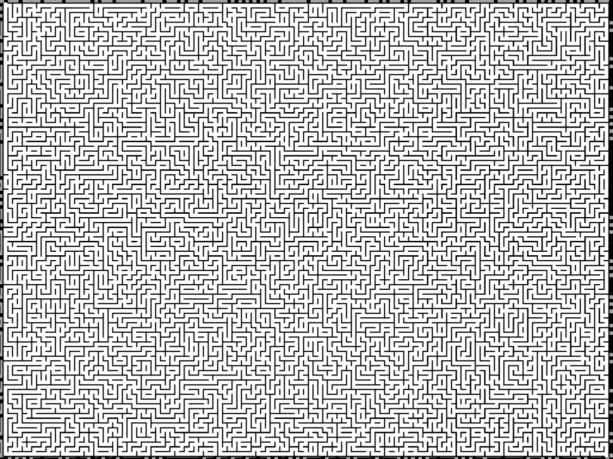
Fabrique - et jeux - de labyrinthes, auto-générés et auto-transformés

Sommaire

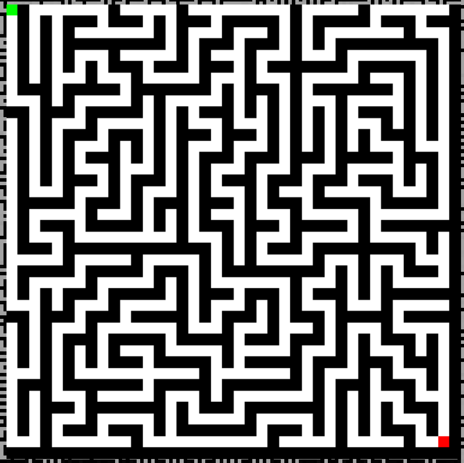
De quelques labyrinthes généraux

“Braid”



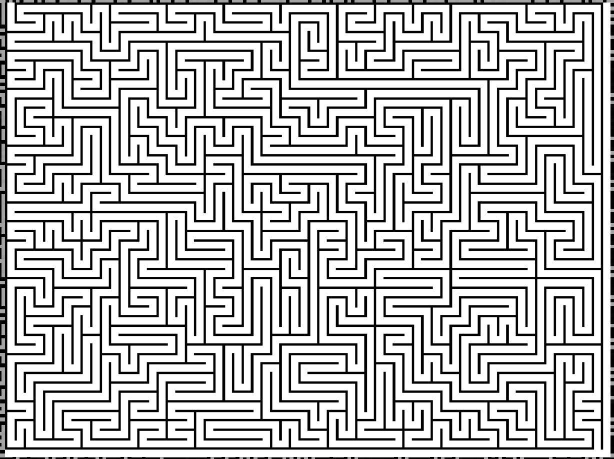
Chaque “case” possède au moins deux “cases” adjacentes.
Il n’existe donc pas de cul-de-sac.

“Parfait”



Deux points quelconques du labyrinthe sont reliés par un et un seul chemin.

“unicursal”



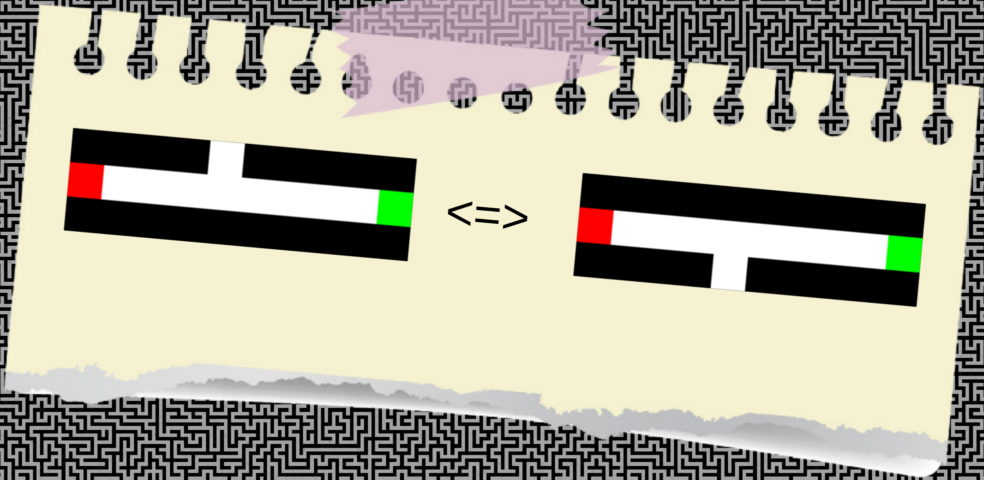
Il n’existe pas d’embranchement.
La solution recouvre donc l’ensemble des “cases” chemins.

