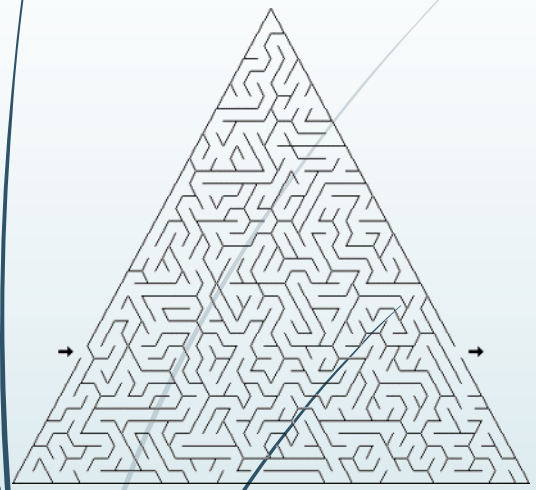
IA
Résolution de labyrinthes

Sébastien GUICHARD - Aurélien RICHETON - Romain LARROQUE
Thomas LOUBIER - Romain CASERY

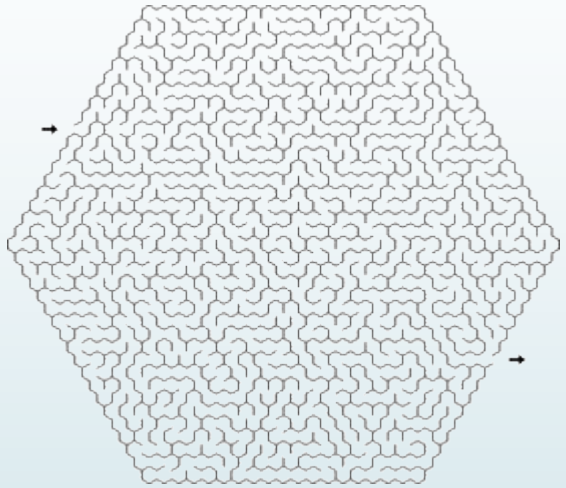
Plan

- Présentation du cadre d'étude
- Lien avec l'intelligence artificielle
- Exemple de résolution
- Algorithmes de résolution
  - Algorithme de Pledge
  - Algorithme du plus court chemin
- Conclusion et ouverture

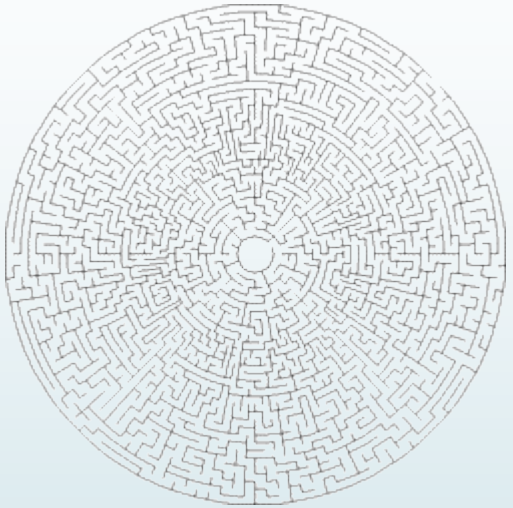
Cadre d'étude: Types de labyrinthes



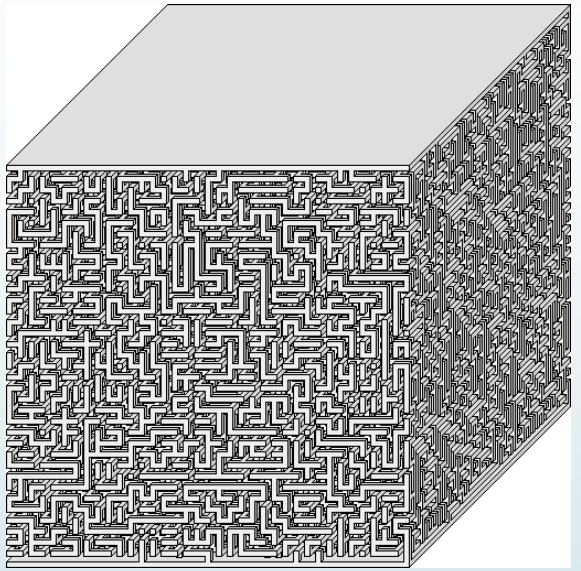
Delta



Sigma

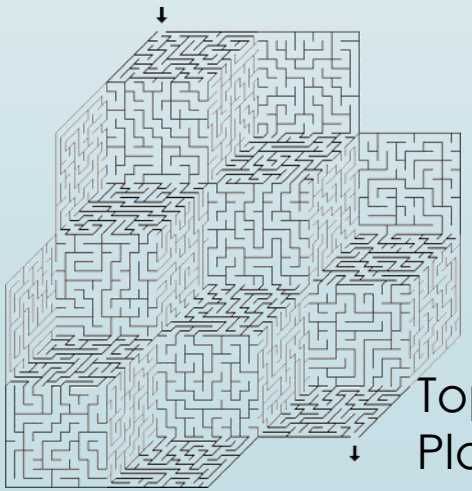


Théta



Hypermaze

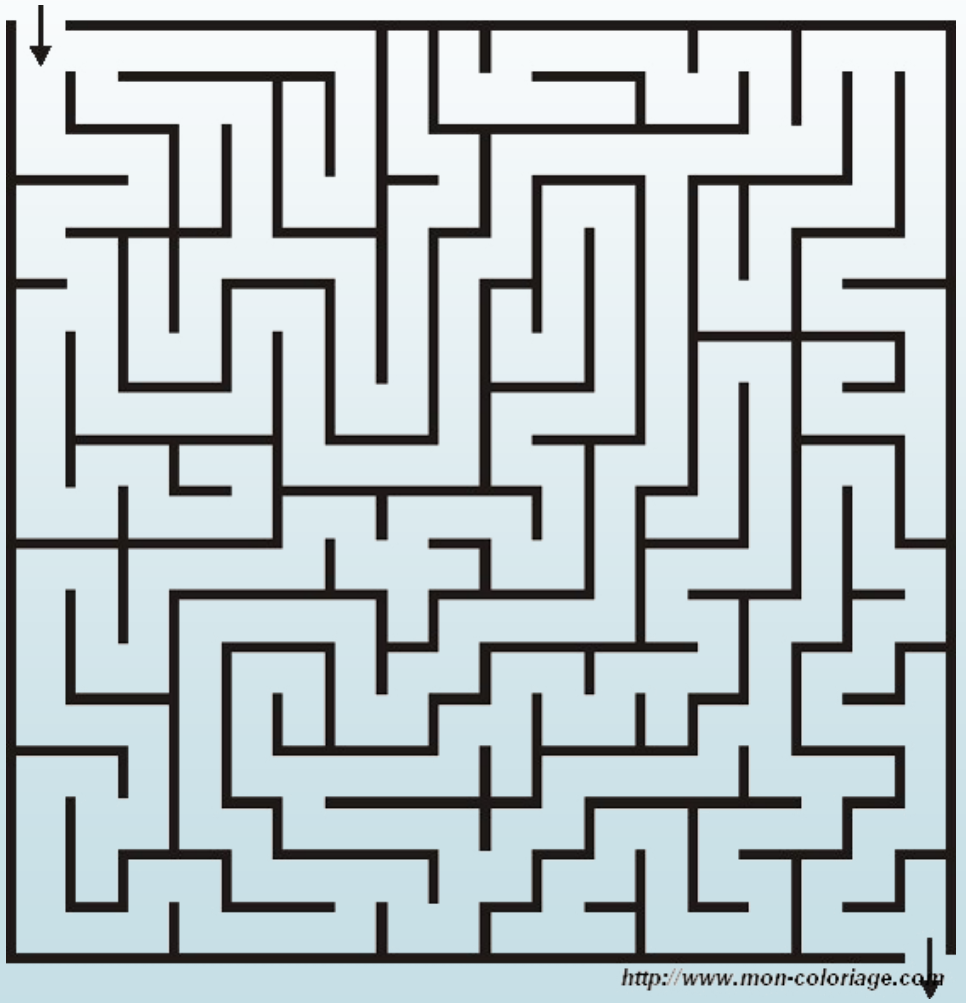
Dimension: 1/2/3D
Hyper dimension: Dimension de l'objet
Topologie: Géométrie
Técélation: Géométrie d'une cellule
Routing: Types de passages
Texture: Agencement des passages



