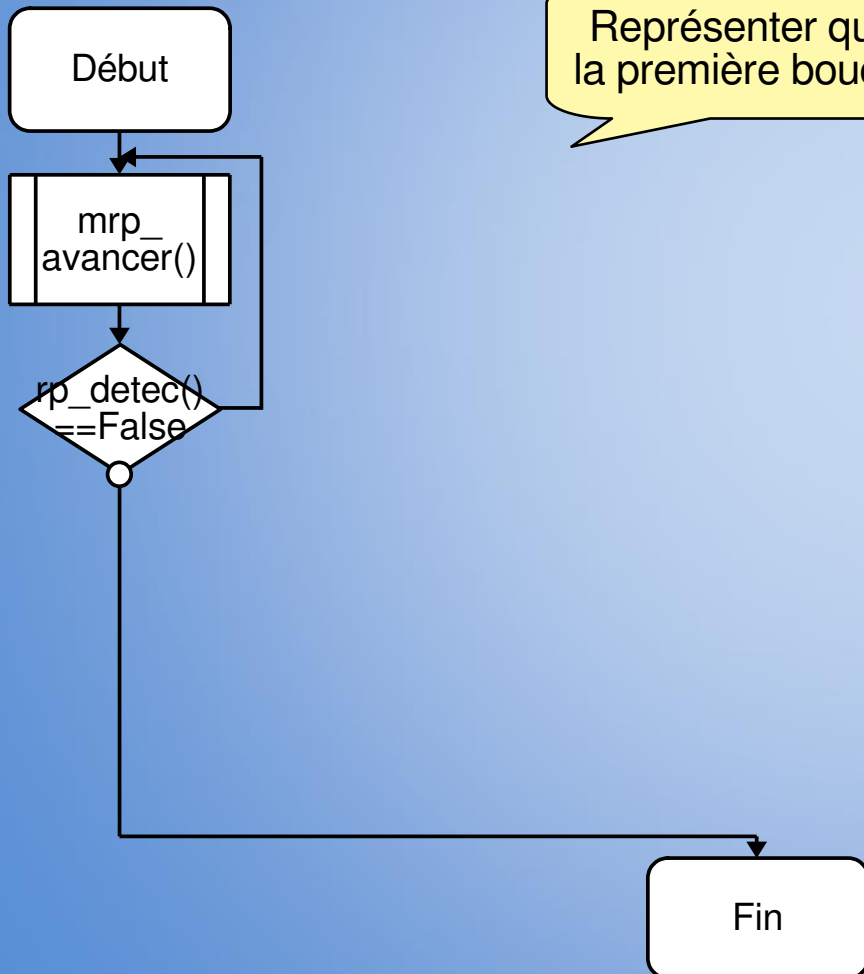
```
#####  
# Commandes  
#####  
  
mrp_avancer_nbpas (19)  
rp_gauche ()  
mrp_avancer_nbpas (5)
```


Mission 5 - Faire face à l'inconnu

Structure itérative - boucle indéfinie (tant que)



Objectif 5.1 : Aller à la mission 5, **Ropy** doit toujours atteindre la même case, mais son lieu de départ change à chaque fois.



Représenter que la première boucle

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

rp_gauche()
while rp_detec() == False:
    mrp_avancer()
rp_droite()
while rp_detec() == False:
    mrp_avancer()
rp_gauche()
while rp_detec() == False:
    mrp_avancer()
```


Mission 5 – Faire face à l’inconnu

Structure itérative - boucle indéfinie (tant que)



Objectif 5.2 : Afin d’enrichir notre bibliothèque, nous allons créer une fonction qui permet d’atteindre un mur : `mrp_avancer_mur()` .

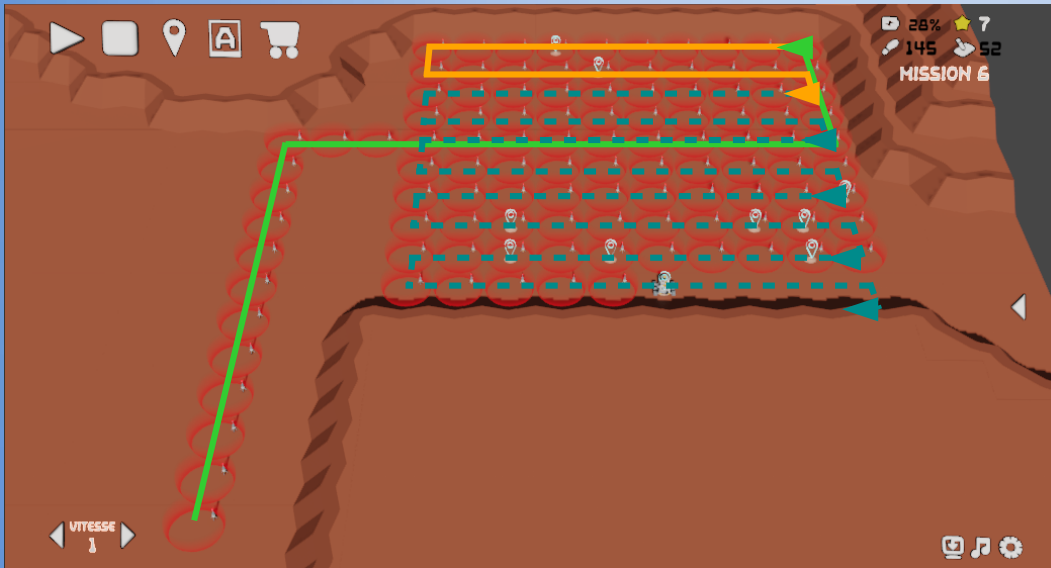
```
#####  
# Fonctions  
#####  
  
def mrp_avancer_mur() :  
    while rp_detect() == False :  
        mrp_avancer()
```

