Université de Caen Basse-Normandie
Département d'informatique, U.F.R. Sciences
L3 informatique, 2015–2016
Unité UE2

Génie logiciel

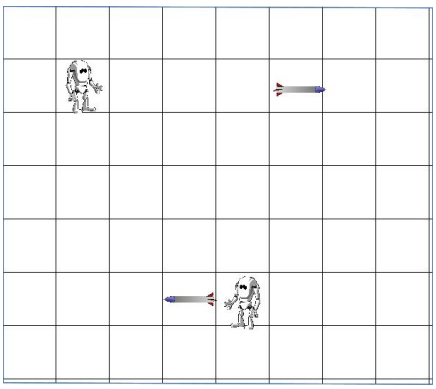
Devoir

Grégory Bonnet, Yann Mathet, Bruno Zanuttini

À rendre pour le **vendredi 27 novembre 2015**.
Ce devoir doit être réalisé par groupes de quatre étudiants. Les groupes devront être formés lors de la deuxième séance de TP (lundi 14 septembre 2015), et un dépôt devra être créé sous subversion pour chaque groupe (obligatoirement comme un sous-projet du projet TP Génie Logiciel L3*) lors de la sixième séance de TP (lundi 12 octobre 2015). Les noms court et long du dépôt devront comporter les noms des quatre membres du groupe, par exemple : nom court* durand-dupont-smith-doe*, et nom long* Génie Logiciel Durand, Dupont, Smith, Doe*. D'autre part, Grégory Bonnet, Yann Mathet et Bruno Zanuttini devront être ajoutés comme managers du dépôt. Si ces points ne sont pas respectés, le devoir ne sera pas corrigé (note de 0). Le 27 novembre, le code de chaque groupe sera extrait de son dépôt pour correction : un répertoire nommé livraison devra être présent à la racine et contenir le code nettoyé (le dépôt pourra contenir d'autres répertoires, non consultés pour la correction). L'énoncé de ce devoir correspond au TP fil rouge. Si vous réalisez les TP au fur et à mesure (en les terminant à chaque fois dans la semaine si besoin), vous aurez de facto terminé le devoir à la fin du neuvième TP de génie logiciel (le lundi 9 novembre 2015)*

Vision synthétique de l'application

Nous souhaitons concevoir une application permettant de jouer des affrontements entre robots sur une grille. Il s'agit d'une variation sur le thème du jeu de stratégie au tour par tour qui pourrait ressembler à cela.

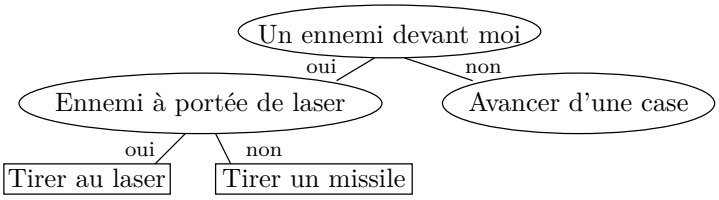


Partant d'une position initiale, chaque robot réalise à tour de rôle une des actions suivantes :

— faire instantanément feu, devant lui, avec un laser qui a une portée limitée,
— lancer un missile, devant lui, qui avance tout droit à chaque tour d'une case,
— avancer, reculer ou tourner de 90 degrés,
— activer un bouclier qui le protège pour ce tour des lasers et des missiles,
— ne rien faire.

Chaque action consomme une énergie du robot. Cette énergie se reconstitue partiellement lorsque le robot ne fait rien. Un robot touché par un laser ou un missile sans être protégé par son bouclier est détruit. La spécificité de l'application est que les robots ne sont pas directement contrôlés par des joueurs humains mais exécutent simplement, à chaque tour, une stratégie scriptée qui leur a été fournie au préalable. Cette stratégie devra en particulier pouvoir être représentée sous la forme d'un arbre de décision.

Un arbre de décision est un arbre binaire dont les nœuds correspondent à des conditions (qui peuvent être vérifiées ou non) et les feuilles des actions à exécuter. Un robot commence à vérifier la condition associée à la racine de l'arbre, puis descend dans son fils de gauche si elle est vraie ou dans son fils de droite si elle est fausse. Le processus est réitéré jusqu'à atteindre une feuille qui indique l'action à exécuter. Par exemple, un arbre de décision (éminemment naïf) peut être :



Notons que l'implantation d'une stratégie aléatoire pour de premiers tests est recommandée.

L'application finale devra posséder :

1. un moteur de jeu permettant de faire combattre les robots et de retourner un gagnant,
2. un volet d'interface graphique représentant l'arène, les robots et les événements du jeu,
3. un volet d'interface graphique représentant la stratégie d'un robot et permettant de la construire ou la modifier en cours d'exécution.

Code et conception

L'application devra être codée en Java. La conception devra comporter trois paquetages isolant (1) le moteur du jeu (robots, actions, règles, etc.), (2) les stratégies et tous les éléments pour les représenter et (3) l'interface graphique. Les méthodes de base des deux premiers paquetages devront être vérifiés par des tests unitaires, fournis avec le code. L'évaluation portera sur la présence d'un code documenté, l'utilisation de patrons de conception appropriés et la propreté générale du code (encapsulation, fonctions atomiques génériques, etc.). Si la qualité esthétique de l'interface graphique ne prime pas, il est nécessaire qu'elle soit fonctionnelle.