

# 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



# 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 posés 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



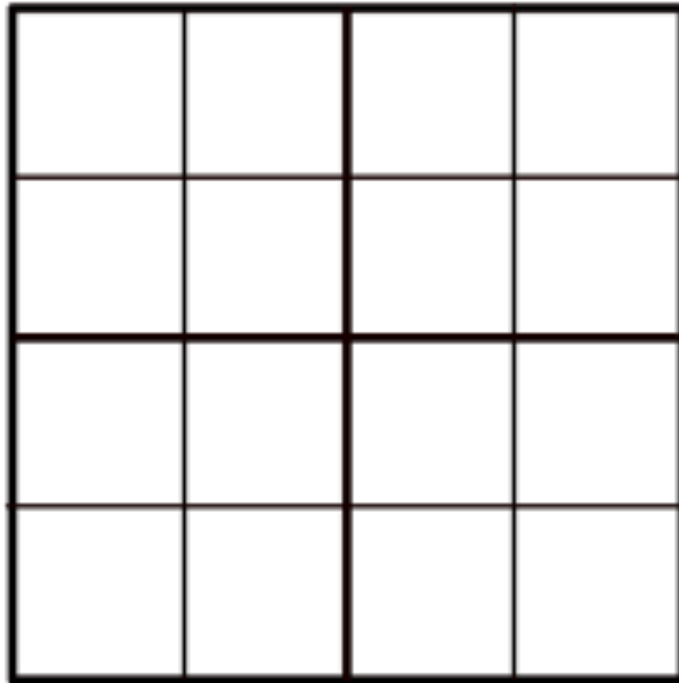
# Nos étapes de recherche:

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

# Passer par un 4×4

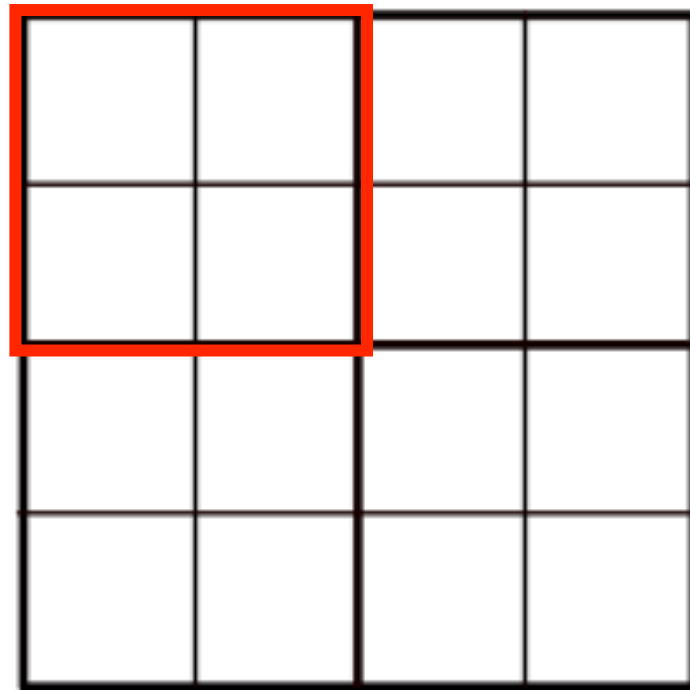
Nous nous sommes d'abord posés la question:

