RAPPORT EXPLICATIF

UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

du projet d'Info 4A - Labyrinthe en C et C++

PARCOURS MAX

Partie 4B - implémentation de distMin
Contribution personnelle - Génération : Algorithme avec backtracking

Par : Evan PETIT - IE4 I912
Professeur de TP : M A. OUAZZANI

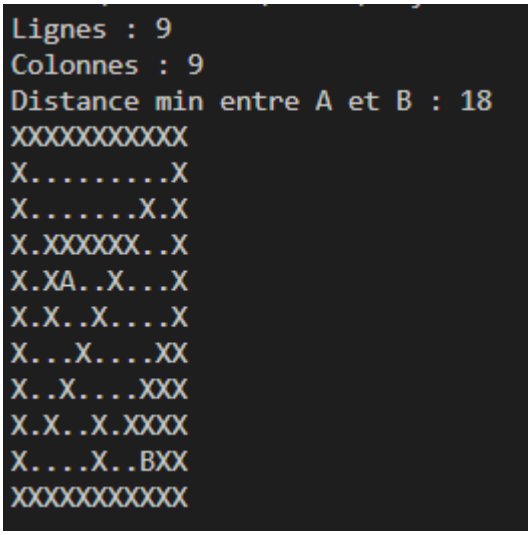
SOMMAIRE

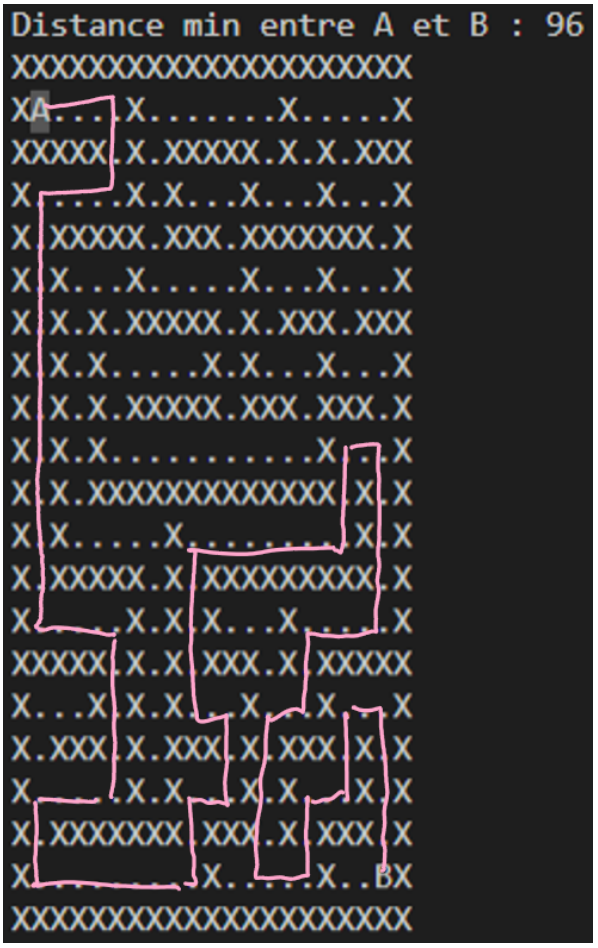
1. Partie 4B

2. Contribution personnelle

Tests effectués

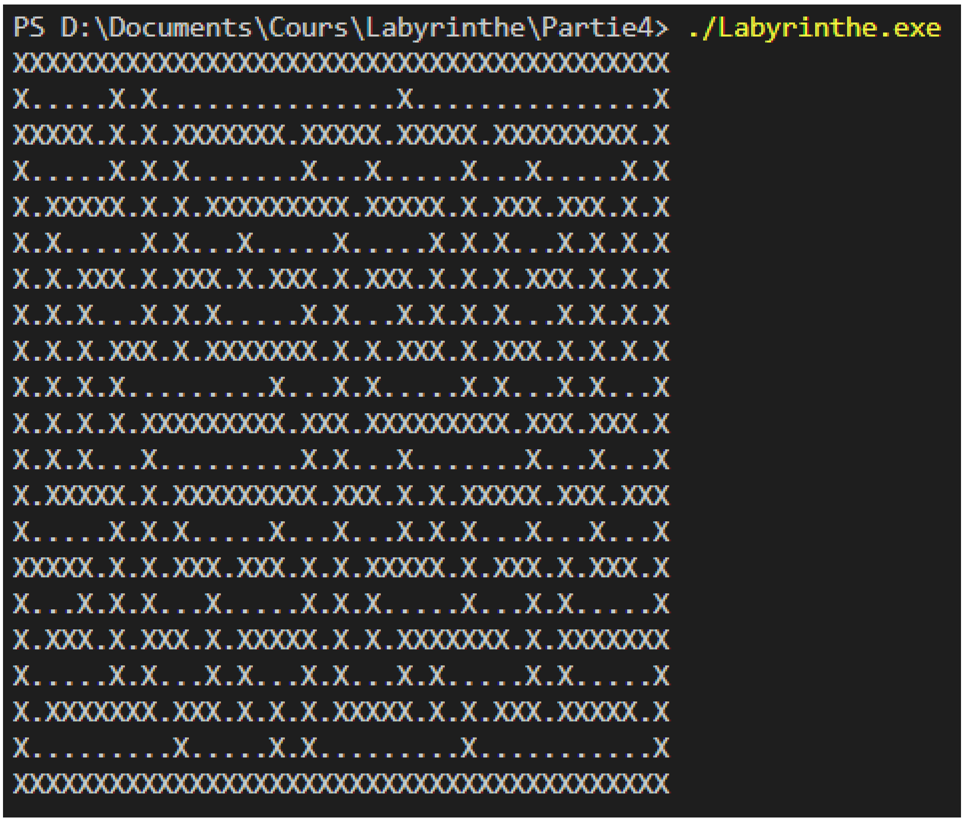
En effectuant quelques tests, on peut vite vérifier que cet algorithme fonctionne correctement.
Il suffit de créer quelques labyrinthes à l'aide de descripteurs, ou bien d'en générer avec les algorithmes de génération. Voici deux exemples :



