Tuto Webots - 1

L'objectif de ce premier tutoriel est de vous familiariser avec l'interface utilisateur et avec les concepts de base de Webots.

Vous allez créer votre première simulation contenant un environnement simple:

- une arène avec plancher et murs,
- quelques boîtes,
- un robot e-puck,
- un contrôleur qui fera mouvoir le robot et mesurer une distance.
- adapter cette simulation au robot Thymio

Créer un Nouveau Monde

- World est un fichier contenant des informations comme où se trouvent les objets, à quoi ils ressemblent, comment ils interagissent les uns avec les autres, quelle est la couleur du ciel, et les définitions de la gravité, de la friction, des masses des objets, etc.
- Lord de la création, il définit l'état initial d'une simulation.
- Les différents objets sont appelés Noeuds et sont organisés hiérarchiquement pour former une Scène.
- Par conséquent, un noeud peut contenir des sous-nœuds.
- Un monde est stocké dans un fichier ayant .wbt comme extension.
- Le format de fichier est dérivé du langage VRML97 (https://en.wikipedia.org/wiki/VRML).
- Les fichiers du monde doivent être stockés directement dans un répertoire appelé worlds.

Mise en pratique – 1

Création nouveau projet

- Créer un nouveau projet à partir de : Wizards / New project directory.
- Nommez le répertoire du projet ma_premiere_simulation au lieu de la proposition my_project.
- Nommez le fichier mondial ma_premiere_simulation.wbt au lieu de la proposition empty.wbt.
- Cochez toutes les cases à cocher, y compris la case "Add a rectangle Area" qui n'est pas cochée par défaut.

Un arbre de nœuds (scène) est créé où leurs propriétés sous forme de champs peuvent être modifiés.
On doit observer les nœuds suivants:

- **WorldInfo** : contient les paramètres globaux de la simulation.
- **Viewport** : définit les principaux paramètres de la caméra de point de vue.
- **TexturedBackground** : définit l'arrière-plan de la scène (vous devriez voir des montagnes par exemple))
- **TexturedBackgroundLight** : définit la lumière associée à l'arrière-plan ci-dessus.
- **RectangleArena** : définit un espace plan pouvant contenir des objets statiques ou dynamiques (robot).

Mise en pratique – 2

Noeud Worldinfo

Modifions des propriétés

- Le nœud WorldInfo est un élément clé dans un fichier de monde Webots (.wbt),
- Il définit les **propriétés globales** de l'environnement de simulation, telles que la gravité, le système de coordonnées, ou encore les paramètres de rendu.
- Ces paramètres influencent le comportement physique et visuel de la simulation.

Nous nous concentrons sur trois paramètres clés : la gravité, le système de coordonnées et le timestep :

- gravity : Accélération gravitationnelle (en m/s²). Valeur typique : **9.81**.

- basicTimeStep : Intervalle entre deux calculs de simulation (en ms). **32**

- coordinateSystem : Définit l'orientation des axes X, Y, Z. Par défaut : **EUN** (X=Est, Y=Up, Z=Nord).

- title : Nom de la simulation (affiché dans la fenêtre).

Vérifier ces 4 paramètres. Les actualiser s'il y a lieu.

Mise en pratique – 3

Neoud Viewport

Modifions des propriétés

- Le nœud Viewpoint définit la position et l'orientation de la caméra par défaut au démarrage de la simulation. Il permet de fixer ce que l'utilisateur voit au lancement.
- Paramètres clés du nœud Viewpoint
  - **position** : Coordonnées (x, y, z) de la caméra (EUN).
    → Pour être 1 m au-dessus du centre : position 0 2 1.
  - **orientation** : Rotation de la caméra sous la forme (x, y, z, angle) :
    (x, y, z) = axe de rotation
    angle = angle en radians
    Exemple pour regarder vers le bas : orientation 1 0 0 -1.57 (rotation de -90° autour de X)
  - **fieldOfView** : Champ de vision en radians (ex. 0.7854 ≈ 45°).
    → Plus petit = zoom avant, plus grand = vision plus large.

Vérifier ces 4 paramètres. Les actualiser s'il y a lieu.

Mise en pratique – 4

Nœud RectangleArena

Modifions des propriétés

Ce nœud permet de créer **une surface plane rectangulaire**, souvent utilisée comme sol pour des simulations mobiles (robots, objets...). Quelques paramètres :

- **floorsize** : Dimensions de l'arène (largeur, longueur) en mètres → Exemple : size 1 1 = carré de 1 m x1 m
- **floorTileSize** : Taille des dalles de sol → Exemple : floorTileSize 0.25
- **wallHeight** : Hauteur des murs autour de l'arène. → 0 = pas de mur

Vérifier / modifier pour tenir compte des valeurs suivants

- **floorTileSize** 0.25 0.25
- **wallHeight** champ et changer sa valeur à 0.05

Mise en pratique – 5

Ajout d’un objet à la scène

Modifions des propriétés du noeud RectangleArena

- Double-cliquez sur le RectangleArena dans l'arborescence de la scène pour la fermer et la sélectionner.
- Cliquez sur le Add bouton en haut de l'arbre de scène.
- Dans la boîte de dialogue ouverte, choisissez PROTO nodes (Webots Projects) / objects / factory / containers / WoodenBox (Solid). Une boîte devrait apparaître au milieu de l'arène.
- Double-cliquez dessus dans l'arborescence de la scène pour ouvrir ses champs.

