Simulation de la ségrégation spatiale

Projet Python — 1A ENSAE

Nathan LANGLOIS, Emma LEGUAY

Mai 2022

1 Introduction

1.1 Objectif

L'objectif principal de ce projet est de montrer comment les préférences de catégories de populations différentes concernant leur voisinage aboutissent à une ségrégation spatiale. Il s'agit dans un second temps d'étudier comment cette ségrégation varie en fonction de divers paramètres, tels que le seuil de tolérance des individus, la portée du voisinage considéré, le nombre de populations différentes.

1.2 Méthodologie

1.2.1 Le modèle de ségrégation spatiale de Schelling

Notre code a été construit initialement à partir du modèle de ségrégation spatiale de Schelling [1]. Nous avons choisi ce modèle car il assez facilement transposable en Python : la ville est représentée par une grille, en fait une matrice, particulièrement simple à manipuler à l'aide du module Numpy. De même, les « déménagements » des habitants de cette « ville » et leur règle de satisfaction se prêtent bien au langage Python.

Le modèle est construit ainsi : une grille de dimension (L, H) avec $N = L \times H$ emplacements qui la composent, dont une petite partie laissés libres et le reste habités par des individus appartenant à l'une des deux catégories rouge et bleu et partageant un même seuil de tolérance (i.e., un seuil $t_{min} \in [0, 1]$ tel que les individus sont satisfaits si la part de leurs voisins de même catégorie qu'eux est supérieure à t_{min}). La dynamique du modèle est la suivante : tant qu'il y a des habitants insatisfaits, on tire au sort un habitant parmi ces insatisfaits qui déménage dans un emplacement libre aléatoire de la ville. La dynamique prend donc fin lorsque tous les habitants sont satisfaits, et on peut alors constater qu'une ségrégation spatiale plus ou moins marquée s'est installée.

1.2.2 Calcul de la ségrégation

Comme présenté dans la section 2.2, nous avons également ajouté à notre modèle le calcul de la ségrégation dans la ville. Pour ce faire, nous nous sommes fondés sur un indice de ségrégation trouvé dans un article [2] de sciences sociales produits par des chercheurs du CNRS et de l'Observatoire Français de la Conjoncture Économique. C'est un indice qui calcule d'abord l'entropie normalisée,

1. Pour plus de détails, voir https://en.wikipedia.org/wiki/Schelling's_model_of_segregation.
2. Lien de l'article : http://piketty.pse.ens.fr/files/Forse2006.pdf.

en fonction des proportions de voisins différents et de voisins similaires dans le voisinage de chaque habitant. L'entropie H s'écrit :

$$H = -\frac{1}{N_{tot} \ln 2} \sum_{i,j} \left(\frac{C_{ij}}{N_{ij}} \ln \left(\frac{C_{ij}}{N_{ij}} \right) + \frac{N_{ij} - C_{ij}}{N_{ij}} \ln \left(\frac{N_{ij} - C_{ij}}{N_{ij}} \right) \right), \tag{1}$$

où N_{tot} représente le nombre total d'habitants ayant des voisins, N_{ij} le nombre d'habitants dans le voisinage de l'habitant résidant en (i, j) (lui-même inclus) et C_{ij} le nombre d'habitants de même catégorie (i.e., de même couleur, dans la modélisation classique, que nous avons adoptée) (à nouveau, lui-même inclus).

L'entropie H ainsi définie varie entre 0 et 1, et on obtient à partir de H l'indice de ségrégation

$$S = 1 - H, \tag{2}$$

qui varie également entre 0 et 1 (0 pour une mixité parfaite, 1 pour une ségrégation totale).

2 Approche adoptée

2.1 Modèle de base

Notre premier objectif était de pouvoir simuler le modèle de Schelling de base, c'est-à-dire comme présenté dans la section 1.2.1. Pour ce faire, nous avons structuré notre code en 3 sections :

- l'importation des packages/modules ;
- la déclaration des variables et fonctions globales et des paramètres par défaut (dimensions, catégories, couleurs, proportions, etc.) ;
- la déclaration des classes (partie centrale de notre code), au nombre de 3 et qui constituent l'armature de notre code : `Habitant`, `Ville` et `Emplacement`.

2.2 Approfondissements successifs

Après avoir réussi ce premier objectif, nous avons décidé d'aller plus loin et d'augmenter le nombre de paramètres ajustables. Voici les approfondissements successifs que nous avons réalisés.