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



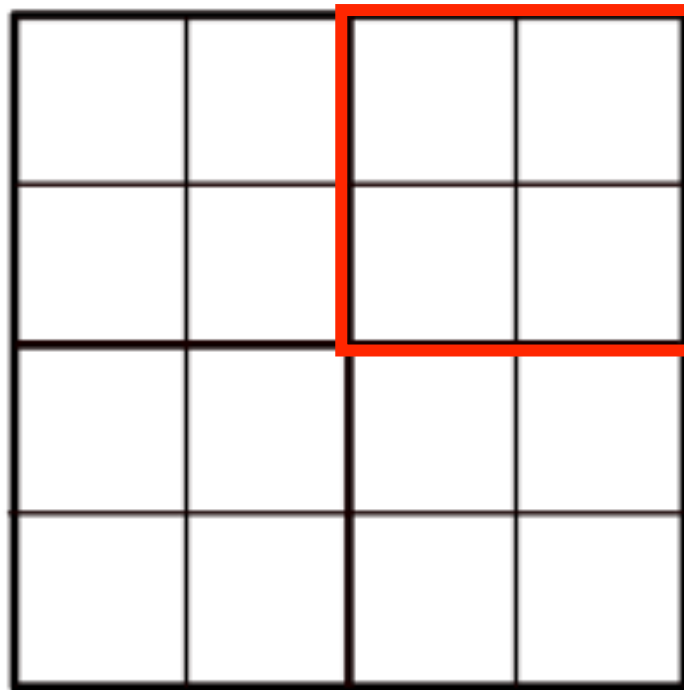
# Passer par un 4×4

Cette zone est la première zone



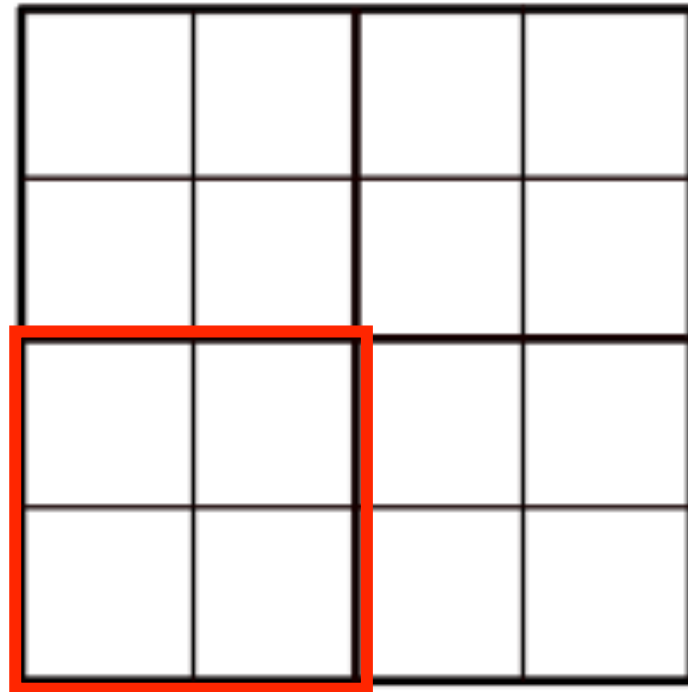
# Passer par un 4×4

Celle-ci est la deuxième zone



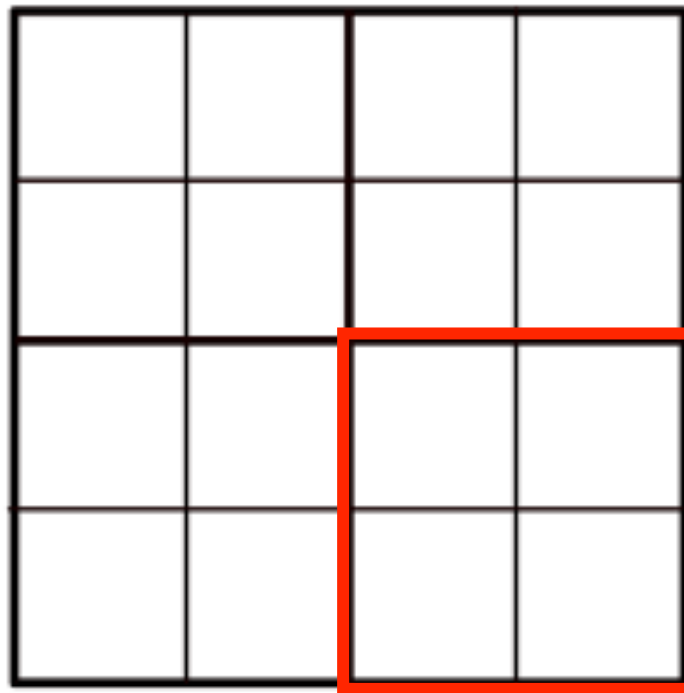
# Passer par un 4×4

Celle-ci est la troisième zone

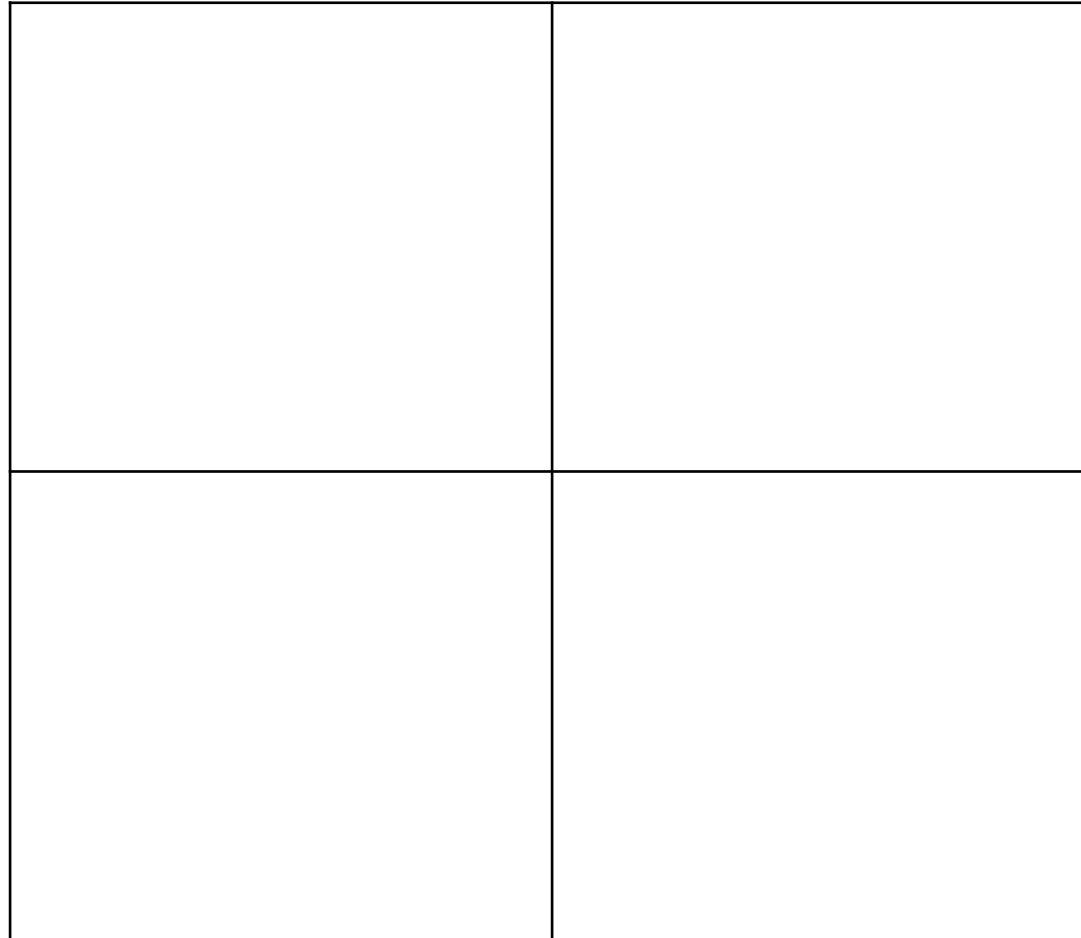


# Passer par un 4×4

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

<b>1</b>	<b>1</b>
<b>1</b>	<b>1</b>



Par exemple, plaçons le 1 en haut à gauche

<b>1</b>	

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

<b>1</b>	2
2	2

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

<b>1</b>	

	<b>2</b>

Il reste alors deux emplacements pour placer le 3

**1**

**2**

3

3

Maintenant, plaçons le 3 en bas à gauche

**1**

**2**

**3**

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

**1**

**2**

**3**

**4**

On procède de la manière suivante :

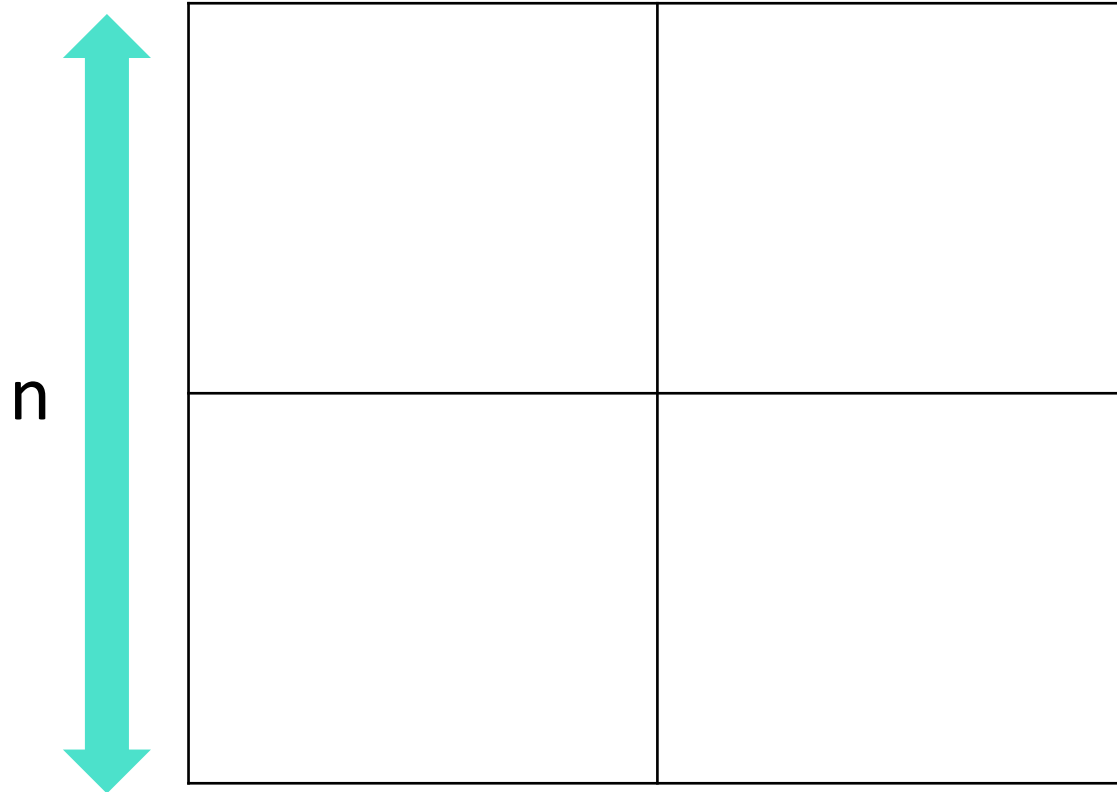
On peut donc dire que pour placer le premier chiffre, il y avait quatre différentes possibilités, ensuite il y a une réduction d'une possibilité à chaque fois que l'on place un nouveau chiffre dans la zone car il y a une case disponible en moins.

