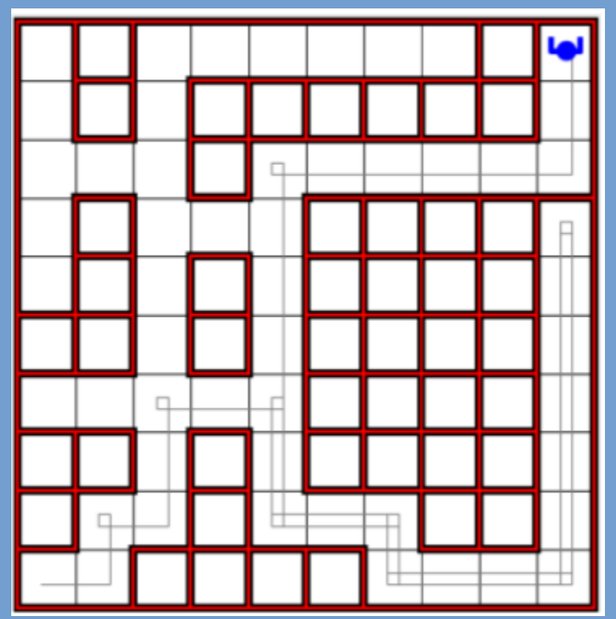
Nos stratégies pour sortir d'un labyrinthe !

La stratégie de la main droite

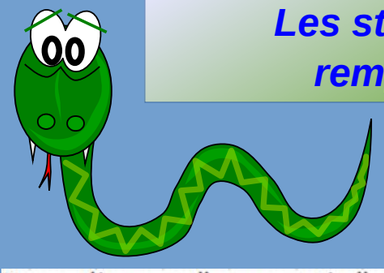
Méthode : Le robot longe un mur, dans ce cas précis le mur de droite, et avance jusqu'à trouver la sortie. Le robot est dépendant de ce mur et ne le lâche pas.

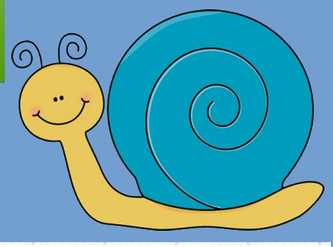


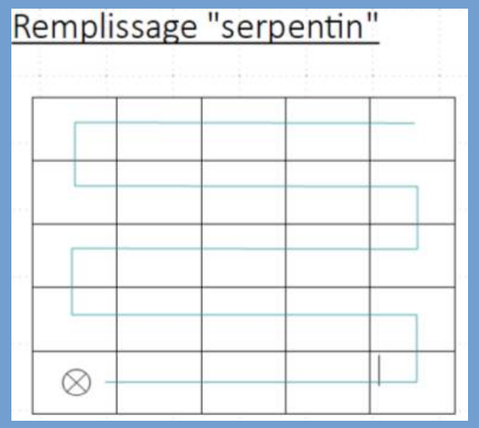
Labyrinthe et programme réalisés avec Rurple

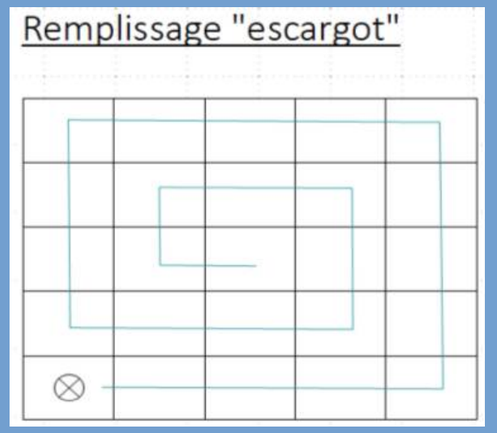
Inconvénient : Ne marche pas si la sortie ou l'entrée se situe sur un îlot !

