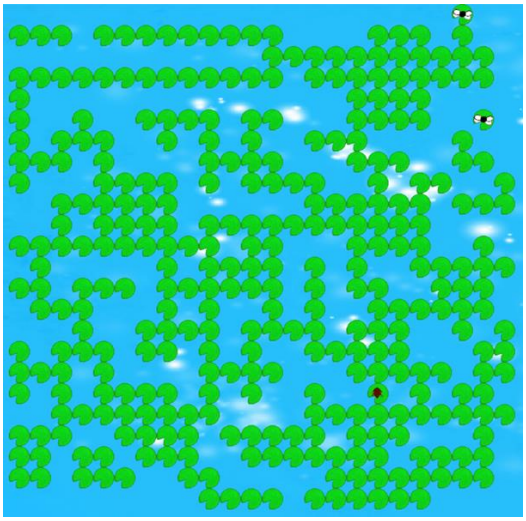
M1 Informatique ISICG, Université de Limoges

Projet IA 1

Recherche de chemin dans un labyrinthe



Etudiants : Antonin BERNARDIN, Steven KERJOUAN
Encadrant : Karim TAMINE

Date : 18/12/2015

Table des matières

Introduction

Ce projet d'intelligence artificielle avait pour but de nous faire implémenter des algorithmes de recherche de chemins. Ces algorithmes sont notamment utilisés dans les jeux vidéo pour le déplacement des personnages non joueurs.

Le programme est un labyrinthe qui prend en compte un fichier texte et qui affiche à partir de celui-ci génère un labyrinthe. Nous testons les algorithmes pour voir l'impact des différentes prises de décision dans le choix des nœuds explorés.

Au niveau de notre interface graphique, il s'agit d'une grenouille qui doit atteindre la mouche la plus proche en passant par le moins de nénuphars possibles, et cela dépendra des algorithmes et heuristiques choisis.

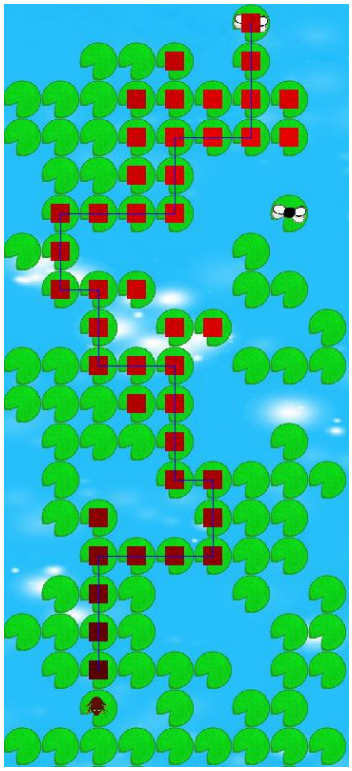
Algorithmes et heuristiques

Meilleur d'abord

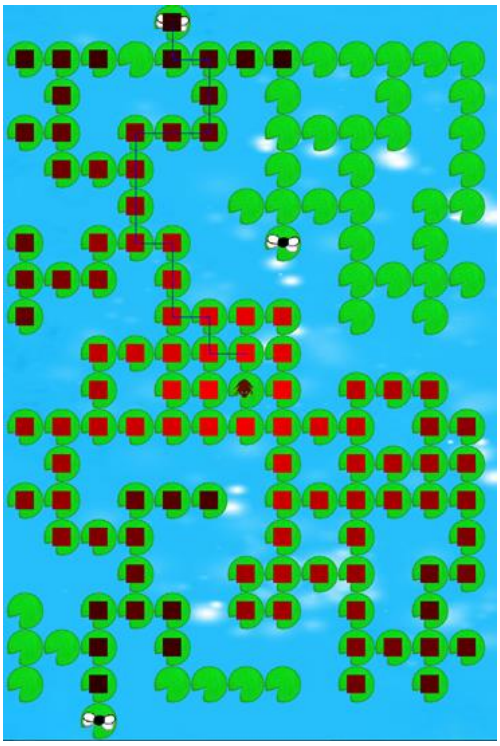
Dans le cadre de l'algorithme « Meilleur d'abord », notre implémentation est de type « gready search » : Le choix du nœud s'effectue en fonction de sa distance par rapport à l'arrivée.

