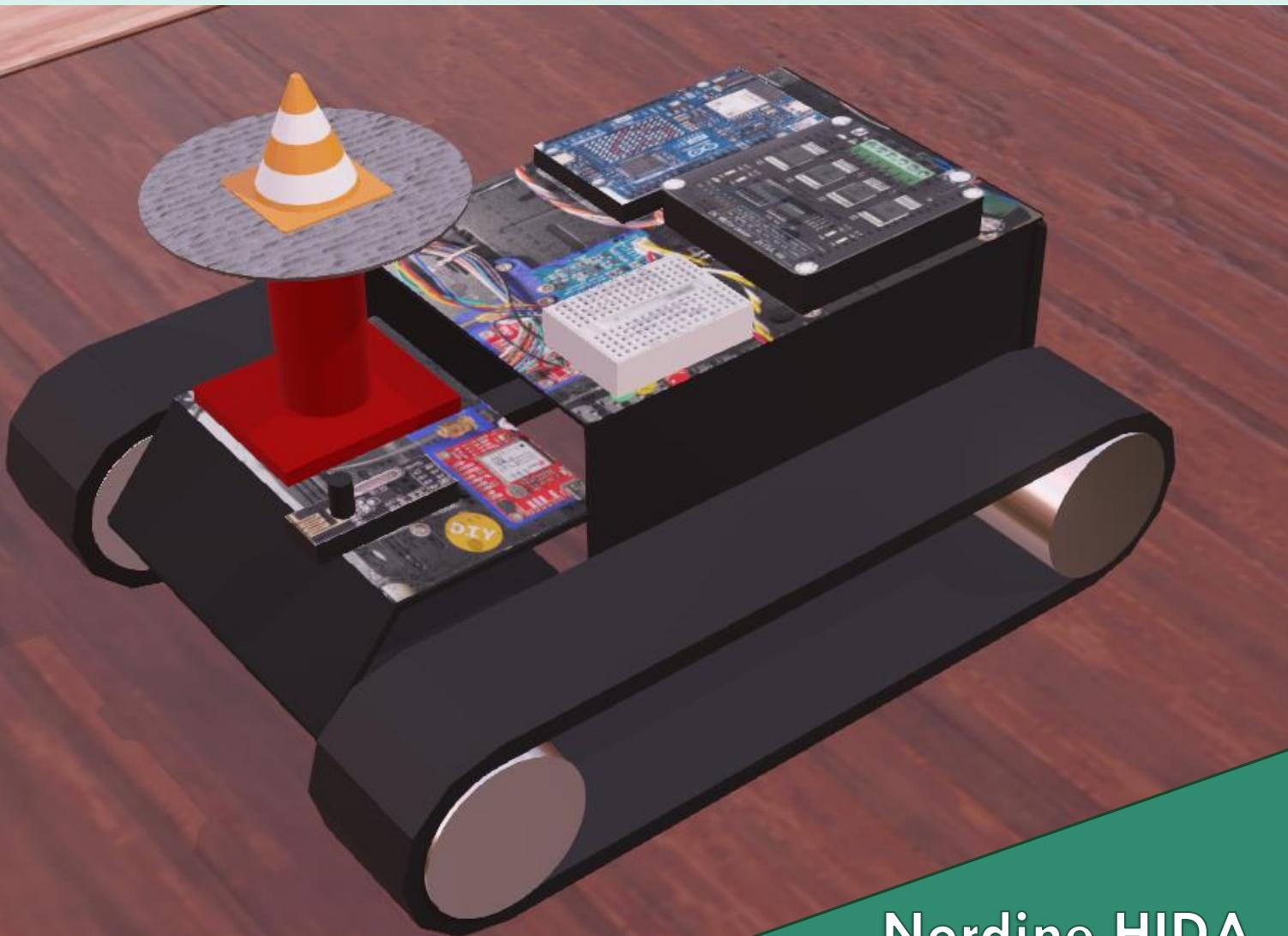


# Guide d'utilisation

# Webots

Ce guide présente Webots et la simulation associée au robot Alphie.



# Sommaire

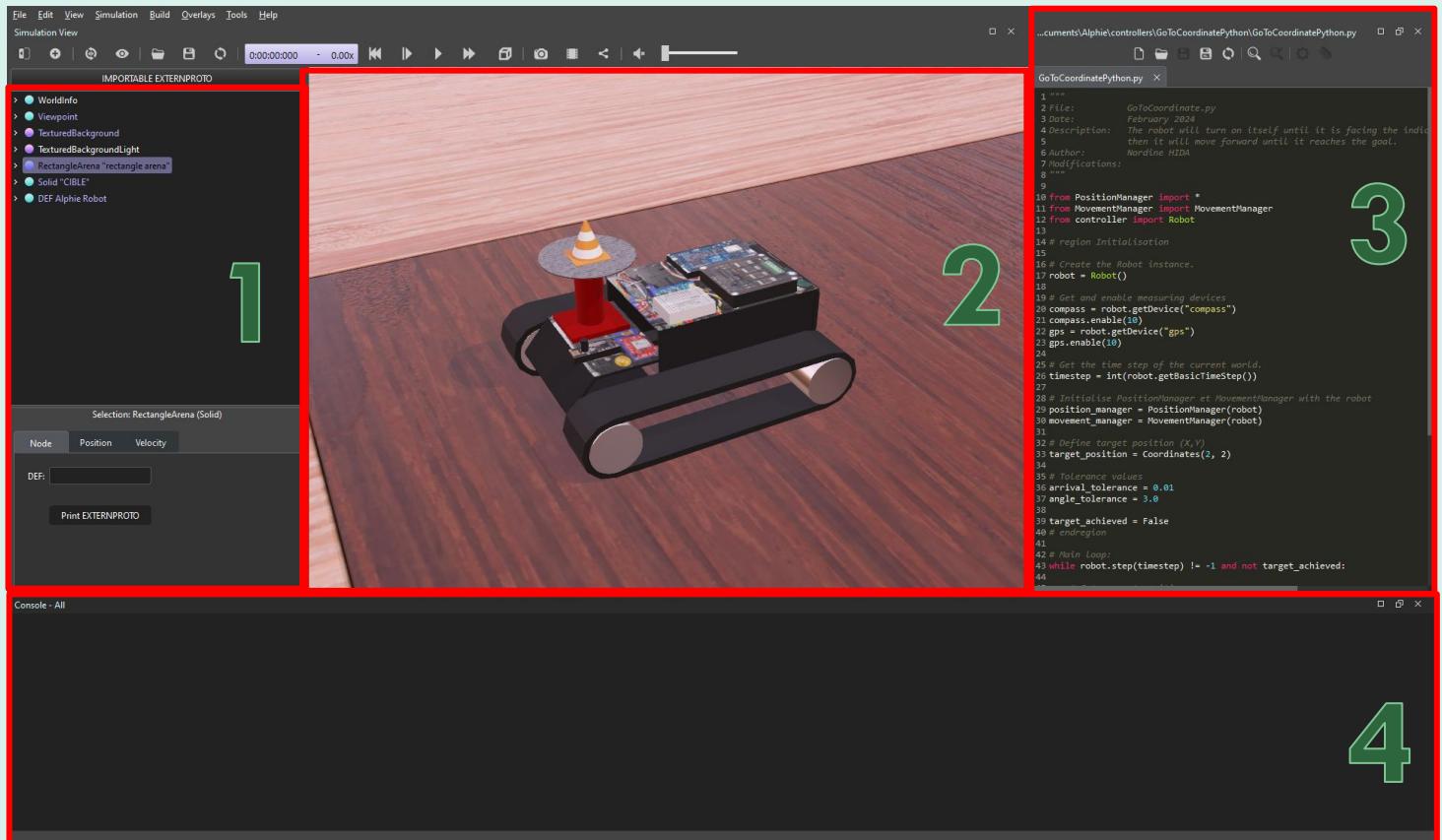
<b>INTRODUCTION A L'ENVIRONNEMENT DE WEBOTS .....</b>	<b>1</b>
<b>L'INTERFACE.....</b>	<b>1</b>
Arbre de scène (1) .....	1
Fenêtre de simulation (2) .....	3
Editeur de texte (3).....	3
Console (4).....	3
LOGIQUE ET STRUCTURE DES SIMULATIONS WEBOTS.....	4
<i>Calcul et interactions :</i> .....	4
<i>Synchronisation et réalisme :</i> .....	4
<i>Contrôle du temps :</i> .....	4
UTILISER PYCHARM POUR CREER UN PROGRAMME.....	5
<i>Prérequis .....</i>	5
<i>Configurer PyCharm pour l'utiliser avec Webots.....</i>	6
<i>Exécuter le programme depuis PyCharm .....</i>	12
<b>UTILISER LA SIMULATION POUR ALPHIE .....</b>	<b>14</b>
PRESENTATION .....	14
LE MONDE DE SIMULATION .....	15
<i>L'environnement .....</i>	16
<i>Les obstacles / décorations.....</i>	17
<i>Les entités robotique communicantes (ERC) .....</i>	18
CHRONOLOGIE DES EVENEMENTS .....	18
LA COMMUNICATION.....	20
<i>Les messages .....</i>	20
Le nom de l'expéditeur .....	20
Les types de messages .....	20
Compteur de transmission .....	21
Charge utile (payload).....	21
Les destinataires.....	21
<i>Protocole de question / réponse.....</i>	22
<i>Les commandes de la remote.....</i>	23
LES ROBOTS (ALPHIE) .....	24
<i>Structure du robot sur Webots .....</i>	24
<i>Programme du robot .....</i>	24
LA TELECOMMANDE (REMOTE).....	25
<i>Structure de la télécommande sur Webots .....</i>	25
<i>Programme de la télécommande .....</i>	25
L'INITIALISEUR (INITIALIZER) .....	26
<i>Structure de l'initialiseur sur Webots .....</i>	26
<i>Programme de l'initialiseur .....</i>	26
<b>ANNEXE .....</b>	<b>27</b>
CONCEPTION SIMPLIFIEE DU CONTROLLER DES ROBOTS (MAINCONTROLLER).....	27
CONCEPTION SIMPLIFIEE DU CONTROLLER DE LA REMOTE (MAINCONTROLLERREMOTE).....	29
CONCEPTION SIMPLIFIEE DU CONTROLLER DE L'INITIALISEUR (MAINCONTROLLERINITIALIZER) .....	30

# Introduction à l'environnement de Webots

Nous partirons du principe que Webots a déjà été installé. (Sinon rendez-vous sur le [site officiel de Cyberbotics](#), et suivez les étapes d'installation). Version utilisée : R2023b

## L'interface

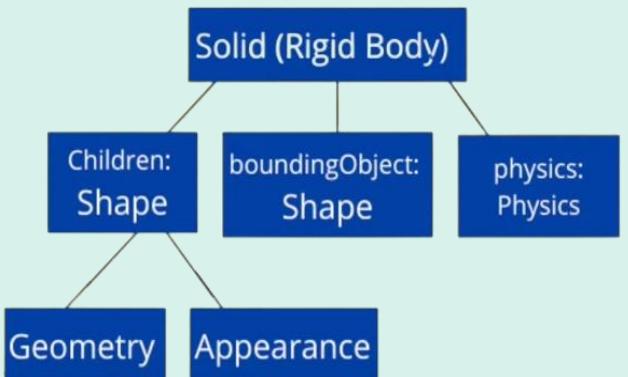
Webots est composé de plusieurs fenêtres qui ont chacune un rôle bien précis. Pour plus de détails, consultez [la documentation de Webots](#)



## Arbre de scène (1)

La simulation est composée d'éléments que l'on appelle des **nœuds**. Chaque nœud représente une entité qui peut être décomposée en sous-nœuds (appelé Children) à la manière d'un arbre. Il existe plusieurs types de nœud, mais nous nous concentrerons sur les principaux.

Prenons l'exemple d'un cube avec une physique (gravité et collision), voici sa composition sous forme d'un arbre. Chaque étage représente un sous-nœud.



"Solid" permet de définir une nouvelle entité sans corps.