Les stratégies de remplissage





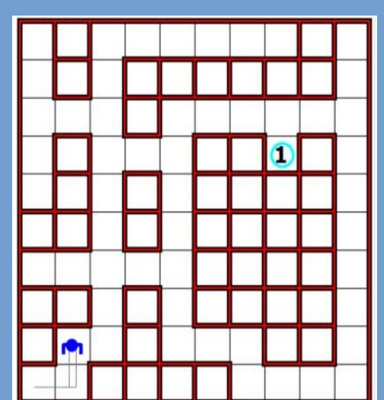




Méthode : On cherche à parcourir toutes les cases du labyrinthe. Ci-dessus, la stratégie est représentée dans un labyrinthe sans mur pour visualiser le motif.
Mais dans la réalité, à chaque fois que le robot rencontre un mur en face de lui, le robot change de direction.

Inconvénient :
Cette méthode ne fonctionne pas dans certaines configurations de labyrinthe !

Par simulation sur des labyrinthes 10x10 générés aléatoirement, on a constaté qu'elle échouait dans environ 2 % des cas.



Le robot essaye ici le motif escargot. Or, dans ce labyrinthe-ci, cette méthode ne fonctionne pas, du fait de la configuration de ce labyrinthe, le robot tourne en rond.

Lycée Marguerite de Navarre
BOURGES

Travaux encadrés par un enseignant-chercheur de :
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

La stratégie de l'aléatoire

Méthode : On considère dans ce programme qu'à chaque intersection, le robot choisit aléatoirement une direction en fonction des chemins qui lui sont disponibles. Le robot peut donc tourner à gauche, à droite ou bien simplement aller tout droit.
En revanche, en cas d'impasse, le robot est capable de faire demi-tour afin d'emprunter un autre chemin.

Avantage :
Si on ne prend pas en compte le temps, le robot a la certitude de trouver la sortie !

Inconvénient :
Dans le cas contraire, ce programme n'est pas le plus optimisé puisque comme le robot n'a pas de mémoire, il peut souvent se tromper de chemin, tourner en rond ou bien passer un grand nombre de
fois par les mêmes chemins.

SUITE : Nous avons donc essayé de déterminer avec cette stratégie **le temps minimum** et **le temps moyen** nécessaires (compter en nombre de déplacements) pour trouver la sortie d'un labyrinthe…

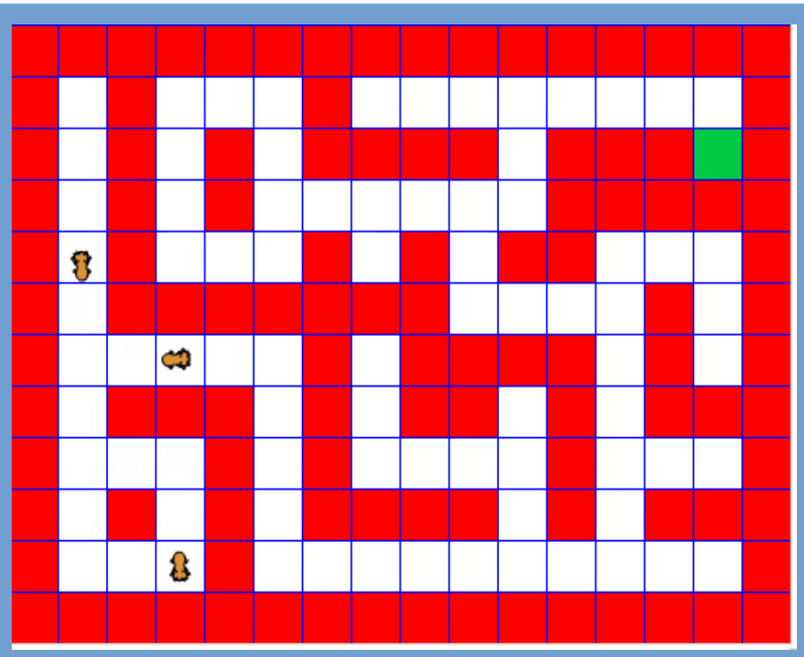
→ **voir « Comment on a utilisé des graphes et des matrices ! »**

La stratégie du parallélisme

Méthode : A chaque intersection, le robot se duplique un certain nombre de fois selon le nombre de
chemins possibles afin d'emprunter toutes les cases blanches d'un labyrinthe. Cela permet de faire
plusieurs tâches à la fois et donc de réduire considérablement le temps.

