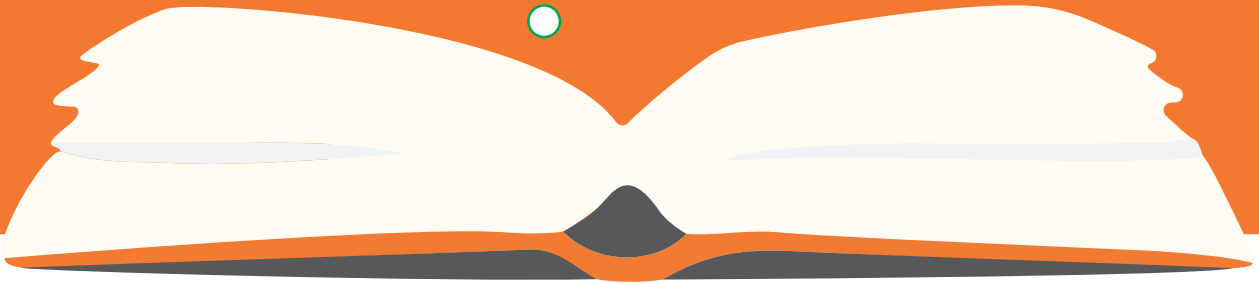
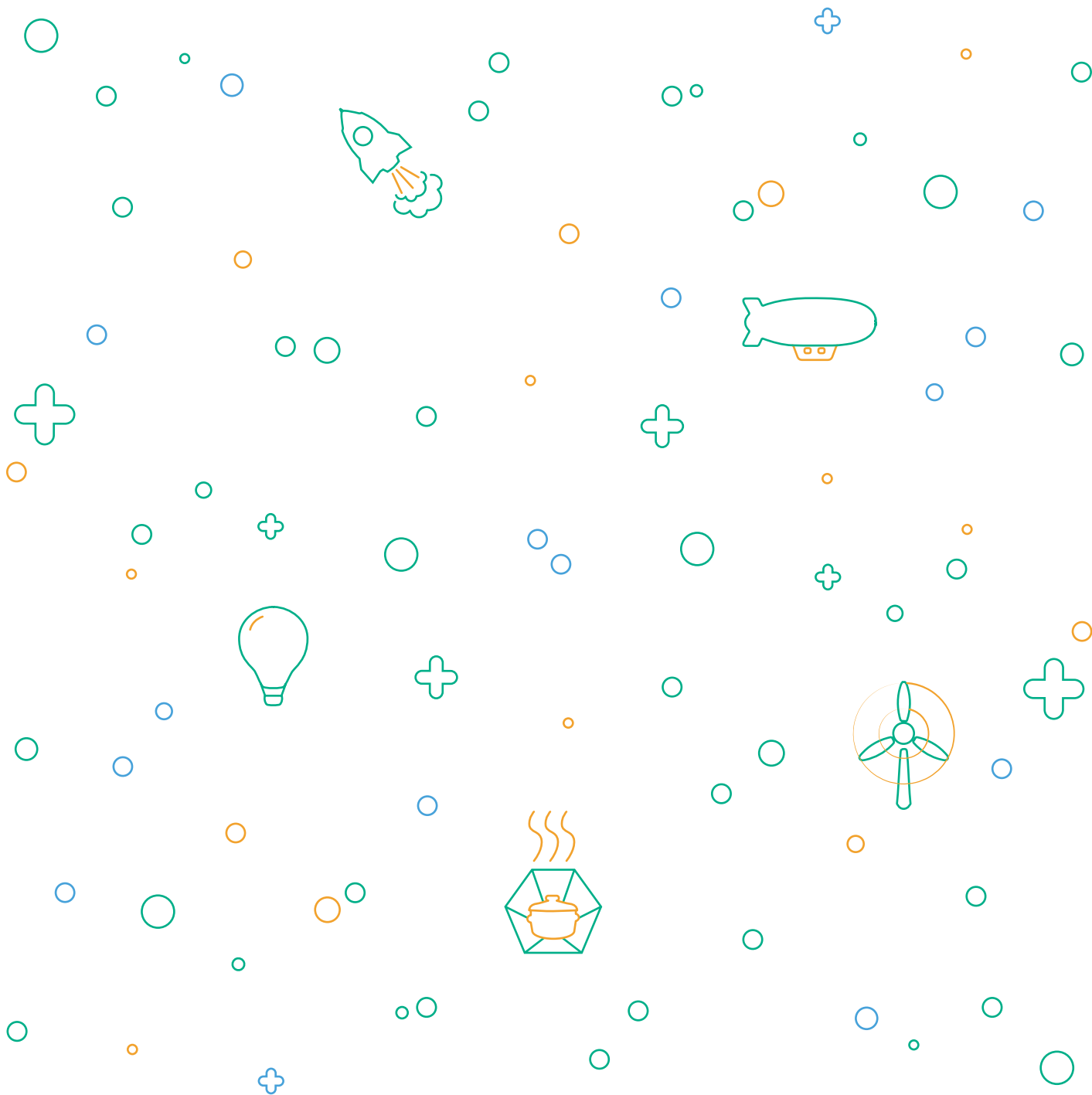




VOYAGER SUR MARS !



GUIDE D'UTILISATION
ROBOT MARTIEN





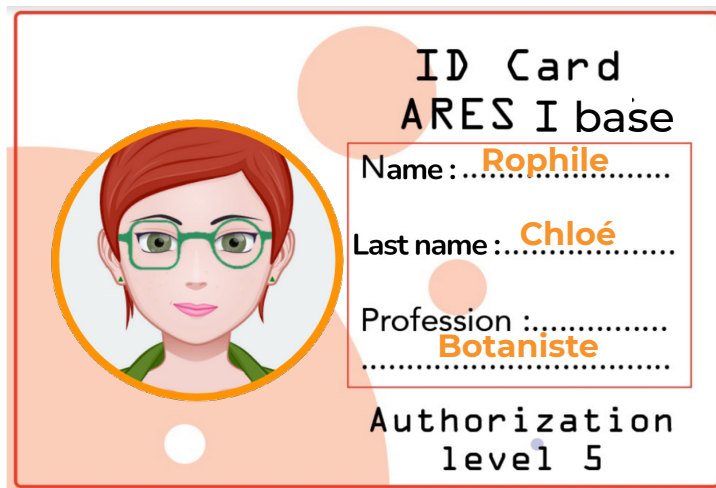
LIVRET PÉDAGOGIQUE ROBOT MARTIEN

Édition rédigée par
@P7i7Plum3 et Damien Vallot
avec le soutien de l'équipe "Survive on Mars"
Mise en page et illustration par
Laura Venezia et Diana Khalipina

Tous droits d'auteurs réservés

• Edition 2021 •

Imprimé en Italie



**NOS DEUX COLONS
MARTIENS PRÊTS À
RELEVER TOUS LES DÉFIS !**

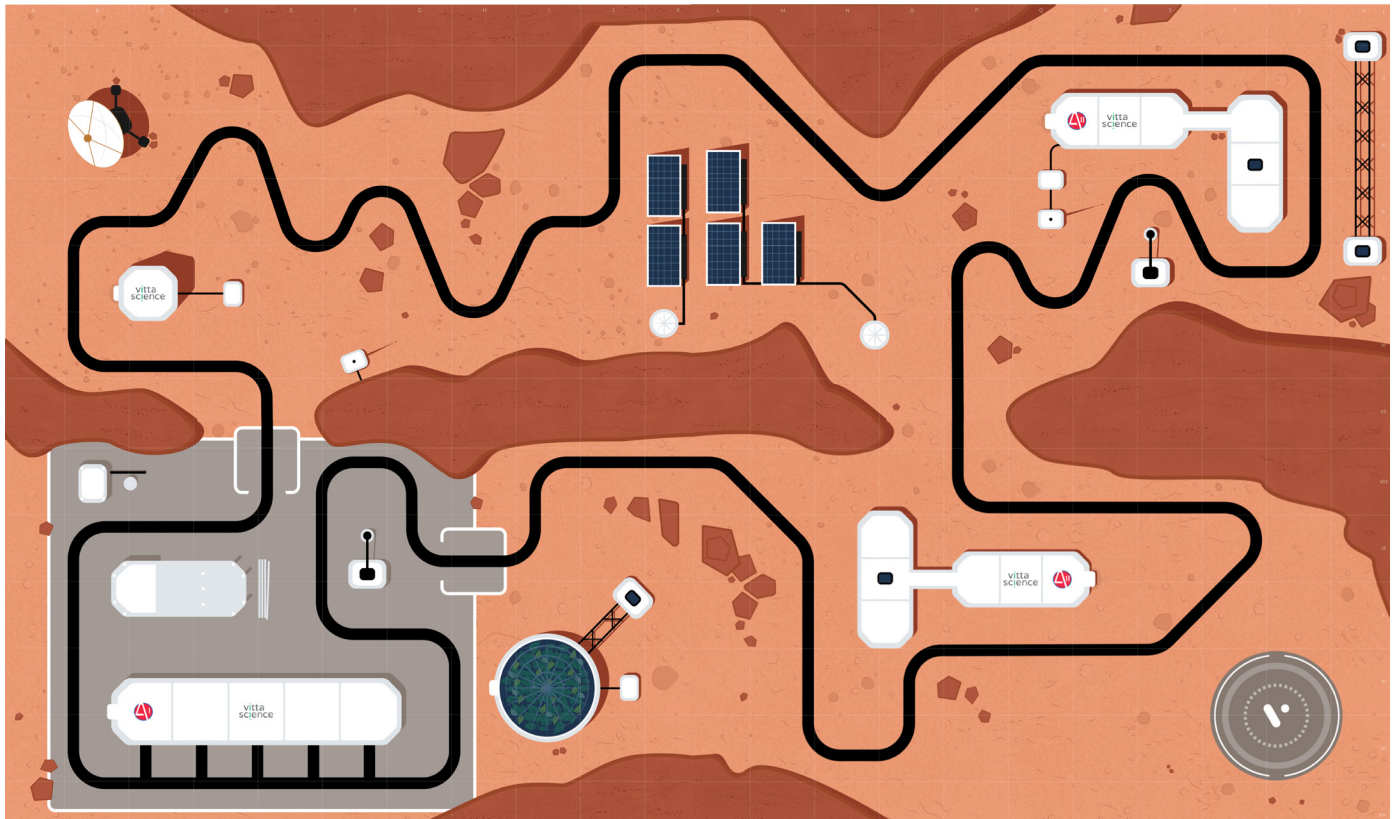
Nous sommes en 2041. Le succès de la mission sur Mars d'Ingenuity en 2021 est encore dans toutes les mémoires. Ce robot volant de la NASA, déployé aux côtés du rover Perseverance, avait validé le concept de robot attaché à un centre de pilotage sur Mars.

Suite à la réussite de cette mission, de nombreuses autres ont été réalisées avec succès. Depuis l'installation sur Mars de la base souterraine Arès I en 2039, nous cherchons à coloniser la surface.

Pour cela, en 2040, une nuée de robots a été déployée en surface afin de patrouiller, analyser et construire la futur base Arès prévue pour 2042. L'ASM (Agence Spatiale Martienne) basée sur Terre pilote l'ensemble du projet avec l'aide des premiers colons martiens sur place.

Dans les missions, nous aurons affaire à deux de nos colons les plus émérites de la base Arès I. En effet, suite à un incident survenu sur la base, ceux-ci font face à plusieurs défis !

LE THÉÂTRE DES OPÉRATIONS

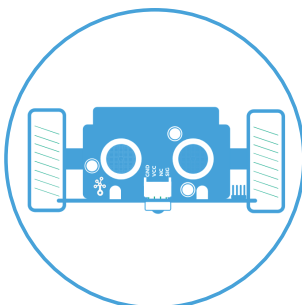


MATÉRIEL NÉCESSAIRE À LA CONSTRUCTION ET À L'UTILISATION DU KIT ROBOT MARTIEN

Contenu du KIT :



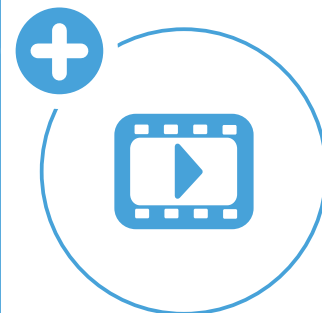
Carte micro:bit



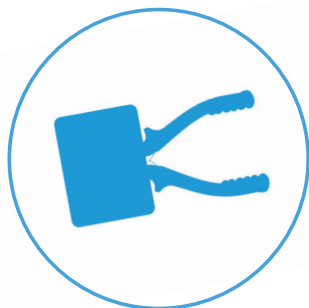
Robot Maqueen



Piste pour robot



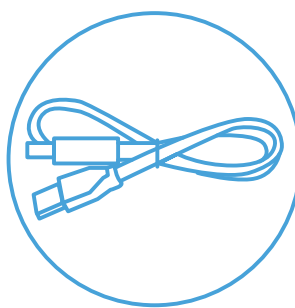
**Ressources
numériques
en ligne**



**Pince pour
Robot Maqueen**



**Support de batterie +
batterie
Télécommande IR**



**Éléments pour le
montage**



**Prévoir un
ordinateur**

AVERTISSEMENTS CONCERNANT L'UTILISATION DU KIT



ATTENTION !

Présence de petits éléments, ne pas ingérer (risque d'étouffement).



Age minimum

Ne convient pas aux enfants de moins de 7 ans.

CONSIGNES POUR BIEN TRIER

L'infographie suivante détaille les consignes de tri des différents éléments du kit. Pour plus d'informations, rendez-vous sur le site www.consignesdetri.fr



SOMMAIRE

PAGE
14

Ici ASM. Chloé Rophile et Rock Fort, rendez-vous au laboratoire de robotique pour un inventaire du matériel.



ASM



CHLOÉ

Bien reçu ASM.

Nous rejoignons immédiatement le Labo.



ROCK

PAGE
17

Le kit Robot Martien et l'environnement de programmation

- Le kit Robot Martien
 - Le contenu du kit
 - La carte micro:bit
- L'environnement Vittascience
 - La création de compte
 - L'interface de programmation

Mission 1 : Découverte et utilisation du matériel

- Mission 1-1 : Afficher icône
- Mission 1-2 : Afficher texte
- Mission 1-3 : Utilisation des boutons
- Mission 1-4 : Utilisation du capteur : Accéléromètre
- Mission 1-5 : Utilisation des capteur : Boussole / Température / Luminosité



PAGE
21

PAGE
26

Mission 2 : Piloter un servo moteur

- Mission 2-1 : Piloter un servo moteur
- Mission 2-2 : Piloter les moteurs

Mission 3 : Tester le détecteur d'obstacle

- Mission 3-1 : Découverte du capteur à ultrason
- Mission 3-2 : Ultrason et moteurs

PAGE
29



PAGE
44

Mission 4 : Communication entre carte

Mission 5 : Pilotage temps réel

- Mission 5-1 : Cas de la télécommande
- Mission 5-2 : Cas d'une deuxième carte micro:bit
- Mission 5-3 : Test en condition réel martienne

Mission 6 : Déplacement en mode case

- Mission 5-1 : Déplacement en mode case
- Mission 5-2 : Détection obstacle avant déplacement

Arès I à l'écoute



ARÈS I



ASM

Chloé Rophile et Rock Fort viennent d'être désignés pour remplacer votre programmeur-roboticien actuellement en réanimation suite à l'incident de dépressurisation. Ils devront mettre de côté leurs missions d'informaticienne et de mécanicien. Qu'ils se tiennent prêts à recevoir nos instructions. Fin de transmission.

Bien reçu. Fin de transmission.



ARÈS I

PAGE
60

PAGE
49

Mission 7 : Suiveur de ligne

- Mission 7-1 : Suivre une ligne
- Mission 7-2 : Suivre une ligne et éviter les obstacles

Mission 8 : Déplacer un objet

- Mission 8-1 : Attraper un objet
- Mission 8-2 : Déplacer un objet d'un endroit à l'autre

Mission 9 : Mise en situation complexe

- Mission 9 : Enfin sur Mars

Pour aller plus loin

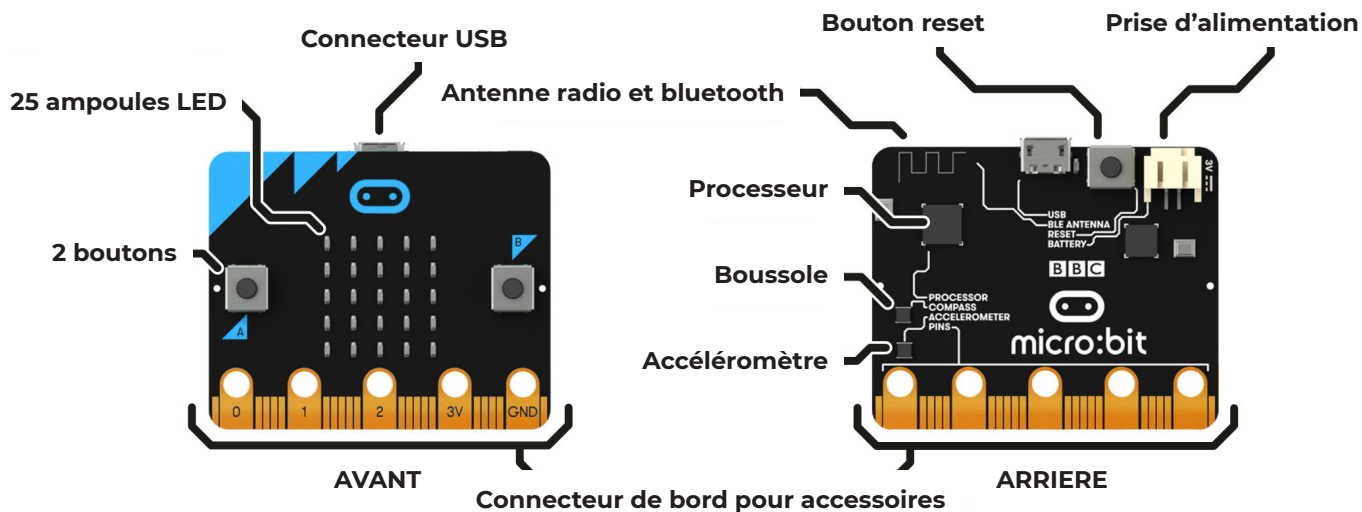
Présentation du microcontrôleur et de l'interface Vittascience

• Présentation de la carte micro:bit à l'appréciation de l'encadrant

Le schéma suivant montre les entrées et sorties de la carte micro:bit. Celle-ci peut servir de support à de nombreux montages différents avec les composants fournis.



Conseil : Des tutoriels interactifs sont disponibles dans l'onglet «Ressources» du site vittascience.com, nous conseillons de les suivre avant de réaliser des montages complexes.



• Programmation de la carte 🌀 à l'appréciation de l'encadrant

Nous détaillons ici le fonctionnement de l'interface de programmation en ligne [Vittascience.com](https://vittascience.com).

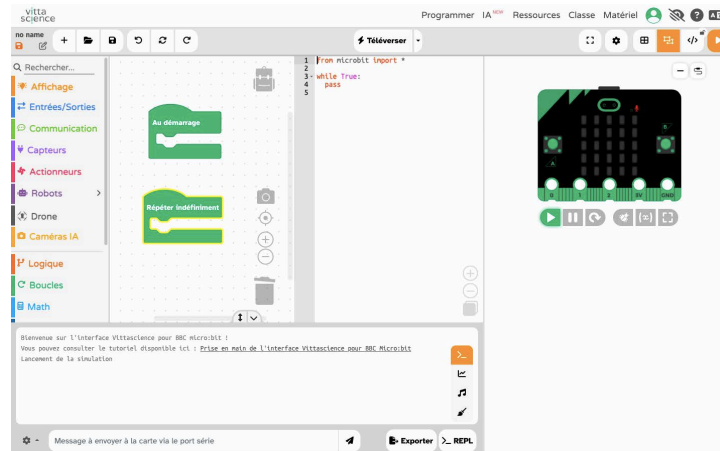
Il est également possible de programmer la carte à l'aide des logiciels Arduino (langage C++). Des tutoriels pour ce logiciel sont disponibles dans la librairie de ressources du site [Vittascience.com](https://vittascience.com).

