Projet *Labyrinthe*

Algorithmes et Structures de Données

Juan-Carlos Barros et Daniel Kessler

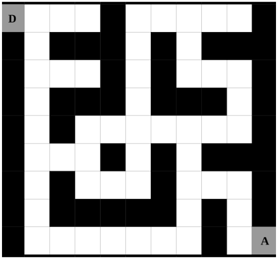
4 juin 2021

Cours d'*Algorithmes* et *Structures de données*

Cours d'*Algorithmes* et *Structures de données*

Projet *Labyrinthe*

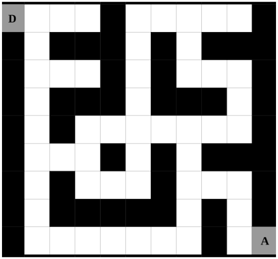
- Algorithme
- Structure de Données



Cours d'*Algorithmes* et *Structures de données*

Projet *Labyrinthe*

- Algorithme
  A*
- Structure de Données



Cours d'*Algorithmes* et *Structures de données*

Projet *Labyrinthe*

- Algorithme
  A*
- Structure de Données
  Priority Queue

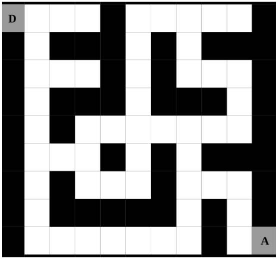


Table des matières

Un labyrinthe, plusieurs problèmes

- Cherche-t-on un chemin quelconque ?
  - Oui, on ne traversera le labyrinthe qu'une seule fois.
  - Non, on veut le chemin le plus court, pour peut-être le réutiliser.

Un labyrinthe, plusieurs problèmes

- Cherche-t-on un chemin quelconque ?
  - Oui, on ne traversera le labyrinthe qu'une seule fois.
  - **Non, on veut le chemin le plus court**, pour peut-être le réutiliser.

Un labyrinthe, plusieurs problèmes

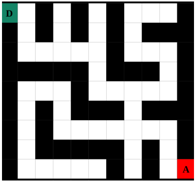
- Cherche-t-on un chemin quelconque ?
  - Oui, on ne traversera le labyrinthe qu'une seule fois.
  - **Non, on veut le chemin le plus court**, pour peut-être le réutiliser.
- Connait-on les coordonnées de la sortie dès le départ ?
  - Oui, et cette information pourra nous aider.
  - Non, le lieu de la sortie fait partie des inconnues.

Un labyrinthe, plusieurs problèmes

- Cherche-t-on un chemin quelconque ?
  - Oui, on ne traversera le labyrinthe qu'une seule fois.
  - **Non, on veut le chemin le plus court**, pour peut-être le réutiliser.
- Connait-on les coordonnées de la sortie dès le départ ?
  - **Oui, et cette information pourra nous aider.**
  - Non, le lieu de la sortie fait partie des inconnues.

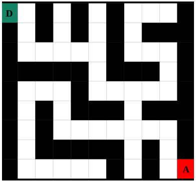
Un problème, plusieurs solutions

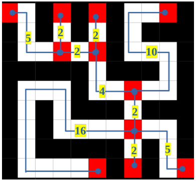
- Breadth-First Search
  - garantit de trouver une solution si elle existe
  - solution optimale si tous les pas sont égaux (même coût)



Un problème, plusieurs solutions

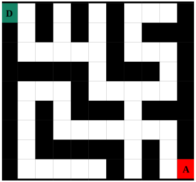
- Breadth-First Search
  - garantit de trouver une solution si elle existe
  - solution optimale si tous les pas sont égaux (même coût)
- Dijkstra
  - choisit où explorer selon les distances déjà parcourues
  - garantit de trouver le plus court chemin (en tenant compte des coûts)

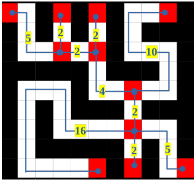




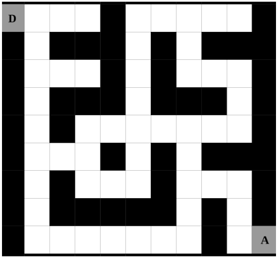
Un problème, plusieurs solutions

- Breadth-First Search
  - garantit de trouver une solution si elle existe
  - solution optimale si tous les pas sont égaux (même coût)
- Dijkstra
  - choisit où explorer selon les distances déjà parcourues
  - garantit de trouver le plus court chemin (en tenant compte des coûts)
- A*
  - nécessite de connaître les coordonnées de la sortie
  - choisit où explorer selon les distances déjà parcourues et une estimation de la distance à la sortie

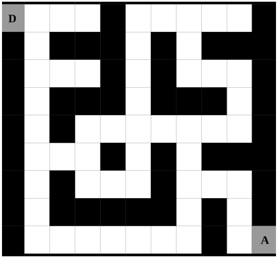




Structure de donnée “labyrinthe”

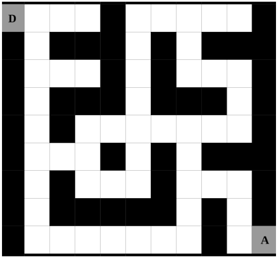


Structure de donnée “labyrinthe”



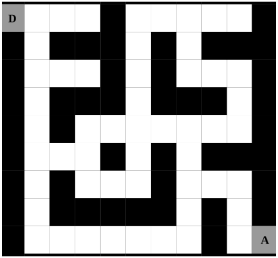
- Solveur peut questionner l’objet :

Structure de donnée “labyrinthe”



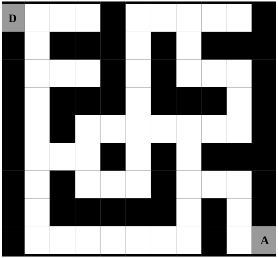
- Solveur peut questionner l’objet :
  - Coordonnées du départ *D*

Structure de donnée “labyrinthe”



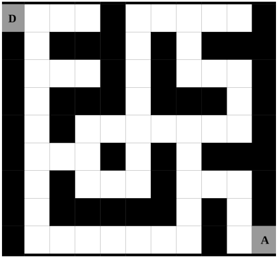
- Solveur peut questionner l’objet :
  - Coordonnées du départ D
  - Coordonnées de l’arrivée A

Structure de donnée “labyrinthe”



- Solveur peut questionner l'objet :
  - Coordonnées du départ D
  - Coordonnées de l'arrivée A
  - Case (x,y) atteignable ?

Structure de donnée "labyrinthe"



- Solveur peut questionner l'objet :
  - Coordonnées du départ D
  - Coordonnées de l'arrivée A
  - Case (x,y) atteignable ?
- Encapsulation pour rendre solveur indépendant de la structure.
- Possible implémentation : via tableau de booléens

Algorithme A*

pseudo-code

Algorithme A*

pseudo-code

Démarrer une **file d'attente** avec la cellule de départ D, un **tableau de prédécesseurs** avec $\{D : Nil\}$ et un **tableau de coûts d'accès** avec $\{D : 0\}$.