"Shape" représente le corps de l'élément (on peut avoir plusieurs corps par entité).

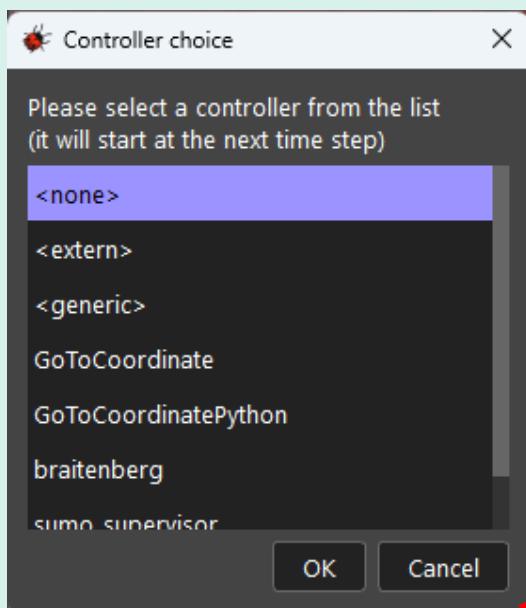
"Geometry" permet de définir la forme de « Shape », logiquement dans notre exemple ce sera « Cube ».

"Appearance" définit le visuel de son nœud parent (« Shape »). De nombreux paramètres sont disponibles comme la couleur ou l'application de textures.

"boundingObject" permet d'ajouter des collisions entre l'objet et son environnement, en spécifiant sa nature ici « Shape ».

"Physics" comme son nom l'indique, ajoute une physique à l'entité.

Concernant la simulation d'un robot, le nœud "controller" permet de sélectionner le programme qui s'exécutera sur le robot.



<none> n'exécutera rien

<extern> permet d'exécuter un programme depuis l'extérieur de Webots

(voir [Exécuter le programme depuis PyCharm](#))

<generic> si le programme sélectionné ne peut être exécuté, c'est le generic qui s'exécute.

DEF Alphie Robot

- translation 3.85 0.18 -4.37e-05
- rotation 0.0672 -0.552 -0.831 0.000196
- children
- name "AlphieBeta"
- model "SRV-1"
- description ""
- contactMaterial "default"
- immersionProperties
- boundingObject Group
- physics Physics
  - locked FALSE
  - translationStep 0.01
  - rotationStep 0.262
  - radarCrossSection 0
  - recognitionColors
  - controller "GoToCoordinatePython"
  - controllerArgs

Selection: controller (String)

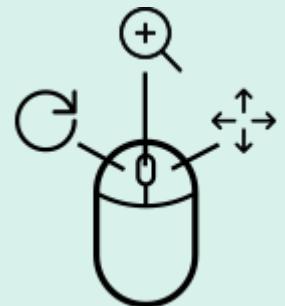
GoToCoordinatePython

Select... Edit

## Fenêtre de simulation (2)

C'est dans cette fenêtre qu'apparaît le rendu de [l'arbre de scène](#).

**Rotation de la caméra** : cliquez sur un objet avec le bouton gauche de la souris et faites glisser la souris pour faire tourner le point de vue autour de lui. Si vous cliquez sur l'arrière-plan, la caméra tournera autour de sa propre position.



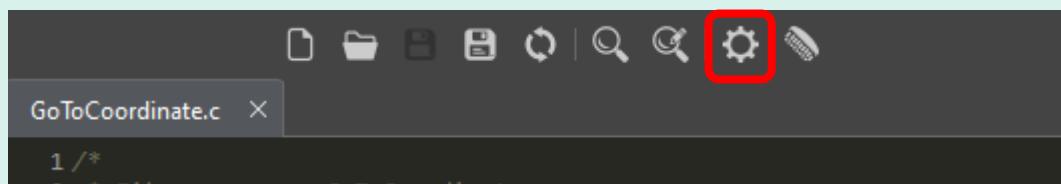
**Translation de la caméra** : appuyez sur le bouton droit et faites glisser la souris pour déplacer la caméra.

**Zoom / Rotation de la caméra** : appuyez simultanément sur les boutons gauche et droit de la souris (ou simplement sur le bouton du milieu) et faites glisser la souris verticalement pour zoomer. Faites glisser la souris horizontalement pour faire tourner la caméra autour de l'axe de vision. Alternativement, la molette de la souris seule peut également être utilisée pour zoomer.

## Editeur de texte (3)

Cette fenêtre permet simplement d'éditer directement le programme que vous utilisez.

Si vous utilisez du C, C++ ou tout autre langage compilé vous pouvez compiler votre code depuis ce bouton :



## Console (4)

La console affiche les éléments de votre programme, les éventuelles erreurs ou toutes autres informations relatives à Webots.

Vous pouvez en ouvrir plusieurs<sup>1</sup> et filtrer la source des informations à afficher (un ou plusieurs robot(s) spécifique(s) par exemple) en cliquant droit et "Filter".

Voici un exemple très simple d'affichage en Python :

A screenshot of the Webots console window titled 'Console - All'. It shows the command 'python.exe -u hello.py' being run, followed by the output 'Hello world ! :)'. At the bottom, a note says 'INFO: 'hello' controller exited successfully.' On the left, there's a preview of a Python file named 'hello.py' containing the single line 'print("Hello world ! :)")'.

<sup>1</sup> Ouvrir une nouvelle console (CTRL + N)

# Logique et structure des simulations Webots

Dans Webots, la simulation est structurée en étapes discrètes appelées "steps". Chaque étape représente un instant dans le temps où les calculs sont effectués pour mettre à jour l'état des objets dans le monde virtuel.

## Calculs et interactions :

Pendant chaque étape de la simulation, Webots effectue des calculs pour mettre à jour la position, l'orientation et l'état des différents objets dans l'environnement virtuel. Cela inclut les robots, les capteurs, les actionneurs, les obstacles et tout autre élément interactif.

## Synchronisation et réalisme :

Faire avancer le "step" de la simulation permet de synchroniser les actions des différents objets dans le monde virtuel, ce qui contribue à une représentation plus réaliste du comportement des robots et de leur interaction avec leur environnement. Par exemple, si un robot se déplace, les capteurs doivent être mis à jour pour refléter les nouvelles informations de l'environnement.

## Contrôle du temps :

Avancer le "step" de la simulation permet également de contrôler le temps dans l'environnement virtuel. Cela peut être utile pour tester le comportement des robots dans des scénarios où le temps est crucial, comme la navigation en temps réel ou la coordination entre plusieurs robots.

En résumé, faire avancer le "step" de la simulation est essentiel pour synchroniser les actions des différents objets, pour réaliser des calculs précis à chaque instant et pour contrôler le temps dans l'environnement virtuel, ce qui contribue à une simulation réaliste et précise du comportement des robots.

Pour plus d'informations, dirigez-vous vers [la documentation de Webots](#).

# Utiliser PyCharm pour créer un programme

## Prérequis

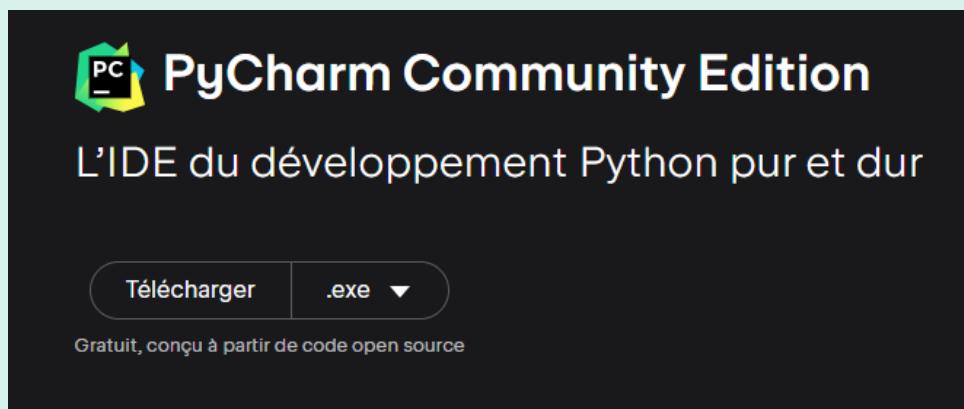
Nous partirons du principe que Python est correctement installé. (Si ce n'est pas le cas, rendez-vous sur [Microsoft Store](#) ou sur [le site officiel de Python](#), et suivez les étapes d'installations). La version utilisée ici est **Python 3.12.1**.

Pour vérifier votre version :

- Ouvrez une invite de commande <sup>2</sup>.
- Entrez la commande suivante : `Python -version`

Nous partirons également du principe que PyCharm a déjà été installé. (Si ce n'est pas le cas, rendez-vous sur [le site officiel de JetBrains](#), et suivez les étapes d'installation)

**⚠ Veillez à bien utiliser la version Communautaire qui est gratuite !**



## Pourquoi utiliser PyCharm ?

- L'interface ergonomique est conçue pour le développement Python.
- L'auto-complétion vous fera gagner du temps et vous évitera des erreurs de syntaxe.
- La détection d'erreurs avec leurs messages associés vous aidera grandement pour leur correction.
- La gestion de plusieurs fichiers/programmes est plus simple depuis le navigateur intégré.
- Vous pouvez exécuter le programme en mode Debug, ce qui vous permet de voir étape par étape comment s'exécute le code.

Bien sûr, ce ne sont que des recommandations, et [l'éditeur de texte de Webots](#) suffit amplement pour faire de petits programmes ou pour modifier des éléments.

---

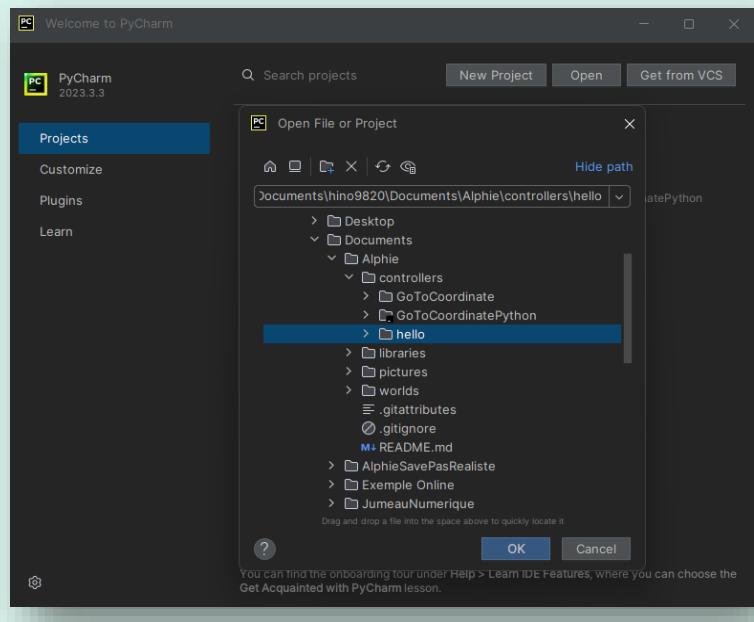
<sup>2</sup> Windows + R et tapez cmd (ou rechercher Invite de commande dans la barre de recherche Windows)

Les étapes de configuration sont adaptables pour d'autres environnements de développement.

## Configurer PyCharm pour l'utiliser avec Webots

### 1. Crédation du projet PyCharm

Lancez PyCharm et ouvrez le répertoire du contrôleur du robot Webots que vous souhaitez modifier en cliquant sur "Open" et en sélectionnant le répertoire approprié.