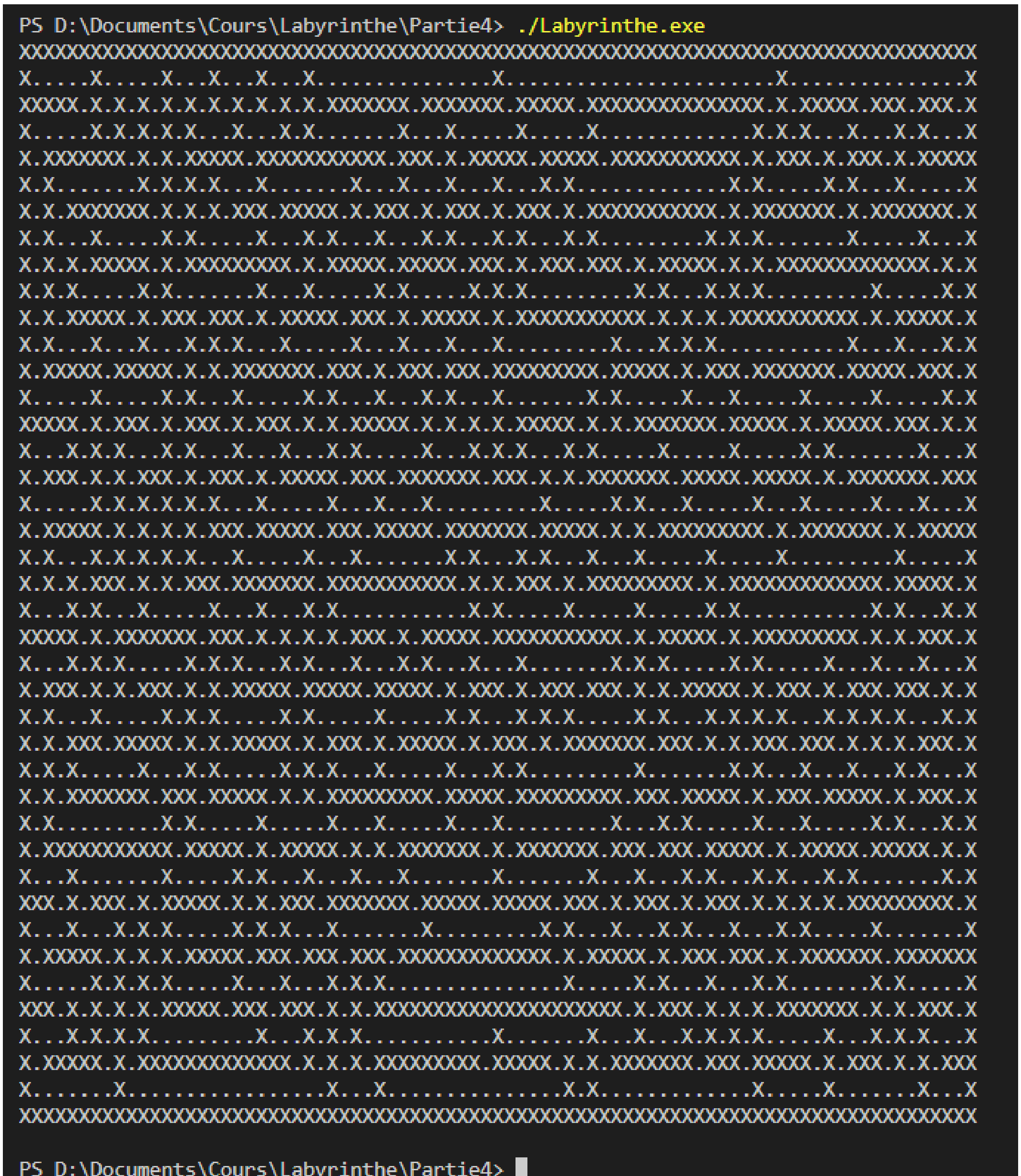


Tester l'algorithme avec backtracking

Il est situé dans la fonction *genLabyBack()* de la classe labyrinthe.
Pour le tester, il suffit de rentrer dans la fonction main() :
labyrinthe lab = new Labyrinthe(lignes, colonnes);*
lab -> genLabyBack();
lab -> affiche();