Algorithme A*

pseudo-code

Démarrer une **file d'attente** avec la cellule de départ D, un **tableau de prédécesseurs** avec $\{D : Nil\}$ et un **tableau de coûts d'accès** avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

- extraire (pop) la cellule prioritaire C de la file d'attente

Algorithme A*

pseudo-code

Démarrer une **file d'attente** avec la cellule de départ D, un **tableau de prédécesseurs** avec $\{D : Nil\}$ et un **tableau de coûts d'accès** avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

- extraire (pop) la cellule prioritaire C de la file d'attente
- si C est la cellule d'arrivée A, retourner le chemin qui y amène (via backtracking sur les prédécesseurs)

Algorithme A*

pseudo-code

Démarrer une **file d'attente** avec la cellule de départ D, un **tableau de prédécesseurs** avec $\{D : Nil\}$ et un **tableau de coûts d'accès** avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

- extraire (pop) la cellule prioritaire C de la file d'attente
- si C est la cellule d'arrivée A, retourner le chemin qui y amène (via backtracking sur les prédécesseurs)
- sinon, pour chaque voisin V de C qui n'est pas déjà accessible à moindre coût

Algorithme A*

pseudo-code

Démarrer une **file d'attente** avec la cellule de départ D, un **tableau de prédécesseurs** avec $\{D : Nil\}$ et un **tableau de coûts d'accès** avec $\{D : 0\}$.

Tant que la file d'attente n'est pas vide,

- extraire (pop) la cellule prioritaire C de la file d'attente
- si C est la cellule d'arrivée A, retourner le chemin qui y amène (via backtracking sur les prédécesseurs)
- sinon, pour chaque voisin V de C qui n'est pas déjà accessible à moindre coût
  - mémoriser le prédécesseur de V (soit la cellule C) et le coût d'accès à V (soit un de plus que le coût d'accès à C)
  - insérer (push) V dans la file d'attente

Heuristique et Priorité

La priorité d'une cellule C en attente est le coût total (estimé) d'un chemin passant par cette cellule.

$$\texttt{priorité = coût\_réel } (D \rightarrow C) \texttt{ + coût\_estimé } (C \rightarrow A)$$

Heuristique et Priorité

La priorité d'une cellule C en attente est le coût total (estimé) d'un chemin passant par cette cellule.

$$\texttt{priorité = coût\_réel } (D \to C) \texttt{ + coût\_estimé } (C \to A)$$

La distance restante depuis une cellule jusqu'à l'arrivée doit être estimée *sans jamais la surestimer*. $\to$ choix d'heuristique !

Heuristique et Priorité

La priorité d'une cellule C en attente est le coût total (estimé) d'un chemin passant par cette cellule.

$$\texttt{priorité = coût\_réel } (D \to C) \texttt{ + coût\_estimé } (C \to A)$$

La distance restante depuis une cellule jusqu'à l'arrivée doit être estimée *sans jamais la surestimer*. $\to$ choix d'heuristique !

- La **distance de Manhattan** $|\Delta x| + |\Delta y|$ est un bon estimateur si les mouvements permis sont horizontaux et verticaux.

Heuristique et Priorité

La priorité d'une cellule C en attente est le coût total (estimé) d'un chemin passant par cette cellule.

$$\texttt{priorité = coût\_réel } (D \to C) \texttt{ + coût\_estimé } (C \to A)$$

La distance restante depuis une cellule jusqu'à l'arrivée doit être estimée *sans jamais la surestimer*. $\to$ choix d'heuristique !

- La **distance de Manhattan** $|\Delta x| + |\Delta y|$ est un bon estimateur si les mouvements permis sont horizontaux et verticaux.
- Une **heuristique nulle** ramène A* à l'algorithme de Dijkstra (ou à Breadth-First Search si on coût identique pour chaque mouvement).

Preuve de l'algorithme (grandes lignes)

L'heuristique $h(C)$ qui estime le chemin restant depuis une cellule C doit satisfaire deux conditions.

Preuve de l'algorithme (grandes lignes)

L'heuristique $h(C)$ qui estime le chemin restant depuis une cellule C doit satisfaire deux conditions.

1. **Monotonicité** : sorte d'inégalité triangulaire faible $h(C_1) \leq r(C_1, C_2) + h(C_2)$ où C_1, C_2 sont deux cellules, $r(C_1, C_2)$ est la distance réelle entre elles.

   Cette propriété garantit de trouver la sortie, sans se "perdre" dans des boucles éventuelles.

Preuve de l'algorithme (grandes lignes)

L'heuristique $h(C)$ qui estime le chemin restant depuis une cellule C doit satisfaire deux conditions.

1. **Monotonicité** : sorte d'inégalité triangulaire faible $h(C_1) \leq r(C_1, C_2) + h(C_2)$ où C_1, C_2 sont deux cellules, $r(C_1, C_2)$ est la distance réelle entre elles.

   Cette propriété garantit de trouver la sortie, sans se "perdre" dans des boucles éventuelles.

2. **"Admissibilité" : l'heuristique ne surestime jamais une distance**

   Cela garantit qu'il n'y a pas de chemin plus court que celui trouvé.

Preuve de l'algorithme (grandes lignes)

L'heuristique $h(C)$ qui estime le chemin restant depuis une cellule C doit satisfaire deux conditions.

1. **Monotonicité** : sorte d'inégalité triangulaire faible $h(C_1) \leq r(C_1, C_2) + h(C_2)$ où C_1, C_2 sont deux cellules, $r(C_1, C_2)$ est la distance réelle entre elles.

   Cette propriété garantit de trouver la sortie, sans se “perdre” dans des boucles éventuelles.

2. **“Admissibilité” : l'heuristique ne surestime jamais une distance**

   Cela garantit qu'il n'y a pas de chemin plus court que celui trouvé.

   Par contradiction :

Preuve de l'algorithme (grandes lignes)

L'heuristique $h(C)$ qui estime le chemin restant depuis une cellule C doit satisfaire deux conditions.

1. **Monotonicité** : sorte d'inégalité triangulaire faible $h(C_1) \leq r(C_1, C_2) + h(C_2)$ où C_1, C_2 sont deux cellules, $r(C_1, C_2)$ est la distance réelle entre elles.

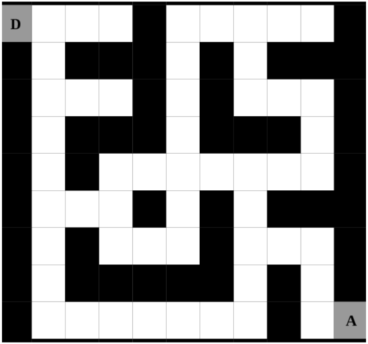
   Cette propriété garantit de trouver la sortie, sans se "perdre" dans des boucles éventuelles.

2. **"Admissibilité" : l'heuristique ne surestime jamais une distance**

   Cela garantit qu'il n'y a pas de chemin plus court que celui trouvé.

