Capacités de la simulation en Python et Unity

Simulation en Python

Après avoir codé des stratégies en python, on veut pouvoir les tester et observer le résultat : c’est le but des simulations.

Nos simulations sont réalisées à l’aide des diverses stratégies que nous avons écrites.

Les principales stratégies sont *AvancerDroit* et *Tourner*, qui permettent, comme leur nom l’indique, de faire avancer le robot et que ce dernier puisse tourner d’un certain angle. A l’aide de ces deux stratégies, nous en avons développé d’autres. La première stratégie, *StrategiePolygone*, permet au robot de dessiner le polygone souhaité; la deuxième, *StrategieAvancerDroitMax*, fait progresser le robot le plus vite possible vers un mur et l’arrête juste avant l’impact. La stratégie *strategieIA* permet, elle, au robot de contourner un objet; en effet, lorsqu’il y a un objet devant lui, le robot effectue une rotation afin d'éviter l’obstacle. Enfin, une stratégie *SuivreBalise* fait tourner le robot sur lui-même jusqu’à détecter la balise, et une fois qu’il l’a trouvé, le fait s’avancer vers elle.

Pour implémenter une simulation, on initialise donc les stratégies utilisées, puis, à l’aide de threads, on met à jour les informations, ce qui nous permettra de voir au fur et à mesure les instructions réalisées.

Nous avons mis en oeuvre deux étapes de validation de nos stratégies.

La première étape consiste à utiliser un simulateur qui exécutera les différentes stratégies et permettra de représenter sur le terminal en 2D le déplacement du robot dans un terrain continu prédéfini. On déclare donc un robot et un terrain continu virtuels sur lequel se déplacera le robot et on peut alors vérifier le bon fonctionnement d’ensemble mais cela ne reste qu’une simulation et quand tout est opérationnel, il faut alors passer à la deuxième étape, qui est de tester notre code directement sur le robot.

Ce dernier se déplace dans le monde physique dans lequel il se trouve.

Il n’est pas possible d’utiliser le même code sur le simulateur et sur le robot. On devra donc avoir deux codes distincts en réécrivant des fonctions distinctes pour chaque stratégie pour pouvoir bénéficier des deux représentations (simulée ou IRL).

On définit donc un Wrapper qui va servir à traduire le code de la simulation pour que le robot IRL comprenne les différentes tâches à réaliser. Pour cela, le Wrapper utilisera directement les commandes du robot.

Représentation Graphique à l’aide de Unity :

Pour obtenir une représentation virtuelle en 3 dimensions de notre robot, nous avons utilisé le logiciel Unity.

Pour pouvoir observer le déplacement du robot, on lance tout d’abord une simulation Python, dans laquelle notre robot peut réaliser diverses stratégies (ici, se déplacer de telle sorte à dessiner un carré).

Ensuite, on doit pouvoir récupérer les différentes informations de la simulation et les transmettre à Unity. Pour effectuer cela, nous utilisons un “protocole de contrôle de transmissions“, autrement dit, une connexion TCP. Cette dernière permet à la simulation de communiquer les données nécessaires à l'exécution du script dans Unity.

On envoie régulièrement les informations de la simulation que l’on récupère dans unity grâce à l’utilisation de threads.

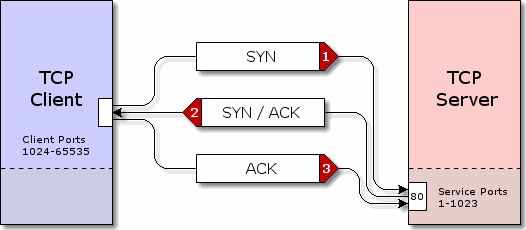
Pour représenter le déroulement de la simulation, on crée les classes qui stockeront les données des différents objets. La deserialisation se fait à l’aide de Json, par le biais de la connexion TCP.

Après cette étape, on dispose de tout ce dont on a besoin pour commencer la représentation graphique.

On instancie tout d’abord le terrain, puis les différents éléments qui le composent tels que le robot ainsi que les obstacles. Les dimensions de ces objets définies au lancement de la simulation sont conservées. Le terrain est représenté par un plan, le robot par une sphère, et les obstacles par des cubes. Nous utilisons des objets simples pour la représentation. Nous n’avons pas modélisé un véritable robot qui évoluerait dans un monde avec les propriétés physiques du monde actuel.

Pour que la représentation graphique se fasse, il faut dans un premier temps lancer la simulation sur python, puis passer sur unity et y lancer le script.

Schéma descriptif d’une connexion tcp :



SYN = demande de synchronisation/connexion

ACK = accusé de réception

La connexion peut se faire sur un même poste, ce qui est notre cas ici.