Topologie Planaire

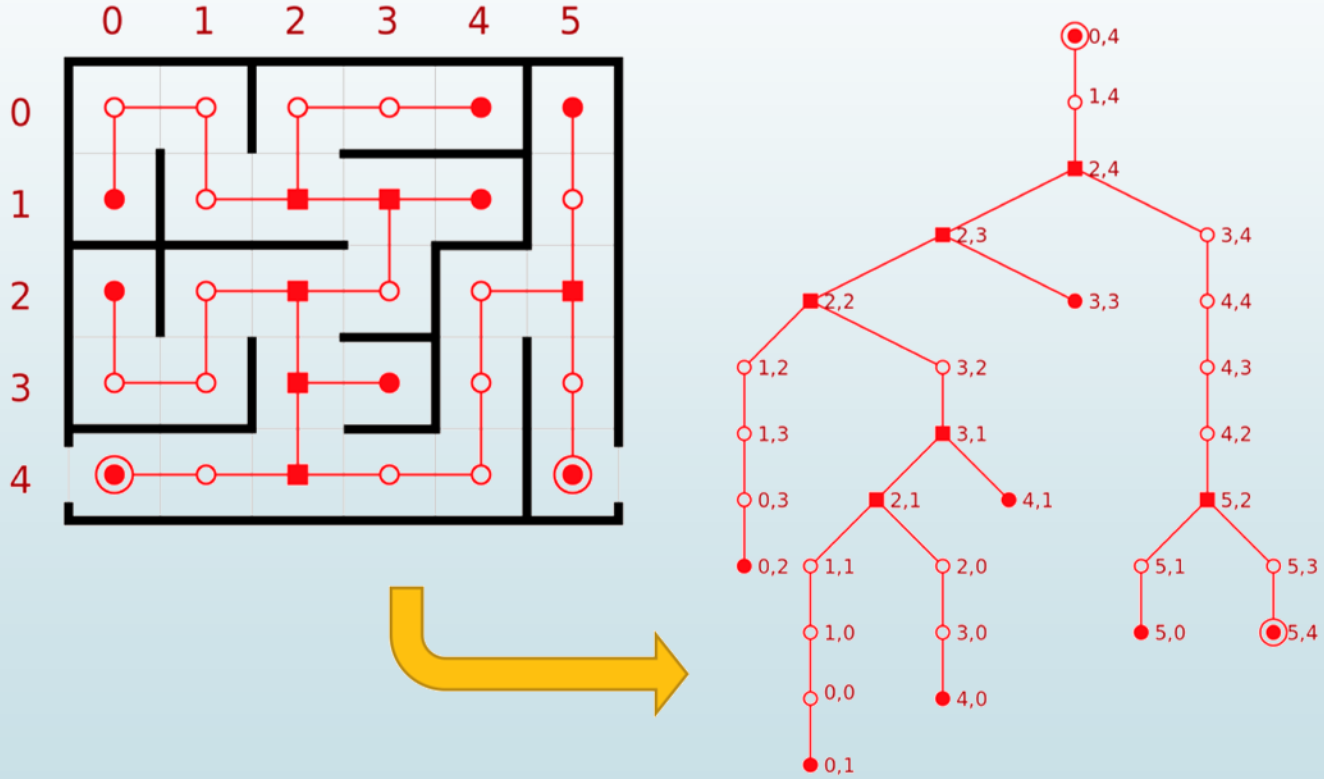
Cadre d'étude: Définition du cadre

Dimension: 2D
Hyper dimension: Non-HyperMaze
Topologie: Normale
Técélation: Carrée
Routing: Normal
Texture: Uniforme



Lien avec l"Intelligence Artificielle

- Résolution labyrinthe -> Résolution de graphe:



Source Image: https://fr.wikipedia.org/wiki/Modélisation_mathématique_d%27un_labyrinthe

- Or certains problème d'Intelligence artificielles sont schématisables en graphe

Exemple de Résolution: Le jeu du Taquin (1)

- Résolution du jeu de plateau du Taquin:

But:

	1	3
7	5	8
4	2	6

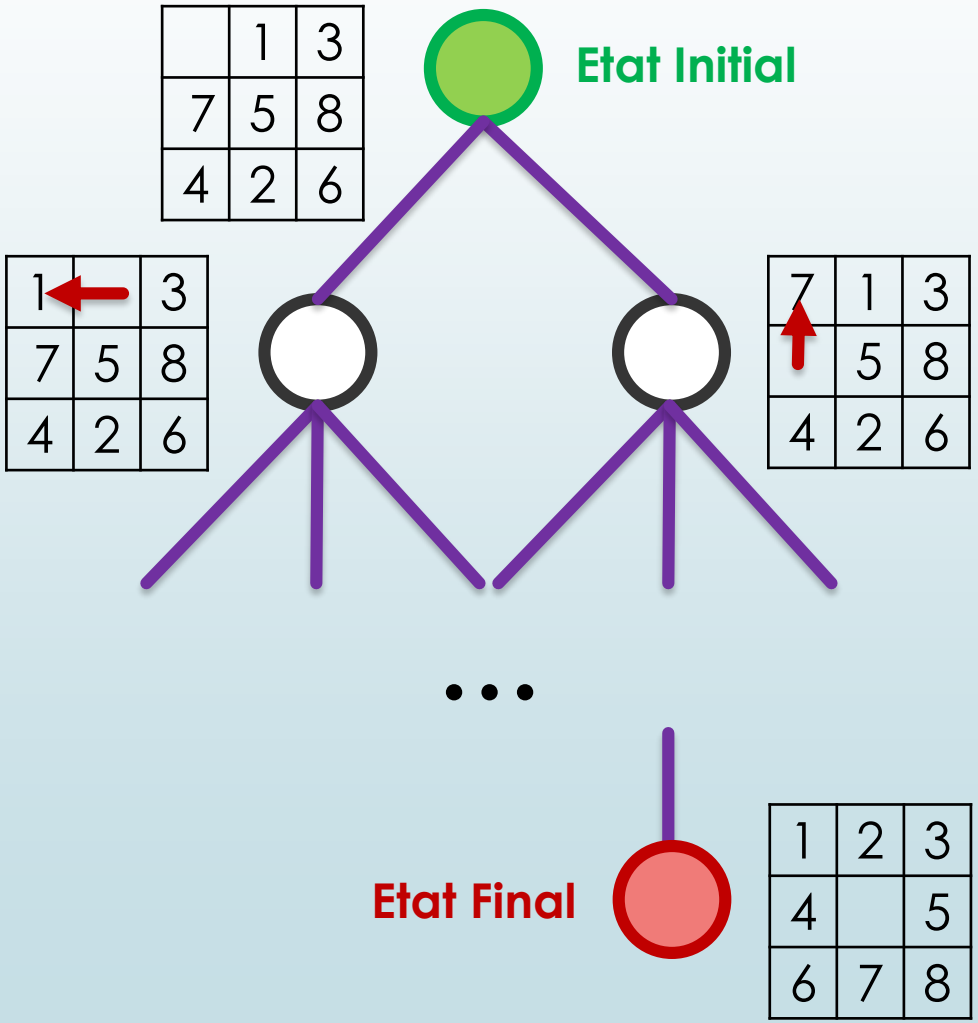
Etat de départ

1	2	3
4		5
6	7	8

Etat d'arrivée

Règle: Déplacement case numéroté vers case vide

Exemple de Résolution: Le jeu du Taquin(2)

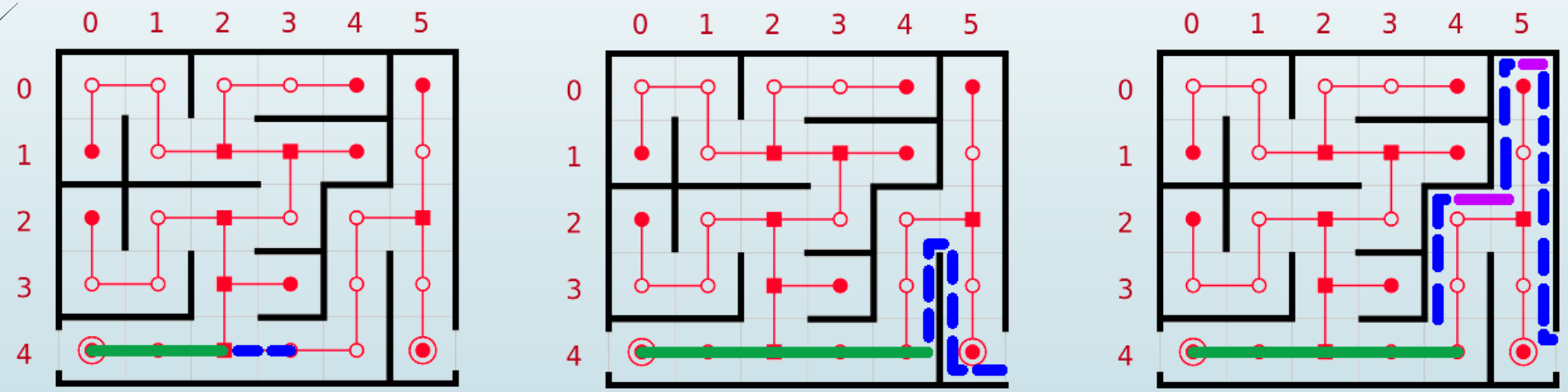


- Un noeud = un état du jeu
- Une arrête = déplacement d'une pièce
- Entrée = état de départ
- Sortie (solution) = état final