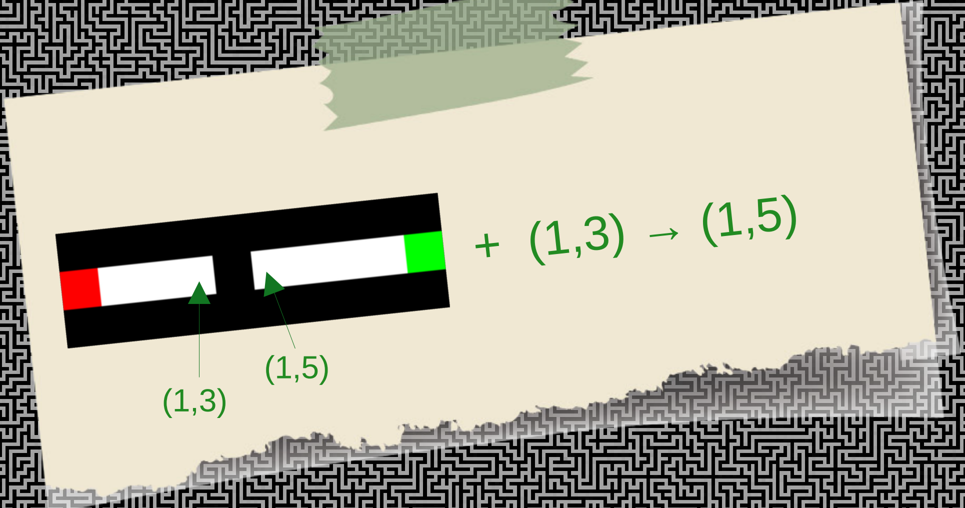
Notions complémentaires

On dit que
deux labyrinthes
sont “égaux”
si et seulement si
ils ont le même graphe.

On définit un labyrinthe “augmenté” comme étant un labyrinthe accompagné d’information relatives a des “portails”.

Ici, un portail relie
la case de coordonnées 1,3
à la case de coordonnées 1,5.



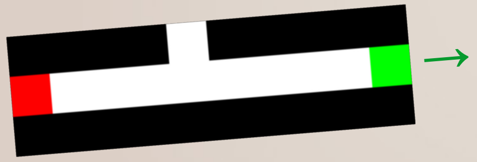


- Conversion -
Des labyrinthes aux graphes

Pour convertir un labyrinthe en graphe, on indexe les sommets sur les coordonnées des “cases” chemins.

Et les arrêtes relient les sommets dont les index correspondent à des “cases” accessibles entre elles.

La représentation du graphe utilisé ici est sa matrice d’adjacence.

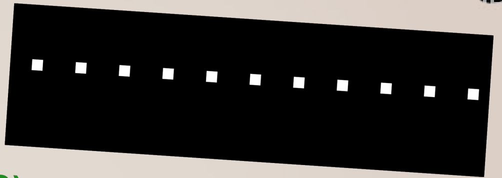


```
{(0, 4): [(1, 4)], (1, 0): [(1, 1)], (1,
1): [(1, 0), (1, 2)], (1, 2): [(1, 1),
(1, 3)], (1, 3): [(1, 2), (1, 4)], (1,
4): [(0, 4), (1, 3), (1, 5)], (1, 5):
[(1, 4), (1, 6)], (1, 6): [(1, 5), (1,
7)], (1, 7): [(1, 6), (1, 8)], (1, 8):
[(1, 7), (1, 9)], (1, 9): [(1, 8)]}
```

- Conversion - Des graphes aux labyrinthes

On peut convertir un graphe en labyrinthe augmenté en posant les sommet à la suite.
Puis, en plaçant des “portails” en fonction des arrêtes du graphe.