Inconvénient :
Cette méthode est très coûteuse en ressources et nécessiterait plusieurs ordinateurs.

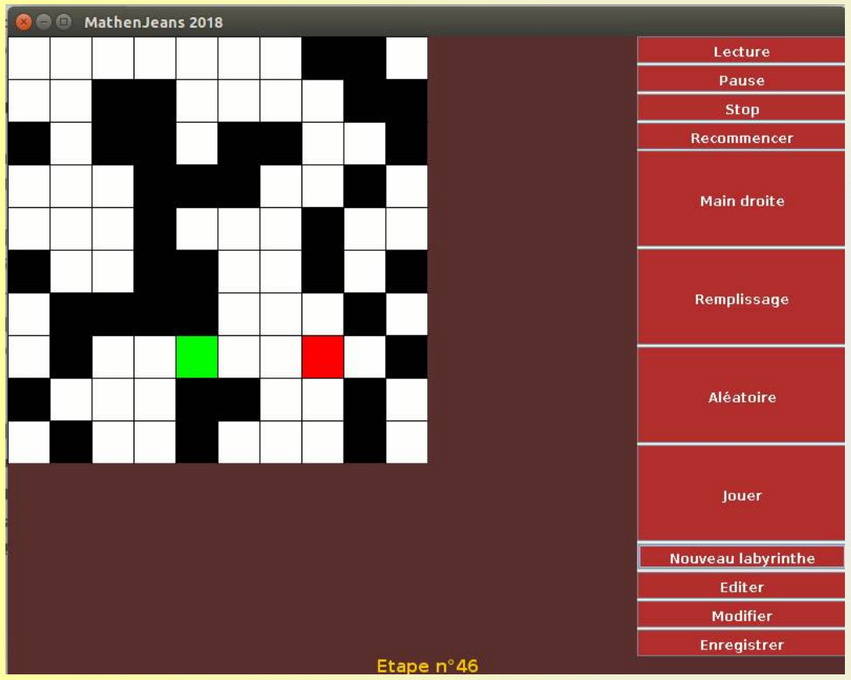
Parallélisme réalisé sous Scratch 2



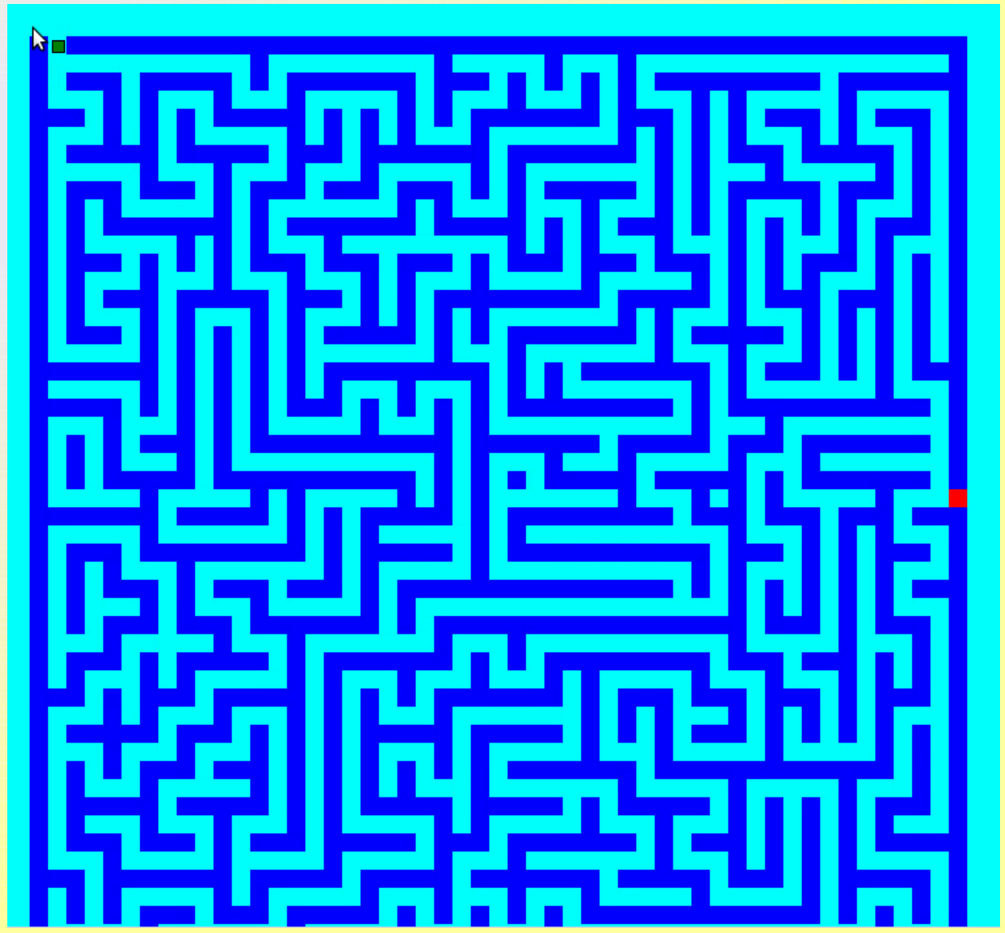
SUITE : Nous avons donc essayé de détermine le nombres de robots nécessaires pour explorer tout le labyrinthe.

Nos réalisations en langage Python et Java :

