PSC : Group Info : Algorithmique du robot : mouvement et évitement d’obstacles.

Etape 1 : Familiarisation avec l’environnement de simulation Vrep

* Lecture de la documentation : interface et fonctionnement du logiciel : objets mobile / non mobiles , scenes et views , proprietes d’objets (collidable/ detectable …) , capteurs (camera, capteur IR ..) , code (sim. …) …
* Simulation de la table de jeu sur le logiciel
* Connection remote en utilisant python : commander des objets Vrep en utilisant du code python
* Reste à faire : simulation du robot

Etape 2 : Architechture du code :

* Contrats avec l’equipe asservisment et l’equipe camera : input de type text => traitement => retour de type text à l’equipe asservisemnt pour leurs donner le prochain mouvement à faire
* Traitement : le code central traitera les données recus pour savoir la position du robot , la position du (des) robots adverse et la position des atomes + l’orientation du robot puis à l’aide d’un graph (discritisation de la table : mouvement du robot discret) => calcul du plus court chemin vers la cible (qui sera une tache choisie par priotrité de nombres de point qu’elle fait gagner : Priority queue) => si il y a pas de chemin passer à la tache suivante… puis passer l’information :’prochaine etape’ à l’asservissment : pk ce choix : question d’independance et de modularité : la respberry pense et l’arduino fait et donne un retour simple de positions et de capteurs.
* Apres ecriture du code : essaie dans le robot Vrep en ecrivant un fichier connect\_Vrep …
* Github : creation d’organisation pour faciliter le travail …

Etape 3 : choix de l’algorithme de calcul du chemin

* 1ere tentative : Dijkestra vu en cours d’INF 421 : defaut : un peu long et on aura besoin de vitesse car chaque pas necessite le calcul d’un plus court chemin
* 2eme tentative : puisque on a besoin à chaque fois juste d’envoyer la prochaine etape à faire : j’ai pensé à utiliser un algorithme de type Greedy : apres recherche j’ai abondoné l’idée car le calcul du chemin a besoin d’etre tres precit pour faire gagner du temps et parfois les greedy algorithms partent dans des pistes qui rendent les choses plus lentes par la suite
* 3eme tentative : A\* : tres reputé et connu dans ce domaine : j’ai trouvé le probleme que les chemins caculées par cette methode sont parfois un peu coupés // => robot besoin du temps pour tourner et pour accelerer ..
* Still looking for the optimal algorithm (idée de s’inspirer d’un algo pour ecrire un autre qui sera bien adapté à la table)