1 • Création d'un compte

Avant tout, nous vous conseillons de vous créer un compte sur notre site. Celui-ci n'est en aucun cas nécessaire pour profiter de votre kit, mais il vous permettra de sauvegarder et partager vos programmes, ressources et retours d'expériences.

Pour cela, rendez-vous sur le site [Vittascience.com](https://vittascience.com) et cliquez sur l'icône verte en haut à droite pour vous inscrire.

2 • L'interface



L'interface permet de programmer en bloc avec une transcription en parallèle en langage Python.

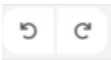


Attention : La carte micro:bit est nécessaire pour exécuter le programme une fois les capteurs positionnés !

Retrouvez sur le site [Vittascience.com](https://vittascience.com) des ressources et des programmes pour apprendre à programmer avec la carte.

Sélection du port :  lorsque vous connectez la carte à l'ordinateur, l'interface détecte automatiquement sur quel port de ordinateur la carte est connectée. Le menu déroulant permet de sélectionner le bon port si plusieurs cartes sont connectées à l'ordinateur.

Transférer le programme vers la carte :  le code est exécuté sur la carte dès la fin du transfert.

Annuler ou rétablir :  les actions précédentes ou suivantes.

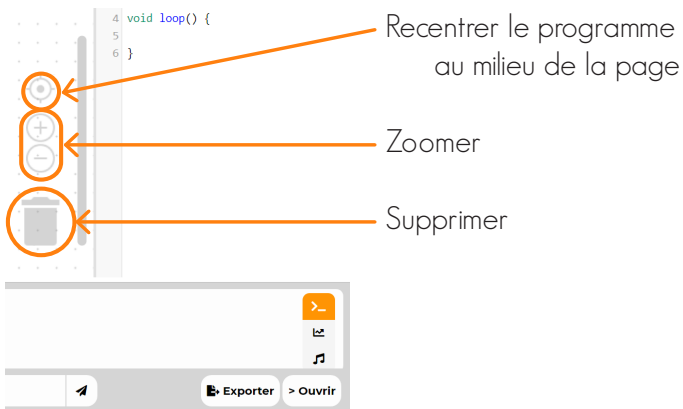
Démarrer un nouveau projet :  pour lancer un nouveau projet vierge, cliquer sur ce bouton.

Sauvegarder le projet :  pour enregistrer votre projet afin de le sauvegarder, cliquer sur ce bouton. Pour les personnes disposant d'un compte, il est possible de partager le programme avec la communauté.

Ouvrir un projet existant :  Si vous souhaitez ouvrir des programmes déjà réalisés, ou bien faire travailler vos élèves sur une trame que vous avez créée, ils peuvent y accéder en cliquant sur ce bouton.

Coder en Python :  si vous souhaitez coder directement en Python.

Accès rapide :



Conseil : Cette interface de programmation est conçue pour être très simple d'utilisation, n'hésitez pas à la tester et à la proposer à vos élèves.

Mission • 1 à l'appréciation de l'encadrant

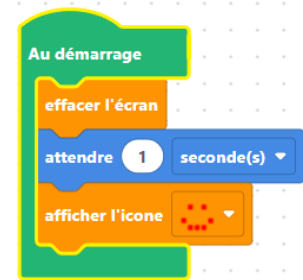
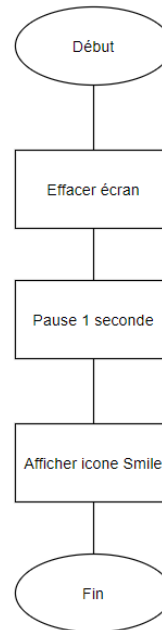
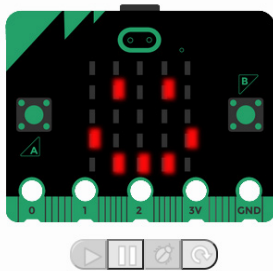
Découverte et utilisation du matériel

Découverte de la carte contrôleur du futur mini rover (mission très simple pour maîtriser le transfert de programme et l'interface de la plateforme Vittascience).

Voici dans l'ordre l'algorithme puis le code de programmation par bloc puis enfin le code Python.

• Mission 1-1 : Afficher icône

Pour débiter cette première mission nous allons afficher un dessin d'un smiley. Comme sur l'image ci contre.



```

1 from microbit import *
2 import time
3
4 display.clear()
5 time.sleep(1)
6 display.show(Image.HAPPY)
7

```

Votre planning sera très chargé afin de maîtriser l'ensemble des technologies utiles à ce projet. Il faudra peut-être faire preuve d'initiative afin de développer de nouvelles compétences qui n'étaient pas prévues dans votre formation initiale pour venir sur Mars. Commençons par quelque chose de simple. Vous allez être confrontés à quelques petits exercices de programmation pour vous familiariser avec les procédures d'utilisation de la carte microcontrôleur présentée dans tous les robots radioguidés de la surface.



ROCK

Ho-la-la, il va falloir qu'on manipule ces petits trucs ?? Mais je vais les écraser entre mes gros doigts !

Pfff, ça fait plus de 30 ans qu'on utilise des microcontrôleurs dans nos rovers et ça fonctionne toujours ! Imagine, il y en avait déjà dans Perseverance ! C'est du solide !



CHLOÉ



Procédure en cours d'envoi. Message terminé.

Dans l'interface de programmation de Vittascience, on trouvera au choix pour programmer soit en Python soit en code bloc. Dans un premier temps et en fonction du niveau, il est conseillé d'utiliser le code bloc qui est plus simple pour débiter.

Les bons blocs sont dans le menu de gauche, pour commencer seuls les deux suivants seront utiles.

 Affichage

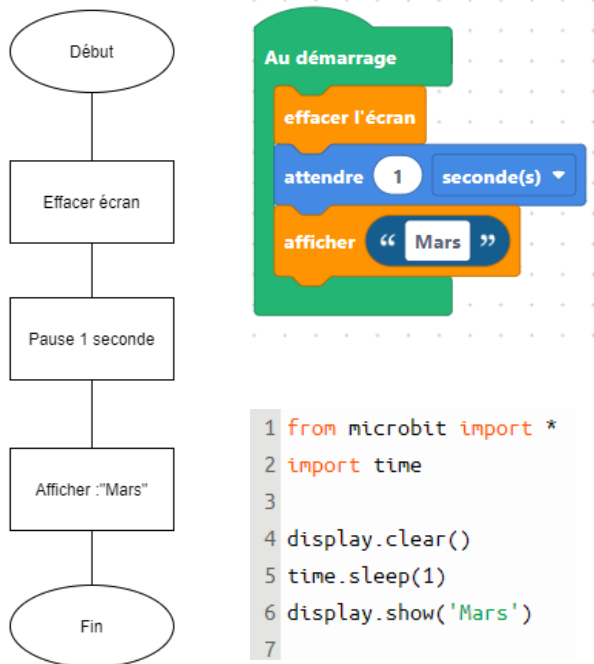
 Entrées/Sorties

• Mission 1-2 : Afficher texte

Maintenant que nous avons réussi à afficher le smiley nous pouvons afficher un texte "Mars". Le principe est le même mais nous utiliserons le bloc "afficher ou faire défiler" et non "afficher l'icône".

Le texte Mars devrait s'afficher sur la carte programmable. Dans le premier cas les lettres se remplacent une à une dans le deuxième le texte défile.

Voici dans l'ordre l'algorithme puis le code de programmation par bloc puis enfin le code Python.



• Mission 1-3 : Utilisation des boutons

Maintenant que nous avons compris comment fonctionnait la carte dans ses fonctions de base pour afficher un texte ou une image. Nous allons interagir avec celle-ci dans un premier temps avec les boutons.

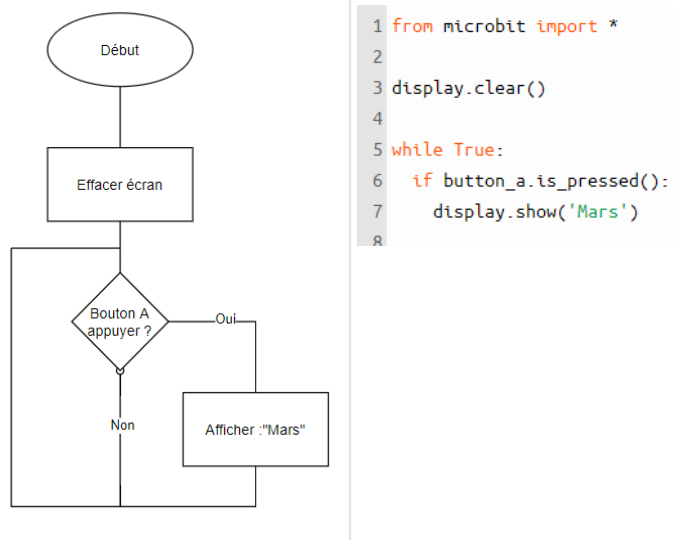
Nous allons garder l'affichage de "Mars" comme code de base mais maintenant celui-ci ne s'affichera que lorsque nous aurons appuyé sur le bouton A de la carte. Pour cela nous allons avoir besoin de nouvelles blocs qui se trouvent dans les menus suivants :

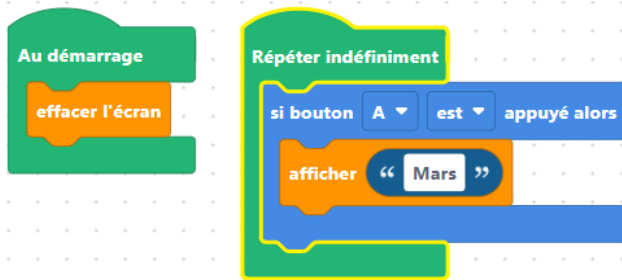
 Robots

 Entrées/Sorties

 Boucles

Voici dans l'ordre l'algorithme puis le code de programmation par bloc puis enfin le code Python.



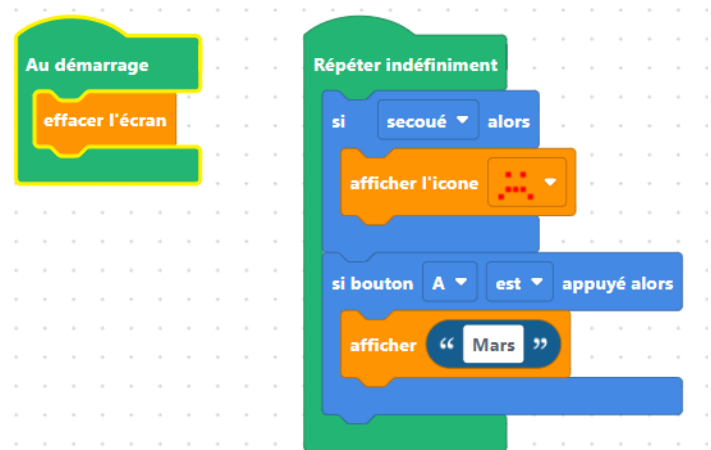
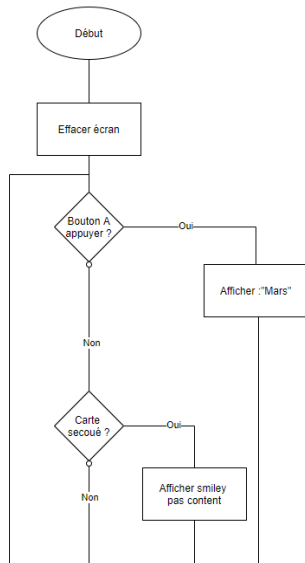


• Mission 1-4 : Utilisation du capteur : Accéléromètre

À la place des boutons qui ne sont que des capteurs de pression basique (appuyer ou pas) nous allons utiliser l'accéléromètre de la carte programmable.

Pour cette mission on va compléter le code précédent avec la fonction suivante : si je secoue la carte elle affiche un smiley pas content. (Là encore l'information reçue sera basique je secoue ou pas.)

Voici dans l'ordre l'algorithme puis le code de programmation par bloc puis enfin le code Python.



```

1 from microbit import *
2
3 display.clear()
4
5 while True:
6     if accelerometer.current_gesture() == 'shake' :
7         display.show(Image.SAD)
8     if button_a.is_pressed():
9         display.show('Mars')
10

```

• Mission 1-5 : Utilisation des capteur : Boussole / Température / Luminosité

Dans cette mission nous allons utiliser des capteurs qui nous donnent des informations de nature plus importantes que les capteurs précédents ou la réponse était "vrai ou faux", ici nous allons obtenir des informations chiffrées. Par exemple : Quelle est la température ? ou Est ce qu'il y a beaucoup de lumière ?

Pour l'instant nous allons faire simple et afficher la température de la pièce quand on appuie sur le bouton A grâce au capteur de la carte programmable.

Pour cela nous allons avoir besoin de blocs qui se trouvent dans :

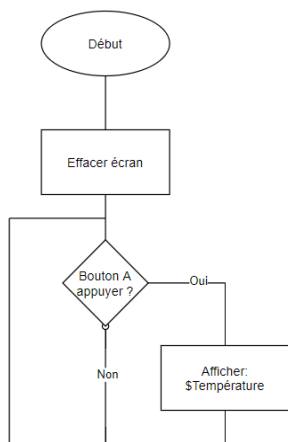
Capteurs

Entrées/Sorties

Boucles

Affichage

Voici dans l'ordre l'algorithme puis le code de programmation par bloc puis enfin le code Python.



```

1 from microbit import *
2
3 display.clear()
4
5 while True:
6     if button_a.is_pressed():
7         display.show(temperature())
  
```

Dans l'algorithme on peut remarquer qu'un symbole \$ a été ajouté devant le mot température, c'est pour signifier que ce n'est pas simplement le mot "température" mais la Variable température. (Une variable est un endroit où l'on stocke une information de manière temporaire afin de la réutiliser plus tard). Par exemple dans notre cas la carte programmable à stocker dans la variable \$température la valeur de la température. À chaque fois que j'aurais besoin du nombre qui représente la température j'appellerais donc \$température.

Pour l'utilisation de la boussole ou de la luminosité, le principe est exactement le même. On utilisera juste les étiquettes luminosité et direction de la boussole à la place.

Et pour nommer les variables pour ne pas confondre avec le mot luminosité on rajoute un \$devant. Cela donnera pour la luminosité : \$luminosité et pour la boussole : \$boussole.

Au démarrage

effacer l'écran

Répéter indéfiniment

si bouton A est appuyé alors

afficher température en (°C)

Mission • 2 à l'appréciation de l'encadrant

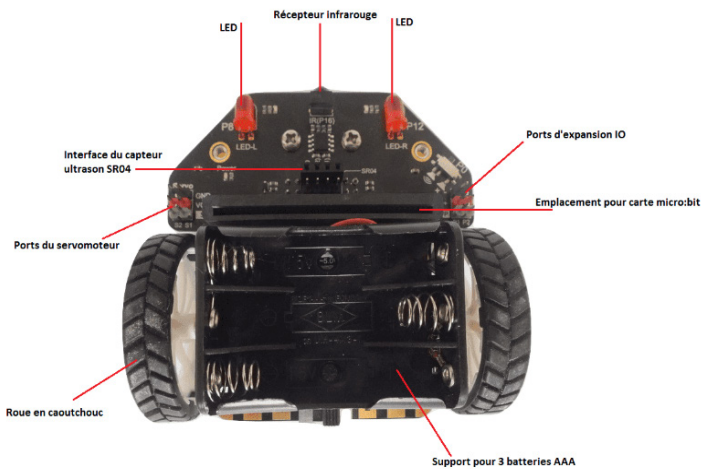
Piloter un servo moteur

• Mission 2-1 : Piloter une servo moteur

Pour commencer il faut déjà brancher sur la carte un servo moteur (attention au sens de branchement du servo moteur).

Il y a trois couleurs sur le câble du servo moteur un jaune un rouge et enfin un marron. Le marron correspond habituellement au noir que l'on utilise en physique. Donc celui-ci sera branché sur la borne GND (qui veut dire ground / terre).

Sur la carte du robot Maqueen il y a un emplacement prévu au branchement des servo moteurs sur le côté de la carte nommé S1 et S2.



Ici ASM. Les robots constructeurs de surface sont actuellement à l'arrêt, et les robots sont dans des positions inconnues. Vous allez maintenant apprendre à déplacer les robots et les remettre en position afin de les rapatrier pour une révision.



ROCK

Cool, il est où le joystick ?



CHLOÉ

À mon avis c'est pas comme avec tes perceuses où tu appuies juste sur un bouton !!

Une fois les branchements effectués nous allons enfin pouvoir programmer le servo moteur.

Pour cette première programmation nous allons juste faire tourner le servo dans un sens avec le bouton A puis faire tourner dans l'autre sens avec le bouton B. Nous irons de 0 à 180° de rotation qui est la limite des petits servo moteurs du Maqueen.

Les servo moteurs sont des actionneurs. Ils font partie de la chaîne d'énergie.

Pour cela nous allons avoir besoin de blocs qui se trouvent dans :

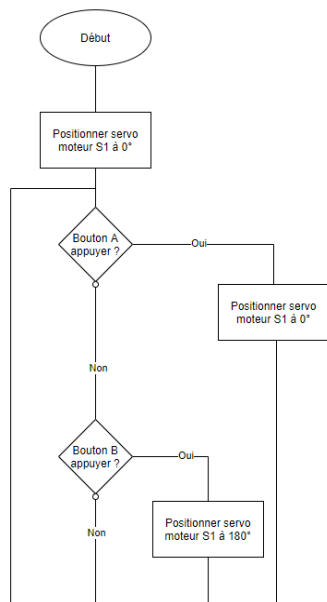
 Capteurs

 Entrées/Sorties

 Boucles

 Robots

Voici dans l'ordre l'algorithme puis le code de programmation par bloc puis enfin le code Python.



Au démarrage

[Maqueen] positionner le servomoteur S1 à l'angle 0

Répéter indéfiniment

si bouton A est appuyé alors

[Maqueen] positionner le servomoteur S1 à l'angle 0

si bouton B est appuyé alors

[Maqueen] positionner le servomoteur S1 à l'angle 180

```

1 from microbit import *
2
3 i2c.write(0x10, bytearray([0x14, 0]))
4
5 while True:
6     if button_a.is_pressed():
7         i2c.write(0x10, bytearray([0x14, 0]))
8     if button_b.is_pressed():
9         i2c.write(0x10, bytearray([0x14, 180]))
  
```

• Mission 2-2 Piloter les moteurs

Il est enfin temps de faire déplacer notre petit robot. Pour l'instant nous allons nous contenter de mouvements simples : avancer/tourner.*

Il y a plusieurs manières de contrôler les roues du robot en fonction des blocs codes choisies qui sont au nombre de trois :

Au démarrage

Commentaire etiquette 1

[Maqueen] contrôler le moteur gauche direction vitesse 255

[Maqueen] contrôler le moteur droit direction vitesse 255

Commentaire etiquette 2

[Maqueen] contrôler le robot reculer vitesse 255

[Maqueen] contrôler le robot avancer vitesse 255

attendre 1 seconde(s)

Commentaire etiquette 3

[Maqueen] arrêter le moteur droit

[Maqueen] arrêter le moteur gauche

[Maqueen] arrêter le moteur droit & gauche

Étiquette 1 : cela permet de contrôler chaque moteur droit ou gauche de manière indépendante, de contrôler le sens de rotation et enfin de contrôler la vitesse de rotation.

Étiquette 2 : Permet de contrôler les deux moteurs droit et gauche en même temps pour leur sens de rotation et leur vitesse de rotation.

Étiquette 3 : Permet d'arrêter les moteurs de manière indépendante ou les deux en même temps.

