MATh.en.JEANS - 2022/2023

2					5	6	9	
	6		1	9				
		5	4	3	6	1		
4		6	5	8				
7		8				4		9
				4	7	2		5
		4	2	6	8	9		
				1	3		4	
	8	9	7					6

Sudoku

2	4	1	8	7	5	6	9	3
3	6	7	1	9	2	8	5	4
8	9	5	4	3	6	1	7	2
4	2	6	5	8	9	7	3	1
7	5	8	3	2	1	4	6	9
9	1	3	6	4	7	2	8	5
5	3	4	2	6	8	9	1	7
6	7	2	9	1	3	5	4	8
1	8	9	7	5	4	3	2	6

Chercheuse associée :
Marie Anastacio
RWTH Aachen University

Professeurs encadrants :
Line Boissonnet, Stéphane Beringue,
Florence Decool, Mathieu Buchwald

Élèves 1ère Lycée van Gogh de la Haye:
Bregje, Myrte, Cesare, Paul, Clément et Yann

Introduction

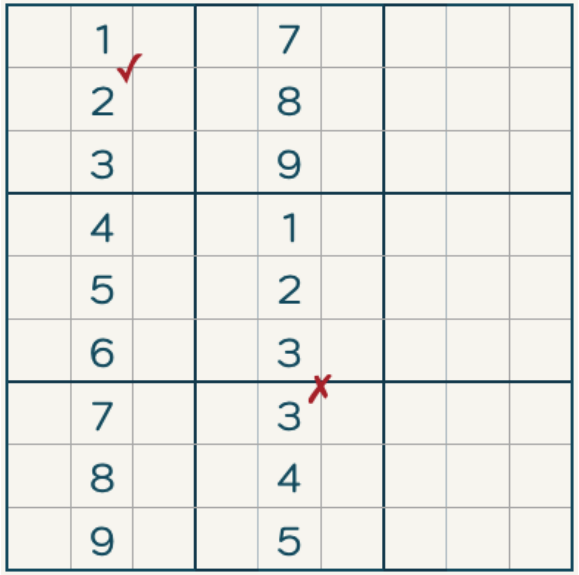
Sujet et rappel des règles

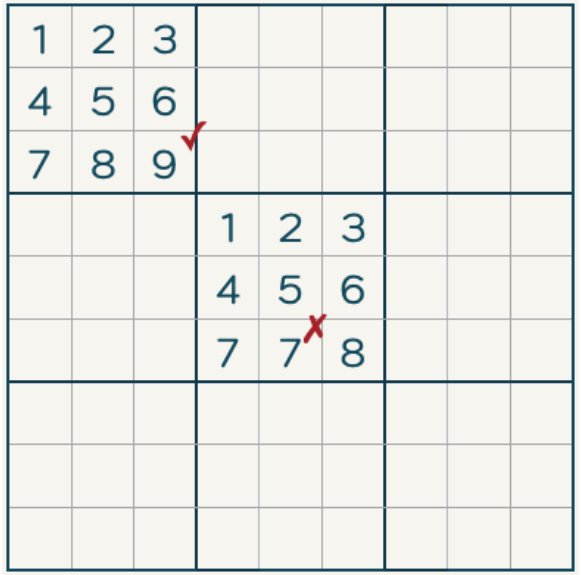
Présentation du Sujet:

Représenter le Sudoku mathématiquement ainsi que ses solutions

Sudoku – rappel des règles

- Grille de 9×9, divisée en 9 carrés de 3×3
- Chaque ligne et colonne doit contenir une occurrence de chaque chiffre de 1 à 9
- Chaque carré de 3×3 ne peut contenir qu'une fois le même chiffre





Sommaire

- Partie I -- Représentation mathématique
  - Les différentes configurations possibles
  - Configurations des zones adjacentes
- Partie II -- Représentation avec Python
  - Listes
  - Matrices
  - Pygame
  - Algorithme solveur

§ Ouverture

Questions intermédiaires

- Quel est le nombre de grilles possibles ?
- Quelles sont les différentes possibilités dans un Sudoku ?
- Comment représenter une grille de Sudoku avec Python ?
- Comment coder un algorithme permettant de résoudre le Sudoku ?

➢ Nous nous sommes posé plusieurs questions au début de nos recherches, telles que le nombre de grilles possibles, comment représenter une grille de sudoku avec Python et comment coder un algorithme permettant de résoudre le Sudoku.

Partie I

La première partie de notre recherche s'est concentrée sur la représentation purement mathématique du Sudoku, sans chercher à le résoudre

Différentes configurations possibles pour un Sudoku 9×9

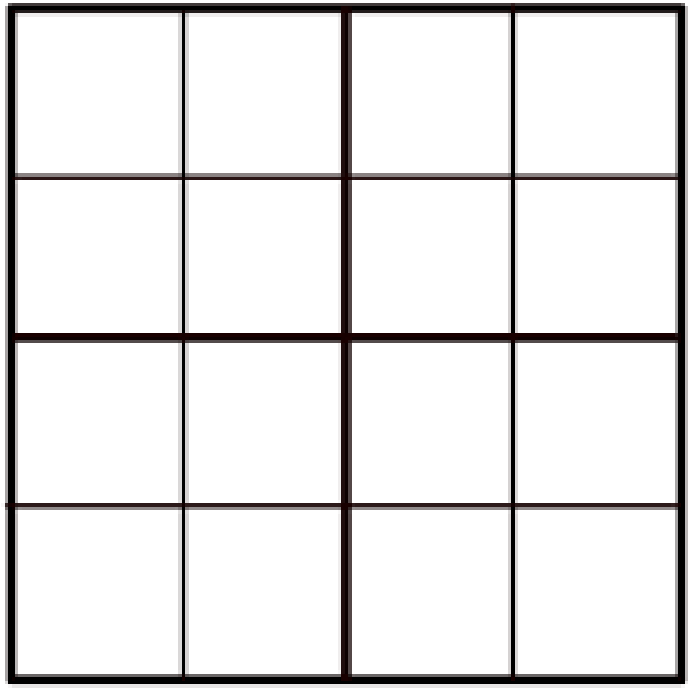
Nos étapes de recherche:

- On commence par calculer les différentes configurations pour un 4×4
- Objectif : trouver une formule qui nous permettra de mieux comprendre la marche à suivre pour le 9×9
- Ensuite, nous avons essayé de calculer pour le 9×9, mais par manque de temps nous n'y sommes pas parvenus
- Nous avons commencé par calculer les différentes configurations pour un 4×4, dans l'optique de discerner d'éventuelles formules qui nous permettraient de résoudre le cas d'un 9×9. Cependant, par manque de ressources et de temps nous n'avons pas réussi à résoudre une grille 9×9.(1)

Passer par un 4×4

Nous nous sommes d'abord posé la question:

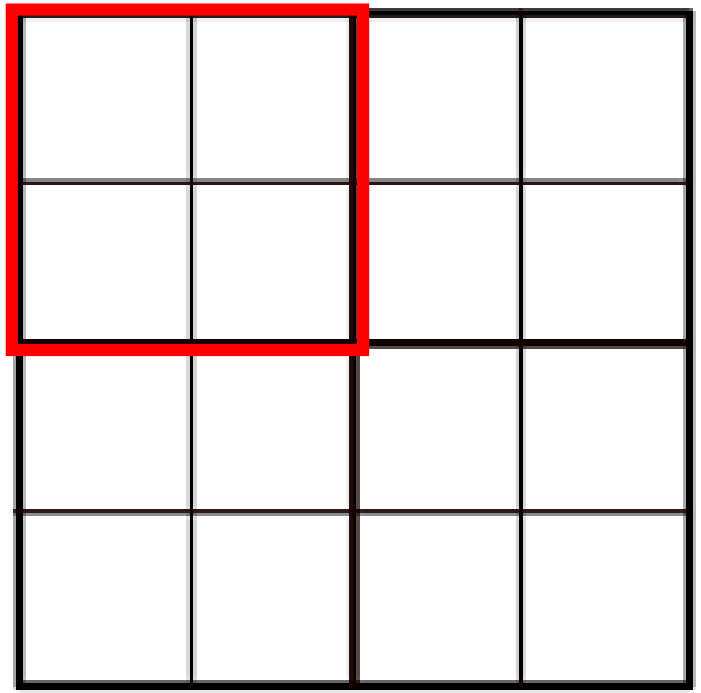
Comment trouver toutes les différentes configurations d'un 4×4 ?