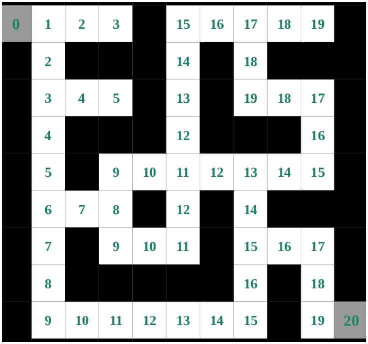
   Par contradiction : s'il y en avait un, il aurait été estimé correctement ou sous-estimé, et serait donc prioritaire par rapport à un chemin complet trop long, vu qu'**un chemin complet a une priorité calculée en coût réel. (coût heuristique de cellule A = 0)**

Exemple de résolution



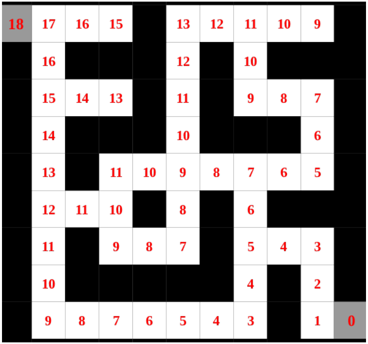
Objectif
Trouver le chemin le plus court entre le **D**épart et l'**A**rrivée

Exemple de résolution



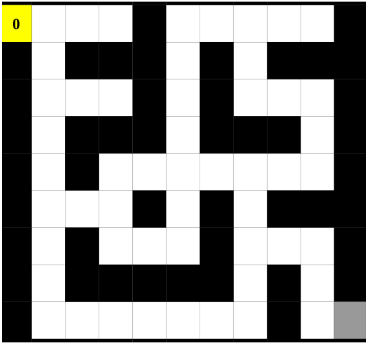
Chaque noeud est à une distance réelle du départ, qui sera découverte en cours de route.

Exemple de résolution



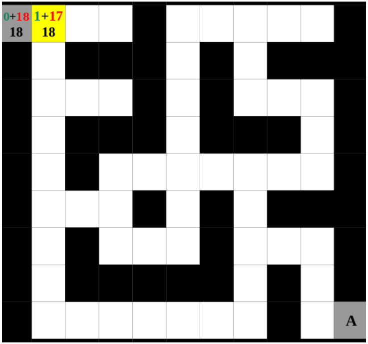
Estimation de la distance réelle de l'arrivée pour chaque cellule (Heuristique "Manhattan").

Exemple de résolution



Le départ est mis en file d'attente, avec une priorité 0.

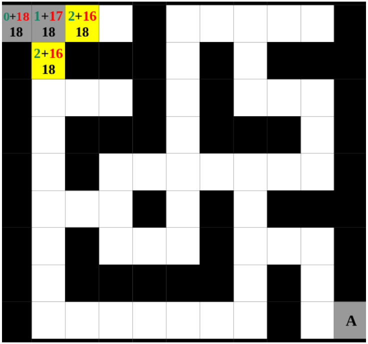
Exemple de résolution



Le seul voisin est évalué :

- coût réel pour y accéder : 1
- coût heuristique pour la suite : 17
- coût heuristique total (priorité) : 18

Exemple de résolution

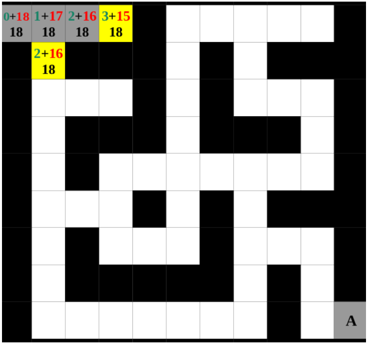


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution



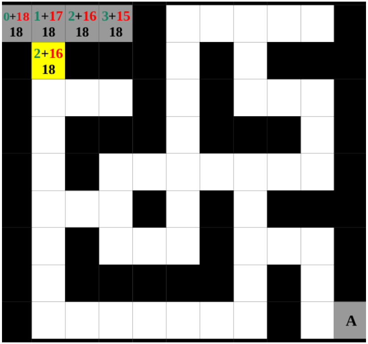
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois deux choix ont la même priorité, le choix est arbitraire.

Exemple de résolution



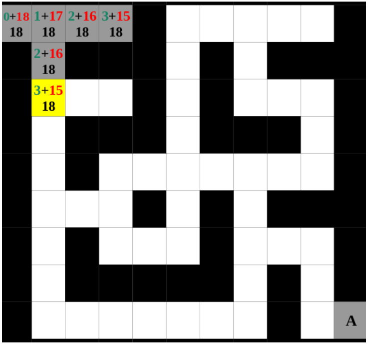
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois deux choix ont la même priorité, le choix est arbitraire.

