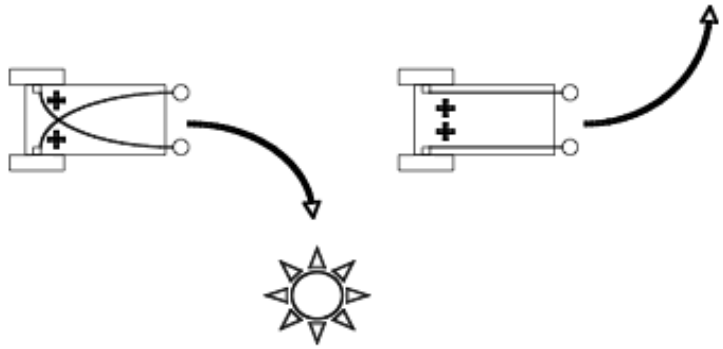
nicolas.bredeche(at)isir.upmc.fr

L'ensemble des exercices qui suivent sont à réaliser avec la version s*imbad-1.4.1.jar*. Le code source *Exemple.java* (livré avec *Misc.java*) montre comment connaître la position et l'orientation d'un robot, comment calculer une distance entre robots et comment accéder à l'état d'un robot depuis un autre.

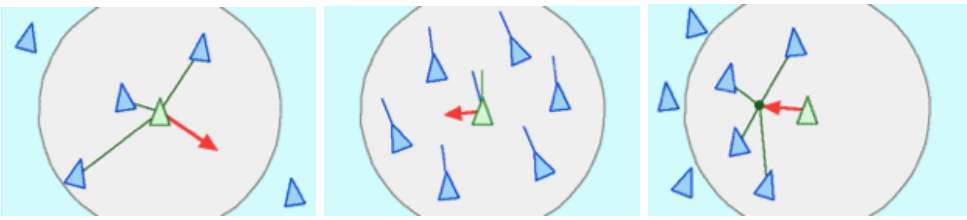
Remarque: la seule différence avec l'archive de Simbad disponible sur sourceforce et celle donnée pour ce TP (simbad-1.4.1.jar) est le changement de la visibilité de la variable rotation de type Transform3D que se trouve dans BaseObject.java. La visibilité doit être mise en mode «public», afin que l'orientation par rapport à l'axe y puisse être connue.

EXERCICE 1 : plusieurs véhicules de Braitenberg



En partant du TP précédent, vous devez créer 10 robots qui devront éviter les obstacles (comportement de Braitenberg d'évitement d'obstacles, une variation des comportements donnés en exemple ci-dessus, et utilisant les senseurs de proximité plutôt que de lumière). Observez les résultats.

EXERCICE 2 : les Boids



a. Vous devez maintenant implémenter un algorithme de type boids sur chaque robot (cf. image du dessus). Observez les résultats.

b. Le fait que l'environnement comporte des murs pose problème. En effet, les robots risquent de rentrer en collision dans les murs et de rester bloquer. Pour éviter cela, intégrer le comportement d'évitement d'obstacle codé précédemment et le comportement de boids dans une architecture de subsomption simple. Par défaut, les comportements boids seront utilisés, et si un obstacle est trop proche, le comportement d'évitement prendra le dessus. Testez plusieurs paramètres (ie. rayons) pour les domaines d'applications des comportement boids afin d'obtenir plusieurs types de comportements.

nicolas.bredeche(at)isir.upmc.fr

EXERCICE 3 : évolution en ligne

Voici l'algorithme dit de (1+1)-online, qui permet d'optimiser en ligne un comportement pour un robot seul.

for $evaluation = 0$ to N **do**
  if random() $< P_{reevaluate}$ **then**
    Recover(Champion)
    $Fitness_{Champion}$ = RunAndEvaluate(Champion)
  else
    Challenger = Champion + $N(0, \sigma)$ {Gaussian mutation}
    Recover(Challenger)
    $Fitness_{Challenger}$ = RunAndEvaluate(Challenger)
    if $Fitness_{Challenger} > Fitness_{Champion}$ **then**
      Champion = Challenger
      $Fitness_{Champion} = Fitness_{Challenger}$
      $\sigma = \sigma_{min}$
    else
      $\sigma = \sigma \cdot 2$
    end if
  end if
end for

Pour cet exercice, on utilisera la fonction fitness donnée ci-dessous. Celle-ci a pour but de favoriser les comportements qui se déplacent rapidement et en ligne droite.

$$fitness = \sum_{t=0}^{evalTime} (v_t * (1 - v_r) * (minSensorValue))$$

Dans cet exercice, comme dans le suivant, on considère que

a. Implémentez cet algorithme et testez le avec 10 robots dans l'environnement. Evaluez la vitesse de convergence et la performance.

b. Modifiez cet algorithme afin que des robots qui passent l'un à coté de l'autre puisse émettre leur genome «champion». Un génome champion importé remplace le génome champion locale seulement s'il est meilleur. Comparer ce nouvel algorithme au précédent, en terme de vitesse de convergence, de robustesse et de performance.

*Remarque: Random.nextGaussian vous permet de générer des nombres aléatoires issues de la distribution N(0,1). Pour mémoire: N(0,sigma) = sigma*N(0,1)*

EXERCICE 4 : Robotique collective auto-adaptative

Voici l'algorithme mEDEA, un algorithme distribué pour l'auto-adaptation d'essaim de robots à un environnement inconnu. Codez cet algorithme.

```
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
```

Observez le résultat en terme 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). Vous étudierez l'influence des différents paramètres, en particulier la durée d'une génération.