Dans le cas où le graphe a été obtenu à partir d’un labyrinthe, la reconversion de ce graphe produit un laby-rinthe “égal” à celui d’origine.



```
{(0, 4): [(1, 4)], (1, 0): [(1, 1)], (1, 1): [(1, 0), (1, 2)], (1, 2): [(1, 1), (1, 3)], (1, 3): [(1, 2), (1, 4)], (1, 4): [(0, 4), (1, 3), (1, 5)], (1, 5): [(1, 4), (1, 6)], (1, 6): [(1, 5), (1, 7)], (1, 7): [(1, 6), (1, 8)], (1, 8): [(1, 7), (1, 9)], (1, 9): [(1, 8)]}
```

→

```
{(5, 2): [(5, 22)], (5, 6): [(5, 10)], (5, 10): [(5, 6), (5, 14)], (5, 14): [(5, 10), (5, 18)], (5, 18): [(5, 14), (5, 22)], (5, 22): [(5, 2), (5, 18), (5, 26)], (5, 26): [(5, 22), (5, 30)], (5, 30): [(5, 26), (5, 34)], (5, 34): [(5, 30), (5, 38)], (5, 38): [(5, 34), (5, 42)], (5, 42): [(5, 38)]}
```

- Conversion -
Retour aux labyrinthes

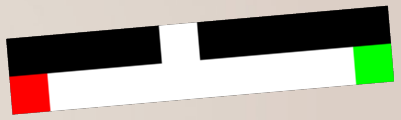
On peut re-convertir un graphe généré à partir d'un labyrinthe.

Le labyrinthe obtenu et le labyrinthe d'origine sont donc identiques.

Le labyrinthe est reconstruit en posant les “cases” chemins aux emplacements correspondant aux noms des sommets du graphe.

```
{(0, 4): [(1, 4)], (1, 0): [(1, 1)], (1,
1): [(1, 0), (1, 2)], (1, 2): [(1, 1),
(1, 3)], (1, 3): [(1, 2), (1, 4)], (1,
4): [(0, 4), (1, 3), (1, 5)], (1, 5):
[(1, 4), (1, 6)], (1, 6): [(1, 5), (1,
7)], (1, 7): [(1, 6), (1, 8)], (1, 8):
[(1, 7), (1, 9)], (1, 9): [(1, 8)]}
```

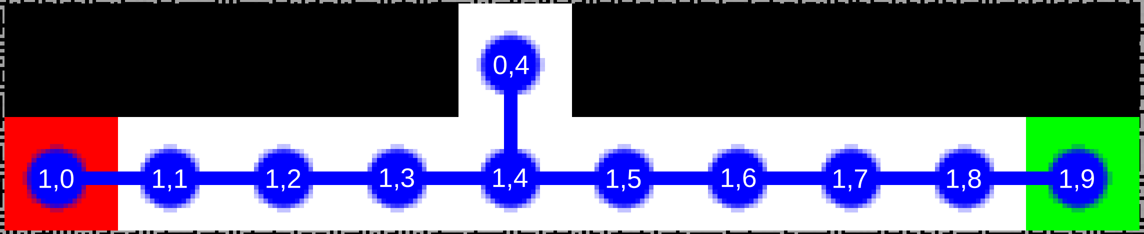
→



Bien que l'entrée et la sortie soient à placer manuellement, cela ne change rien aux labyrinthes considérés.

Bijection
Les labyrinthes et les jeux

En théorie des jeux, les jeux ont des graphes associés. Or, pour tout graphe, on peut créer *un* labyrinthe correspondant. De plus, pour tout labyrinthe, on peut créer le graphe correspondant. L'ensemble des labyrinthes est donc en double surjection - et donc en bijection - avec l'ensemble des graphes et par conséquent avec l'ensemble des jeux de la théorie des jeux.

