Algorithme A*

Résolution du problème des robots ricochets

L'*algorithme A**[1] fait partie des *stratégies d'exploration informée* et plus particulièrement des explorations "meilleur d'abord". L'exploration meilleur d'abord est une *exploration de graphe*. Dans notre cas, chaque noeud représente le déplacement d'un robot dans une certaine direction.

La stratégie consiste à sélectionner le prochain noeud en suivant une fonction d'évaluation f(n).

On a $f(n) = g(n)+h(n)$ avec :

. $g(n)$: le coût du chemin entre le noeud de départ et le noeud n

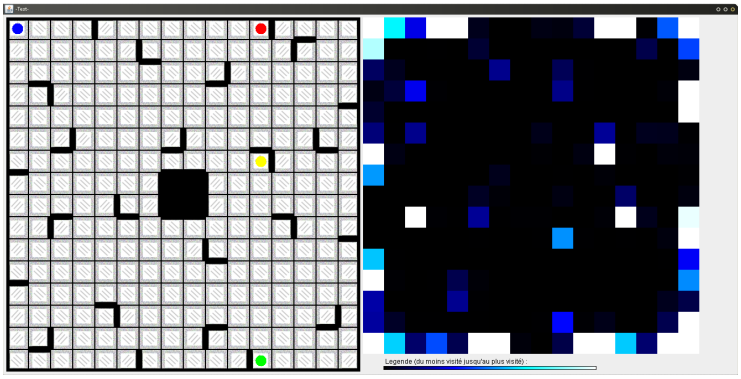
. $h(n)$: le coût estimé du chemin le moins onèreux de n au but

Pas de problème pour $g(n)$ mais comment définir $h(n)$?

Il est démontré que A* est optimal si $h(n)$ est une heuristique admissible, c'est à dire que $h(n)$ ne surestime jamais le coût pour atteindre le but.

Dans la plupart des cas, l'heuristique admissible se réduit à la distance à "vol d'oiseau" vers le but. Mais pour notre problème, il ne semble pas exister de solution intuitive. Il faut découvrir si certaines cases sont plus importantes que d'autres.

Idée : on laisse les robots parcourir aléatoirement l'environnement en comptant le nombre d'arrêt sur chaque case et ainsi trouver celles qui sont le plus visitées.



Bien que nous n'ayons pas encore découvert de fonction heuristique, on remarque qu'il est inutile de visiter un grand nombre de noeuds.

[1]Source : *Intelligence artificiel de Stuart Russell et Peter Norvig*

Pour aller plus loin, on sait également que A* est optimal si *h(n)* est *consistante*. Une heuristique est consistante si, pour tout noeud n et pour tout successeur n' de n généré par une action a, le coût estimé pour atteindre le but en partant de n n'est pas supérieur au coût de l'étape pour aller à n' plus le coût estimé pour atteindre le but à partir de n'.

Pseudo code de A*[2]

```
function A*(start,goal)
     // The set of nodes already evaluated
     closedset := the empty set
     // The set of tentative nodes to be evaluated.
     openset := set containing the initial node
     // Distance from start along optimal path
     g_score[start] := 0
     h_score[start] := heuristic_estimate_of_distance(start, goal)
     // Estimated total distance from start to goal through y
     f_score[start] := h_score[start]
     while openset is not empty
         x := the node in openset having the lowest f_score[] value
         if x = goal
             return reconstruct_path(came_from[goal])
         remove x from openset
         add x to closedset
         foreach y in neighbor_nodes(x)
             if y in closedset
                 continue
             tentative_g_score := g_score[x] + dist_between(x,y)

             if y not in openset
                 add y to openset

                 tentative_is_better := true
             elseif tentative_g_score < g_score[y]
                 tentative_is_better := true
             else
                 tentative_is_better := false
             if tentative_is_better = true
                 came_from[y] := x
                 g_score[y] := tentative_g_score
                 h_score[y] := heuristic_estimate_of_distance(y, goal)
                 f_score[y] := g_score[y] + h_score[y]
     return failure

 function reconstruct_path(current_node)
     if came_from[current_node] is set
         p = reconstruct_path(came_from[current_node])
         return (p + current_node)
     else
         return current_node
```

[2]Source : http://en.wikipedia.org/wiki/A*_search_algorithm