TME SOLO

Dans ce TME SOLO, vous pouvez (et devez) réutiliser tout ce que vous avez fait durant le projet. Lorsque l'on parle de centimètre dans l'énoncé, il s'agit d'une unité de mesure constante quelconque dans votre arène simulée.

Avant de commencer le TME SOLO, créez une nouvelle branche dans votre dépôt `github` qui porte votre nom. Vous travaillerez uniquement dans cette branche. A la fin du TME, vous enverrez un email à `nicolas.baskiotis@sorbonne-universite.fr` qui contiendra l'adresse du `github`, le nom de la branche et un petit descriptif de ce que vous avez fait **en indiquant obligatoirement les questions auxquelles vous avez répondu et quels fichiers ont été modifiés**. En dernier recours, si vous n'arrivez pas à sauver votre travail dans la branche, vous enverrez par email votre code.

Dans un fichier `tmesolo.py` à la racine de votre dépôt, vous regrouperez les fonctions `qX.Y()` qui permettent d'exécuter le code correspondant aux questions `X.Y` ainsi que les affichages graphiques si vous en avez.

Les parties sont indépendantes, vous pouvez les traiter dans l'ordre que vous voulez. Si votre plateforme ne dispose pas de certaines fonctionnalités, vous pouvez les supposer implémentées.

Exercice 1 – Modifications graphiques

Q 1.1 Changer la couleur de votre robot afin qu'elle soit rose (par exemple `#FD6C9E` en codage RGB hexadécimal).

Q 1.2 Le robot a en fait 2 leds qui peuvent être allumées ou éteintes avec la commande `set_led(ID_LED,statut)` de la classe `Robot2IN013` : `ID_LED` est l'identifiant de la led (1 pour la première led, 2 pour la deuxième), et `statut` un booléen (vrai pour led allumée, faux pour led éteinte). Elles sont positionnées à gauche et à droite au devant du robot. La première à gauche est rouge, la deuxième à droite est bleue.

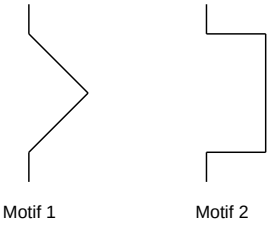
Q 1.2.1 Modifiez votre simulateur pour ajouter cette fonctionnalité.

Q 1.2.2 Pour démonstration, faites faire au robot une ligne droite qu'il parcourt en alternant la led allumée.

Exercice 2 – Stratégies Simples

Q 2.1 Implémentez une stratégie qui permet d'effectuer le motif 1.

Q 2.2 Implémentez une stratégie qui permet d'effectuer le motif 2.



Q 2.3 Implémentez une stratégie qui permet d'effectuer le motif 1 de manière répétée jusqu'à rencontrer un obstacle, puis de faire un demi-tour est de repartir en exécutant le même motif et ainsi de suite.

Q 2.4 Implémentez une stratégie qui permet à partir d'une liste de motifs (du type 1 et 2 par exemple), d'effectuer le premier motif de la liste jusqu'à rencontrer un obstacle, faire demi-tour, puis d'exécuter le deuxième motif jusqu'à rencontrer un obstacle, faire demi-tour et ainsi de suite en parcourant tous les motifs de la liste.

Exercice 3 – Stratégie avancée et modification de l'arène

On considère que l'arène dispose d'un nouveau type d'objets, des gemmes que le robot doit ramasser. La gemme émet un signal que le robot peut capter à l'aide d'une fonction `get_distance_gemme()` disponible dans l'API. Cette fonction retourne la distance à laquelle se trouve la gemme par rapport au robot (on suppose pour l'instant qu'une gemme dans l'arène). Si le robot atteint la gemme, le robot ramasse la gemme et celle-ci disparaît et une autre gemme apparaît aléatoirement.

Q 3.1 Modifiez votre arène pour inclure la génération de gemmes.

Q 3.2 Modifiez vos classes pour inclure la fonction `get_distance_gemme()`.

Q 3.3 Proposez une stratégie pour que le robot ramasse le plus de gemmes possible le plus rapidement possible.

Q 3.4 Les gemmes sont en fait mobiles : elles se déplacent en ligne droite selon un angle tiré aléatoirement lors de leur initialisation. Lorsqu'un obstacle est rencontré, elles rebondissent et font demi-tour. Toutes les 10 secondes, un nouvel angle est tiré aléatoirement et leur direction change. Implémentez les changements.