Nous avons amélioré le code, dans le sens ou à chaque calcul d'heuristique nous calculons par rapport à la sortie la plus proche. Dans le cas où le chemin fait un grand détour pour atteindre une mouche, il est possible qu'il passe devant une autre, il faut donc s'adapter et choisir la nouvelle plus proche.

Sur l'image, c'est la mouche la plus proche (celle de droite) qui est choisie au départ, c'est pourquoi la grenouille effectue un petit détour sur la droite. En conséquence, le chemin choisi n'est pas le plus court chemin.



Coûts uniformes



Nous avons implémenté deux types de coûts uniformes : Le premier calcule le nombre de sauts depuis le départ alors que le second calcule la distance « à vol d'oiseau » depuis le départ.

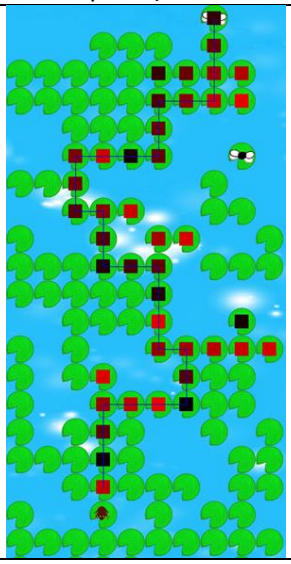
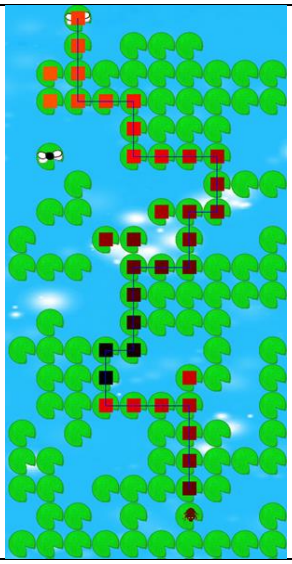
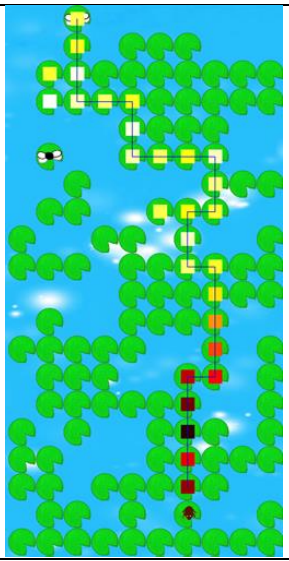
Cette différence influe sur le choix des nœuds explorés. Nous pouvons voir cette influence s'accroître si par exemple le coût des sauts n'est constant.

Par exemple si dans un jeu vidéo il y a plusieurs types de terrains, gravier, boue, sable, etc… nous pouvons imaginer que chaque type de terrain a un poids différent et que donc, selon le but désiré, soit le déplacement s'effectue en fonction de la distance, ou bien en fonction du coût du déplacement.

A Étoile

A Étoile est une combinaison du coût uniforme et de « gready search », c'est-à-dire qu'il prend en compte le coût depuis le départ (fonction *g(x)*) et la distance à vol d'oiseau (fonction *h(x)*) par rapport à l'arrivée.

Au niveau de notre heuristique, nous avons joué sur les poids p1 et p2 de ces fonctions qui composent notre fonction finale *f(x) = p1*g(x) + p2*h(x)* afin d'obtenir des résultats différents :

La priorité au départ est égale à la priorité à l'arrivée (mêmes valeurs de poids)	Le poids du départ est de 5 et l'arrivée est de 1	Le poids du départ est de 1 et l'arrivée est de 5
		

Implémentation

L'implémentation générale s'effectue dans une méthode "*execute*" (extrait de code ci-dessous) définie dans une classe abstraite "*Controller*". Pour chaque algorithme, on hérite de cette classe et on surcharge seulement la méthode virtuelle "*heuristique*".