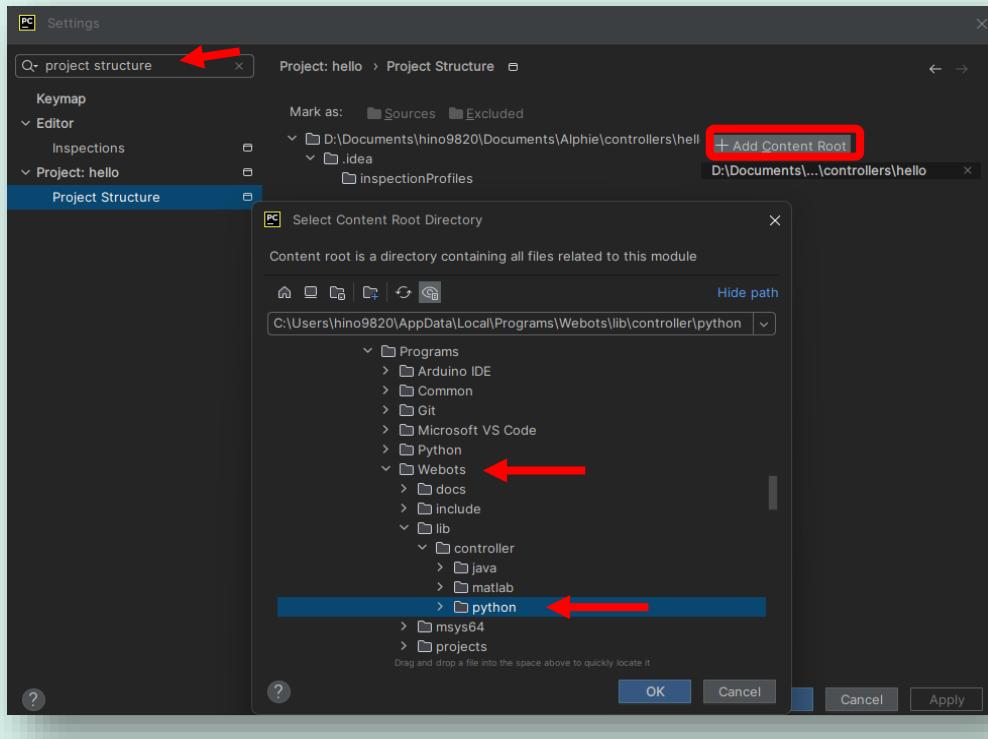
### 2. Ajout des bibliothèques de Webots au projet

- Sur PyCharm, allez dans "File" > "Settings", puis sélectionnez "Project Structure" (vous pouvez utiliser la barre de recherche en haut à gauche).
- Cliquez sur le bouton "Add Content Root" et ajoutez le dossier "`<CheminVersWebots>/lib/controller/python`".

Par exemple :

`C:\Users\user404\AppData\Local\Programs\Webots\lib\controller\python`

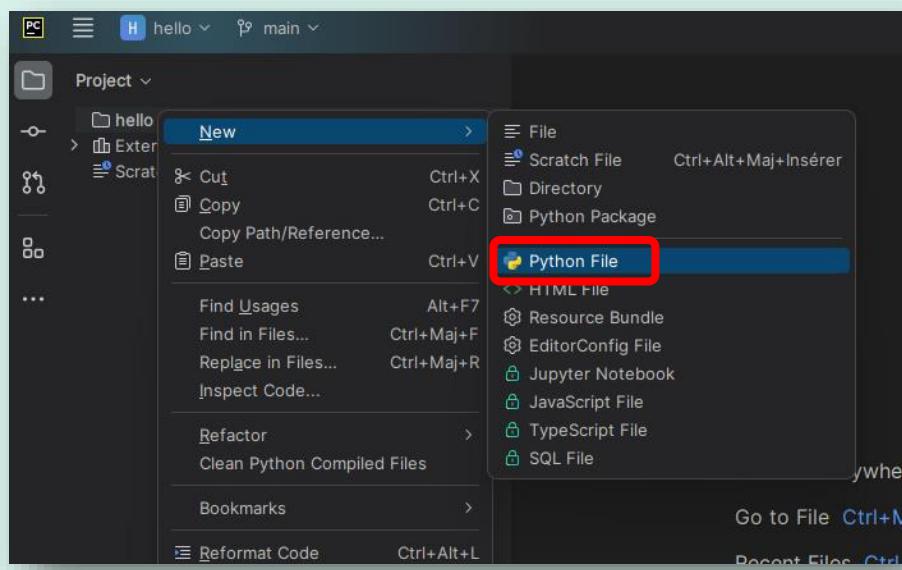
Voir capture ci-dessous :



### 3. Création de votre programme python

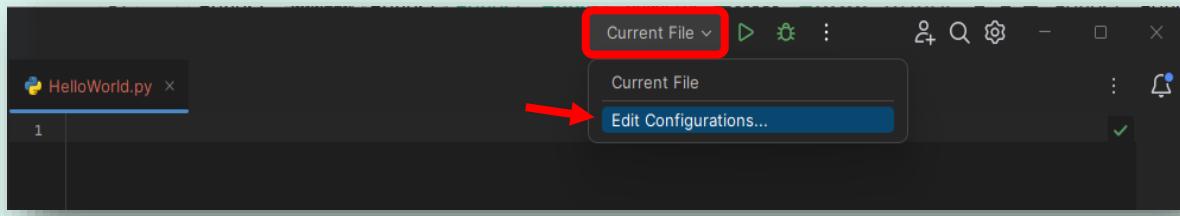
Cliquez droit sur votre dossier, puis ajoutez un fichier Python "VotreNomDeProgramme.py". Voir capture ci-dessous :

**⚠ Le nom de votre programme principal doit être identique au nom du dossier dans lequel il est situé !**



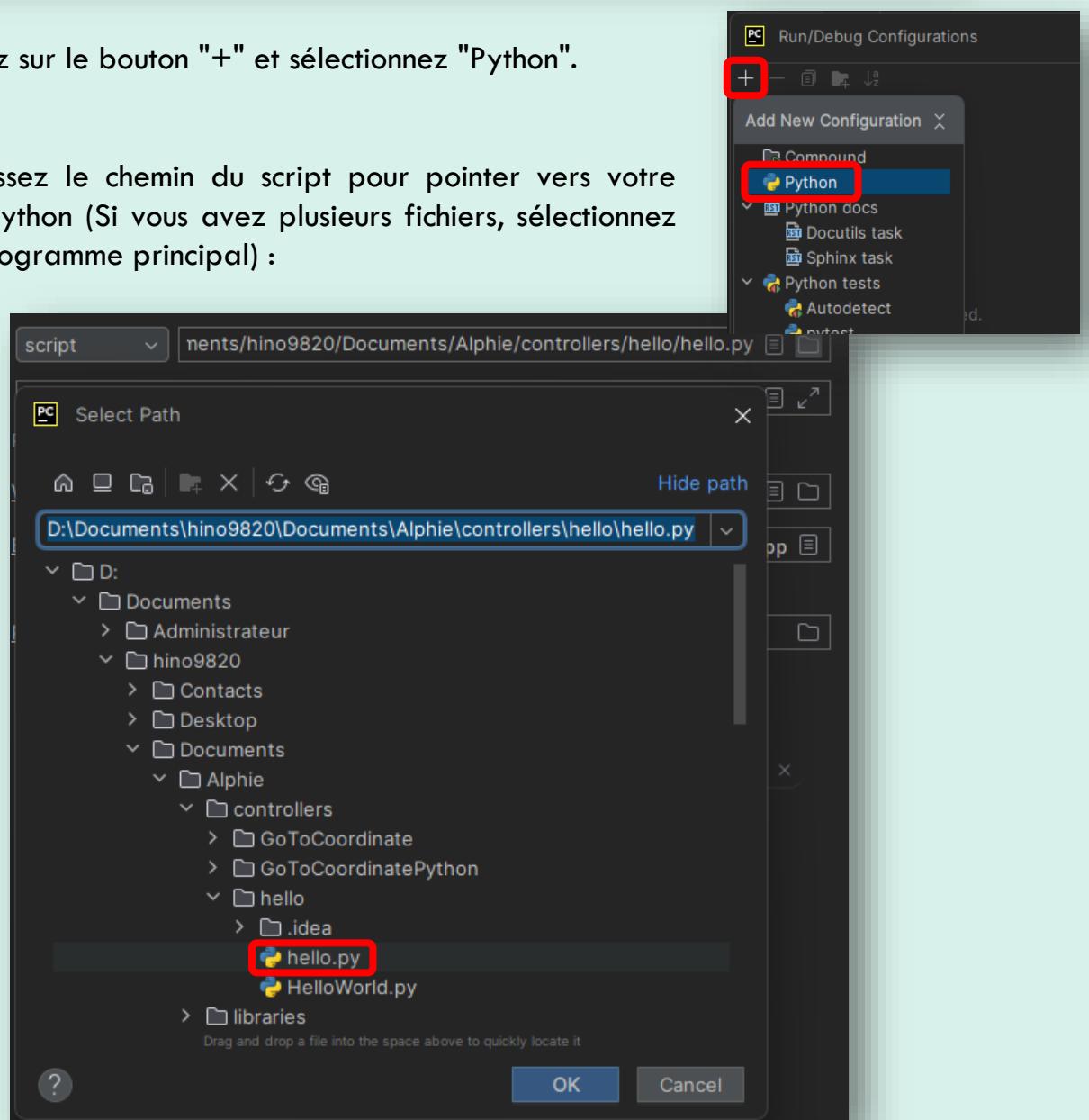
#### 4. Configuration des chemins d'exécution

- Une fois sur votre fichier Python, cliquez sur "Current File" > "Edit Configurations".

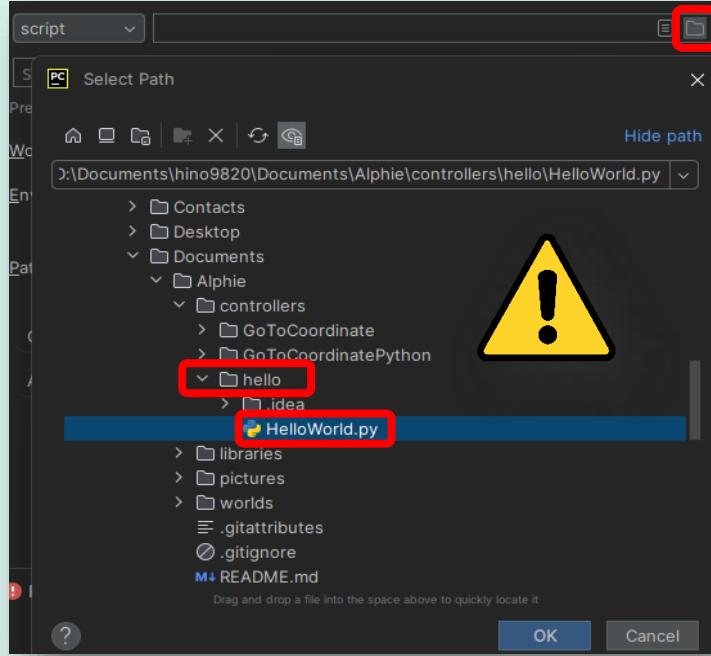


- Cliquez sur le bouton "+" et sélectionnez "Python".

- Définissez le chemin du script pour pointer vers votre fichier Python (Si vous avez plusieurs fichiers, sélectionnez votre programme principal) :



Voici ci-dessous un exemple qui ne **fonctionnera pas** car `HelloWorld.py` (programme principal) n'a pas le même nom que son dossier `hello`.

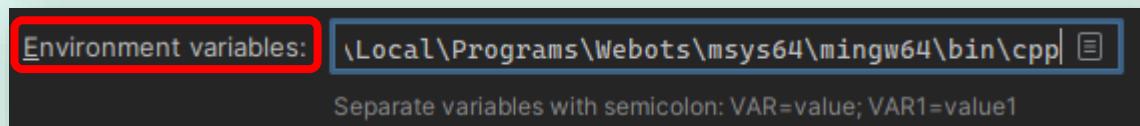


- Dans "Environment variables" ajoutez les chemins suivants en replaçant bien <CheminVersWebots> par votre chemin d'accès vers le dossier d'installation de webots (**⚠ Ne pas retirer les paramètres existant**) :

```
;Path=<CheminVersWebots>\lib\controller\;<CheminVersWebots>\msys64\mingw64\bin\cpp
```

Par exemple:

```
;Path=C:\Users\user404\AppData\Local\Programs\Webots\lib\controller\;C:\Users\user404\AppData\Local\Programs\Webots\msys64\mingw64\bin\cpp
```



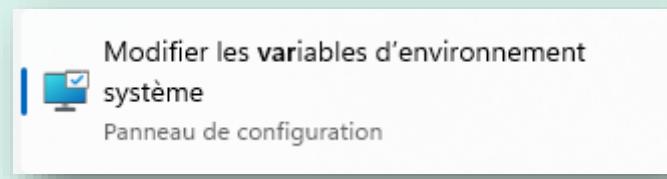
- Vous pouvez valider les paramètres.

