# TP de révision
# Le solitaire de Schelling

Thomas Crombie Schelling est un économiste américain ayant obtenu le « prix de la Banque de Suède en sciences économiques en mémoire d'Alfred Nobel », parfois abrégé en « prix Nobel d'économie ». L'un de ses travaux porte sur la ségrégation urbaine. Il a imaginé un modèle très simple, appelé « solitaire de Schelling ».

On suppose que la population d'une ville est séparée en deux groupes totalement distincts, que nous nommerons noirs et blancs. La ville est modélisée par un damier, chaque case est occupée par un habitant (qui est soit noir, soit blanc) ou est vide. Nous appelons **voisins** d'un individu tous les individus situées sur les cases adjacentes (diagonales comprises) ; ainsi, chaque individu a au plus 8 voisins. Chaque habitant est supposé avoir une tolérance limitée vis à vis des personnes de l'autre couleur, et devient **mécontent** lorsque les voisins de l'autre couleur représentent deux tiers ou plus de l'ensemble de ses voisins. Par exemple, un habitant ayant cinq voisins est mécontent s'il n'a pas au moins deux voisins de sa couleur.

Ce modèle va évoluer en plusieurs tours. À chaque tour, un individu mécontent tiré au hasard déménage. Il se déplace sur un emplacement vide tiré au hasard (il peut alors devenir content ou rester mécontent). On itère ce procédé jusqu'à obtenir une ville stable, c'est-à-dire une ville sans mécontent. Enfin, on observe si il y a, dans la ville finale, des « ghettos »de couleur.

**Proposition 1** (Modèle informatique)**.** Informatiquement, le modèle est constitué de 4 éléments :
— Une matrice carrée `M` qui représente la ville. C'est un `array` à deux dimensions. Les `1` de la matrices représentent les blancs, les `2` les noirs, et les `0` les emplacements vides.
— Un entier `n`, qui est la taille de la matrice `M` (son nombre de colonnes et de lignes).
— Une liste `LV` des cases vides de `M`, c'est-à-dire des cases de `M` contenant un zéro.
— Une liste `LM` des cases contenant des habitants mécontents.

Une « case »est un couple d'entier `i,j` où `i` et `j` sont deux entiers compris entre `0` inclus et `n` exclus qui représentent respectivement le numéro de ligne et le numéro de colonne de la case.

Dans toute la suite, `M`, `n`, `LV` et `LM` auront le sens décrit ci-dessus sans que ce soit à chaque fois rappelé. Ces différentes valeurs doivent rester « cohérentes »entre elles, c'est pour cela que tout au long du programme, on veille à ce que les conditions suivantes (appelées invariants) soient préservés :

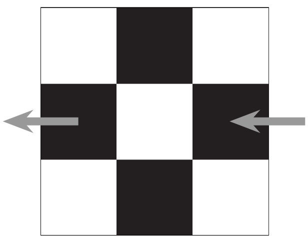
**Proposition 2** (Invariants)**.** Lors de l'évolution de la simulation, 3 propriétés restent invariantes :

1. Il n'y a pas de doublons (i.e., pas deux fois la même valeur) dans `LV`, ni dans `LM`.
2. Une case `i,j` est dans `LV` si et seulement si `M[i, j] = 0`.
3. Une case est dans `LM` si et seulement elle contient un individu mécontent.

Pour éviter des effets liés aux bords de la matrice, et pour nous simplifier la programmation en évitant à avoir à considérer différents cas (selon que la case considérée est ou non sur un bord de la matrice), nous utilisons le modèle torique.

**Définition 1.** Dans le **modèle torique**, si on se déplace d'une case « hors »de la matrice, on se retrouve de l'autre côté, comme illustré sur la figure 1. Ce modèle est dit torique, car il revient à considérer que notre damier a été dessiné sur un tore, voir figure 2.

Dans le modèle torique, chaque case est voisine de huit autres cases, y compris les cases au bord du damier. Attention toutefois, un habitant peut avoir moins de huits voisins, car des cases peuvent être inoccupées.



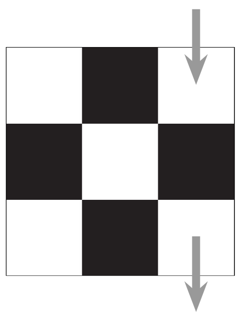


FIGURE 1 – Lorsqu'on sort de la matrice par la gauche, on se retrouve à droite. Lorsqu'on sort de la matrice par le bas, on se retrouve en haut.