Passer par un 4×4

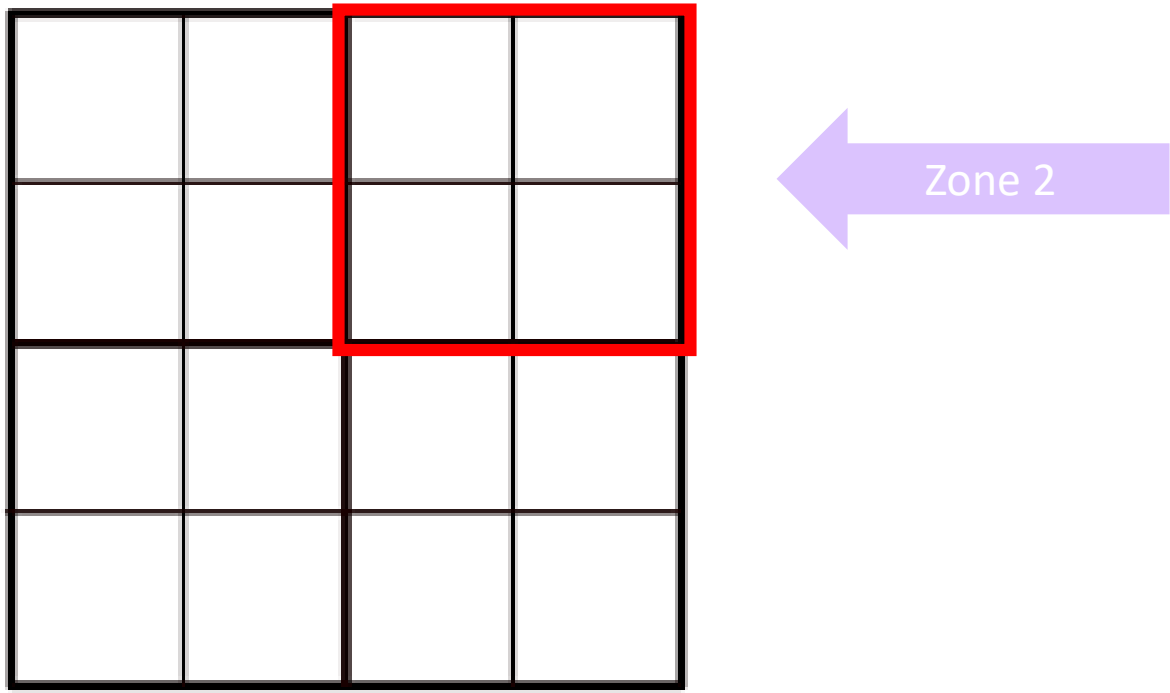
Cette zone est la première zone

Zone 1



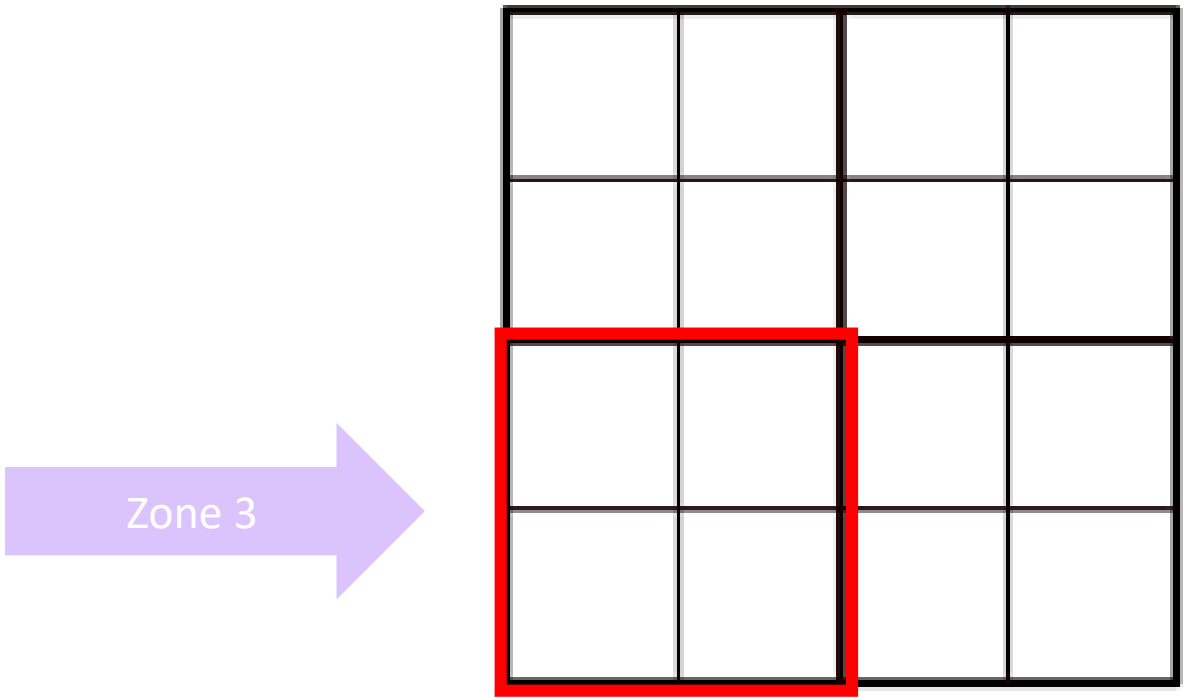
Passer par un 4×4

Celle-ci est la deuxième zone



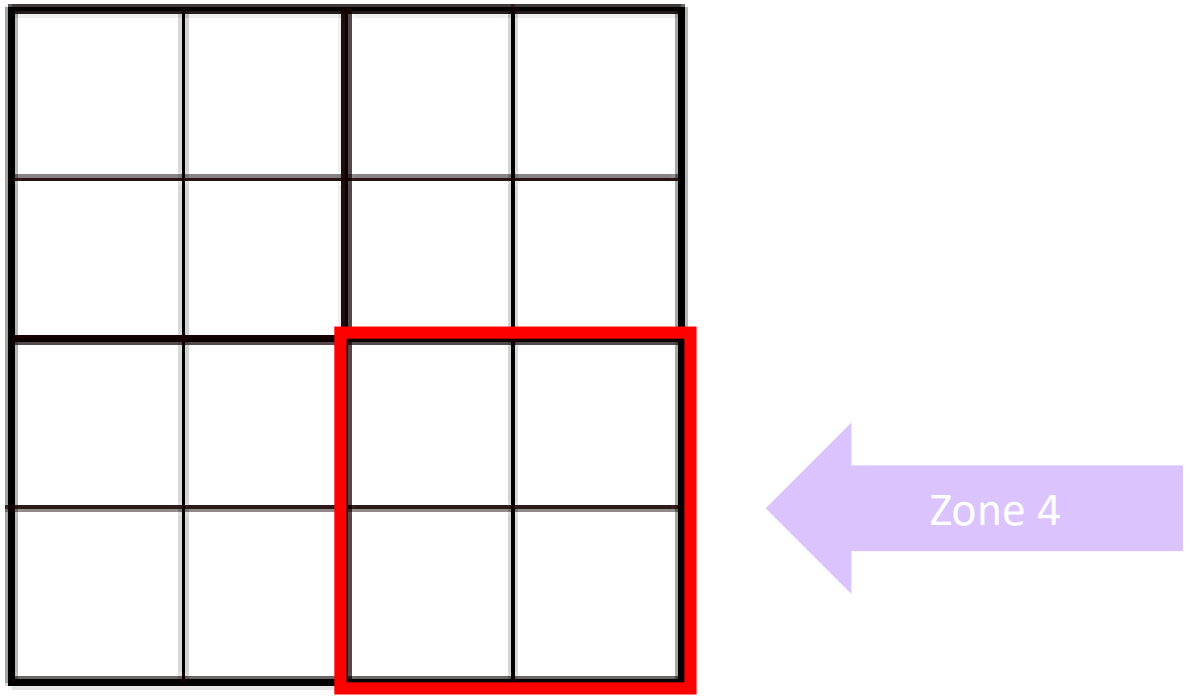
Passer par un 4×4

Celle-ci est la troisième zone

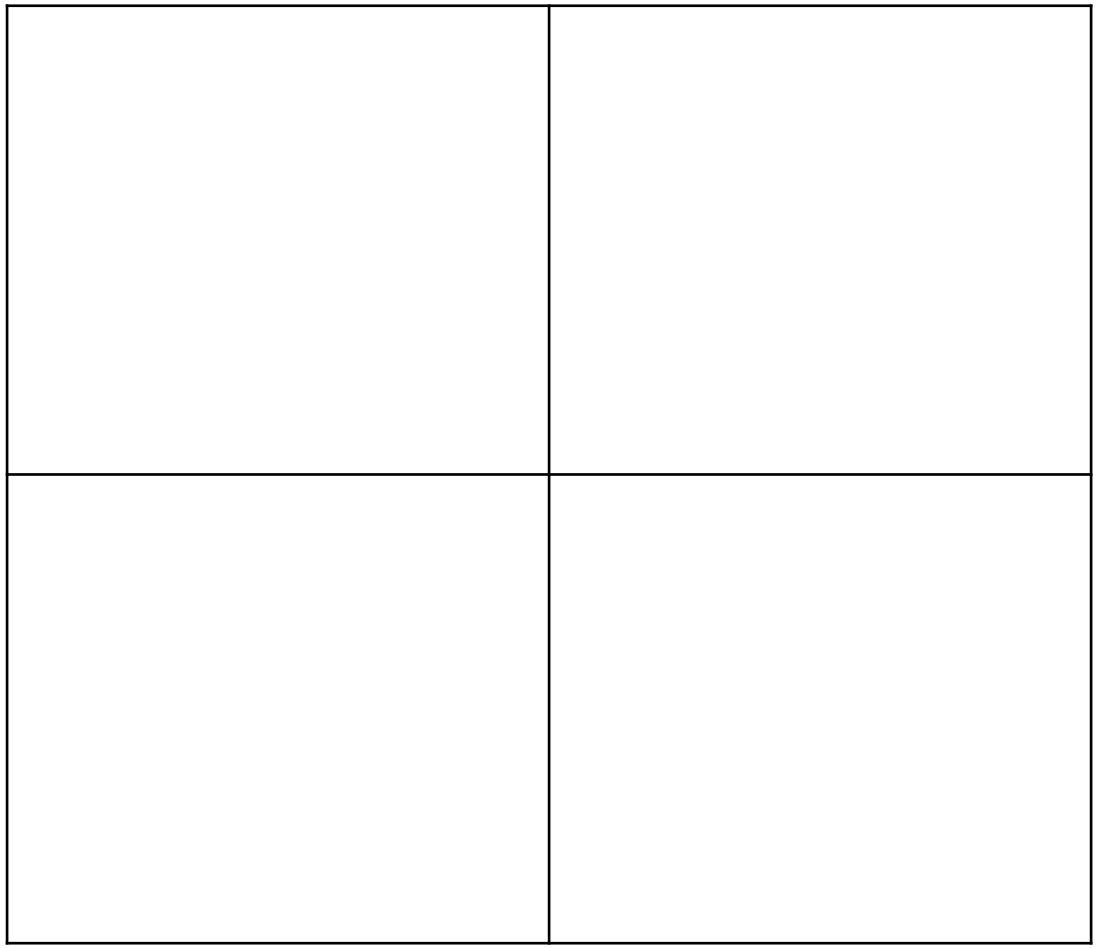


Passer par un 4×4

Celle-ci est la quatrième zone



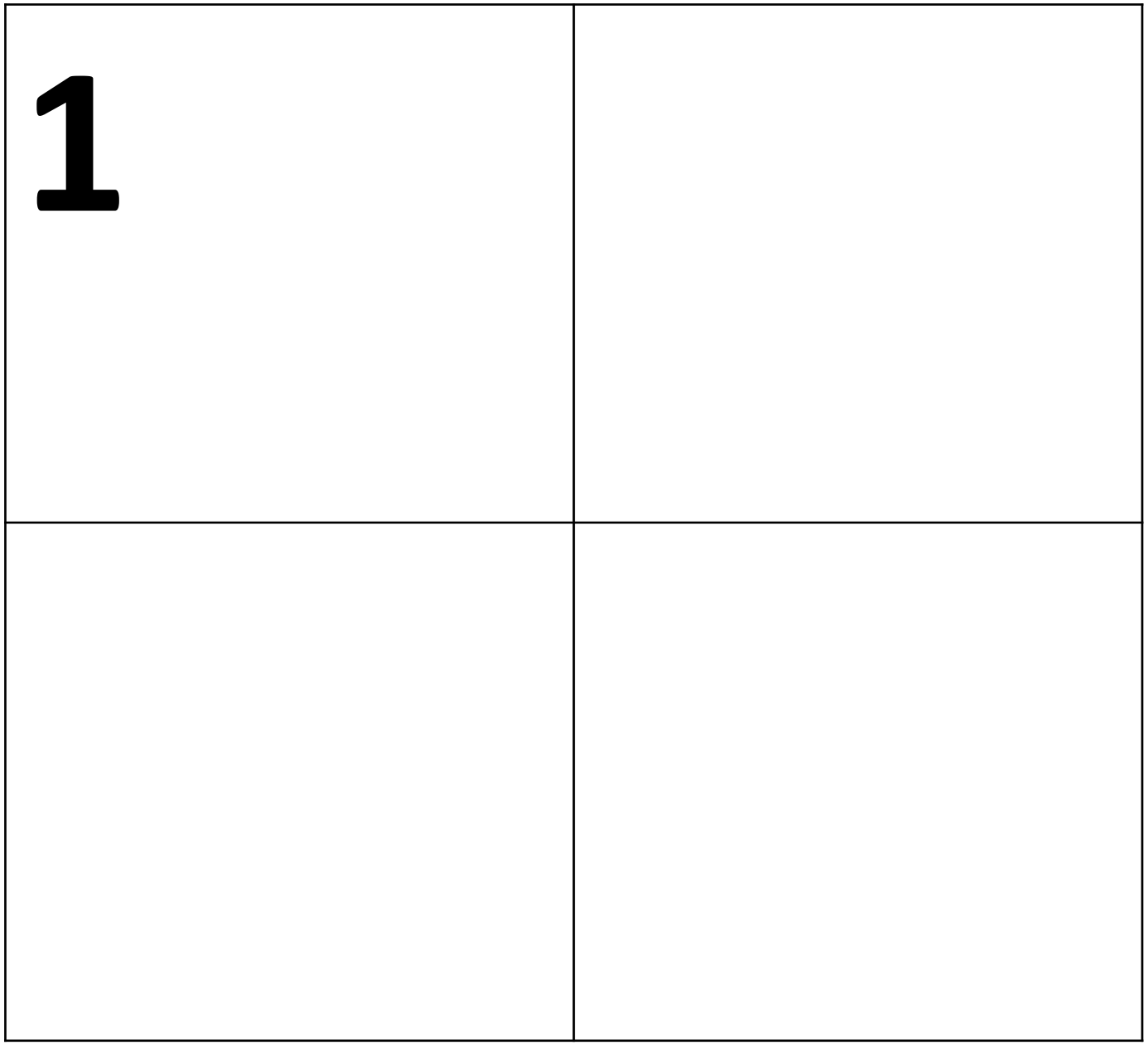
Nous avons commencé par calculer le nombre de configurations différentes dans une grille 2×2 aussi appelée une zone.



On peut voir qu'il y a quatre emplacements donc quatre configurations différentes pour placer le 1

1	**1**
1	1

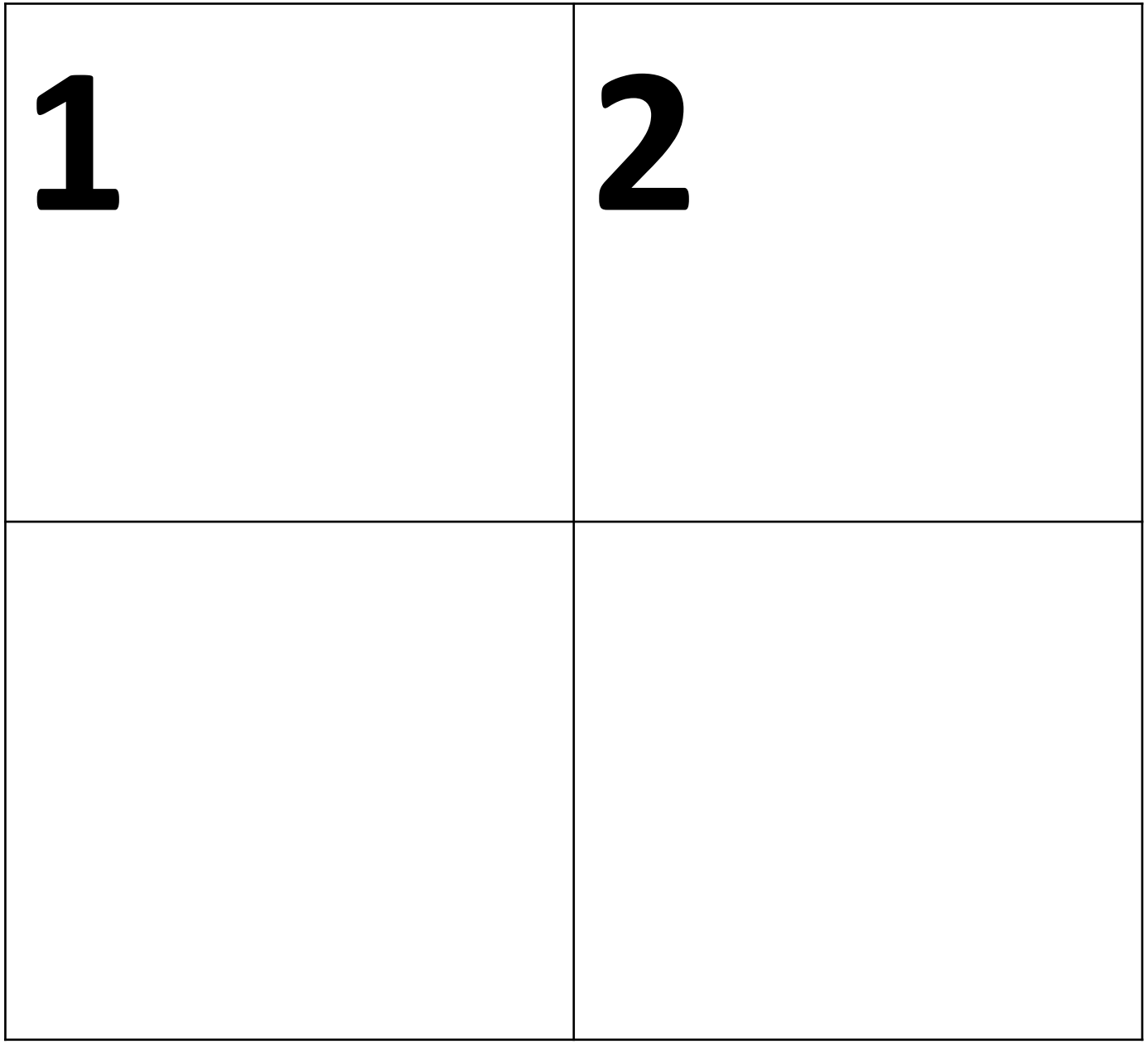
Par exemple, plaçons le 1 en haut à gauche