Algorithme de Pledge

- Présentation et fonctionnement :

  fonctionnement simple et visuel

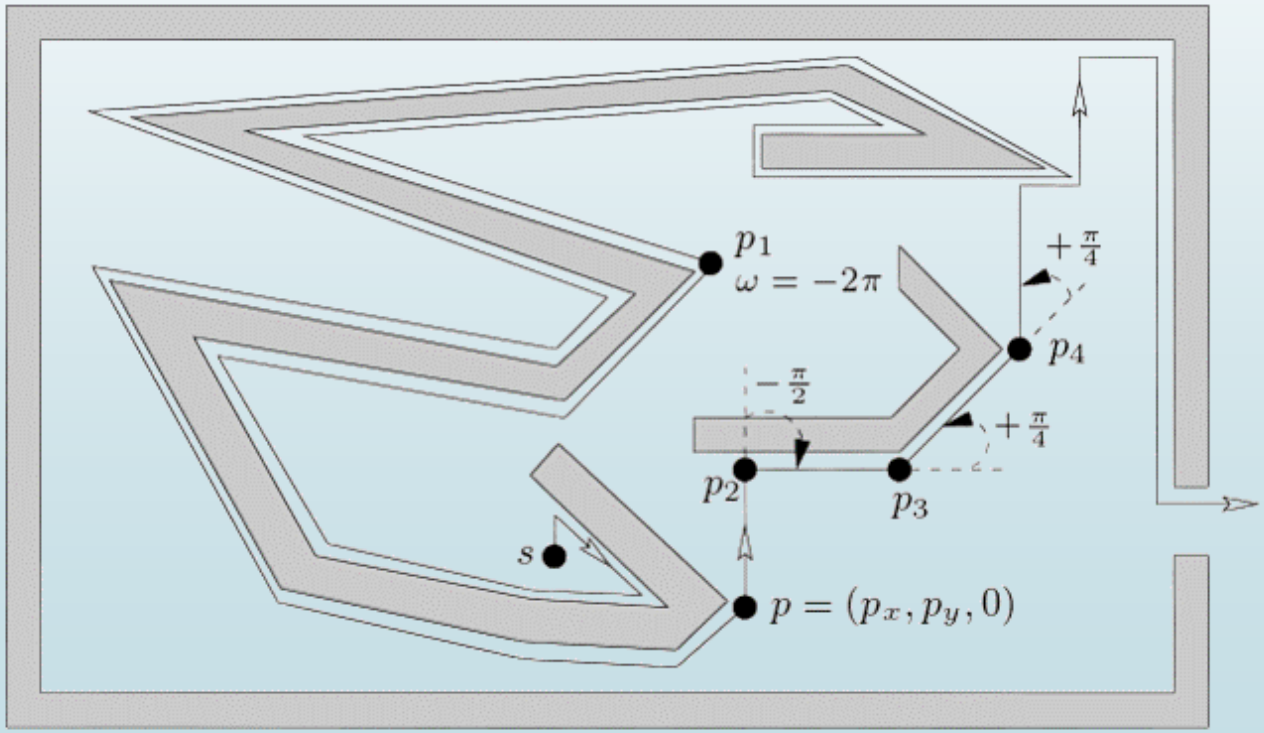


Algorithme de Pledge

- Cadre d'utilisation:

  Utilisation simple => implémentation simple

  Exemple pour la robotique:



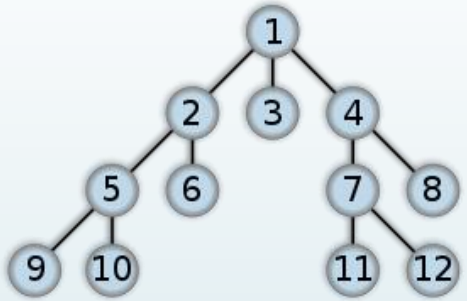
Algorithme de Pledge

- Limites de Pledge:

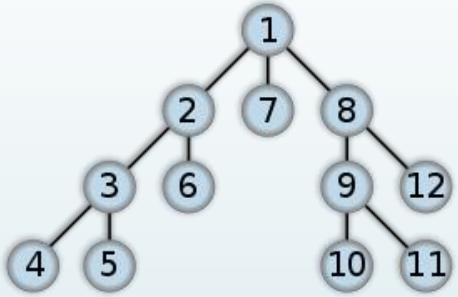
Source Image: http://www.apprendre-en-ligne.net/info/algo/algorithmique.pdf

Algorithmes “Shortest-path”

- 3 algorithmes de base:
  - Depth-first
  - Breadth-first
  - Best-first
- Des algorithmes plus élaborés:
  - L'algorithme de Dijkstra
  - A*



