Percolation

BONNAZ Aymeric, MOUILLET Claude

18 mars 2021

Nom apprenant : _____

Date de réalisation : _____

Question:	1	2	3	4	5	6	Total
Points:	2	2	5	1	2	2	14
Score:							

1 Énoncé

La percolation est un concept abstrait permettant de décrire de nombreux phénomènes comme l'infiltration de l'eau à travers une roche poreuse. Le présent exercice présentera un modèle discret décrivant la couche de roche par une grille discrète de taille N * N cases.
Chaque case est caractérisée par un état ouvert ou bloqué. Dans l'exemple de l'écoulement d'eau, l'état ouvert représente un espace vide et l'état bloqué de la pierre.
L'eau étant présente en haut de la grille, l'objectif de cet exercice est de déterminer statistiquement la fraction de cases vides nécessaires à l'infiltration de l'eau jusqu'en bas de la grille.

Une case ouverte est pleine si elle est en contact avec le haut de la grille. Les voisins d'une case sont les cases qui partagent avec elle des arêtes. Dans le cas courant, il s'agit de la case de gauche, de droite, du haut et celle du bas. Une case pleine est soit une case de la première ligne de la grille, soit un ou plusieurs de ses voisins sont des cases pleines.

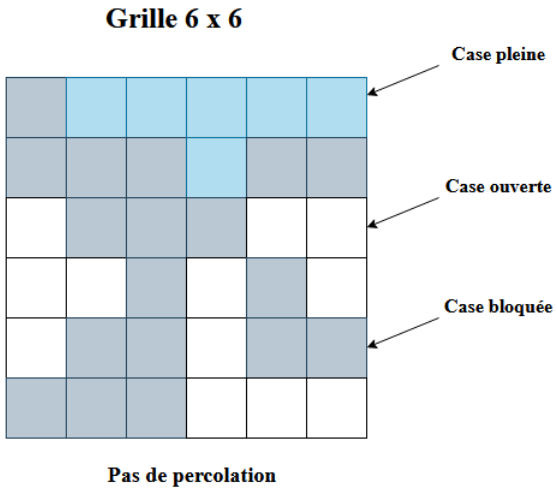

FIGURE 1 – Pas de percolation

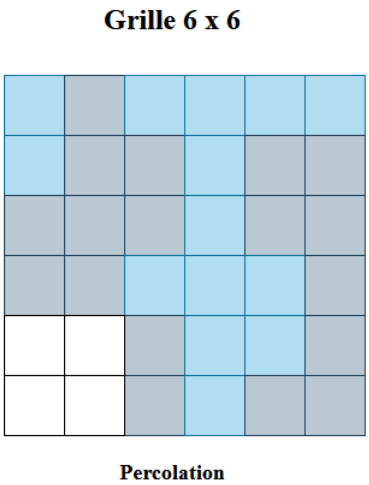

FIGURE 2 – Percolation

2 Exercice

Afin de déterminer la quantité de cases ouvertes nécessaires à la percolation, toutes les cases de la grille seront initialement bloquées puis ouvertes au fur et à mesure de manière aléatoire. La première partie aborde l'implémentation du comportement de la grille. La dernière partie s'occupera de répondre à la question de l'énoncé.

Implémentation : 10 pts

L'objectif de cette partie est l'implémentation des différentes méthodes de la structure de données **Percolation** définie ci-dessous.
Dans celle-ci, la grille de taille N * N est représentée par deux tableaux de booléen à deux dimensions, **Open** et **Full**. Les propriétés de la case (i, j) sont accessibles par les éléments **Open**[i][j], **Full**[i][j]. La case (0, 0) représente le coin supérieur gauche.

Percolation :

- **IsOpen**(int i, int j) : bool
- **IsFull**(int i, int j) : bool
- **Open**(int i, int j) : void
- **Percolate**() : bool
- **CloseNeighbors**(int i, int j) : List<KeyValuePair<int,int> >

1. Propriétés du système :
   (a) (1 point) Est ce que la case (i, j) est ouverte/pleine ?
   Implémenter les méthodes **IsOpen**(int i, int j) : bool et **IsFull**(int i, int j) : bool.
   (b) (1 point) Est-ce que la percolation a lieu ?
   Implémenter la méthode **Percolate**() : bool.