Il ne reste plus que
trois emplacements disponibles pour
placer le 2

1	**2**
2	2

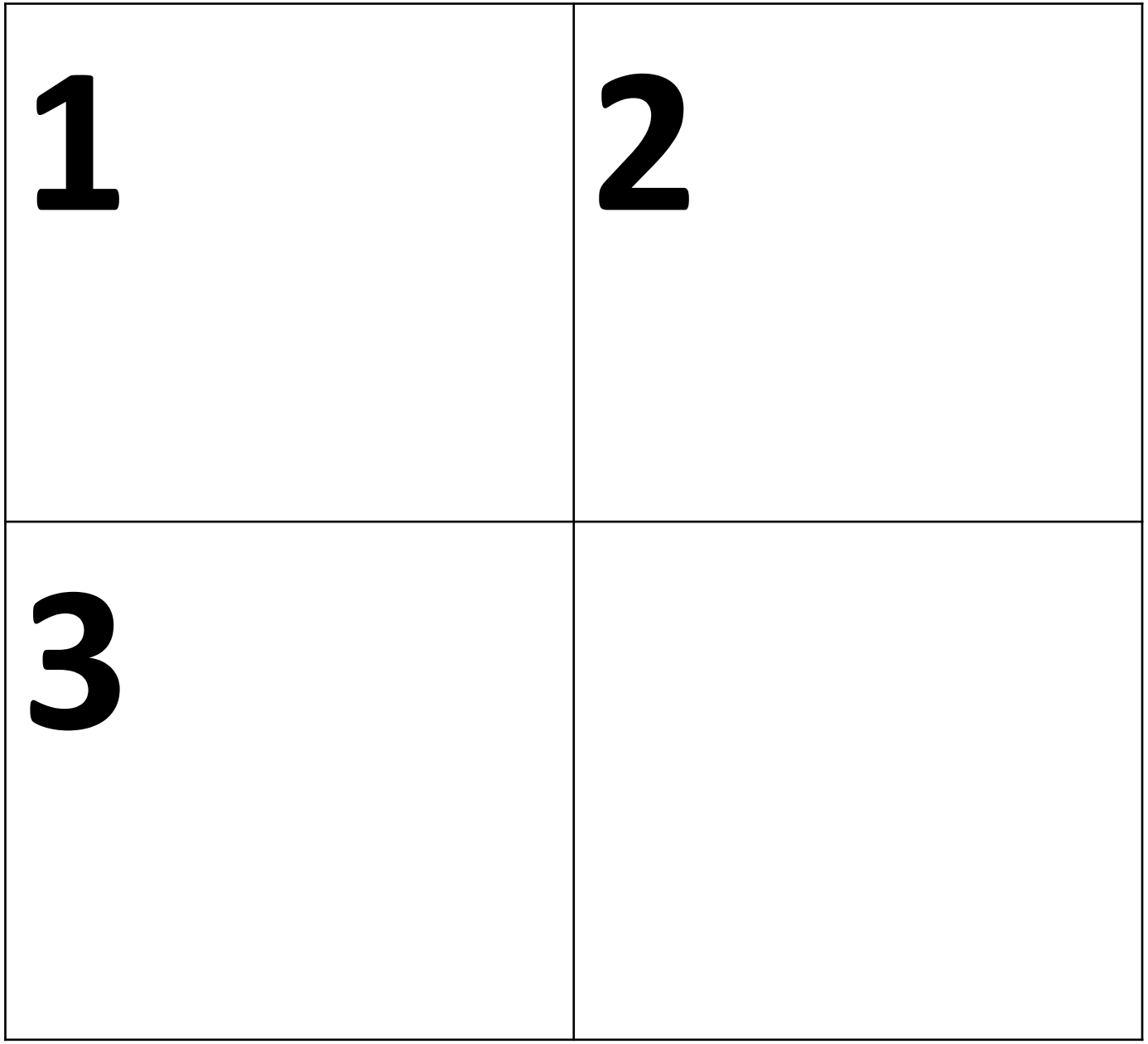
En poursuivant notre exemple, plaçons le 2 en haut à droite



Il reste alors deux emplacements pour placer le 3

1	**2**
3	3

Maintenant, plaçons le 3 en bas à gauche



Finalement, pour placer le 4, il ne reste qu'un emplacement

1	2
3	4

On procède de la manière suivante :

On peut donc dire que pour placer le premier chiffre, il y avait quatre différentes possibilités, ensuite il y a une réduction d'une possibilité à chaque fois que l'on place un nouveau chiffre dans la zone car il y a une case disponible en moins.

On note donc qu'il y a $4 \times 3 \times 2 \times 1 = 24$ différentes possibilités pour la première zone soit 4!.

Pour résumer

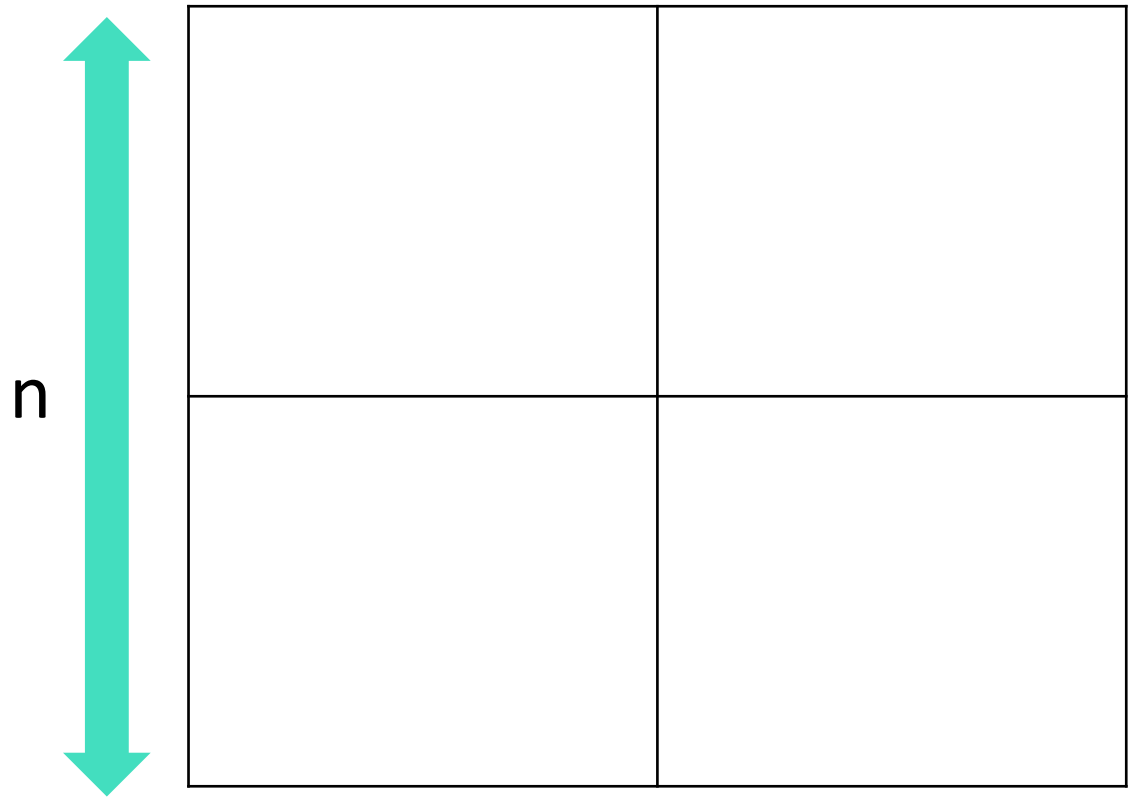
On a trouvé une formule pour calculer toutes les différentes configurations d'un carré de côté n, on peut utiliser la formule $(n^2)!$

Par exemple pour le 2×2 on applique la fomule et on trouve

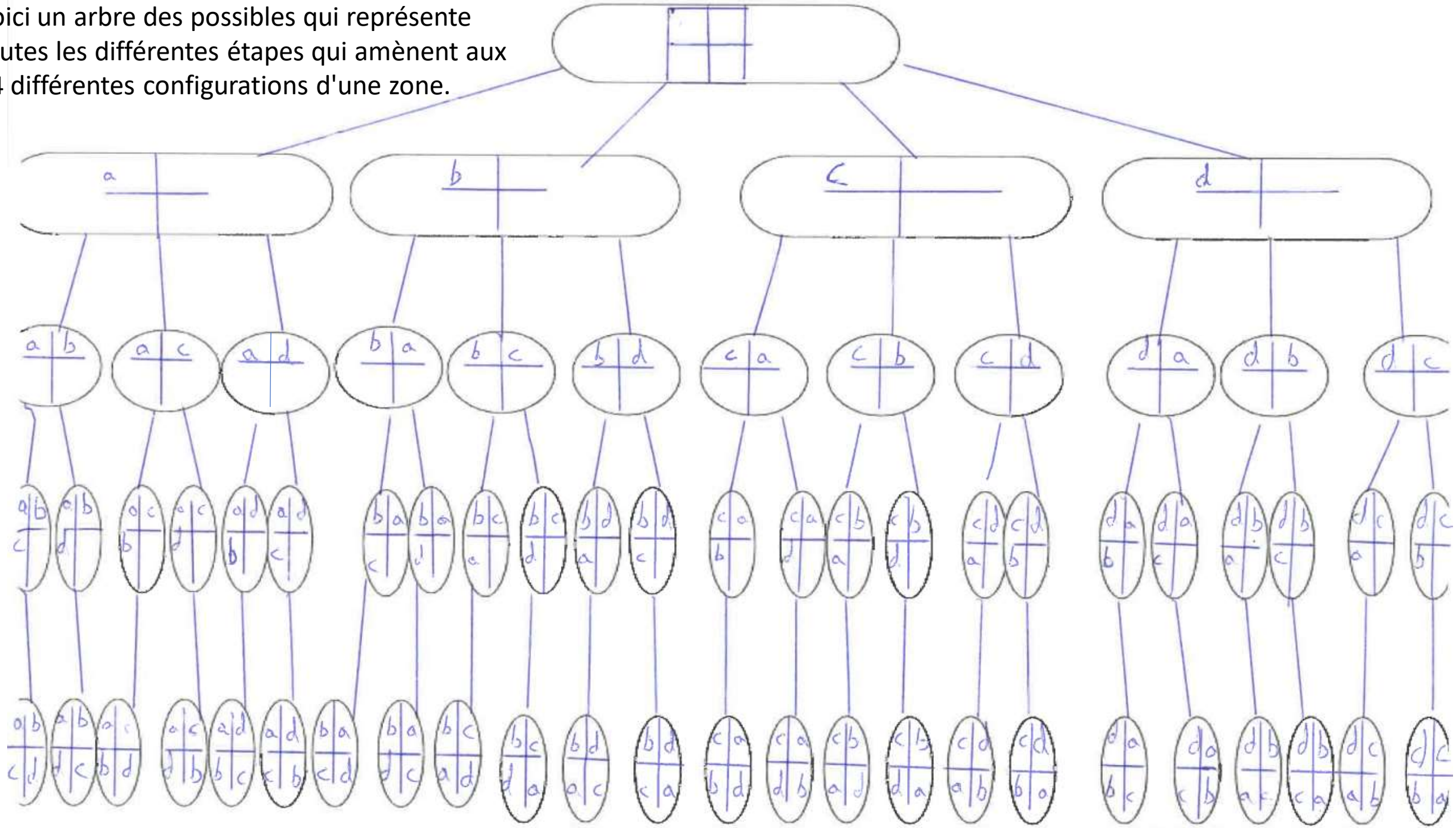
$= (2^2)!$

$= 4!$

$= 24$



Voici un arbre des possibles qui représente toutes les différentes étapes qui amènent aux 24 différentes configurations d'une zone.



On calcule ensuite les configurations pour les zones adjacentes à la première zone. Suivant les règles du Sudoku, nous pouvons voir que seuls le 3 et le 4 peuvent être présents dans les cases du dessus.

1	2	3;4	3;4
3	4	1;2	1;2

On constate donc qu'il y a 4 configurations différentes : **3;4;1;2**

1	2	3	4
3	4	1	2

4,3,1,2

1	2	4	3
3	4	1	2

4,3,2,1

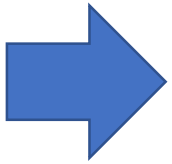
1	2	4	3
3	4	2	1

Et 3,4,2,1

1	2	3	4
3	4	2	1

A gauche, on voit la première zone et à droite, on a représenté les différentes configurations pour une zone adjacente.

1	2
3	4



case adjacente : 4 possibilités

3	4
1	2

3	4
2	1

4	3
1	2

4	3
2	1

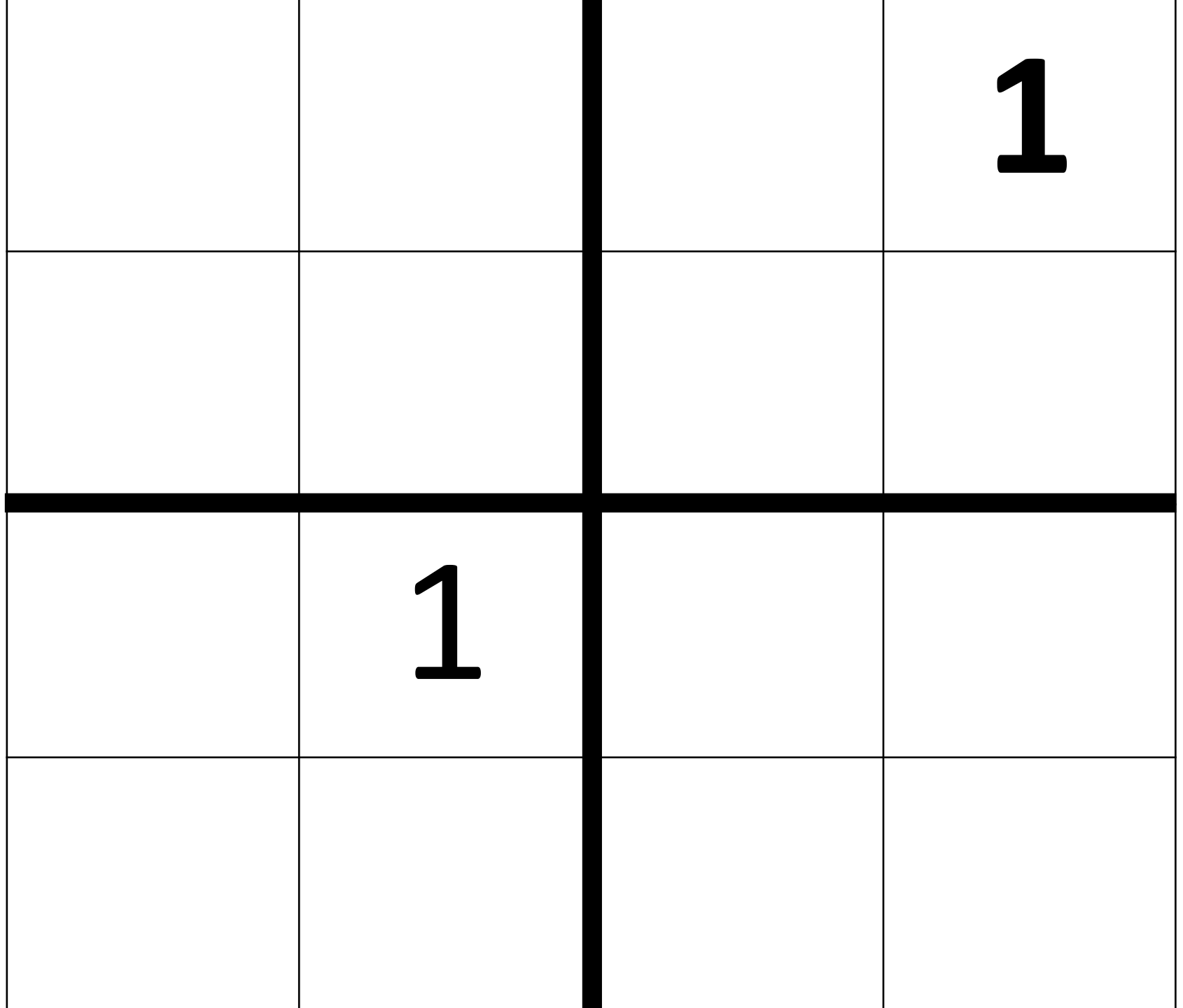
On voit mieux les quatre différentes configurations ici.(2)

On sait donc qu'il y a quatre différentes configurations pour la zone 2, pour chaque configuration de la zone 1.