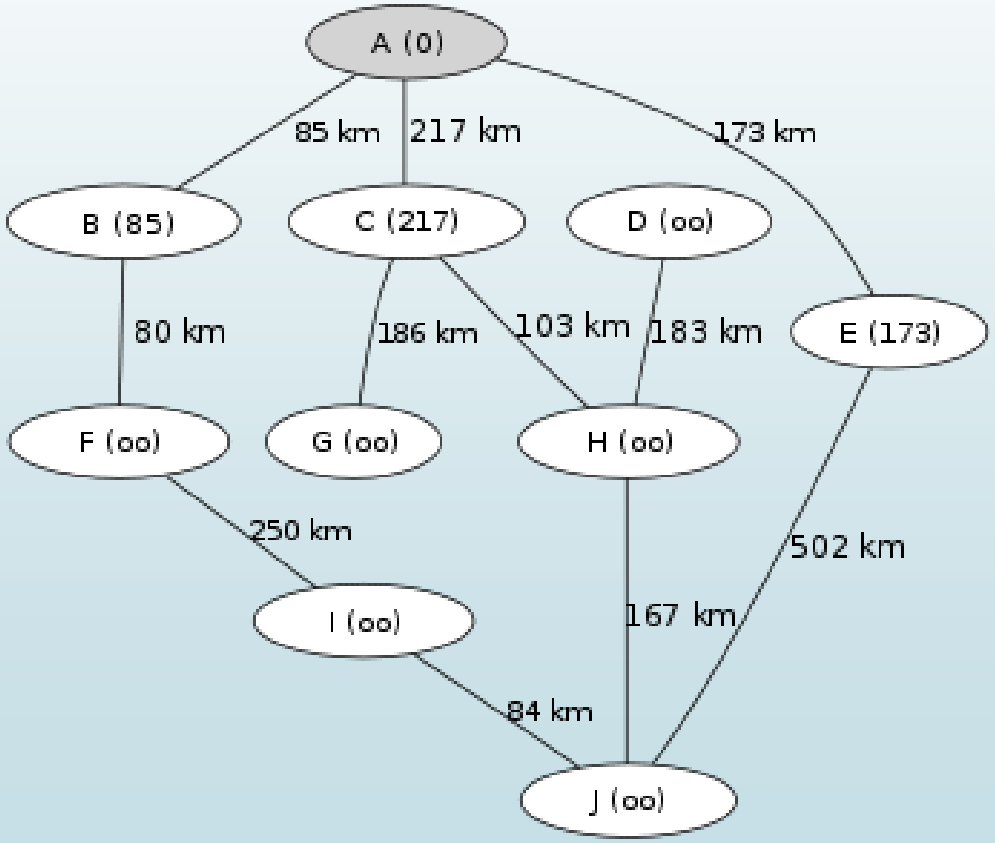
Breadth-first



Depth-first

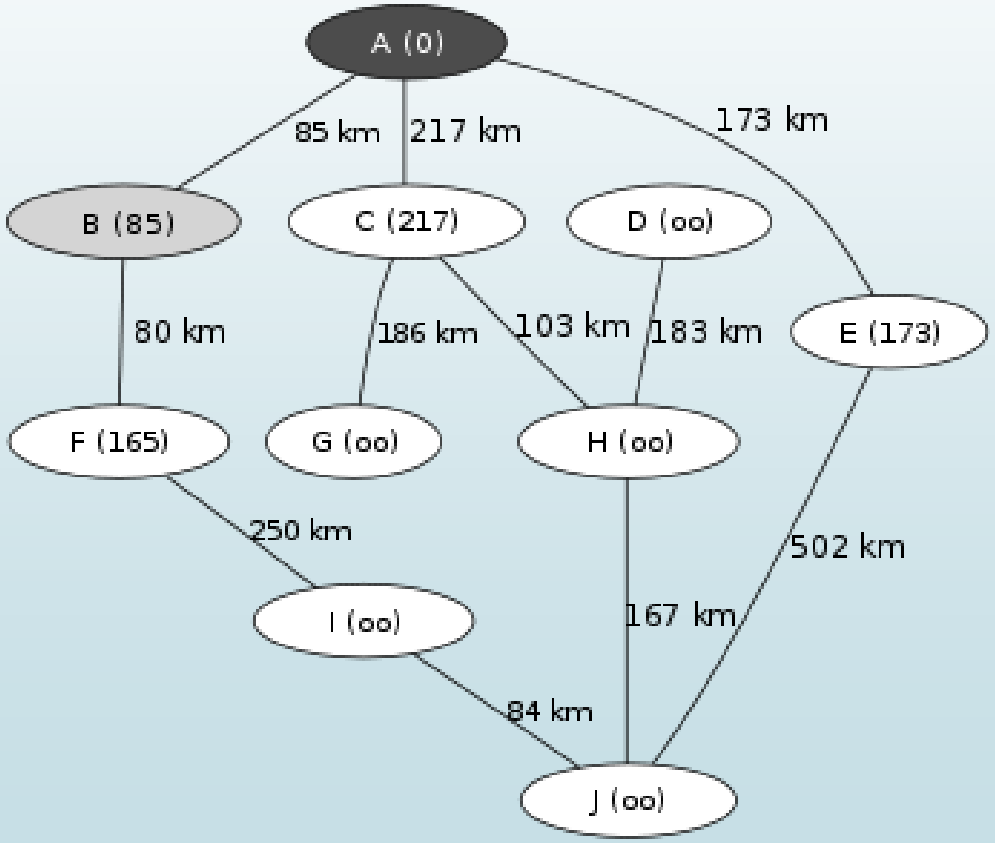
Algoritme de Dijkstra

	A	B	C	D	E	F	G	H	I	J
A	0	85	217	∞	173	∞	∞	∞	∞	∞
B	85	0	∞	∞	∞	80	∞	∞	∞	∞
C	217	∞	0	∞	∞	∞	186	103	∞	∞
D	∞	∞	∞	0	∞	∞	∞	183	∞	∞
E	173	∞	∞	∞	0	∞	∞	∞	∞	502
F	∞	80	∞	∞	∞	0	∞	∞	250	∞
G	∞	∞	186	∞	∞	∞	0	∞	∞	∞
H	∞	∞	103	183	∞	∞	∞	0	∞	167
I	∞	∞	∞	∞	∞	250	∞	∞	0	84
J	∞	∞	∞	∞	502	∞	∞	167	84	0



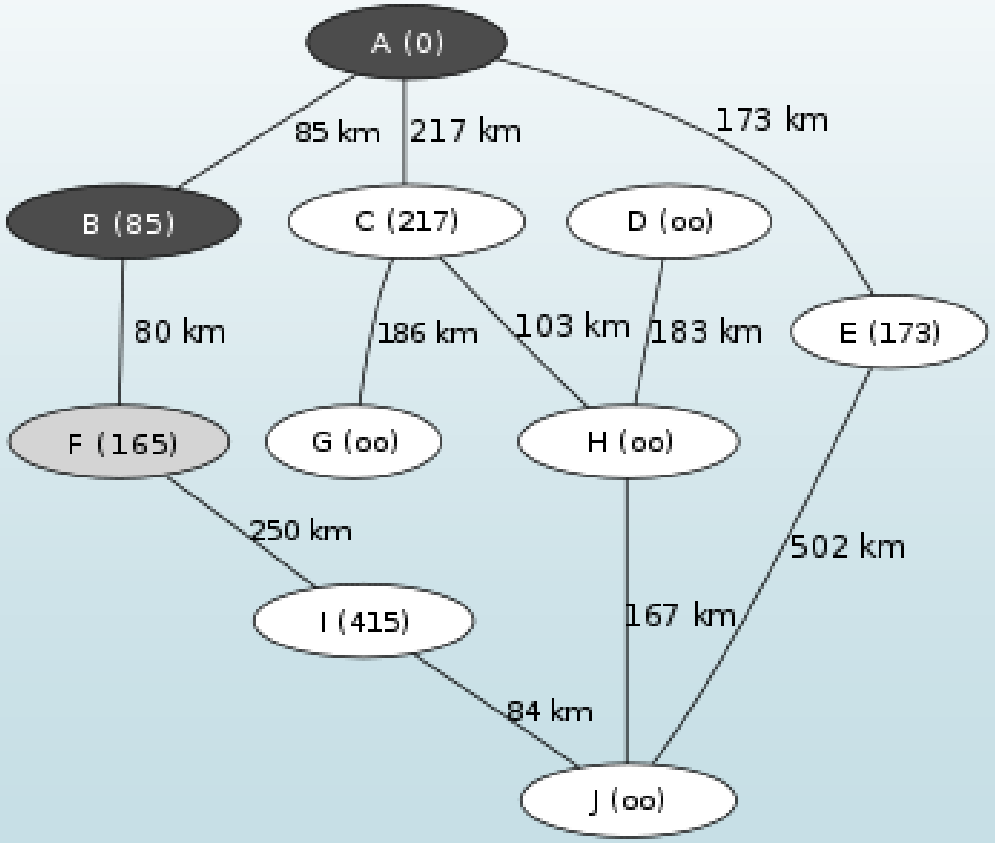
Algoritme de Dijkstra

	A	B	C	D	E	F	G	H	I	J
A	0	85	217	∞	173	165	∞	∞	∞	∞
B	85	0	∞	∞	∞	80	∞	∞	∞	∞
C	217	∞	0	∞	∞	∞	186	103	∞	∞
D	∞	∞	∞	0	∞	∞	∞	183	∞	∞
E	173	∞	∞	∞	0	∞	∞	∞	∞	502
F	165	80	∞	∞	∞	0	∞	∞	250	∞
G	∞	∞	186	∞	∞	∞	0	∞	∞	∞
H	∞	∞	103	183	∞	∞	∞	0	∞	167
I	∞	∞	∞	∞	∞	250	∞	∞	0	84
J	∞	∞	∞	∞	502	∞	∞	167	84	0



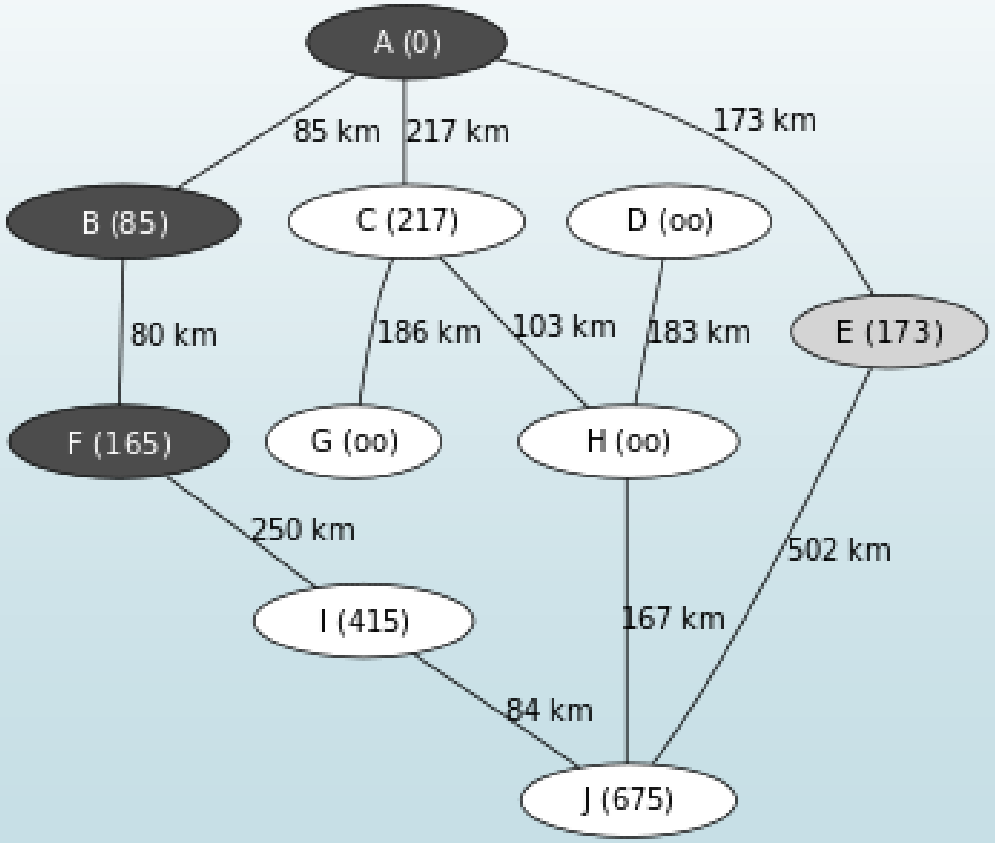
Algoritme de Dijkstra

	A	B	C	D	E	F	G	H	I	J
A	0	85	217	∞	173	165	∞	∞	415	∞
B	85	0	∞	∞	∞	80	∞	∞	330	∞
C	217	∞	0	∞	∞	∞	186	103	∞	∞
D	∞	∞	∞	0	∞	∞	∞	183	∞	∞
E	173	∞	∞	∞	0	∞	∞	∞	∞	502
F	165	80	∞	∞	∞	0	∞	∞	250	∞
G	∞	∞	186	∞	∞	∞	0	∞	∞	∞
H	∞	∞	103	183	∞	∞	∞	0	∞	167
I	415	330	∞	∞	∞	250	∞	∞	0	84
J	∞	∞	∞	∞	502	∞	∞	167	84	0