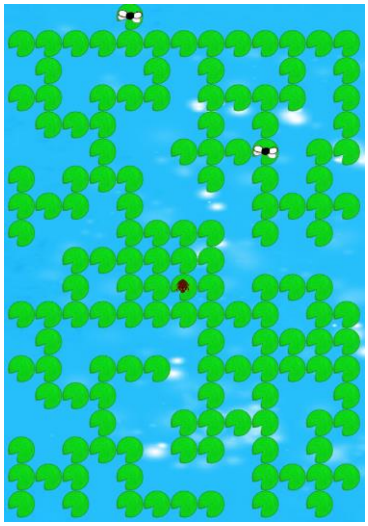
```
PileNoeud aExplo = PileNoeud();
PileNoeud dejaExplo = PileNoeud();
Noeud* curNoeud = NULL;
aExplo.empil(etatInitial);

while (aExplo.size() != 0) {
    curNoeud = aExplo.depil();
    dejaExplo.empil(curNoeud);
    info.charted.push_front(curNoeud);

    if (isBut(curNoeud)) break;
    else {
        std::list<Direction>::iterator it = op->begin();
        for (; it != op->end(); it++) {
            Noeud* curNoeudEnfant = curNoeud->successeur(*it);
            heuristique(curNoeudEnfant);
            if (isValid(curNoeudEnfant) && (!dejaExplo.isIn(curNoeudEnfant) && !aExplo.isIn(curNoeuc
                aExplo.empil(curNoeudEnfant);
            }
        }
    }
}
```

Analyse de performances

Propriétés des labyrinthes testés

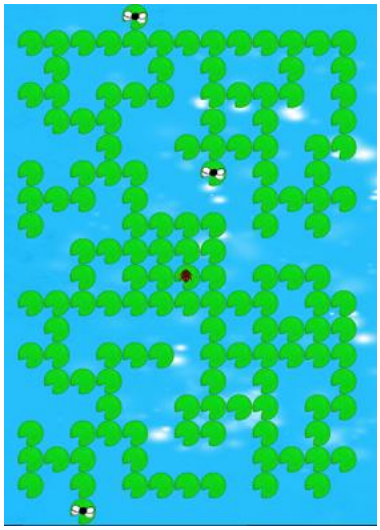
Labyrinthe N°1 :

- Dimensions : 15 x 20
- Nombre de sorties : 2
- Position de départ : 7 ; 10



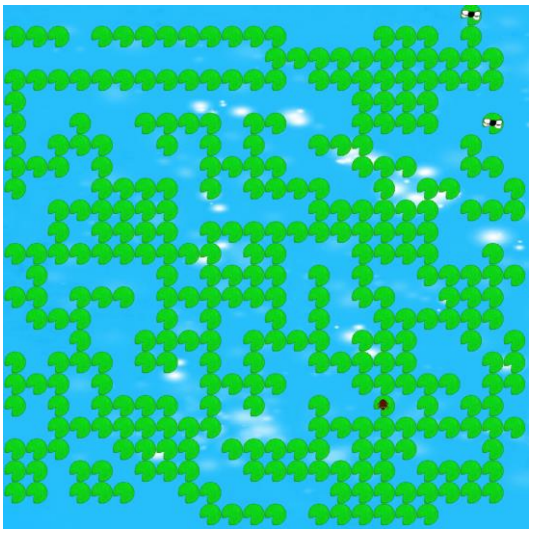
Labyrinthe N°2 :