  - Changer size à **0.1 0.1 0.1** au lieu de 0.6 0.6 0.6.

  - Changer translation à **0,25 0.05 -0,25** au lieu de 0 0 0.3. Alternativement, vous pouvez utiliser la flèche bleue qui apparaît dans la vue 3D pour ajuster sa translation sur un axe

  - Maintenant, déplacez-cliquez et faites glisser la boîte dans la vue 3D et déplacez-la dans un coin de l'arène.

  - Sélectionnez WoodenBox créé précédemment, le dupliquer et le positionner à autre endroit (**0,25 , 0,05 et -0,25**).

  - Créez une troisième WoodenBox et la positionner (**-0,25 , 0,05 et 0,25**)
- Déplacez les boîtes, de sorte qu'aucune boîte n'est au centre de l'arène. Vous pouvez également utiliser les flèches de rotation bleues pour faire pivoter les boîtes le long de l'axe vertical. Cela peut être fait aussi par shift-click et glisser avec le bouton droit de la souris. Alternativement, vous pouvez modifier l'angle de la rotation champ de la WoodenBox nœuds dans l'arbre de scène.
- **Sauvegarder World ma_premiere_simulation.**

Mise en pratique – 6

Ajout d’un robot

Ajout du robot e-puck

Le e-puck est un petit robot ayant des roues différentielles, 10 LED, et plusieurs capteurs dont 8 Capteurs de distance et une Caméra. Dans le cas présent, nous allons nous restreindre à l’utilisation de ses roues.

- Assurez-vous que la simulation est en pause et que le temps virtuel écoulé est de 0. Si ce n'est pas le cas, réinitialisez la simulation avec le bouton **Reset**.
- **Lorsqu'un monde Webots est modifié avec l'intention d'être sauvegardé, il est fondamental que la simulation soit d'abord mise en pause et rechargée à son état initial, c'est-à-dire que le compteur de temps virtuel de la barre d'outils principale affiche 0:00:00:000. Sinon à chaque sauvegarde, la position de chaque objet 3D peut accumuler des erreurs. Par conséquent, toute modification du monde doit être effectuée dans cet ordre: mettre en pause, réinitialiser, modifier et enregistrer la simulation.**
- Nous n'avons pas besoin de créer le robot e-puck à partir de zéro, nous aurons juste à importer un noeud E-puck noeud. Ce nœud est en fait un PROTO node, comme RectangleArena ou WoodenBox.
- Le prototypage vous permet de créer des objets personnalisés et de les réutiliser.

- Sélectionnez le dernier nœud WoodenBox dans l'arbre de la scène.

- Cliquez sur le Add bouton en haut de la vue de l'arbre de la scène.

- Dans la boîte de dialogue, choisissez PROTO nodes (Webots Projects) / robots / gctronic / e-puck / E-puck (Robot). Un robot e-puck devrait apparaître au milieu de l'arène.

- Déplacez et faites pivoter ce robot, comme vous l'avez fait avec les WoodenBox.

- Enregistrez la simulation

- Activer la touche Run real-time bouton .

Mise en pratique – 7

Créer un Nouveau Contrôleur

Création d’un contrôleur pour le robot e-puck

- Un contrôleur est un programme qui définit le comportement d'un robot.

- Les contrôleurs Webots peuvent être écrits dans les langages de programmation C, C++, Java, Python, MATLAB, etc.

- Les contrôleurs C, C++ et Java doivent être **compilés** avant de pouvoir être exécutés en tant que contrôleurs de robot.

- Les contrôleurs Python et MATLAB sont implémentés en langages interprétés (sans compilation).

- Le champ controller d'un Robot est un nœud qui spécifie quel contrôleur est associé au robot.

- **Le même contrôleur peut être utilisé par plusieurs robots**, mais un robot ne peut utiliser qu'un seul contrôleur à la fois.

- Chaque contrôleur est exécuté dans un processus enfant séparé généralement engendré par Webots.

- Comme ce sont des processus indépendants, les contrôleurs ne partagent pas le même espace d'adressage et peuvent s'exécuter sur différents cœurs de processeur.

Mise en pratique – 7 ...