• Mission 2-2 a

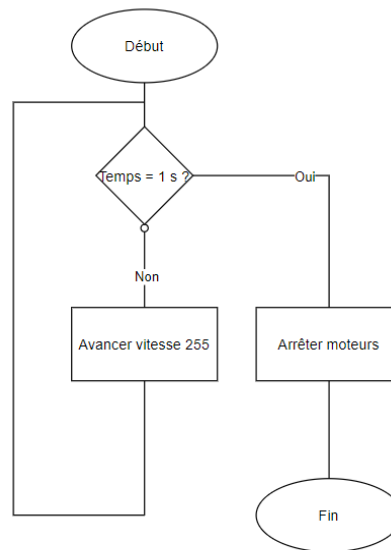
Pour notre premier programme nous allons nous contenter de faire avancer le robot en marche avant pendant 1 seconde.

Pour cela nous allons avoir besoin de blocs qui se trouvent dans :

 **Robots**

 **Entrées/Sorties**

Voici dans l'ordre l'algorithme puis le code de programmation par bloc puis enfin le code Python.



```
1 from microbit import *
2 import time
3
4 i2c.write(0x10, bytearray([0x00, 0x0, 255]))
5 i2c.write(0x10, bytearray([0x02, 0x0, 255]))
6 time.sleep(1)
7 i2c.write(0x10, bytearray([0x00, 0, 0]))
8 i2c.write(0x10, bytearray([0x02, 0, 0]))
```

• Mission 2-2 b

Un robot qui avance tout droit c'est bien mais c'est dangereux il va nous falloir lui apprendre à tourner pour cela nous allons utiliser le contrôle indépendant de chaque moteur.

Deux choix s'offrent à nous pour tourner la méthode char d'assaut ou la méthode voiture. Le char d'assaut fait tourner ses roues en sens opposé de chaque côté ce qui lui permet de tourner sur place. La voiture avec le système de différentiel a généralement une roue qui tourne plus vite que l'autre ce qui permet de tourner.

Chaque technique a ses avantages et ses inconvénients en fonction de son utilisation. La technique avec sens opposé permet de tourner plus vite mais au détriment de la vitesse globale de déplacement. Et bien sûr l'autre méthode c'est l'inverse.

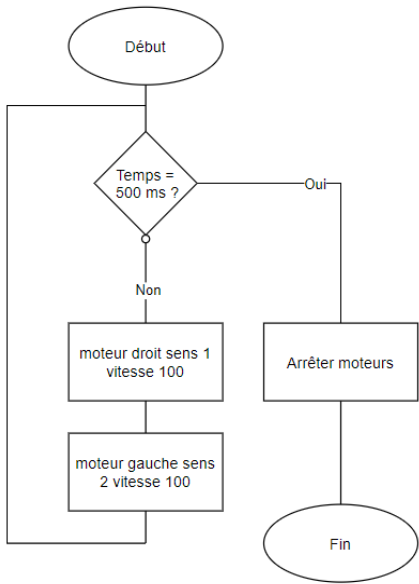
Dans notre cas nous allons nous déplacer dans un milieu hostile Mars donc la prudence et la manoeuvrabilité est plus importante donc nous allons commencer par utiliser la méthode char d'assaut avec un virage à 90°.

Pour cela nous allons avoir besoin de blocs qui se trouvent dans :

 Robots

 Entrées/Sorties

Voici dans l'ordre l'algorithme puis le code de programmation par bloc puis enfin le code Python.



Au démarrage

[Maqueen] contrôler le moteur droit direction vitesse 100

[Maqueen] contrôler le moteur gauche direction vitesse 100

attendre 500 milliseconde(s)

[Maqueen] arrêter le moteur droit & gauche

```
1 from microbit import *
2 import time
3
4 i2c.write(0x10, bytearray([0x02, 0x0, 100]))
5 i2c.write(0x10, bytearray([0x00, 0x1, 100]))
6 time.sleep_ms(500)
7 i2c.write(0x10, bytearray([0x00, 0, 0]))
8 i2c.write(0x10, bytearray([0x02, 0, 0]))
```

Le programme ne sera pas parfait, il faudra ajuster le temps afin d'obtenir un angle de 90°. Vous pouvez le faire par essai/erreur ou alors utiliser un tableau de proportionnalité. Ne pas hésiter à réduire la vitesse afin de mieux contrôler la direction.

Nous contrôlons maintenant parfaitement notre robot et pouvons nous déplacer où nous voulons.

• Mission 2-2 c

Nous savons avancer/reculer en ligne droite. Tourner à 90°. Montrer votre maîtrise en effectuant un carré.

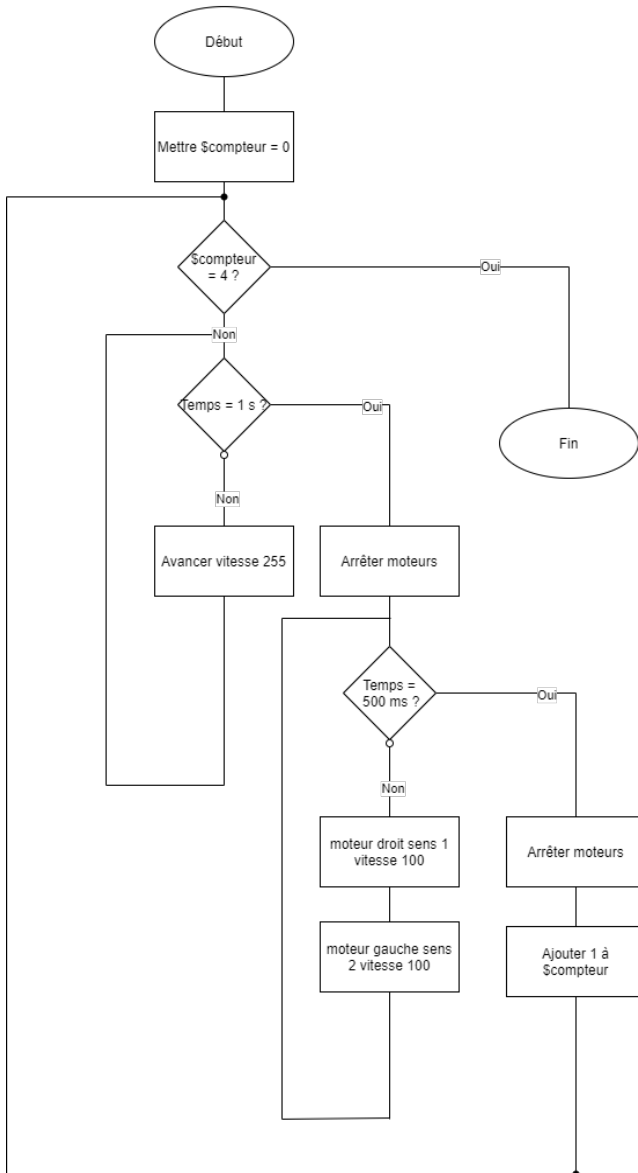
Pour cela nous allons avoir besoin de blocs qui se trouvent dans :

 Robots

 Entrées/Sorties

 Boucles

Attention l'algorithme devient plus compliqué en écriture à cause de la boucle qui va compter combien on de fois on a fait une série d'action, on va donc y voir apparaître une variable \$compteur.



Au démarrage

répéter 4 fois



```

1 from microbit import *
2 import time
3
4 for count in range(4):
5     i2c.write(0x10, bytearray([0x00, 0x0, 50]))
6     i2c.write(0x10, bytearray([0x02, 0x0, 50]))
7     time.sleep_ms(1000)
8     i2c.write(0x10, bytearray([0x00, 0, 0]))
9     i2c.write(0x10, bytearray([0x02, 0, 0]))
10    i2c.write(0x10, bytearray([0x02, 0x0, 50]))
11    i2c.write(0x10, bytearray([0x00, 0x1, 50]))
12    time.sleep_ms(450)
13    i2c.write(0x10, bytearray([0x00, 0, 0]))
14    i2c.write(0x10, bytearray([0x02, 0, 0]))
  
```

Mission • 3 à l'appréciation de l'encadrant

Tester le détecteur d'obstacle

• Mission 3-1 : Découverte du capteur à ultrason

ASM, on a un problème ! L'un des robots semble bloqué.



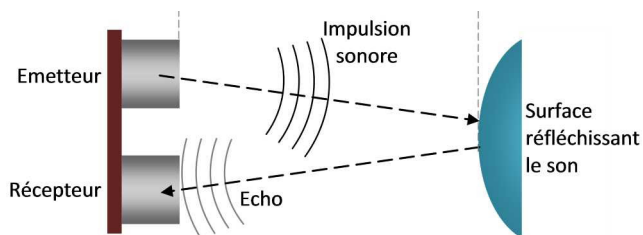


Ça doit sûrement être un problème de détecteur d'obstacles. A mon avis, il est coincé contre un rocher.

Bien reçu Arès I. Voici le protocole d'utilisation du capteur à Ultrason et du capteur TOF



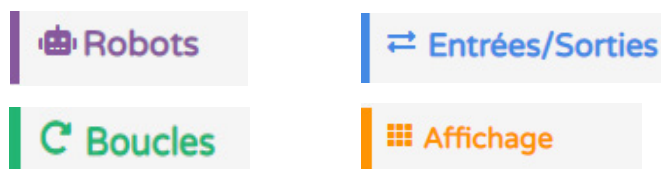
Le capteur à ultrason est un capteur qui permet de détecter un obstacle devant lui et même de déterminer une distance.



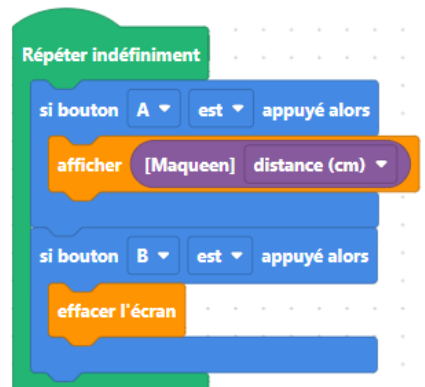
Il y a d'un côté un émetteur qui envoie une impulsion sonore. Lorsque l'impulsion percute un obstacle, celui-ci renvoie cette même impulsion. Dès lors un calculateur "compte" combien de temps l'impulsion sonore a mis de temps pour faire l'aller et le retour. Connaissant la vitesse de déplacement du son dans l'air on peut en déduire la distance ($V=d/t$).

Pour cette mission nous allons afficher la distance d'un objet à chaque fois que l'on appuiera sur le bouton A et nous effacerons l'écran à chaque fois que l'on appuiera sur B.

Pour cela nous allons avoir besoin de blocs qui se trouvent dans :



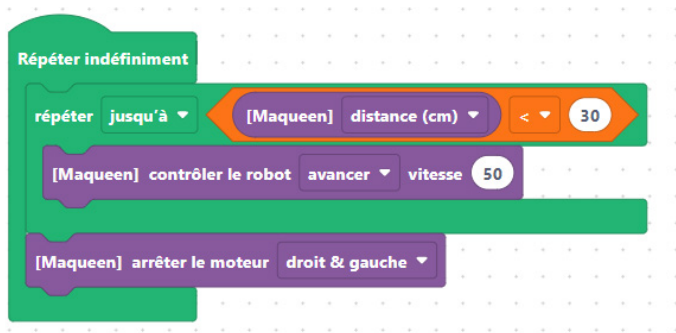
Ici le programme Python devient plus compliqué car le début du programme est une fonction (un ensemble d'instruction que l'on peut rappeler plus tard, une sorte de sous-programme) qui sert à faire le calcul de la distance on y repère le chiffre 343 qui est la distance en m/s de la vitesse des ultrason le temps divisé par deux car il y a l'aller et le retour et le multiplie par 100 pour remettre le tout en cm.



```

1 from microbit import *
2 from machine import time_pulse_us
3 import utime
4
5 def getUltrasonicData(trig, echo, data='distance', timeout_us=30000):
6     trig.write_digital(0)
7     utime.sleep_us(2)
8     trig.write_digital(1)
9     utime.sleep_us(10)
10    trig.write_digital(0)
11    echo.read_digital()
12    duration = time_pulse_us(echo, 1, timeout_us)/1e6 # t_echo in seconds
13    if data == 'distance':
14        #sound speed, round-trip/2, get in cm
15        return 343 * duration/2 * 100
16    elif data == 'duration':
17        return duration
18    else:
19        raise ValueError("Data option '" + data + "' is not valid")
20
21 while True:
22     while not getUltrasonicData(pin1, pin2, 'distance') < 30:
23         i2c.write(0x10, bytearray([0x00, 0x0, 50]))
24         i2c.write(0x10, bytearray([0x02, 0x0, 50]))
25     i2c.write(0x10, bytearray([0x00, 0, 0]))
26     i2c.write(0x10, bytearray([0x02, 0, 0]))

```



Si vous mesurez la distance obtenue après déplacement du robot, il se peut qu'il y ait une légère différence. Cela est dû à deux paramètres

La première du a une erreur humaine, on oublie qu'il ne faut pas mesurer aux roues ou à l'avant du robot mais à la base du capteur à ultrasons qui est celui qui fait la mesure.

La deuxième du à l'inertie du robot et à la vitesse de calcul du robot. Le robot ne peut pas s'arrêter instantanément et plus la vitesse est grande plus l'erreur du au freinage est grande. De même, le temps de calcul de la carte qui la plupart du temps est négligeable peut engendrer un léger écart de quelques mm si la vitesse est grande.

• Mission 3-3 : capteurs de distance et moteurs

Nous allons tout de suite exploiter ce capteur à ultrasons avec le déplacement du robot. Celui-ci devra se déplacer jusqu'à ce que la distance entre le robot et un mur soit la bonne distance par exemple 30 cm.

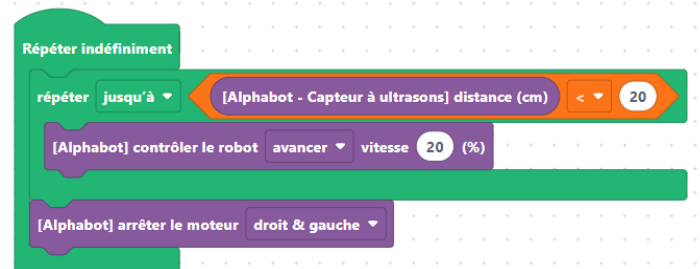
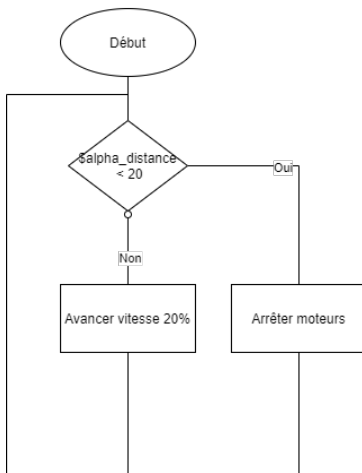
Pour cela nous allons avoir besoin de blocs qui se trouvent dans :

Robots

Logique

Boucles

Nous allons utiliser dans cette mission les comparateurs logiques, ceci permettant de vérifier si une information est vraie ou fausse. Une fois cette vérification faite, cela permet d'enclencher telle ou telle action. Dans notre cas, la question sera : Est ce que la distance entre mon robot et le mur est inférieure à 20 cm ?



```

1 import machine
2 from stm32_ssd1306 import SSD1306, SSD1306_I2C
3 from stm32_alphabot_v2 import AlphaBot_v2
4
5 oled = SSD1306_I2C(128, 64, machine.I2C(1))
6 alphabot = AlphaBot_v2()
7
8 oled.fill(0)
9 oled.show()
10
11 while True:
12     while not alphabot.readUltrasonicDistance() < 20:
13         alphabot.moveForward(20)
14     alphabot.stop()
  
```

Si vous mesurez la distance obtenue après déplacement du robot, il se peut qu'il y ait une légère différence. Cela est dû à deux paramètres :

- Le premier paramètre est dû à une erreur humaine, on oublie qu'il ne faut pas mesurer au niveau des roues mais plutôt au niveau du capteur de distance qui réalise la mesure.
- Le deuxième paramètre est dû à l'inertie du robot et à la vitesse de calcul du robot. Le robot ne peut pas s'arrêter instantanément et plus la vitesse est grande plus l'erreur due au freinage est grande. De même, le temps de calcul de la carte qui la plupart du temps est négligeable peut engendrer un léger écart de quelques mm si la vitesse est grande.

Mission • 4 à l'appréciation de l'encadrant

Communication entre carte

Mission 4-1 émetteur

ASM, ici Arès I. Nous confirmons le problème: le robot bloqué est trop proche d'un obstacle.

On pourrait peut-être mettre nos combinaisons pour aller le décoincer ?

Hors de question Chloé ! C'est trop dangereux pour vous, vous allez utiliser une télécommande pour robot pour le dégager. Voici une procédure pour apprendre à communiquer avec les robots.


ROCK


CHLOÉ


ASM

Ceci est une petite mission afin de comprendre le principe de fonctionnement de transmissions d'une carte programmable à une autre.

Pour commencer il faut comprendre que l'on va avoir 2 cartes programmables à programmer en même temps une émettrice et une réceptrice. Il ne faut pas perdre de vue que la communication s'effectue dans les deux sens donc la carte sera à tour de rôle réceptrice et émettrice.

Pour ce premier petit programme nous allons juste envoyer d'une carte à l'autre un nombre et l'afficher. Bien sûr pour envoyer ce nombre il faudra appuyer sur le bouton A.

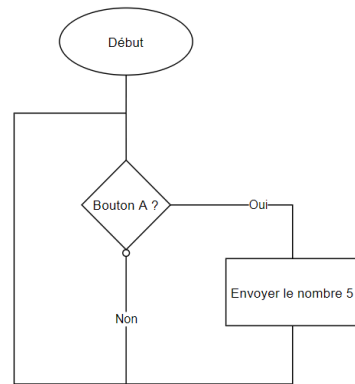
Pour cela nous allons avoir besoin de blocs qui se trouvent dans :

 **Affichage**

 **Boucles**

 **Communication**

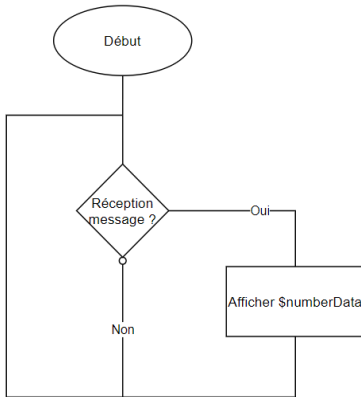
 **Variables**



```

1 from microbit import *
2 import radio
3
4 def radio_sendNumber(data, type='number'):
5     if type == 'number':
6         if data > int(data):
7             radio.send("%%float:" + str(data) + "%%")
8         else:
9             radio.send("%%int:" + str(data) + "%%")
10    elif type == 'boolean':
11        radio.send("%%bool:" + str(int(data)) + "%%")
12    else:
13        raise ValueError("Unable to send number: " + str(data) + "")
14
15 radio.on()
16
17 while True:
18     if button_a.is_pressed():
19         radio_sendNumber(5, type='number')
  
```

Mission 4-1 récepteur



```

1 from microbit import *
2 import radio
3
4 def radio_sendNumber(data, type='number'):
5     if type == 'number':
6         if data > int(data):
7             radio.send("%%float:" + str(data) + "%%")
8         else:
9             radio.send("%%int:" + str(data) + "%%")
10    elif type == 'boolean':
11        radio.send("%%bool:" + str(int(data)) + "%%")
12    else:
13        raise ValueError("Unable to send number: " + str(data) + "")
14
15 radio.on()
16
17 while True:
18     if button_a.is_pressed():
19         radio_sendNumber(5, type='number')
  
```

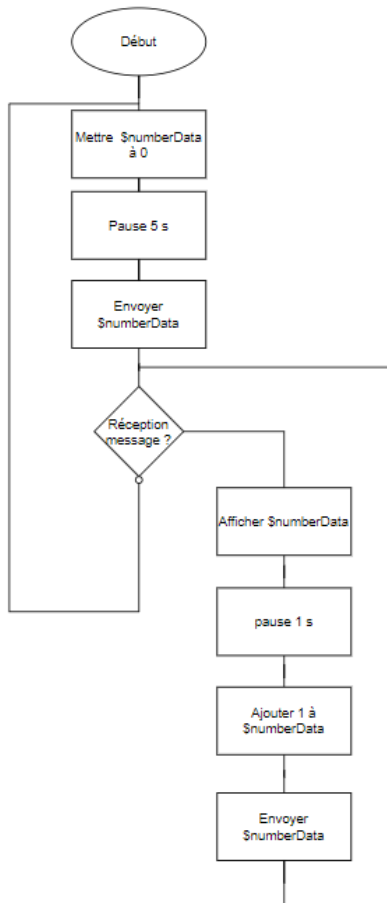
Grâce à cette fonction de communication nous pouvons faire de nombreuses actions.

