Master Informatique 1ère année:

Algorithmes d'exploration et de mouvement

Intervenant (mais non responsable du module): Jacques Ferber

HMIN233B - Année 2018-2019

TP3 - Algorithmes de recherche de chemin dans un espace

1. Algorithmes de base, sans information

On veut construire un espace avec des obstacles dans lequel la tortue peut évoluer dans les 8 directions de l'espace.

a) Créer un environnement (c'est préférable d'utiliser un mécanisme de création de murs par la souris comme vu dans le template du TP1) dans lequel il puisse y avoir des obstacles et des cases "but". On pourra utiliser le template du cours proposé ici.

b) Implémenter un comportement de base de la tortue sans exploration: aller devant soi jusqu'à rencontrer un mur. S'il y a un mur, alors où il n'y a une seule issue (à gauche ou à droite) et en suivre un au hasard, et continuer jusqu'à (éventuellement) trouver "par hasard" la cible. On supposera que la tortue "marque" et donc colorie le patche sur lequel elle se trouve.

c) Implémenter un algorithme d'exploratoin en utilisant d'abord l'algorithme en profondeur/largeurd'abord. On rappelle l'algorithme général de base:

```
to Search
list-nodes <- insert(initial-node)
Tant que pas trouvé
     si list-nodes est vide, alors échec        // on n'a pas trouvé de solution)
     cnode <- first (sort(list-nodes)) et supprimer cnode de la liste
     evaluer(cnode)                                    // Exécuter ce qu'il y a à faire, évaluer le noeud. si on trouve le but directement, sortir et retourner la solution
     list-nodes <- add-list (generate (cnode), list-nodes) ;; en profondeur d'abord si les noeuds sont placés devant,
                                                                                                  ;; en largeur d'abord si les noeuds sont placés après
```

On considèrera qu'un état (un noeud de l'exploration) est donné par : [patch, heading, type, cout] où type indique s'il s'agit d'un but ou non et où coup correspond à la profondeur dans l'exploration.

Implémenter les procédures suivantes:

- `search` (attention, vous pouvez utiliser le mécanisme de 'forever' de la procédure 'go' pour implémenter la boucle),
- `sort`: tri les noeuds si nécessaire
- `add-list` qui place la liste produite par generate au début de la liste des noeuds.
- `generate` : qui crée une liste de noeud à partir du noeud courant, lorsque la tortue se trouve devant un choix.

2. Algorithme avec information: Dijskstra et A*

Implémenter Dijkstra et A* à partir du template en NetLogo. On codera les noeuds sous la forme d'un doublet: (patch valeur), valeur étant la fonction de cout associé au noeud lors d'une exploration.

Dijkstra utilise une fonction de cout qui porte uniquement sur le chemin effectué..

A* utilise deux fonctions : l'estimation (h(n) et le cout g(n), pour calculer une valeur d'un noeud: f(n) = g(n) + h(n). Le cÃ´ut est donné par le nombre de mouvements nécessaires pour venir jusqu'ici, et l'estimation comme une évaluation du nombre de cout qu'il serait nécessaire pour aller directement au but.

On ajoutera le champ h (estimated cost) à la définition des états.

La fonction h est une heuristique (d'où son nom). Par défaut on peut prendre une distance (distance euclidienne, ou distance de manhattan) comme estimation.