Exemple de résolution

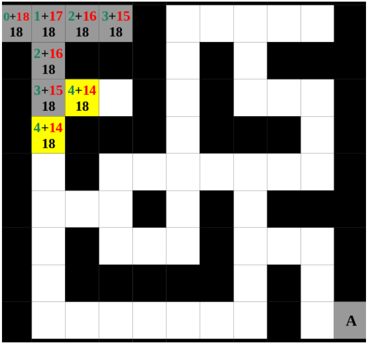


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

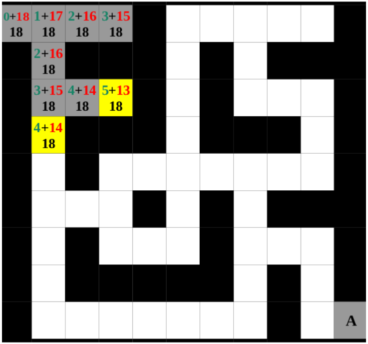


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

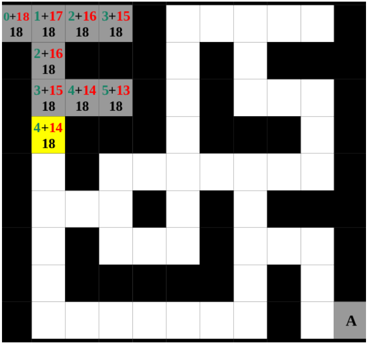


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

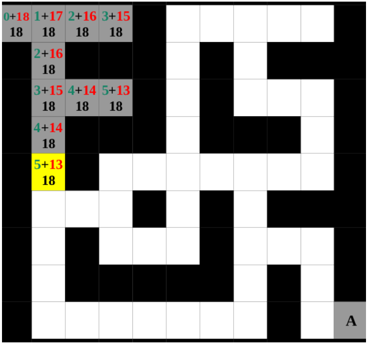


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

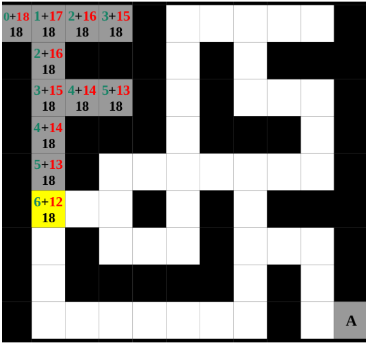


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

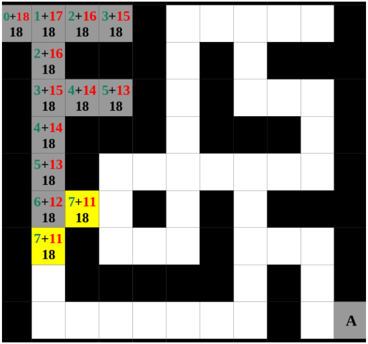


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

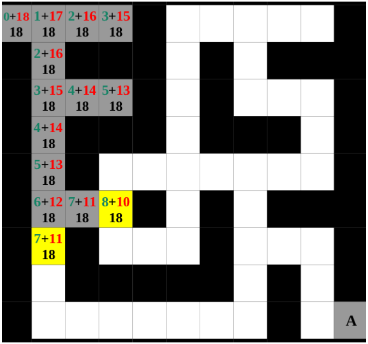


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

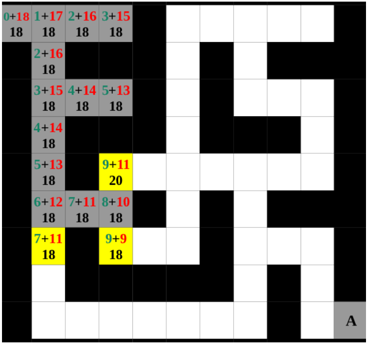


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

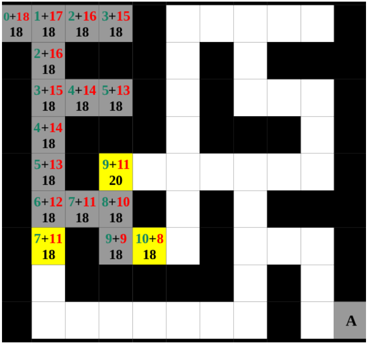


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution



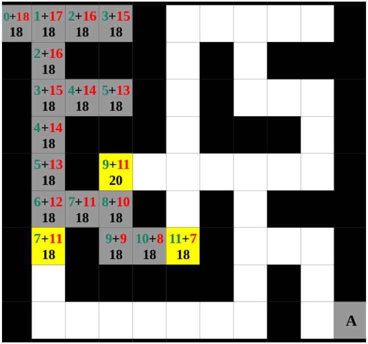
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



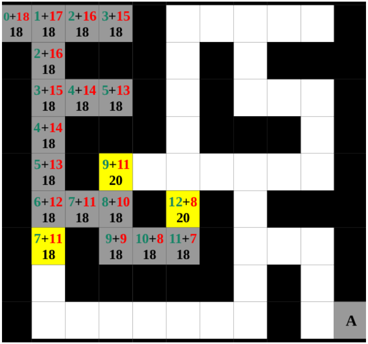
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



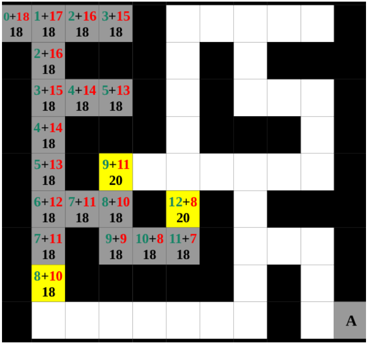
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



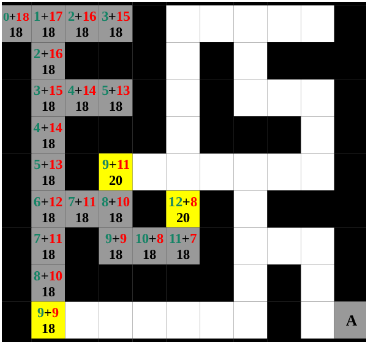
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



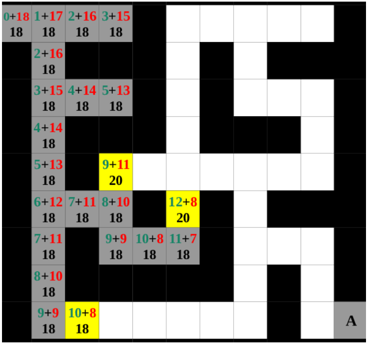
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



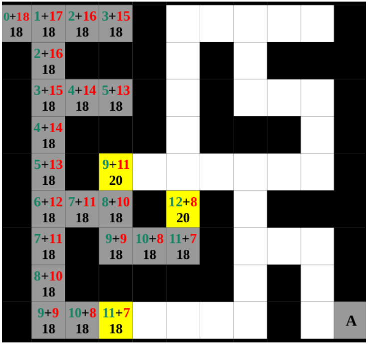
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



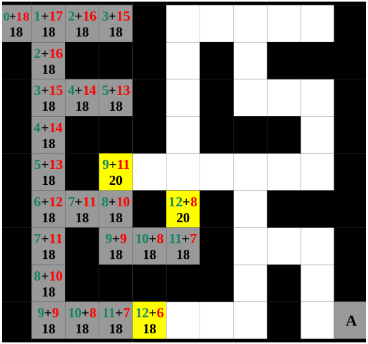
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



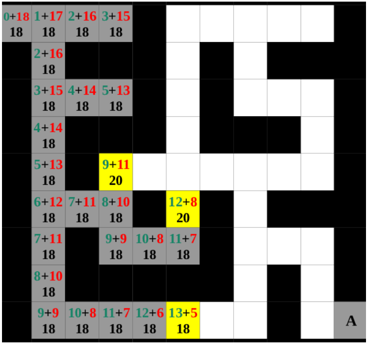
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



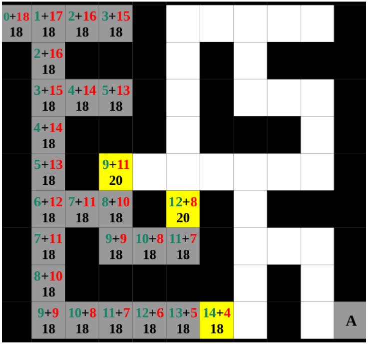
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



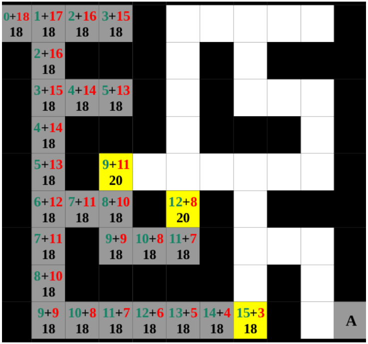
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



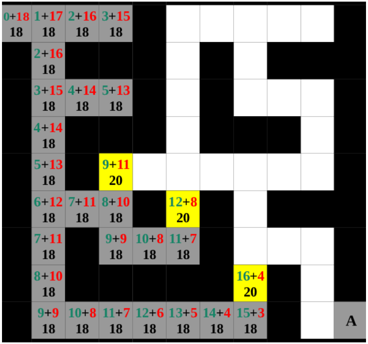
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution



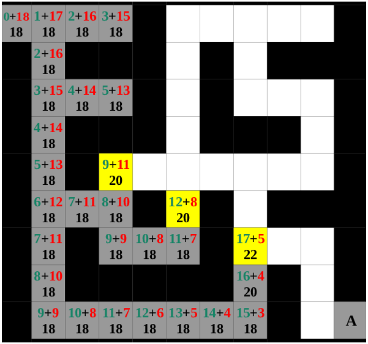
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois la priorité participe vraiment au choix.