Créer un Nouveau Contrôleur

Création d'un contrôleur pour le robot e-puck

- Créer un nouveau contrôleur python appelé epuck_go_forward en utilisant :

  Wizards / New Robot Controller

- Cela va créer un nouveau epuck_go_forward (répertoire dans my_first_simulation/controllers).
- Sélectionnez l'option vous proposant d'ouvrir le fichier source dans l'éditeur de texte.
- Le nouveau fichier source est affiché dans la fenêtre de l'éditeur de texte Webots.
- Ce fichier source peut être compilé (s'il est écrit en C, C++ ou Java) sans aucune modification.
- Associez le nouveau controleur epuck_go_forward au noeud E-puck :
  - Dans l'arborescence de la scène, sélectionnez le champ controller du noeud E-puck,
  - Utilisez l'éditeur de champs en bas de la vue arborescence de la scène:
    - appuyez sur le bouton Select puis sélectionnez epuck_go_forward dans la liste.
- Une fois que le contrôleur est associé au robot, sauvez le monde.
- Modifiez le programme en obtenant les dispositifs du moteur (leftMotor = robot.getDevice('left wheel motor')), et en appliquant une commande moteur (leftMotor.setPosition(10.0)') **(page suivante)**

Résumé des api Webots

Catégorie	**Fonction**	**Description**
Robot	`Robot()`	Crée une instance du robot
	`robot.step(timestep)`	Avance la simulation d'un pas
	`robot.getDevice(name)`	Accède à un dispositif par son nom
	`robot.getBasicTimeStep()`	Retourne le pas de temps (ms)
Capteurs	`device.enable(timestep)`	Active le capteur
	`device.getValue()`	Lit la valeur du capteur
Moteurs	`motor.setPosition(pos)`	Définit la position cible
	`motor.setVelocity(vel)`	Définit la vitesse du moteur
Caméra	`camera.enable(timestep)`	Active la caméra
	`camera.getImage()`	Récupère l'image brute
Communication	`emitter.send(message)`	Envoie un message
	`receiver.getData()`	Lit un message reçu
Temps	`robot.getTime()`	Temps écoulé dans la simulation
LED	`led.set(value)`	Allume/éteint la LED (0=éteint, 1=allumé)
	`led.setColor([r, g, b])`	Définit la couleur (RVB, valeurs 0-255)

Notes:
- **Structure** : Initialiser `robot = Robot()`, activer capteurs/LED avec `enable(timestep)`, boucle avec `robot.step(timestep)`.
- **LED** : Accéder via `led = robot.getDevice('led_name')`. Vérifiez le nom dans le fichier `.wbt`.
- **Conseils** : Vérifiez les noms des dispositifs dans `.wbt`, utilisez `print()` pour déboguer, désactivez capteurs inutiles avec `disable()`.

Mise en pratique – 7 ...

Créer un Nouveau Contrôleur

Exemple de contrôleur en python pour le robot e-puck

```
from controller import Robot

TIME_STEP = 64
# création de l'instance du Robot
robot = Robot()
# affecter chaque moteur (droit et gauche)
leftMotor = robot.getDevice('left wheel motor')
rightMotor = robot.getDevice('right wheel motor')
# Fixer la destination
leftMotor.setPosition(10.0)
rightMotor.setPosition(10.0)
# Boucle d'exécution
while robot.step(TIME_STEP) != -1:
    pass
```

Exercice 1

- Adaptez le programme précédent pour afficher le distance parcourue par le robot

Solution exercice 1

```
from controller import Robot
TIME_STEMP = 64
WHEEL_RADIUS = 0.0205  # rayon roues e-puck en m
robot = Robot()
leftMotor = robot.getDevice('left wheel motor')
rightMotor = robot.getDevice('right wheel motor')
leftMotor.setPosition(10.0)
rightMotor.setPosition(10.0)
leftSensor = robot.getDevice('left wheel sensor')
rightSensor = robot.getDevice('right wheel sensor')
leftSensor.enable(TIME_STEP)
rightSensor.enable(TIME_STEP)
print("Démarrage de la simulation ...")
# Initialisation
initial_left = 0.0
initial_right = 0.0
initialized = False
final_distance = 0.0
```

```
while robot.step(TIME_STEP) != -1:
    left_pos = leftSensor.getValue()
    right_pos = rightSensor.getValue()
    if not initialized:
        initial_left = left_pos
        initial_right = right_pos
        initialized = True
        continue
    # Vérifie si le mouvement est terminé (proche de la cible 10.0)
    if abs(left_pos - 10.0) < 0.01 and abs(right_pos - 10.0) < 0.01:
        # Calcul de la distance parcourue
        delta_left = left_pos - initial_left
        delta_right = right_pos - initial_right
        avg_rotation = (delta_left + delta_right) / 2.0
        final_distance = avg_rotation * WHEEL_RADIUS
        break  # quitte la boucle

# Affichage final
print(f"Distance totale parcourue : {final_distance:.3f} m")
```

Exercice 2

- Remplacer le robot e-puck par le robot Thymio.
- Changer le fond de rectangleArena tenant compte du fichier « paper.jpg »