Pour plus d'informations, vous pouvez consulter [la documentation de Webots](#).

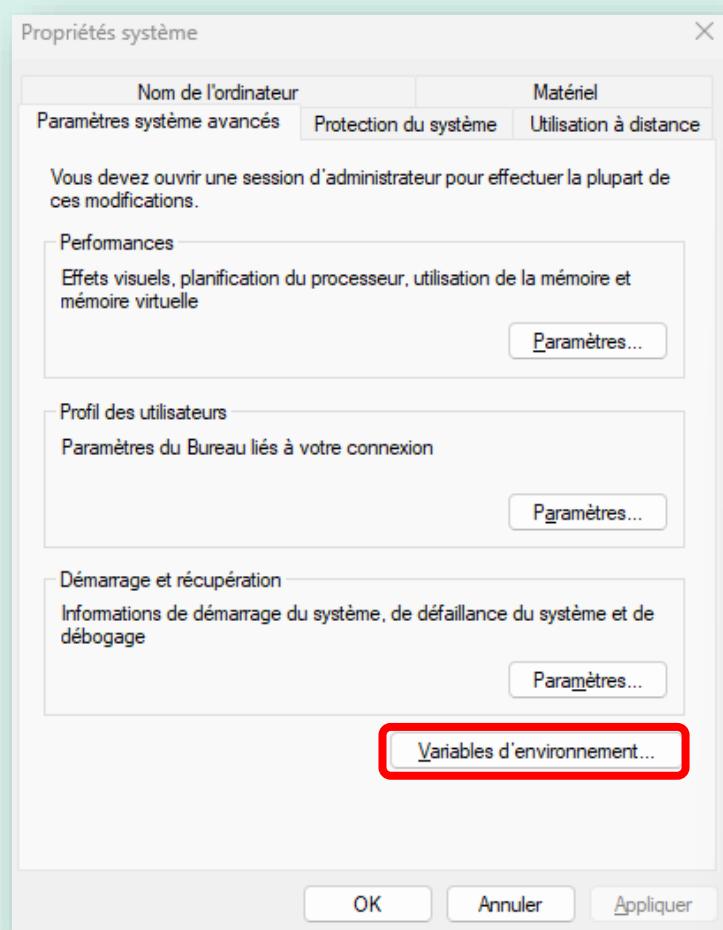
## 5. Ajout de la variable d'environnement WEBOTS\_HOME

Dans votre barre de recherche Windows, recherchez "Modifier les variables d'environnement système"

(  Cette opération nécessite les droits administrateurs)

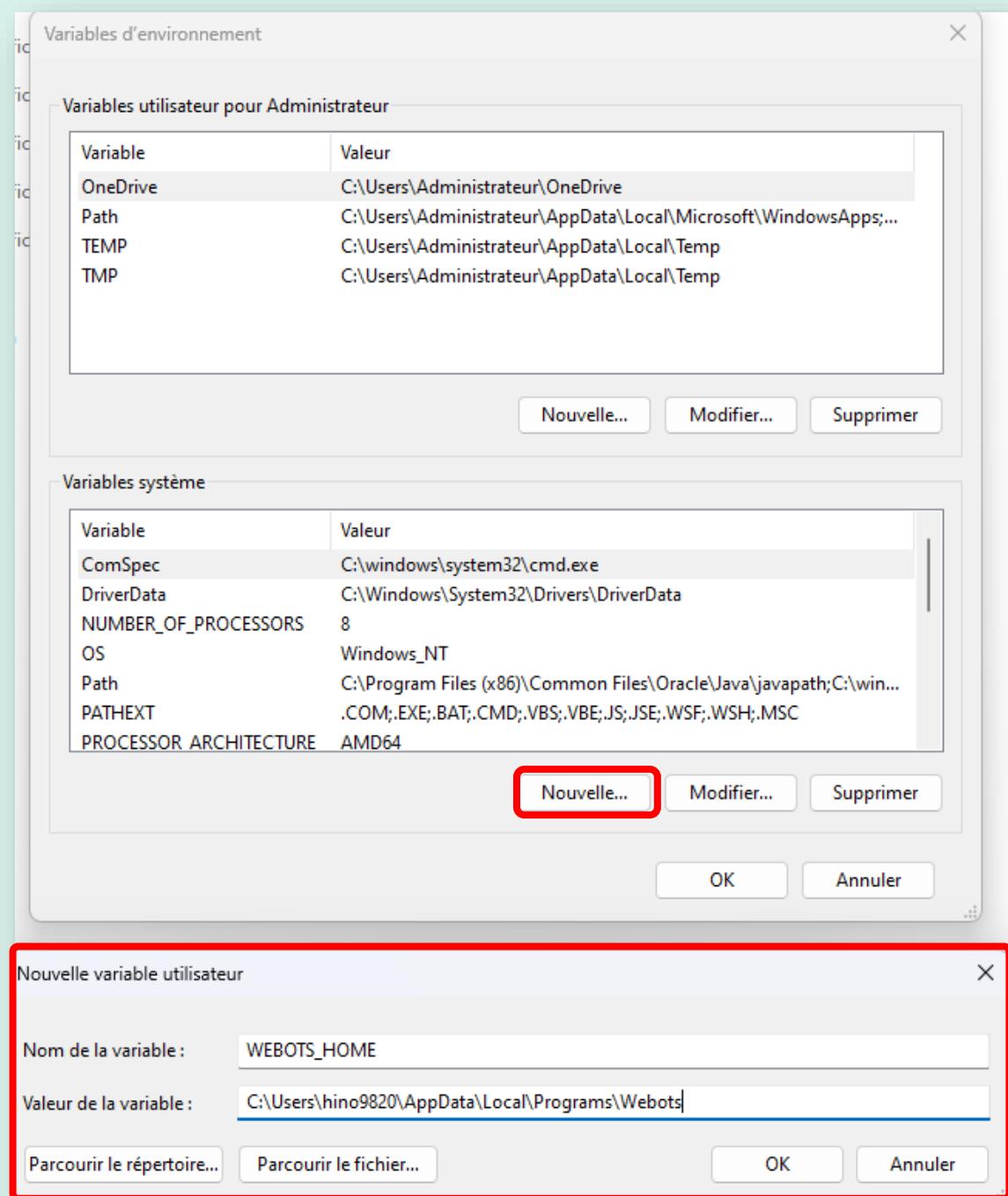


Appuyez sur "Variables d'environnement" en bas à droite :



Dans la partie "Variables système", cliquez sur "Nouvelle..." puis dans la nouvelle fenêtre, saisissez : (Voir ci-dessous)

- Nom de la variable : WEBOTS\_HOME
- Valeur de la variable : le chemin vers votre dossier d'installation Webots



Pour vérifier la manipulation, ouvrez une invite de commande (Windows + R puis tapez "cmd") et saisissez la commande suivante :

```
echo %WEBOTS_HOME%
```

Vous devriez retrouver le chemin saisi précédemment :

```
C:\Users\hino9820>echo %WEBOTS_HOME%
C:\Users\hino9820\AppData\Local\Programs\Webots
```

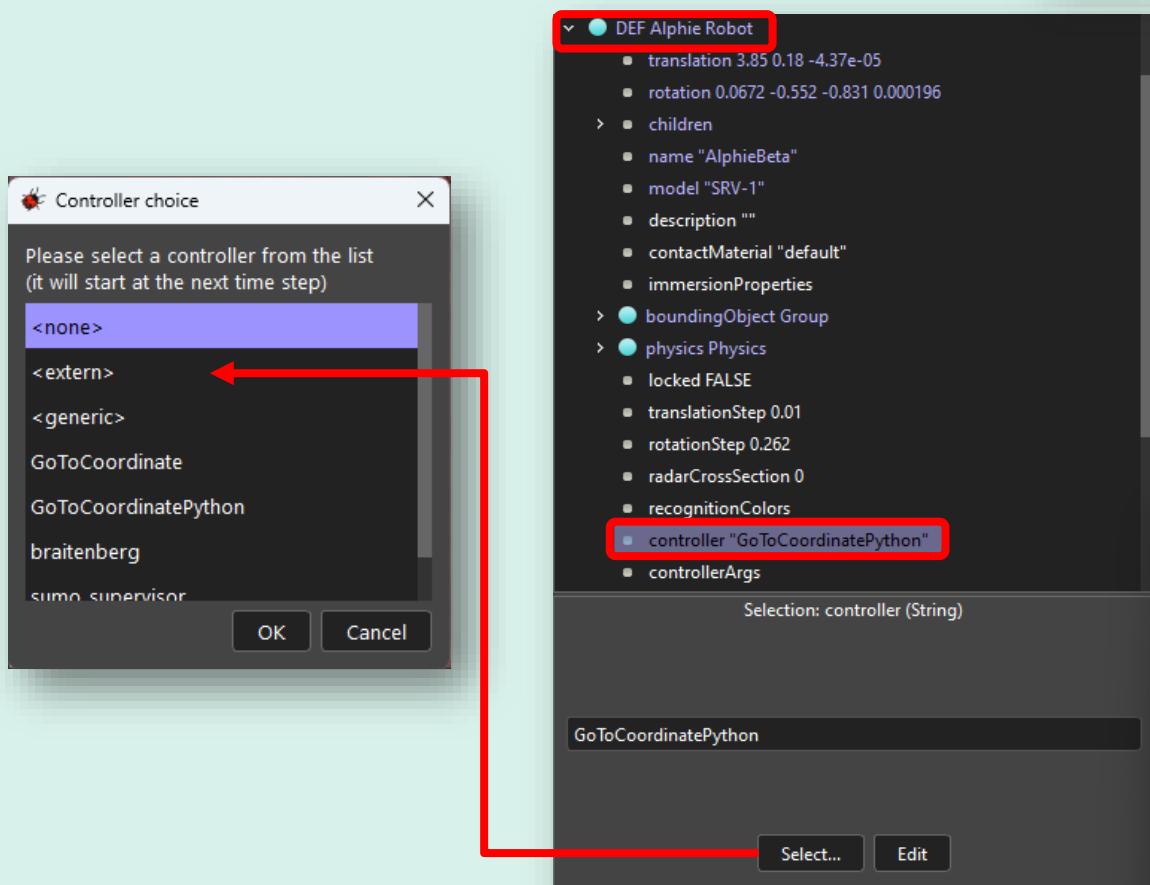
## Exécuter le programme depuis PyCharm

1. Démarrez Webots et ouvrez le monde souhaité.

Vous pouvez retrouver votre monde dans le dossier "worlds".  
Par défaut il s'appelle "flatWorld".