Exemple de résolution

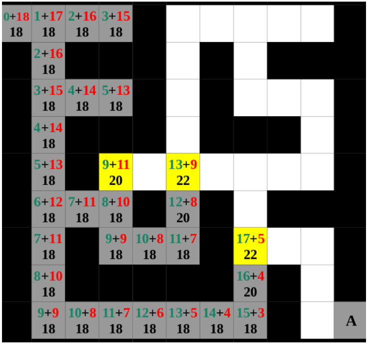


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

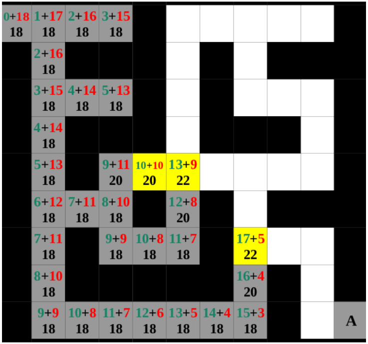


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

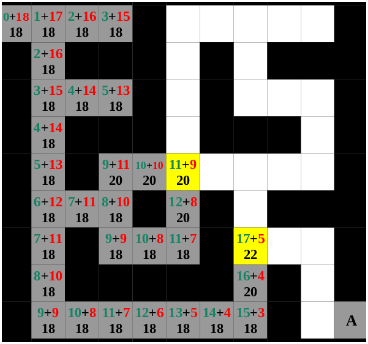


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

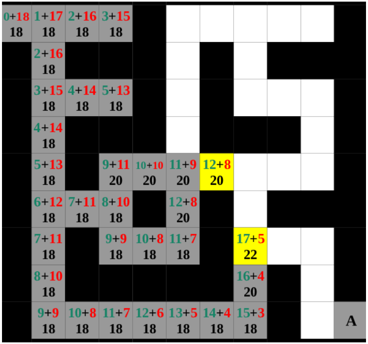


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution



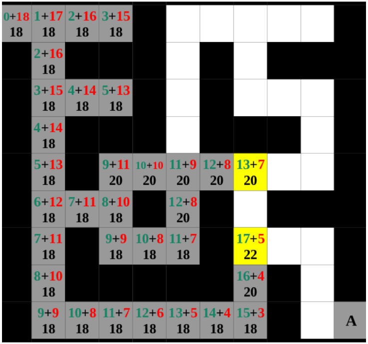
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois un nouveau chemin améliore l'accès à une même cellule.

Exemple de résolution

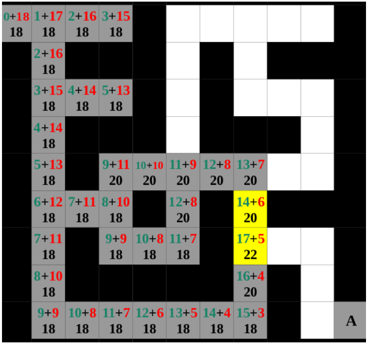


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

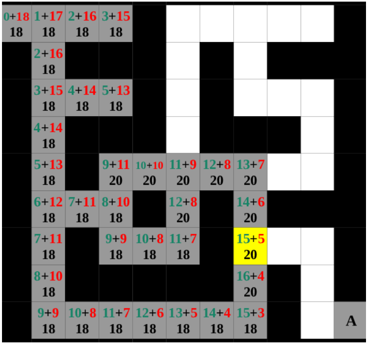


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

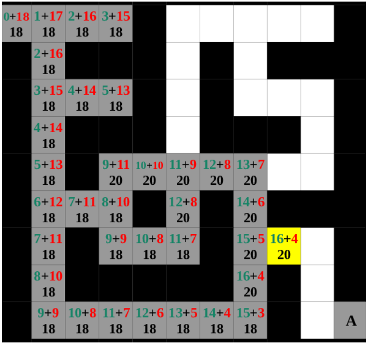


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution



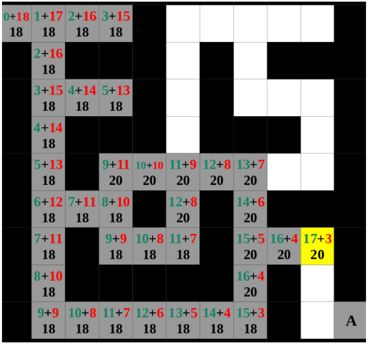
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Parfois un nouveau chemin améliore l'accès à une même cellule.

Exemple de résolution

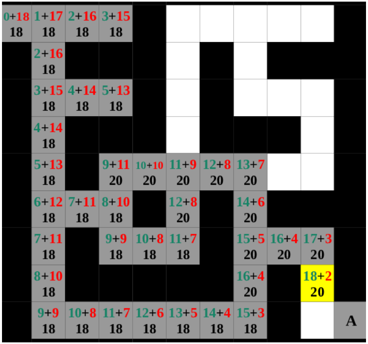


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

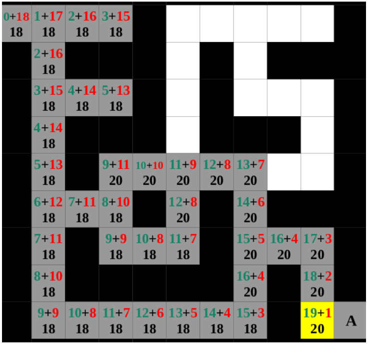


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

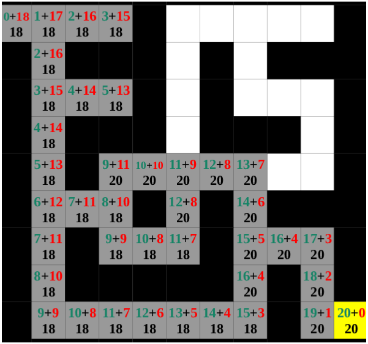


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution

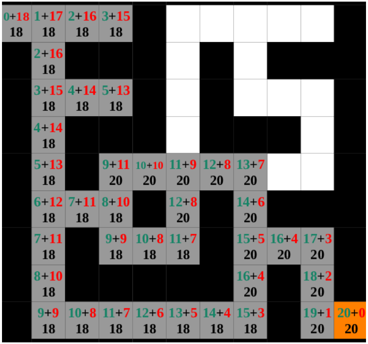


Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

Exemple de résolution



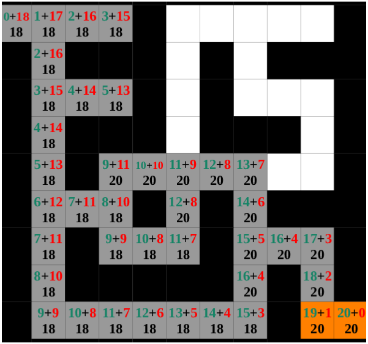
Légende :

- coût réel jusqu'ici
- coût heuristique pour la suite
- coût heuristique total

L'algorithme poursuit son chemin.

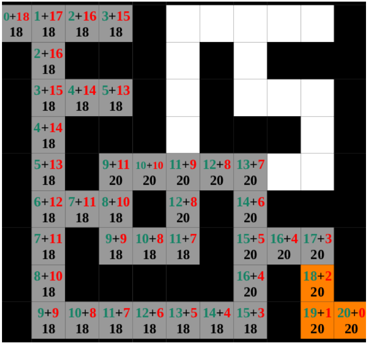
Un chemin vers la sortie a été trouvé !