Il faut bien comprendre que le transfert d'information est bi directionnelle et que cela fonctionne dans les deux sens. Les cartes peuvent se "parler". Il est possible de transférer bien plus que des nombres. On peut envoyer des chaînes de caractères comme des mots par exemple ou des codes. De plus, il est possible d'avoir des communications privées entre certaines cartes. Il suffit pour cela de configurer les canaux dans lesquels les cartes vont communiquer. Par exemple, on peut décider qu'une salle de classe communique sur le canal 1 et une autre salle de classe sur le canal 2.

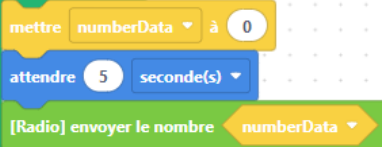
On peut imaginer comme petite mission de faire compter les cartes de 1 en 1 chacune leur tour. Ne pas oublier de mettre une pause entre chaque communication car les machines parlent trop vite pour nous sinon. Pour que le programme soit plus simple cette fois le programme démarrera dès l'activation de la carte après une pause de 5 secondes.

Le code émetteur est à gauche et le récepteur à droite.

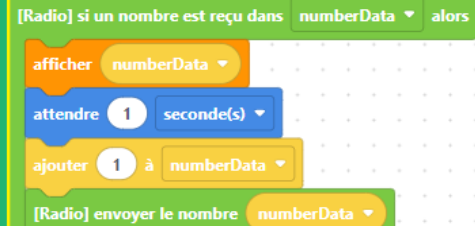
Mission 4-2 émetteur



Au démarrage



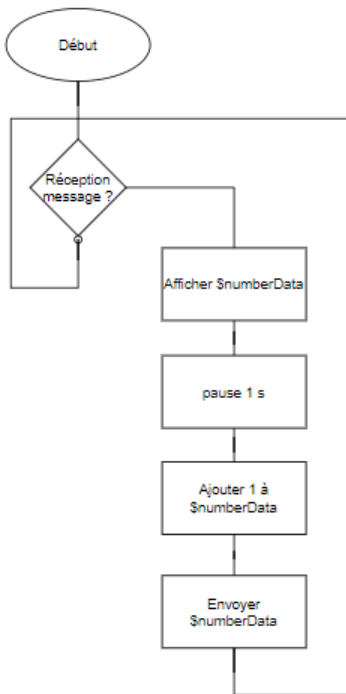
Répéter indéfiniment



```

1 from microbit import *
2 import time
3 import radio
4
5 def radio_sendNunber(data, type='number'):
6     if type == 'number':
7         if data > int(data):
8             radio.send("%%float:" + str(data) + "%%")
9         else:
10            radio.send("%%int:" + str(data) + "%%")
11    elif type == 'boolean':
12        radio.send("%%bool:" + str(int(data)) + "%%")
13    else:
14        raise ValueError("Unable to send number: " + str(data) + "!")
15
16 def radio_receiveData():
17     data = radio.receive()
18     if data:
19         if data.find('%%int:') != -1:
20             return int(data[7:-3])
21         elif data.find('%%float:') != -1:
22             return float(data[9:-3])
23         elif data.find('%%bool:') != -1:
24             return bool(data[8:-3])
25     else:
26         return data
27 else:
28     return None
29
30 radio.on()
31
32 numberData = 0
33 time.sleep(5)
34 radio_sendNunber(numberData, type='number')
35
36 while True:
37     numberData = radio_receiveData()
38     if numberData is not None:
39         display.show(numberData)
40         time.sleep(1)
41         numberData = numberData + 1
42         radio_sendNunber(numberData, type='number')
  
```

Mission 4-2 récepteur



Répéter indéfiniment

[Radio] si un nombre est reçu dans `numberData` alors

- afficher `numberData`
- attendre 1 seconde(s)
- ajouter 1 à `numberData`
- [Radio] envoyer le nombre `numberData`

```

1 from microbit import *
2 import radio
3 import time
4
5 def radio_receiveData():
6     data = radio.receive()
7     if data:
8         if data.find('iint:') != -1:
9             return int(data[7:-3])
10        elif data.find('float:') != -1:
11            return float(data[9:-3])
12        elif data.find('bool:') != -1:
13            return bool(data[8:-3])
14        else:
15            return data
16    else:
17        return None
18
19 def radio_sendNumber(data, type='number'):
20     if type == 'number':
21         if data > int(data):
22             radio.send("iifloat:" + str(data) + "iif")
23         else:
24             radio.send("iifint:" + str(data) + "iif")
25     elif type == 'boolean':
26         radio.send("iifbool:" + str(int(data)) + "iif")
27     else:
28         raise ValueError("Unable to send number: " + str(data) + "iif")
29
30 radio.on()
31
32 while True:
33     numberData = radio_receiveData()
34     if numberData is not None:
35         display.show(numberData)
36         time.sleep(1)
37         numberData = numberData + 1
38         radio_sendNumber(numberData, type='number')
39

```

Mission • 5 à l'appréciation de l'encadrant

Pilotage temps réel

ASM Nous avons réussi à communiquer avec le robot mais nous avons un doute sur la procédure pour le faire bouger.



CHLOÉ



ROCK

On pourrait peut-être utiliser le robot du hangar pour établir la liaison !

Pas d'inquiétude! On vous envoie une procédure. Et pas de souci pour le robot du Hangar.



ASM

Afin de piloter le petit robot nous avons deux possibilités soit 2 cartes micro bit soit la télécommande infra rouge fourni dans le kit.

Le principe reste le même dans les deux cas, le robot reçoit une information et réagit en fonction de l'information qu'il reçoit.

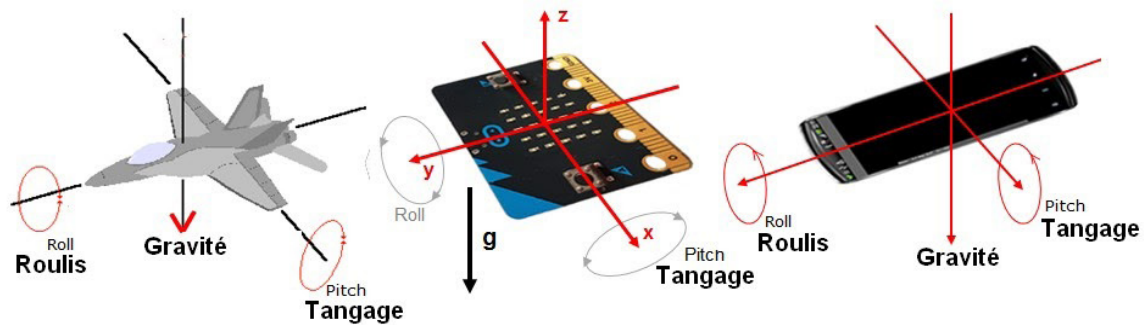
• Mission 5-1 : Cas de la télécommande

Dans une première mission nous allons faire afficher sur la carte une flèche vers le haut quand on appuie sur le bouton haut puis une flèche vers la droite quand on appuie sur le bouton droit etc .. pour toutes les directions et enfin un stop quand on appuie sur le bouton stop .

• Mission 5-2 : Cas d'une deuxième carte microbit

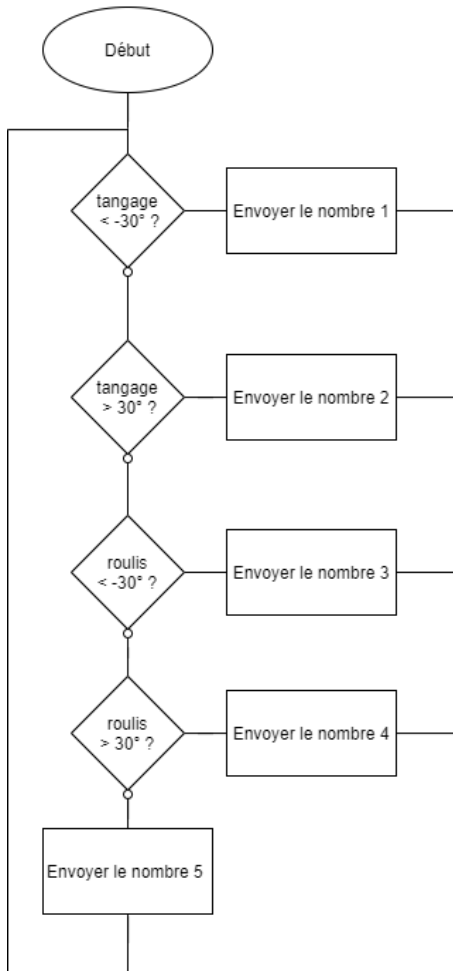
La carte micro-bit ne possédant que deux boutons, nous allons utiliser un capteur. Dans notre cas l'accéléromètre. Celui-ci peut décomposer les mouvements en deux composantes essentielles : le tangage et le roulis.

Nous allons programmer pour que le robot avance si le tangage positif est supérieur à 30° , que le robot recule si il est inférieur à -30° . De plus nous allons utiliser le roulis pour les virages à droite et à gauche avec la même valeur de 30° .



Dans un premier temps la carte émettrice enverra une des cinq valeurs possibles pour chacune des positions possibles, la carte émettrice affichera les flèches ou la croix puis dans un deuxième temps nous rajouterons le mouvement.

Mission 5-2 émetteur



Répéter indéfiniment

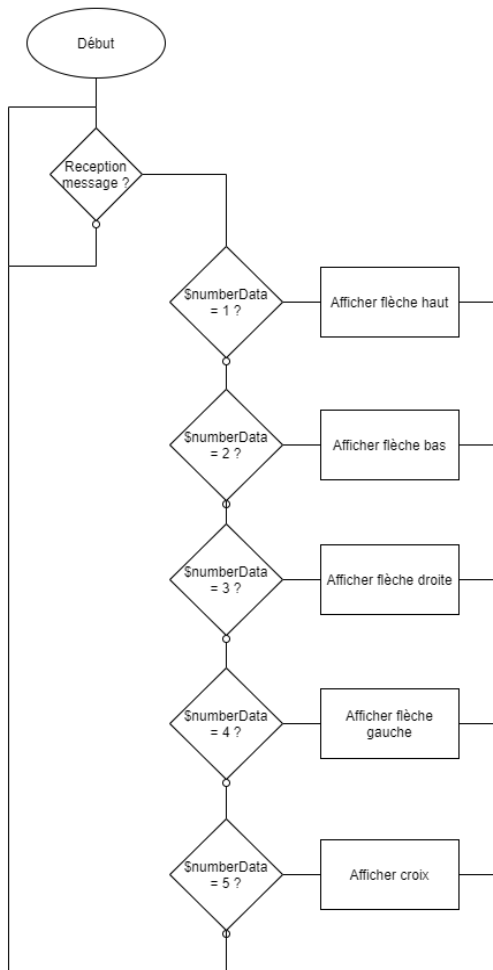


```

1 from microbit import *
2 import math
3 import radio
4
5 def radio_sendNumber(data, type='number'):
6     if type == 'number':
7         if data > int(data):
8             radio.send("&&float:" + str(data) + "&&")
9         else:
10            radio.send("&&int:" + str(data) + "&&")
11     elif type == 'boolean':
12         radio.send("&&bool:" + str(int(data)) + "&&")
13     else:
14         raise ValueError("Unable to send number: '" + str(data) + "'")
15
16 radio.on()
17
18 while True:
19     if math.atan2(accelerometer.get_y(), -accelerometer.get_z()) * 180.0/math.pi < -30:
20         radio_sendNumber(1, type='number')
21     elif math.atan2(accelerometer.get_y(), -accelerometer.get_z()) * 180.0/math.pi > 30:
22         radio_sendNumber(2, type='number')
23     elif math.atan2(accelerometer.get_x(), math.sqrt(accelerometer.get_y()**2 + accelerometer.get_z()**2)) * 180.0/math.pi < -30:
24         radio_sendNumber(3, type='number')
25     elif math.atan2(accelerometer.get_x(), math.sqrt(accelerometer.get_y()**2 + accelerometer.get_z()**2)) * 180.0/math.pi > 30:
26         radio_sendNumber(4, type='number')
27     else:
28         radio_sendNumber(5, type='number')

```

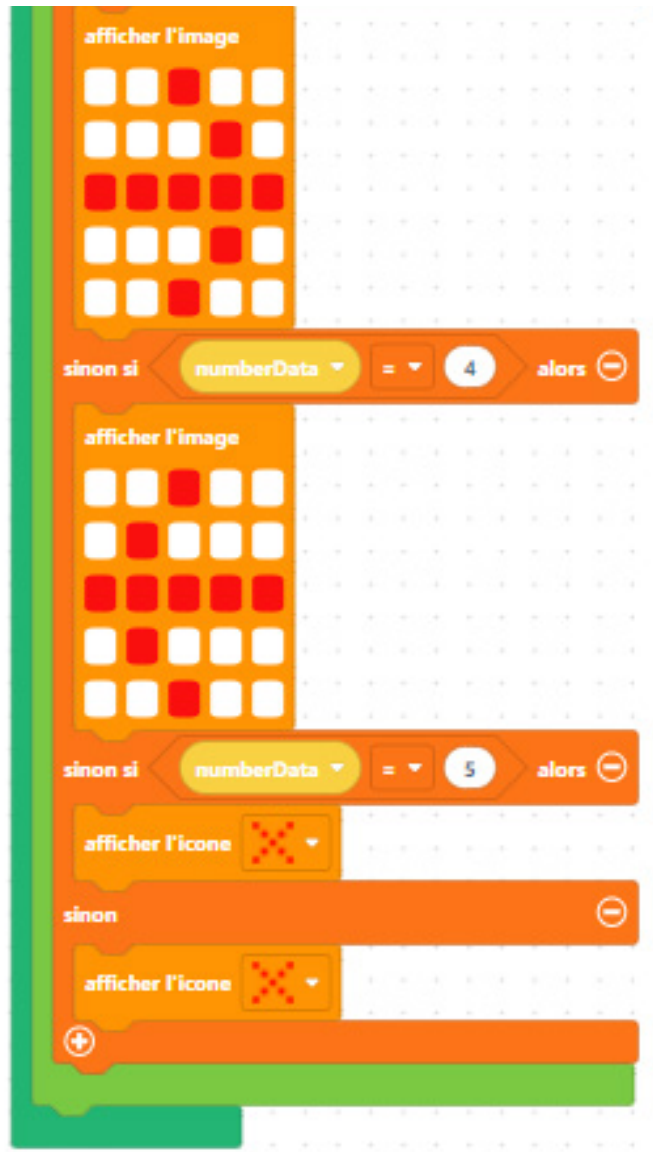
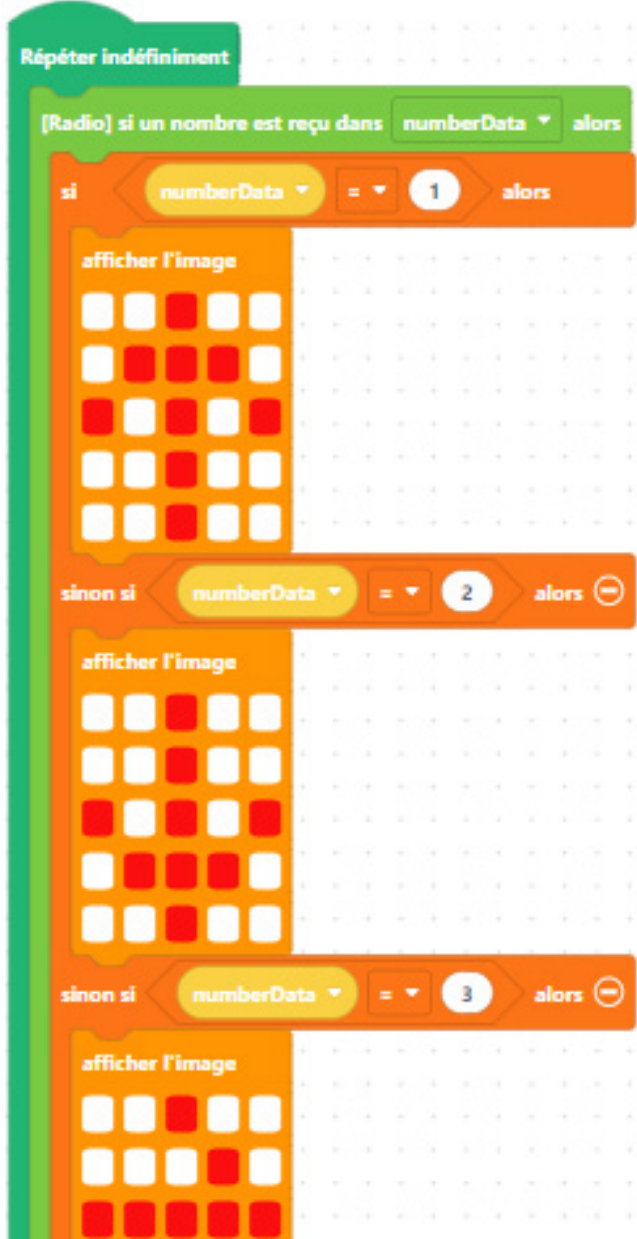
Mission 5-2 récepteur



```

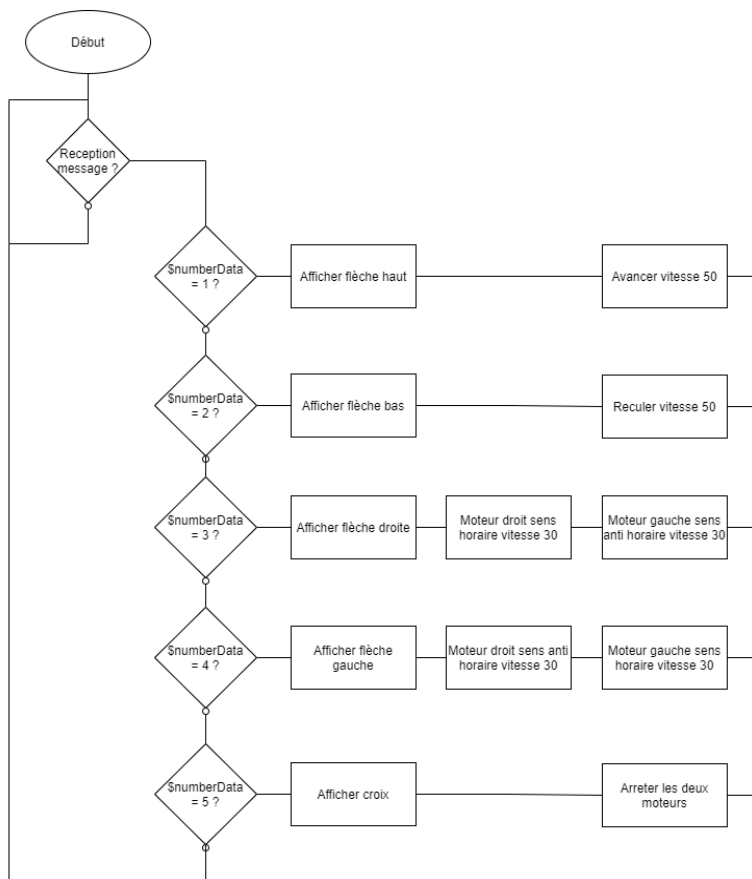
1 from microbit import *
2 import radio
3
4 def radio_receiveData():
5     data = radio.receive()
6     if data:
7         if data.find('&&int:') != -1:
8             return int(data[7:-3])
9         elif data.find('&&float:') != -1:
10            return float(data[9:-3])
11        elif data.find('&&bool:') != -1:
12            return bool(data[8:-3])
13        else:
14            return data
15    else:
16        return None
17
18 radio.on()
19
20 while True:
21     numberData = radio_receiveData()
22     if numberData is not None:
23         if numberData == 1:
24             led_image = Image('00900:09990:90909:00900:00900')
25             display.show(led_image)
26         numberData = radio_receiveData()
27         if numberData is not None:
28             if numberData == 2:
29                 led_image = Image('00900:00900:90909:09990:00900')
30                 display.show(led_image)
31             numberData = radio_receiveData()
32             if numberData is not None:
33                 if numberData == 3:
34                     led_image = Image('00900:00090:99999:00090:00900')
35                     display.show(led_image)
36             numberData = radio_receiveData()
37             if numberData is not None:
38                 if numberData == 4:
39                     led_image = Image('00900:09000:99999:09000:00900')
40                     display.show(led_image)
41             numberData = radio_receiveData()
42             if numberData is not None:
43                 if numberData == 5:
44                     display.show(Image.NO)

```



Maintenant on va faire une légère modif du programme pour tester avec le robot là où il y avait l'affichage de flèche ou de croix on va rajouter le déplacement du robot.