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

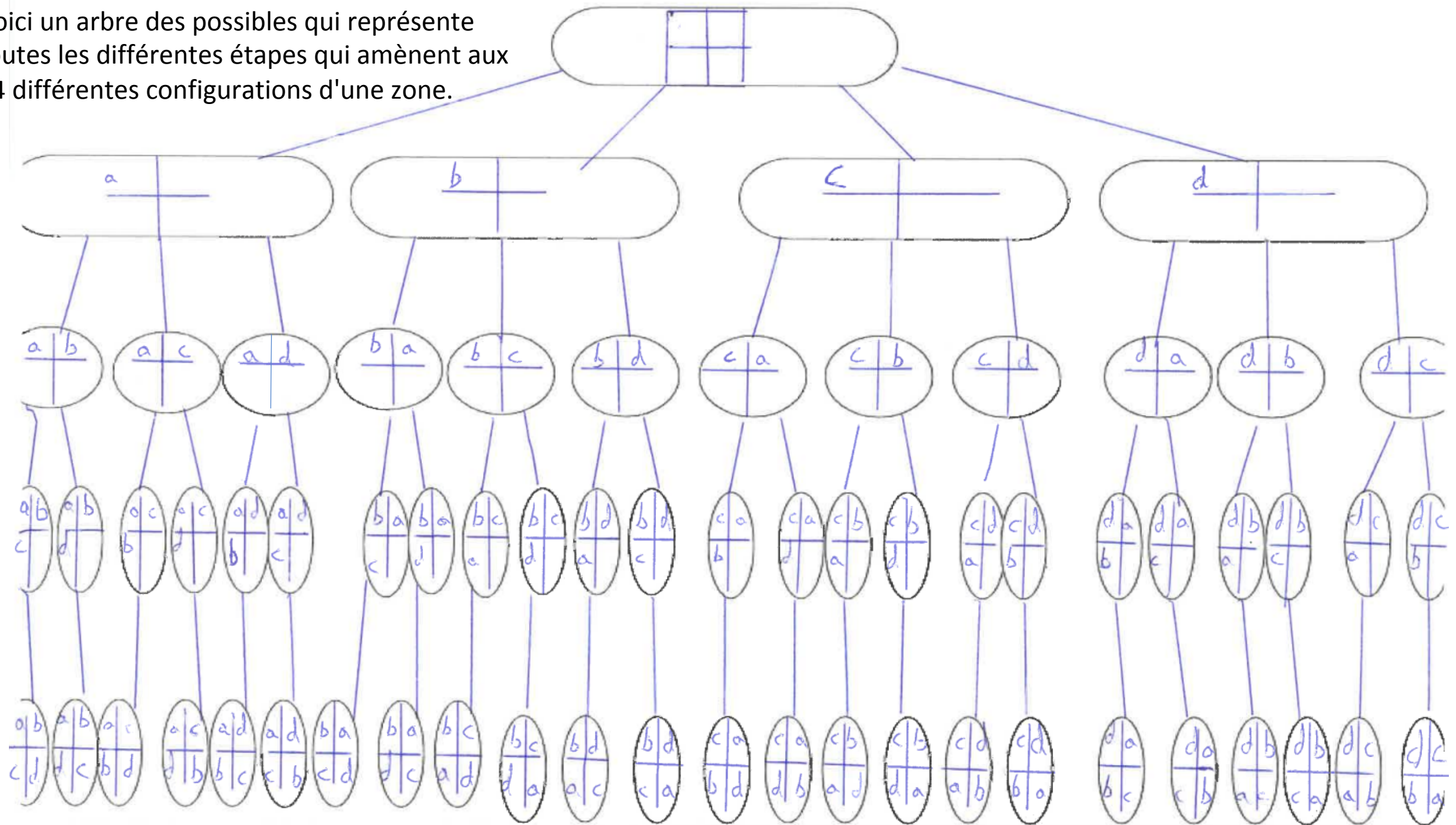
On a trouvé une formule pour calculer toutes les différentes configurations d'un carré de côté  $n$ , on peut utiliser la formule  $(n^2)!$

Par exemple pour le  $2 \times 2$  on applique la formule 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

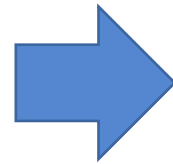
Et 3,4,2,1

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.

<b>1</b>	<b>2</b>
<b>3</b>	<b>4</b>

On voit mieux les quatre différentes configurations ici.



case adjacente : 4 possibilités

<b>3</b>	<b>4</b>
<b>1</b>	<b>2</b>

<b>3</b>	<b>4</b>
<b>2</b>	<b>1</b>

<b>4</b>	<b>3</b>
<b>1</b>	<b>2</b>

<b>4</b>	<b>3</b>
<b>2</b>	<b>1</b>

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  $24 \times 4$  configurations si on regroupe les deux premières zones, soit 96.

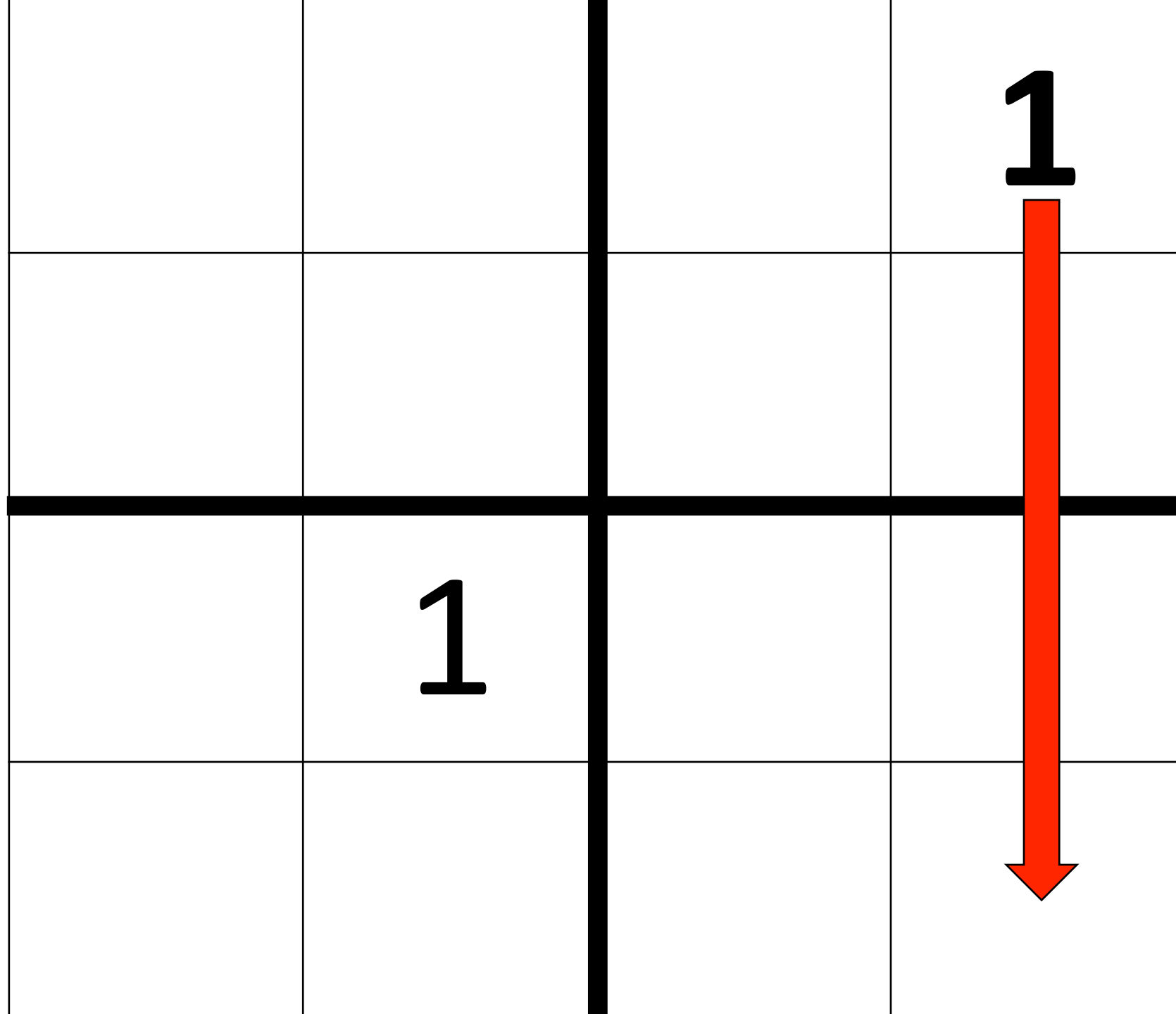


La deuxième zone n'a aucune relation avec la 3ème zone (en diagonale), donc on peut simplement multiplier à nouveau par 4 pour avoir toutes les configurations des trois premières zones, soit 384.