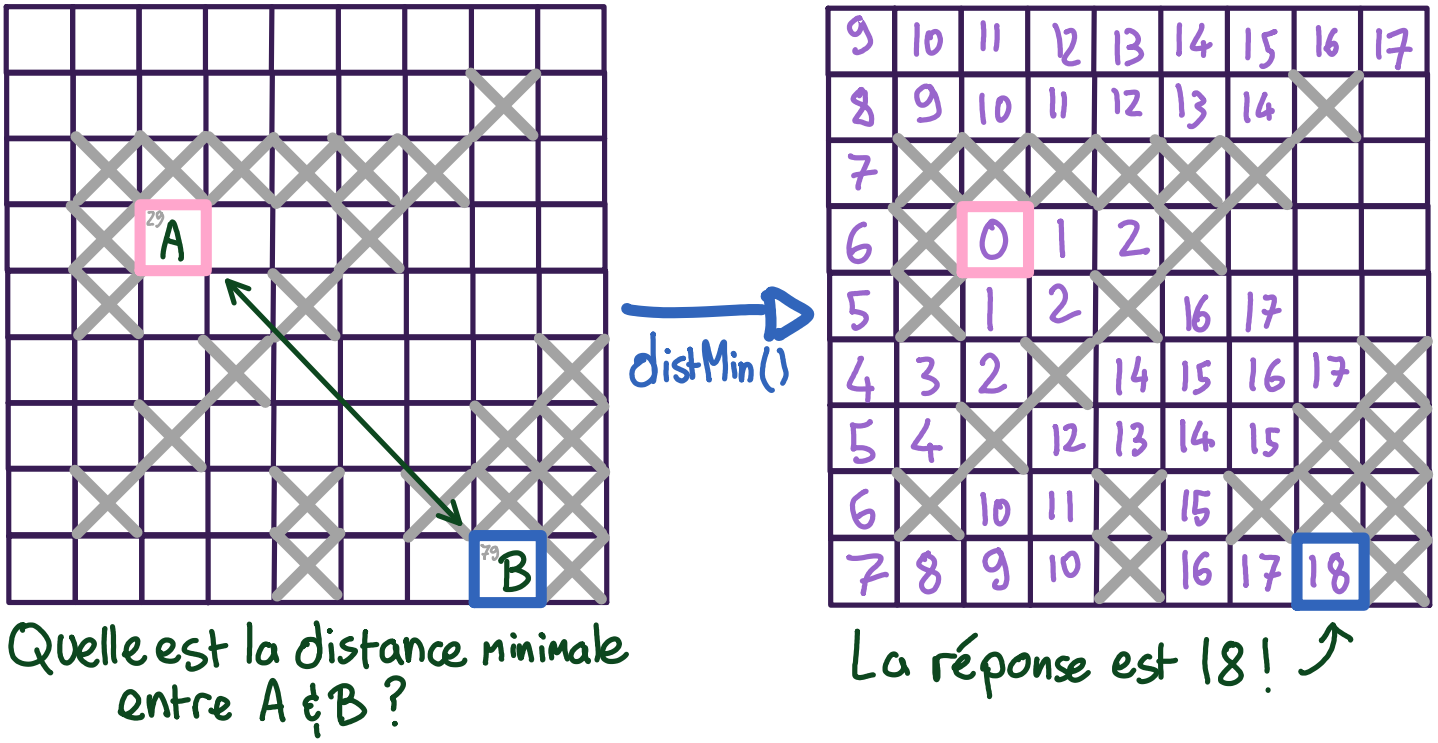
Ainsi on peut le tester. Les labyrinthes peuvent atteindre des tailles phénoménales car l'algorithme en un temps quasi linéaire (on casse un mur par case) + quelques autres constantes (remonter la pile pour le backtracking). Des labyrinthes 1000*1000 sont générés en une poignée de secondes.