- Dimensions : 15 x 20
- Nombre de sorties : 3
- Position de départ : 7 ; 10



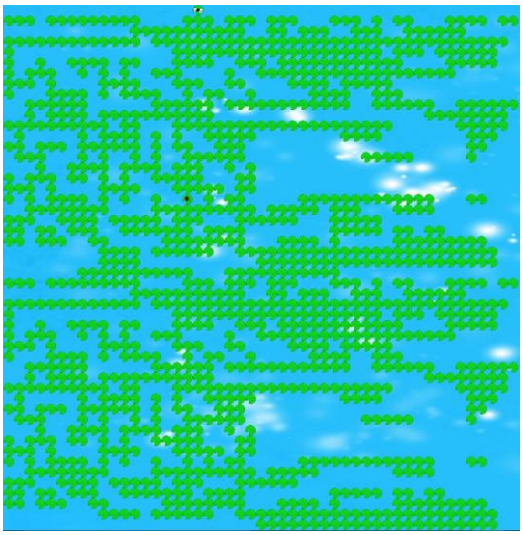
Labyrinthe N°3 :

- Dimensions : 25 x 25
- Nombres de sorties : 2
- Position de départ : 18 ; 18



Labyrinthe N°4

- Dimensions : 50 x 50
- Nombre de sorties : 1
- Position de départ : 18 ; 18



Benchmarks

Labyrinthe N°1

	Heuristique	Nombre de coups joués	Taille (nb de cases) du chemin trouvé	Temps d'exécution
Meilleur d'abord	Priorité à l'arrivée	37	21	3 ms
	Priorité au départ	112	21	12 ms
Coûts uniformes	Priorité au départ	104	15	9 ms
A Etoile	Même poids pour l'arrivée et le départ	25	21	2 ms
	Poids de 1 pour le départ et 5 pour l'arrivée	26	21	2 ms
	Poids de 5 pour le départ et 1 pour l'arrivée	24	21	1 ms

Labyrinthe N°2

	Heuristique	Nombre de coups joués	Taille (nb de cases) du chemin trouvé	Temps d'exécution
Meilleur d'abord	Priorité à l'arrivée	18	16	1 ms
	Priorité au départ	102	16	12 ms
Coûts uniformes	Priorité au départ	104	15	10 ms
A Etoile	Même poids pour l'arrivée et le départ	18	16	3 ms
	Poids de 1 pour le départ et 5 pour l'arrivée	18	16	1 ms
	Poids de 5 pour le départ et 1 pour l'arrivée	47	30	5 ms

Labyrinthe N°3

	Heuristique	Nombre de coups joués	Taille (nb de cases) du chemin trouvé	Temps d'exécution
Meilleur d'abord	Priorité à l'arrivée	44	29	4 ms
	Priorité au départ	288	29	59 ms
Coûts uniformes	Priorité au départ	276	25	58 ms
A Etoile	Même poids pour l'arrivée et le départ	37	29	5 ms
	Poids de 1 pour le départ et 5 pour l'arrivée	43	29	4 ms
	Poids de 5 pour le départ et 1 pour l'arrivée	30	25	3 ms

Labyrinthe N°4

	Heuristique	Nombre de coups joués	Taille (nb de cases) du chemin trouvé	Temps d'exécution
Meilleur d'abord	Priorité à l'arrivée	25	22	2 ms
	Priorité au départ	577	26	238 ms
Coûts uniformes	Priorité au départ	379	22	110 ms
A Etoile	Même poids pour l'arrivée et le départ	945	76	643 ms
	Poids de 1 pour le départ et 5 pour l'arrivée	25	22	2 ms
	Poids de 5 pour le départ et 1 pour l'arrivée	949	62	770 ms

Conclusion

Grâce à ce projet, nous avons mieux compris l'intérêt des différents algorithmes étudiés et les différences qui existent entre eux. Ils ont tous des avantages et des inconvénients. Certains ne vont pas trouver la meilleure sortie, mais une sortie tout de même, ce qui n'est pas forcément problématique dans certaines situations. Ces algorithmes s'adaptent donc chacun à différentes problématiques.

Évidemment la qualité d'un algorithme ne dépend pas que de son chemin trouvé mais aussi du temps de calcul pour trouver ce chemin, et là aussi il y a des différences de performances. Le tout est donc un compromis entre la performance et la qualité de la solution.