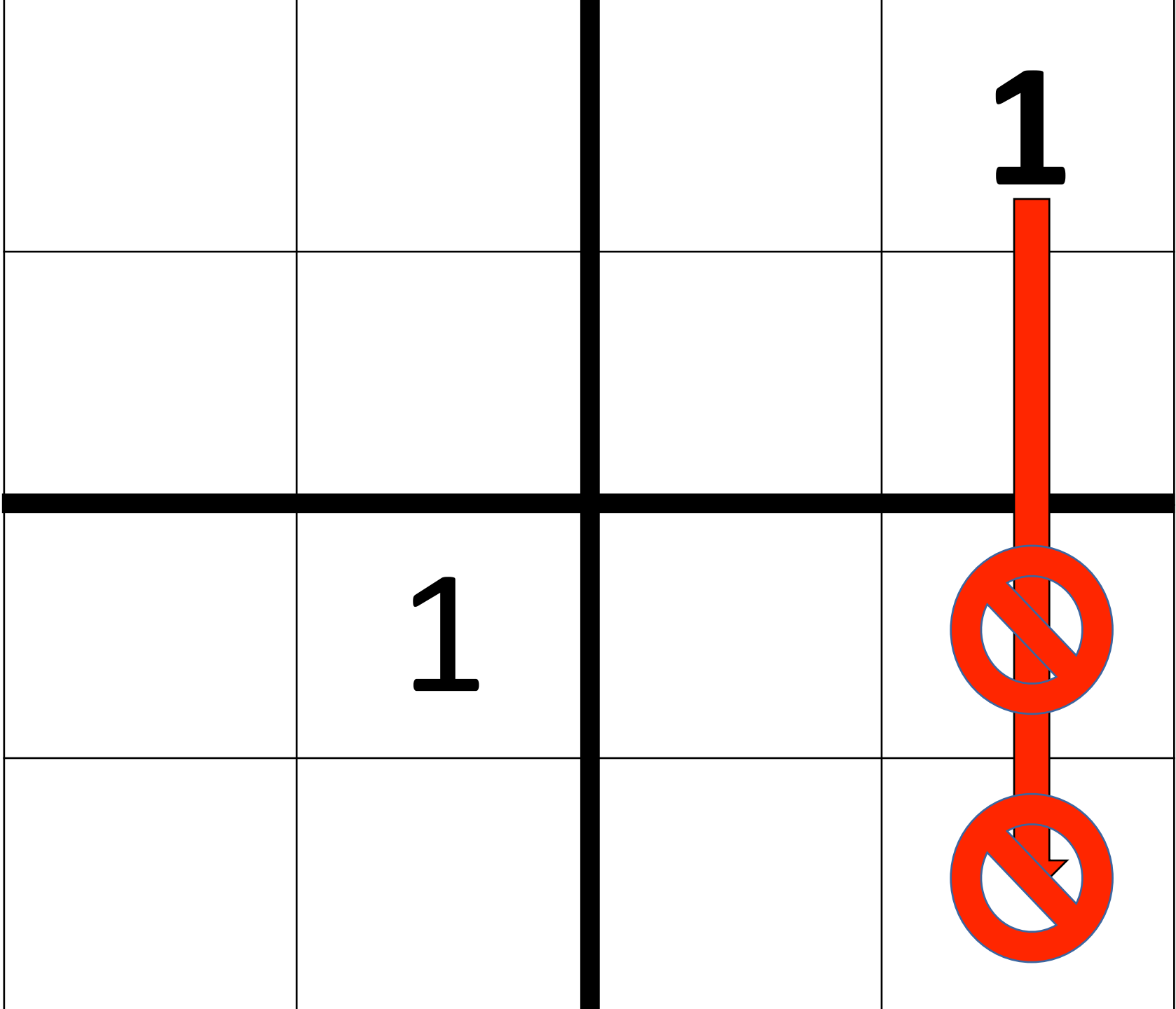
Nous allons maintenant montrer que la dernière zone n'augmente pas le nombre de possibilités.

			<b>1</b>
	<b>1</b>		

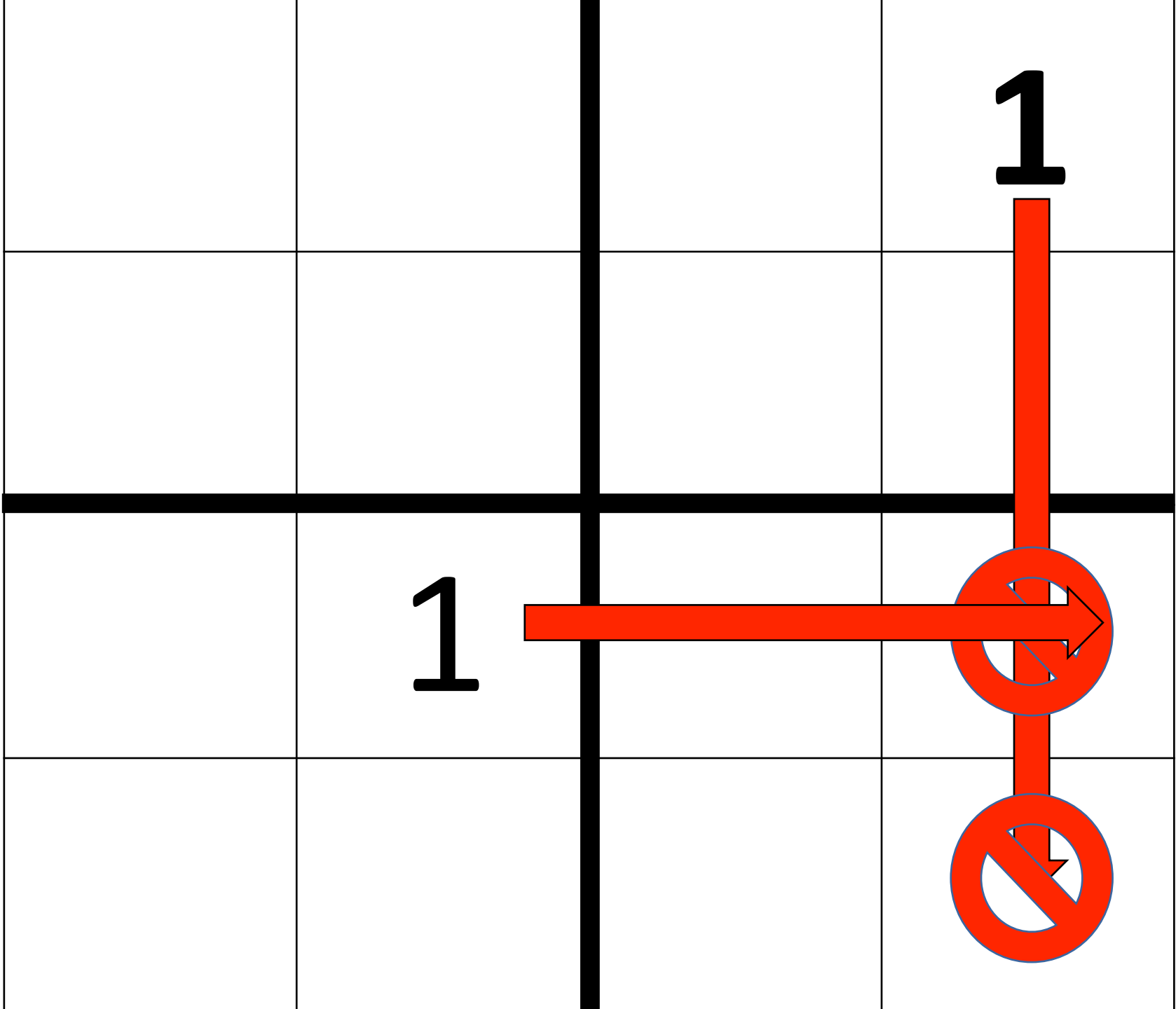
Ici nous avons  
placé  
aléatoirement  
deux 1 dans  
les zones 2 et  
3.