Algorithme de recherche de la distance minimale

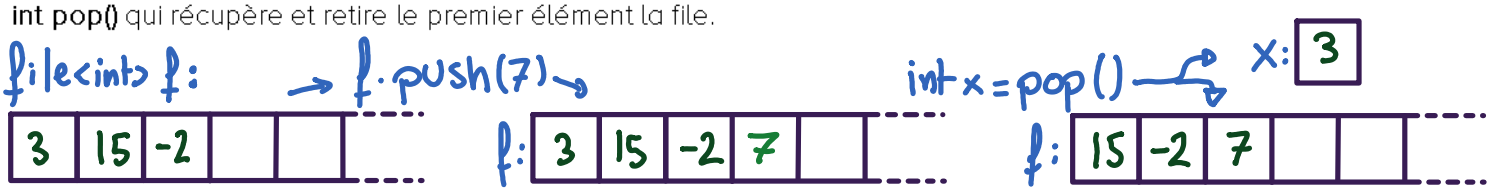
L'algorithme de recherche de la distance minimale (permettant de trouver la plus petite distance entre une cellule de départ et une cellule d'arrivée) s'effectue de manière itérative.
On associe à la cellule de départ la valeur 0. Ensuite, la valeur 1 à toutes les cellules vides voisines. Ensuite la valeur 2 à toutes les cellules vides voisines à celles ci, etc... Jusqu'à atteindre la cellule d'arrivée. La valeur associée à cette cellule est alors la distance minimale recherchée.

Exemple de fonctionnement :