Extension de la notion de tolérance Dans le modèle de base, la tolérance est définie comme la proportion minimale de voisins de même catégorie pour qu'un habitant soit satisfait. Pour obtenir des résultats plus intéressants [3], nous avons décidé d'ajouter également un seuil maximal pour la proportion de voisins de même catégorie : en abaissant ce seuil, les habitants deviennent insatisfaits si l'entre soi est trop fort. Finalement, la tolérance dans notre code est un couple (t_{min}, t_{max}), assimilable à un intervalle, tel que l'habitant est satisfait si la part de ses voisins de même catégorie que lui est supérieure à t_{min} ET inférieure à t_{max}.

Ajout de catégories Si nous avions initialement prévu de ne considérer que deux catégories d'habitants pour simuler la ségrégation – c'est ce que propose le modèle de Schelling –, nous avons choisi par la suite d'étendre notre modèle aux cas à plusieurs catégories (il est en pratique très simple, dans notre code, d'étendre le nombre de catégories).

Cet ajout a nécessité une refonte de plusieurs fonctions et d'une partie du code, qui étaient codées pour certaines catégories aux noms explicites (`'rouge'` et `'bleu'`, précisément) et donc peu flexibles face à l'ajout de nouvelles catégories.

3. Ces résultats nous ont été inspirés par le site https://ncase.me/polygons/.

Extension de la notion de voisins Dans un premier temps, pour le calcul de la satisfaction, chaque habitant ne se basait que sur la catégorie de ses voisins directs (les 8 l'entourant) – encore une fois, c'est ainsi qu'est conçu le modèle de Schelling. Cependant, par souci de réalisme, un habitant se souciant en pratique de tout son voisinage et pas uniquement de ses voisins directs, nous avons opté pour une extension de la notion de voisins (à l'aide de l'argument `portee`, présent dans plusieurs méthodes de nos classes), selon une logique de pondération (plus un voisin de l'habitant considéré est proche, plus sa catégorie influe sur la satisfaction de cet habitant – dans les faits, la pondération est tout à fait personnalisable et n'est donc pas limité à cette intuition).

Optimisation du temps d'exécution L'une des problématiques essentielles pour optimiser le temps d'exécution est de réussir à savoir quelles opérations sont les plus coûteuses en temps. C'est un travail long et fastidieux, que nous avons effectué. Il est apparu que l'opération la plus coûteuse était celle de calcul des voisins (effectuée à chaque déménagement dans les zones de départ et d'arrivée de l'habitant ayant déménagé, et donc au cœur de la structure du modèle). Nous l'avons donc optimisé : la version initiale recalculait tous les voisins de tous les habitants dans les zones autour (i.e., toutes les cases à une distance inférieure à la portée entrée en paramètre) des emplacements de départ et d'arrivée du « nomade » ; la nouvelle version ne recalcule pas tous leurs voisins, mais se contente de supprimer le nomade de leurs voisins pour les habitants dans la zone de départ, et d'ajouter le nomade à leurs voisins pour les habitants dans la zone d'arrivée.

L'amélioration de la fonction `actualiser_zone` a ainsi permis de diviser par 1,4 le temps de résolution pour une portée de 1, et par 5 pour une portée de 3 (c'était bien l'intérêt de cette amélioration que de pouvoir plus facilement étudier de longues portées).

Après cette amélioration, il apparaît encore que 5/8 du temps de résolution est consacré aux exécutions de la fonction `actualiser_zone`, ce qui reste considérable. L'essentiel du reste du temps d'exécution est consacré au choix aléatoire d'un habitant insatisfait et d'un emplacement libre, mais nous n'avons pas réussi à l'optimiser de façon significative (nous avons simplement tenter de choisir aléatoirement un nombre puis de prendre l'élément à cet indice dans la liste, plutôt que de choisir aléatoirement un habitant et un emplacement, qui sont des objets plus complexes et donc potentiellement plus coûteux à manipuler...).

Personnalisation de la tolérance selon les catégories Dans le modèle de base, tous les habitants sont aussi tolérants aux habitants des autres catégories, quelque soit leur catégorie. Pour plus de réalisme, nous avons choisi de laisser la possibilité d'avoir des tolérances différentes selon les catégories (par exemple, que les rouges soient plus tolérants que les bleus).

Calcul de la ségrégation Après l'ajout de tous ces paramètres, bien que la visualisation des villes était déjà un élément tout à fait pertinent pour constater la ségrégation spatiale qui s'était mise en place, il nous a paru indispensable de quantifier ce résultat, et nous avons donc cherché à implémenter un indice de ségrégation nous permettant de mesurer objectivement (et non plus « à l'instinct ») la ségrégation des villes que nous construisions. L'indice choisi est présenté en détails dans la section 1.2.2.