```
#####  
# Commandes  
#####  
  
rp_gauche()  
mrp_avancer_mur()  
rp_droite()  
mrp_avancer_mur()  
rp_gauche()  
mrp_avancer_mur()
```


Mission 6 - Se rendre utile



Objectif 6.1 : Afin d'analyser la roche sur une zone, **Ropy** doit y prélever des carottes. Pour effectuer les forages, **Ropy** doit passer sur toutes les cases.



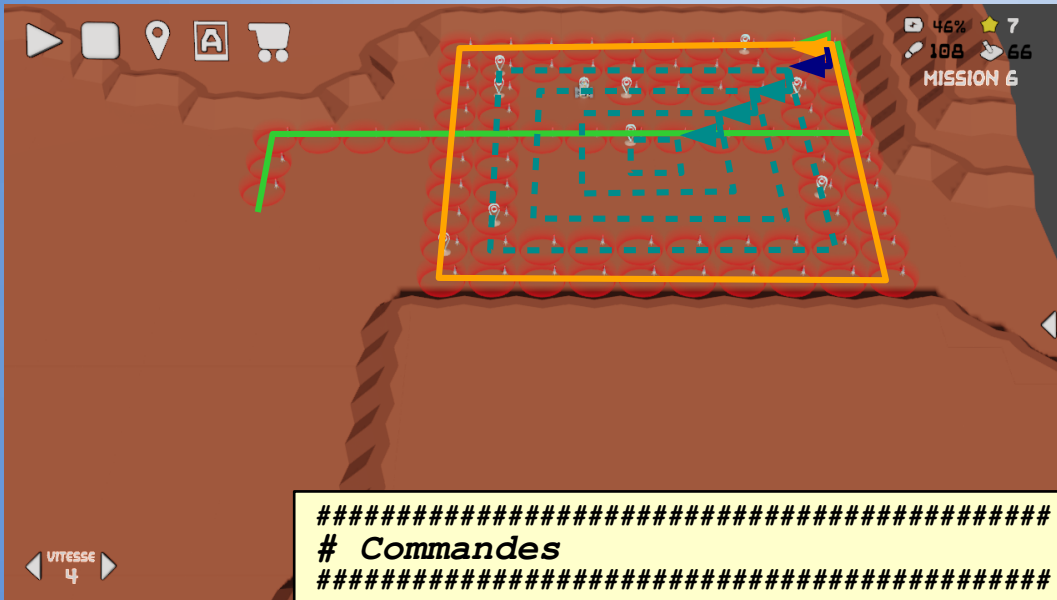
```
#####  
# Commandes  
#####  
  
mrp_depart () ←  
  
for i in range (5): ←  
    mrp_aller_retour () ←
```

```
#####  
# Fonctions  
#####  
  
# Aller à la case origine  
def mrp_depart () :  
    rp_gauche ()  
    mrp_avancer_mur ()  
    rp_droite ()  
    mrp_avancer_mur ()  
    rp_gauche ()  
    mrp_avancer_mur ()  
  
# Faire un aller-retour  
def mrp_aller_retour () :  
    mrp_avancer_nbpas (9)  
    rp_gauche ()  
    mrp_avancer ()  
    rp_gauche ()  
    mrp_avancer_nbpas (9)  
    rp_droite ()  
    mrp_avancer ()  
    rp_droite ()
```


Mission 6 – Se rendre utile ... certes, mais avec classe !



Objectif 6.2 : **Ropy** est devenu esthète. C'est le même objectif, mais il faut parcourir le terrain en **colimaçon**.



```
#####  
# Commandes  
#####  
  
rp_depart () ←  
  
pas = 9  
for i in range (5): ←  
    mrp_carre (nb_pas) ←  
    mrp_avancer_so () ←  
    nb_pas=nb_pas-2
```

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

```
# Faire un carre  
def mrp_carre (pas) :  
    for i in range (4) :  
        mrp_avancer_nbpas (pas)  
        rp_gauche ()
```

```
# Avance d'une case  
# en diagonale sud-ouest  
def mrp_avancer_so () :  
    rp_gauche ()  
    mrp_avancer ()  
    rp_droite ()  
    mrp_avancer ()
```

Référence du langage de programmation de Ropy



Instructions de base (rp_*) :

- Avancer : `rp_avancer()`
- Reculer : `rp_reculer()`
- Tourner à gauche : `rp_gauche()`
- Tourner à droite : `rp_droite()`
- Marquer la case : `rp_marquer()`
- Détection d'un obstacle: `rp_detect()`
 - retourne **True** si il y a un obstacle
 - retourne **False** si il n'y a pas d'obstacle

Instructions de base à créer (mrp_*) :

- Avancer amélioré (marquage et sécurisation) : `mrp_avancer()`
- Avancer d'un nombre de pas : `mrp_avancer_nbpas(nb)`
- Avancer jusqu'à un obstacle : `mrp_avancer_mur()`

Instructions avancées à créer (mrp_*) :

- Aller à l'origine du balayage : `mrp_depart()`
- Faire un allée-retour : `mrp_aller_retour()`
- Faire un carré : `mrp_carre(nb_pas)`