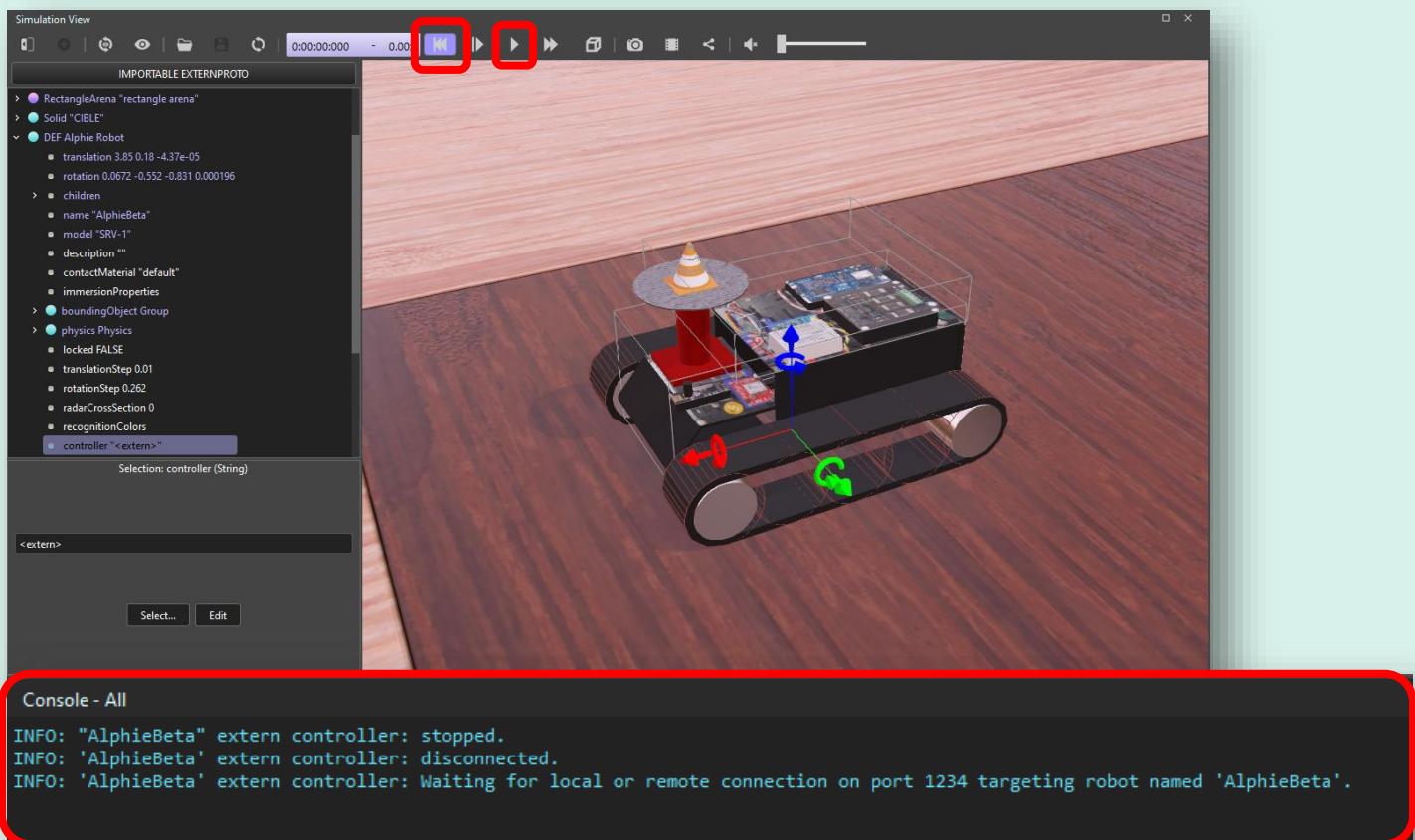
2. Configurez le contrôleur du robot sur <extern> dans Webots.



The screenshot shows the PyCharm IDE with two windows open. On the left, a 'Controller choice' dialog box is displayed, prompting the user to select a controller from a list. The list includes '<none>', '<extern>' (which is highlighted with a red arrow), '<generic>', 'GoToCoordinate', 'GoToCoordinatePython', 'braitenberg', and 'sumo supervisor'. At the bottom of this dialog are 'OK' and 'Cancel' buttons. On the right, a larger window shows the configuration of a robot named 'Alphie Robot'. Under the 'physics Physics' section, the 'controller' field is set to 'GoToCoordinatePython' (also highlighted with a red box). Below this window, a 'Selection: controller (String)' dropdown also contains 'GoToCoordinatePython'. At the bottom of the configuration window are 'Select...' and 'Edit' buttons.

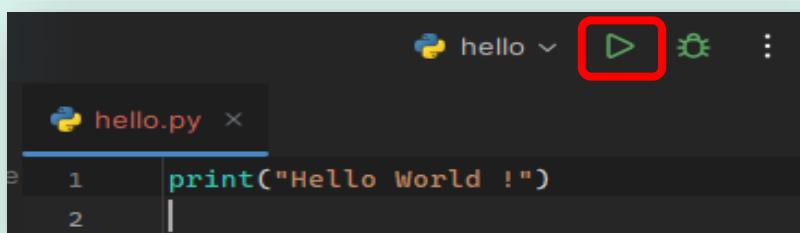
### 3. Réinitialiser la simulation

Appuyez sur le bouton "Reset simulation" pour mettre à jour la simulation avec le nouveau controller. Vous devriez avoir un message bleu dans la console comme ci-dessous, sinon appuyez sur le bouton "Run simulation" (voir ci-dessous) :



Dès lors, Webots est prêt à recevoir votre programme.

### 4. Depuis PyCharm, lancez le contrôleur à l'aide du menu "Run".

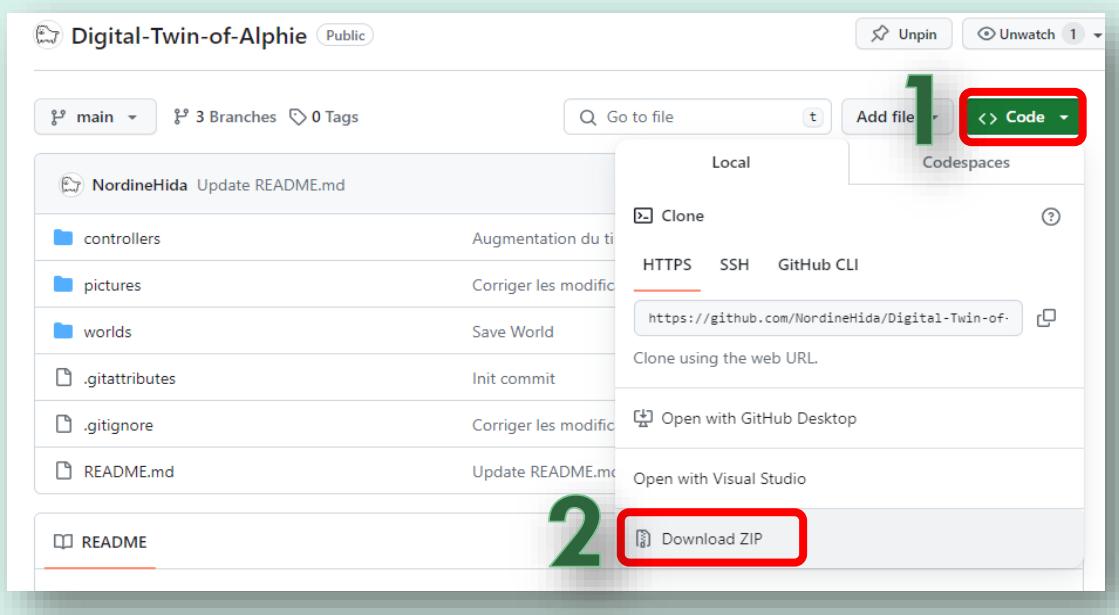


**⚠ Si vous utilisez une console, c'est celle de PyCharm qui sera utilisée et non celle de Webots !**

# Utiliser la simulation pour Alphie

Cette partie présente le jumeau numérique du rover Alphie. Tous les éléments présentés sont disponibles sur [le dépôt GitHub Digital-Twin-of-Alphie](#).

Téléchargez le dossier comme ci-dessous :



## Présentation

La simulation est composée de plusieurs entités dans un monde de simulation (chaque élément sera détaillé dans la partie qui lui est dédié).

Pour résumer, plusieurs robots sont disposés sur un plateau plat. Au démarrage, l'Initializer va instaurer une conscience d'environnement aux robots pour qu'ils aient connaissances de tous les robots existants.

Ces robots vont ensuite et tout au long de la simulation demander qui est présent autour d'eux afin d'avoir un maximum d'informations sur leur contexte proche. C'est ensuite à l'utilisateur (vous) d'utiliser le clavier pour envoyer des missions aux robots via la télécommande virtuelle.

Dans le dépôt que vous avez téléchargé vous retrouverez 3 dossiers :

### 1. controllers

Regroupe l'ensemble des controllers (programme) de la simulation. Il y en a un pour chaque type d'entité :

- [MainController](#) pour les **robots**.
- [MainControllerRemote](#) pour la **télécommande**.
- [MainControllerInitializer](#) pour l'**initialiseur**.

## 2. pictures

Regroupe les images utilisées pour modéliser les composants du robot. Il ne faut pas les supprimer ou les renommer, mais il est possible d'en ajouter.

## 3. worlds

Contient le monde dans lequel se déroule la simulation, aussi bien le décor que les entités (robots, télécommandes, etc...).

**Pour lancer la simulation il faut exécuter "flatWorld.wbt"**

Voir la partie suivante sur [le monde de simulation](#).



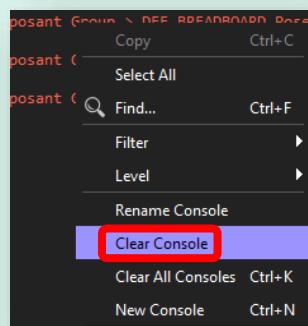
# Le monde de simulation

**⚠ Remarque : vous pouvez ignorer les warnings au lancement du monde.**

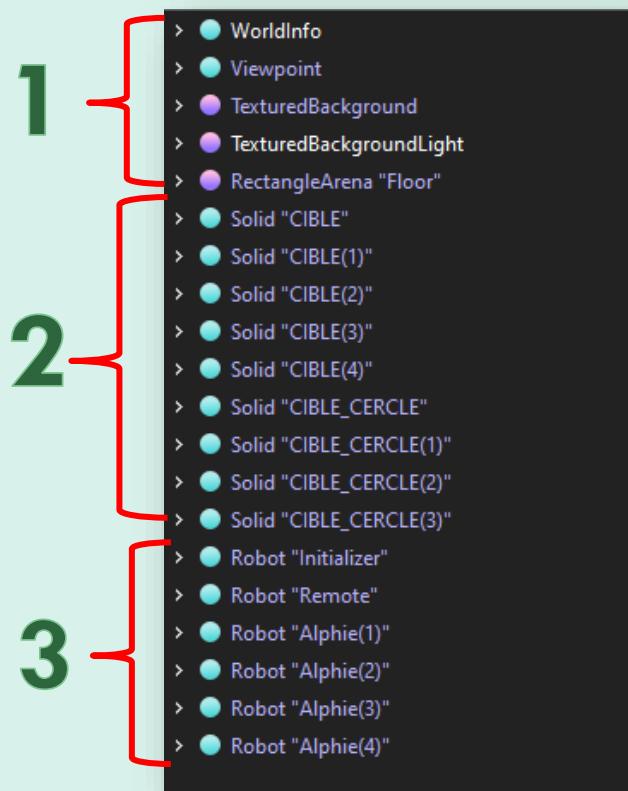
```
Console - All
WARNING: Robot "Alphie(1)" > DEF BODY Group > DEF FRONT_SUPPORT Pose > DEF Composant Group > DEF CARTEMOTEUR P
power of two: rescaling it from 514x486 to 1024x512.
WARNING: Robot "Alphie(1)" > DEF BODY Group > DEF FRONT_SUPPORT Pose > DEF Composant Group > DEF BREADBOARD Po
of two: rescaling it from 732x541 to 1024x1024.
WARNING: Robot "Alphie(1)" > DEF BODY Group > DEF FRONT_SUPPORT Pose > DEF Composant Group > DEF RADIO Pose >
```

(C'est simplement dû aux images qui n'ont pas une bonne taille mais Webots corrige automatiquement le problème).

Vous pouvez nettoyer la console en cliquant droit dessus et en sélectionnant "Clear Console".