Nous utilisons les bibliothèques suivantes.

```
import numpy as np
```

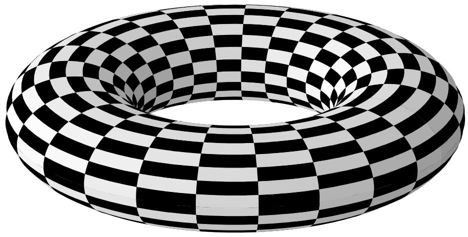


FIGURE 2 – Un damier sur un tore.

```
import matplotlib.pyplot as plt
import matplotlib.colors as col
import random
```

En particulier, il faudra utiliser les fonctions suivantes du module random :

— `random()` : renvoie un nombre aléatoire $x \in [0; 1]$ sous forme de flottant

— `randint(a,b)` : renvoie un nombre aléatoire $n \in [a; b]$ sous forme d'entier

**Question 1.** Écrire une fonction `rand_couple(n)` qui étant donné un entier `n`, tire au hasard (uniformément) un couple d'entiers chacun compris entre 0 inclus et `n` exclus.

**Question 2.** Écrire une fonction `rand_pop(L)` qui, étant donné une liste `L`, retire au hasard à `L` un élément et renvoie sa valeur. La méthode `L.pop(i)` peut être ici utile.

**Question 3.** Écrire une fonction `quadrillage(n)` qui étant donné un entier `n`, renvoie une matrice de dimension $n \times n$,contenant alternativement des uns et des deux comme il y a alternance de blancs et noirs sur un vrai damier. Comme nous n'utilisons dans ce TP que 3 valeurs (0, 1 et 2), la fonction renvoie un `ndarray` contenant des entiers 8 bits non signés (utiliser l'argument optionnel `dtype = np.uint8`).

Pour afficher une matrice `M`, on utilise la commande suivante :

```
plt.imshow(M, interpolation='Nearest', cmap=col.ListedColormap(['grey', 'white', 'black']), vmin=0)
```

**Question 4.** Écrire une fonction `voisinage(M, n, i, j)` qui renvoie un triplet de valeurs : le nombre de 0, le nombre de 1 et le nombre de 2 dans les 8 cases voisines de la case `i, j`.

**Question 5.** Écrire une fonction `pascontent(M, n, i, j)` qui renvoie `True` si la case `i, j` contient un individu mécontent et `False` sinon.

Pour initialiser une matrice, nous utiliserons le procédé suivant :

— Nous créons une matrice avec la fonction `quadrillage`.

— Nous tirons au hasard `nbvides` cases, et les vidons (i.e. remplaçons leur contenu par des zéros).

— Nous remplissons au hasard `nbnouv` cases vides par des 1 ou des 2 (on tire au hasard avec une probabilité de 50%/50%).

*On suppose bien évidemment que `nbvides` est strictement plus grand que `nbnouv`. À la fin de l'initialisation, la matrice contient exactement `nbvides - nbnouv` cases vides (pas une de plus, pas une de moins).*

**Question 6.** Écrire une fonction `init(n, nbvide, nbnouv)` qui initialise une matrice. Cette fonction doit renvoyer `M`, `LV` et `LM`.

**Question 7.** Écrire une fonction `deplace(M, n, LV, LM)` qui tire au hasard un mécontent et le déplace au hasard dans une place vide, tout en maintenant les invariants.

**Question 8.** Compléter la fonction précédente pour qu'elle mette à jour les listes `LM` et `LV` suite au déplacement. Attention, cette opération peut être très gourmande. Votre proposition doit être de compléxité inférieure à $\mathbf{O}(n)$.

**Question 9.** Écrire une fonction `evolution(M, n, LV, LM, limit=1000)` qui fait évoluer le modèle jusqu'à ce que `LM` soit vide ou qu'on atteigne un nombre d'itérations égal à `limit`.

**Question 10.** Simuler quelques villes avec les paramètres suivants : `n = 10`, `nbvides = 20`, `nbnouv = 5`. Après ces simulations, qu'observe-t-on ?

**Question 11.** Simuler quelques villes avec les paramètres suivants : `n = 100`, `nbvides = 2000`, `nbnouv = 500`. Observe-t-on des ghettos, c'est-à-dire des zones habités par une seule couleur ?

*Si vous n'avez pas fait attention à la complexité, le cas `n = 100` pourrait poser problème.*