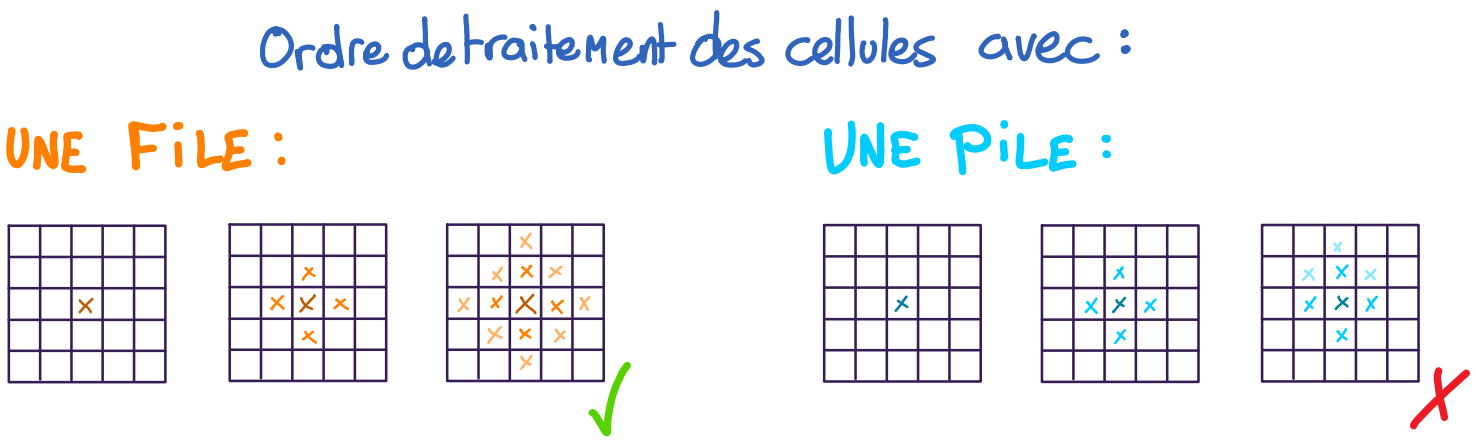
Structure de donnée utilisée

On utilisera une structure de donnée particulière : La file. Il s'agit d'une structure "FIFO", *last-in last-out*, ce qui signifie premier entré, premier sorti. La file est munie de deux méthodes : **push(int x)** qui ajoute un entier en dernière position, **int pop()** qui récupère et retire le premier élément la file.



Pourquoi utiliser une file et pas une autre structure, comme une pile? A chaque étape, on s'éloigne de plus en plus de la case de départ.
La file permet d'évoluer couche par couche autour de la case de départ car on traite les cellules dans leur ordre d'arrivée, on explore donc tous les chemins en même temps. Avec une pile, on traiterait d'abord un chemin, puis ensuite l'autre (ce qu'on ne souhaite pas).

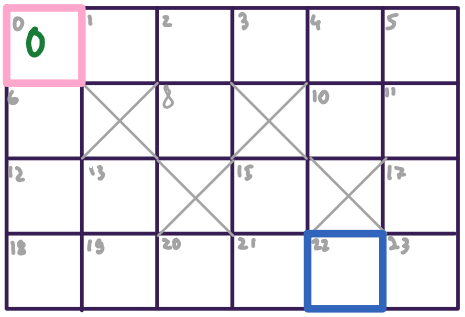
Ordre de traitement des cellules avec :



Traitement étape par étape

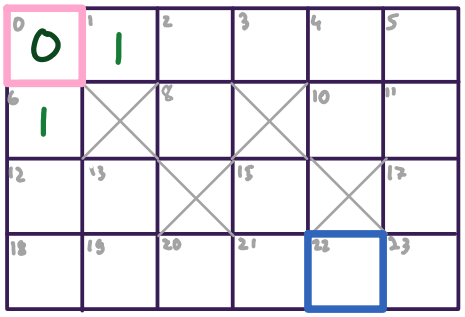
Pour commencer, on marque la case de départ d'un 0 et on push son id dans la file.
Ensuite, tant qu'on a pas marqué la case finale, on pop le premier élément de la file dans la variable *current*, marque toutes ses cases voisines qui ne sont pas un mur ou déjà marquées de sa valeur + 1 (dans l'ordre haut, droite, gauche puis bas), et on push leur id dans la file (dans le même ordre) -> Ordre arbitraire.
Si la case finale est marquée, on peut renvoyer la valeur qu'elle contient : Il s'agit de sa distance à la case de départ.

