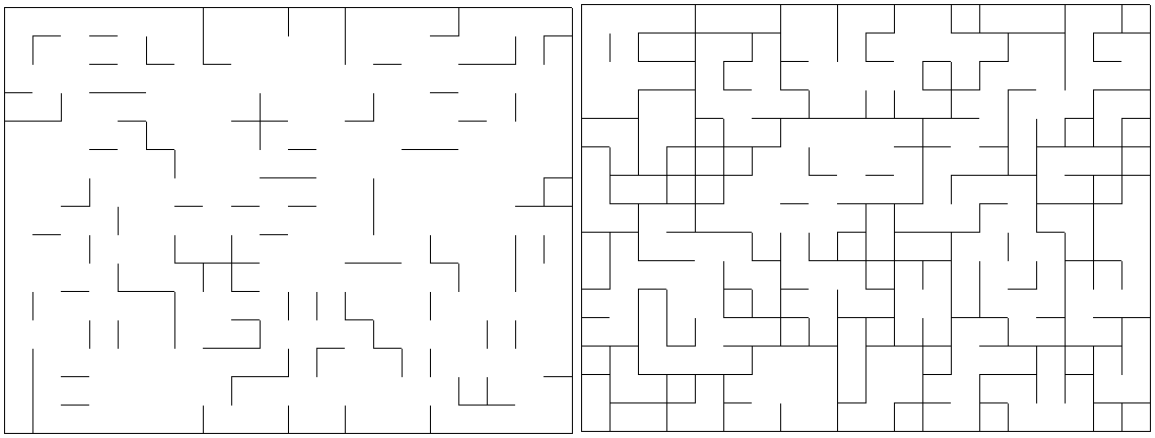
Labyrinthes parfaits

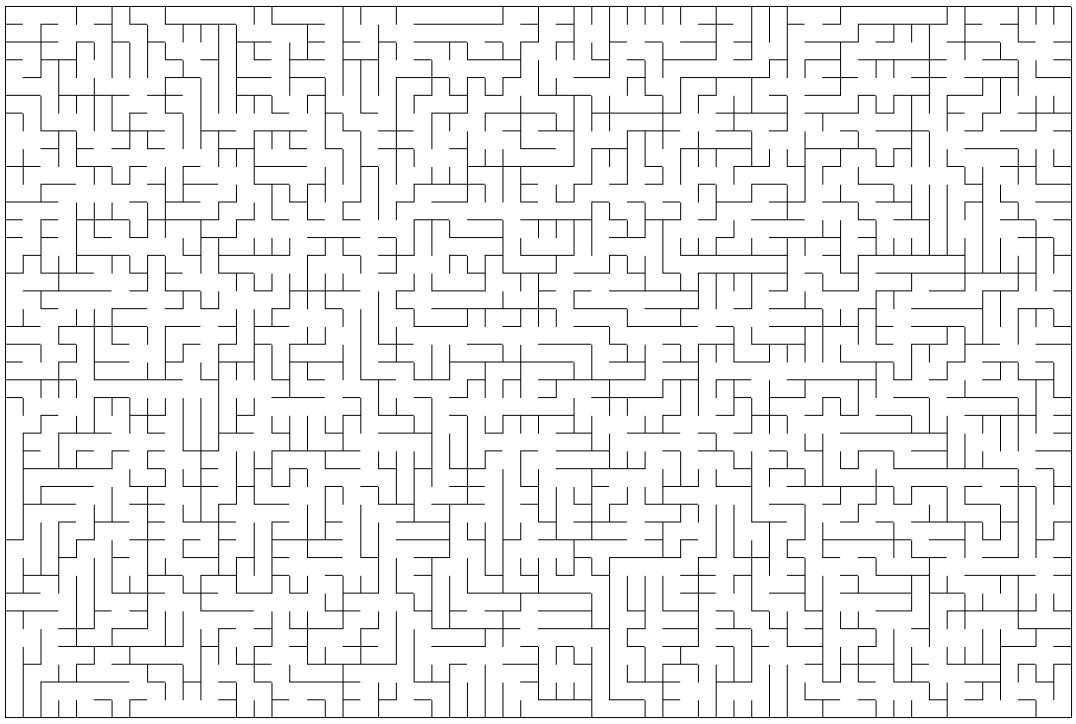
I. La voie de la perfection

Les exemples ci-dessous illustrent les deux tendances que l'on peut observer en jouant sur la densité de murs dans nos labyrinthes :

- Avec une faible densité de murs, le labyrinthe est trivialement simple
- Avec une forte densité de murs, le labyrinthe est impossible à résoudre



L'objectif de ce TD est de construire des labyrinthes parfaits, c'est-à-dire tel que pour tout couple de cases, il existe un unique chemin entre ces deux cases.



II. Quelques petites fonctions sur les listes

Pour créer ce type de labyrinthes, nous aurons besoin de quelques fonctions auxiliaires, dont l'implémentation vous permettra de manipuler à nouveau les listes.