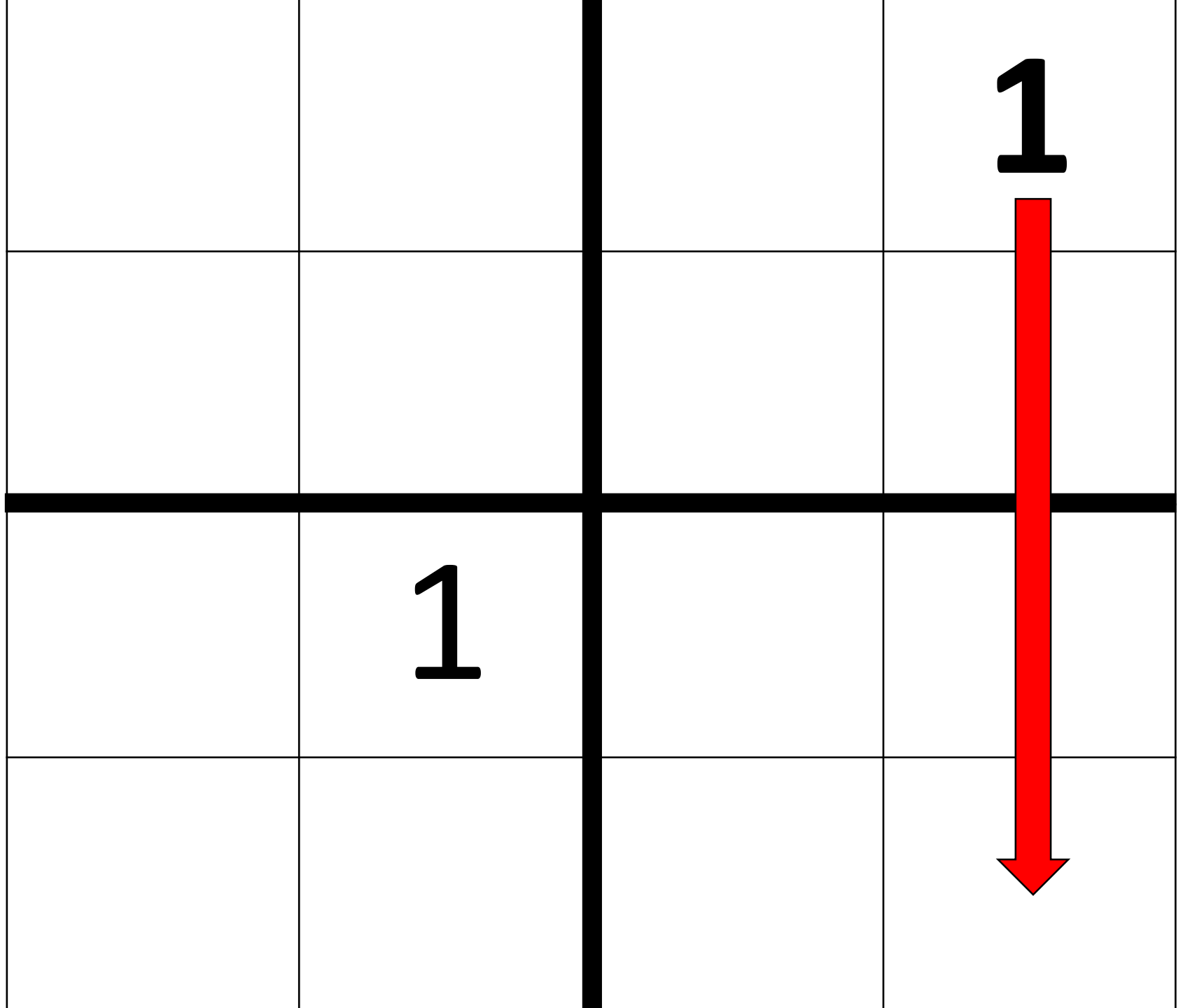
Il y a donc 24×4 configurations si on regroupe les deux premières zones, soit 96.

La deuxième zone n'a aucune relation avec la 3ème zone (en diagonale), donc on peut simplement multiplier à nouveau par 4 pour avoir toutes les configurations des trois premières zones, soit 384.

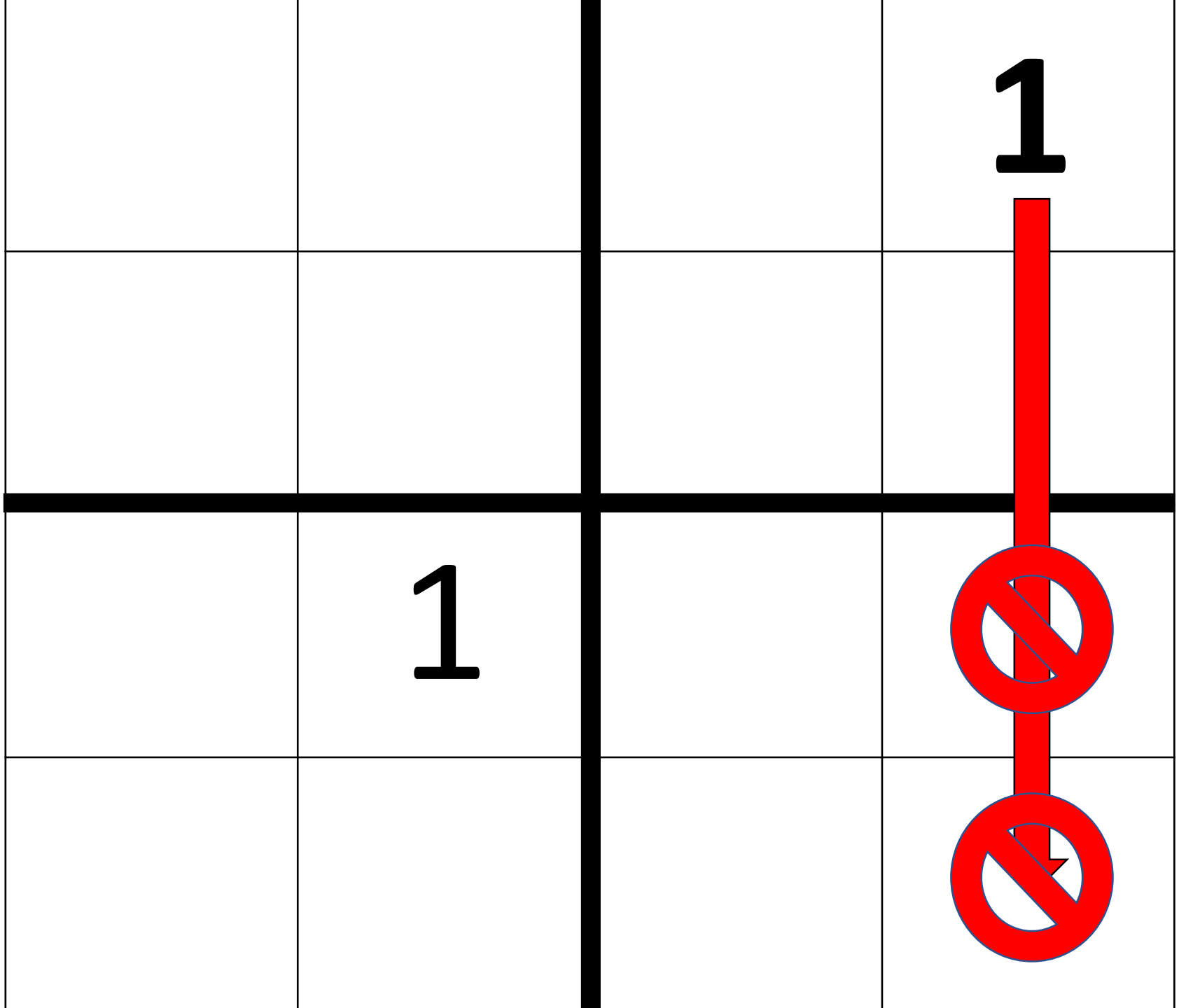
Nous allons maintenant montrer que la dernière zone n'augmente pas le nombre de possibilités.



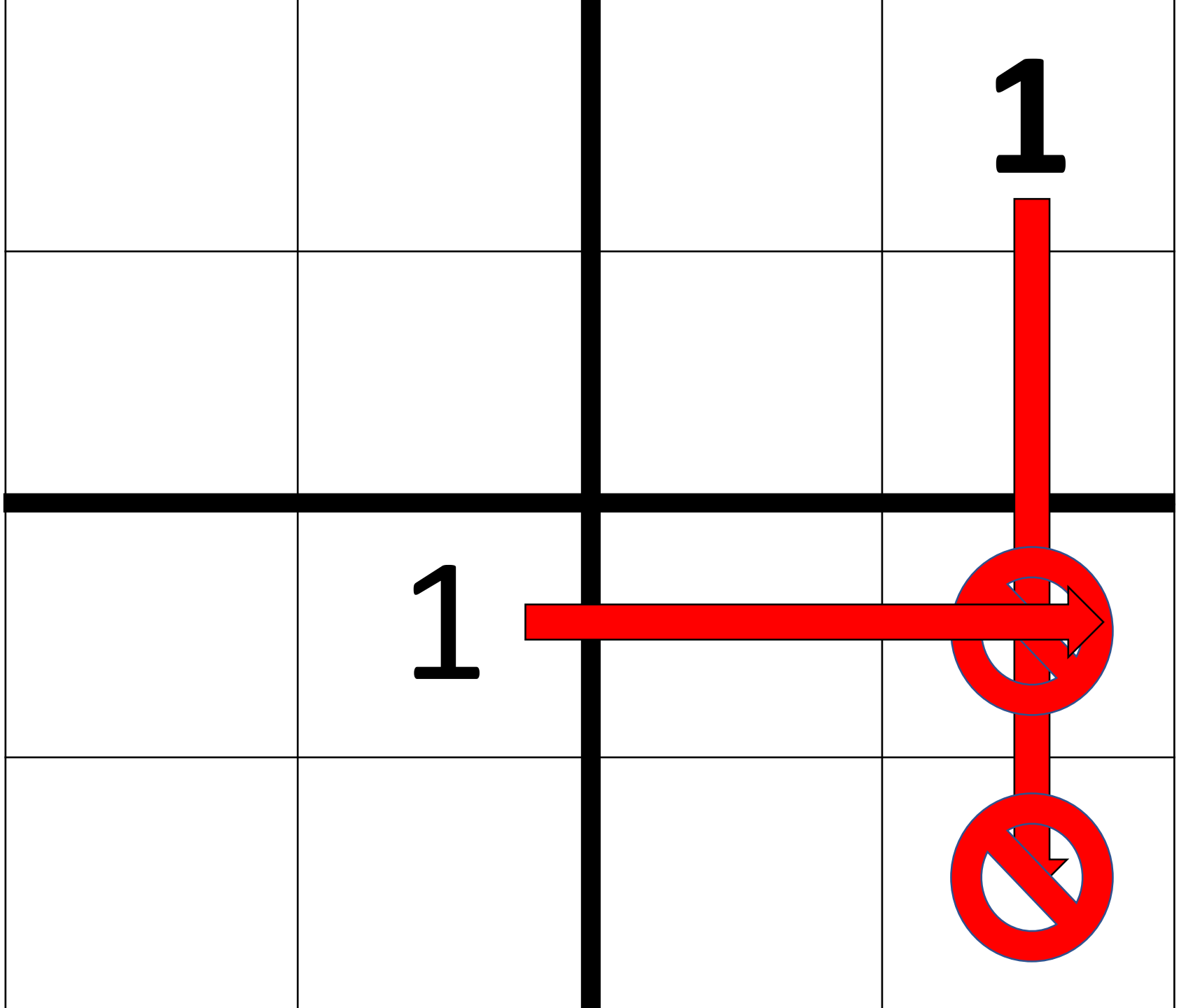
Ici nous avons placé aléatoirement deux 1 dans les zones 2 et 3.

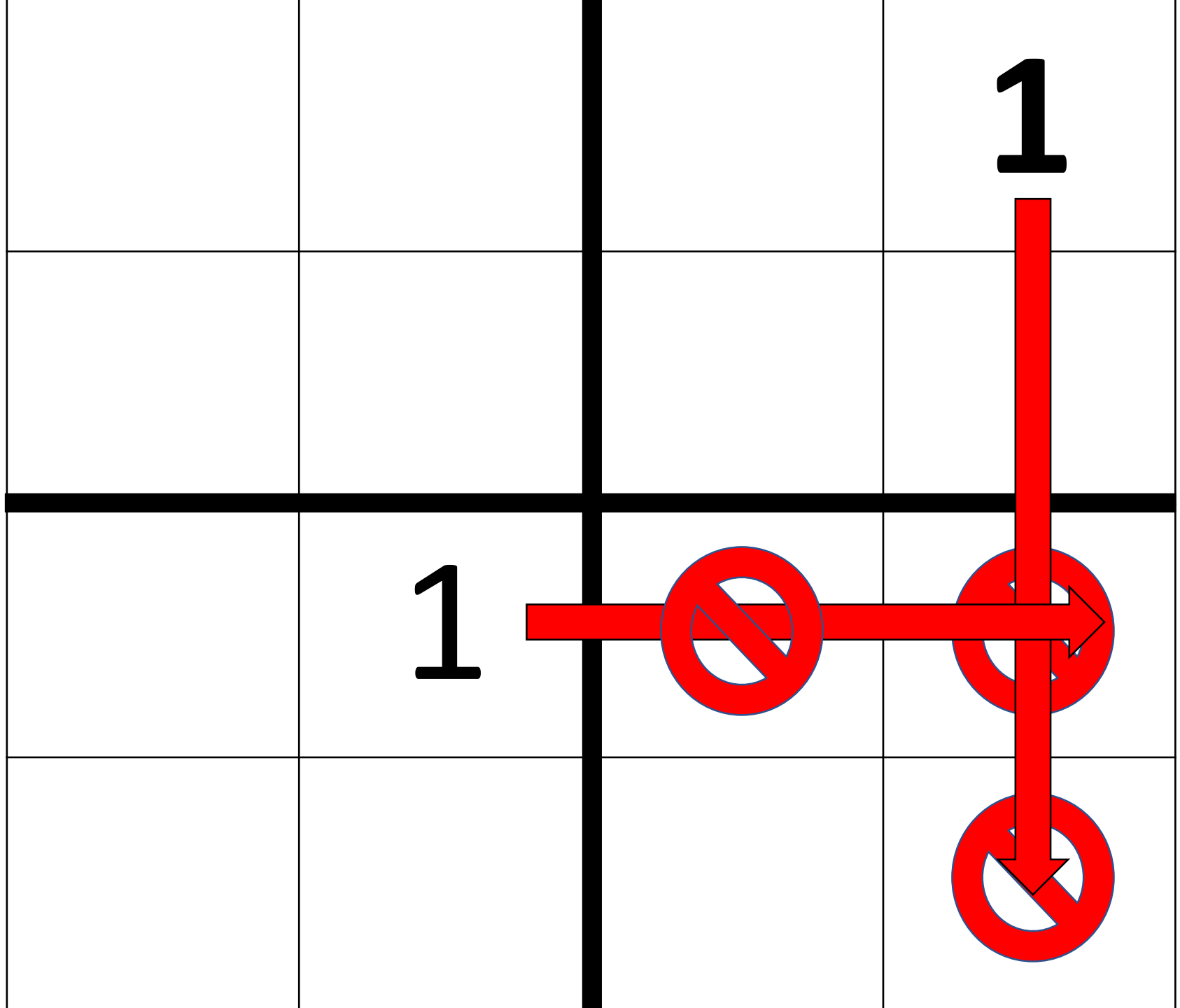


Le 1 de la zone 2 obstrue la colonne dans laquelle il est placé.

