|  |
| --- |
| UniversitÉ du quÉbec en outaouais |
| Mini-Projet : Agent intelligent dans un environnement à obstacle |
| INF1183 - Hiver 2014 – Intelligence artificielle |
| **Guillaume Plouffe**  **Jean-Philippe Gauthier**  **Julien Bassompierre** |
| **2/28/2014** |

Table des matières

[Énoncé 1 1](#_Toc381861182)

[Phase 1 1](#_Toc381861183)

[Phase 2 2](#_Toc381861184)

[Phase 3 3](#_Toc381861185)

[Enoncé 2 7](#_Toc381861186)

[Références 8](#_Toc381861187)

Table des figures

[Figure 1 : Algorithme de colonie de fourmis de *Samwdon [1].* 2](#_Toc381861174)

[Figure 2 : Graphe utilisé pour faire nos expériences. Les nœuds 0 et 13 sont respectivement la source et la destination. 3](#_Toc381861175)

# Introduction

Pour ce mini-projet nous avons chosis l’énoncé 2 consistant à définir un agent intelligent qui se deplace dans une enivronnement pouvant avoir des obstacles. Le but (performance) de l’agent est de trouver le plus court chemin entre un point de depart et un point d’arrivé tout en contournant les obstacles.

Nous allons tout d’abord definir le PEAS de l’agent, donner une breve description sur le fonctionnement du programme et sur l’algorithme et les heuristiques utilisé. Ensuite, une analyse sera presenté en fonction de different heuristique, action possible, arrangement des obstacles et propriete de l’environnement.

# Description du PEAS

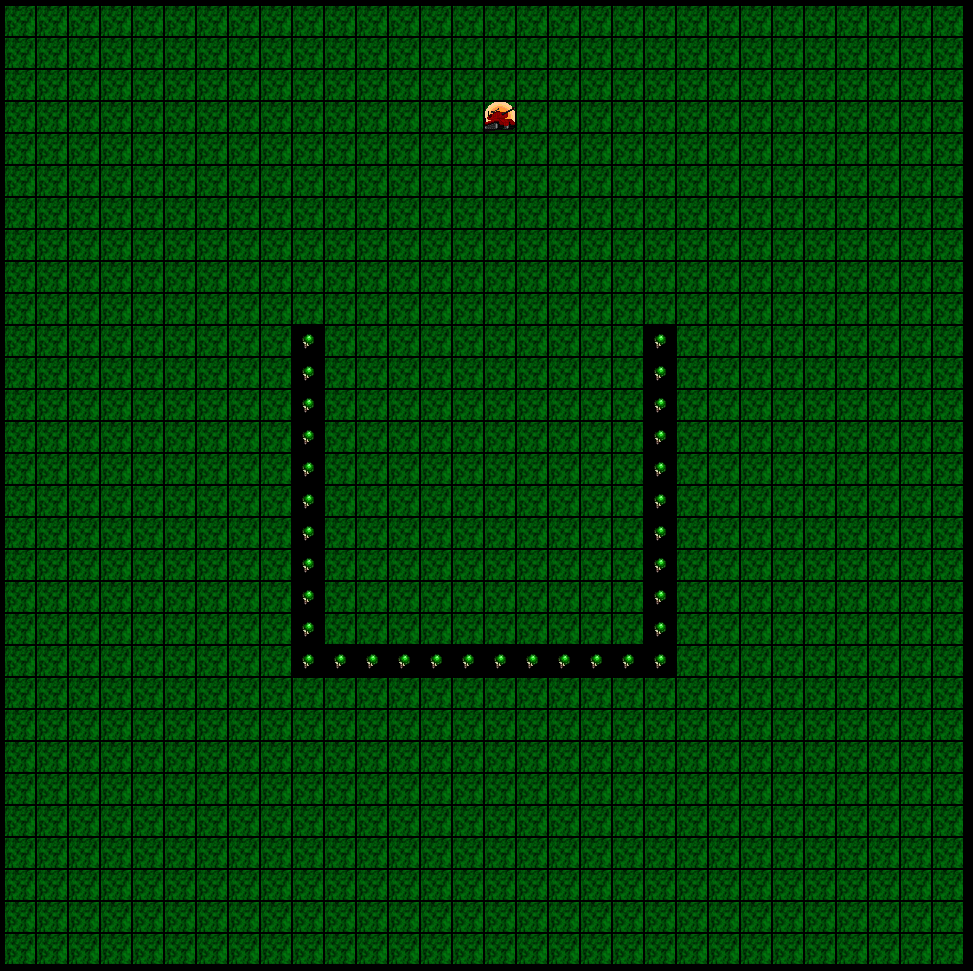
Dans cette section du document, nous allons décrire les différentes composantes du PEAS entourant l’agent : La performance, l’environnement, les actuateurs et les senseurs.

