Boyer Noa
Perret Kyllian

Programme de génération de labyrinthe via un arbre binaire

Fichiers :

-LabyrintheGénèration.py
-classABR.py
-File.py

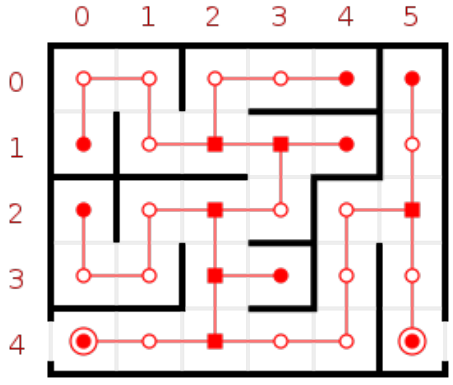
Fonction principal : laby(n,m) dans le fichier LabyrintheGénèration.py

Source :

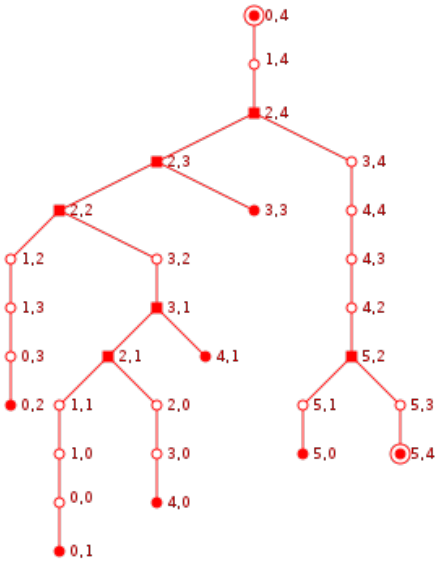
- https://fr.wikipedia.org/wiki/Mod%C3%A9lisation_math%C3%A9matique_d%27un_labyrinthe

Méthode :

Pour réaliser le labyrinthe, nous sommes partie d'une classe ABR. Celle ci est un arbre binaire composé: d'une racine , d'un fils Gauche et d'un fils Droit. Les différents fils représenteront les différents chemins du labyrinthe.
Nous entrerons dans les racines une coordonnée :

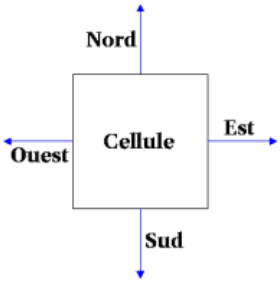


Ainsi dans l'exemple ci contre , la racine de l'entrée sera [0,4] et son fils [1,2].



Ce qui donnerait un arbre ressemblant à ça.

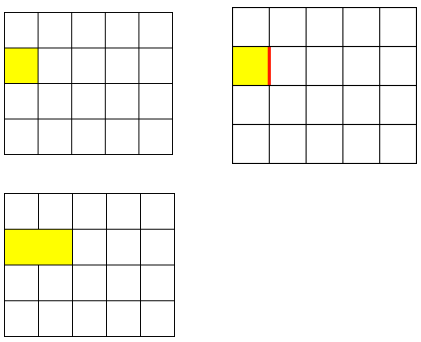
Donc cela implique que chaque case est maximum 4 voisins.



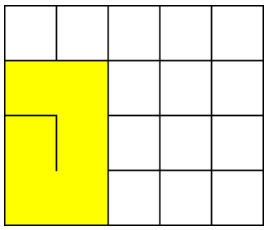
Sur un repère orthonormé où x est l'axe des abscisse et y l'axe des ordonnées.
Les 4 cas sont : - vers le haut/Nord , [x,y+1]
- vers la gauche/ouest , [x-1,y]
- vers le bas/sud , [x,y-1]
- vers la droite/est , [x+1,y]

Pour créer le labyrinthe, nous allons utiliser la méthode d'exploration exhaustive. Celle ci consiste en :

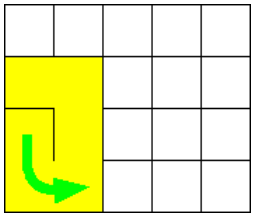
-Prendre un case , regarder ses plus proches voisins.
-Choisir aléatoirement parmi eu une direction a prendre. Et on recommence .



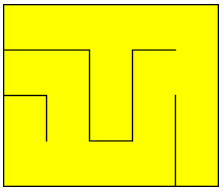
-Au bout d’un moment , la case actuelle n’a pas de voisin libre.



-Alors on remonte à la dernière case ayant minimum un voisin libre et on l’empreinte.

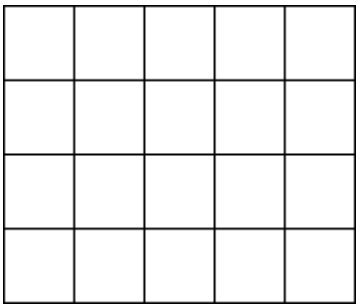


-Au bout d’un moment le labyrinthe n’aura plus aucune case avec un voisin libre , alors le labyrinthe est fini.