Mission 5-2 moteur récepteur

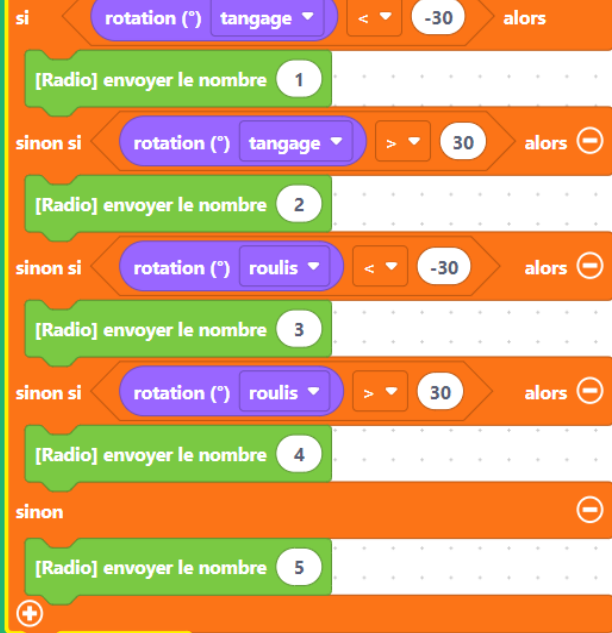


```

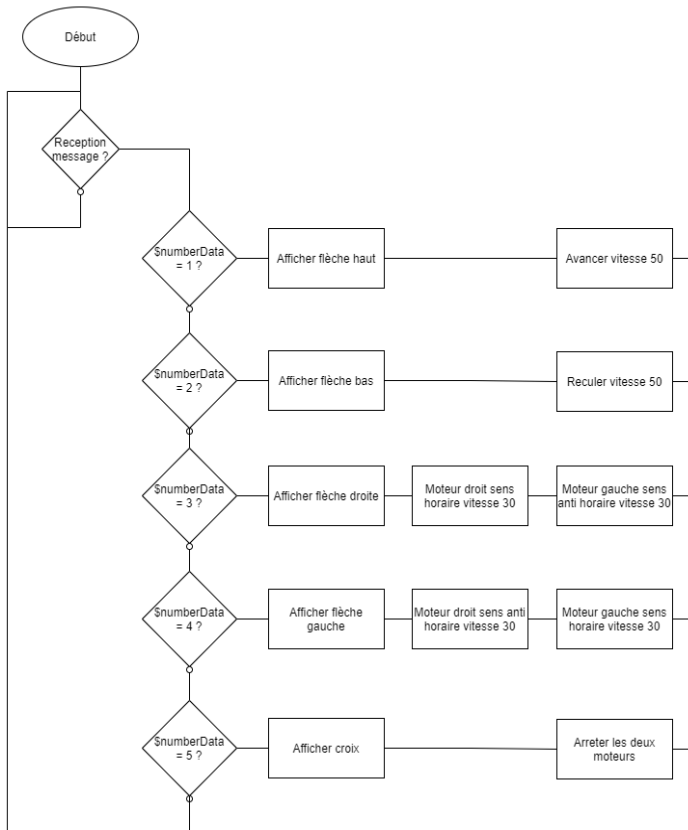
1 from microbit import *
2 import math
3 import radio
4
5 def radio_sendNumber(data, type='number'):
6     if type == 'number':
7         if data > int(data):
8             radio.send("%%f" % float(data))
9         else:
10            radio.send("%%i" % int(data))
11     elif type == 'boolean':
12         radio.send("%%b" % bool(data))
13     else:
14         raise ValueError("Unable to send number: '" + str(data) + "'")
15
16 radio.on()
17
18 while True:
19     if math.atan2(accelerometer.get_y(), -accelerometer.get_z()) * 180.0/math.pi < -30:
20         radio_sendNumber(1, type='number')
21     elif math.atan2(accelerometer.get_y(), -accelerometer.get_z()) * 180.0/math.pi > 30:
22         radio_sendNumber(2, type='number')
23     elif math.atan2(accelerometer.get_x(), math.sqrt(accelerometer.get_y()**2 + accelerometer.get_z()**2)) * 180.0/math.pi < -30:
24         radio_sendNumber(3, type='number')
25     elif math.atan2(accelerometer.get_x(), math.sqrt(accelerometer.get_y()**2 + accelerometer.get_z()**2)) * 180.0/math.pi > 30:
26         radio_sendNumber(4, type='number')
27     else:
28         radio_sendNumber(5, type='number')

```

Répéter indéfiniment



Mission 5-2 moteur récepteur



```

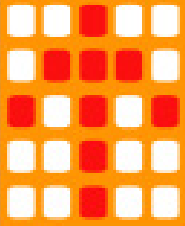
1 from microbit import *
2 import radio
3
4 def radio_receiveData():
5     data = radio.receive()
6     if data:
7         if data.find('&&int:') != -1:
8             return int(data[7:-3])
9         elif data.find('&&float:') != -1:
10            return float(data[9:-3])
11        elif data.find('&&bool:') != -1:
12            return bool(data[8:-3])
13        else:
14            return data
15    else:
16        return None
17
18 radio.on()
19
20 while True:
21     numberData = radio_receiveData()
22     if numberData is not None:
23         if numberData == 1:
24             led_image = Image('00900:09990:90909:00900:00900')
25             display.show(led_image)
26         numberData = radio_receiveData()
27         if numberData is not None:
28             if numberData == 2:
29                 led_image = Image('00900:00900:90909:09990:00900')
30                 display.show(led_image)
31             numberData = radio_receiveData()
32             if numberData is not None:
33                 if numberData == 3:
34                     led_image = Image('00900:00090:99999:00090:00900')
35                     display.show(led_image)
36             numberData = radio_receiveData()
37             if numberData is not None:
38                 if numberData == 4:
39                     led_image = Image('00900:09000:99999:09000:00900')
40                     display.show(led_image)
41             numberData = radio_receiveData()
42             if numberData is not None:
43                 if numberData == 5:
44                     display.show(Image.NO)
45 
```

Répéter indéfiniment

(Radio) si un nombre est reçu dans numberData ▾ alors

si numberData ▾ = ▾ 1 ▾ alors

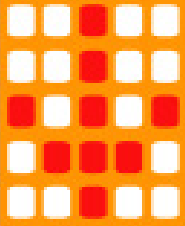
afficher l'image



(Maqueen) contrôler le robot avancer ▾ vitesse 50

sinon si numberData ▾ = ▾ 2 ▾ alors ⌚

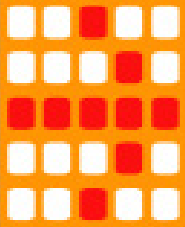
afficher l'image



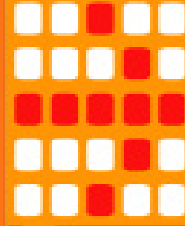
(Maqueen) contrôler le robot reculer ▾ vitesse 50

sinon si numberData ▾ = ▾ 3 ▾ alors ⌚

afficher l'image



afficher l'image

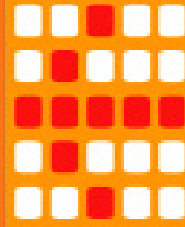


(Maqueen) contrôler le moteur droit ▾ direction ⌚ ▾ vitesse 30

(Maqueen) contrôler le moteur gauche ▾ direction ⌚ ▾ vitesse 30

sinon si numberData ▾ = ▾ 4 ▾ alors ⌚

afficher l'image



(Maqueen) contrôler le moteur droit ▾ direction ⌚ ▾ vitesse 30

(Maqueen) contrôler le moteur gauche ▾ direction ⌚ ▾ vitesse 30

sinon si numberData ▾ = ▾ 5 ▾ alors ⌚

afficher l'icone ✕ ▾

(Maqueen) arrêter le moteur droit & gauche ▾

sinon ⌚

afficher l'icone ✕ ▾

(Maqueen) arrêter le moteur droit & gauche ▾

+

• Mission 5-3 : Test en condition réel Martienne

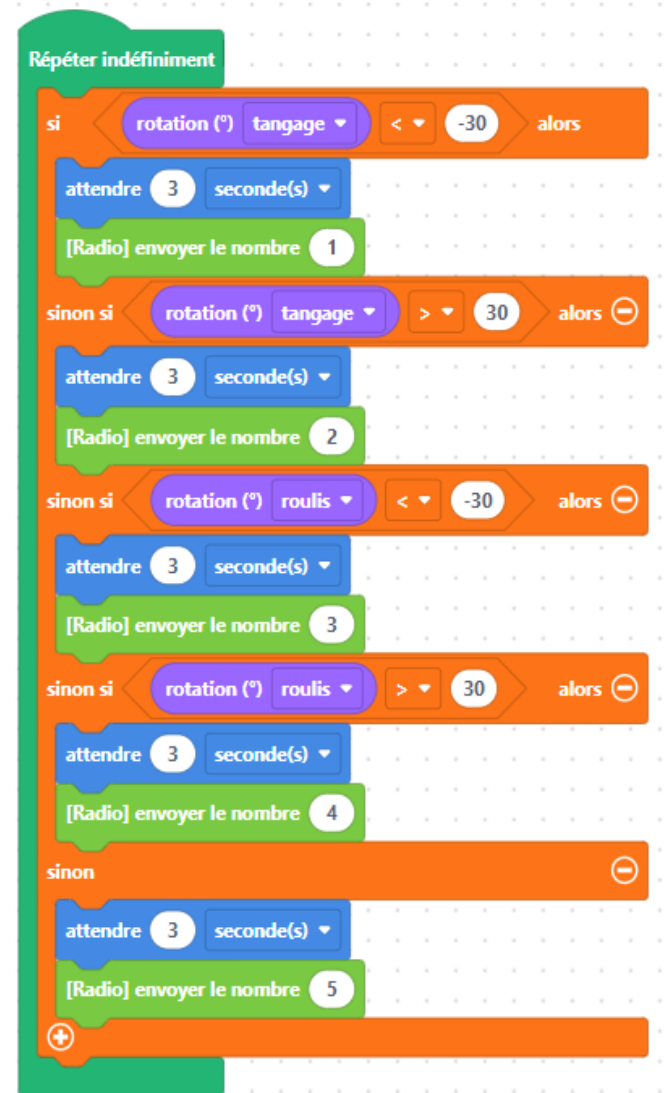
Nous savons désormais piloter notre petit robot mais sur Terre. Malheureusement les conditions de pilotage sur Mars ont une légère contrainte. Le pilote ne sera pas juste à côté pour piloter donc un décalage temporel se crée entre le moment où le pilote donne un ordre et où celui-ci est répercuté.

Pour simuler cela nous allons rajouter des petites temporisations au programme avant chaque envoi d'ordre à la carte du robot.

```

1 from microbit import *
2 import math
3 import time
4 import radio
5
6 def radio_sendNumber(data, type='number'):
7     if type == 'number':
8         if data > int(data):
9             radio.send("###float:" + str(data) + "###")
10        else:
11            radio.send("###int:" + str(data) + "###")
12    elif type == 'boolean':
13        radio.send("###bool:" + str(int(data)) + "###")
14    else:
15        raise ValueError("Unable to send number: " + str(data) + "")
16
17 radio.on()
18
19 while True:
20     if math.atan2(accelerometer.get_y(), -accelerometer.get_z()) * 180.0/math.pi < -30:
21         time.sleep(3)
22         radio_sendNumber(1, type='number')
23     elif math.atan2(accelerometer.get_y(), -accelerometer.get_z()) * 180.0/math.pi > 30:
24         time.sleep(3)
25         radio_sendNumber(2, type='number')
26     elif math.atan2(accelerometer.get_x(), math.sqrt(accelerometer.get_y()**2 + accelerometer.get_z()**2)) * 180.0/math.pi < -30:
27         time.sleep(3)
28         radio_sendNumber(3, type='number')
29     elif math.atan2(accelerometer.get_x(), math.sqrt(accelerometer.get_y()**2 + accelerometer.get_z()**2)) * 180.0/math.pi > 30:
30         time.sleep(3)
31         radio_sendNumber(4, type='number')
32     else:
33         time.sleep(3)
34         radio_sendNumber(5, type='number')

```

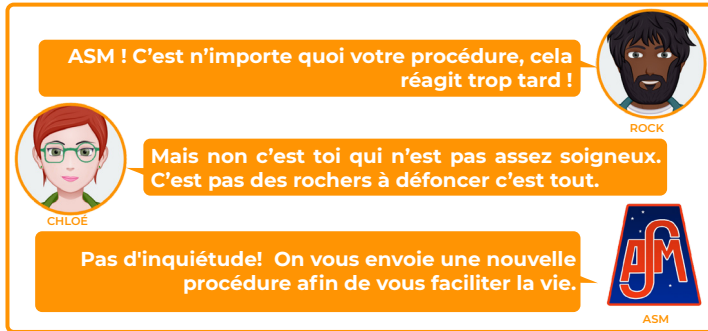


Suite à cette simulation en mode décalage temporel une solution est toute trouvée l'autonomie du robot seul face au element.

On lui donne un ordre et il va se débrouiller seul face à l'ordre global. Par exemple, allez à 100 mètres au nord et il devra grâce à sa programmation relier le point de rendez vous qui se situe 100 mètres au nord.

Mission • 6 à l'appréciation de l'encadrant

Déplacement en mode case



• Mission 6-1 : Déplacement en mode case

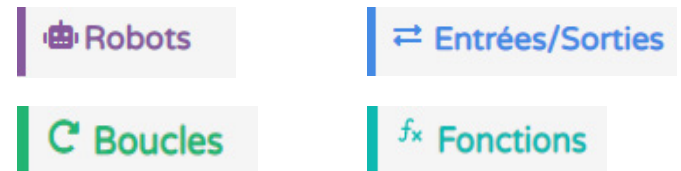
Pour cette mission nous allons commencer à programmer en mode case le robot fera de petit déplacement comme s' il était sur un échiquier ou un plateau de dames. Cela permettra de simplifier la programmation pour débiter et nous permettra de dire au robot par exemple de faire un déplacement de 3 cases en avant et de 2 sur la droite.

Pour commencer nous allons exploiter les missions précédentes avec le déplacement à 90° qui conviendra parfaitement pour un déplacement sur case et ensuite nous mesurerons combien de temps en déplacement tout droit il faut pour avancer d'une case.

Nous allons fixer la taille de la case à 10 cm cela permet de faire un quadrillage de 4 cases par 3 cases sur une feuille A3

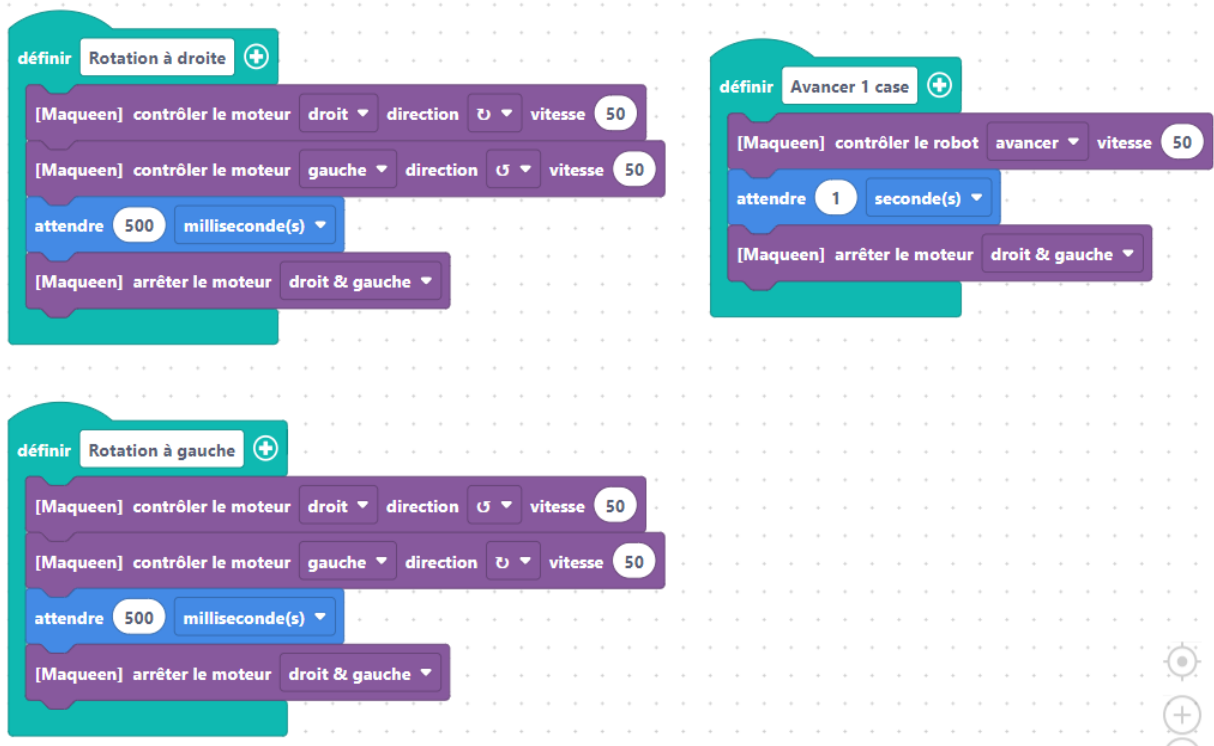
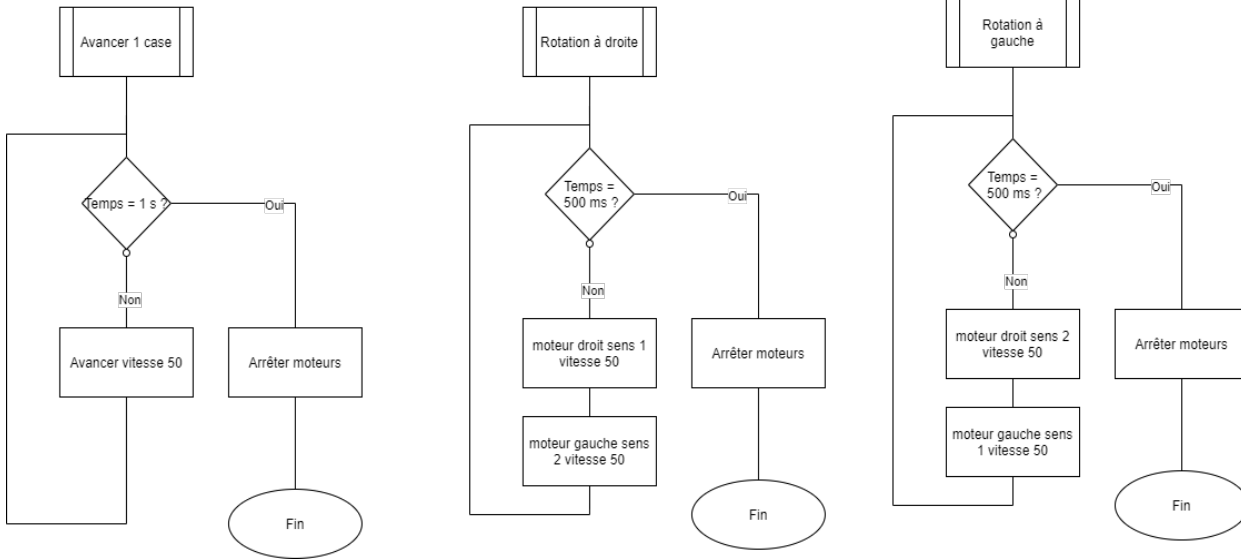
Le code de mission que nous allons utiliser est celui de la 2-2 a et de la 2-2 b.