## Performances

L’objectif principale de l’agent est de trouvée le chemin le plus court entre un point départ et d’arrivée. Également, il doit pouvoir trouver ce chemin rapidement, et donc essayé un minimum de case de l’environnement dans sa recherche.

## Environnement

Il s’agit d’un terrain de tuile à deux dimension pouvant être considérer comme un graphe dont les cases sont les nœuds et dont les arêtes sont placé entre les tuiles adjacente. Les cases case peuvent contenir des obstacles ou non. Les obstacles sont représenter en noir avec un arbre vert au centre et les espace libre sont représenter par de l’herbe verte. L’agent est représenté par le tank en rouge. La grandeur du terrain est de 30 x 20 tuiles. La case départ est là ou apparait l’agent, alors que la case d’arriver et la case au-dessus de laquelle se trouve notre curseur souris (non représenté). Dans notre environnement de test, les obstacles sont configuré afin de constituer un cul-de-sac. Ce dernier est donc un piege que notre algorithme tentera au meilleur de ses moyens d’éviter à moindre cout pour trouver la solution du plus court chemin.

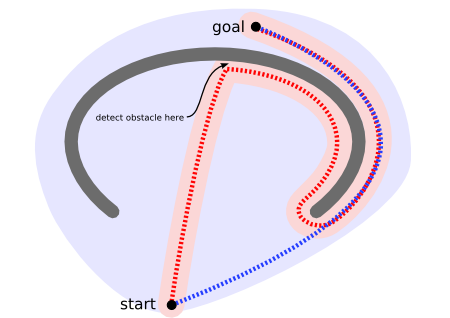


## Actuateurs

L’agent (tank rouge) peut réaliser quatre actions de déplacement : Vers le nord, sud, est, ouest. Le cout de déplacement par case libre est 1 et il ne peut passer par une case obstacle.

## Senseurs

Il s’agit de la camera de notre agent représenter par sa capacité à planifier d’avance son parcours et évaluer n’importe quel case par-rapport au point de départ et d’arrivé. Ce senseur est crucial dans le calcul de l’algorithme A\*. Dans la réalité, ce genre d’algorithme est plus lent comparativement à un algorithme qui réalise une action et qui effectue une évaluation plus restreinte des cases à proximité pour réaliser la suivante. Cependant, cette dernière ne pourra donner un parcours optimal en distance du fait de son manque de connaissance sur son environnement. En effet, il ne pourra éviter de souvent tomber et rester pris dans des pièges (cul-de-sac).



# Fonctionnement du programme

Il s’agit ici du guide utilisateur.

# Description du code

## Algorithme A\*

L’algorithme A\*est la méthode la plus populaire dans la recherche du plus court chemin. Il combine les avantage de l’alorithme de Disjstra et de la recherche « best-first » (un algorithme de type glouton). L’algorithme de Disjstra consiste a visité de manière incremental le nœud le plus pres encore inexplorer. Il s’agit d’une procedure longue, mais qui grarantie d’obtenir le chemin optimal. De son coté, l’algorithme « Best-first » effectue une estimation (heuristique) sur le chemin qu’il reste a parcourir à chacune des case visité. Il va donc visité de manière incrémental les nœuds les plus proches du but. Cette algorithme est beaucoup plus rapide que Disjstra, mais la solution obtenue n’est pas garantie d’être la distance la plus courte.