Logiciel réalisé en Java et permettant de comparer et d'essayer nos stratégies sur de multiples labyrinthes générés aléatoirement :



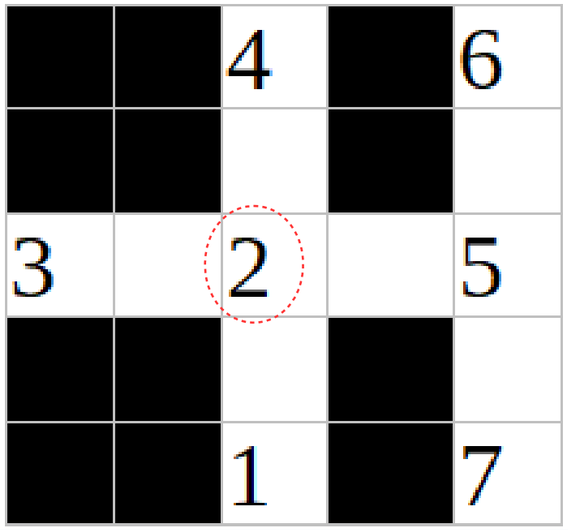
Logiciel réalisé en Python où l'objectif est de sortir d'un labyrinthe avec plusieurs niveaux de difficultés



Essayez-les !!!

Comment on a utilisé des graphes et des matrices ?