L'arbre de scène de notre simulation peut être décomposé en 3 groupes :



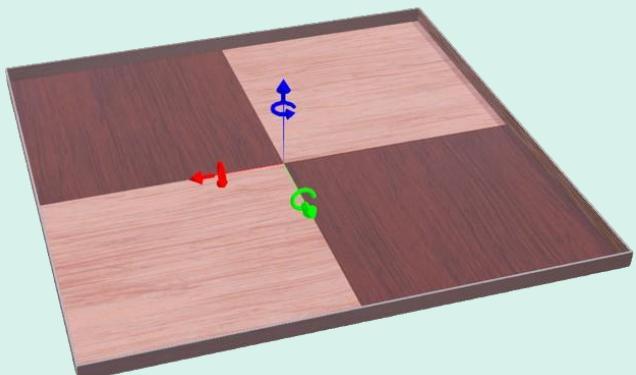
## L'environnement

"WorldInfo" permet de modifier les paramètres du monde (gravité, système de coordonnées...).

"RectangleArena" représente le sol. Il est possible de modifier sa taille et son apparence.

La simulation utilise un système de coordonnée ENU<sup>3</sup> dont le (0;0) est au centre du rectangle.

**Les axes :** rouge X, vert Y, bleu Z



Les autres nœuds permettent de modifier l'apparence du fond et des lumières.

<sup>3</sup> ENU : East North Up

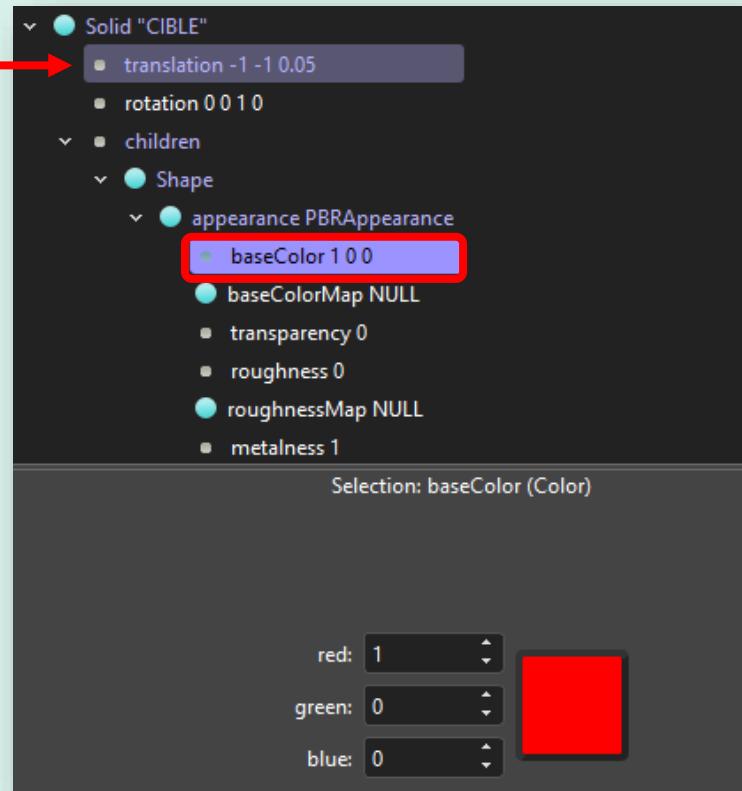
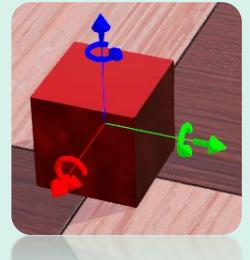
## Les obstacles / décos

Chaque "Solid Cible" représente un cube présent sur le "RectangleArena". Ils n'ont pas de physique et ont pour seules vocations de décorer la scène en présentant par exemple, les points d'un chemin ou une coordonnée à atteindre.

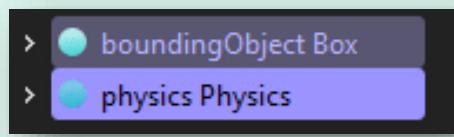
Vous pouvez en ajouter un simplement en copiant collant<sup>4</sup> un cube existant. Pour le déplacer, cliquez dessus et utilisez les flèches (voir ci-contre).

Sinon il est possible de le déplacer à des coordonnées spécifiques en les saisissant dans le paramètre "translation".

Son apparence, notamment sa couleur est aussi ajustable en suivant les noeuds suivant :



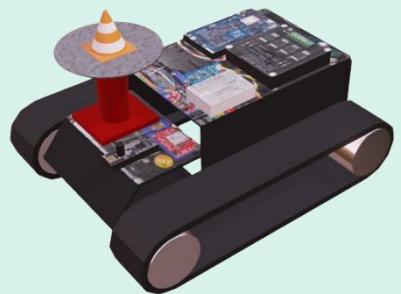
Ces cubes peuvent servir d'obstacle. Il suffit de leur ajouter une physique ([voir arbre de scène](#)) :



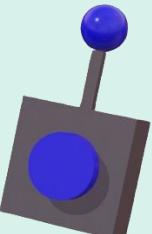
<sup>4</sup> Copier : CTRL + C | Coller : CTRL + V

## Les entités robotique communicantes (ERC)

**Les robots** (Alphie) : Ils sont les actionneurs de la simulation et tout l'intérêt est de les programmer de manière à ce qu'ils puissent interagir entre eux dans le but de répondre à une mission. Il est possible d'en ajouter<sup>5</sup> ou d'en supprimer (voir [la partie sur les robots](#)).



**La télécommande** (Remote) : Elle s'occupe d'envoyer les missions aux robots et de les arrêter au besoin (voir [la partie sur télécommande](#)).



**L'initialiseur** (Initializer) permet, comme son nom l'indique, d'initialiser chaque robot en leur partageant la liste complète de tous les robots de la simulation. Cette conscience du groupe est nécessaire aux robots pour interagir entre eux (voir [la partie sur l'initialiseur](#)).  
*NB : Cette entité n'existe pas dans la version physique du robot car la liste est directement implémentée manuellement.*

## Chronologie des événements

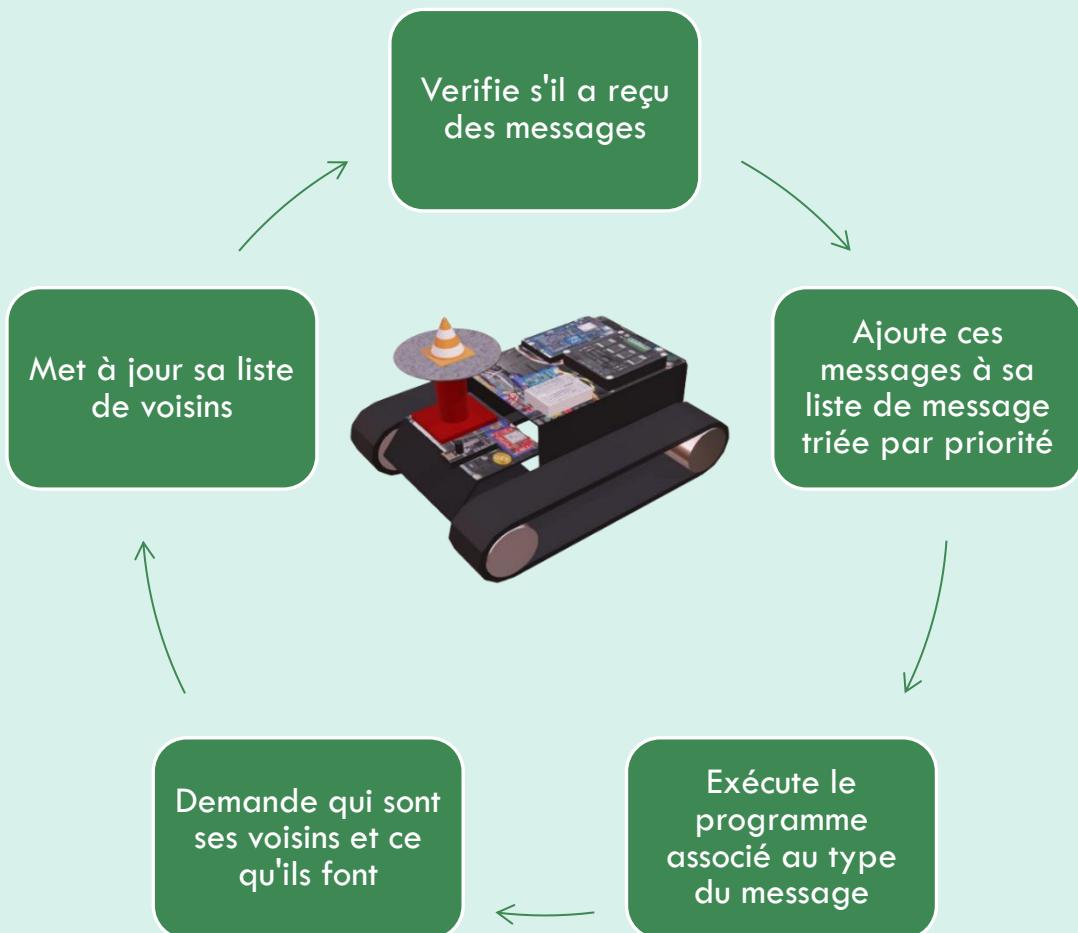
Voici ce qu'il se passe au lancement de la simulation :



Toutes ces étapes sont documentées dans le code source et sont détaillées dans la partie [Communication](#).

<sup>5</sup> Comme précédemment, en le copiant collant

Voici le déroulé de la boucle interne à chaque robot :



A chaque itération de cette boucle, un compteur appelé `timer.asking_neighbor` va s'incrémenter et s'il dépasse X itérations<sup>6</sup>, un message va s'envoyer à tous les robots connus afin de mettre à jour les robots proches.

Voici le code gérant cette variable, il est disponible dans le fichier du **MainController > NetworkManger.py**. (Ici le robot se met à jour toutes les 50 itérations).

```
# If the timer is over, we refresh our neighborhood by asking who's nearby
if self.timer.asking_neighbor > 50:
    # Ask who is nearby and send its own current task
    self.communication.send_message_all(self.robot_name,
MESSAGE_TYPE_PRIORITY.REPORT_STATUS, 0, self.robot.robot_current_task)
    self.timer.asking_neighbor = 0
else:
    self.timer.asking_neighbor += 1
```

<sup>6</sup> La valeur n'étant pas définitive, il vous est possible de la modifier afin d'essayer d'optimiser le flux de communication tout en permettant un rafraîchissement assez régulier.

# La communication

## Les messages

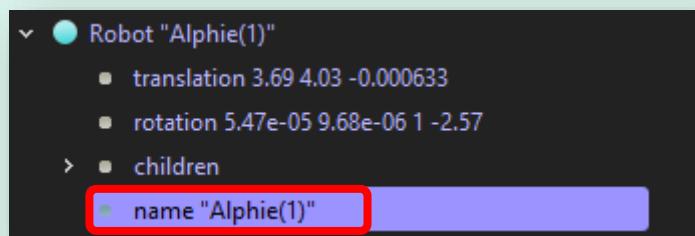
Les messages dans le contexte de Webots sont essentiels pour permettre aux robots et aux dispositifs distants de s'échanger des informations. Dans notre implémentation, nous utilisons une structure de message définie dans le fichier **Message.py**.

Un message est composé des éléments suivants séparé par un point-virgule ; :

```
nom_expediteur;type_message;compteur_transmission;payload;destinataire
```

### Le nom de l'expéditeur

L'expéditeur représente simplement le nom du robot qui envoie le message. Ce nom est défini dans le paramètre "name" du robot. Il peut être modifié à votre guise.



### Les types de messages

Les messages sont classés en différents types, chacun ayant une signification spécifique et une priorité associée.

- **REPORT\_...** : Ces messages sont des questions qui permettent de récupérer des informations sur les robots voisins.

Exemple : REPORT\_STATUS : Pour demander l'état actuel d'un robot.

- **STATUS\_...** : Ces messages sont des réponses qui indiquent l'état actuel du robot.

Exemple : REPORT\_GOTOCORDINATES : Indique que le robot est en train de se déplacer vers des coordonnées.

- Les autres messages sont des ordres, notamment pour réaliser une mission.

Exemple :

GO\_TO\_COORDINATES : Indique au robot qu'il doit se rendre à des coordonnées (précisé dans le payload).