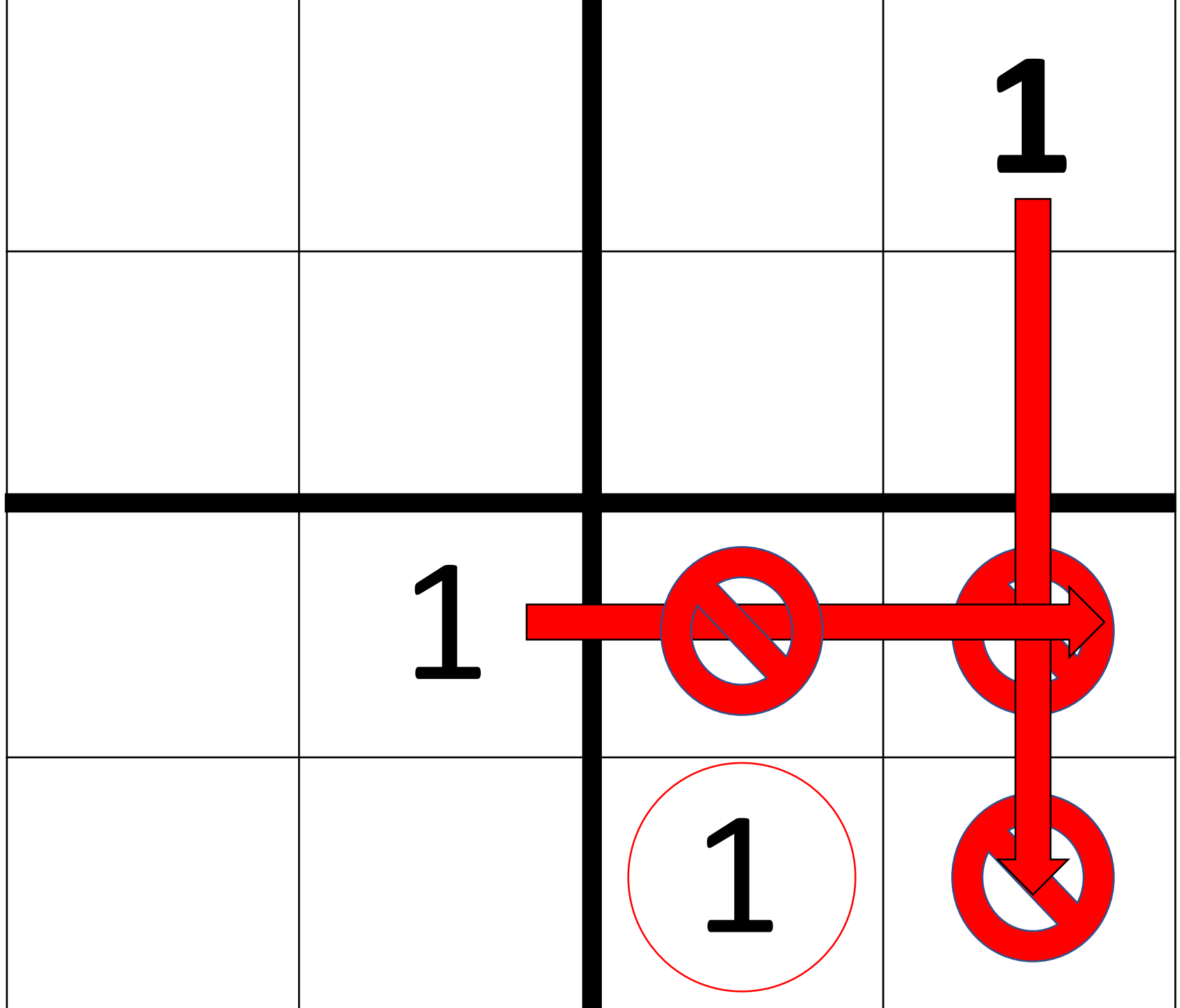


Et le 1 de la zone 3 obstrue la ligne dans laquelle il est placé.