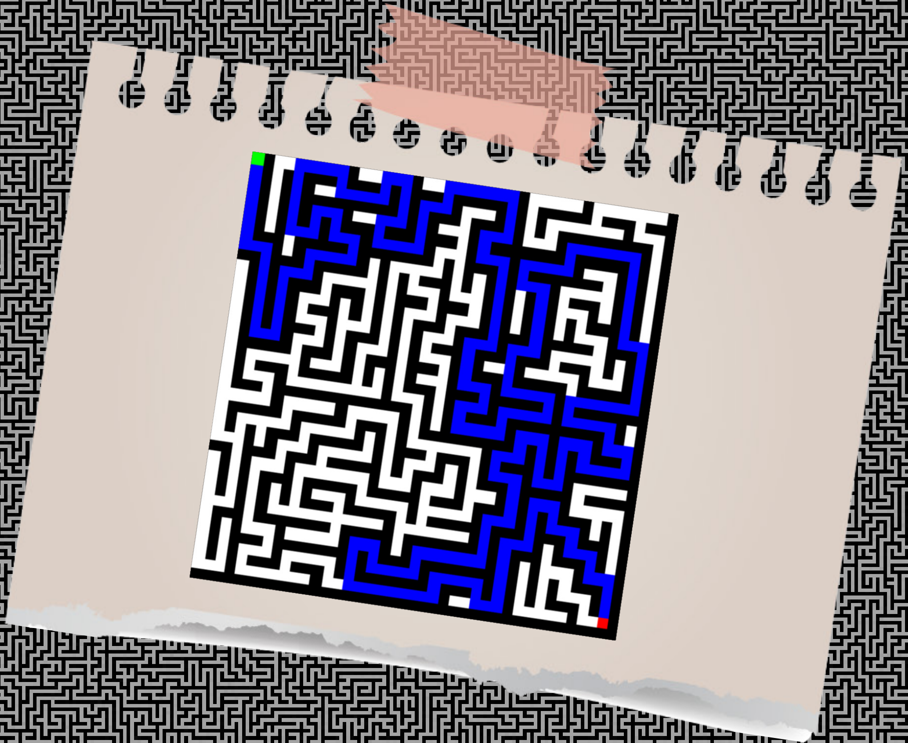


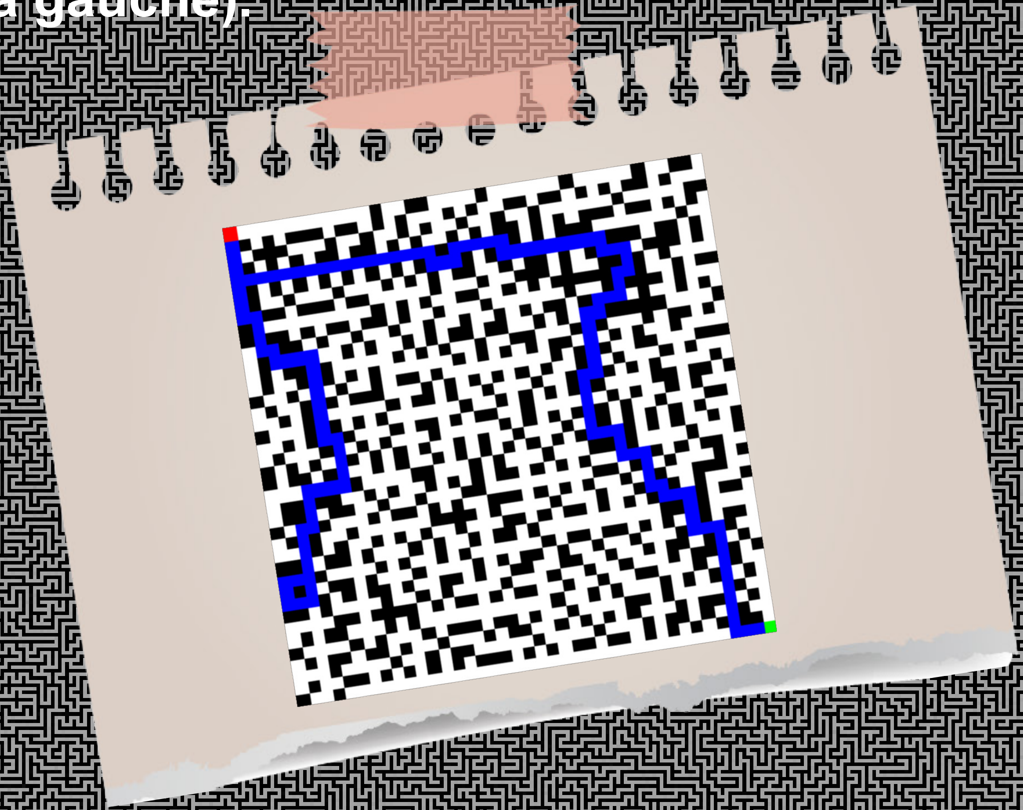
Considérant un labyrinthe parfait, on oriente son graphe de sorte qu'on ne peut pas revenir sur ses pas.
Tracer l'ensemble des solutions de ce labyrinthe équivaut alors à tracer l'attracteur d'un jeu d'accessibilité à 1 joueur. En effet, l'attracteur correspond a l'ensemble des sommets des stratégies gagnantes du joueur.