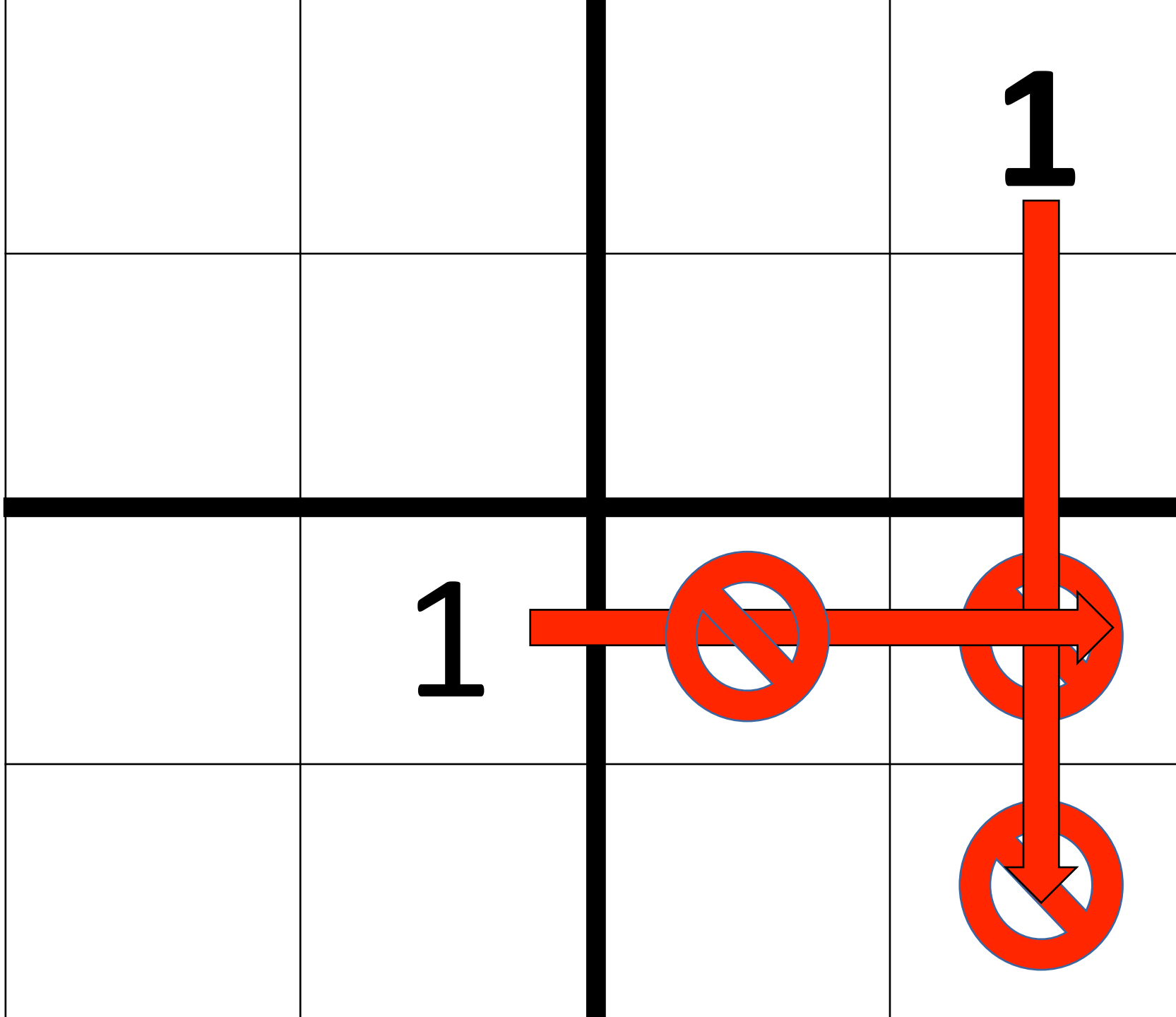


Le 1 de la zone 2  
obstrue la  
colonne dans  
laquelle il est  
placé.



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



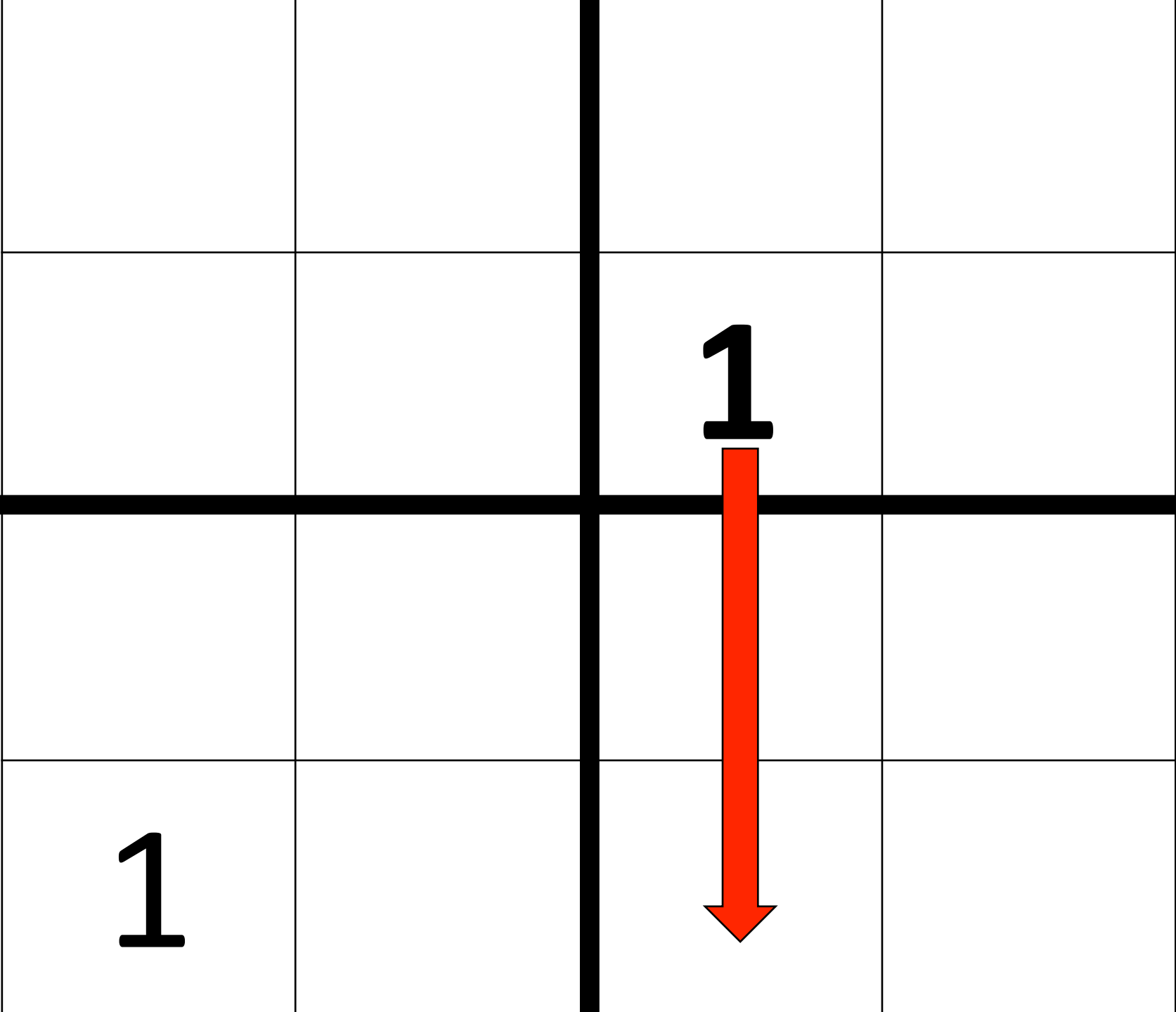




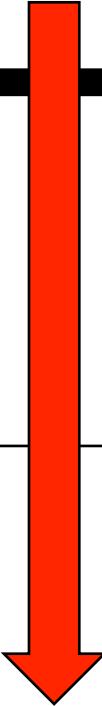
Ici, nous avons suivi la même démarche avec des placements différents.

