Pour mettre en place l'environnement logiciel de travail utilisé dans ce TME, suivez la procédure décrite ci-après:

1. télécharger le simulateur (roborobo3_iar_2018.tgz)
   http://pages.isir.upmc.fr/~bredeche/Teaching/IAR/Ressources
2. dans un répertoire ou vous aurez placer l'archive du simulateur, désarchivez:
   tar xvfz roborobo3_iar_2018.tgz
3. dans le répertoire du simulateur, installez et compilez le simulateur:
   make -j8
4. pour tester:
   ./roborobo -l config/tutorial.properties
   Remarques: tapez « h » dans la fenêtre de simulation pour avoir de l'aide. Tapez « d » pour accélérer la simulation (3 modes). Tapez "f" pour activer le suivi d'un agent en particulier. Tapez "<tab>" pour passer d'un agent à l'autre.

Au cours de ce TME, vous allez utiliser plusieurs projets. Chaque projet correspond à un point de départ pour un exercice. Les projets se trouvent dans le répertoire **prj/** et sont nommés *TME_(...)*.

Vous ne modifierez que les classes *(...)Controller(.h/.cpp)* et *(...)WorldObserver(.h/.cpp)*. ***La classe (...)Controller*** *permet de programmer le comportement d'un robot,* ***la classe (...)WorldObserver*** *permet de manipuler l'environnement,* Il est possible de faire tous le sujet en modifiant uniquement les méthodes **step()** de ces deux classes, et (éventuellement) le constructeur de la classe (...)Controller.

Les robots dont vous disposez ont 12 senseurs, positionnés comme indiqué sur l'image ci-dessous. L'avant du robot dispose de plus de senseurs.

Il s'agit d'un programme en C++, cependant il n'est pas nécessaire de maîtriser le C++ pour faire le TME. **Tous les éléments nécessaires** pour vous aider à répondre aux questions se trouvent dans l'exemple que fourni le projet qui se trouve dans le répertoire **./prj/Tutorial**. Pour commencer, étudiez le code de la fonction **step()** de la classe *TutorialController*. Cette méthode *step()* vous montre comment écrire un comportement dépendant des entrées sensorielles (les valeurs de effecteurs dépendent d'une combinaison linéaire des entrées), et illustre comment accéder aux données capteurs qui vous seront utiles au cours du TME.

En plus des classes mentionnées, vous pouvez modifier le fichier de paramètres **TME(...).properties** du répertoire **./config/** — ce fichier vous permet de changer facilement les paramètres de votre simulation, par exemple le nombre de robots (par défaut: *gInitialNumberOfRobots vaut* 200), voire les obstacles présents ou la carte de l'environnement. Il n'est pas nécessaire de modifier les fichiers properties pour faire le TME.

M2 ANDROIDE, UE IA et Robotique nicolas.bredeche(at)sorbonne-universite.fr
TME ROBOTIQUE COLLECTIVE rev. 2018-10-17

EXERCICE 1: *Braitenberg (prise en main et rappel)*

Projet: <roborobo3>/prj/TMEbraitenberg
Commande: ./roborobo -l config/TMEbraitenberg.properties

Ecrivez un comportement d'évitement d'obstacles. Pour mémoire, les connexions dans un robot de Braitenberg sont excitatrices *ou* inhibitrices (ie. pas de seuils).

Evaluation: implémentation, identification de cas difficiles.

EXERCICE 2 : *Agrégation*

Projet: <roborobo3>/prj/TMEaggregation
Commande: ./roborobo -l config/TMEaggregation.properties

Ecrivez un comportement d'agrégation. Chaque robot doit se diriger vers le robot le plus proche. Vous pouvez augmenter le nombre de robots (*gInitialNumberOfRobots*) en éditant votre fichier *properties* dans le répertoire *config/.*

Evaluation: avec la touche « entrée » vous pouvez prendre le contrôle d'un robot. Déplacer vous afin de perturber le comportement d'agrégation. Si votre robot est bloqué, vous pouvez changer de robot avec la touche tabulation (shift+tab pour revenir au précédent).