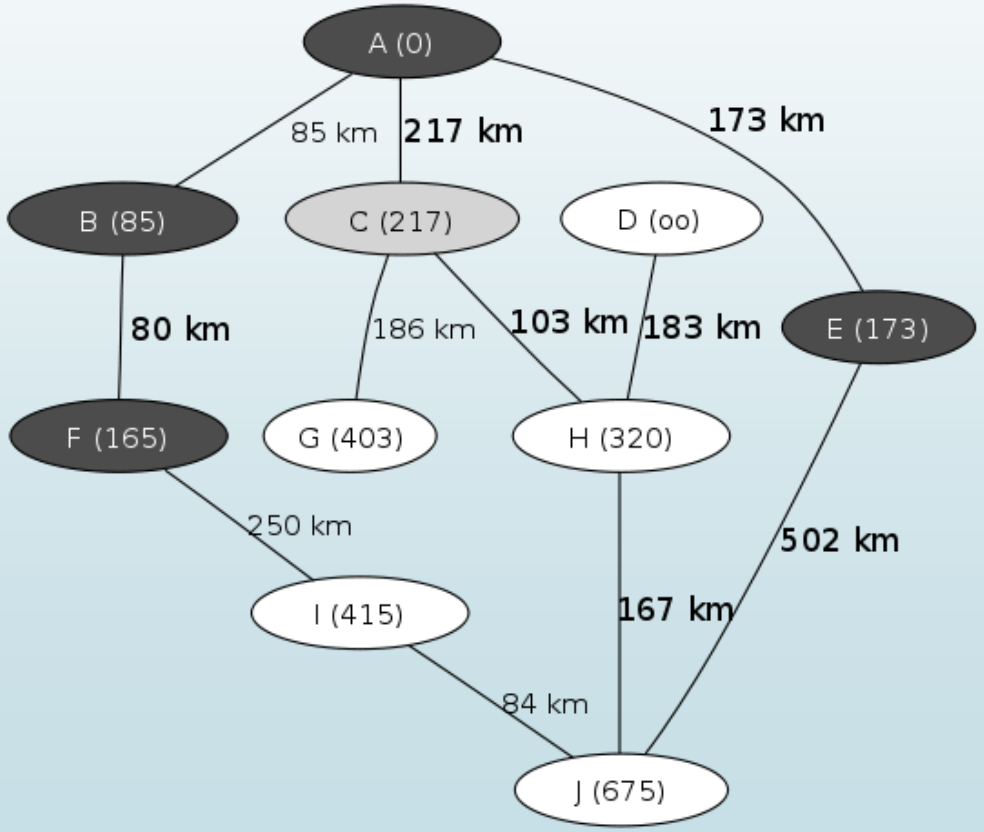
Algoritme de Dijkstra

	A	B	C	D	E	F	G	H	I	J
A	0	85	217	∞	173	165	∞	∞	415	675
B	85	0	∞	∞	∞	80	∞	∞	330	∞
C	217	∞	0	∞	∞	∞	186	103	∞	∞
D	∞	∞	∞	0	∞	∞	∞	183	∞	∞
E	173	∞	∞	∞	0	∞	∞	∞	∞	502
F	165	80	∞	∞	∞	0	∞	∞	250	∞
G	∞	∞	186	∞	∞	∞	0	∞	∞	∞
H	∞	∞	103	183	∞	∞	∞	0	∞	167
I	415	330	∞	∞	∞	250	∞	∞	0	84
J	675	∞	∞	∞	502	∞	∞	167	84	0



Algoritme de Dijkstra

	A	B	C	D	E	F	G	H	I	J
A	0	85	217	∞	173	165	403	320	415	675
B	85	0	∞	∞	∞	80	∞	∞	330	∞
C	217	∞	0	∞	∞	∞	186	103	∞	∞
D	∞	∞	∞	0	∞	∞	∞	183	∞	∞
E	173	∞	∞	∞	0	∞	∞	∞	∞	502
F	165	80	∞	∞	∞	0	∞	∞	250	∞
G	403	∞	186	∞	∞	∞	0	∞	∞	∞
H	320	∞	103	183	∞	∞	∞	0	∞	167
I	415	330	∞	∞	∞	250	∞	∞	0	84
J	675	∞	∞	∞	502	∞	∞	167	84	0



Algoritme de Dijkstra

	à A	à B	à C	à D	à E	à F	à G	à H	à I	à J
A	-	85	217	∞	173	∞	∞	∞	∞	∞
B(85A)	-	-	217	∞	173	165	∞	∞	∞	∞
F(165B)	-	-	217	∞	173	-	∞	∞	415	∞
E(173A)	-	-	217	∞	-	-	∞	∞	415	675
C(217A)	-	-	-	∞	-	-	403	320	415	675
H(320C)	-	-	-	503	-	-	403	-	415	487
G(403C)	-	-	-	503	-	-	-	-	415	487
I(415F)	-	-	-	503	-	-	-	-	-	487
J(487H)	-	-	-	503	-	-	-	-	-	-
D(503H)	-	-	-	-	-	-	-	-	-	-

- Plus court chemin: A-C-H-J
- On obtient aussi toutes les distances minimales du point A aux autres points.

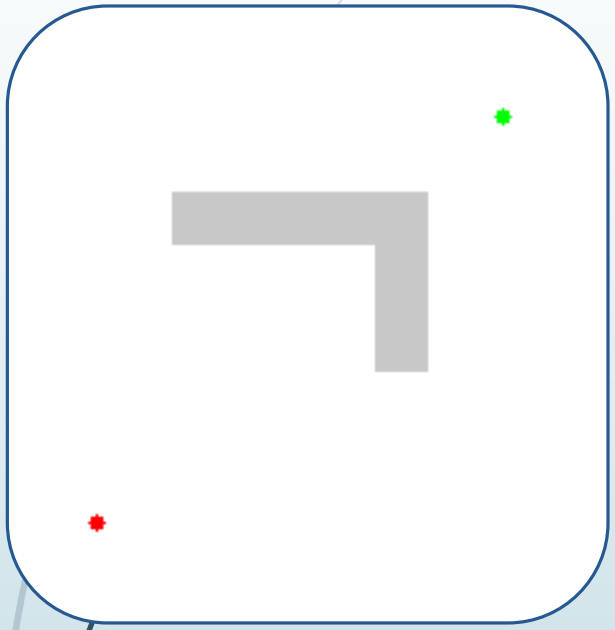
Algorithme A*

- Une extension de l'algorithme de Dijkstra avec ajout d'une heuristique.
- Cout de déplacement d'un nœud p à un nœud n:

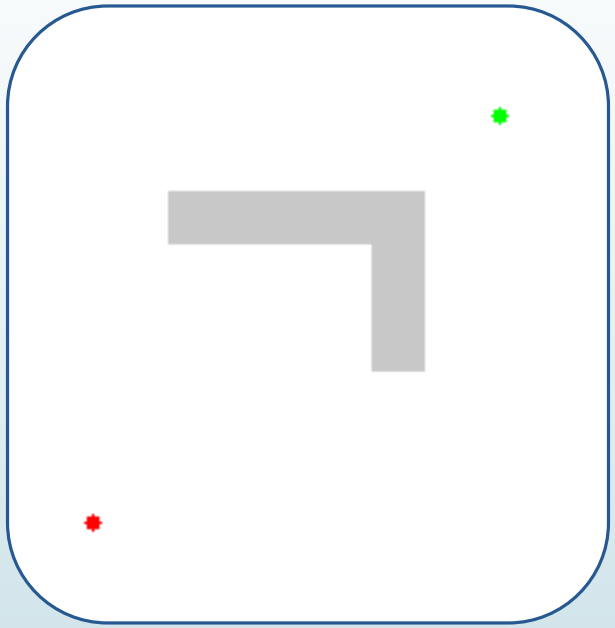
  $$F(n) = Heuristic(n) + stepcost(p,n)$$

- L'heuristique peut être:
  - Une distance (Euclidienne, maximum, de Manhattan, etc..)
  - Une direction à prendre
  - Une orientation
  - Etc..