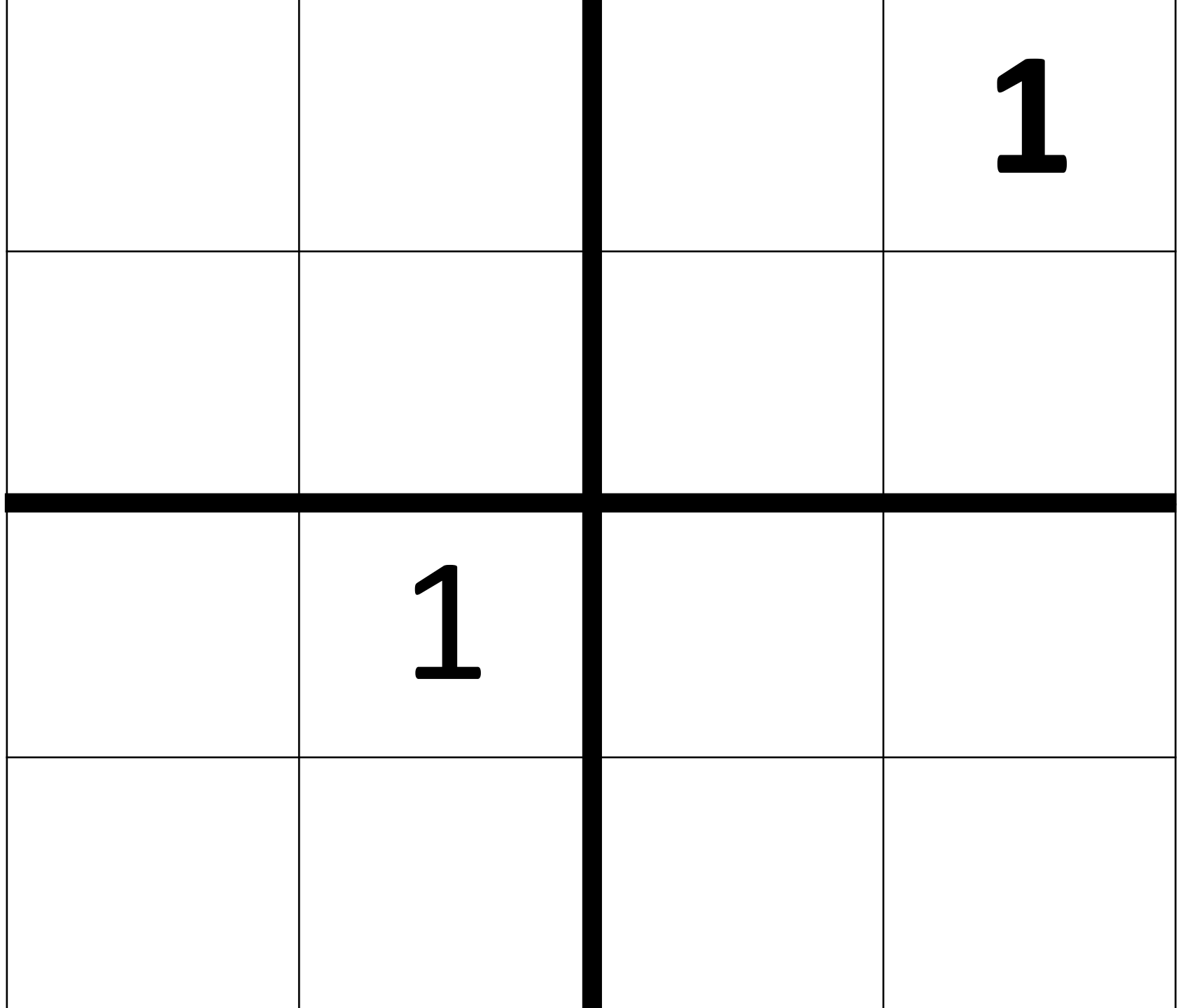


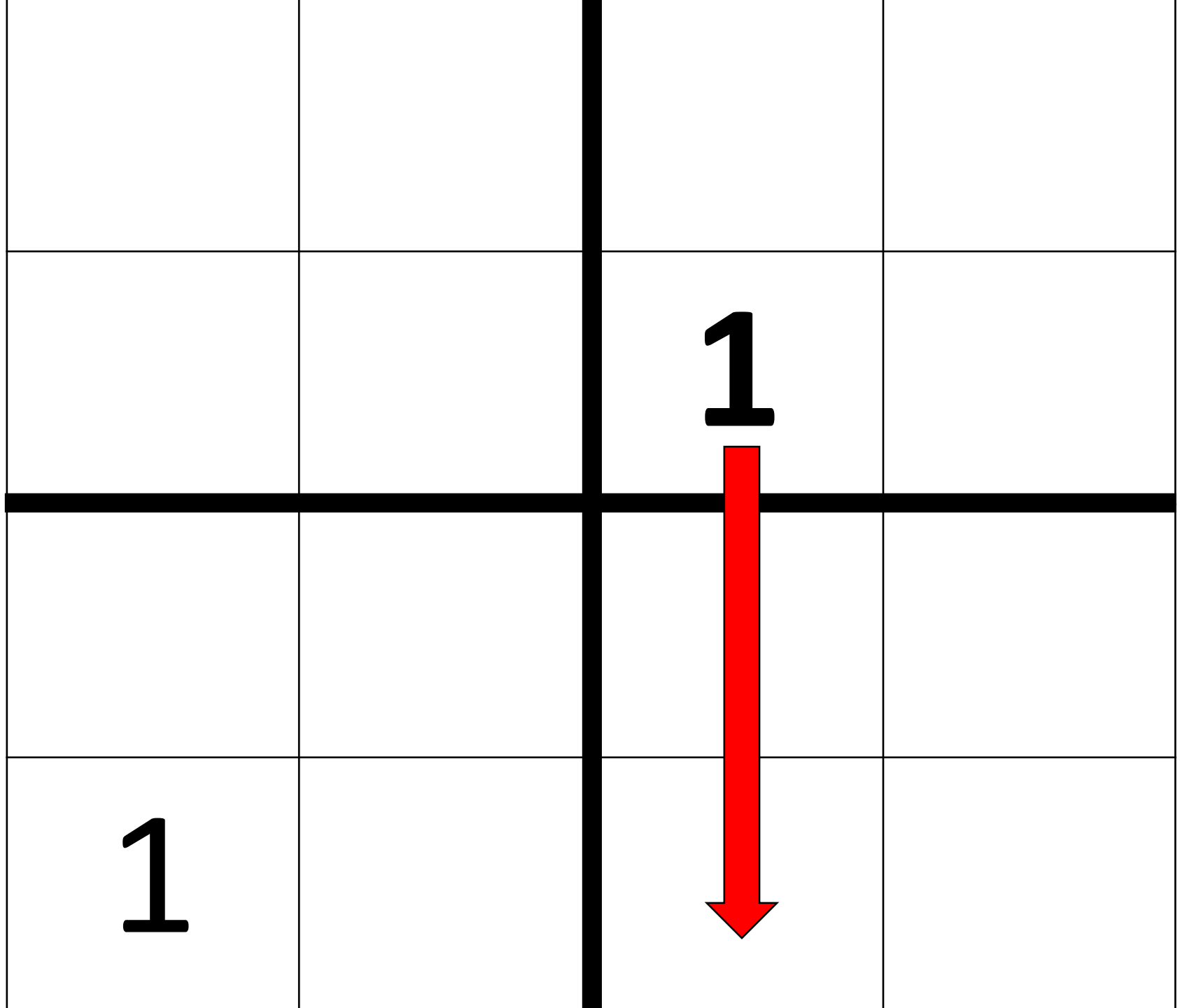

1
1

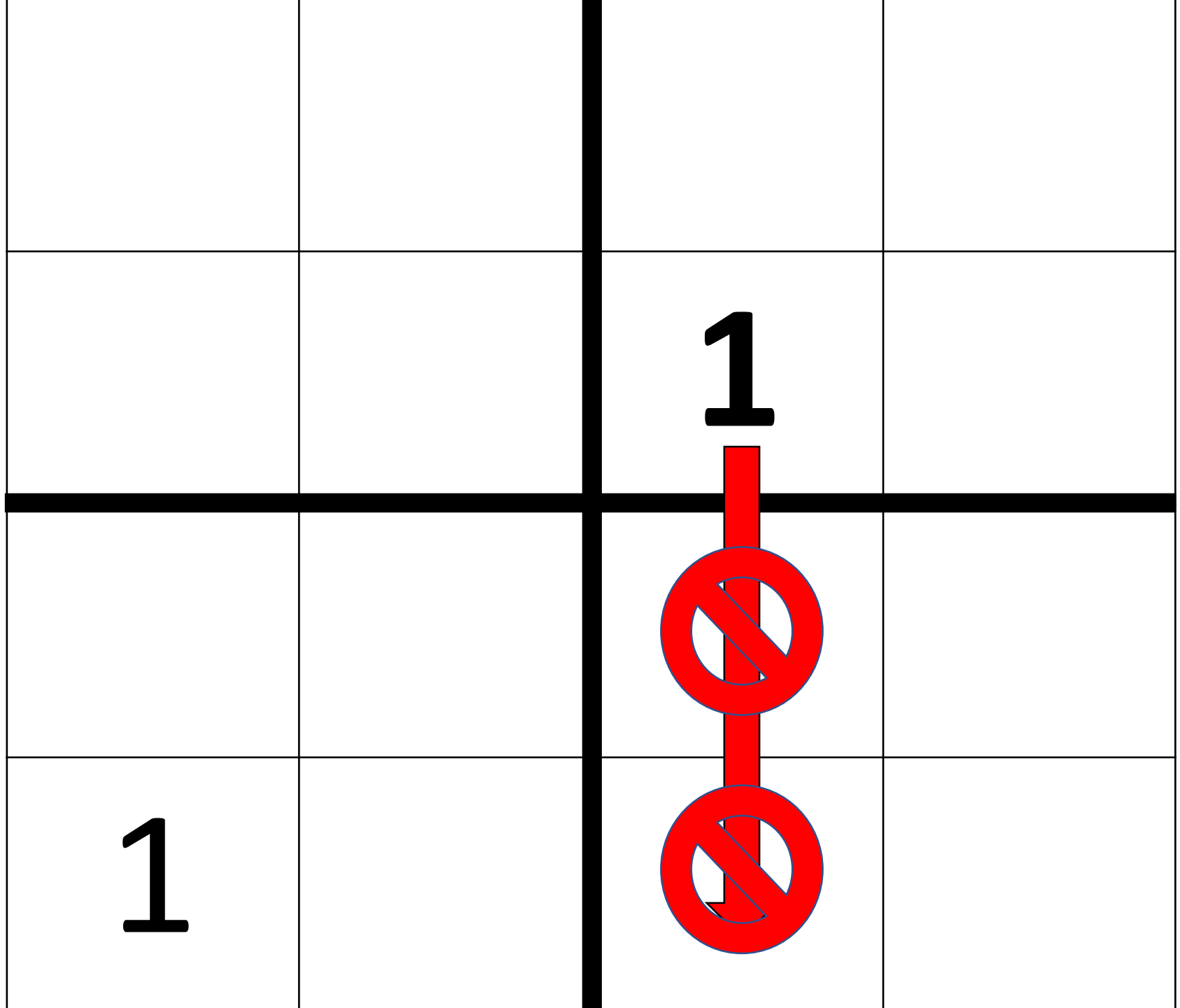
On voit que quand deux mêmes chiffres sont placés dans les zones 2 et 3, cela ne laisse qu'un seul emplacement de libre pour placer ce même chiffre dans la zone 4.

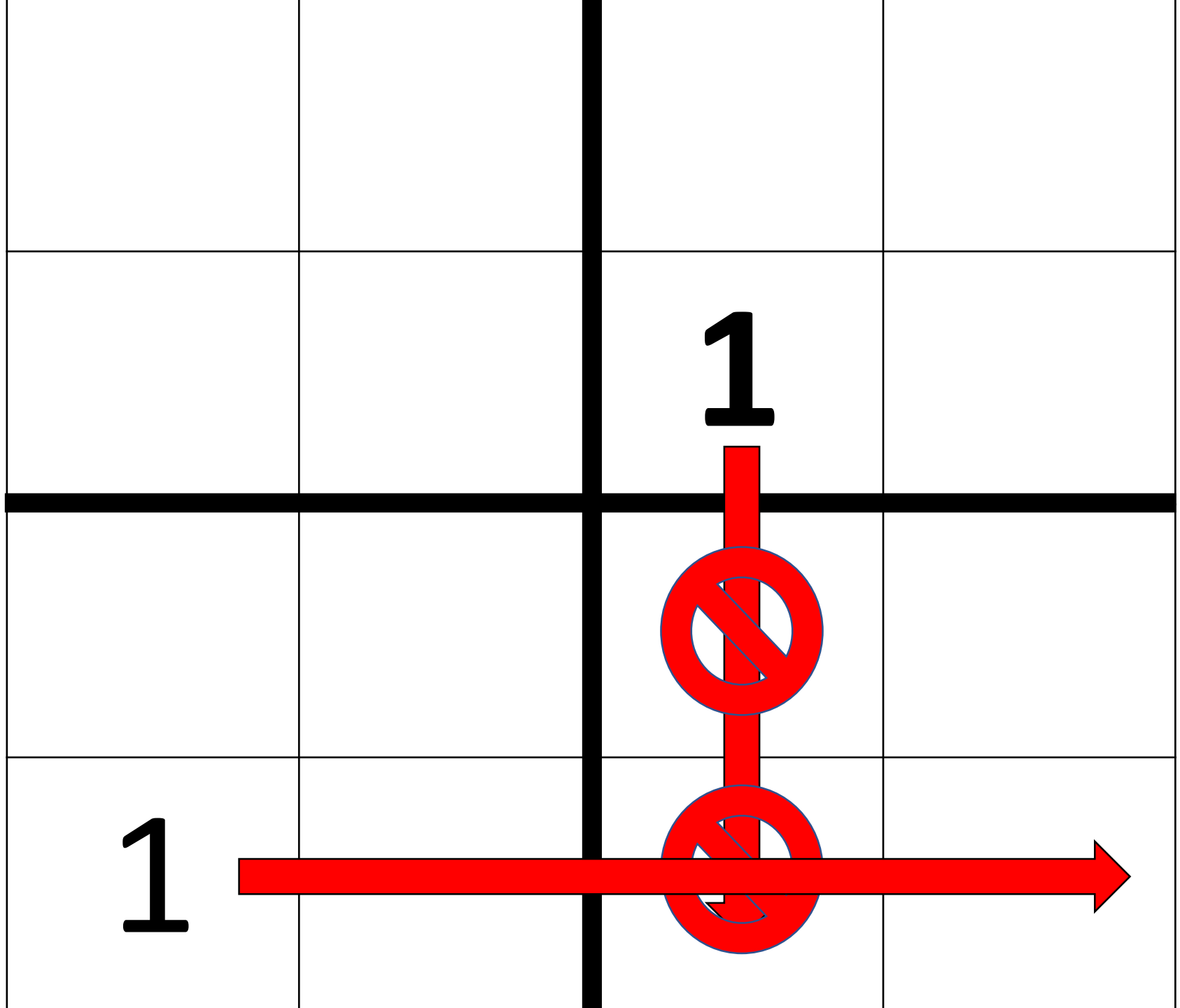


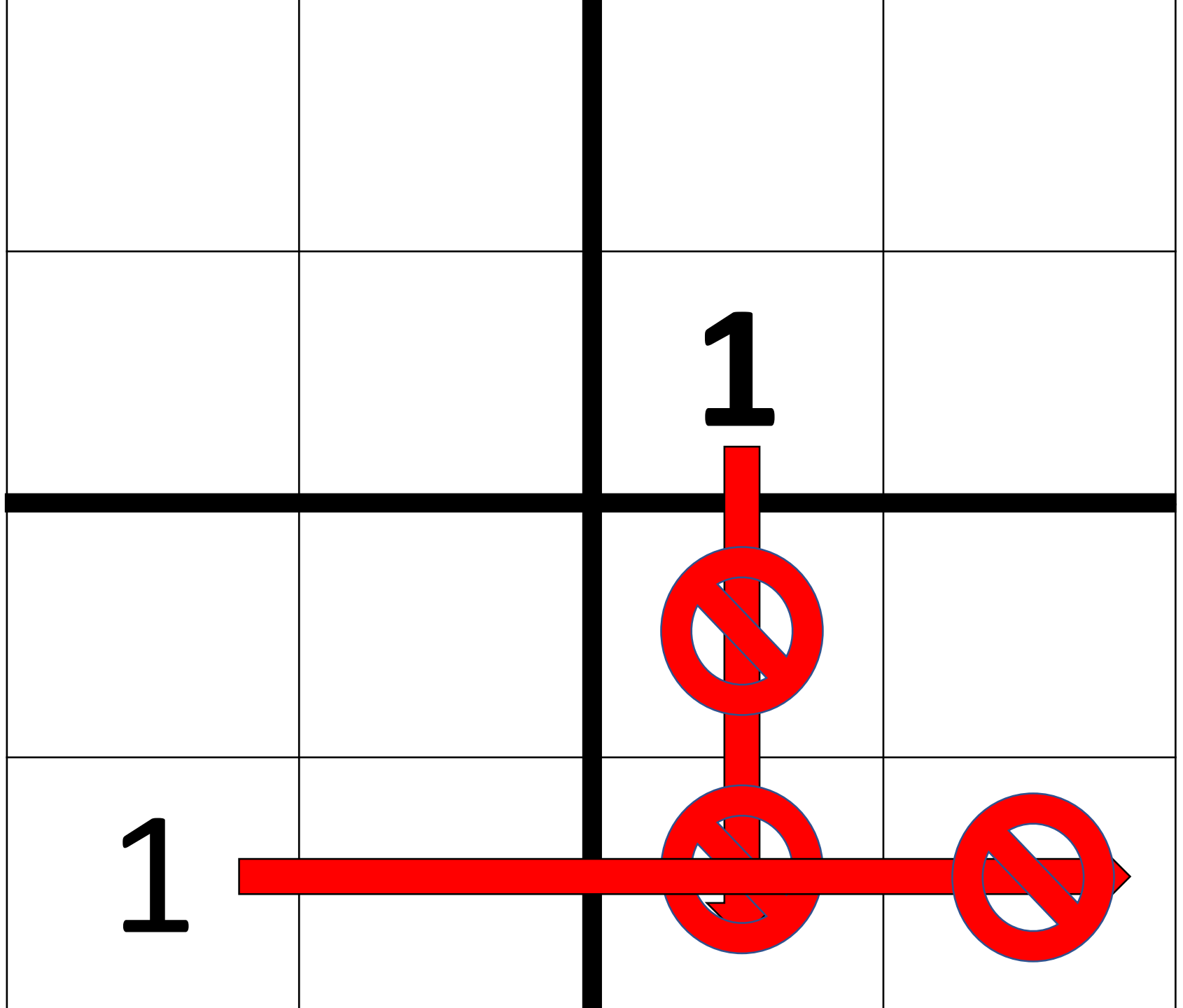
Ici, nous avons suivi la même démarche avec des placements différents.

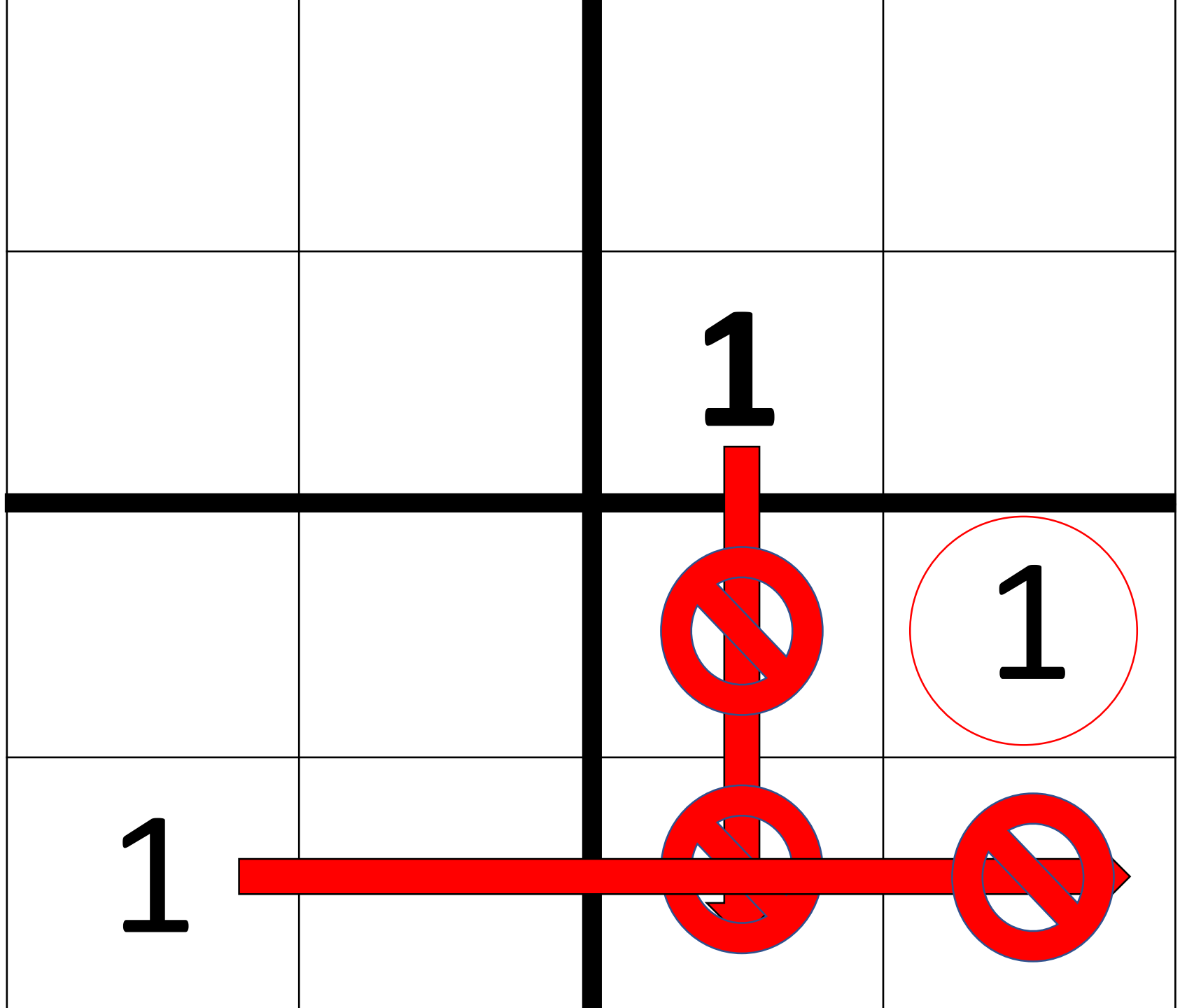



1
1


1
1


1
1


1
1


1
1
1

Conclusion

En conclusion, la 4ème zone est déjà déterminée lorsque les trois premières le sont. Elle n'augmente donc pas les configurations possibles.

Le nombre de configurations possibles pour les quatre zones est donc le même que celui des trois premières zones qui est 384.

Partie II

Nous nous sommes intéressés à une représentation avec Python du Sudoku, menant à sa résolution algorithmique.