Nous sommes allé plus loin, en incluant la possibilité de tracer le graphique représentant l'évolution de la ségrégation d'une ville en fonction du temps (compté en nombre de déménagements)

Enfin, nous avons créé une fonction qui lance n simulations de paramètres identiques à entrer en arguments (i.e., crée n villes, les « résout » et stocke l'indice de ségrégation constaté) puis renvoie la moyenne (et, si désiré, la variance) des ségrégations observées. L'objectif est, pour la moyenne, de pouvoir obtenir des résultats plus solides à l'aide de la loi des grands nombres, et pour la variance de quantifier la variation de ségrégation entre plusieurs simulations pour des paramètres identiques.

2.3 Fonctionnement de notre code final

Une simulation de la ségrégation spatiale prend ainsi la forme suivante.

— La ville est déclarée avec ses paramètres (proportions de chaque catégorie, tolérances des habitants, portée de considération des voisins, pondération des voisins selon leur distance, initialisation des habitants ou non). Par défaut, les habitants sont initialisés lorsque la ville est déclarée, ce qui consiste à :
  — créer N emplacements dans la ville,
  — attribuer (en fonction du paramètre proportions) un certain nombre d'emplacements aux habitants de chaque catégorie et à laisser libres le reste,
  — attribuer à chaque habitant sa tolérance en fonction de sa catégorie, puis ses voisins et donc sa satisfaction ;
  — on peut alors créer la liste des habitants insatisfaits.

— La ville est ensuite « résolue » :
  — on affiche éventuellement la carte de la ville au début (l'affichage de la ville consiste à convertir la carte de la ville en une matrice où à chaque catégorie correspond une valeur, à tracer avec `imshow` les cartes de chaque catégorie puis à les superposer pour obtenir la carte finale) ;
  — tant que la liste des habitants insatisfaits est non-vide (donc tant qu'il reste des habitants insatisfaits), on procède comme suit :
    — on sélectionne un habitant aléatoire parmi les habitants insatisfaits ;
    — on sélectionne un emplacement aléatoire parmi les emplacements libres de la ville ;
    — on opère le déménagement de l'habitant insatisfait vers l'emplacement libre, et on actualise toutes les données susceptibles de changer (les voisins et donc la satisfaction des habitants autour de l'emplacement libéré et de l'emplacement nouvellement occupé) ;
  — la boucle s'arrête lorsque tous les habitants sont satisfaits, et on affiche (par défaut) la carte de la ville à présent ségréguée ;
  — il est également possible d'afficher le graphe de la ségrégation en fonction du temps (compté en nombre de déménagements), ou simplement la valeur finale de la ségrégation dans la ville.

3 Résultats

3.1 Résultats graphiques

Même si nous nous sommes avant tout focalisés sur les résultats quantitatifs en termes de ségrégation, il nous a semblé important de proposer un rendu graphique, pour des paramètres donnés, de ce que peut proposer notre code. Le voici.

Pour les paramètres

— `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1, 'vert': 1}`,

— `tolerances=[0.3, 1]`,

— `portee=2`,

— `ponderation={1: 1, 2: 0.5}`,

on obtient les graphiques suivants :

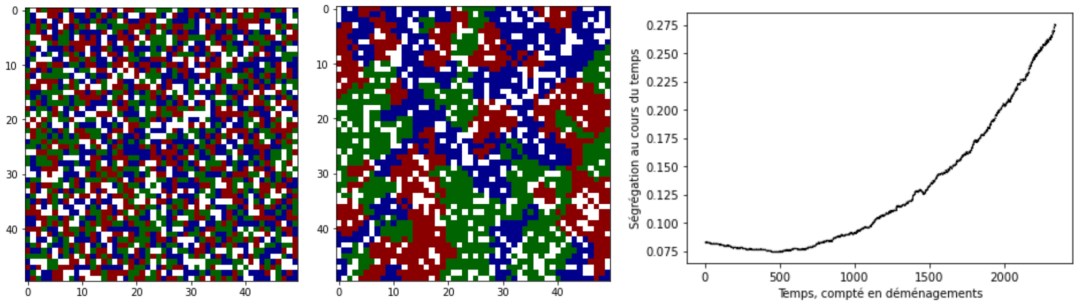

FIGURE 1 – Initialisation, résolution, ségrégation au cours du temps

3.2 Problèmes de convergence

La convergence du problème a été un problème majeur pour la résolution. En fait, dans énormément de cas, et la grande majorité des cas un peu complexes, le problème semblait ne pas converger, ou alors très lentement.