- Résolution -
Résoudre un labyrinthe parfait

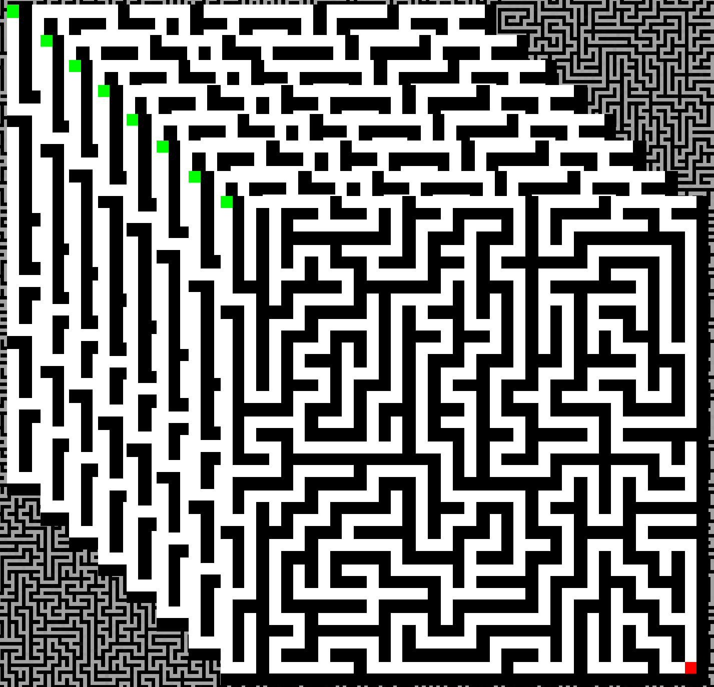
L'algorithme utilisé peut être simplement décrit en précisant que "tout ce qui est un cul-de-sac est un mur". La solution apparaît d'elle-même après itération de cette proposition.

Si on exécute le programme sur un labyrinthe cyclique, alors la solution accompagnée des cycles et de tous leurs accès est mise en évidence.
Ici, un mur a été supprimé dans un labyrinthe parfait pour créer un cycle (en bas à gauche).





- Résolution -
Résoudre un labyrinthe parfait



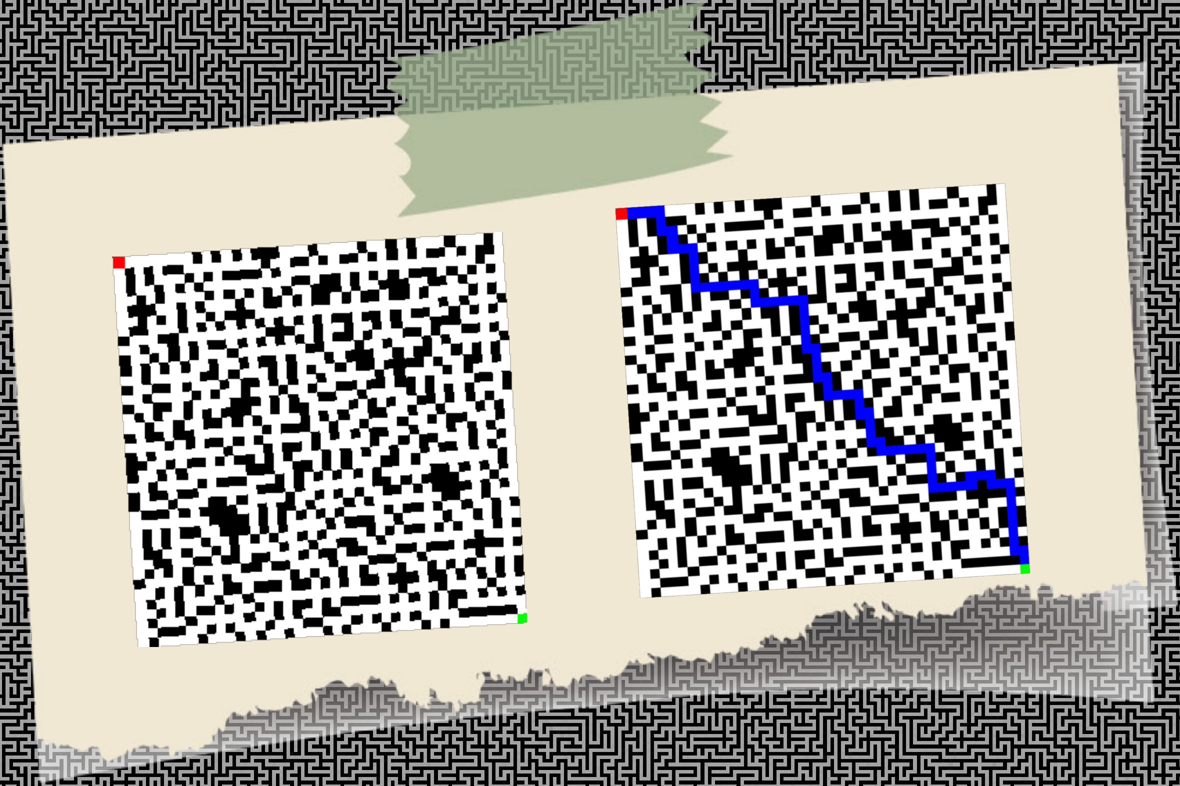
Le cœur du programme, la fonction “solv” s’exécutant sur des graphes, peut donc résoudre des labyrinthes en n dimensions, après leurs conversions en graphes.

