Dans un labyrinthe et ... autour

1. Quelques exemples

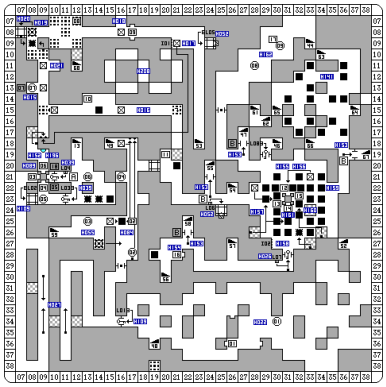


FIGURE 1 – Un niveau de Dungeon Master

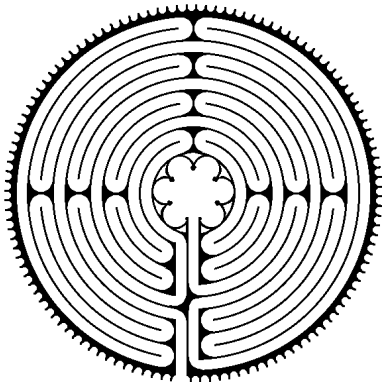


FIGURE 2 – Au sol de la cathédrale de Chartres

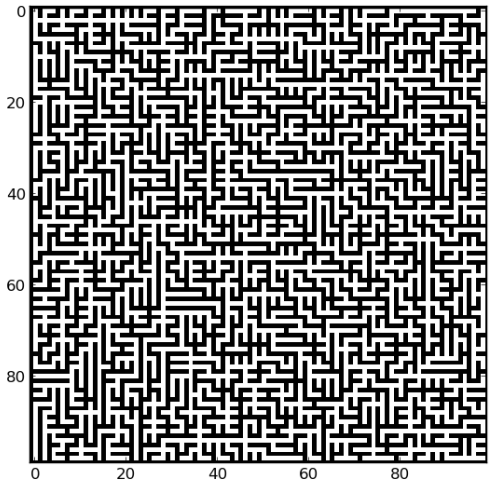


FIGURE 3 – Un labyrinthe de taille 100×100 généré par ordinateur

2. Quelques hypothèses

Un labyrinthe est constitué d'un ensemble de cases. Ces cases seront de deux types :
- un morceau de couloir, c'est à dire une case sur laquelle on peut marcher (en blanc sur les dessins précédents),
- ou un mur, c'est à dire une case sur laquelle on ne peut pas marcher (en noir).

Dans un labyrinthe on veut pouvoir aller d'un point A (l'entrée) à un point B (la sortie) et ces points ne sont pas nécessairement sur la périphérie du labyrinthe (on peut penser au niveau successifs d'un donjon relié par des escaliers). Dans certains labyrinthes, aller de A à B sera
- impossible (le constructeur du labyrinthe est un sadique),
- possible par un seul chemin (le constructeur est sympa mais pas trop),
- possible par plusieurs chemins (le constructeur est très sympa)

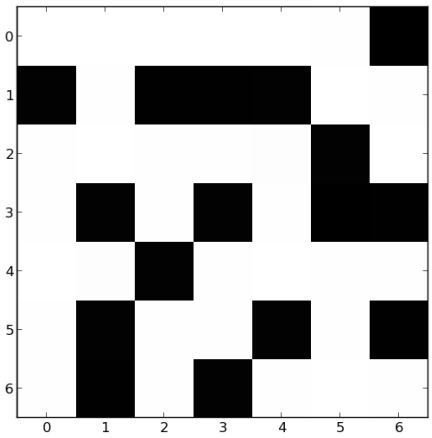
Ces distinctions conduisent à envisager une classification des labyrinthes. Dans la suite on tendra à privilégier le labyrinthes du deuxième type qui seront dit parfaits [1]. [2]

1. on ira se référer à `http://fr.wikipedia.org/wiki/Mod%C3%A9lisation_math%C3%A9matique_d%27un_labyrinthe`
2. Le labyrinthe de la figure 3 est parfait

3. Représentation du problème

On choisit de représenter les cases d'un labyrinthe par les cases d'un tableau à double entrée dans lequel on codera par 0 la présence d'un mur et par 1 l'absence de mur. Un tel tableau s'appelle une *matrice*.

Exemple :



j \ i	0	1	2	3	4	5	6
0	1	1	1	1	1	1	0
1	0	1	0	0	0	1	1
2	1	1	1	1	1	0	1
3	1	0	1	0	1	0	0
4	1	1	0	1	1	1	1
5	1	0	1	1	0	1	0
6	1	0	1	0	1	1	1

4. Objectifs

- Étudier/Établir un algorithme (en particulier l'algorithme de Prim) de génération d'un labyrinthe parfait. Visualiser les étapes de construction à l'écran.
- Étudier / Établir un algorithme pour sortir du labyrinthe.
  Visualiser les étapes de résolution à l'écran.
- D'autres applications de l'algorithme de Prim. (Optionnel)

5. Modalités

Le langage retenu pour mener le projet est *python*. Nous utiliserons la librairie graphique *matplotlib* pour les visualisations et *numpy* pour les manipulations de matrices.

- Séance 1 : se familiariser avec l'environnement de travail.
  Sur un exemple facile (génération aléatoire). Jeux avec les paramètres. Sauvegarde sous forme d'une vidéo. Sauvegarde sous forme d'un fichier.
  Affichage d'un labyrinthe à partir d'un fichier Distinguer les différents types de labyrinthes.
- Séance 2 : programmer quelques fonctions.
  ≪ Qualité ≫ d'un labyrinthe :
  – longueur de la plus grand ligne droite,
  – nombre de cul-de-sac,
  – nombre virages ...
  Des fichers de labyrinthe seront fournis.
- Séance 3 : l'algorithme de Prim et mise en œuvre (première version).
- Séance 4 : l'algorithme de Prim (version améliorée).
  parcours d'un labyrinthe.
- Séance 5 : ajustement / développement en fonction du temps.
- Séance 6 : présentation des élèves.