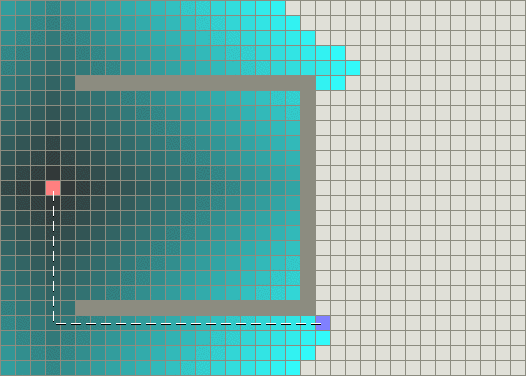


Figure : Recherche du plus court chemin avec l'algorithme de Disjstra

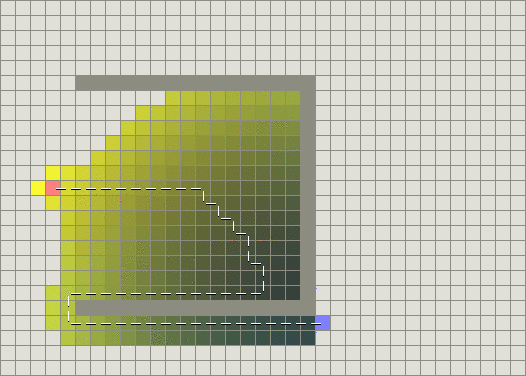


Figure : Recherche du plus court chemin avec Greedy Best first search

L’algorithme A\* est communément représenter par F(n) = g(n)+h(n). La partie Disjstra est représentée par g(n). Elle calcule le cout exact du chemin parcouru depuis la case départ jusqu'à la case n. La partie heuristique (best first search), est représentée par h(n). Elle calcul le cout estimé entre la case n et celle d’arrivée. L’algorithme A\* va réaliser une boucle afin de vérifier a chaque itération la case ayant la plus faible valeur de f(n) = g(n)+h(n).

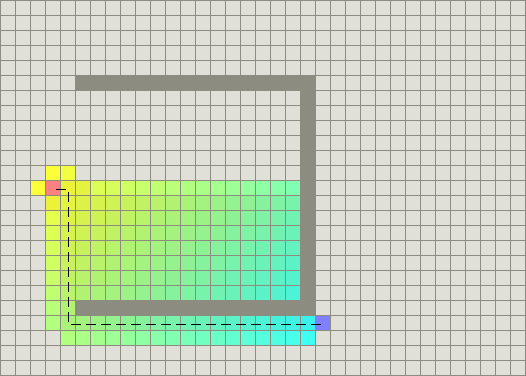


Figure : Recherche du plus court chemin avec l'agorithme A\*

## Heuristique utilisées

# Analyse des performances

Pour chacun des essaie décrit ci-dessus nous allons faire une étude en fonction de différente action possible, de différentes topologie de terrain et propriété de l’environnement.

## Essaie avec heuristique dite chemin de manathan

Pour cette analyse nous avons effectuer 450 test dans un envorinnement statique avec des obstacle. Chaque test utilise une combinaisson different de position de depart et d’arrivee. L’ensemble des position de depart utiliser sont contenue dans la boite blanche superieur et ceux d’arrivee dans la boite blanche inferieur de la Figure 1.