EXERCICE 3 : *Dispersion*

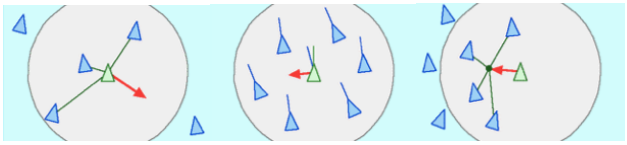
Projet: <roborobo3>/prj/TMEdispersion
Commande: ./roborobo -l config/TMEdispersion.properties

Ecrivez un comportement de dispersion. Chaque robot doit s'éloigner du robot le plus proche. Lorsqu'il ne voit rien, il tourne sur lui même.

Evaluation: avec la touche « entrée » vous pouvez prendre le contrôle d'un robot. Déplacer vous afin de perturber le comportement de dispersion.

EXERCICE 4 : *Les Boïds -- Boucles de retro-action positives et négatives*

Projet: <roborobo3>/prj/TMEboids
Commande: ./roborobo -l config/TMEboids.properties



Le comportement de répulsion fait prendre une direction qui éloigne du centre de masse
Le comportement d'attraction fait prendre une direction rapprochant du centre de masse
Le comportement d'orientation fait prendre une direction identique aux voisins.

a. Vous devez maintenant implémenter un algorithme de type boids sur chaque robot (cf. image ci-dessus). Implémentez et testez *séparément* les comportements d'orientation, d'attraction et de répulsion. Puis tous ensemble en réglant les seuils.

b. La présence des murs n'est *(a priori)* pas gérée par les boids. Modifier votre programme en ajoutant le comportement d'évitement d'obstacles fait à l'exercice 1. Ce comportement sera activé uniquement lorsqu'un robot est proche d'un mur.

c. Etudiez l'influence des différents paramètres permettant d'équilibrer les comportements d'attraction, répulsion et orientation.

Evaluation:
(a) Montrez séparément chaque comportement (répulsion, attraction, orientation)
(b) Montrez l'intérêt d'ajouter le comportement d'évitement d'obstacles
(c) Trouver les paramètres d'application des règles favorisant les groupes

Bonus : en prenant le contrôle d'un robot, déplacer vous afin de « recruter » un maximum de Boids.

Ce qui suit est facultatif -- à faire uniquement si vous avez du temps.

EXERCICE 5 : *Robotique collective et « embodied evolution » [facultatif]*

Projet: <roborobo3>/prj/TMEevolution
Commande: roborobo -l config/TMEevolution.properties

l'algorithme ci-dessous (mEDEA) est inspiré du processus de sélection naturelle. C'est un algorithme d'évolution artificielle qui *n'utilise pas de fonction fitness*. Les individus les plus aptes à diffuser leur génome (ie. ceux qui rencontrent le plus de partenaires) génèrent, par définition, plus de copies d'eux-mêmes que les autres. Pour fonctionner, cet algorithme a besoin de grande taille de population (p.ex. 200).

```
genome.randomInitialize()
while forever do
  if genome.notEmpty() then
    agent.load(genome)
  end if
  for iteration = 0 to lifetime do
    if agent.energy > 0 and genome.notEmpty() then
      agent.move()
      broadcast(genome)
    end if
  end for
  genome.empty()
  if genomeList.size > 0 then
    genome = applyVariation(select_random(genomeList))
  end if
  genomeList.empty()
end while
```

a. Implémentez cet algorithme. On considère qu'un robot sans génome à la fin d'une génération devient inactif. Il n'y a pas de re-localisation à la fin de l'évaluation - c'est à dire que l'algorithme est distribué et que le superviseur humain n'intervient pas.
b. Observez le résultat en terme de dynamique comportementale, et de nombre de robots «survivants», c'est à dire qui dispose toujours d'un génome au début d'une génération (i.e. nombre de rencontres non nul pendant la précédente génération).
c. Modifiez votre algorithme pour inclure une fonction fitness à maximiser. Cette fonction comportera trois termes: max(nombre de voisins), max(vitesse de translation), min(vitesse de rotation). Comment garantir que les performances soient comparables?