Labyrinthe représenté par des nœuds et des arêtes

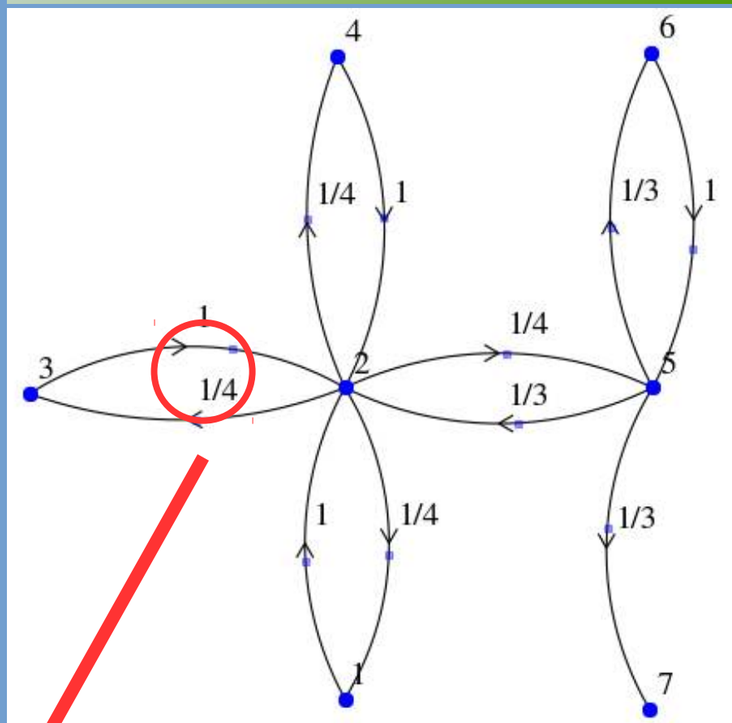


On transforme **chaque intersection en sommet et chaque chemin par un arc** sur le graphe.

<u>Exemple</u> : 2 est un sommet et le chemin de 2 à 3 est un arc sur le graphe.

1/4 signifie qu'on a 1 chance sur 4 de passer de 2 à 3. On appelle cela ***une probabilité.***

Labyrinthe <u>mis en graphe</u> avec les probabilités de passer dans chaque chemin



La matrice de transition du graphe est :

$$M=\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

On **reporte toutes les probabilités dans un grand tableau qu'on appelle matrice** de transition.

<u>Exemple :</u> Le nombre 1/4 à la ligne 2 colonne 3 indique la probabilité de passer du sommet 2 au sommet 3.

Avec du calcul matriciel!

Pour cela, on utilise une matrice à une ligne pour indiquer **la place du robot au départ :**

$X_0 = (1\ 0\ 0\ 0\ 0\ 0\ 0)$

Le 1 indique que le robot est au sommet 1.

Les zéros, là où il n'est pas !

Ensuite, **on cherche la position du robot après n déplacements** (c'est le nombre de choix de direction réalisé à chaque intersection rencontrée).

Enfin, **on effectue des calculs entre les matrices : M et X_0.**

Cela revient à faire des additions et des multiplications entre les nombres présents dans ces matrices !!!

Exemple : On obtient ainsi au bout de 3 déplacements,

$X_3 = (0 \quad 0{,}833 \quad 0 \quad 0 \quad 0 \quad 0{,}083 \quad \mathbf{0{,}083})$.

Donc le robot a 0,083 chances d'être sorti du labyrinthe en utilisant la méthode aléatoire au bout de trois déplacements.

TABLEAU DES RÉSULTATS :

Nombre de pas	Probabilités d'atteindre la sortie en n pas exactement	Probabilités d'atteindre la sortie en n pas maximum
1	0	0
2	0	0
3	0,083	0,083
4	0	0,083
5	0,097	0,181
10	0	0,357
15	0,058	0,556
20	0	0,653
29	0,025	0,813
100	0	0,998

Pour les Matheux !

Nous obtenons des états probabilistes que l'on entre dans des matrices.

On note S_i le sommet pour tout entier i tel que $1 \leq i \leq 7$.

Soit $P_{i,n}$ la probabilité de se trouver au sommet i au bout de n déplacements.

On définit $X_n = (P_{1,n} \quad P_{2,n} \quad \dots \quad P_{7,n})$ l'état probabiliste au bout de n déplacements.

Au départ le robot est en S_1 donc $X_0 = (1\ 0\ 0\ 0\ 0\ 0\ 0)$.

Pour tout entier naturel n on a :

$$X_{n+1} = X_n \times M$$

On en déduit que pour tout n naturel on a :

$$X_n = X_0 \times M^n$$

Finalement, en calculant l'espérance, on obtient un **temps moyen égal à 19 pas pour sortir du labyrinthe.**