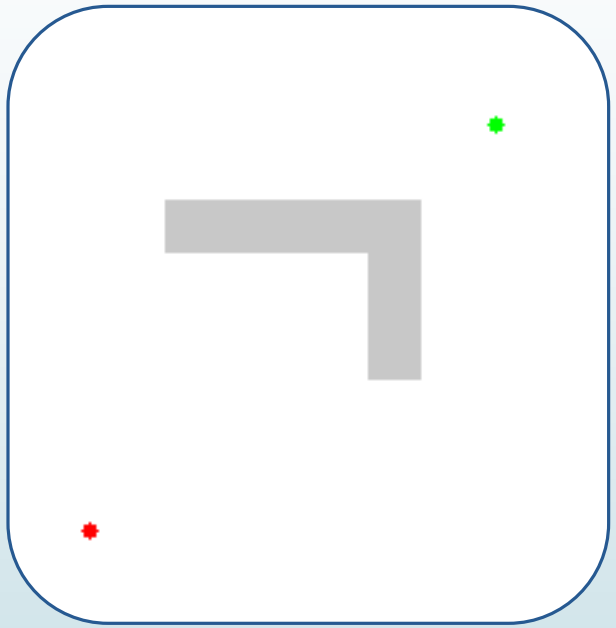
Algorithme A* : comparaison



Dijkstra

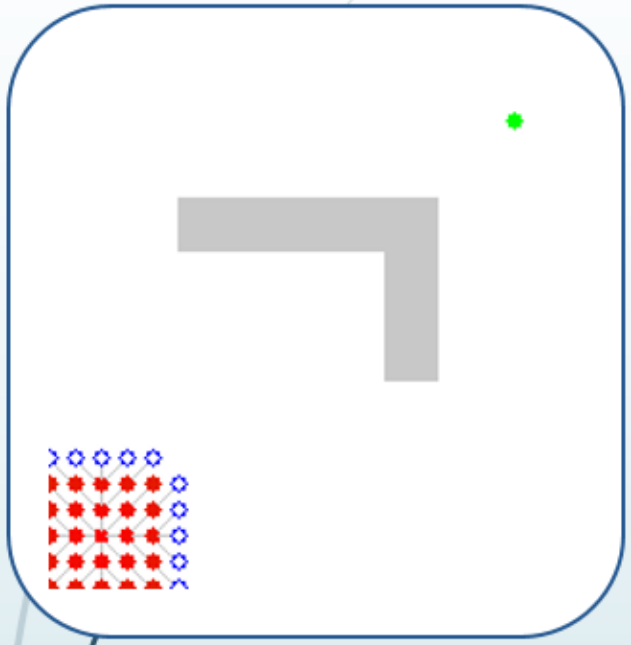


A* heuristique 1

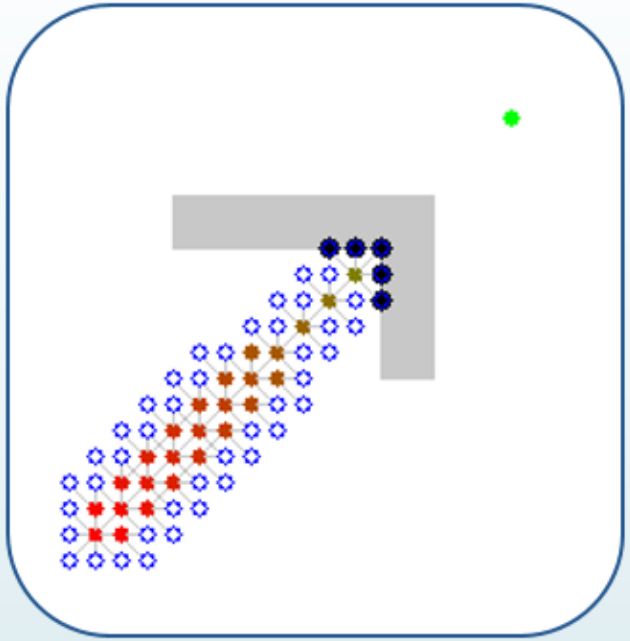


A* heuristique 2

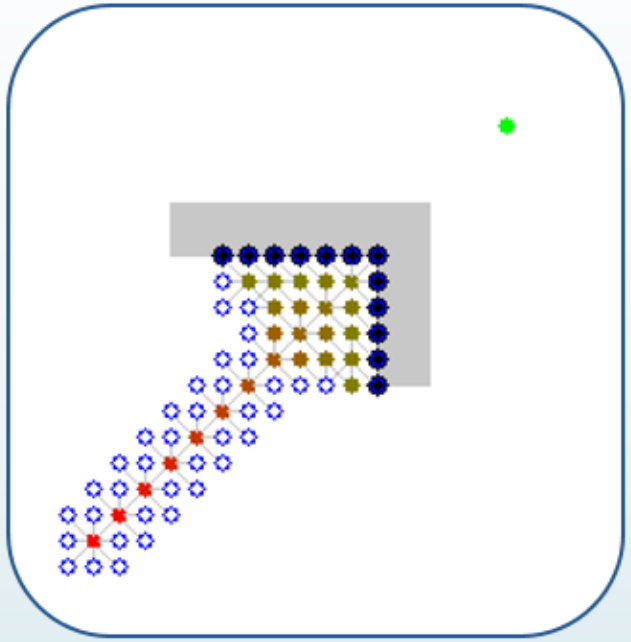
Algorithme A* : comparaison



Dijkstra

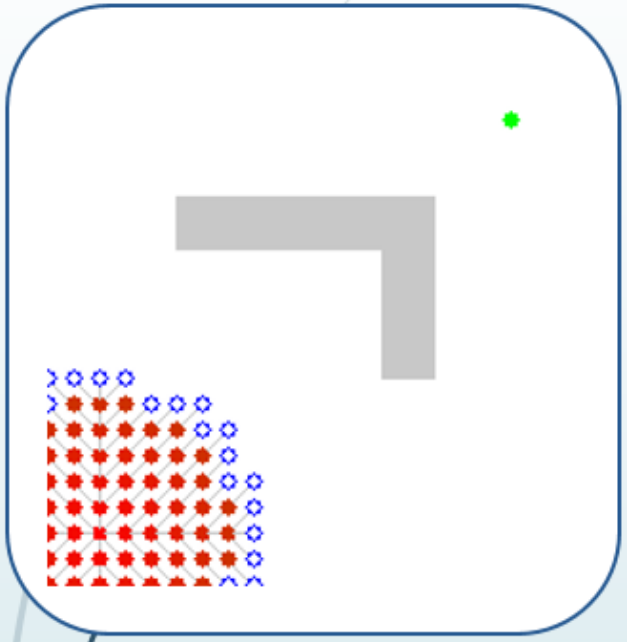


A* heuristique 1

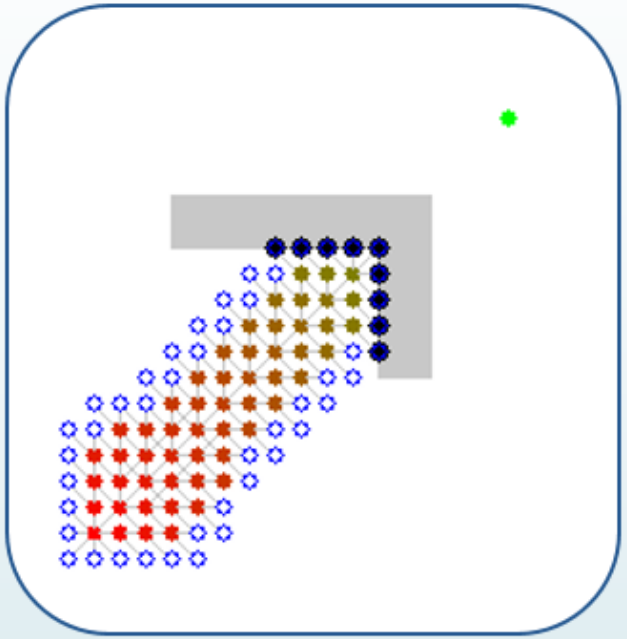


A* heuristique 2

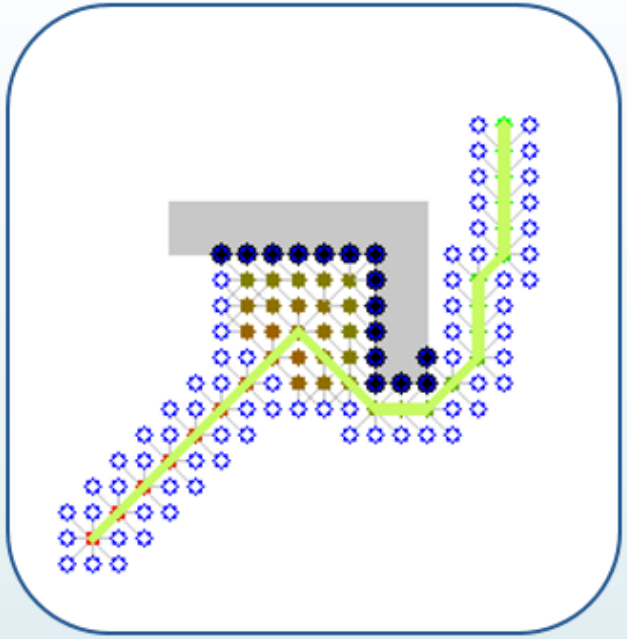
Algorithme A* : comparaison



Dijkstra

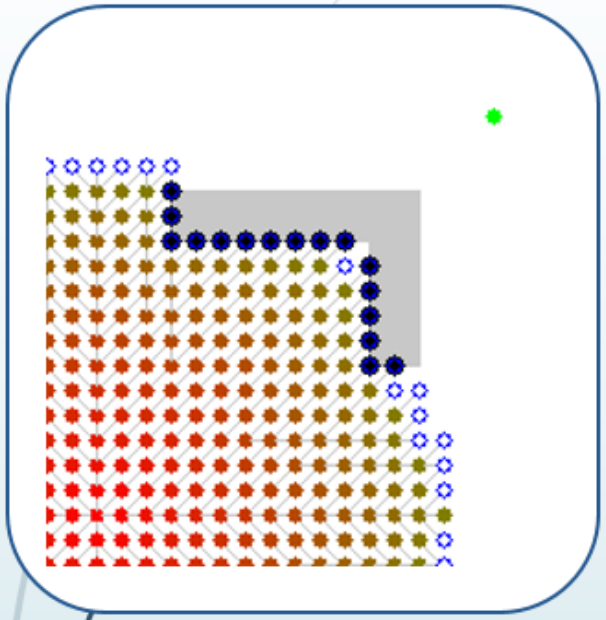


A* heuristique 1

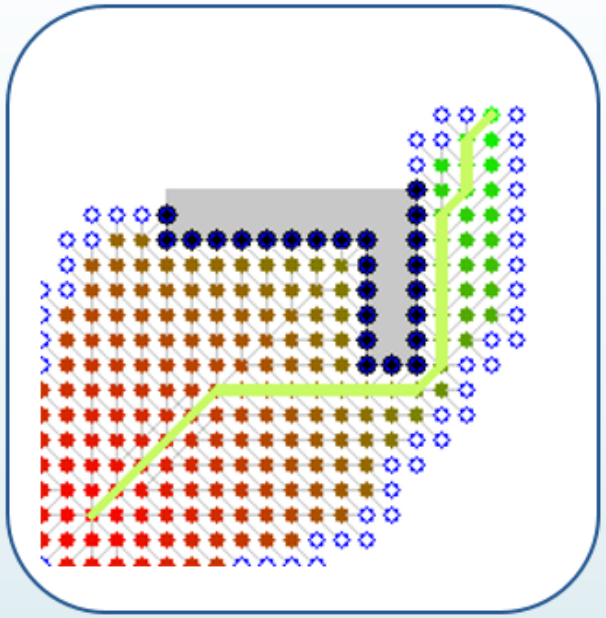


A* heuristique 2

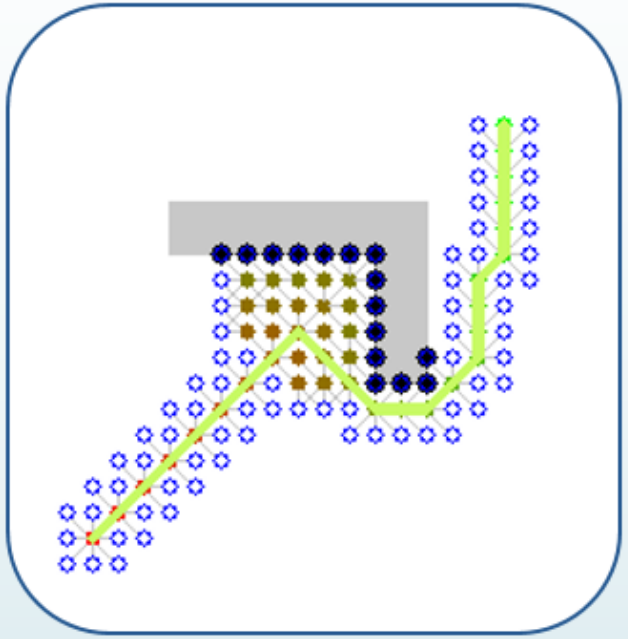
Algorithme A* : comparaison



Dijkstra

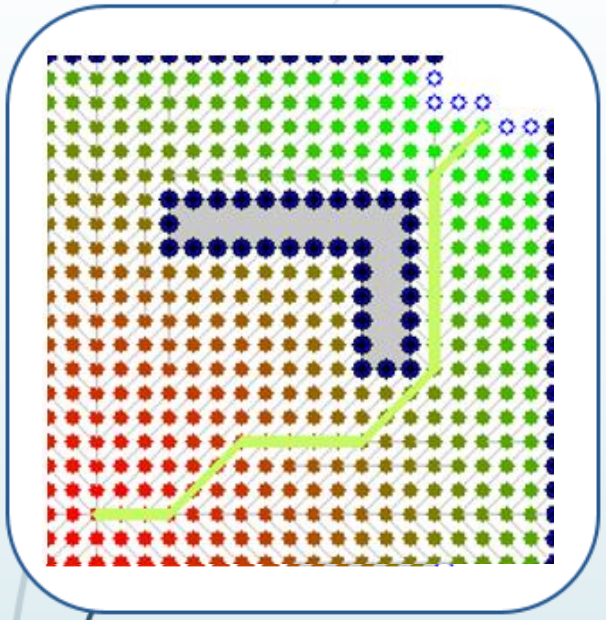


A* heuristique 1

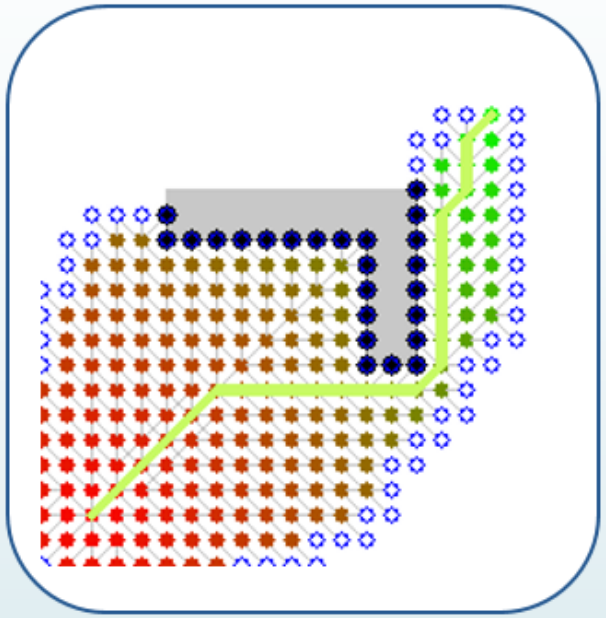


A* heuristique 2

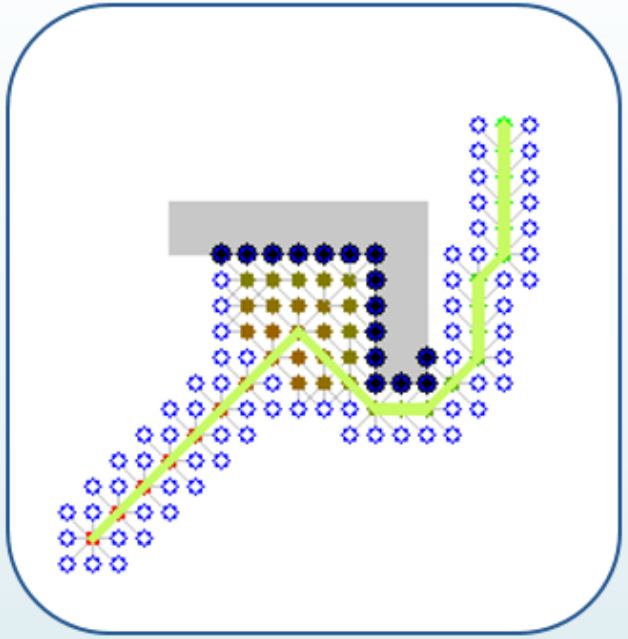
Algorithme A* : comparaison



Dijkstra



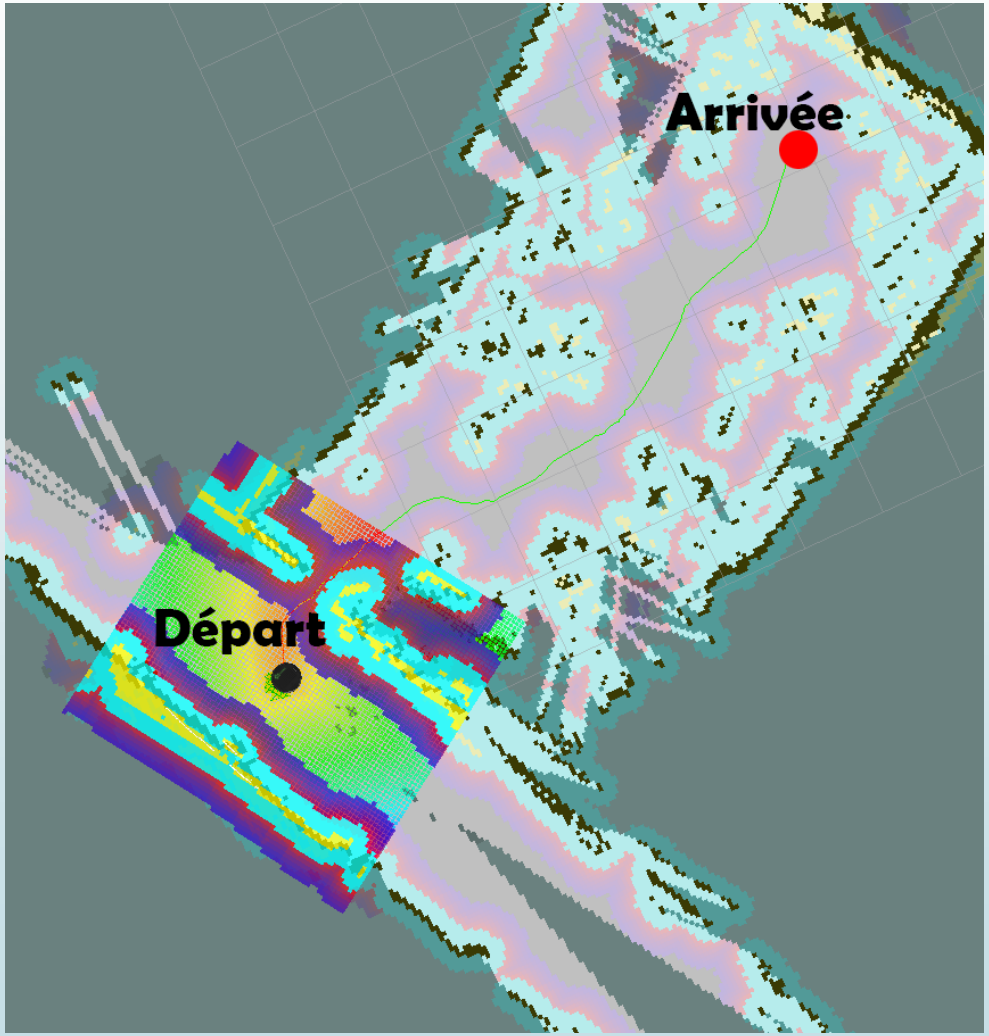
A* heuristique 1



A* heuristique 2

Conclusion et Utilisation

- Résolution de problème à contraintes (schématisation de problèmes…) – Peu utilisé
- Cas de la robotique, utilisation très forte de l'algorithme A*
- Utilisation des algorithmes de plus court chemin dans les GPS



Blibliographie et Suggestions

- Sites:

Site référence et complet sur les différent labyrinthes et leur résolution:
http://www.astrolog.org/labyrnth/algrithm.htm