Il est possible d’adapter le solveur pour retirer la phase lente de conversion et limiter la consommation de mémoire en ne faisant pas de graphe et en itérant donc directement sur l’image.

- Génération -
Créer un labyrinthe parfait

Directement inspiré de l'algorithme utilisé pour la résolution, on peut décrire simplement l'algorithme de génération en précisant que "tout ce qui est un mur ayant un unique chemin adjacent peut être transformé en chemin".

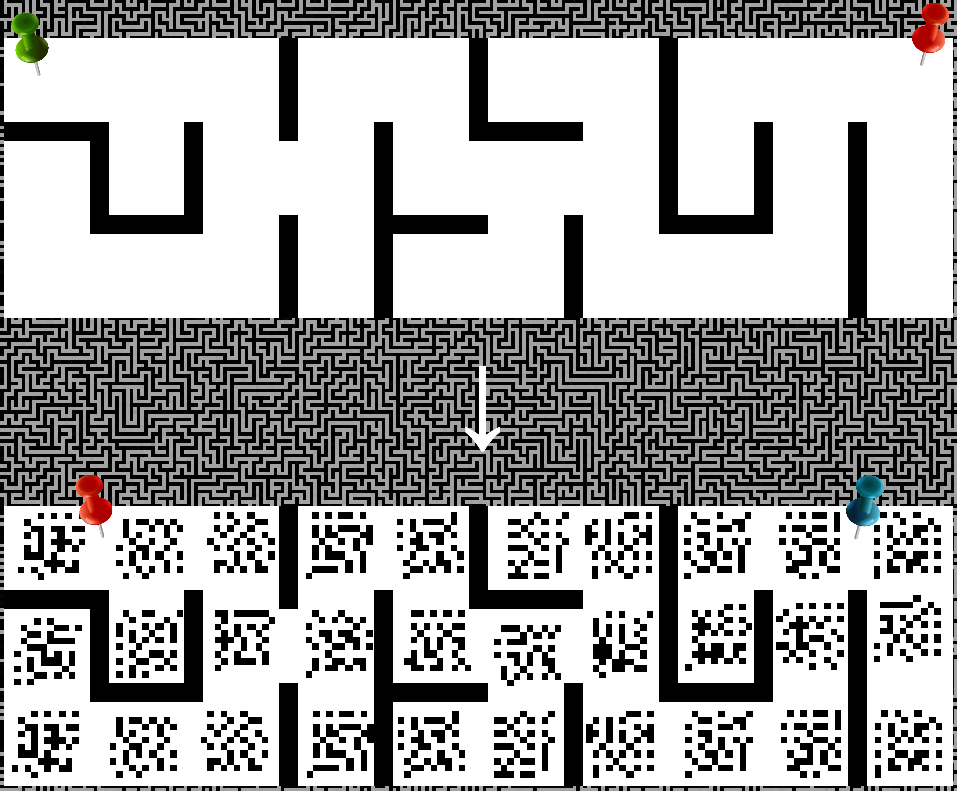
Le labyrinthe parfait créé apparaît de lui-même après itération de cette proposition.



L'un des problèmes de cet algorithme est que les solutions sont systématiquement proches de la diagonale.

- Génération -
Créer un labyrinthe parfait

Il serait possible d'accélérer la génération de très grands labyrinthes en créant une "carte", un labyrinthe, puis en construisant les labyrinthes correspondant aux pixels de la carte mais connectés entre eux.