Par exemple, lorsqu'on entre (pour une grille 50×50 et une portée de 1) `tolerances_test = {'rouge': {'min': 0.285, 'max': 1}, 'bleu': {'min': 0.5, 'max': 1}, 'vert': {'min': 0.5, 'max': 1}}`, pas de convergence, mais si on passe 0.285 à 0.286, on converge en moins de 2 secondes... Bref, lorsqu'on personnalise les tolérances selon la catégorie, on arrive vite à des soucis de convergence.

Un autre exemple concerne les seuils de tolérance min et max, et sera évoqué dans une note de la section suivante.

Un dernier concerne la portée. À mesure qu'on l'augmente, la convergence est de plus en plus lente. À partir du seuil `portee=4`, il est vraiment difficile d'obtenir des résultats...

3.3 Ségrégation selon les paramètres

3.3.1 Tolérance variable

Modèle de base : faire varier t_{min}, pour 2 catégories Pour `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1}, tolerances=[`t_{min}`, 1], portee=1`, on obtient sur 10 simulations, selon t_{min}, les couples (moyenne, écart-types) relatifs à la ségrégation suivants :

t_{min}	0,1	0,2	0,3	0,4	0,5	0,6	0,7
$\bar{s}$	0,13	0,19	0,41	0,56	0,68	0,89	0,97
σ	0,01	0,02	0,02	0,03	0,02	0,01	0,01

Conclusion : la ségrégation augmente plus ou moins linéairement à mesure que l'on augmente t_{min}. L'écart-type, lui, reste faible (ce qui nous conforte dans l'idée que 10 simulations suffisent à donner une idée de ce qui se passe !), et atteint ses valeurs les plus hautes autour de $t_{min} = 0,4$.

En abaissant t_{max} Pour `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1}, portee=1`, on obtient sur 10 simulations, selon la tolérance, les ségrégations moyennes suivantes :

Conclusion : l'abaissement de t_{max} permet de diminuer significativement la ségrégation. Il suffit donc que les individus rejettent, même légèrement, l'entre-soi total, pour que la ségrégation se réduise.

t_{max} / t_{min}	0,1	0,3
1	0,13	0,42
0,9	0,8	0,15
0,85	0,7	0,1

Note : en pratique, le problème ne semble « converger » (i.e., le nombre d'habitants insatisfaits descend à 0 en un temps raisonnable) uniquement lorsque $t_{max} \geq 0,85$, et dans le cas où $t_{max} < 1$, uniquement lorsque dans le même temps $t_{min} \leq 0,3$, ce qui explique nos choix paramétriques.

En personnalisant la tolérance selon la catégorie Les résultats ne sont pas extrêmement démonstratifs, car pour cause de problème de convergence évoqués ci-après, nous ne pouvons pas trop augmenter l'écart entre le minimum de tolérance des deux catégories. Cependant, on observe quand même les résultats suivants, pour `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1}, tolerances={'rouge': [`t_R`, 1], 'bleu': [`t_B`, 1]}, portee=1` (les rouges sont extrêmement tolérants et les bleus également mais de façon variable), on obtient sur 10 simulations, selon t_min^B, les ségrégations moyennes suivantes :

(t_R, t_B)	$t_R = 0,1, t_B = 0,1$	$t_R = 0,1, t_B = 0,2$	$t_R = 0,1, t_B = 0,275$
$\bar{s}$	0,12	0,13	0,16

Conclusion : de manière assez peu surprenante, si la tolérance d'une catégorie diminue, cela augmente la ségrégation. Cela est peut-être d'autant plus vrai qu'on ne considère ici que deux catégories, et que le désir d'entre soi d'une catégorie a un impact symétrique sur l'autre catégorie et isole celle-ci.

3.3.2 Proportions variables

En changeant les proportions Pour `proportions={'libre': 0.2, 'rouge': `p_R`, 'bleu': `p_B`}, portee=1`, on obtient sur 10 simulations, selon B et R, les ségrégations moyennes et écarts-types suivants :

(p_R, p_B)	$p_R = 1, p_B = 1$	$p_R = 1, p_B = 2$	$p_R = 1, p_B = 2,75$
$\bar{s}$	0,42	0,50	0,58
σ	0,017	0,016	0,021

Conclusion : plus les proportions sont déséquilibrées, plus la ségrégation augmente. Ceci est assez logique car une catégorie sur les deux a beaucoup moins de chance de pouvoir avoir des voisins différents d'elle-même (ici la catégorie bleue).