STOP : Pour demander l'arrêt du robot et réinitialiser les données acquises (ses voisins, sa tâche actuelle, sa liste de message à traiter etc...).

**⚠️ Attention : Si vous ajoutez un nouveau type de message, veillez à bien spécifier sa priorité dans la méthode priority. Plus la valeur de priorité est importante, plus le message sera prioritaire.**

Le détail est disponible dans le fichier **MESSAGE\_TYPE\_PRIORITY.py**.

## Compteur de transmission

Chaque message contient un compteur de transmission qui indique le nombre de fois que le message a été retransmis d'un robot à l'autre. Cela peut être utile pour s'assurer qu'un message ne soit pas répété indéfiniment, notamment lors de la diffusion d'un message à travers l'ensemble du réseau de robot. Le compteur s'incrémentera automatiquement dans la méthode `send_message` du **CommunicationManager.py**. On peut voir ce compteur comme la durée de vie du message : lorsqu'il dépasse la valeur maximum, le message meurt et n'est plus retransmis.

**⚠️ Cette durée de vie est définie manuellement dans RobotUp.py dans la variable `max_counter`. Il serait intéressant de l'initialiser automatiquement avec le nombre de robot disponible par exemple.**

## Charge utile (payload)

Le payload permet d'ajouter des informations supplémentaires au message. Il est par exemple utilisé pour transmettre des coordonnées X et Y (séparées par deux points " : "), ou encore indiquer la tâche que le robot vient de terminer. Libre à vous de bien l'utiliser pour qu'il soit efficace à envoyer et à recevoir.

## Les destinataires

Chaque message doit être destiné à un ERC bien précis. Le destinataire est identifié par le nom Webots de l'entité. Si vous souhaitez envoyer un message à tous vos voisins, utilisez la méthode `send_message_all` qui va envoyer ce message à tous les robots connus, et donc seuls ceux à votre portée les recevront.

En utilisant ces concepts de base, notre protocole de communication permet aux robots de partager des informations de manière claire et efficace.

**⚠ Les fichiers `Message.py` et `MESSAGE_TYPE_PRIORITY.py` doivent être les mêmes pour tous les controllers (afin qu'ils parlent tous « la même langue »).** Pour plus d'information, voir [la conception des controllers](#).

## Protocole de question / réponse

La communication entre les ERC repose sur une structure Question / Réponse, c'est-à-dire que chacun attend un message "question" pour répondre un message "réponse" (voir [Les types de messages](#)).

Chaque ERC réagit différemment aux messages reçus. Par exemple, les robots vont s'immobiliser à la réception d'un message STOP, alors que la télécommande ou l'initialiseur ne vont rien faire.

Le détail de chaque interaction est détaillé dans la méthode `update` des différents fichiers `NetworkManager.py`. Pour les robots par exemple, selon le type de message, on appelle la fonction associée, voir le code ci-dessous :

```
match message_type:
    case MESSAGE_TYPE_PRIORITY.REPORT_STATUS:
        self.case_REPORT_STATUS(id_sender)

    case MESSAGE_TYPE_PRIORITY.STATUS_CURRENT_TASK:
        self.case_STATUS_CURRENT_TASK(id_sender, payload)

    case MESSAGE_TYPE_PRIORITY.REPORT_POSITION:
        self.case_REPORT_POSITION(id_sender, payload)

    case MESSAGE_TYPE_PRIORITY.STATUS_GOTOCOORDINATES:
        self.case_STATUS_GOTOCOORDINATES(id_sender, payload)

    case MESSAGE_TYPE_PRIORITY.STATUS_FREE:
        case_executed = self.case_STATUS_FREE(id_sender, payload)

    case MESSAGE_TYPE_PRIORITY.STOP:
        self.case_STOP(send_counter)

    case MESSAGE_TYPE_PRIORITY.GO_TO_COORDINATES:
        self.case_GO_TO_COORDINATES(id_sender, payload)

    case MESSAGE_TYPE_PRIORITY.STATUS_OUT_RANGE:
        self.case_STATUS_OUT_RANGE(payload)

    case _:
        print("Unknown message received")
        pass
```

## Les commandes de la remote

Pour initier une mission, il faut envoyer un message depuis la télécommande. Pour ce faire il suffit d'appuyer sur la touche du clavier associée.

**⚠ Assurez-vous d'avoir la simulation au premier plan en cliquant sur la fenêtre de simulation !**

Voici les commandes disponibles pour l'instant (vous pouvez l'afficher dans la console en appuyant sur n'importe quelle touche du clavier non-associée) :

Any key	: Show all commands
END	: Stop everything and reset robot's data
PAGEDOWN	: Go to coordinates (Circle around the (0;0))
RIGHT	: Go to coordinates (1;1)
LEFT	: Go to coordinates (-1;1)
UP	: Go to coordinates (1;-1)
DOWN	: Go to coordinates (-1;-1)

Si vous souhaitez ajouter une nouvelle commande, il vous suffit d'ajouter le code suivant dans la méthode `update` du fichier **NetworkManagerRemote.py** :

```
elif key == Keyboard.'La touche souhaité':
    self.communication.send_message('Ici mettre le message souhaité')
    'ou le code à executer'
```

Voici un exemple avec la commande pour arrêter les robots :

```
elif key == Keyboard.END:
    self.communication.send_message_all(self.robot_name,
MESSAGE_TYPE_PRIORITY.STOP, 0)
```

(Il pourrait être intéressant de remplacer les `if-elif-elif...` par un `match case`)

# Les robots (Alphie)

## Structure du robot sur Webots

Le robot de simulation est composé de groupes de nœuds, chacun responsable d'une partie du robot.

**BODY** : Corps du robot. Regroupe le châssis, ses composants et les chenilles.

**ANTENNA\_EMITTER/RECEIVER** :  
Antenne de communication du robot.

Vous y trouverez **RangeEmitter** qui est une sphère permettant de visualiser la portée de communication (attention à bien définir son rayon avec la même valeur que la portée de l'émetteur).

**WHEEL** : moteurs qui font tourner les chenilles.

- **children**
  - > ● Group
  - > ● DEF BODY Group
  - > ● DEF ANTENNA\_EMITTER Emitter
  - > ● DEF ANTENNA\_RECEIVER Receiver
  - > ● DEF FIRST\_LEFT\_WHEEL HingeJoint
  - > ● DEF SECOND\_LEFT\_WHEEL HingeJoint
  - > ● DEF THIRD\_LEFT\_WHEEL HingeJoint
  - > ● DEF FOURTH\_LEFT\_WHEEL HingeJoint
  - > ● DEF LAST\_LEFT\_WHEEL HingeJoint
  - > ● DEF FIRST\_RIGHT\_WHEEL HingeJoint
  - > ● DEF SECOND\_RIGHT\_WHEEL HingeJoint
  - > ● DEF THIRD\_RIGHT\_WHEEL HingeJoint
  - > ● DEF FOURTH\_RIGHT\_WHEEL HingeJoint
  - > ● DEF LAST\_RIGHT\_WHEEL HingeJoint

## Programme du robot

Cette partie détaille le rôle de chaque classe (fichier) du programme du robot (**MainController**). La conception est [disponible en annexe](#).

**RobotUp** : Surcouche du robot de Webots avec des variables supplémentaires. Cette classe est au centre du programme puisqu'elle représente le robot. Par exemple elle stocke la liste des messages du robot, sa portée d'émission, la liste de ses voisins ou encore la liste des coordonnées qu'il doit parcourir. Il est possible de reset ces attributs via la méthode du même nom.

**MainController** : programme principal qui initialise le RobotUp puis rentre dans une boucle infinie (voir [Chronologie des événements](#)). **Ce fichier doit avoir le même nom que le dossier du controller.**

**InitialisationManager** : Initialise les appareils du robot (boussole, moteurs, émetteur...)

**CommunicationManager** : Gère les messages du RobotUp. Cette classe permet d'envoyer, de recevoir, d'effacer et comparer les messages.

**Message** et **MESSAGE\_TYPE\_PRIORITY** : corps du message (voir [les messages](#)).

**NetworkManager** : Interprète les messages reçus et réagit en conséquence. La méthode `update` traite les messages par ordre de priorité, et mets à jour la liste des robots voisins. Cette classe est essentielle dans le fonctionnement du robot vis-à-vis de son environnement.

**Task\_GoToCoordinates** : Permet de déplacer le robot à des coordonnées en continuant son actualisation via le NetworkManager pour que le robot reste à l'écoute même s'il se déplace.

**PositionManager** : Gère le positionnement et l'orientation en utilisant la boussole et le GPS du robot. Il permet de s'orienter vers une direction ou de vérifier si des coordonnées sont atteintes.

**MovementManager** : Gère les déplacements de base du robot (avancer, reculer, tourner et s'arrêter).

**Coordinates** : Structure représentant des coordonnées bi-dimensionnelles X et Y.

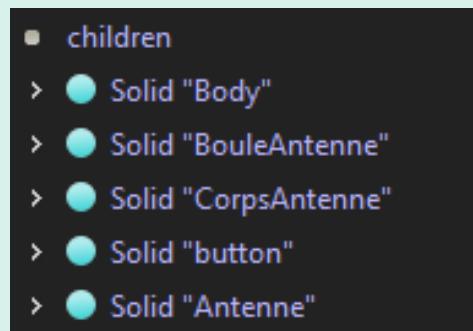
## La télécommande (Remote)

### Structure de la télécommande sur Webots

**BODY** : Corps de la télécommande. (Pavé gris)

**BouleAntenne - CorpsAntenne - Button** : Sont les autres parties de télécommande.

**Antenne** : regroupe l'émetteur et le receveur. Permet de modifier la portée de la télécommande.



## Programme de la télécommande

Cette partie détaille le rôle de chaque classe (fichier) du programme de la télécommande (MainControllerRemote). La conception est [disponible en annexe](#).

**RobotUpRemote** : Surcouche du robot<sup>7</sup> de Webots avec des variables supplémentaires. Cette classe est au centre du programme puisqu'elle représente

<sup>7</sup> Dans le contexte de Webots la télécommande est un robot (c'est un nœud robot).

la télécommande. Par exemple elle stocke la liste des messages du robot ou la liste de ses voisins. Elle est similaire au RobotUp sans les attributs qui lui sont inutiles.

**MainControllerRemote** : programme principal qui initialise le RobotUpRemote puis rentre dans une boucle infinie (voir [Chronologie des événements](#)). **Ce fichier doit avoir le même nom que le dossier du controller.**

**NetworkManagerRemote** : Interprète les messages reçus et réagit en conséquence. La méthode `update` traite les messages par ordre de priorité, et met à jour la liste des robots voisins. Elle permet d'envoyer des messages depuis le clavier (voir [Les commandes de la remote](#)). Cette classe est essentielle dans le fonctionnement de la télécommande vis-à-vis de son environnement.

**InitialisationManager - CommunicationManager - Message - Coordinates - MESSAGE\_TYPE\_PRIORITY** : Ces fichiers sont similaires à ceux du robot (voir [programme du robot](#)).

## L'initialiseur (Initializer)

### Structure de l'initialiseur sur Webots

L'initialiseur a la même [structure que la télécommande](#) à la différence que sa portée doit rester infinie.

### Programme de l'initialiseur

Cette partie détaille le rôle de chaque classe (fichier) du programme de l'initialiseur (MainControllerInitializer). La conception est [disponible en annexe](#).

**RobotUpInitializer** : Surcouche du [supervisor de Webots](#). Un Superviseur est un robot spécial doté de pouvoirs supplémentaires pour modifier l'environnement de simulation. Il peut récupérer les nœuds du monde et c'est à partir de ces derniers que l'initialiseur peut déterminer le nombre de robots pour les initialiser dans le MainControllerInitializer.