			<b>1</b>
	<b>1</b>		

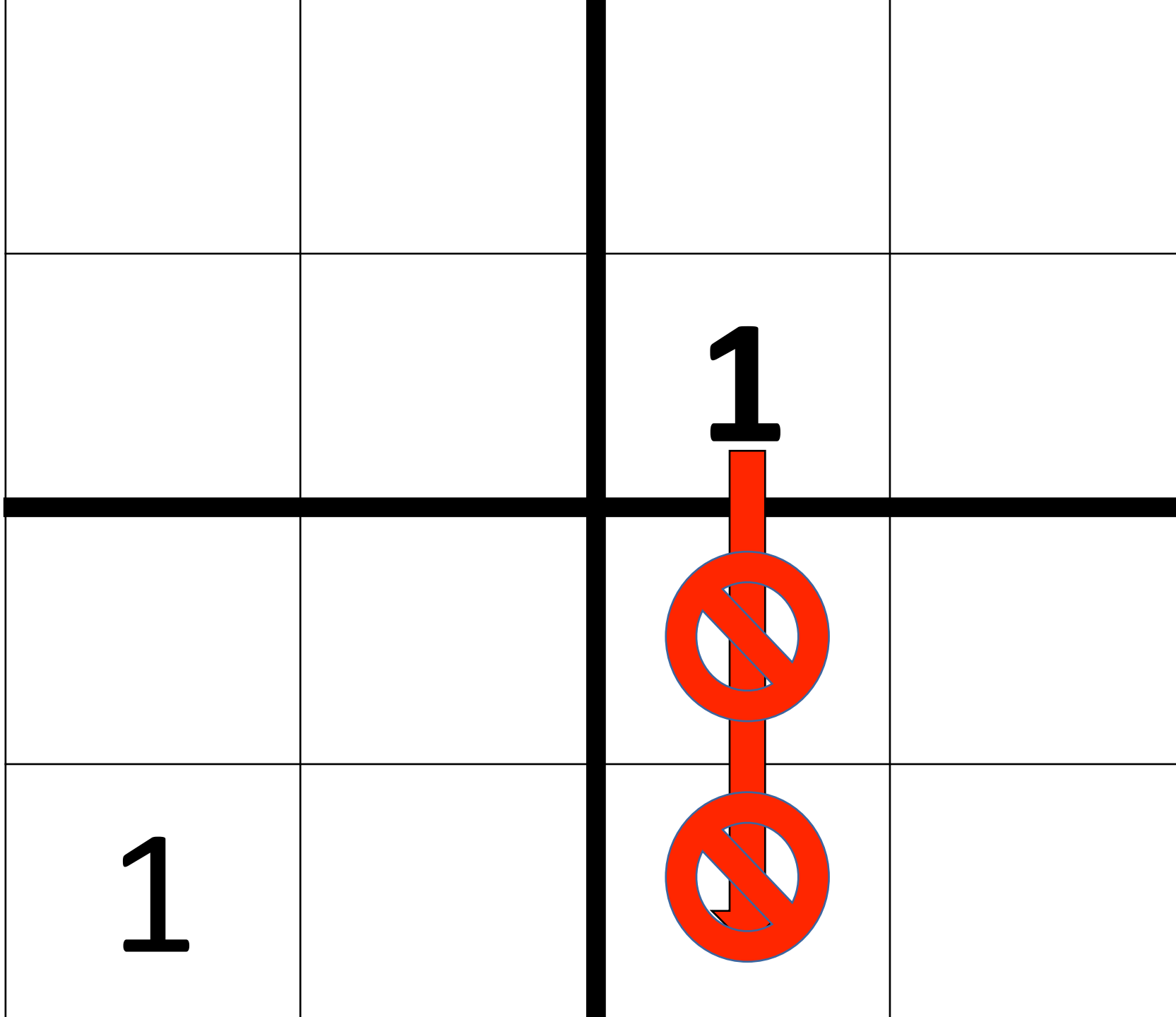


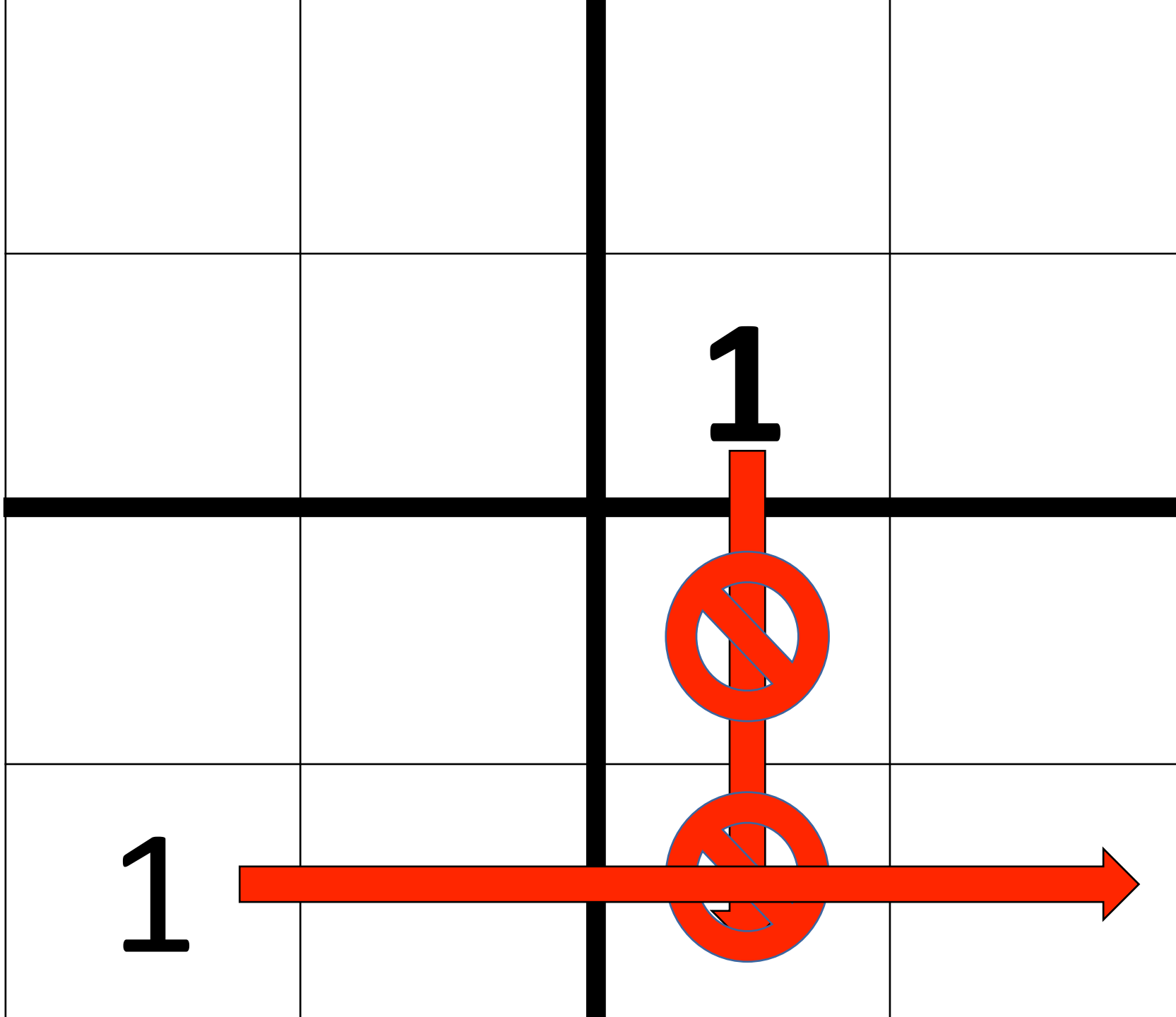


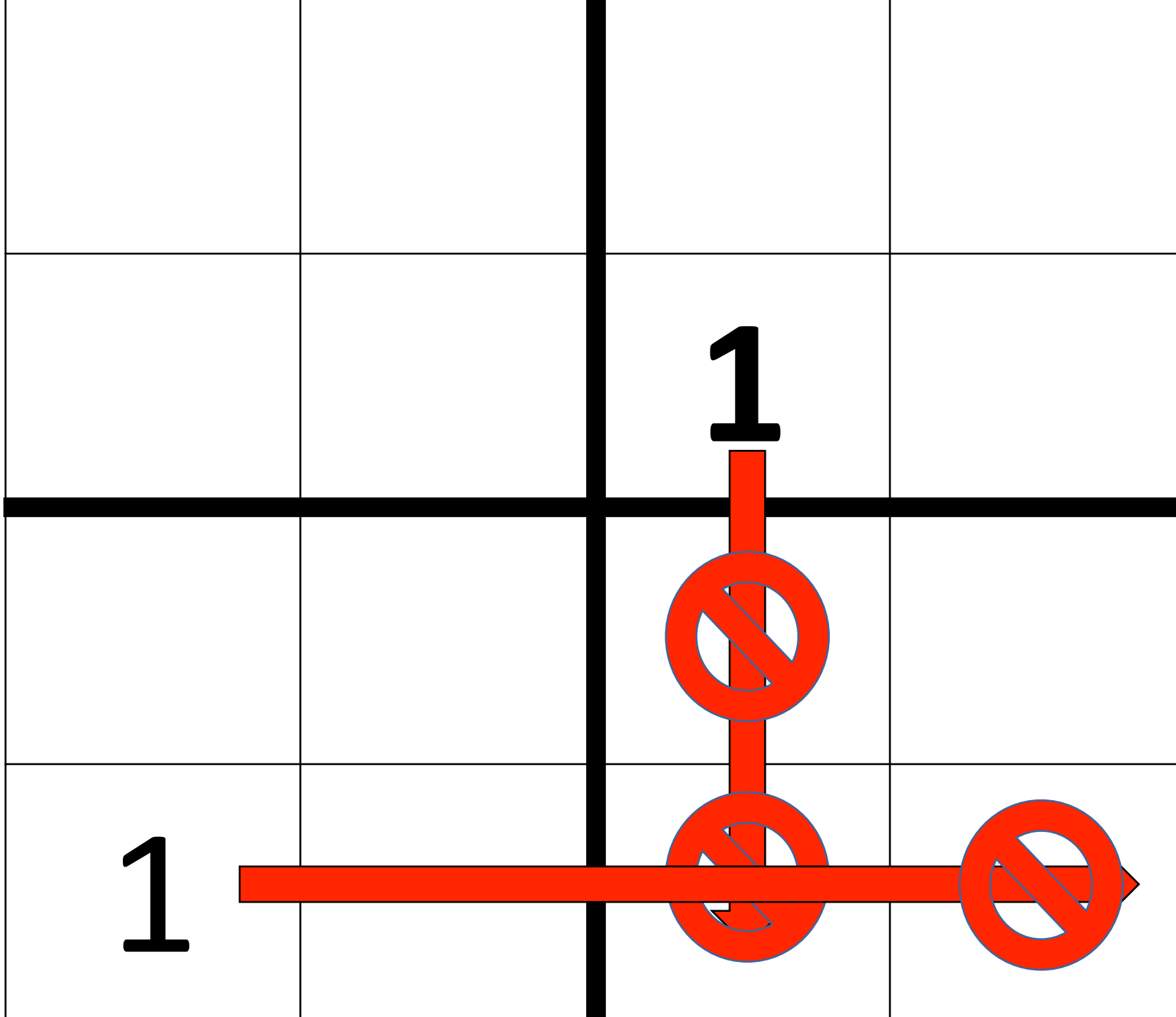
1

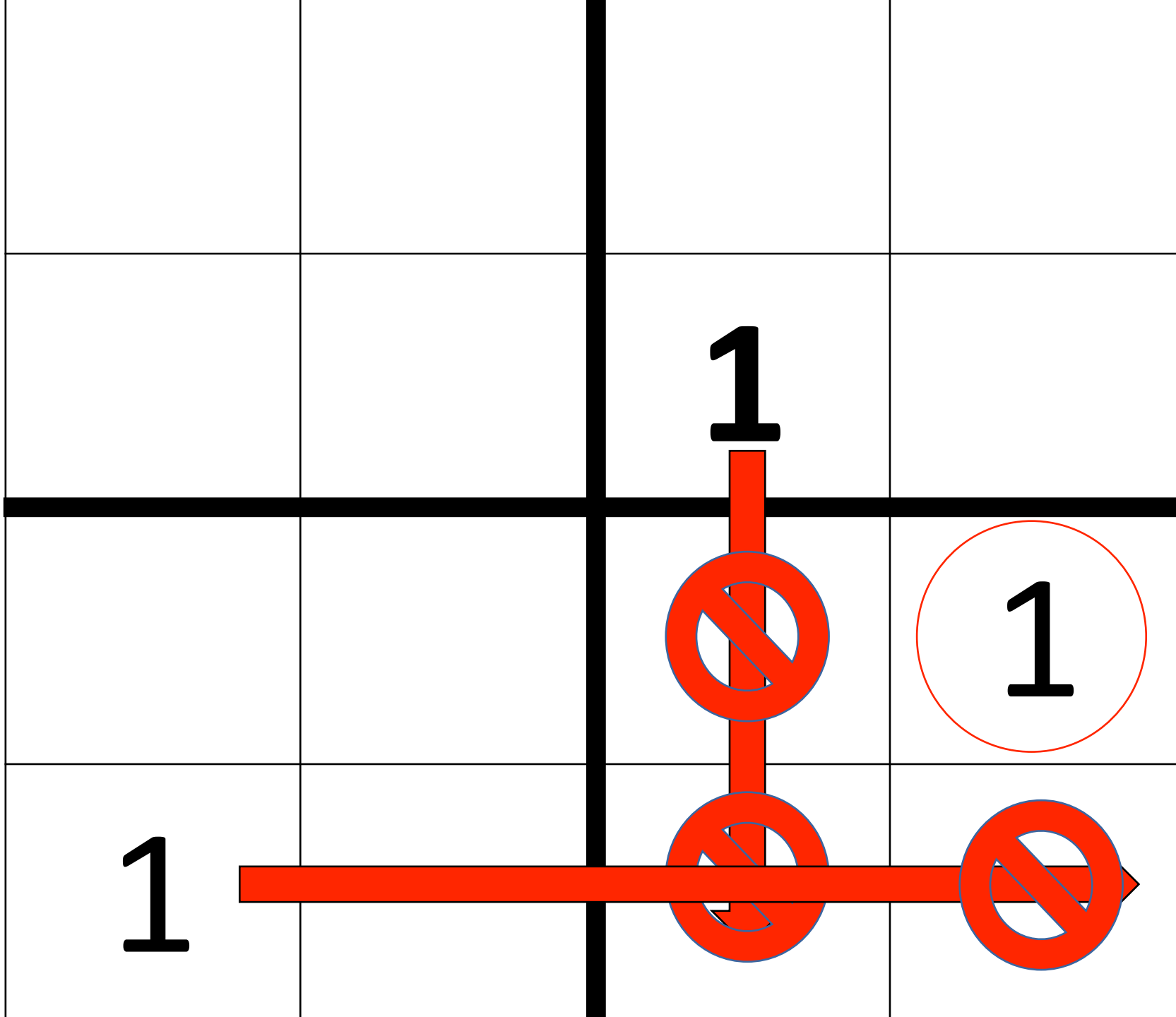


1









# 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 )

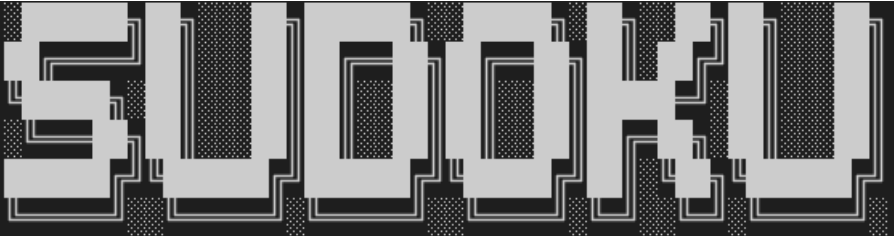
```
a = [[0,3,0], [0,0,0], [0,0,1]]
b = [[9,0,7], [0,8,0], [0,3,0]]
c = [[0,8,0], [2,0,4], [6,0,0]]
d = [[4,0,0], [6,0,7], [3,1,0]]
e = [[8,7,0], [0,2,0], [0,0,9]]
f = [[0,0,3], [1,0,0], [0,4,2]]
g = [[0,1,0], [0,0,0], [0,8,0]]
h = [[5,0,2], [0,6,0], [0,7,0]]
i = [[0,0,0], [3,0,5], [4,0,6]]
```

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



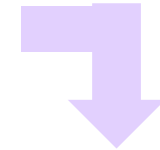
# Grille de sudoku dans la console Python ( Liste )



```
[[0, 3, 0], [0, 0, 0], [0, 0, 1]]  
[[9, 0, 7], [0, 8, 0], [0, 3, 0]]  
[[0, 8, 0], [2, 0, 4], [6, 0, 0]]  
  