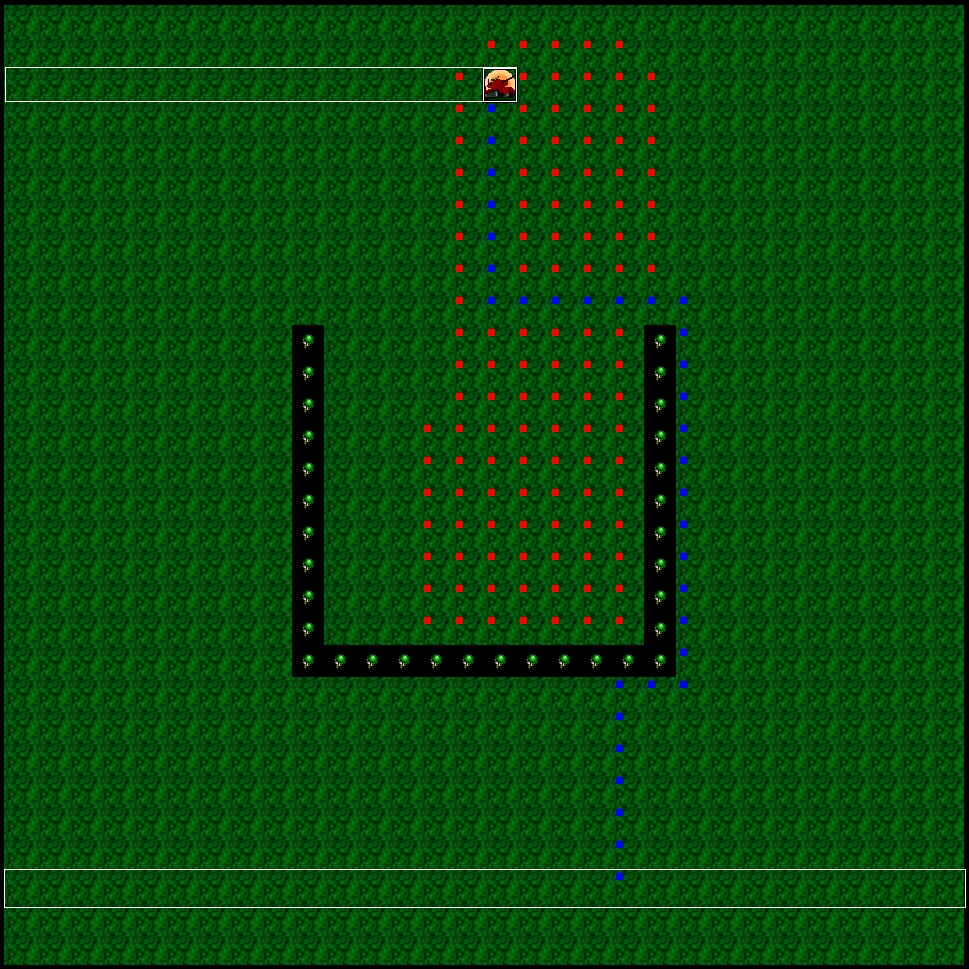


Figure  : Exemple de test utilisant l’heuristique de Manhattan avec un poid de 1.2.

Pour chacun de ces tests, nous avons incrémenté de 0.2 le poids de notre heuristique par rapport à g(n). Nous avons additionné ensemble tous les pas de chaque chemin optimal trouvé et de chaque case essayer par l’algorithme nous retournant un échec. Le resultat est presenter dans Tableau 1.

Tableau Etude de la precision et de la rapidité de l’heuristique de Manhantan en fonction du poid

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Essaie | Poid | TotalTest | totalCheminNonTrouver | totalPath (pas) | totalCaseEssayerEchec |
| 0 | 0 | 450 | 30 | 11380 | 214106 |
| 1 | 0.2 | 450 | 30 | 11380 | 185575 |
| 2 | 0.4 | 450 | 30 | 11380 | 153781 |
| 3 | 0.6 | 450 | 30 | 11380 | 126727 |
| 4 | 0.8 | 450 | 30 | 11380 | 98662 |
| 5 | 1 | 450 | 30 | 11380 | 75179 |
| 6 | 1.2 | 450 | 30 | 11380 | 31033 |
| 7 | 1.4 | 450 | 30 | 11694 | 31530 |
| 8 | 1.6 | 450 | 30 | 12000 | 29746 |
| 9 | 1.8 | 450 | 30 | 12102 | 27287 |
| 10 | 2 | 450 | 30 | 12192 | 26494 |
| 11 | 2.2 | 450 | 30 | 12292 | 26841 |
| 12 | 2.4 | 450 | 30 | 12356 | 26721 |
| 13 | 2.6 | 450 | 30 | 12410 | 27325 |
| 14 | 2.8 | 450 | 30 | 12430 | 28032 |
| 15 | 3.000001 | 450 | 30 | 12456 | 28915 |
| 16 | 3.200001 | 450 | 30 | 12546 | 28887 |
| 17 | 3.400001 | 450 | 30 | 12588 | 29475 |
| 18 | 3.600001 | 450 | 30 | 12670 | 29871 |
| 19 | 3.800001 | 450 | 30 | 12756 | 30279 |
| 20 | 4.000001 | 450 | 30 | 12786 | 30636 |
| 21 | 4.2 | 450 | 30 | 12890 | 30750 |
| 22 | 4.4 | 450 | 30 | 12948 | 31093 |
| 23 | 4.6 | 450 | 30 | 13000 | 31296 |
| 24 | 4.8 | 450 | 30 | 13054 | 31398 |
| 25 | 5 | 450 | 30 | 13082 | 31602 |

Comme on peut le constater, cette heuristique est optimale en précision et rapidité avec un poids égale à environ 1.2. C'est-à-dire f(n) = g(n) + 1.2 \* h(n). Il est ainsi préférable dans ce type d’environnement de légèrement surestimer le cout donner par l’heuristique.

## Essaie avec heuristique ….

## Essaie avec heuristique de notre propre concoction

Idee : Lorsqu’un obstacle est trouver, chercher le long de son contour.

# Conclusion

# Références

1. Site web : *https://github.com/samwdon/AntColonyOptimization*, accédé le 28 février 2014.
2. Site web : *http://www.epi.asso.fr/revue/articles/a0309b.htm*, accédé le 28 février 2014.