TME 2
Proies et Prédateurs

Dans ce TD et TME, nous continuons l'exploration du *framework* Tamago, avec notamment les notions de composants composites et d'interfaces de contrôle (ou contrôleurs).

Le point de départ de ce TME est l'archive `TME2-Maleva-skel.jar.tgz` disponible sur le site de l'UE

Maleva, développé à l'origine par Thomas Meurisse, est un système très simples d'agents proies & prédateurs dans lequel chaque agent est conçu par assemblage de comportements élémentaires, de capteurs et d'effecteurs sur l'environnement. Chaque agent effectue un cycle comportement à chaque pas de simulation (cf. contrôleur `LCStepperControler`).

EXERCICE : COMPOSANTS MALEVA

Question 1 : Architecture logicielle

Parcourir les sources de Maleva pour en extraire les composants, les composites (composants contenant un assemblage) et les assemblages. Dans un document *UMLet* réaliser un diagramme de composant pour chaque composant Maleva. Dans un second document décrire le composant composite `Mechant` avec son assemblage interne et ses services requis et fournis. Dans un dernier document, réaliser l'assemblage du programme principal `MalevaTest1`.

A ce stade, l'architecture logicielle du simulateur ne doit plus avoir de secret pour vous.

Remarque : le code source exploite allègrement l'héritage, il faut considérer toute la hiérarchie pour comprendre l'interface complète d'un composant.

Important : on ne demande pas de diagramme de classe dans cette question.

Question 2 : Déplacement aléatoire

Dans Maleva, les comportements canoniques des agents héritent de la classe abstraite `ComportementAgent`.

En vous inspirant des comportements existants, décrire un comportement supplémentaire `RandomMouvement` qui déplace un agent au hasard. Ce composant génère dans sa méthode `step()` un déplacement élémentaire dont l'angle est calculé à partir de l'angle courant de l'agent (`outer.getAngle()`) auquel on ajoute une valeur prise au hasard en -angle_step et +angle_step.

Implémenter ce comportement.

Question 3 : Agent trouillard

Décrire l'architecture d'une classe d'« agents trouillards » dans le fichier `Trouillard.java`. Un agent trouillard fuit les prédateurs s'il en voit, ou alors évolue au hasard.
On pourra tester l'architecture avec la cible `execute` du fichier de construction build.xml.