[[4, 0, 0], [6, 0, 7], [3, 1, 0]]  
[[8, 7, 0], [0, 2, 0], [0, 0, 9]]  
[[0, 0, 3], [1, 0, 0], [0, 4, 2]]  
  
[[0, 1, 0], [0, 0, 0], [0, 8, 0]]  
[[5, 0, 2], [0, 6, 0], [0, 7, 0]]  
[[0, 0, 0], [3, 0, 5], [4, 0, 6]]  
  
ligne 1: █
```



```
ligne 1: 2 3 4 9 7 6 8 5 1  
[[2, 3, 4], [9, 7, 6], [8, 5, 1]]  
  
ligne 2: 9 6 7 5 8 1 2 3 4  
[[9, 6, 7], [5, 8, 1], [2, 3, 4]]  
  
ligne 3: 1 8 5 2 3 4 6 9 7  
[[1, 8, 5], [2, 3, 4], [6, 9, 7]]  
  
ligne 4: 4 2 9 6 5 7 3 1 8  
[[4, 2, 9], [6, 5, 7], [3, 1, 8]]  
  
ligne 5: 8 7 1 4 2 3 5 6 9  
[[8, 7, 1], [4, 2, 3], [5, 6, 9]]  
  
ligne 6: 6 5 3 1 9 8 7 4 2  
[[6, 5, 3], [1, 9, 8], [7, 4, 2]]  
  
ligne 7: 3 1 6 7 4 2 8 8 5  
[[3, 1, 6], [7, 4, 2], [8, 8, 5]]  
  
ligne 8: 5 4 2 8 6 9 1 7 3  
[[5, 4, 2], [8, 6, 9], [1, 7, 3]]  
  
ligne 9: 7 9 8 3 1 5 4 2 6  
[[7, 9, 8], [3, 1, 5], [4, 2, 6]]
```