ETAPE 0



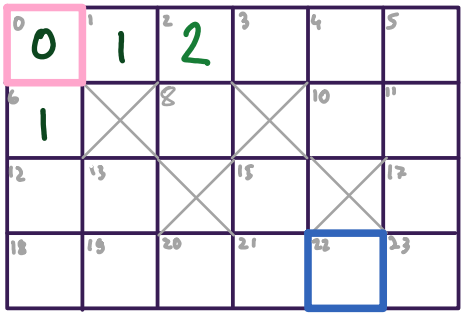
FILE: [0] CURRENT: ☐

1 :



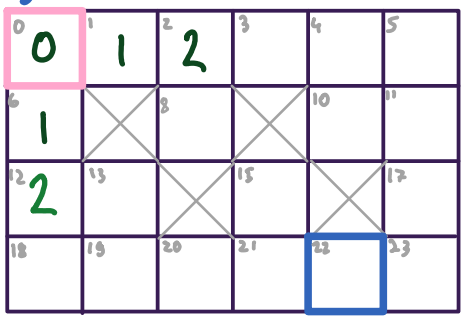
FILE: [1|6] CURRENT: 0

2 :



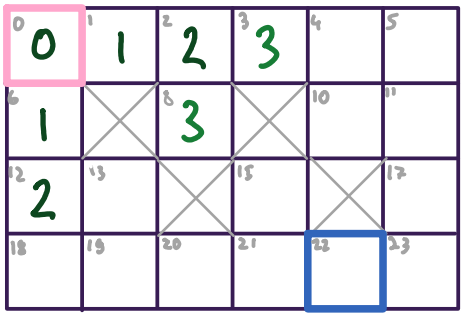
FILE: [6|2] CURRENT: 1

3 :



FILE: [21|2] CURRENT: 6

4 :



FILE: [12|3|8] CURRENT: 2

→ • • •

ETAPE N:

FILE: [17|15|22] CURRENT: 2

LA DERNIÈRE CASE EST MARQUÉE D'UN 7. C'EST LE RÉSULTAT !

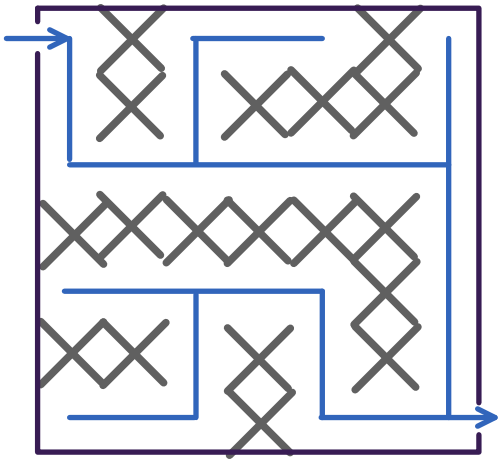
• • • →

Pseudocode

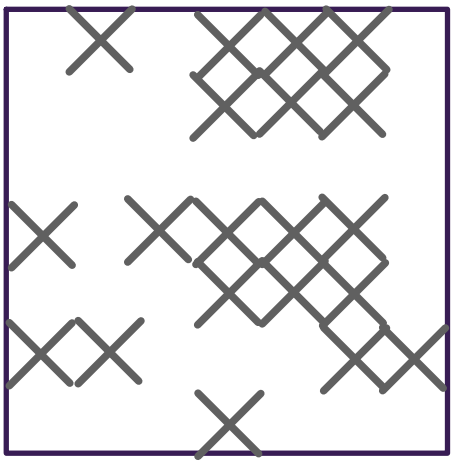
```
1 Soit A la cellule de départ, B d'arrivée
2 valeur de A ← 0
3 pile.push(A)
4 tant que B n'est pas marquée
5 |   current = pile.pop()
6 |   voisins(current) ← valeur de current +1
7 fin tant que
8 retourner la valeur de B
```

Algorithme de génération itératif avec backtracing

Cet algorithme permet de créer un labyrinthe parfait, c'est-à-dire que chaque cellule vide est reliée à toutes les autres et ce de manière unique (c'est-à-dire qu'il n'y a qu'un moyen d'aller d'un point à un autre - en particulier il n'y a qu'un chemin qui mène de l'entrée à la sortie)