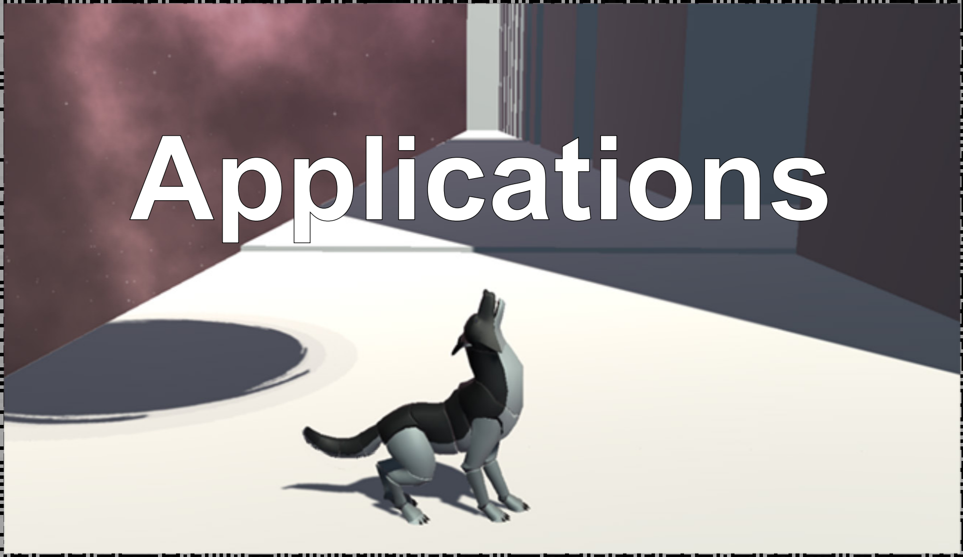


L'algorithme peut fonctionner en n dimensions. Cependant, pour adapter le programme en n dimensions, il faudrait adapter les indexations dans la liste "nxt" et les fonctions "adj" et "possible".

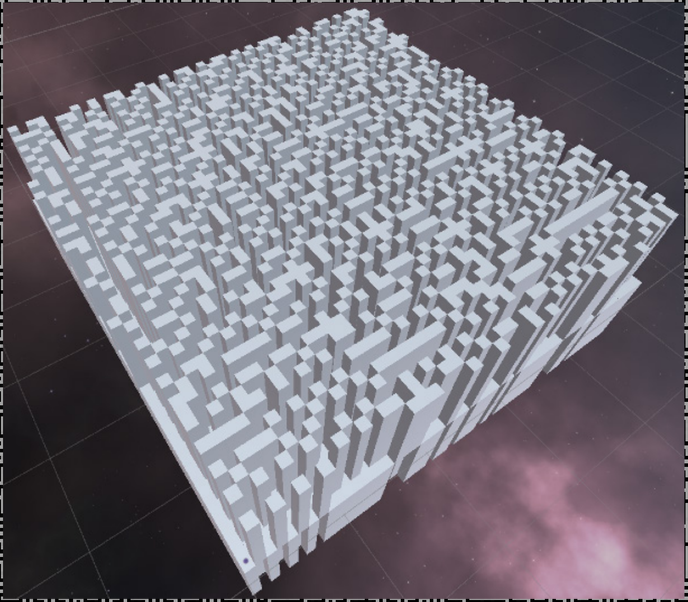
Il est possible de résoudre simultanément un grand nombre de labyrinthes en utilisant un module de multi-threading. De sorte qu'on pourrait théoriquement, diviser le temps de résolution par le nombre de cœur disponibles du processeur.

Applications

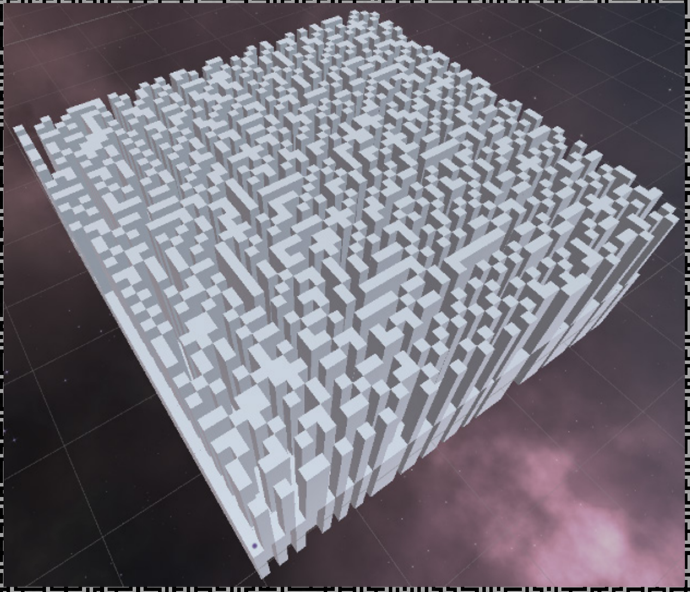
Il devient donc possible de créer un jeu vidéo de labyrinthe sur un moteur comme Unity, après traduction du générateur de labyrinthe en c#.