Afin de correctement de compléter la méthode **Open**(i,j), il est important de déterminer l'ensemble des voisins d'une case.

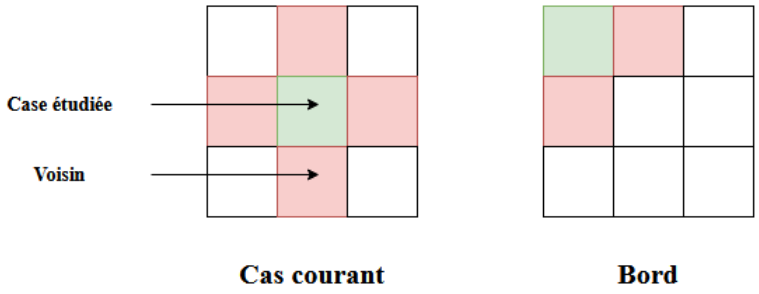

FIGURE 3 – Détermination des voisins d'une case de la grille

2. (2 points) Implémenter la méthode déterminant les proches voisins :
   CloseNeighbors(int i, int j) : List<KeyValuePair<int,int> >.

Suggestion : Faire attention aux effets de bords.

Ouvrir une case peut avoir un impact sur le reste de la grille. Rappelons que si une case ouverte a pour voisin une case pleine, elle devient pleine. Si une case est pleine, ses voisins ouverts deviennent pleins.

3. Ouverture d'une case :
   (a) (3 points) Implémenter la méthode **Open**(int i, int j) : void.
   (b) (1 point) Quelle est la performance de cette méthode dans le pire cas ?

   (c) (1 point) Expliquer, intuitivement, pour quelles raisons ce cas a-t-il peu de chances de se produire ?

Suggestion : Si la case à ouvrir devient pleine, répercuter cette nouvelle situation.

4. (1 point) Modifier **Open**(i,j) afin d'obtenir une version à temps d'exécution constant pour **Percolate**.

À chaque ouverture d'une case, si celle-ci devient pleine, sa nouvelle situation doit être répercutée à toutes les cases ouvertes qui lui sont connectées.

Simulation : 4 pts

À présent, l'implémentation précédente permet de répondre à la question de l'énoncé à savoir la proportion moyenne de cases à ouvrir afin que la percolation se produise.
La structure **PercolationSimulation** contient les méthodes à implémenter dans cette dernière partie. **PclData** contient deux grandeurs, la moyenne, l'écart-type de la fraction de cases ouvertes à calculer.

PercolationSimulation :

- **PercolationValue**(int size) : double
- **MeanPercolationValue**(int size, int t) : **PclData**

L'algorithme permettant d'obtenir suit les étapes suivantes :

- Initialiser une grille de taille N * N avec l'ensemble des cases bloquées.
- Choisir aléatoirement une case bloquée et l'ouvrir.
- Tester si la percolation se produit.
- Réaliser la deuxième et troisième étape jusqu'à ce que la percolation est lieue.
- Retourner la valeur cases ouvertes/nombre total de cases.

5. (2 points) Implémenter la méthode suivante : **PercolationValue**(int size) : double

Suggestion : La classe **Random** permet de générer des entiers aléatoires.

Afin d'obtenir une valeur plus précise, il s'agit de réaliser un nombre **t** de simulations. On obtient aisément la moyenne et l'écart type des valeurs obtenues. La moyenne et l'écart-type de ces valeurs sont définies dans l'équation (1) où p_i représente la valeur de la simulation i.

$$\bar{p} = \frac{1}{t}\sum_{i=1}^{t} p_i \qquad \sigma = \sqrt{\left(\frac{1}{t}\sum_{i=1}^{t} p_i^2\right) - \bar{p}^2} \tag{1}$$

6. (2 points) Implémenter la méthode : **MeanPercolationValue**(int size, int t) : **PclData**