3.3.3 Nombre de catégories variable

Modèle de base, faire varier la tolérance, avec 4 catégories Que se passe-t-il si on simule le modèle de base mais en passant de 2 à 4 catégories ? Pour `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1, 'vert': 1, 'jaune': 1}, tolerances=[`t_{min}`, 1], portee=1`, on obtient sur 10 simulations, selon t_{min}, les ségrégations moyennes suivantes :

t_{min}	0,1	0,2	0,3	0,4	0,5	0,6
$\bar{s}$	0,09	0,15	0,38	0,52	0,70	0,96

Conclusion : pour des valeurs basses de t_{min}, la société semble être encore moins ségréguée (ce qui est peu surprenant, car on a plus de catégories donc plus de mixité naturelle), et pour des valeurs hautes, l'être davantage que dans le modèle de base avec 2 catégories.

En personnalisant les tolérances Pour `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1, 'vert': 1}, tolerances={'rouge': [`t_R`, 1], 'bleu': [`t_B`, 1], 'vert': [`t_V`, 1]}, portee=1`, on obtient sur 10 simulations, selon (p_R, p_B, p_V), les ségrégations moyennes suivantes :

p_R, p_B, p_V	$0,3, 0,3, 0,3$	$0,2, 0,3, 0,4$	$0,25, 0,25, 0,4$	$0,2, 0,35, 0,35$	$0,1, 0,4, 0,4$
$\bar{s}$	0,37	0,35	0,28	0,34	0,30

Conclusion : il est difficile de dégager une tendance claire. L'impression est tout de même que plus l'écart de tolérance entre les catégories est important, plus la ségrégation est faible.

3.3.4 Portée variable

En augmentant la portée Que se passe-t-il à présent si on augmente la portée de considérations des voisins ? Pour `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1}, tolerances=[0.5, 1], portee=`P`, ponderation={1: 1, 2: 0.5, 3: 0.2, 4: 0.1}`, on obtient sur 5 simulations, selon P, les ségrégations moyennes et écarts-types suivants :

P	1	2	3	4
$\bar{s}$	0,68	0,68	0,58	0.57
σ	0,01	0,03	0,03	0.10

Conclusion : la portée a un impact sur la ségrégation finale moyenne, qui tend à être moindre à mesure que la portée considérée augmente. C'est un résultat particulièrement intéressant ! En effet, on pourrait critiquer le modèle de base qui, par simplification des choix des agents (considérer uniquement son voisinage immédiat semble moins réaliste aurait tendance, peut-être, à surestimer la ségrégation spatiale.

L'écart-type augmente également, mais il faut garder en tête que ce tableau présente des résultats pour 5 simulations (contre 10 pour les précédents), ce qui amène nécessairement une augmentation de l'écart-type.

Encore plus discriminant À présent, tentons de régler tous les autres paramètres de telle sorte à ce que la variation de la portée ait l'impact le plus important. À défaut d'avoir trouvé le cas où cette variation a les conséquences les plus fortes, en voici un plutôt intéressant.

Pour `proportions={'libre': 0.3, 'rouge': 1, 'bleu': 1, 'vert': 1}, tolerances=[0.3, 1], portee=`P`, ponderation={1: 1, 2: 0.5, 3: 0.2, 4: 0.1}`, on obtient sur 10 simulations, selon P, les ségrégations moyennes et écarts-types suivants :

Ici l'indice de ségrégation varie considérablement (plus que du simple au double !) selon la portée considérée. C'est un résultat particulièrement fort.

P	1	2	3	4
$\bar{s}$	0,39	0,23	0,20	0,16
σ	0,02	0,02	0,04	0,03

Graphiquement, on constate que l'augmentation de la portée laisse apparaître des sortes d'entrelacements entre les différentes catégories (ce qui n'est pas surprenant, mais très joli !).

Le choix de $t_{min} = 0,5$ nous permet de constater que les importants écarts-types de la simulation précédente étaient sûrement dûs au nombre de simulations effectuées, car ici, avec 10 simulations, on retrouve des écarts-types comparables aux autres simulations.

4 Difficultés rencontrées

4.1 Difficultés informatiques

Mise en place d'une animation Bien que ce fût l'un de nos grands souhaits au début du projet, nous n'avons pas réussi à produire des rendus animés de la ségrégation de nos villes, à cause de difficultés techniques et d'un manque de maîtrise de l'extension `matplotlib`.