Ici, un jeu ou le joueur incarne un loup piégé dans un labyrinthe sans sortie, généré procéduralement et qui s'actualise toute les 10 secondes.

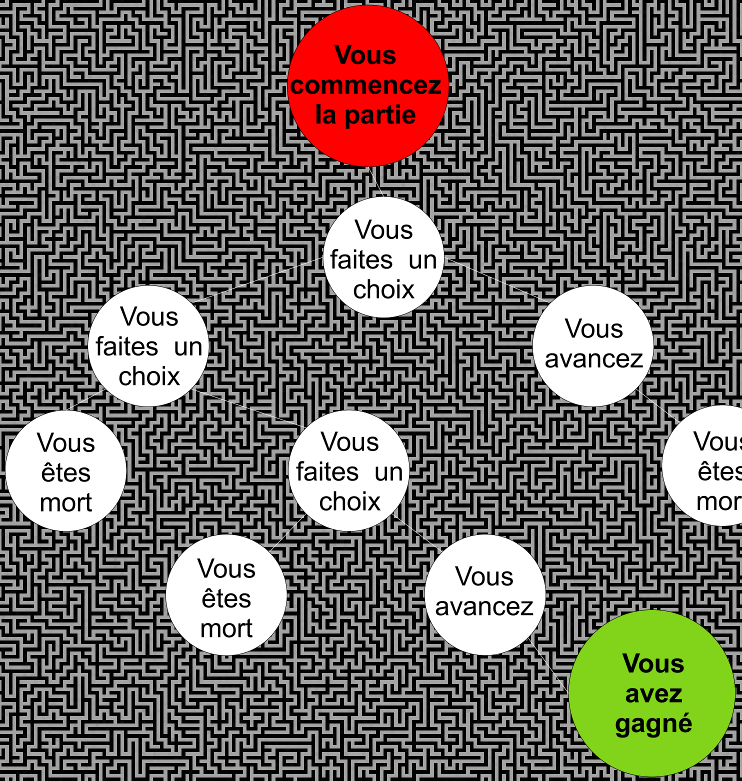


On peut comparer le labyrinthe à deux moments différents du jeu : on constate qu'il est bien différent, régénéré entre-temps.



Un livre-jeu est visible comme un arbre, or un labyrinthe parfait est un graphe acyclique connexe (donc un arbre) d’arité 4 au plus (en fait 3).
Ainsi, en générant un labyrinthe parfait, on obtient la structure d’un livre-jeu. Dés lors, en habillant automatiquement cette structure avec du texte créé par une intelligence artificielle telle que Mistral AI, il est possible de créer automatiquement des livres-jeux.

UN LIVRE DONT VOUS ÊTES LE HÉROS
Ian Livingstone
La Forêt de la Malédiction
UN LIVRE DONT VOUS ÊTES LE HÉROS
Steve Jackson et Ian Livingstone présentent
Les Trafiquants de Kelter
UN LIVRE DONT VOUS ÊTES LE HÉROS
Ian Livingstone
Le Labyrinthe de la Mort
UN LIVRE DONT VOUS ÊTES LE HÉROS
Steve Jackson
La Galaxie Tragique
CHOISIS TON APOCALYPSE
AVEC KIM JONG-UN & SES AMIS
Robert Laffont



Enfin, toute situation requérant une prise de décision peut se modéliser par un graphe de “possibilités”, ayant pour sommets les “états” possibles et pour arrêtes les “itinéraires” possibles entre ces états.
De ce fait, ces situations de choix sont modélisables par des labyrinthes dont l’entrée est l’état de départ, la sortie, l’état souhaité et dont le graphe est celui des possibilités de ces situations.