M1Info/BIA/2013
TP5-6 – IA & Mindstorm

Nadia Kabachi, Erwan Guillou, Alain Mille,
Marie Lefevre, Aurélien Tabard

28/11/2013 et 5/12/2013

Avertissement : Ce TP sous forme de projet sera noté. Vous devez rendre un Compte Rendu correspondant au plan proposé (fichier : TP5_6_Nom1_Nom2.pdf) ***avant le 18/12/2013 minuit*** accompagné du code source commenté (fichier : TP5_6_Nom1_Nom2.nxc). Le tout sous format électronique envoyé à votre encadrant de TP, avec pour objet de l'email : BIA_TP5_6_Nom1_Nom2.

Dans le CR doivent figurer :
- Les auteurs : Noms Prénoms
- Ce qui a été programmé par rapport au sujet (ce qui tourne et ce qui ne tourne pas)
- Les questions que vous vous êtes posées, les choix que vous avez faits

Objectif du TP : Illustrer le cours et le TD sur l'algorithme A*.

Organisation du travail : Le travail se fait en binôme. La question 1 doit être traitée lors de la première séance. Préparez les questions 2, 3 et 4 à la maison. Tout doit être terminé à la fin de la seconde séance.

Evaluation : Vous ferez vos tests sur un premier labyrinthe. Vous serez évalué sur un autre labyrinthe. Votre code doit donc être générique.

L'objectif du TP est de programmer un robot Mindstorm (http://mindstorms.lego.com) pour que celui-ci se promène intelligemment dans un labyrinthe.

Vous disposez pour cela :
- d'un robot muni de roues, d'un capteur trois couleurs et de deux moteurs permettant d'actionner les roues (cf. Figure 1) ;
- d'un labyrinthe où les traits noirs sont les chemins possibles, les cases bleues des intersections, les cases vertes des obstacles impossibles à franchir et la case rouge la sortie (cf. Figure 2) ;
- d'un code source de base permettant de manipuler le robot (*BIA-TP-5-6-prog.nxc*) ;
- d'une fiche indiquant les bases de la programmation en NXC (*BIA-AideMemoire.pdf*).

Fig1 : Robot Mindstorm

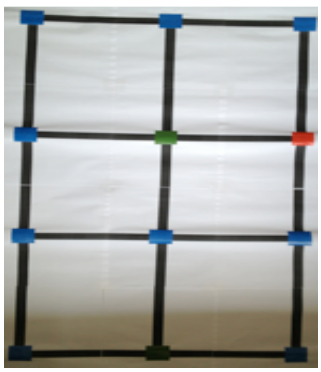

Fig2 : Labyrinthe

Coder sur un robot Mindstorm

Pour programmer sur un lego Mindstorm, plusieurs solutions sont possibles. Durant ce TP, nous utiliserez un éditeur de texte classique et vous compilerez en ligne de commande.

Pour cela, suivez la procédure suivante :

1. Allumez votre robot à l'aide du bouton Orange et branchez-le en USB à votre ordinateur.
2. Ouvrez le fichier *BIA-TP-5-6-prog.nxc* avec un éditeur de texte et sauvegardez-le avec le nom de fichier définitif, tout en gardant l'extension.
3. Faites vos modifications dans le fichier et sauvegardez.
4. Compilez le fichier à l'aide de la commande : ***nbc - d -S=usb nomProg.nxc***
   Cette commande compile votre ficher et le télécharge sur votre robot branché en USB.
5. Débranchez votre robot et placez-le sur le labyrinthe.
6. Pour lancer votre programme, utilisez les commandes du robot :
   a. Naviguez avec les flèches jusqu'au menu *« My Files »* puis entrez avec le bouton Orange.
   b. Naviguez jusqu'à *« Software Files »* et validez.
   c. Naviguez jusqu'à votre fichier et validez.
   d. Votre nom de fichier s'affiche suivi de « Run » : validez.
   e. Votre programme s'exécute.
   f. Pour l'interrompre avant la fin, utilisez le bouton Gris.
7. Répétez cette procédure autant que nécessaire pour tester votre code.

Pour programmer, vous utilisez le **langage NXC** (http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html). Ce langage est un langage dérivé du C. L'aide mémoire fourni avec ce TP vous donne les bases du langage. Si vous avez besoin de plus d'information, de multiples tutoriel sont disponibles sur le net.

Question 1 : Trouver la sortie du labyrinthe (4 pts)

Programmer votre robot pour que celui-ci trouve la sortie en explorant le labyrinthe avec un algorithme de recherche non informé (par exemple, tourner toujours à gauche).