Temps d'exécution Notre aptitude à produire des données les plus fiables possibles pour les indices de ségrégation moyen a été en partie limitée par le temps d'exécution de notre méthode de résolution d'une ville, en particulier pour des portées élevées ou des tolérances particulièrement contraintes.

5 Extensions possibles

Bien que nous voulions aller plus loin, nous avons été contraints par le temps de renoncer à certains de nos idées visant à complexifier le modèle. Toutefois, si notre travail devait être repris dans l'optique d'être approfondi et amélioré, voici quelques pistes possibles sur lesquels il serait intéressant de se pencher :

- renforcer l'hétérogénéité à plusieurs niveaux :
  - la tolérance :
    - des catégories : on pourrait imaginer que chaque catégorie ait une tolérance spécifique envers chaque autre catégorie (au lieu d'une tolérance uniforme envers toutes les autres catégories), avec une sorte de matrice de tolérance (pas nécessairement symétrique, car une catégorie A peut tolérer une catégorie B davantage que la B ne tolère la A) ;
    - des individus : on pourrait faire varier légèrement de façon aléatoire la tolérance des individus autour de la tolérance de leur catégorie ;
  - la portée : dans notre code, la portée de considération des voisins est la même pour tous les habitants d'une ville. On pourrait également imaginer la personnaliser, soit à l'échelle des catégories, soit (toujours plus ambitieux !) à l'échelle des individus. Mais cela rendrait significativement plus complexe la gestion de l'actualisation des zones de provenance et d'arrivée lors des déménagements, car les individus affectés dépendraient de leur portée et plus seulement de l'individu qui déménage... ;

— systématiser l'étude des résultats pour en tirer le plus intéressant : nous n'avons malheureusement pas eu suffisamment de temps pour tester autant de combinaisons de paramètres que nous le souhaitions. C'est pourtant ce qu'il faudrait faire pour tirer les résultats les plus intéressants de notre modèle. Nous n'en avons donné qu'un aperçu.

6 Annexe : quelques cartes ségréguées

Dans cette section, le lecteur pourra trouver quelques cartes, belles ou intrigantes, auquel certains paramètres (indiqués) peuvent mener (avec `np.random.seed(0)`).

Paramètres :

— `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1}`,

— `tolerances=[0.3, 0.9]`,

— `portee=1`.

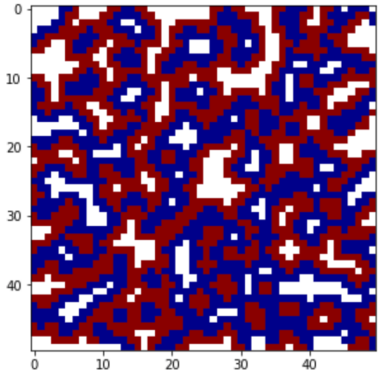

FIGURE 2 – Quelle harmonie !

Paramètres :

— `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1}`,

— `tolerances=[0.7, 1]`,

— `portee=1`.

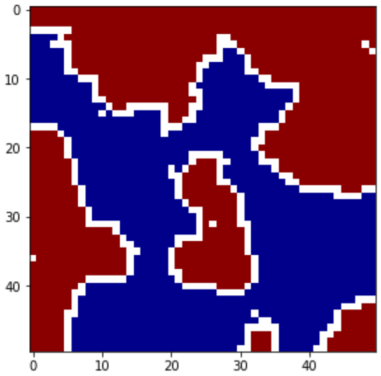

FIGURE 3 – Le drapeau de la Norvège ?

Paramètres :

— `proportions={'libre': 0.2, 'rouge': 1, 'bleu': 1, 'vert': 1}`,

— `tolerances={'rouge': [0.1, 1], 'bleu': [0.4, 1], 'vert': [0.4, 1]}`,

— `portee=1`.

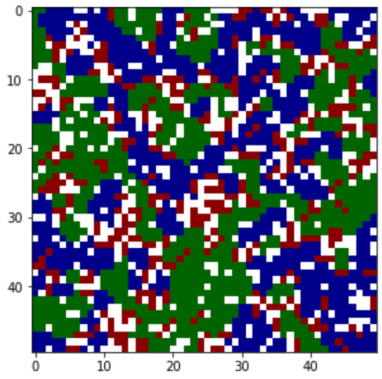

FIGURE 4 – Quand les rouges arbitrent