Exemple de résolution



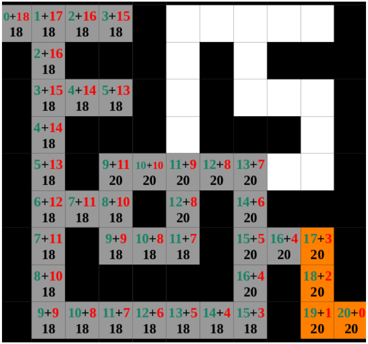
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



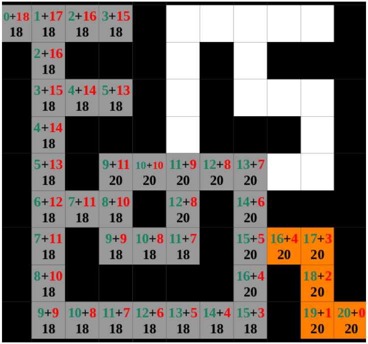
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



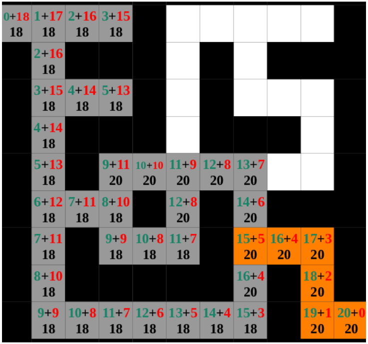
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



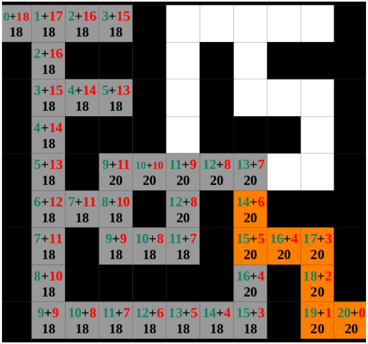
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



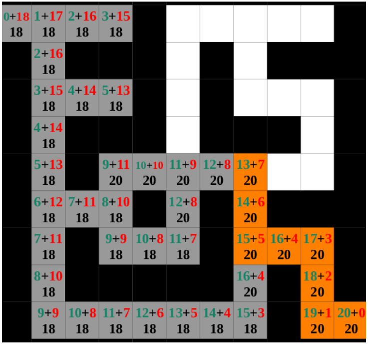
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



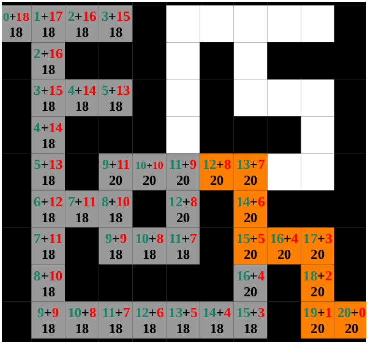
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



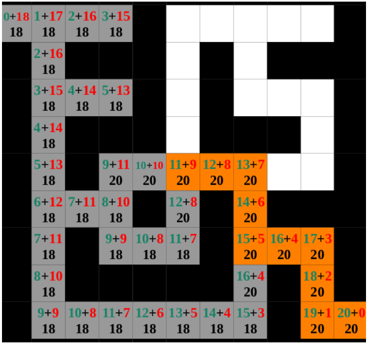
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



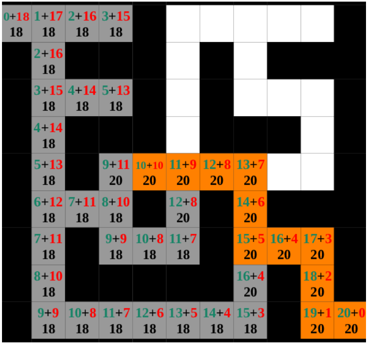
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



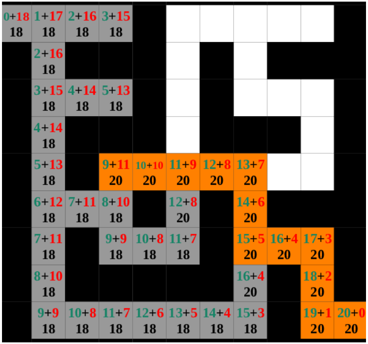
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



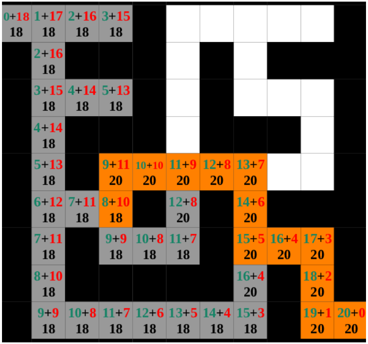
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



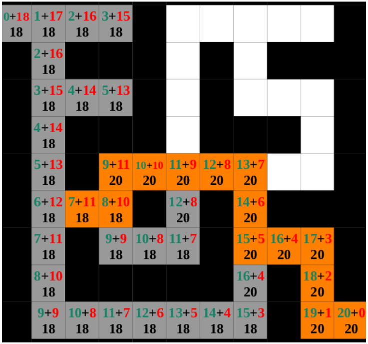
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



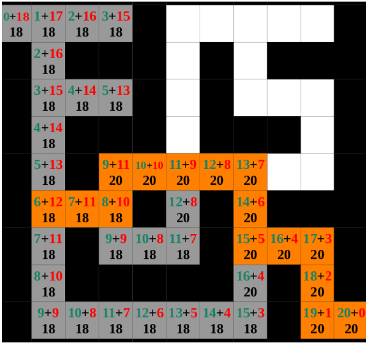
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



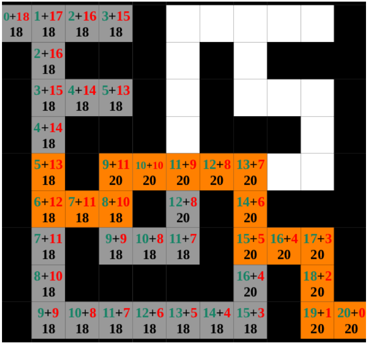
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



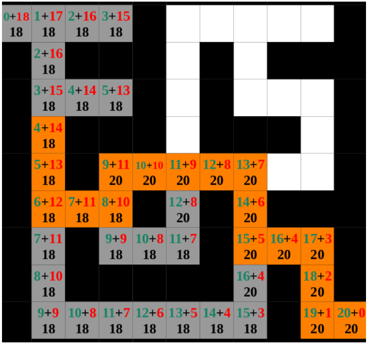
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



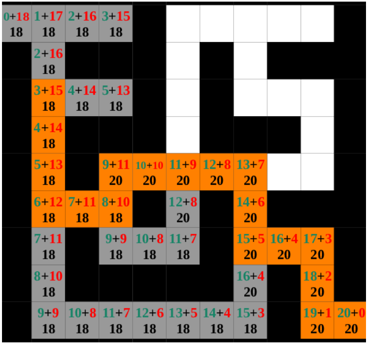
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



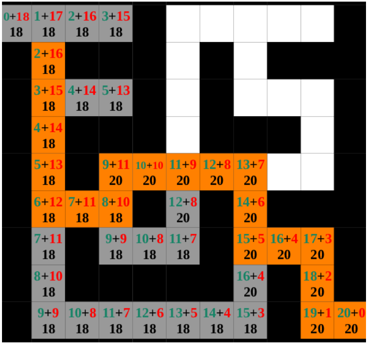
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