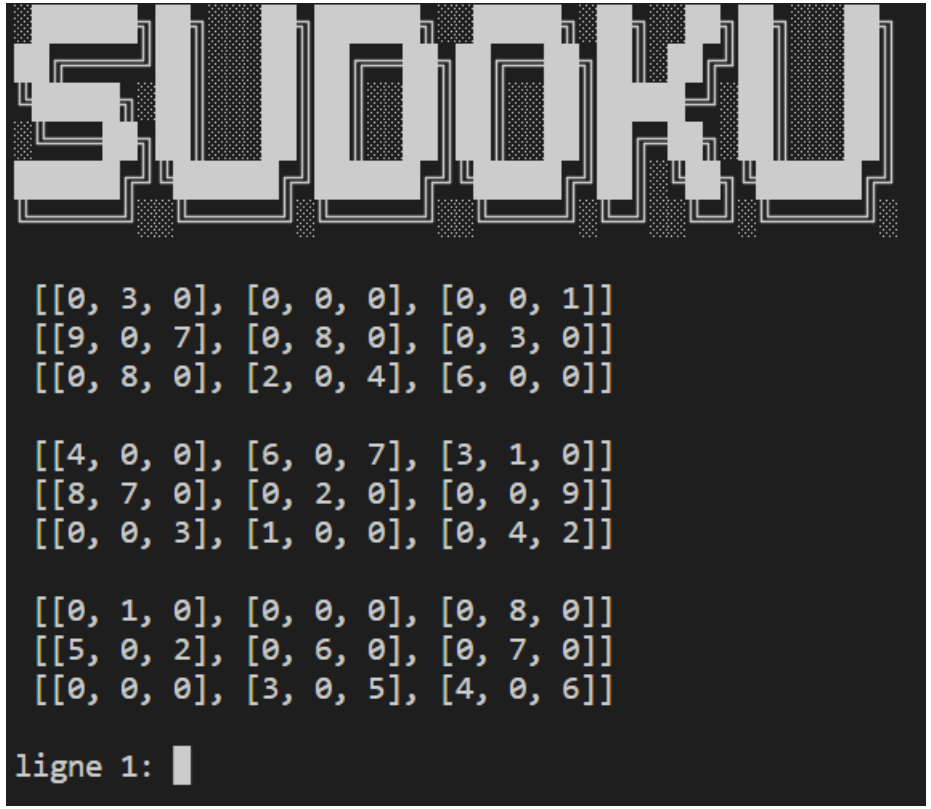
Grille de sudoku dans la console Python (Liste)

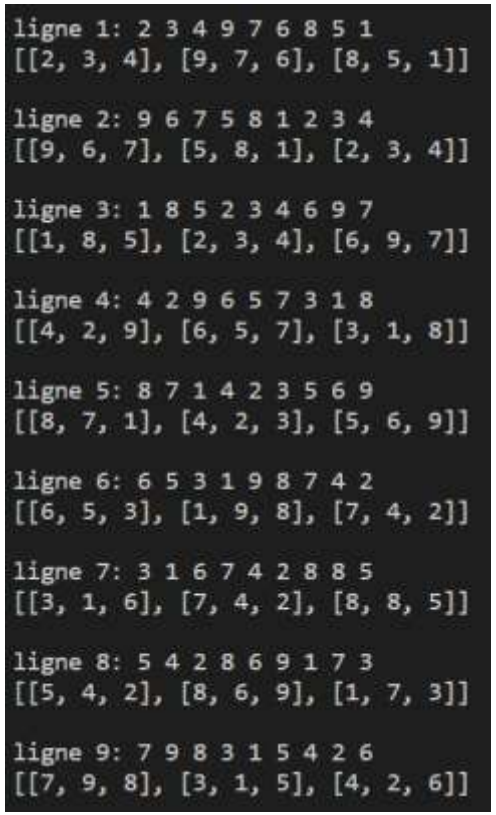
```
a = [[0,3,0], [0,0,0], [0,0,1]]
b = [[9,0,7], [0,8,0], [0,3,0]]
c = [[0,8,0], [2,0,4], [6,0,0]]
d = [[4,0,0], [6,0,7], [3,1,0]]
e = [[8,7,0], [0,2,0], [0,0,9]]
f = [[0,0,3], [1,0,0], [0,4,2]]
g = [[0,1,0], [0,0,0], [0,8,0]]
h = [[5,0,2], [0,6,0], [0,7,0]]
i = [[0,0,0], [3,0,5], [4,0,6]]
```

```
ax = [[2,3,4], [9,7,6], [8,5,1]]
bx = [[9,6,7], [5,8,1], [2,3,4]]
cx = [[1,8,5], [2,3,4], [6,9,7]]
dx = [[4,2,9], [6,5,7], [3,1,8]]
ex = [[8,7,1], [4,2,3], [5,6,9]]
fx = [[6,5,3], [1,9,8], [7,4,2]]
gx = [[3,1,6], [7,4,2], [8,8,5]]
hx = [[5,4,2], [8,6,9], [1,7,3]]
ix = [[7,9,8], [3,1,5], [4,2,6]]
```

On a representé la grille de Sudoku en utilisant des listes, chacune composée de trois sous-listes afin de différencier les zones du Sudoku.

Grille de sudoku dans la console Python (Liste)





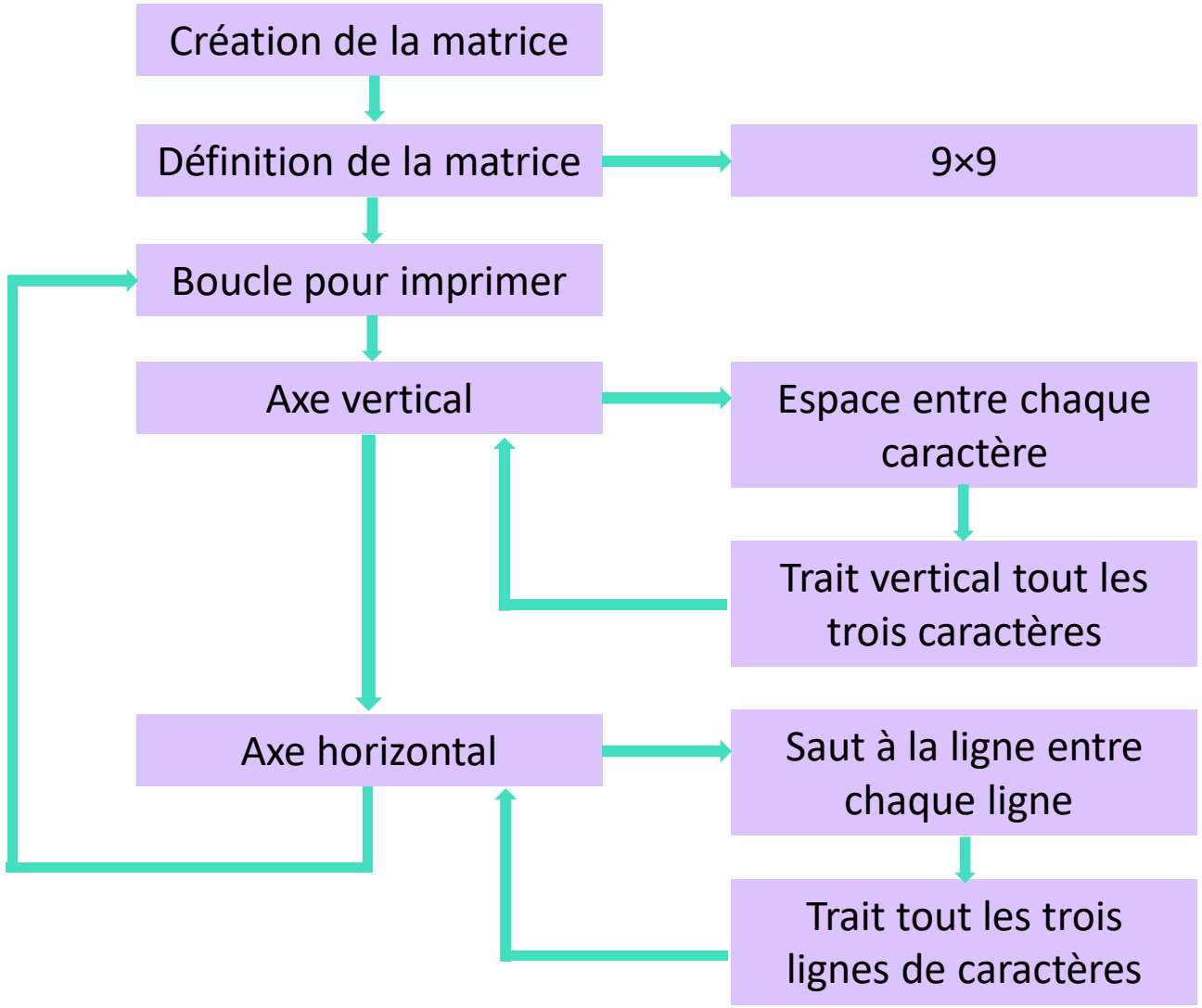
Console affiche:
"Vous avez bien résolu le sudoku!"

Ce premier programme a pour but de vérifier les résultats d'une grille de Sudoku que nous avons nous-même résolue sur papier. Il les compare à une grille corrigée. Cela fonctionne de la manière suivante:

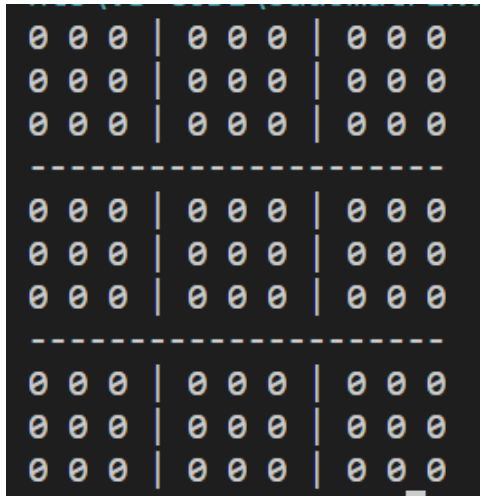
- Le programme imprime ligne par ligne la grille dans la console
- Il nous demande nos résultats pour le Sudoku
- On entre manuellement les résultats de chaque ligne dans la console
- Le programme compare les valeurs qu'on a entrées dans la console aux lignes corrigées
- Si toutes les lignes sont justes, il écrit dans la console: "Vous avez bien résolu le Sudoku!"(3)

Nous avons essayé de faire un progamme pour représenter le Sudoku à l'aide de matrices. Nous avons représenté le programme avec un organigramme:

Grille de sudoku dans la console Python (Matrice)



Résultat dans la console:



- Le programme crée la matrice et configure la matrice 9×9.
- Ensuite, une boucle pour imprimer la matrice fait en sorte que sur l'axe vertical, il y a un espace entre chaque caractère et un trait vertical tout les trois caractères. Sur l'axe horizontal, il fait un saut à la ligne entre chaque ligne et une ligne de trait toutes les trois lignes.

Malheureusement, en raison de nos lacunes dans le domaine des matrices, nous n'avons pas réussi à les utiliser de manière optimale.

En effet, nous avions espéré pouvoir sélectionner chaque valeur de la matrice et la changer pour y mettre les valeurs d'une grille de Sudoku.

Finalement, nous avons découvert le module Python Pygame :

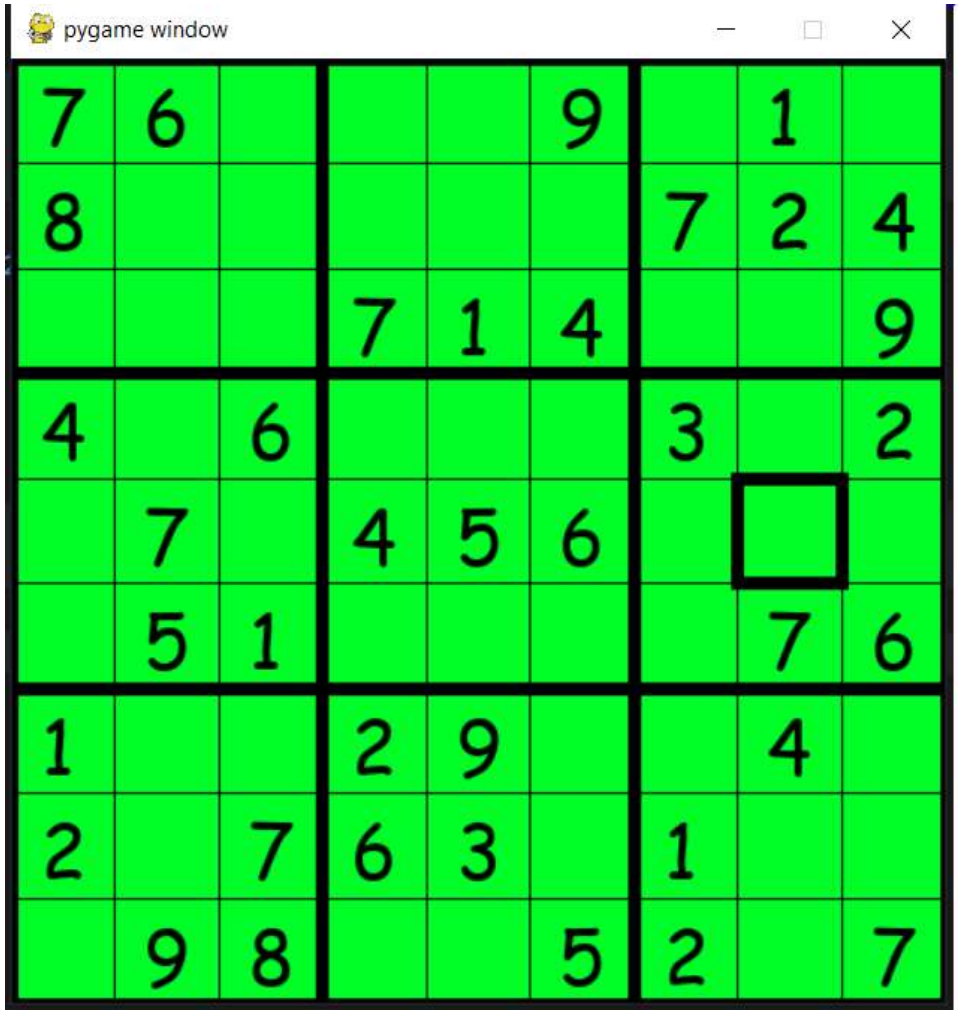
un module permettant de représenter graphiquement des interfaces interactives.

