Devoir Maison n°11

Informatique : étude d'un jeu de plateau

Dans tout le problème, n désigne un entier naturel non nul.
On dispose d'un plateau carré de côté n, comportant donc n^2 cases.
On dispose de n^2 jetons, qu'on lance successivement (ou ensemble) sur le plateau.
Certains jetons vont tomber du plateau, d'autres vont atterrir dans une case du plateau.
On suppose qu'une case du plateau peut contenir un nombre illimité de jetons.
Des joueurs, à tour de rôle, lancent les n^2 jetons sur un plateau vide.
On s'intéresse à deux façons de gagner le jeu :

- Obtenir la case la plus peuplée, c'est-à-dire contenant le plus de jetons,
- Obtenir la plus grande proportion de cases vides.

Le but du problème est d'analyser informatiquement ce à quoi on peut s'attendre, en modélisant le jet d'un jeton de la façon suivante :

- une case du plateau est désignée au hasard, et équiprobablement,
- si cette case est sur le bord du plateau (ou dans un coin), alors le jeton tombe du plateau,
- sinon, le jeton tape cette case puis effectue un rebond qui l'amène finalement, et équiprobablement, dans une des 8 cases adjacentes à cette case.

Un plateau est modélisé informatiquement par un tableau `numpy` (une matrice) carré de taille n.
Le coefficient de position (i, j) de ce tableau représente le nombre de jetons qui ont finalement atterri dans la case (i, j).

Il est conseillé (mais pas obligatoire) de traiter ce problème devant un ordinateur, on pourra alors rendre le script réalisé (ou une copie d'écran).

1. **Reconnaissance du bord**
   Écrire une fonction `bord(i,j,n)` qui renvoie `True` si la case (i, j) d'un plateau de taille n est sur le bord (ou dans un coin), et `False` sinon.
   On rappelle que dans un tableau numpy de taille $n \times n$, les indices de lignes et de colonnes i et j varient dans $[\![0, n-1]\!]$.
2. **Traitement du rebond**
   Écrire une fonction `rebond(i,j)` qui renvoie le *tuple* (k, ℓ) désignant une case aléatoire voisine de la case (i, j), chacune avec une probabilité $\frac{1}{8}$.
3. **Simulation d'un jeu**
   (a) Quelle fonction du module `numpy` permet de créer la matrice nulle de taille $n \times n$?
   (b) En se servant des fonctions précédentes, écrire une fonction `jouer(n)` renvoyant la matrice obtenue à l'issue d'une manche du jeu.
4. **Case la plus peuplée**
   (a) Écrire une fonction `populationMax(n)` renvoyant le nombre maximal de jetons dans une seule case, à l'issue de la simulation d'une manche du jeu.
   (b) En déduire une fonction `popMaxMoyenne(n,N)` renvoyant une estimation du nombre maximal de jetons dans une case, qu'on obtient, en moyenne, après N simulations du jeu.
   (c) Écrire une fonction `graphePopMax(n,N)` représentant graphiquement, par des points séparés, les valeurs données par `popMaxMoyenne(k,N)` pour $k \in [\![1, n]\!]$.
   (d) En utilisant `graphePopMax(20,1000)`, que peut-on conjecturer sur le nombre maximal de jetons dans une seule case, en moyenne, quand n augmente ? (∗)
5. **Proportion de cases vides**
   (a) Écrire une fonction `casesVides(n)` renvoyant une simulation du nombre de cases vides après une manche du jeu.
   (b) En déduire une fonction `ratioVide(n,N)` renvoyant la proportion moyenne de cases vides après N simulations du jeu.
   (c) Écrire une fonction `grapheCasesVides(n,N)` représentant graphiquement, par des points séparés, les valeurs données par `ratioVide(k,N)` pour $k \in [\![1, n]\!]$.
   (d) En utilisant `grapheCasesVides(20,1000)`, que peut-on conjecturer sur la proportion de cases vides quand n augmente ? (∗)

(∗) *Temps de calcul : environ 1 minute.*