Algorithme de Pledge:
https://interstices.info/jcms/c_46065/l-algorithme-de-pledge
http://www.apprendre-en-ligne.net/info/algo/algorithmique.pdf

Algorithme Shortest-Path:
http://www.redblobgames.com/pathfinding/a-star/implementation.html
https://en.wikipedia.org/wiki/A*_search_algorithm
https://en.wikipedia.org/wiki/Best-first_search
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Questions?

Complément(1): Limites des algorithmes

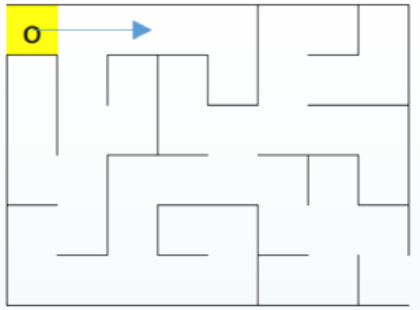
	Pledge	Retour sur trace (Backtracking)	Best-first	Dijkstra	A*
Type de labyrinthe	Graphes	Graphes	Graphes connexes	Graphes connexes	Graphes connexes
Complet/Incomplet	Incomplet	Incomplet	Incomplet	Complet	Complet
Complexité	$O(n^2)$	$O(2^{(n^2/2)})$	O(m^p) (avec p la profondeur maximale de la sortie)	O((m+n)*ln(n)) m arcs et n nœuds.Complexité linéarithmique.	O((m+n)*ln(n)) (dans le pire des cas = complexité de Dijkstra)
Espace mémoire	Très faible	faible	faible	3 listes: • Open: noeud à visiter • Close: noeud visité • Queue: poids des noeuds	Identique à Dijkstra mais en moyenne moins gourmand en ressources mais plus gourmand en calcul
Rapidité	Rapide mais dépend de l'heuristique	Rapide	Rapide	Rapide	Rapide
Solution obtenue	Une solution non garantie	Une solution parmis toute les autres	Une solution parmis toute les autres	La solution la plus rapide	La solution la plus rapide