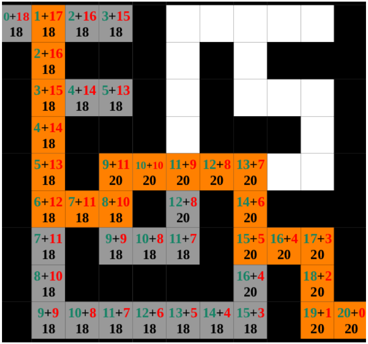
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



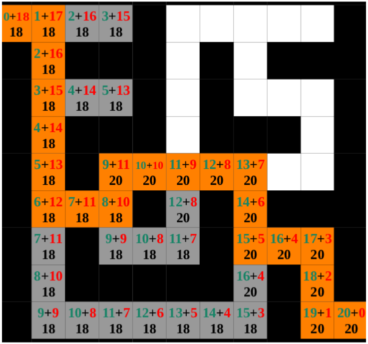
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



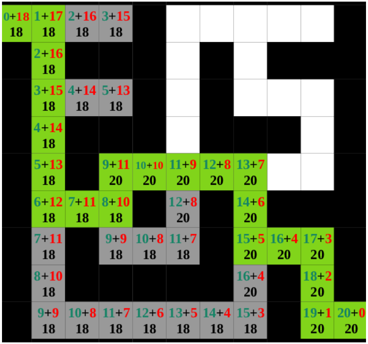
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



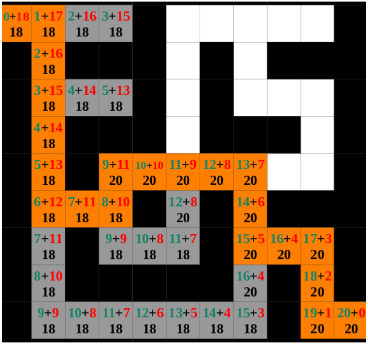
Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Exemple de résolution



Un chemin vers la sortie a été trouvé !
Backtracking pour reconstituer le chemin

Complexité abstraite de l'algorithme - 1

Le pire des cas sera réalisé pour un labyrinthe dont le meilleur chemin revient souvent en arrière (s'éloigne de l'arrivée). Dans le cas extrême, aucun gain n'est réalisé par rapport à l'algorithme de Dijsktra (ou "heuristique nulle").

Complexité abstraite de l'algorithme - 1

Le pire des cas sera réalisé pour un labyrinthe dont le meilleur chemin revient souvent en arrière (s'éloigne de l'arrivée). Dans le cas extrême, aucun gain n'est réalisé par rapport à l'algorithme de Dijsktra (ou "heuristique nulle").

Cependant, le "worst case" ne rend pas justice à A* dont le but est justement d'éviter la plupart du temps le "worst case" avec un bon choix d'heuristique.

Complexité abstraite de l'algorithme - 2

- "worst case" $\rightarrow$ explore (pop) toutes les N cellules.

Complexité abstraite de l'algorithme - 2

- "worst case" $\rightarrow$ explore (pop) toutes les N cellules.
  - **Pop** pour chaque cellule, **push** de chaque voisin atteignable.

Complexité abstraite de l'algorithme - 2

- "worst case" $\rightarrow$ explore (pop) toutes les N cellules.
  - **Pop** pour chaque cellule, **push** de chaque voisin atteignable.
- Le nombre de **pop** sera donc au maximum N.

Complexité abstraite de l'algorithme - 2

- "worst case" $\rightarrow$ explore (pop) toutes les N cellules.
  - **Pop** pour chaque cellule, **push** de chaque voisin atteignable.
- Le nombre de **pop** sera donc au maximum N.
- Le nombre de **push** sera au maximum de A où $A =$ nombre de mouvements possibles.

Complexité abstraite de l'algorithme - 2

- "worst case" $\rightarrow$ explore (pop) toutes les N cellules.
  - **Pop** pour chaque cellule, **push** de chaque voisin atteignable.
- Le nombre de **pop** sera donc au maximum N.
- Le nombre de **push** sera au maximum de A où $A =$ nombre de mouvements possibles.
  - $O(A) \sim O(N)$ vu que chaque cellule possède au maximum 4 voisins atteignables.

Complexité abstraite de l'algorithme - 2

- "worst case" $\rightarrow$ explore (pop) toutes les N cellules.
  - **Pop** pour chaque cellule, **push** de chaque voisin atteignable.
- Le nombre de **pop** sera donc au maximum N.
- Le nombre de **push** sera au maximum de A où $A =$ nombre de mouvements possibles.
  - $O(A) \sim O(N)$ vu que chaque cellule possède au maximum 4 voisins atteignables.

complexité algorithmique

$N \cdot (compl(pop) + compl(push))$

Complexité abstraite de l'algorithme - 2

- "worst case" $\rightarrow$ explore (pop) toutes les N cellules.
  - **Pop** pour chaque cellule, **push** de chaque voisin atteignable.
- Le nombre de **pop** sera donc au maximum N.
- Le nombre de **push** sera au maximum de A où $A =$ nombre de mouvements possibles.
  - $O(A) \sim O(N)$ vu que chaque cellule possède au maximum 4 voisins atteignables.

complexité algorithmique

$N \cdot (compl(pop) + compl(push))$

L'implémentation de la liste d'attente est cruciale. La complexité de **push** et **pop** dépendent en effet du type de file mise en oeuvre !

Complexité abstraite de l'algorithme - 2

- "worst case" $\rightarrow$ explore (pop) toutes les N cellules.
  - **Pop** pour chaque cellule, **push** de chaque voisin atteignable.
- Le nombre de **pop** sera donc au maximum N.
- Le nombre de **push** sera au maximum de A où $A =$ nombre de mouvements possibles.
  - $O(A) \sim O(N)$ vu que chaque cellule possède au maximum 4 voisins atteignables.

complexité algorithmique

$N \cdot (compl(pop) + compl(push))$

L'implémentation de la liste d'attente est cruciale. La complexité de **push** et **pop** dépendent en effet du type de file mise en oeuvre !
Attention : Nous avons négligé les coûts de la gestion des listes de coûts réels et des prédécesseurs. Selon l'implémentation (si lire ou écrire une valeur $> O(\log(N))$), elle pourrait être non négligeable...

File d'attente : "Priority Queue"

- Structure permettant "push" avec priorité et "pop" rapide de l'élément prioritaire
- Implémentation en Python en tant que **binary heap** avec le module *heapq*
- Dans cette implémentation, vérifier si vide en $O(1)$, "push" et "pop" en $O(\log(n))$ où n est le nombre d'objets en attente [1]

1. cf. https ://www.cs.princeton.edu/ wayne/kleinberg-tardos/pdf/BinomialHeaps.pdf

Complexité - structure de données

complexité algorithmique

$N \cdot (compl(pop) + compl(push))$

Complexité - structure de données

complexité algorithmique

$N \cdot (compl(pop) + compl(push))$

- simple tableau : **push** en $O(1)$, **pop** en $O(N)$

complexité totale avec simple tableau

$N \cdot (O(N) + O(1)) = O(N^2)$.