Nous allons éviter de tout recoder ce n'est pas pertinent en informatique nous allons donc utiliser le code déjà fait et l'améliorer. Nous allons utiliser pour cela les fonctions.



Pour ce faire nous allons déjà crée trois fonction une pour avancer et une pour tourner à droite et une pour la gauche.

Il n'y a pas de changement par rapport au mission 2-2 sauf qu' au lieu de mettre les étiquettes dans la boucle "Répéter indéfiniment" ou "Au démarrage" nous allons y mettre dans un bloc fonction auquel nous donnerons un nom reconnaissable afin de ne pas se perdre dans le



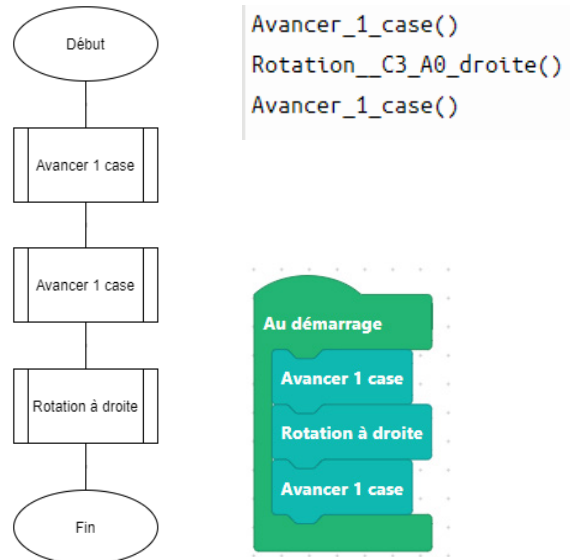
```

1 from microbit import *
2 import time
3
4 def Rotation__C3_A0_droite():
5     i2c.write(0x10, bytearray([0x02, 0x0, 50]))
6     i2c.write(0x10, bytearray([0x00, 0x1, 50]))
7     time.sleep_ms(500)
8     i2c.write(0x10, bytearray([0x00, 0, 0]))
9     i2c.write(0x10, bytearray([0x02, 0, 0]))
10
11 def Avancer_1_case():
12     i2c.write(0x10, bytearray([0x00, 0x0, 50]))
13     i2c.write(0x10, bytearray([0x02, 0x0, 50]))
14     time.sleep(1)
15     i2c.write(0x10, bytearray([0x00, 0, 0]))
16     i2c.write(0x10, bytearray([0x02, 0, 0]))
17
18 def Rotation__C3_A0_gauche():
19     i2c.write(0x10, bytearray([0x02, 0x1, 50]))
20     i2c.write(0x10, bytearray([0x00, 0x0, 50]))
21     time.sleep_ms(500)
22     i2c.write(0x10, bytearray([0x00, 0, 0]))
23     i2c.write(0x10, bytearray([0x02, 0, 0]))

```

Maintenant que nous avons nos fonctions pour programmer cela devient plus simple il suffit juste d'appeler la fonction dans le programme principal pour que celle ci s'exécute en entier .

Voici le reste du code à ajouter au précédent qui va avec pour par exemple avancer d'une case et ensuite tourner sur la droite et avancer de nouveau d'une case.



Comme on peut le constater cela simplifie grandement le programme, plus besoin de faire des lignes et des lignes de code avec le principe des fonctions.

• Mission 6-2 : Détection obstacle avant déplacement

Dans cette mission nous allons rajouter les éléments de la mission 3 en effet se déplacer sans regarder ou l'on va. Au bout d'un moment cela risque de poser des problèmes dans la vie de tous les jours et pour les humains. Les yeux sont les organes principaux de détection d'obstacle. En effet, notre vue en 3 dimensions nous permet de repérer les obstacles et d'évaluer leur distance. Dans notre cas, ce sera le capteur ultrason qui jouera ce rôle.

L'objectif de cette mission sera de se déplacer sur le damier mais en vérifiant avant chaque déplacement si la case ou l'on doit se déplacer est libre. Et si ce n'est pas le cas ne pas avancer sur la case encombrée et afficher une croix sur la carte.

Dans le code python je n'ai pas mis la fonction relative au capteur ultrason c'est le même que dans la mission 3. Nous voilà déjà bien avancer notre robot est maintenant capable de se déplacer seul en évitant les obstacles ou plutôt en les ignorant c'est déjà un bon début.

ASM ! Super la procédure, cela fonctionne presque bien. Le souci est lorsque l'on rentre dans les couleurs de la base le capteur s'affole et cela devient impossible à piloter.



ROCK



CHLOÉ

On pourrait pas suivre les lignes au sol.

Bien joué, Chloé c'est l'idée. On vous envoie les procédures pour régler les problèmes.



ASM

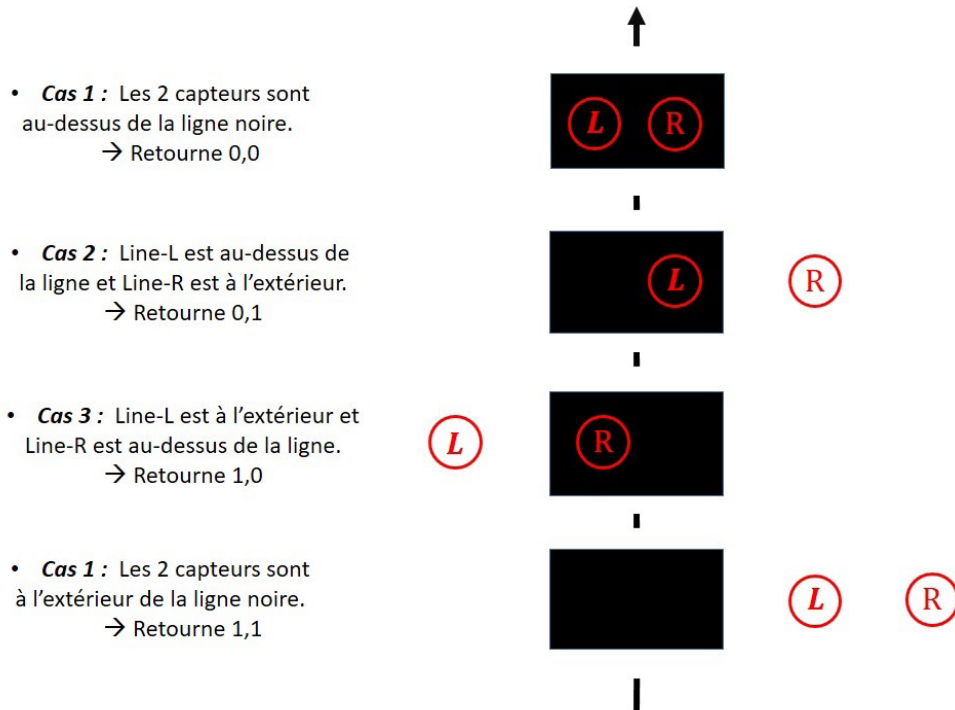
Mission • 7 à l'appréciation de l'encadrant

Suiveur de ligne

• Mission 7-1 : Suivre une ligne

Le suiveur de ligne intégré au robot Maqueen est composé de 2 capteurs à infrarouge situé sous le châssis du robot.

Le principe de fonctionnement est le suivant : Le suiveur de ligne est composé de 2 capteurs à infrarouge Line-R (pin14) et Line-L (pin13) ayant chacun une LED émettrice infrarouge et un photo-transistor sensible. La LED émet un signal et le photo-transistor le détecte. Le noir absorbe le rayonnement infrarouge, le récepteur ne détecte pas le signal émis lorsque le capteur est au-dessus de la ligne, celui-ci retourne donc 0. Dans le cas contraire, le signal est réfléchi par le blanc, le capteur retourne alors 1.



Ici nous pouvons voir le fonctionnement des capteurs et l'information que l'on récupère au niveau de la carte microcontrôleur.

Grâce à ces informations, nous allons pouvoir réfléchir à ce que fait chaque moteur en fonction des informations des capteurs.

Pour faciliter la tâche, nous allons utiliser un tableau logique dans lequel il y aura deux colonnes pour les capteurs et deux colonnes pour les moteurs. Nous utiliserons le code 0 pour moteur éteint ou couleur blanche et 1 pour moteur actif et couleur noire.

Capteur gauche	Capteur droite	Moteur gauche	Moteur droite
0 / blanc	0 / blanc	?	?
0 / blanc	1 / noir	1 / activer	0 / éteint
1 / noir	0 / blanc	0 / éteint	1 / activer
1 / noir	1 / noir	1 / activer	1 / activer

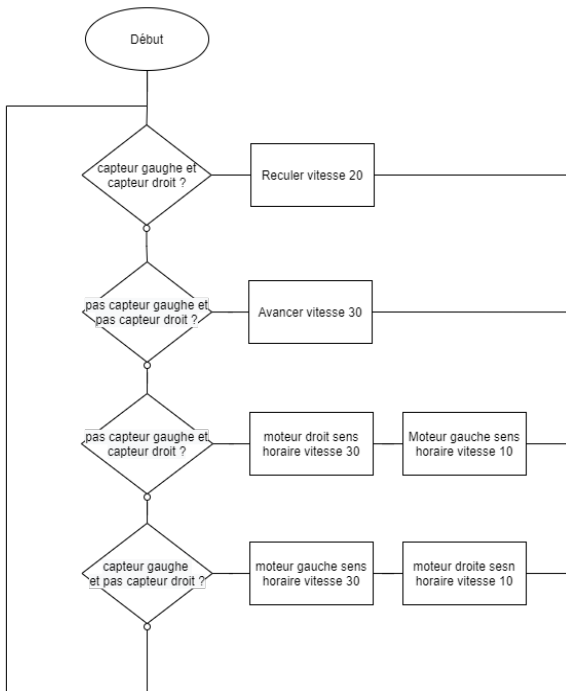
On se rend compte que dans le premier cas, il n'y a plus de ligne et dans ce cas il faut mettre en place une stratégie de recherche de ligne. Cela peut être par exemple « avancer de 5 cm puis tourner en rond jusqu'à détection de ligne » ou alors simplement « reculer » car on a été trop vite et perdu la ligne.

Essayons de mettre cela en place.

Pour cela nous aurons besoin des blocs suivantes :



Dans le programme, on voit au départ dans le bloc "Au démarrage" un "arrêter le moteur". C'est une mini initialisation, cela permet d'être sûr que l'on commence moteur éteint. Dans cette initialisation, on peut rajouter de nombreuses choses pour être sûr de commencer le programme dans de bonnes conditions; par exemple un « effacer l'écran » pour être sûr que rien ne reste sur l'écran. On peut y ajouter une pause pour avoir le temps de bien positionner le robot avant que celui-ci démarre.



```

1 from microbit import *
2
3 i2c.write(0x10, bytearray([0x00, 0, 0]))
4 i2c.write(0x10, bytearray([0x02, 0, 0]))
5
6 while True:
7     if pin13.read_digital() and pin14.read_digital():
8         i2c.write(0x10, bytearray([0x00, 0x1, 20]))
9         i2c.write(0x10, bytearray([0x02, 0x1, 20]))
10    if not pin13.read_digital() and not pin14.read_digital():
11        i2c.write(0x10, bytearray([0x00, 0x0, 30]))
12        i2c.write(0x10, bytearray([0x02, 0x0, 30]))
13    if not pin13.read_digital() and pin14.read_digital():
14        i2c.write(0x10, bytearray([0x02, 0x0, 30]))
15        i2c.write(0x10, bytearray([0x00, 0x0, 10]))
16    if pin13.read_digital() and not pin14.read_digital():
17        i2c.write(0x10, bytearray([0x00, 0x0, 30]))
18        i2c.write(0x10, bytearray([0x02, 0x0, 10]))

```

Au démarrage

[Maqueen] arrêter le moteur droit & gauche ▾

Répéter indéfiniment

```

si [Maqueen] état du capteur de ligne gauche ▾ et ▾ [Maqueen] état du capteur de ligne droit ▾ alors
  [Maqueen] contrôler le robot reculer ▾ vitesse 20
+
si pas [Maqueen] état du capteur de ligne gauche ▾ et ▾ pas [Maqueen] état du capteur de ligne droit ▾ alors
  [Maqueen] contrôler le robot avancer ▾ vitesse 30
+
si pas [Maqueen] état du capteur de ligne gauche ▾ et ▾ [Maqueen] état du capteur de ligne droit ▾ alors
  [Maqueen] contrôler le moteur droit ▾ direction U ▾ vitesse 30
  [Maqueen] contrôler le moteur gauche ▾ direction U ▾ vitesse 10
+
si [Maqueen] état du capteur de ligne gauche ▾ et ▾ pas [Maqueen] état du capteur de ligne droit ▾ alors
  [Maqueen] contrôler le moteur gauche ▾ direction U ▾ vitesse 30
  [Maqueen] contrôler le moteur droit ▾ direction U ▾ vitesse 10
+

```

• Mission 7-2 : Suivre une ligne et éviter les obstacles

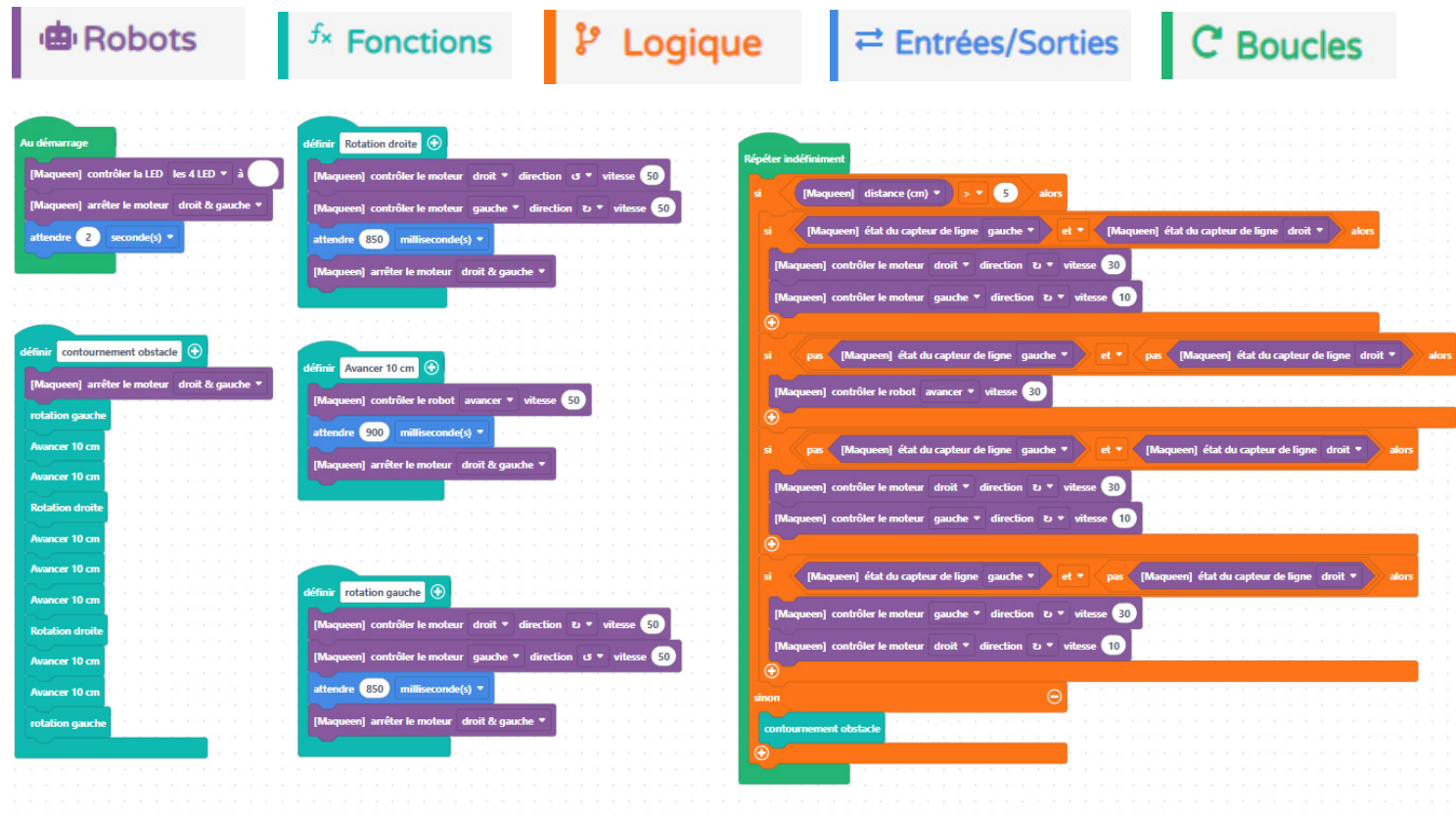
Pour cette mission nous allons imaginer que le robot détecte un objet alors qu'il suit sa ligne. Dans ce cas de figure, il devra arrêter de suivre la ligne. Contourner l'objet puis retrouver la ligne de l'autre côté de l'obstacle.

Dans notre cas, nous limiterons la taille de l'objet gênant à 10 cm de diamètre.