Pour cela, compléter le ficher source qui vous est fourni. Celui-ci contient :

- Un programme principal *« task main() »* qui permet au robot d'avancer sur les lignes noires et qui s'arrête quand il trouve une case de couleur.
- Une fonction *« obstacle() »* que vous devez compléter pour indiquer au robot quoi faire lorsqu'il se trouve sur ce type de case. Attention, vous ne pouvez pas franchir cette case… il faut demander au robot de repartir en arrière.
- Une fonction *« intersection() »* que vous devez compléter pour indiquer au robot la direction à prendre quand il est sur une intersection.
- Les fonctions *« gauche(), droite(), demiTour(), devant() »* qui permettent de faire avancer le robot.

Question 2 : Revenir à la case départ le plus rapidement possible (10 pts)

Programmer votre robot pour que celui-ci revienne à la case départ en calculant le plus court chemin avec l'algorithme de recherche A*.
Pour cela, vous devez modifier votre programme afin que le robot mémorise les parties du labyrinthe qu'il a visitées. Attention, il ne faut pas coder en dur le labyrinthe, mais mémoriser le contenu des intersections découvertes lors du parcours de la question 1. Vous disposez pour cela :

- des variables :
  - *int labyrinthe[] = {6,6,6,6,6,6,6,6,6,6,6,6}* qui représente les 12 cases du labyrinthe initialisées à la couleur blanche ;
  - *int direction = 2* qui représente la direction de votre robot : 0 tourné vers la gauche, 1 le haut, 2 la droite, 3 le bas ;
  - *int position[] = {1,0}* qui indique où vous positionnez votre robot au départ, puis la position courante du robot.
- des fonctions :
  - *memoriserLabyrinthe(int c)* qui sauvegarde la couleur de la case sur laquelle se trouve le robot ;
  - *majPosition()* qui met à jour la position du robot ;
  - *afficheLab()* qui affiche sur l'écran du robot l'état du labyrinthe ;
  - *couleurCapteur(int &c)* qui retourne la couleur détectée par le capteur 3 couleurs : 1 Noir, 2 Bleu, 3 Vert, 4 Jaune, 5 Rouge, 6 Blanc.

Pour information, les cases du labyrinthe sont numérotées par 2 entiers x, y sachant que x représente les colonnes (de 0 à 2) et y les lignes (de 0 à 3). La case (0,0) se trouve en haut à gauche et la case (3,2) en bas à droite. La variable labyrinthe[] décrit les couleurs des cases de la façon suivante :

{case(0,0), case(0,1), case(0,2),
case(1,0), case(1,1), case(1,2),
case(2,0), case(2,1), case(2,2),
case(3,0), case(3,1), case(3,2)}

Ensuite, programmer la fonction *AStar()* qui calcule le plus court chemin entre la sortie et votre case de départ. Utilisez pour l'instant uniquement une fonction de coût, sans heuristique.

Programmez ensuite la fonction *RetourDepart()* qui permet au robot de suivre le chemin identifié. Vous disposez pour cela :

- des variables :
  - *struct voisin{*
    int positionVoisin;
    int gVoisin;
    int pereVoisin;};
    qui permet de mémoriser la position d'une case sur le labyrinthe, le cout pour venir sur cette case à partir de la sortie et la case précédente dans le chemin ;
  - *voisin EnsOuverts[12]* : l'ensemble des ouverts pour l'algorithme A* ;
  - *voisin EnsFermes[12]* : l'ensemble des fermés ;
  - *int nbEnsOuverts = 0* : le nombre d'éléments dans l'ensemble des ouverts ;
  - *int nbEnsFermes = 0* : le nombre d'éléments dans l'ensemble des fermés.
- des fonctions :
  - *convertirPosition(int x, int y, int &c)* qui permet à partir d'une case dans le labyrinthe d'avoir son index dans le tableau le représentant ;
  - *CalculVoisinCase(int courant, int &gch, int &dr, int &bas, int &haut)* qui retourne l'index des 4 voisins d'une case.

Question 3 : Améliorer la recherche de la sortie du labyrinthe (3 pts)

Améliorez vos fonctions permettant de sortir du labyrinthe afin d'éviter les bouclages dans votre exploration du labyrinthe.

Question 4 : Améliorer l'algorithme A* avec des heuristiques (3 pts)

Améliorez votre fonction A* en utilisant en plus du coût, des heuristiques pour trouver le plus court chemin.

Bonus : Détecter les blocages (3 pts)

Essayez de détecter les situations dans lequel il est impossible d'atteindre la sortie afin d'éviter à votre robot se s'épuiser…