Complément(2): Pledge Algorithme

```
repeat
        ω=0
    repeat
        Move in direction ω in the free space
    until Robot hits an obstacle
    repeat
        Follow the wall in counter-clockwise direction Count the
        overall turning angle in ω
    until Angle Counter ω =0
until Robot is outside the maze
```

Complément(3) – Backtracking

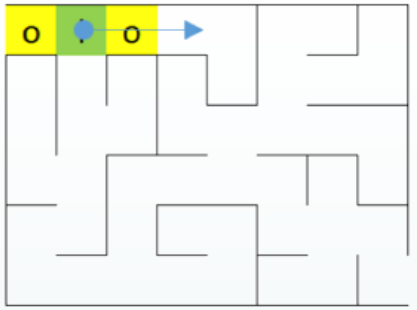
ETAPE 1



- Départ : direction droite choisie

- Chaque case parcourue est mémorisée et marquée par un « o »

ETAPE 2

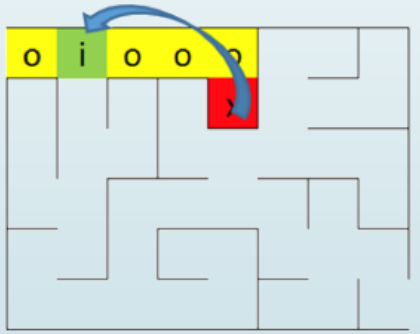


- Rencontre de la première intersection (« i »)

- 2 choix possibles, ces choix peuvent être aléatoires ou basés sur une heuristique

- Choix de la direction droite

ETAPE 3



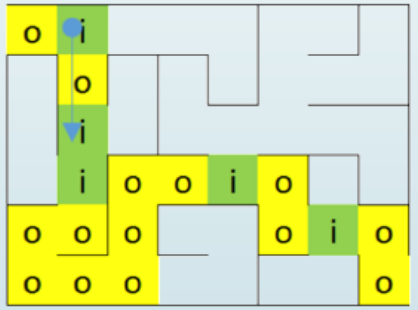
- Parcours de 4 cases vers la droite puis arrivée dans une voie sans issue

- Retour en arrière vers la dernière intersection connue

- Elimination de la direction précédemment choisie

- Choix d'une direction possible, ici en bas

ETAPE 4



- L'algorithme continue ensuite comme ça jusqu'à la sortie

Complément(4) – Backtracking algo

```
fonction Backtracking (cell){
        if cell == exit
                return success ;
        if cell == visited or cell
        == wall
                return fail ;
        Backtracking(cell_right) ;
        Backtracking(cell_forward) ;
        Backtracking(cell_left) ;
```