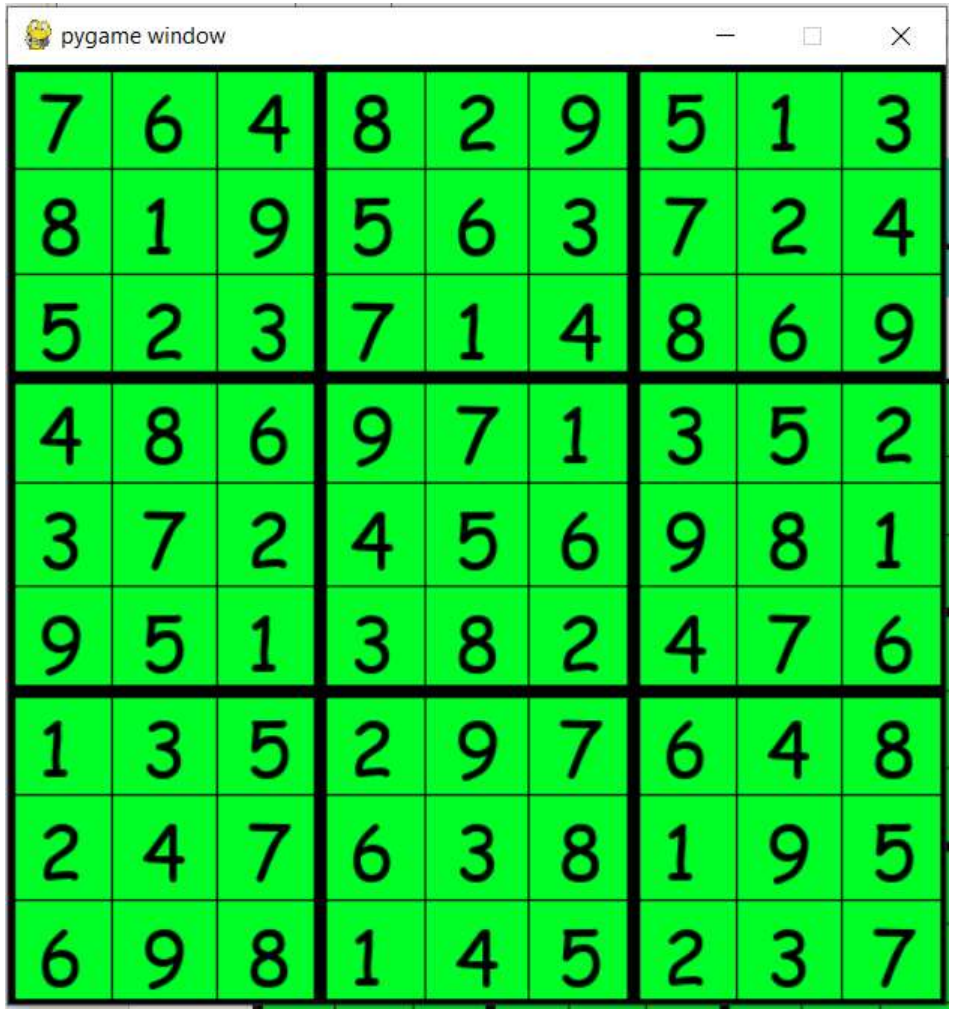
Le résultat : un programme qui affiche une grille de Sudoku interactive avec les valeurs initiales de la grille.

L'utilisateur peut donc remplir le Sudoku directement en sélectionnant la case, soit en cliquant dessus, soit en utilisant les flèches directionnelles.

Cependant, le programme n'est pas tout à fait au point : il ne vérifie pas que les règles fondamentales sont respectées.

Grille de Sudoku avec Pygame





Algorithme pour résoudre un Sudoku : *Backtracking*

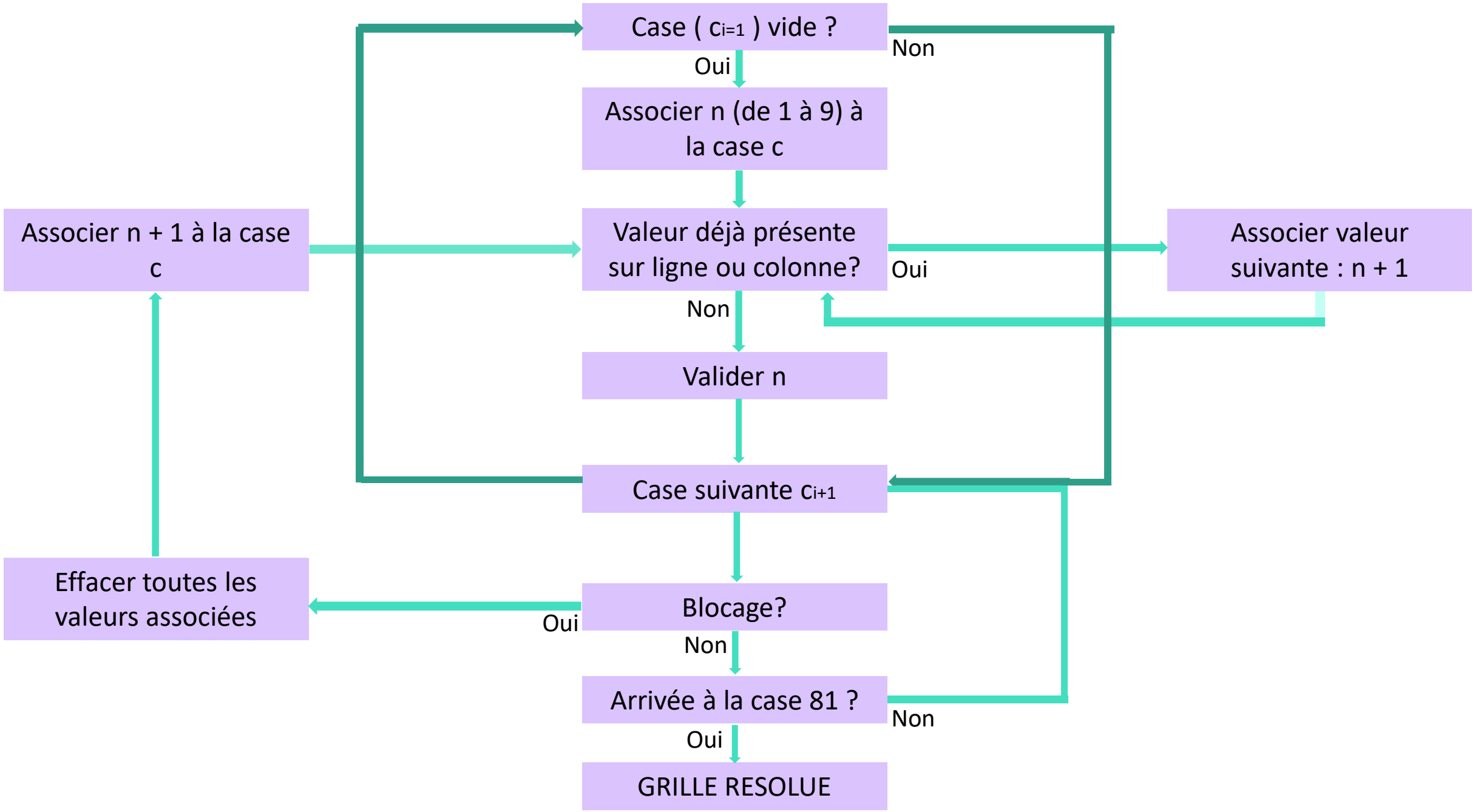
Nous avons cherché à résoudre le Sudoku par un algorithme. Celui-ci se base sur le principe du *Backtracking*. C'est une catégorie d'algorithme qui explore toutes les possibilités pour résoudre un problème. Autrement dit, il va essayer de résoudre la grille en essayant tous les chiffres possibles, toutes les configurations possibles, jusqu'à en trouver une qui respecte les règles du jeu.

Algorithme : Principales étapes

Pour faire cela, nous avons procédé en plusieurs grandes étapes :

- la représentation de la grille
- définir des conditions, suivant les règles du Sudoku
- créer une fonction qui suit le principe du *Backtracking*

Nous avons créé un organigramme afin de représenter le code de manière synthétique.


Case (ci=1) vide ?
Non
Oui
Associer n (de 1 à 9) à la case c
Associer n + 1 à la case c
Valeur déjà présente sur ligne ou colonne?
Oui
Associer valeur suivante : n + 1
Non
Valider n
Case suivante ci+1
Effacer toutes les valeurs associées
Blocage?
Oui
Non
Arrivée à la case 81 ?
Non
Oui
GRILLE RESOLUE

Ordre des cases:

La première case est représentée par $C_{i=1}$ soit C1, puis l'on ajoute la valeur de 1 à chaque case et ainsi de suite. Donc par exemple, la deuxième case est C1+1 = C2 et la troisième est C2+1=C3.

C1	C1+1	C+3	C+4	C+5	C+6	C+7	C+8	C+9
C28	C29	C30	C31	C32	C33	C34	C35	C36
C55	C56	C57	C58	C59	C60	C61	C62	C63
								C81

Algorithme pour résoudre un Sudoku : *Backtracking*

L'algorithme pour résoudre la grille :

- chercher une case vide
- essayer un nombre n (de 1 à 9) sur la case en suivant les conditions; vérifier si le nombre n'est pas déjà sur la ligne ou la colonne
- faire de même pour la case suivante

→ Condition de blocage : il est important de comprendre que l'algorithme remplit les nombres possibles dans l'ordre croissant. Si sur une case, il peut y avoir le 2 ou le 8 par exemple, alors l'algorithme associe le 2 car 2<8.
Néanmoins, le 8 peut correspondre à cette case. Donc avec le 2, la grille est fausse et l'on constatera un "blocage" donc une situation où la grille ne respecte plus les règles du jeu.
→ L'algorithme prend cela en compte, c'est le principe du *Backtracking*. En effet, si l'on arrive à une situation telle que celle-ci, il retournera à la case fautive et changera le 2 en 8(4).

Résultat dans la console

Grille initiale

```
7 8 0|4 0 0|1 2 0
6 0 0|0 7 5|0 0 9
0 0 0|6 0 1|0 7 8
-----------------
0 0 7|0 4 0|2 6 0
0 0 1|0 5 0|9 3 0
9 0 4|0 6 0|0 0 5
-----------------
0 7 0|3 0 0|0 1 2
1 2 0|0 0 7|4 0 0
0 4 9|2 0 6|0 0 7
```


Grille intermédiaire

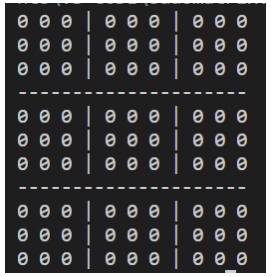
```
7 8 5|4 3 9|1 2 6
6 1 2|8 7 5|3 4 9
4 9 3|6 2 1|5 7 8
-----------------
8 5 7|1 4 0|2 6 0
0 0 1|0 5 0|9 3 0
9 0 4|0 6 0|0 0 5
-----------------
0 7 0|3 0 0|0 1 2
1 2 0|0 0 7|4 0 0
0 4 9|2 0 6|0 0 7
```

Résultat final

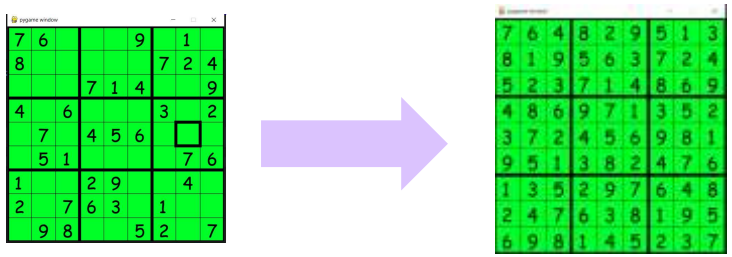
```
7 8 5|4 3 9|1 2 6
6 1 2|8 7 5|3 4 9
4 9 3|6 2 1|5 7 8
-----------------
8 5 7|9 4 3|2 6 1
2 6 1|7 5 8|9 3 4
9 3 4|1 6 2|7 8 5
-----------------
5 7 8|3 9 4|6 1 2
1 2 6|5 8 7|4 9 3
3 4 9|2 1 6|8 5 7
```

Conclusion

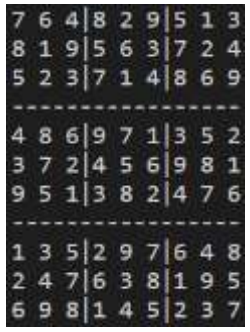
- Une représentation dans la console:
  - Listes
  - Emploi d'une matrice
    - Résultat :



- Une représentation interactive:
  - Le module Pygame



- Une résolution du programme:
  - *Backtracking*
    - Résultat :



Ouverture : pistes de recherches à suivre

Il reste de nombreuses pistes de recherche à exploiter :

- Trouver le nombre de configurations possibles dans un Sudoku 6×6 et 9×9
- Améliorer la version avec Pygame en permetant de :
  - Mettre en valeur la colonne et la ligne de la case selectionnée
  - Vérifier si les conditions sont respectées
  - Rendre l'algorithme compatible avec la version interactive

Notes d’édition I

- (1) Pour la question du nombre de grilles, on peut mentionner une question importante, souvent présente dans les problèmes de dénombrement : est-ce qu’on identifie certaines grilles par symétries ou non ? Cette identification peut rendre le problème plus compliqué, mais peut être plus intuitive dans le résultat renvoyé. Par exemple, considère-t-on une grille et la grille « miroir » comme la même grille ou deux grilles différentes ?

Notes d’édition II

- Sans aller jusqu’à la vraie grille de sudoku, après avoir trouvé $(n^2)!$ pour un premier carré de côté n, il aurait été intéressant de décrire, même sans trouver de formule, comment on pourrait remplir la zone 2 dans un cas plus général (une grille avec 2x2 « zones », chaque zone étant un carré de nxn cases). En particulier, pour n=2, le nombre total de nombres à placer vaut $n^2=4$ et comme on en a utilisé 2 sur la première ligne de la zone 1 cela « force » les deux nombres à placer sur la première ligne de la zone 2. Dans le cas général, on aura utilisé n nombres parmi n^2, donc il y a encore une étape de choix pour la première ligne de la zone 2, ce qui ferait intervenir des coefficients plus compliqués. Pour le cas n=3 par exemple, pour la première ligne de la zone 2 il faut choisir un arrangement de 3 nombres parmi 9-3=6.

Notes d’édition III

- Il aurait peut-être été possible de prendre une grille entrée par l’utilisateur et non pas de la comparer à une grille corrigée mais de vérifier si elle satisfait les conditions pour être une solution.

Notes d’édition IV

- La résolution du sudoku par backtracking est intéressante, il aurait été intéressant d’avoir plus de détails sur la façon dont l’algorithme « retournera à la case fautive » ou « toutes les valeurs associées » en cas de blocage et de s’assurer que le programme trouve forcément une solution s’il y en a une.