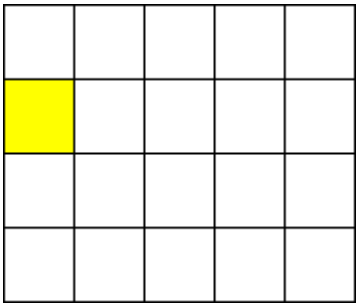


Le traçage du labyrinthe :

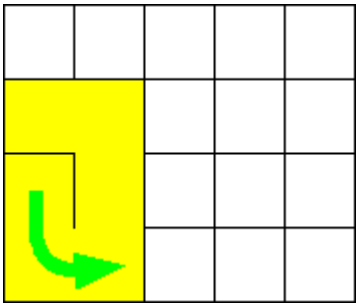
-Dans un premier temps on trace une grille , celle ci faisant la taille du labyrinthe.



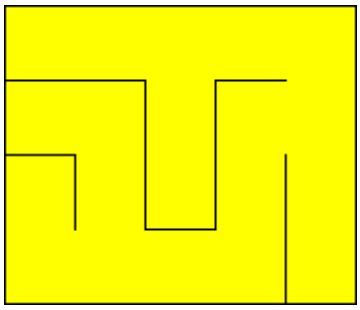
-On prend la première case , on regarde ses voisins puis on en choisi un , et on casse le mur. On recommence avec le chemin ouvert.



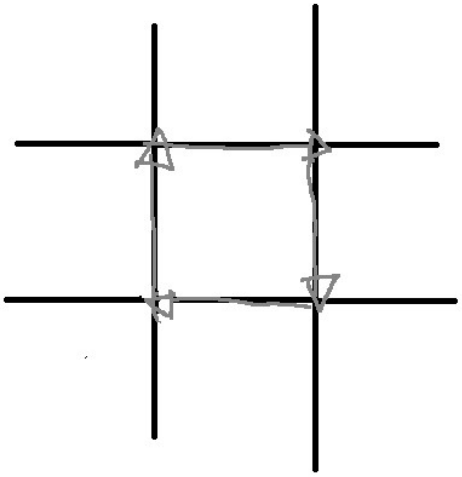
-Lorsqu’on arrive dans une impasse , on remonte jusqu’à la dernière case ayant un voisin.



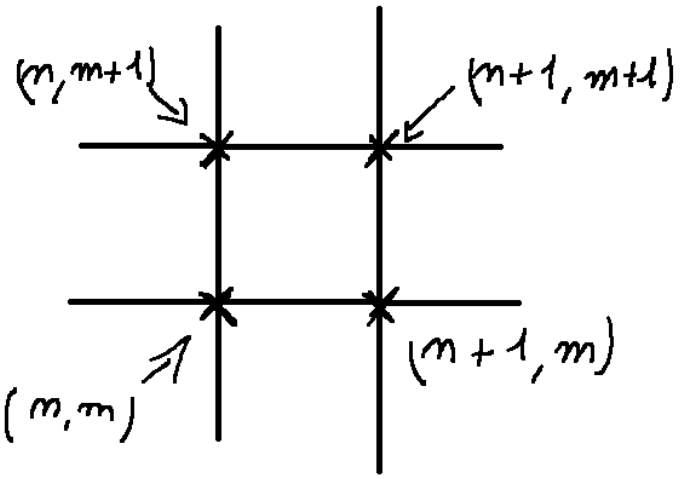
-Le labyrinthe a fini d’être tracé quand tout les chemins on été parcouru.



Globalement la création de labyrinthe et son affichage se font de la même manière. La principale différence est l’ouverture de chemin : Techniquement nous ne « cassons » pas un mur , nous le repassons en blanc. Selon la direction du chemin , on place le point de départ du trait a différent endroit :



La flèche grise représente la direction



Et ainsi ,on peut alors indiquer la position a turtle où aller en fonction du voisin.

La solution :

Pour trouver la solution, il suffit de faire un parcours en profondeur et une fois la racine égale à la fin du labyrinthe, on remonte et on ajoute chaque racine à une liste.

Bilan :

C’est un projet plutôt compliqué. La partie la plus compliquer est, sans nul doute, la génération de labyrinthe. L’un des problèmes les plus pénibles était la remontée de l’arbre, cela provoquait un dédoublement de celui ci. Le second problème majeure était turtle, nous ne connaissions pas ce module ou plutôt nous n’avons pas énormément bosser dessus. Mais une fois turtle bien assimiler , le reste ne pausa pas de problème.

La répartition du travail :

Noa → l’aspect graphique

Kyllian → algorithme et débuggage