Complexité - structure de données

complexité algorithmique

$N \cdot (compl(pop) + compl(push))$

- simple tableau : **push** en $O(1)$, **pop** en $O(N)$

complexité totale avec simple tableau

$N \cdot (O(N) + O(1)) = O(N^2)$.

- queue prioritaire binaire : **push** en $O(\log(N))$, **pop** en $O(1)$.

complexité totale avec queue prioritaire binaire

$N \cdot (O(1) + O(\log(N))) = O(N \cdot \log(N))$

Tests avec Python

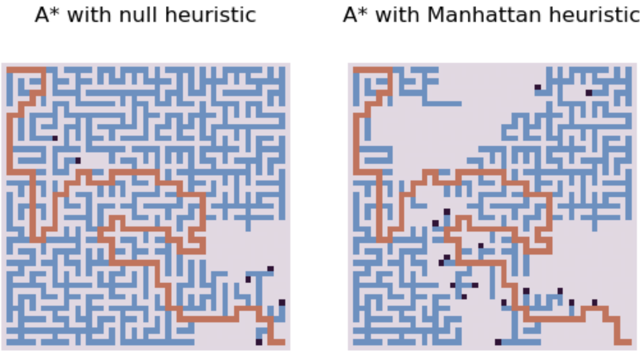
```
marge = QueuePrioritaire(grid.start)
cout_reel = {grid.start: 0}
parent = {grid.start: None}

while True:
    noeud_courant = marge.pop()
    if noeud_courant is None:
        raise ValueError(``la grille n'a pas de solution'')
    if noeud_courant == grid.out:
        break  # chemin optimal trouvé
    # ... traiter noeud courant
```

la suite dans : https://github.com/Dalker/ASD_labyrinthe/

1. cloner le dossier `implementation`
2. exécuter `visual_test.py` et `time_test.py`

Tests avec Python



Tests avec Python

```
* Comparaison heuristique nulle vs Manhattan distance *
solveur1 = heuristique 0, solveur2 = heuristique Manhattan
30x30 : generate=0.1019s solve1=0.0060s solve2=0.0022s
40x40 : generate=0.1814s solve1=0.0096s solve2=0.0086s
50x50 : generate=0.5002s solve1=0.0228s solve2=0.0204s
60x60 : generate=0.8701s solve1=0.0280s solve2=0.0199s
70x70 : generate=1.0461s solve1=0.0451s solve2=0.0391s
80x80 : generate=1.4218s solve1=0.0563s solve2=0.0423s
```

Conclusion - Autres “Path Finding” Ants

≪ACO≫ : Ant Colony Optimisation

Conclusion - Autres “Path Finding” Ants

≪ACO≫ : Ant Colony Optimisation

- imite la nature avec dépôt de phéromones

Conclusion - Autres "Path Finding" Ants

≪ACO≫ : Ant Colony Optimisation

- imite la nature avec dépôt de phéromones
- évaporation des phéromones → disparition des chemins longs

Conclusion - Autres “Path Finding” Ants

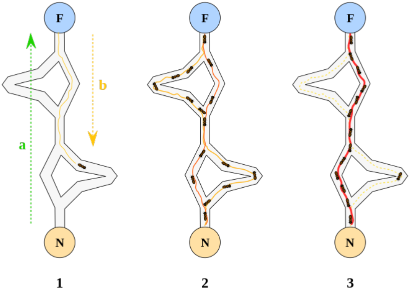
≪ACO≫ : Ant Colony Optimisation

- imite la nature avec dépôt de phéromones
- évaporation des phéromones → disparition des chemins longs
- Avantage : adaptable si le labyrinthe évolue au cours du temps !

Conclusion - Autres "Path Finding" Ants

≪ACO≫ : Ant Colony Optimisation

- imite la nature avec dépôt de phéromones
- évaporation des phéromones → disparition des chemins longs
- Avantage : adaptable si le labyrinthe évolue au cours du temps !
- Désavantage : necessite un grand nombre d'individus



- 1. 1ère fourmi revient de l'arrivée (nourriture) vers le départ (nid) en laissant une piste de phéromone
- 2. Autres fourmis parcourent des chemins parfois légèrement différents
- 3. Les chemins plus longs disparaissent à cause de l'évaporation plus forte des phéromones.

Johann Dréo —
https://commons.wikimedia.org/w/index.php?curid=815564

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

- But un peu différent car doivent passer partout 1 seule fois

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

- But un peu différent car doivent passer partout 1 seule fois
- mais doivent aussi trouver un chemin d’un point A vers un point B

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

- But un peu différent car doivent passer partout 1 seule fois
- mais doivent aussi trouver un chemin d’un point A vers un point B
- utilisent souvent des algorithmes évolutionnistes :

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

- But un peu différent car doivent passer partout 1 seule fois
- mais doivent aussi trouver un chemin d’un point A vers un point B
- utilisent souvent des algorithmes évolutionnistes :
- Gènes = paramètres à optimiser (chemins)

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

- But un peu différent car doivent passer partout 1 seule fois
- mais doivent aussi trouver un chemin d’un point A vers un point B
- utilisent souvent des algorithmes évolutionnistes :
- Gènes = paramètres à optimiser (chemins)
- Transmission de gènes, via reproduction ”d’individus”

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

- But un peu différent car doivent passer partout 1 seule fois
- mais doivent aussi trouver un chemin d’un point A vers un point B
- utilisent souvent des algorithmes évolutionnistes :
- Gènes = paramètres à optimiser (chemins)
- Transmission de gènes, via reproduction ”d’individus”
- Mélanges + mutations → calcul d’adaptabilité

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

- But un peu différent car doivent passer partout 1 seule fois
- mais doivent aussi trouver un chemin d’un point A vers un point B
- utilisent souvent des algorithmes évolutionnistes :
- Gènes = paramètres à optimiser (chemins)
- Transmission de gènes, via reproduction ”d’individus”
- Mélanges + mutations → calcul d’adaptabilité
- Elimination statistique des individus moins performants

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

- But un peu différent car doivent passer partout 1 seule fois
- mais doivent aussi trouver un chemin d'un point A vers un point B
- utilisent souvent des algorithmes évolutionnistes :
- Gènes = paramètres à optimiser (chemins)
- Transmission de gènes, via reproduction "d'individus"
- Mélanges + mutations → calcul d'adaptabilité
- Elimination statistique des individus moins performants
- Avantage : Adaptable si le labyrinthe se modifie

Conclusion - Autres “Path Finding” Robots

Robots aspirateurs

- But un peu différent car doivent passer partout 1 seule fois
- mais doivent aussi trouver un chemin d’un point A vers un point B
- utilisent souvent des algorithmes évolutionnistes :
- Gènes = paramètres à optimiser (chemins)
- Transmission de gènes, via reproduction "d’individus"
- Mélanges + mutations → calcul d’adaptabilité
- Elimination statistique des individus moins performants
- Avantage : Adaptable si le labyrinthe se modifie
- Désavantage : Besoin de beaucoup d’individus et beaucoup de générations pour parvenir à un bon résultat

Références

- Liste des sources consultées :
  https://github.com/Dalker/ASD_labyrinthe/wiki/Sources
- Notre implémentation en Python, avec tests temporels et “tests visuels” (animations) :
  https://github.com/Dalker/ASD_labyrinthe/tree/main/implementation