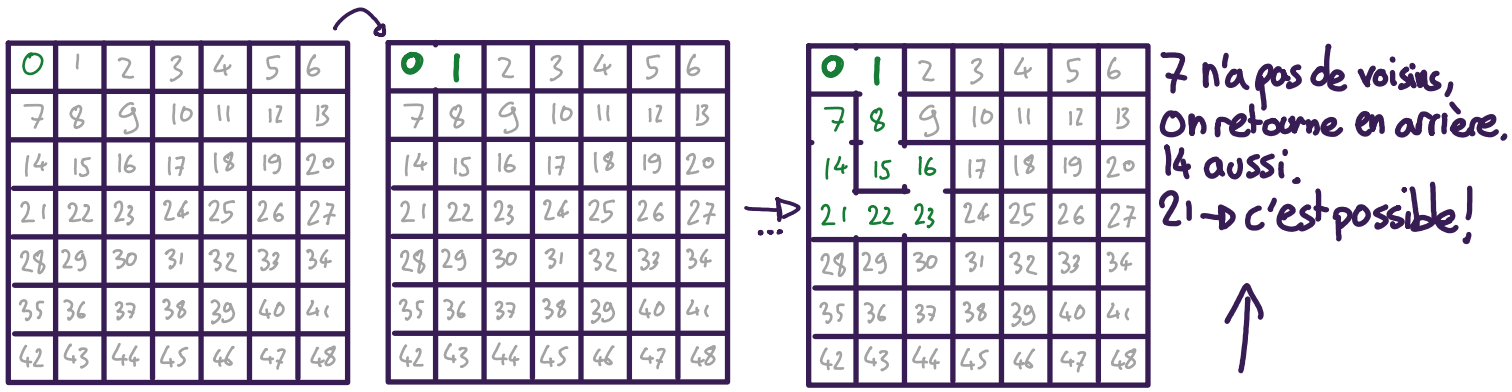
Labyrinthe parfait

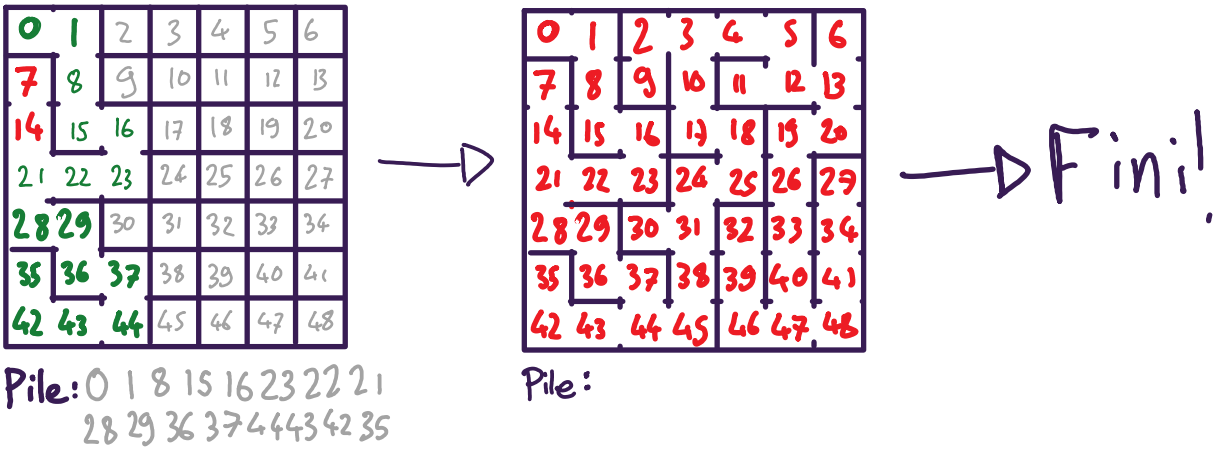


Labyrinthe imparfait

Fonctionnement de l'algorithme

L'algorithme tient en quelques étapes. En plus de la grille, il faut une pile (stack) pour pouvoir gérer le backtracking.
En commençant avec un labyrinthe avec des murs entre chaque case, on marque la première case (on la visite), et on l'ajoute à la pile.
Ensuite on récupère la cellule au sommet de la pile, et si c'est possible, on choisit une case parmi ses voisines, la visite, casse le mur entre les deux, et ajoute cette case à la pile.
On répète tant que la pile n'est pas vide.
A partir de maintenant, on représentera les murs par des barres, pour que les schémas puissent tenir sur la page :





↳ on backtrack jusqu'à 44!