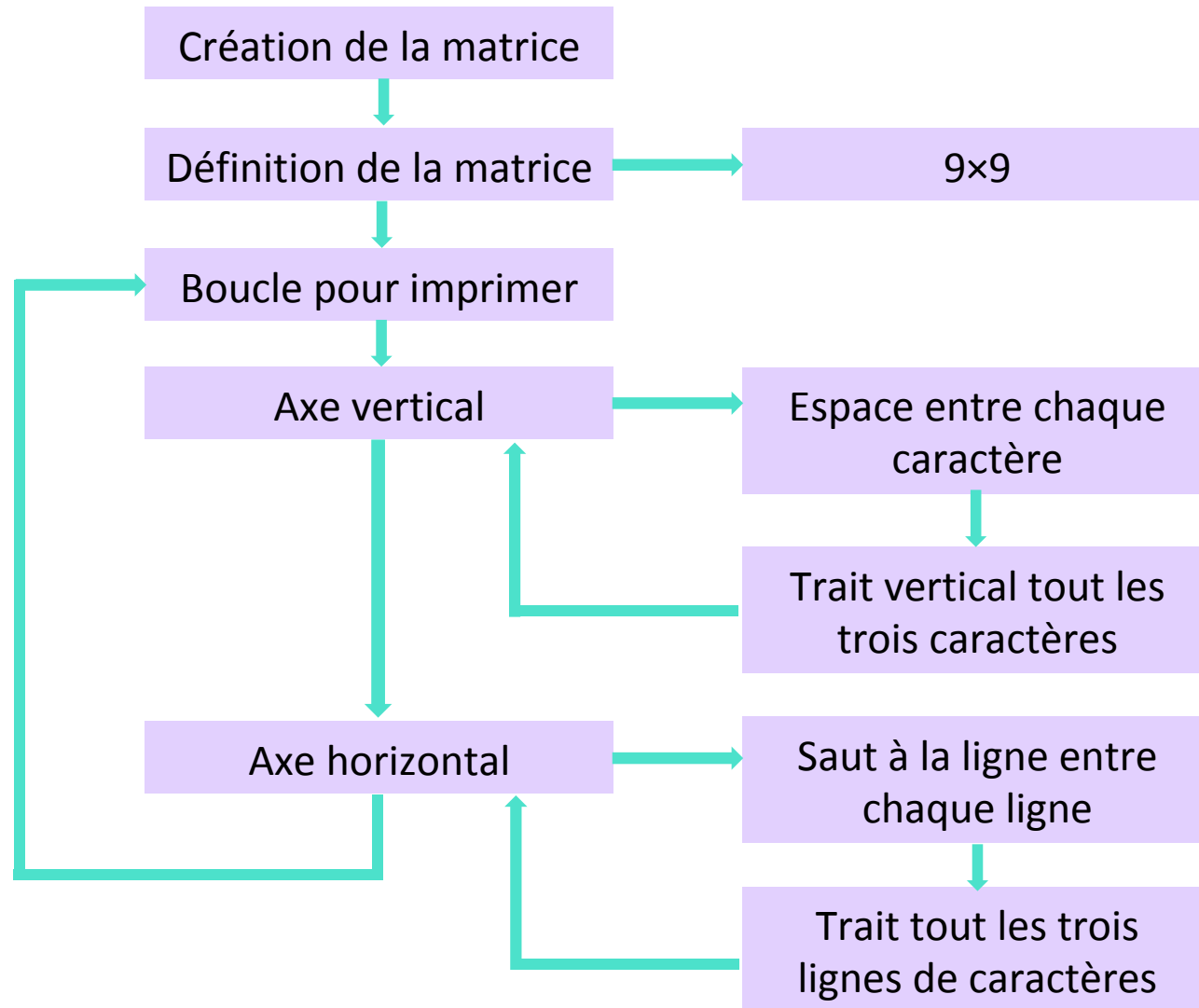
Console affiche:  
"Vous avez bien résolu le  
sudoku!"

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!"

Nous avons essayé de faire un programme 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:

```
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
-----
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
-----
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
```

- 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 avons espéré pouvoir sélectionner chaque valeur de la matrice et la changer pour y mettre les valeurs d'une grille de Sudoku.

Enfin, 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

pygame window

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



pygame window

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



# Algorithme pour résoudre un Sudoku : *Backtracking*

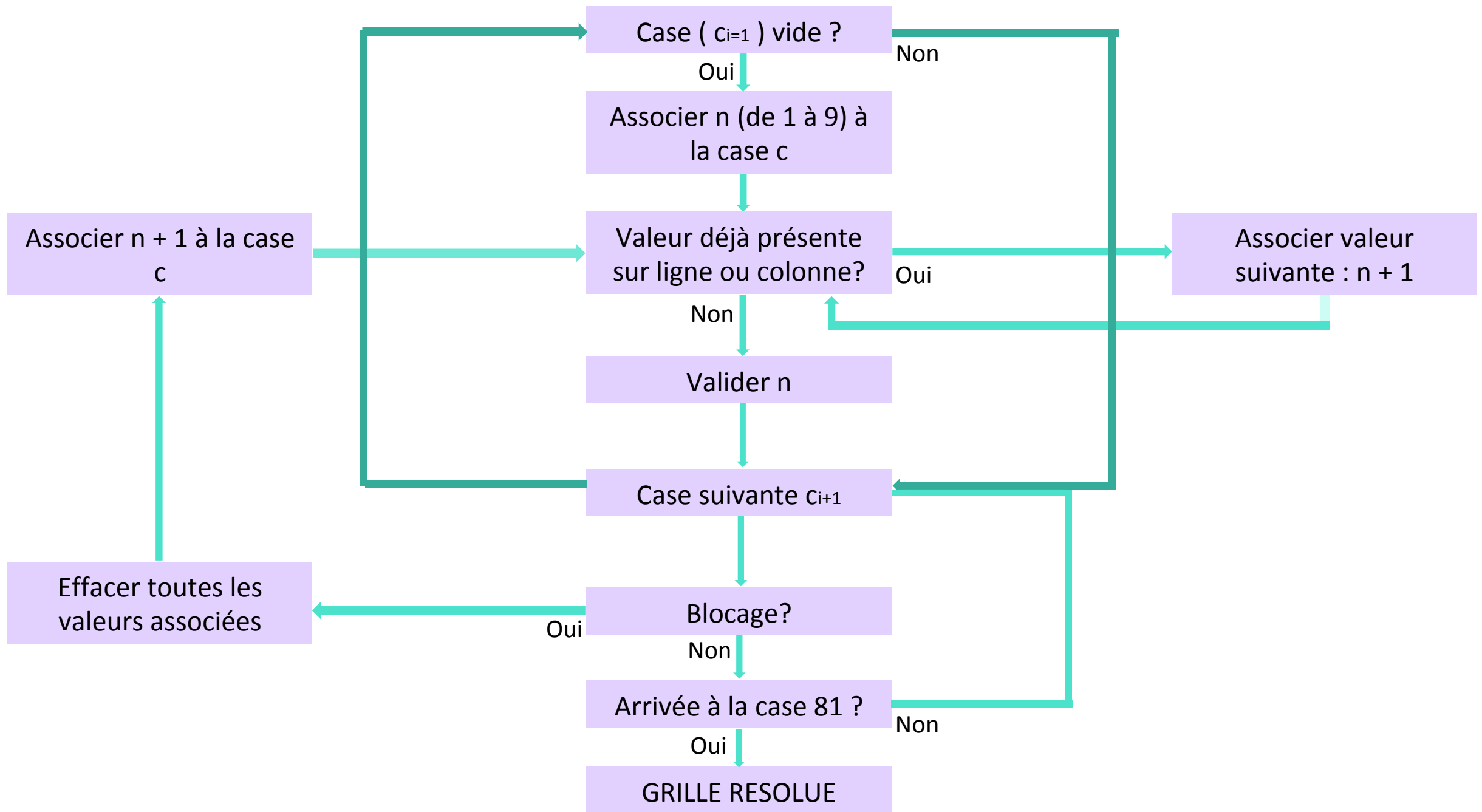
Nous avons cherché à résoudre le Sudoku par un algorithme. Celui-ci se base sur le principe du *Backtracking*. C'est une catégorie d'algorithme qui explore toutes les possibilités pour résoudre un problème. Autrement dit, il va essayer de résoudre la grille en essayant tous les chiffres possibles, toutes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





# 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  $n$  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 fautive 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.

# 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:

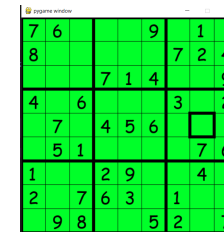
- Listes
- Emploi d'une matrice

- Résultat :

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
-----								
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
-----								
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

- Une représentation interactive:

- Le module Pygame



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



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

- Une résolution du programme:

- *Backtracking*

- Résultat :

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

# 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 permettant de :
  - Mettre en valeur la colonne et la ligne de la case sélectionnée
  - Vérifier si les conditions sont respectées
  - Rendre l'algorithme compatible avec la version interactive