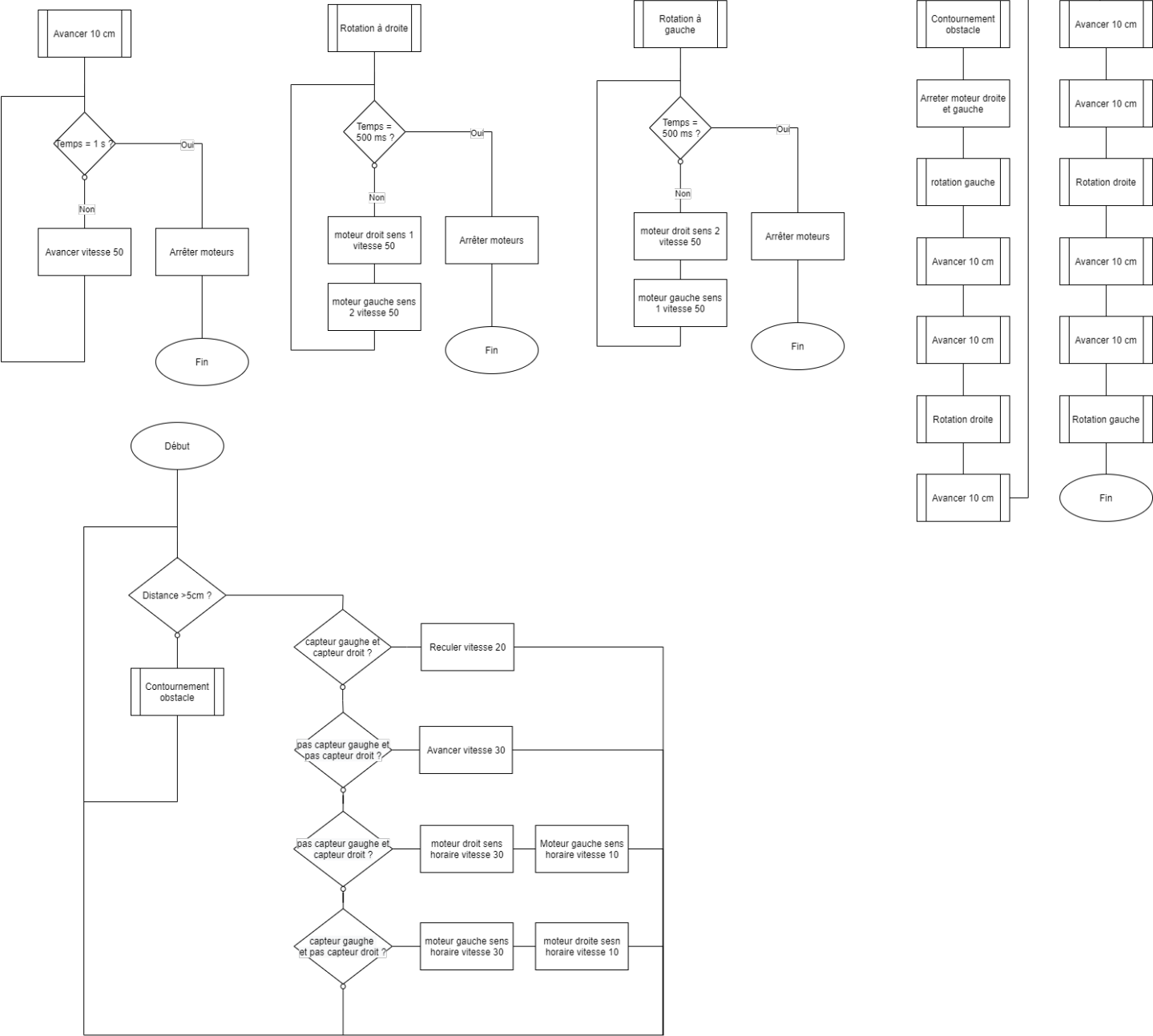
Nous allons utiliser le travail précédent de la mission 6-1 où nous avons utilisé les fonctions. On garde les fonctions "Avancer 10 cm" et "Tourner à droite" ou à gauche de 90°. On rajoute juste une fonction contournement d'obstacle. Et nous améliorerons notre initialisation avec une pause et de la lumière sous le robot pour éviter les problèmes dus à l'environnement lumineux.

Essayons de mettre cela en place.

Pour cela nous aurons besoin des étiquettes suivantes :



Mission 7-1



```

1 from microbit import *
2 import neopixel
3 import time
4 from machine import time_pulse_us
5 import utime
6
7 npMq = neopixel.NeoPixel(pin15, 4)
8
9 def getUltrasonicData(trig, echo, data='distance', timeout_us=30000):
10     trig.write_digital(0)
11     utime.sleep_us(2)
12     trig.write_digital(1)
13     utime.sleep_us(10)
14     trig.write_digital(0)
15     echo.read_digital()
16     duration = time_pulse_us(echo, 1, timeout_us)/1e6 # t_echo in seconds
17     if data == 'distance':
18         #sound speed, round-trip/2, get in cm
19         return 343 * duration/2 * 100
20     elif data == 'duration':
21         return duration
22     else:
23         raise ValueError("Data option '" + data + "' is not valid")
24
25 def Rotation_droite():
26     i2c.write(0x10, bytearray([0x02, 0x1, 50]))
27     i2c.write(0x10, bytearray([0x00, 0x0, 50]))
28     time.sleep_ms(850)
29     i2c.write(0x10, bytearray([0x00, 0, 0]))
30     i2c.write(0x10, bytearray([0x02, 0, 0]))

```

Alors les algorithmes et le programme deviennent plus conséquent mais on retrouve de nombreuses partie d'anciens programme de même pour les algorithmes très de peu de changement juste du complément.

```

32 def contournement_obstacle():
33     i2c.write(0x10, bytearray([0x00, 0, 0]))
34     i2c.write(0x10, bytearray([0x02, 0, 0]))
35     rotation_gauche()
36     Avancer_10_cm()
37     Avancer_10_cm()
38     Rotation_droite()
39     Avancer_10_cm()
40     Avancer_10_cm()
41     Avancer_10_cm()
42     Rotation_droite()
43     Avancer_10_cm()
44     Avancer_10_cm()
45     rotation_gauche()
46
47 def Avancer_10_cm():
48     i2c.write(0x10, bytearray([0x00, 0x0, 50]))
49     i2c.write(0x10, bytearray([0x02, 0x0, 50]))
50     time.sleep_ms(900)
51     i2c.write(0x10, bytearray([0x00, 0, 0]))
52     i2c.write(0x10, bytearray([0x02, 0, 0]))
53
54 def rotation_gauche():
55     i2c.write(0x10, bytearray([0x02, 0x0, 50]))
56     i2c.write(0x10, bytearray([0x00, 0x1, 50]))
57     time.sleep_ms(850)
58     i2c.write(0x10, bytearray([0x00, 0, 0]))
59     i2c.write(0x10, bytearray([0x02, 0, 0]))
60
61 for i in range(4):
62     npMq[i] = (255, 255, 255)
63     npMq.show()
64     i2c.write(0x10, bytearray([0x00, 0, 0]))
65     i2c.write(0x10, bytearray([0x02, 0, 0]))
66     time.sleep(2)
67
68 while True:
69     if getUltrasonicData(pin1, pin2, 'distance') > 5:
70         if pin13.read_digital() and pin14.read_digital():
71             i2c.write(0x10, bytearray([0x02, 0x0, 30]))
72             i2c.write(0x10, bytearray([0x00, 0x0, 10]))
73         if not pin13.read_digital() and not pin14.read_digital():
74             i2c.write(0x10, bytearray([0x00, 0x0, 30]))
75             i2c.write(0x10, bytearray([0x02, 0x0, 30]))
76         if not pin13.read_digital() and pin14.read_digital():
77             i2c.write(0x10, bytearray([0x02, 0x0, 30]))
78             i2c.write(0x10, bytearray([0x00, 0x0, 10]))
79         if pin13.read_digital() and not pin14.read_digital():
80             i2c.write(0x10, bytearray([0x00, 0x0, 30]))
81             i2c.write(0x10, bytearray([0x02, 0x0, 10]))
82     else:
83         contournement_obstacle()
84

```

Mission • 8

🌀 à l'appréciation de l'encadrant

Déplacer un objet

Chloé, Rock ! Il est temps de ramener les échantillons et les robots à la base. La procédure suit. Mais ne vous inquiétez pas vous connaissez déjà tout .

ASM



C'est partie enfin !

Bien reçu !



CHLOÉ

L'objectif de cette mission sera d'attraper et de relâcher après 5 secondes un objet directement posé devant le robot. Pour cela nous allons utiliser le savoir accumulé dans la mission 2.

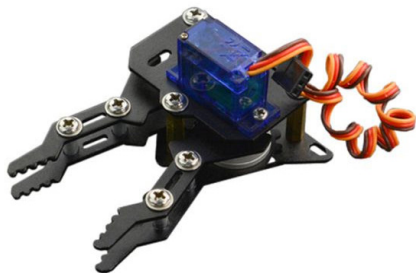
Essayons de mettre cela en place.
Pour cela nous aurons besoins des étiquettes suivante :

🤖 Robots

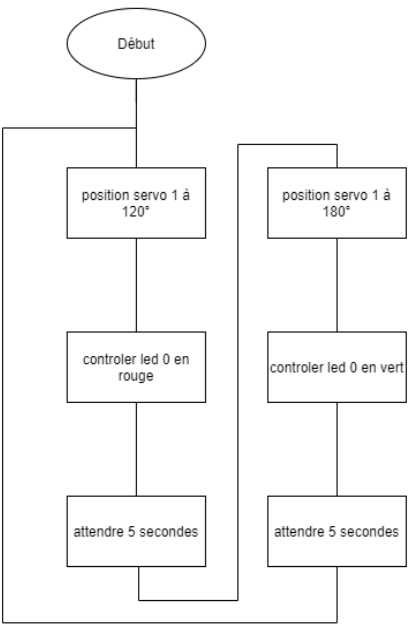
↔ Entrées/Sorties

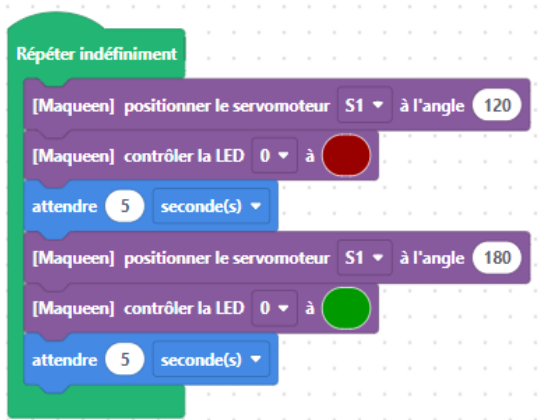
🔄 Boucles

Nous allons démonter le capteur à ultrason et le remplacer par la pince afin de saisir un objet.



⚠ Attention : Pour le montage de la pince je conseille de positionner, avec un programme le servo, en position 180° puis de fixer la pince en position fermée. Cela permet un débattement utilisable de 120° à 180° dans les programmes.





```

1 from microbit import *
2 import neopixel
3 import time
4
5 npMaa = neopixel.NeoPixel(pin15, 4)
6
7 while True:
8     i2c.write(0x10, bytearray([0x14, 120]))
9     npMaa[0] = (153, 0, 0)
10    npMaa.show()
11    time.sleep(5)
12    i2c.write(0x10, bytearray([0x14, 180]))
13    npMaa[0] = (0, 153, 0)
14    npMaa.show()

```

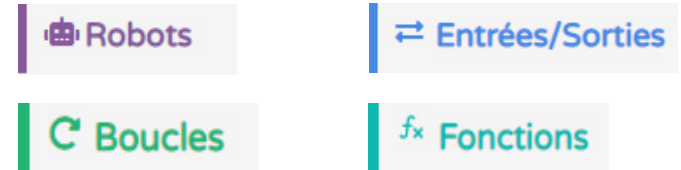
Pour le plaisir je me suis permis de rajouter des led de couleur qui indique quand la pince s'ouvre et quand elle se ferme.

• Mission 6-1 : Déplacer un objet d'un endroit à l'autre

L'objectif de cette mission sera d'attraper un objet directement posé devant le robot et de le déposer ensuite au même endroit mais par un autre côté. Pour cela nous allons utiliser les programmes utilisés dans la mission 8-1 et 7-1.

Essayons de mettre cela en place.

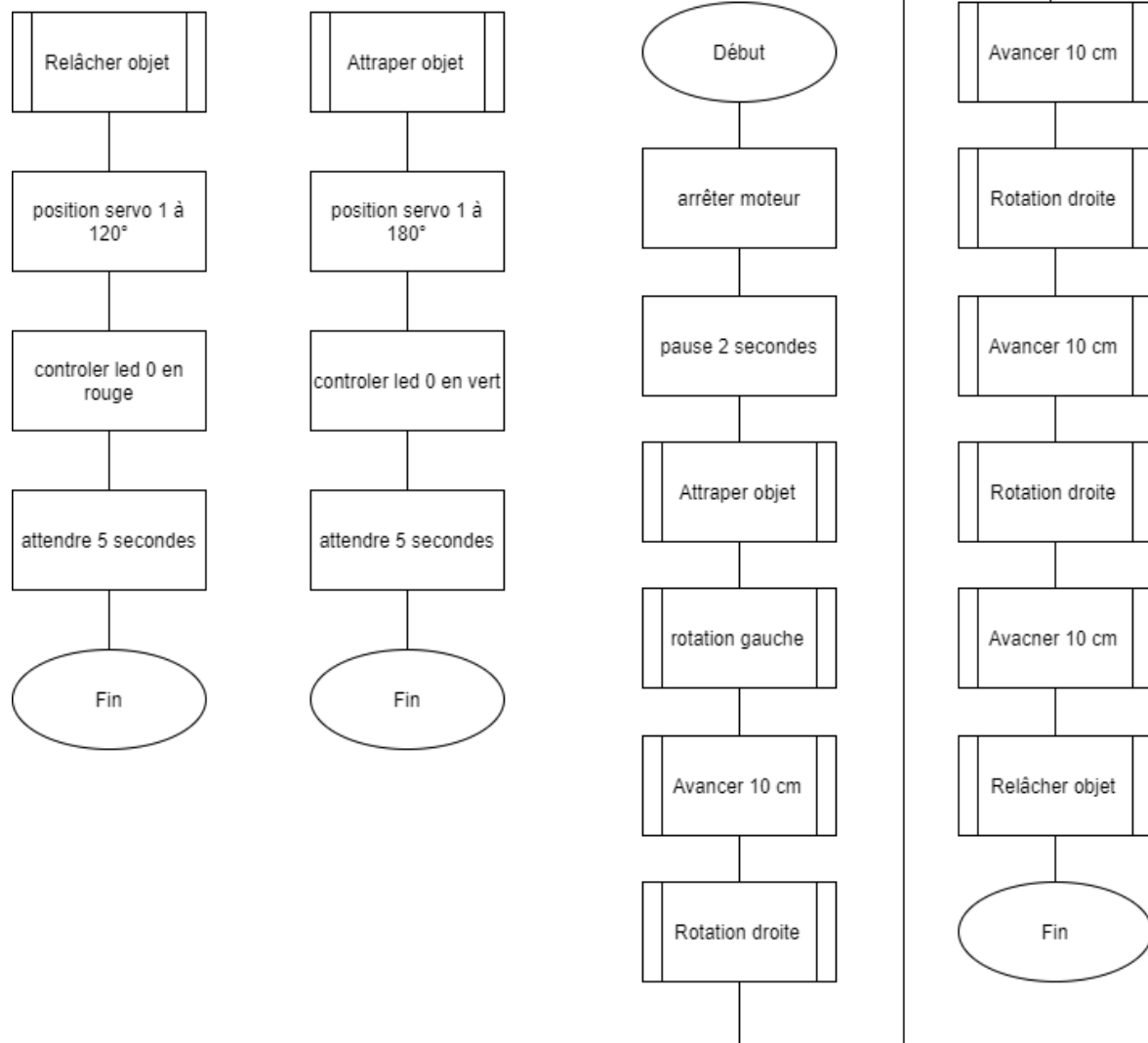
Pour cela nous aurons besoin des étiquettes suivantes :



Nous maîtrisons maintenant beaucoup mieux les fonctions donc nous allons décomposer l'ensemble du programme en fonction :

- Attraper objet
- Relâcher objet
- Rotation droite
- Rotation gauche
- Avancer 10 cm

Pour les deux premières, nous allons couper le programme précédent en deux. Pour les suivantes nous les avons déjà faites donc pas de soucis. Et enfin il suffira de faire un programme avec uniquement les fonctions.



Au démarrage

[Maqueen] arrêter le moteur droit & gauche ▾

attendre 2 seconde(s) ▾

attraper objet

rotation gauche

Avancer 10 cm

Rotation droite

Avancer 10 cm

Rotation droite

Avancer 10 cm

Rotation droite

Avancer 10 cm

relacher objet

définir relacher objet +

[Maqueen] positionner le servomoteur S1 ▾ à l'angle 120

[Maqueen] contrôler la LED 0 ▾ à 

attendre 2 seconde(s) ▾

définir attraper objet +

[Maqueen] positionner le servomoteur S1 ▾ à l'angle 180

[Maqueen] contrôler la LED 0 ▾ à 

attendre 2 seconde(s) ▾

```

1 from microbit import *
2 import time
3 import neopixel
4
5 npMaq = neopixel.NeoPixel(pin15, 4)
6
7 def attraper_objet():
8     i2c.write(0x10, bytearray([0x14, 180]))
9     npMaq[0] = (0, 153, 0)
10    npMaq.show()
11    time.sleep(2)
12
13 def Rotation_droite():
14     i2c.write(0x10, bytearray([0x02, 0x1, 50]))
15     i2c.write(0x10, bytearray([0x00, 0x0, 50]))
16     time.sleep_ms(750)
17     i2c.write(0x10, bytearray([0x00, 0, 0]))
18     i2c.write(0x10, bytearray([0x02, 0, 0]))
19
20 def rotation_gauche():
21     i2c.write(0x10, bytearray([0x02, 0x0, 50]))
22     i2c.write(0x10, bytearray([0x00, 0x1, 50]))
23     time.sleep_ms(750)
24     i2c.write(0x10, bytearray([0x00, 0, 0]))
25     i2c.write(0x10, bytearray([0x02, 0, 0]))
26
27 def relacher_objet():
28     i2c.write(0x10, bytearray([0x14, 120]))
29     npMaq[0] = (153, 0, 0)
30     npMaq.show()
31     time.sleep(2)
32
33 def Avancer_10_cm():
34     i2c.write(0x10, bytearray([0x00, 0x0, 50]))
35     i2c.write(0x10, bytearray([0x02, 0x0, 50]))
36     time.sleep_ms(900)
37     i2c.write(0x10, bytearray([0x00, 0, 0]))
38     i2c.write(0x10, bytearray([0x02, 0, 0]))
39
40 i2c.write(0x10, bytearray([0x00, 0, 0]))
41 i2c.write(0x10, bytearray([0x02, 0, 0]))
42 time.sleep(2)
43 attraper_objet()
44 rotation_gauche()
45 Avancer_10_cm()
46 Rotation_droite()
47 Avancer_10_cm()
48 Rotation_droite()
49 Avancer_10_cm()
50 Rotation_droite()
51 Avancer_10_cm()
52 relacher_objet()

```

Mission • 9

Mise en situation complexe

🕒 à l'appréciation de l'encadrant

ASM ! Tous les robots sont rentrés ! On peut retourner à nos missions de bases



ROCK



ASM

Presque Rock. Il faut maintenant relancer la procédure automatique. Et surtout surveillé en cas de problème. Vous allez devoir tenir des quarts pour rattraper le temps perdu.

Bien reçu ASM!

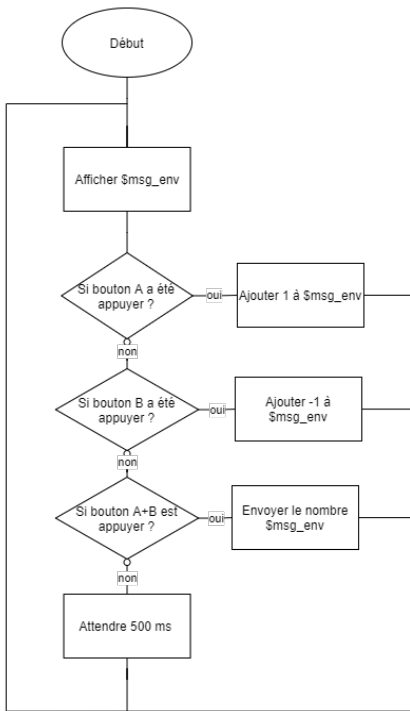
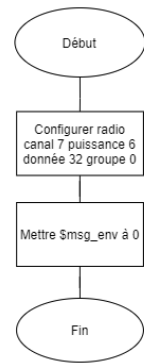


CHLOE

• Mission 9-1 : Enfin sur Mars

Mise en situation complexe : Le robot suit la boucle en permanence mais une chute des températures brutales arrive. Le robot arrête toutes activités et doit se mettre à l'abri. Pour cela on passe en commande manuel par déplacement de grille. Et retour au centre du tapis. Les déplacements seront sécurisés en utilisant les capteurs à ultrasons pour vérifier si un déplacement est valide. Le retour d'information se fera sur la carte de pilotage à distance. Il faudra prévoir des déplacements de plusieurs natures par tranche de 5, 10 ou 20 cm. de même pour les angles il faudra prévoir différents angles afin de réussir à le déplacer. Attention les commandes du micro:bit sont limitées dans les entrées avec seulement 2 boutons et quelques gestes.

