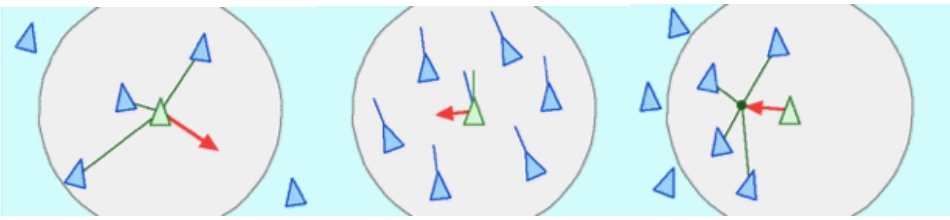
EXERCICE 1 : *Les Boïds -- Boucles de retro-action positives et négatives*

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. Il s'agit ici de fixer les conditions où s'applique chacun des trois comportements (i.e. distance min et max du barycentre des voisins par rapport à la position du robot focal).

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

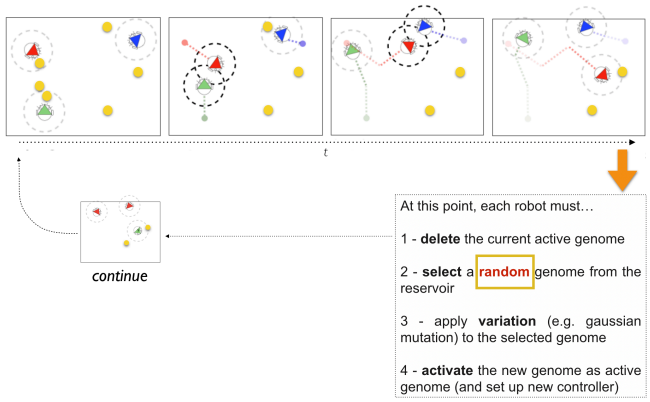
Bonus : en prenant le contrôle d'un robot, déplacer vous afin de « recruter » un maximum de Boids.

EXERCICE 2 : *Robotique en essaim et apprentissage en ligne distribué*

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 robots, voire plus).

```
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.
   i. contrôleur d'un robot: on utilise un perceptron, c'est à dire que les sorties motrices sont données par une combinaison linéaire des entrées sensorielles (+ un biais), à laquelle on appliquer un fonction d'activation tangente hyperbolique. Les entrées sensorielles seront limitées aux *senseurs de proximité* (distance au plus proche obstacle).
   ii. au démarrage de la simulation, chaque robot initialise au hasard les poids du perceptron entre -10 et +10.
   iii. à chaque pas de temps, le robot transmet à tous ses voisins les poids du perceptron, et stocke les génomes reçus. Aide: méthodes *sendMessage* et *receiveMessage*).
   iv. lors de la sélection, on appliquera ensuite une mutation sous la forme d'un bruit gaussien sur l'ensemble des paramètres, avec un écart type très faible (p.ex. 10^{-6}, voire moins). Aide: méthode *getBoundedGaussianMutatedValue* dans *<roborobo_basedir>/src/core/Misc.cpp*
b. Testez plusieurs valeurs de paramètres de contrôle: taille de la population, amplitude de la mutation gaussienne. Observez les stratégies obtenues.
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). Vous utiliserez une fenêtre glissante pour avoir une approximation de la performance de chaque robot (i.e. le score est moyenné sur *n* itérations, pendant les premières *n* itérations le robot ne peut pas transmettre ni recevoir de génomes).