Remarque : Pour chacune des fonctions demandées, il est recommandé de tester son bon fonctionnement sur quelques exemples avant de passer à la question suivante.

a) Insérer si nouveau

Question 1. Imaginer une fonction `inserer_si_nouveau` qui prend en argument une liste `l1` et un élément `x`, et qui ajoute l'élément `x` à la liste `l1` s'il n'est pas déjà présent dans cette liste.

Question 2. Utiliser cette fonction pour implémenter une fonction `inserer_si_nouveaux`, qui prend en argument deux listes `l1` et `l2`, et qui ajoute dans la liste `l1` les éléments de la liste `l2` qui ne sont pas déjà présents dans `l1`.

Remarque : La liste `l2` ne doit pas être modifiée.

b) Choisir un élément au hasard dans une liste

Question 3. Ecrire une fonction `element_au_hasard`, qui prend en argument une liste `l`, et qui renvoie un élément choisi au hasard dans cette liste.

Remarque : L'élément choisi doit être retiré de la liste avant d'être renvoyé par la fonction.

III. Fonctions auxiliaires sur les labyrinthes

Vous aurez également besoin d'un certain nombre de fonctions auxiliaires opérant sur nos labyrinthes pour créer des labyrinthes parfaits.

a) Choisir une case au hasard

Question 4. Imaginer une fonction `case_au_hasard` qui prend en argument un labyrinthe et qui renvoie un couple de coordonnées correspondant à une case choisie au hasard dans le labyrinthe.

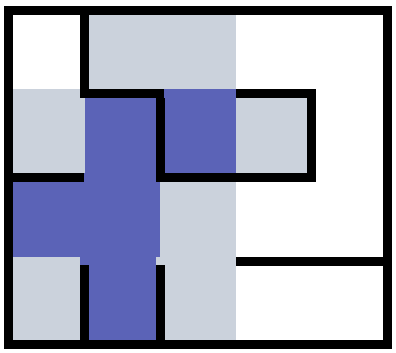
b) Ajouter et supprimer des murs

Question 5. Imaginer une fonction `ajouter_des_murs_partout` qui prend en argument un labyrinthe et qui ajoute des murs partout dans le labyrinthe.

Question 6. Ecrire une fonction `retirer_un_mur`, qui prend en argument un labyrinthe, ainsi que les coordonnées de deux cases voisines de ce labyrinthe, et qui supprime le mur situé entre ces deux cases.

c) Trouver les "voisins potentiels"

Un voisin potentiel d'une case (i, j) est une case située à proximité directe de la case (i, j). Contrairement à la notion de "voisins" évoquée dans le TD précédent, la notion de "voisins potentiels" ne tient pas compte des murs entre les cases.



Il sera utile de pouvoir déterminer l'ensemble des voisins potentiels d'un ensemble de cases données.

Par exemple, sur l'illustration ci-contre, on a représenté en bleu clair l'ensemble des voisins potentiels des cases représentées en bleu foncé.

Question 7. Ecrire une fonction `voisins_potentiels`, qui prend en argument un labyrinthe, ainsi qu'un couple de coordonnées correspondant à une case de ce labyrinthe, et qui renvoie la liste des "voisins potentiels" de cette case.

IV. Labyrinthes parfaits

Pour obtenir un labyrinthe parfait, on suit la méthode suivante :

1) On crée un labyrinthe vide
2) On ajoute des murs partout
3) On crée une liste de "cases accessibles", initialement vide
4) On choisit une case au hasard, qu'on marque comme "visitée"
5) On ajoute les "voisins potentiels" de cette case à la liste des "cases accessibles"
6) Tant qu'il reste des "cases accessibles"
   - On choisit une case au hasard parmi ces "cases accessibles"
   - On la marque comme "visitée"
   - On cherche quels sont ses voisins potentiels
   - Parmi ces voisins, on identifie ceux qui sont déjà marqués comme "visités"
   - On choisit au hasard un de ces voisins "déjà visités", et on retire le mur entre ces 2 cases
   - Les voisins qui n'ont pas déjà été visités sont ajoutés à la liste des "cases accessibles"
7) Lorsque la liste des "cases accessibles" est vide, il n'y a plus rien à faire : notre labyrinthe est parfait.

Question 8. Ecrire une fonction qui créé un labyrinthe parfait.

V. Labyrinthes presque parfaits

La principale caractéristique des labyrinthes parfaits est que pour tout couple de cases, il existe exactement un chemin dans le labyrinthe entre ces deux cases.

Dans un tel labyrinthe, la notion de "plus court chemin" n'a donc pas beaucoup de sens : comme il n'existe qu'un seul chemin entre deux cases données, il s'agit forcément du plus court chemin.

Pour étudier les algorithmes de calcul du plus court chemin, nous allons donc utiliser des labyrinthes "presque-parfaits", c'est-à-dire des labyrinthes parfaits dans lesquels certains murs ont été retirés (on obtient ainsi plusieurs chemins possibles entre deux cases).

Question 9. Ecrire une fonction qui génère un labyrinthe presque-parfait.