Mission 9 Emetteur



Au démarrage

[Radio] configurer Canal 7 Puissance 6 Taille des données 32 Groupe 0

mettre msg env à 0

Répéter indéfiniment

afficher msg env

si bouton A a été appuyé alors

ajouter 1 à msg env

si bouton B a été appuyé alors

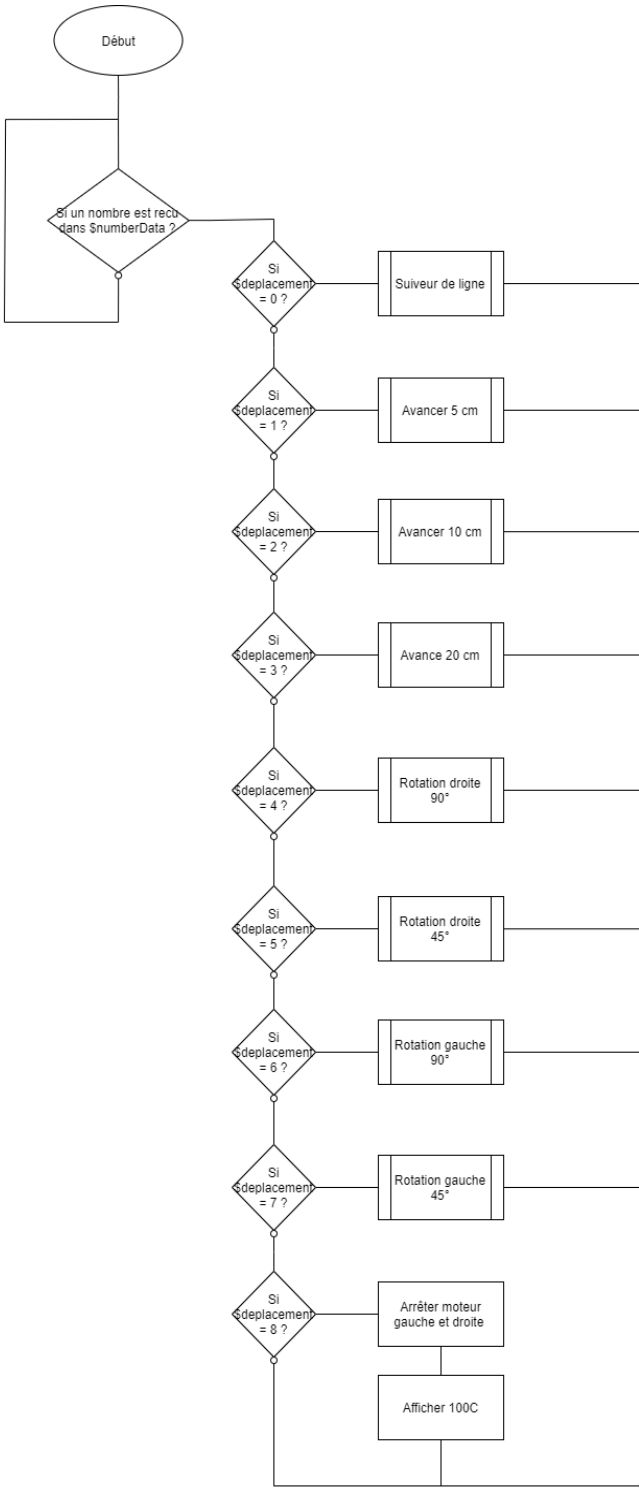
ajouter -1 à msg env

si bouton A+B est appuyé alors

[Radio] envoyer le nombre msg env

attendre 500 milliseconde(s)

```
1 from microbit import *
2 import radio
3 import time
4
5 def radio_sendNumber(data, type='number'):
6     if type == 'number':
7         if data > int(data):
8             radio.send("%%&float:" + str(data) + "%%&")
9         else:
10            radio.send("%%&int:" + str(data) + "%%&")
11     elif type == 'boolean':
12         radio.send("%%&bool:" + str(int(data)) + "%%&")
13     else:
14         raise ValueError("Unable to send number: " + str(data) + "!")
15
16 radio.on()
17
18 radio.config(channel = 7, power = 6, length = 32, group=0)
19 msg_env = 0
20
21 while True:
22     display.show(msg_env)
23     if button_a.was_pressed():
24         msg_env = msg_env + 1
25     if button_b.was_pressed():
26         msg_env = msg_env + -1
27     if button_a.is_pressed() and button_b.is_pressed():
28         radio_sendNumber(msg_env, type='number')
29     time.sleep_ms(500)
```



Au démarrage

[Radio] configurer Canal 7 Puissance 6 Taille des données 32 Groupe 0

mettre msg env à 0

Répéter indéfiniment

afficher msg env

si bouton A a été appuyé alors

ajouter 1 à msg env

si bouton B a été appuyé alors

ajouter -1 à msg env

si bouton A+B est appuyé alors

[Radio] envoyer le nombre msg env

attendre 500 milliseconde(s)

définir Avancer 5 cm

si [Maqueen] distance (cm) > 5 alors

[Maqueen] contrôler le robot avancer vitesse 50

attendre 450 milliseconde(s)

[Maqueen] arrêter le moteur droit & gauche

sinon

afficher l'icone

[Maqueen] arrêter le moteur droit & gauche

définir Rotation droite 90

[Maqueen] contrôler le moteur droit direction vitesse 50

[Maqueen] contrôler le moteur gauche direction vitesse 50

attendre 750 milliseconde(s)

[Maqueen] arrêter le moteur droit & gauche

définir Rotation droite 45

[Maqueen] contrôler le moteur droit direction vitesse 50

[Maqueen] contrôler le moteur gauche direction vitesse 50

attendre 375 milliseconde(s)

[Maqueen] arrêter le moteur droit & gauche

définir Avancer 10 cm

si [Maqueen] distance (cm) > 10 alors

[Maqueen] contrôler le robot avancer vitesse 50

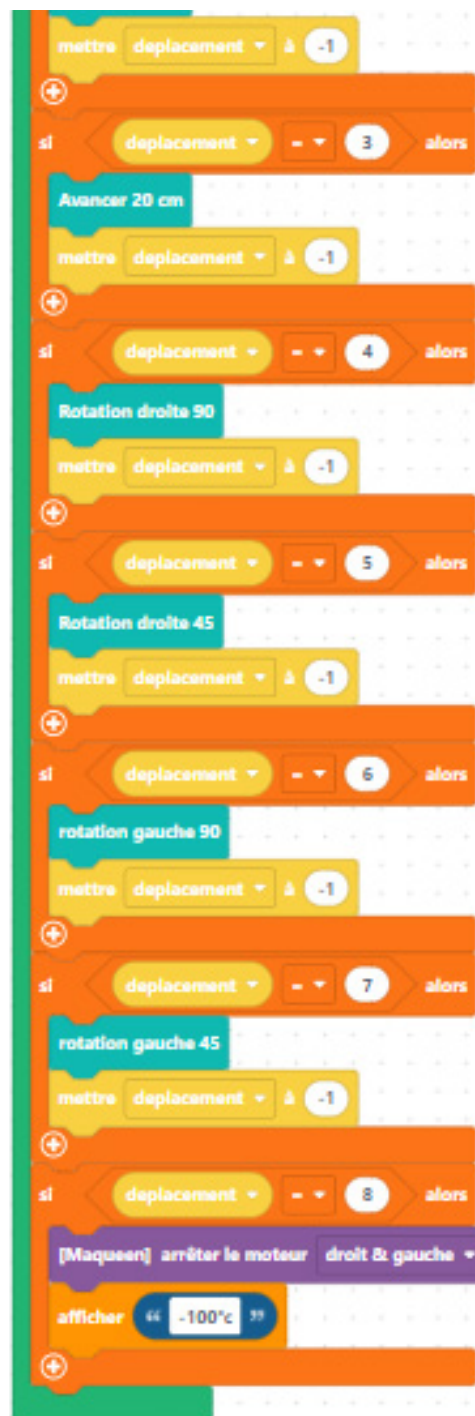
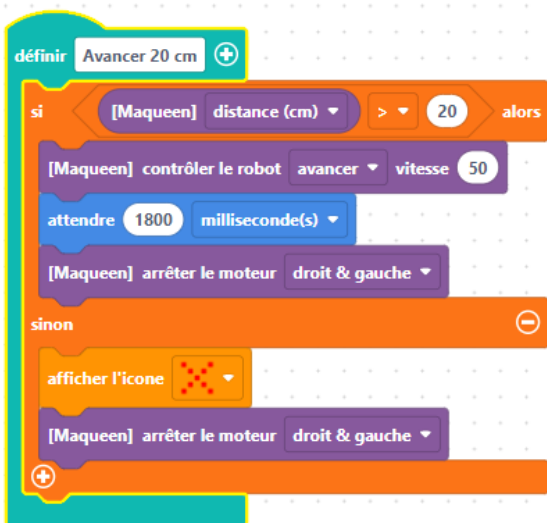
attendre 900 milliseconde(s)

[Maqueen] arrêter le moteur droit & gauche

sinon

afficher l'icone

[Maqueen] arrêter le moteur droit & gauche



```

1 from microbit import *
2 import radio
3 from machine import time_pulse_us
4 import utime
5 import time
6
7 def radio_receiveData():
8     data = radio.receive()
9     if data:
10         if data.find('&&int:') != -1:
11             return int(data[7:-3])
12         elif data.find('&&float:') != -1:
13             return float(data[9:-3])
14         elif data.find('&&bool:') != -1:
15             return bool(data[8:-3])
16         else:
17             return data
18     else:
19         return None
20
21 def getUltrasonicData(trig, echo, data='distance', timeout_us=30000):
22     trig.write_digital(0)
23     utime.sleep_us(2)
24     trig.write_digital(1)
25     utime.sleep_us(10)
26     trig.write_digital(0)
27     echo.read_digital()
28     duration = time_pulse_us(echo, 1, timeout_us)/1e6 # t_echo in seconds
29     if data == 'distance':
30         #sound speed, round-trip/2, get in cm
31         return 343 * duration/2 * 100
32     elif data == 'duration':
33         return duration
34     else:
35         raise ValueError("Data option '" + data + "' is not valid")
36
37 radio.on()
38

```

```

39 def suiveur_de_ligne():
40     if pin13.read_digital() and pin14.read_digital():
41         i2c.write(0x10, bytearray([0x00, 0x1, 20]))
42         i2c.write(0x10, bytearray([0x02, 0x1, 20]))
43     if not pin13.read_digital() and not pin14.read_digital():
44         i2c.write(0x10, bytearray([0x00, 0x0, 40]))
45         i2c.write(0x10, bytearray([0x02, 0x0, 40]))
46     if not pin13.read_digital() and pin14.read_digital():
47         i2c.write(0x10, bytearray([0x02, 0x0, 40]))
48         i2c.write(0x10, bytearray([0x00, 0x0, 20]))
49     if pin13.read_digital() and not pin14.read_digital():
50         i2c.write(0x10, bytearray([0x00, 0x0, 40]))
51         i2c.write(0x10, bytearray([0x02, 0x0, 20]))
52
53 def Avancer_5_cm():
54     if getUltrasonicData(pin1, pin2, 'distance') > 5:
55         i2c.write(0x10, bytearray([0x00, 0x0, 50]))
56         i2c.write(0x10, bytearray([0x02, 0x0, 50]))
57         time.sleep_ms(450)
58         i2c.write(0x10, bytearray([0x00, 0, 0]))
59         i2c.write(0x10, bytearray([0x02, 0, 0]))
60     else:
61         display.show(Image.NO)
62         i2c.write(0x10, bytearray([0x00, 0, 0]))
63         i2c.write(0x10, bytearray([0x02, 0, 0]))
64
65 def Rotation_droite_90():
66     i2c.write(0x10, bytearray([0x02, 0x1, 50]))
67     i2c.write(0x10, bytearray([0x00, 0x0, 50]))
68     time.sleep_ms(750)
69     i2c.write(0x10, bytearray([0x00, 0, 0]))
70     i2c.write(0x10, bytearray([0x02, 0, 0]))
71
72 def rotation_gauche_90():
73     i2c.write(0x10, bytearray([0x02, 0x0, 50]))
74     i2c.write(0x10, bytearray([0x00, 0x1, 50]))
75     time.sleep_ms(750)
76     i2c.write(0x10, bytearray([0x00, 0, 0]))
77     i2c.write(0x10, bytearray([0x02, 0, 0]))
78

```

```

79 def Rotation_droite_45():
80     i2c.write(0x10, bytearray([0x02, 0x1, 50]))
81     i2c.write(0x10, bytearray([0x00, 0x0, 50]))
82     time.sleep_ms(375)
83     i2c.write(0x10, bytearray([0x00, 0, 0]))
84     i2c.write(0x10, bytearray([0x02, 0, 0]))
85
86 def rotation_gauche_45():
87     i2c.write(0x10, bytearray([0x02, 0x0, 50]))
88     i2c.write(0x10, bytearray([0x00, 0x1, 50]))
89     time.sleep_ms(375)
90     i2c.write(0x10, bytearray([0x00, 0, 0]))
91     i2c.write(0x10, bytearray([0x02, 0, 0]))
92
93 def Avancer_10_cm():
94     if getUltrasonicData(pin1, pin2, 'distance') > 10:
95         i2c.write(0x10, bytearray([0x00, 0x0, 50]))
96         i2c.write(0x10, bytearray([0x02, 0x0, 50]))
97         time.sleep_ms(900)
98         i2c.write(0x10, bytearray([0x00, 0, 0]))
99         i2c.write(0x10, bytearray([0x02, 0, 0]))
100     else:
101         display.show(Image.NO)
102         i2c.write(0x10, bytearray([0x00, 0, 0]))
103         i2c.write(0x10, bytearray([0x02, 0, 0]))
104
105 def Avancer_20_cm():
106     if getUltrasonicData(pin1, pin2, 'distance') > 20:
107         i2c.write(0x10, bytearray([0x00, 0x0, 50]))
108         i2c.write(0x10, bytearray([0x02, 0x0, 50]))
109         time.sleep_ms(1800)
110         i2c.write(0x10, bytearray([0x00, 0, 0]))
111         i2c.write(0x10, bytearray([0x02, 0, 0]))
112     else:
113         display.show(Image.NO)
114         i2c.write(0x10, bytearray([0x00, 0, 0]))
115         i2c.write(0x10, bytearray([0x02, 0, 0]))
116
117 radio.config(channel = 7, power = 6, length = 32, group=0)
118 temp = 0
119 deplacement = 10
120

```

```

20
21 while True:
22     numberData = radio_receiveData()
23     if numberData is not None:
24         deplacement = numberData
25     if deplacement == 0:
26         suiveur_de_ligne()
27     if deplacement == 1:
28         Avancer_5_cm()
29         deplacement = -1
30     if deplacement == 2:
31         Avancer_10_cm()
32         deplacement = -1
33     if deplacement == 3:
34         Avancer_20_cm()
35         deplacement = -1
36     if deplacement == 4:
37         Rotation_droite_90()
38         deplacement = -1
39     if deplacement == 5:
40         Rotation_droite_45()
41         deplacement = -1
42     if deplacement == 6:
43         rotation_gauche_90()
44         deplacement = -1
45     if deplacement == 7:
46         rotation_gauche_45()
47         deplacement = -1
48     if deplacement == 8:
49         i2c.write(0x10, bytearray([0x00, 0, 0]))
50         i2c.write(0x10, bytearray([0x02, 0, 0]))
51         display.show('-100°C')
52

```

Pour aller plus loin

Afin de compléter la programmation de ce livret il est possible de rajouter les activités suivantes en cours d'année ou sur d'autres niveaux en parallèle.

Réalisation d'objets en 3D :

- Mini cratère qui s'adapte au parcours
- Rocher
- Base Martienne
- Panneau solaire

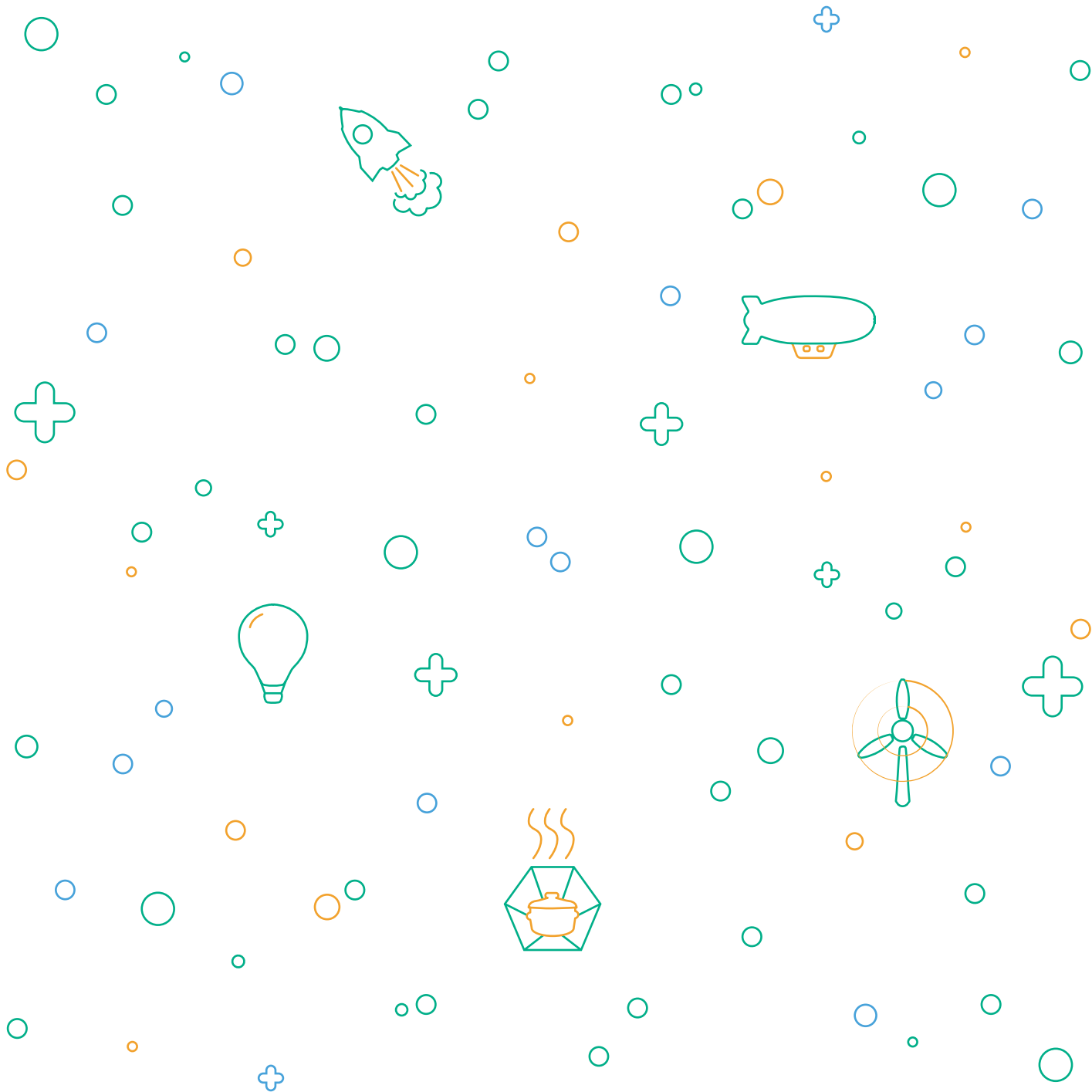
Un projet plus conséquent serait de prévoir une base pour accueillir le robot. On peut imaginer de partir de l'idée d'un garage avec porte automatique, cela permettra d'utiliser le capteur à ultrason pour détecter la présence du robot pour l'ouverture et la fermeture du sas d'entrée de la base. Dans ce projet, il y a de la programmation, mais aussi de la conception et de la fabrication avec machine outils, imprimante 3D ou découpe laser.



Un exemple de rocher martien implimé en 3D, qui fait un obstacle idéal pour le capteur de distance !

Notes

This image shows a full page of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page, typical of notebook paper. There are no margins, text, or other markings on the page.





Voici le livret d'activité du robot martien.
Il détaille chaque activité nécessaire à la réalisation de l'expérience.
Ce n'est pas exhaustif, l'imagination et les ressources disponibles
sur le site sont là pour aider à approfondir l'expérience !

vitta
science