# Devoir Maison
# Systèmes Multi-Agents
# Architecture Logicielle

L'objectif de cette petite épreuve est de vous donner les moyens de montrer vos compétences en ce qui concerne les systèmes multi-agents, l'architecture logicielle et la programmation par composants. Pour cette dernière partie, nous vous conseillons d'utiliser le plug-in MAY.

Le résultat attendu est une architecture logicielle pour implanter des systèmes multi-agents (et en particulier le cas d'étude ci-dessous). Les livrables demandés sont :

1. Un document synthétique de réponse au questionnaire, commun à toute la classe. De sa qualité dépendra la note maximum de la partie pratique pour chacun des groupes.
2. Le code source de votre solution sous la forme d'un lien vers une archive.
3. Un manuel utilisateur décrivant comment faire une démo de votre résultat.
4. Un document d'architecture très détaillé qui décrit précisément comment chaque exigence est prise en compte.

Pour vous aider ou pour approfondir vos connaissances sur le type de SMA que nous vous proposons d'implanter, vous avez à disposition un certain nombre de ressources sur le site Moodle dédié à cet enseignement (http://moodle.univ-tlse3.fr/course/view.php?id=2113). Par ailleurs, nous avons préparé un forum sur lequel vous pourrez poser des questions et chercher de l'aide de la part de vos collègues ou d'enseignants/chercheurs du domaine.

Pour la réalisation de votre solution, vous serez assistés en séances présentielles les 30 avril, 7 mai et 28 mai.

Date limite proposée de dépôt des livrables sur le dépôt Moodle : samedi 6 juin 2015 à 19h00.

## Cas d'étude

Pour rendre concret la conception et le développement qui vous sont demandés, voici un exemple de SMA que vous devrez implanter pour valider vos résultats selon deux stratégies : traditionnelle et coopérative.

Il s'agit de robots élémentaires, dont l'objectif consiste à rapporter toutes les boites de couleurs (rouges, vertes, ou bleues) dans des zones spécifiques de l'environnement, appelées « nids ». Chaque nid ne peut accueillir qu'une couleur de boite (rouge, verte ou bleue). Le système est composé de robots de couleur (rouge, vert, ou bleu) qui possèdent les compétences nécessaires à la résolution du problème (perceptions, actions, communication, mécanisme de coopération, etc.). Chaque robot est doté d'une quantité initiale d'énergie qu'il consomme à chaque pas de déplacement. Chaque robot doit trouver, transporter et déposer une boite dans le nid ayant la même couleur que la boîte. A chaque dépôt de boite, le robot reçoit une récompense sous forme d'énergie lui permettant de rester efficace plus longtemps. La valeur de la récompense dépend de la couleur de la boite ramenée par rapport à la couleur du robot.

L'environnement est composé de 3 types d'objets différents :

- La grille de déplacement : Elle est composée arbitrairement de 2500 cellules carrées (50x50) qui peuvent contenir plusieurs composants (boite, robot, nid) ou être vides.

- Les nids : Ce sont les objets dans lesquels les robots déposent les boites. Il en existe 3 : un rouge, un vert, un bleu et sont disposés à équidistance les uns des autres afin de ne pas introduire de biais lié à la proximité de deux nids dans les simulations.

- Les boites : Elles sont transportées et déposées par les robots dans les nids. Elles sont de trois types distincts : rouge, vert et bleu. Elles apparaissent de manière aléatoire sur la grille à intervalle de temps régulier.

CoCaRo est composé d'un type d'agents, paramétré par leur couleur: les robots rouges, les robots verts et les robots bleus. Chaque agent est capable de transporter tous types de boites mais un robot obtient une meilleure récompense s'il ramène une boite de sa couleur. Les agents sont capables des perceptions suivantes et sont dotés des attributs suivants.

- Perceptions : Les agents *robots* ont la capacité de percevoir l'ensemble des objets (autres robots et/ou boites) autour de lui et les coordonnées de sa position sur la grille. Ils connaissent les positions des différents nids à tout instant.
- Attributs : L'agent possède les attributs suivants : (*i*) un niveau d'énergie, (*ii*) une vitesse de déplacement.

Le niveau d'énergie NE est plafonné par un maximum de MaxNRJ unités. La valeur de récompense est égale à 1/3 de MaxNRJ si la couleur de la boite ramenée et celle du nid ne correspondent pas ; elle est égale à 2/3 de MaxNRJ sinon. A chaque action, un robot dépense Conso unités d'énergie.

La vitesse de déplacement dépend du niveau d'énergie NE. La vitesse est de 1 si NE>2/3MasNRJ, 0,5 si NE>1/3MaxNRJ, 0,3 si 1/3MaxNRJ>=NE>0 et 0 si NE=0.

## Exigences

Voici un certain nombre d'exigences. Pour chacune d'entre elles, une valeur/récompense est donnée entre parenthèses.

**ENF 1.** (5) De sa création jusqu'à son suicide, un agent répète un cycle d'exécution composé de trois opérations séquentielles Percevoir-Décider-Agir.

**ENF 2.** (1) Un agent peut créer d'autres agents.

**ENF 3.** (2) Un agent peut se suicider mais ne peut détruire un autre agent.

**ENF 4.** (5) Il doit être possible de visualiser l'exécution du système.

**ENF 5.** (8) Il doit être possible de visualiser l'état du système et/ou d'agents sélectionnés.

**ENF 6.** (8) Le système de visualisation doit être le plus découplé possible du reste du système.

**ENF 7.** (3) Le langage de programmation doit être Java.

**ENF 8.** (8) Le langage de description des composants et des assemblages doit être SpeADL.

**ENF 9.** (5) Vous devez quantifier le nombre de classes et de lignes de code écrites (et pas générées).

**ENF 10.** (13) La réutilisabilité/généricité de votre solution doit être évaluable objectivement, mesurée et la plus forte possible.

**ENF 11.** (5) La paramétrisation de votre solution pour le cas d'étude doit être complètement spécifiée.

**ENF 12.** (8) Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation)

**ENF 13.** (5) Il doit être possible de contrôler l'exécution du système (mettre en pause, pas à pas , vitesse plus ou moins rapide).

**ENF 14.** (8) Il doit être possible d'obtenir une trace d'une exécution d'un agent sous forme de log.

**ENF 15.** (8) Il doit être possible de rejouer une exécution d'un scenario avec un état initial donné, y compris en présence de phénomènes aléatoires.

**ENF 16.** (8) Il doit être possible de persister l'état du système.

**ENF 17.** (21) Il doit être possible de répartir l'exécution du système.