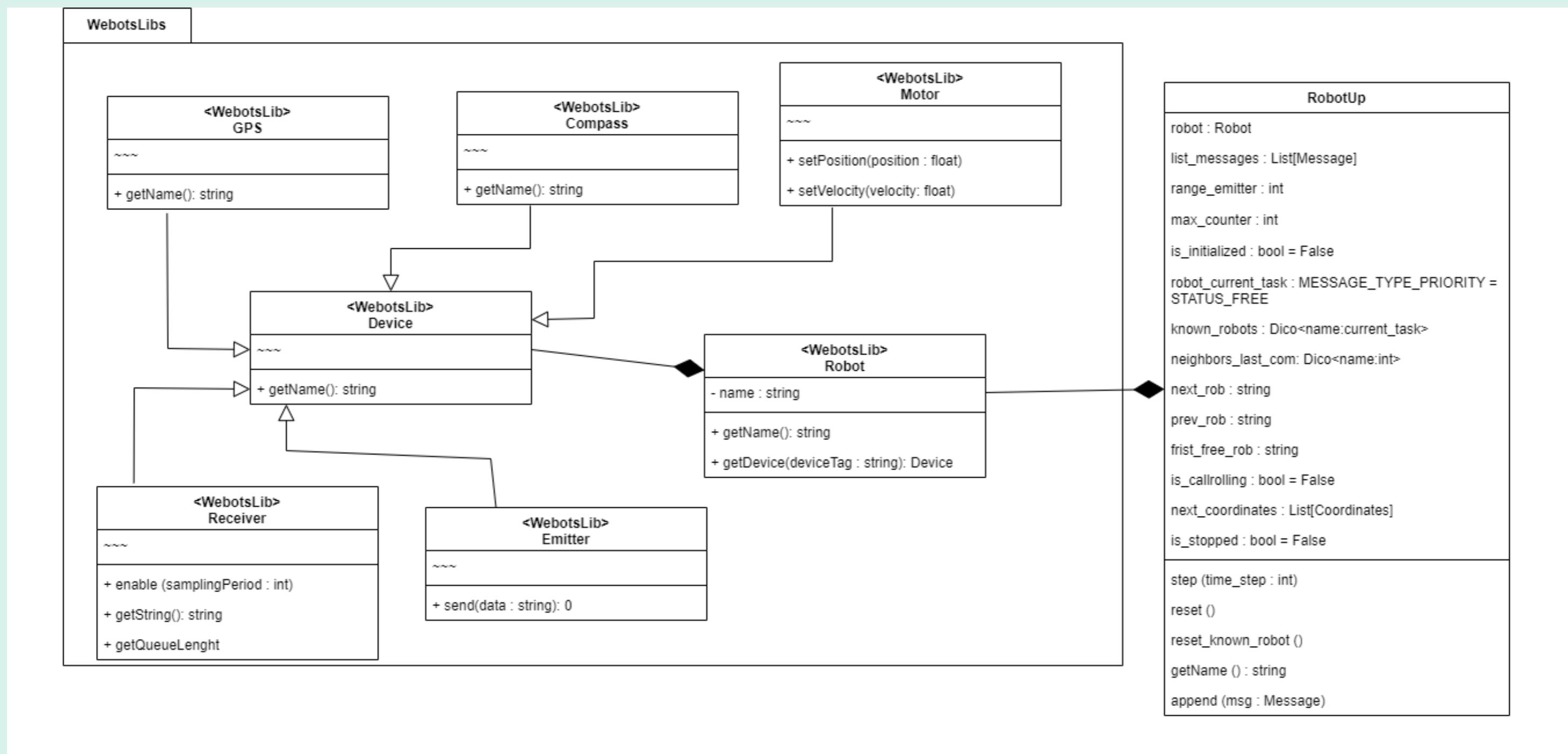
**MainControllerInitializer** : programme principal qui crée l'initialiseur. À partir de ce dernier, il récupère les robots de la simulation, puis les initialisent en leur envoyant la liste de tout les robots disponibles. **Ce fichier doit avoir le même nom que le dossier du controller.**

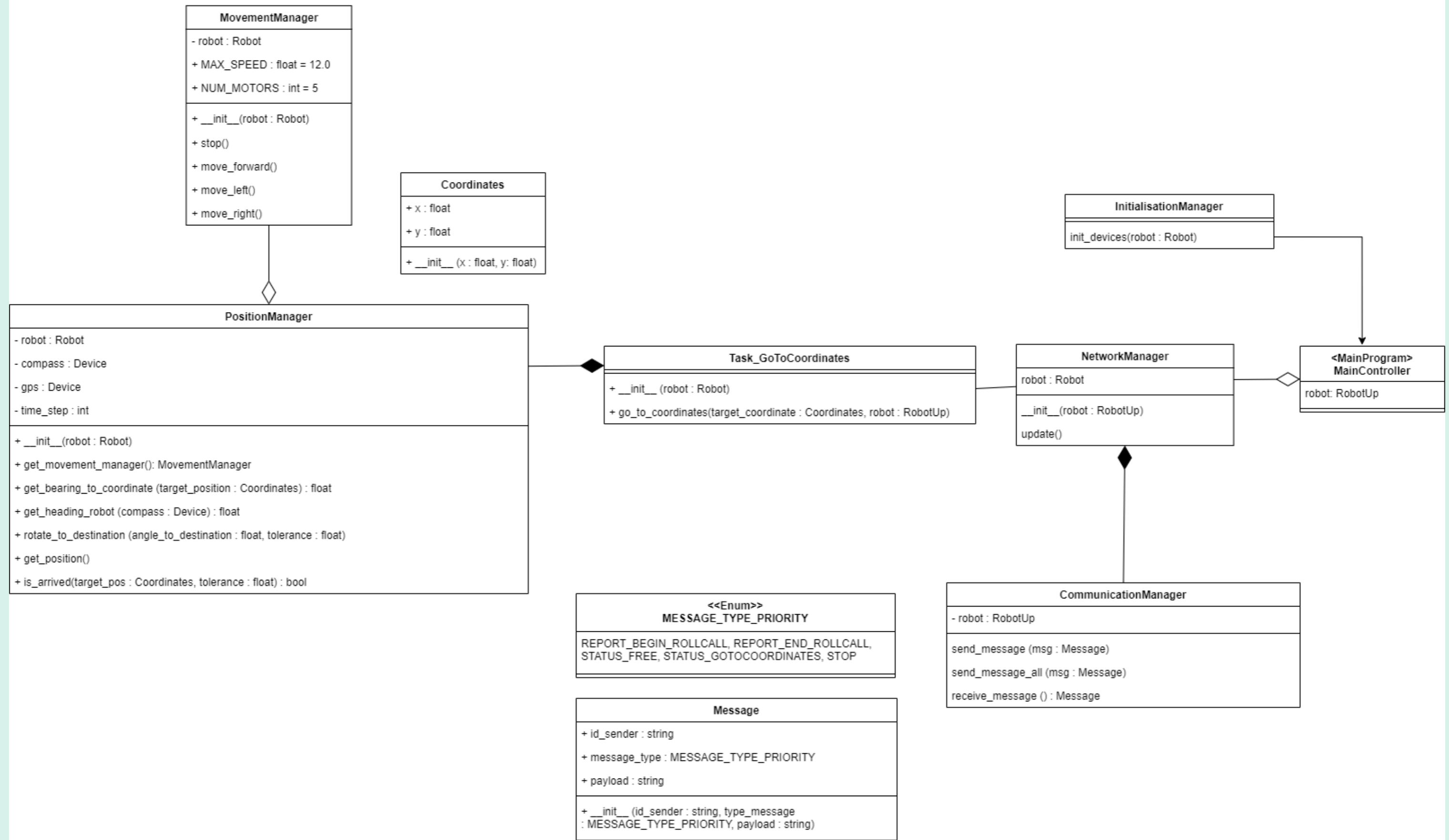
**NetworkManagerInitializer** : Interprète les messages reçus et réagit en conséquence. Seul le "rollcall" (appel) est utilisé ; il permet de récupérer les noms des robots et de les partager.

**InitialisationManager - CommunicationManager - Message - MESSAGE\_TYPE\_PRIORITY** : Ces fichiers sont similaires à ceux du robot (voir [programme du robot](#)).

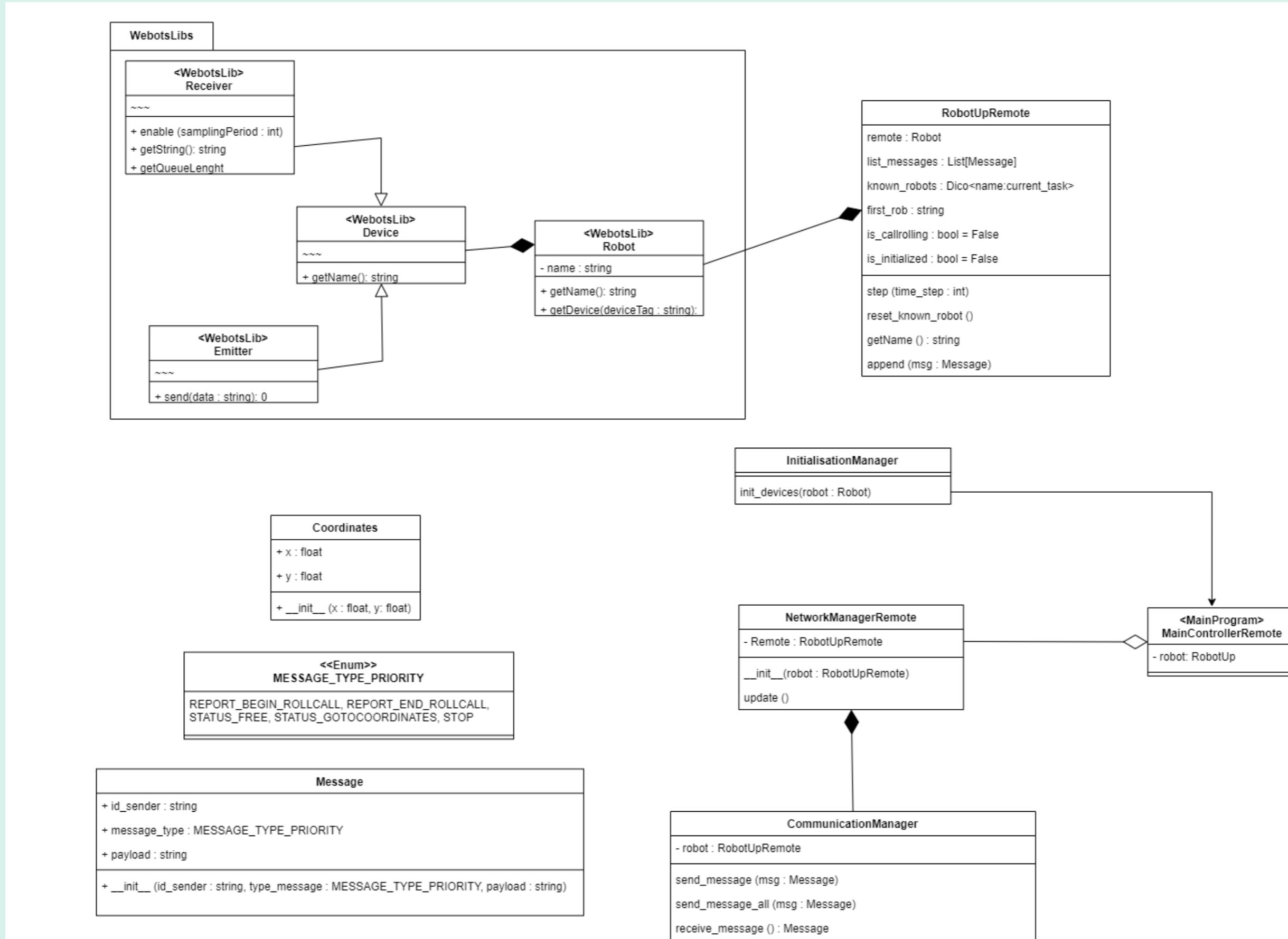
## Annexe

### Conception simplifiée du controller des robots (MainController)





## Conception simplifiée du controller de la remote (MainControllerRemote)



## Conception simplifiée du controller de l'initialiseur (MainControllerInitializer)

