

Sudoku

par Anne & Lou

Répartitions



Nom de la tâche	Personnes assignées
Résoudre le sudoku manuellement - ECHEC	Anne
Checker les lignes	Anne
Résoudre le sudoku de manière automatique	Anne & Lou
Checker les colonnes	Lou
Créer une grille de sudoku	Anne & Lou
Checker les carrés	Anne & Lou
Tenter le Sudoku en pygame	Lou
Résoudre le sudoku via un solver	Anne & Lou
Créer un creator Facile	Anne & Lou
Créer un creator Moyen	Anne & Lou
Créer un creator Difficile	Anne & Lou



Initialisation



Grille vraie

Grille fausse

Grille random



ne of ye
someth
lackth
ay.
watch tie
night
ating,
or all
-place
f big-c
me year
rich I
berwe
tencio

Base de travail pour tester notre solver

1.1.2 Grille vraie

Entrée [14]: #Grille donnée par Anne

```
good_grid = [
    [5, 3, 4, 6, 7, 8, 9, 1, 2],
    [6, 7, 2, 1, 9, 5, 3, 4, 8],
    [1, 9, 8, 3, 4, 2, 5, 6, 7],
    [8, 5, 9, 7, 6, 1, 4, 2, 3],
    [4, 2, 6, 8, 5, 3, 7, 9, 1],
    [7, 1, 3, 9, 2, 4, 8, 5, 6],
    [9, 6, 1, 5, 3, 7, 2, 8, 4],
    [2, 8, 7, 4, 1, 9, 6, 3, 5],
    [3, 4, 5, 2, 8, 6, 1, 7, 9]
]
```

```
print_grid(good_grid)
```

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

1.1.1 Grille fausse

Entrée [13]: def print_grid(grid):
 for row in grid:
 print(row)

```
false_grid = [[1, 2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 4, 5, 6, 7, 8, 9]]
```

```
print_grid(false_grid)
```

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

1.1.3 Grille random

Entrée [17]: random_grid = [[1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 2, 3, 4, 5, 6, 7, 8, 9]]

Mélanger les lignes de la grille
random.shuffle(random_grid)

Mélanger les éléments de chaque ligne de la grille
for row in random_grid:
 random.shuffle(row)

```
print_grid(random_grid)
```

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

Mise en forme de la grille

Affichage de la grille de Sudoku

- `(i + 1) % 3 == 0` vérifie si nous sommes sur la 3e, la 6e, la 9e ligne pour diviser la grille
- `i != 8` vérifie que nous ne sommes pas sur la dernière ligne

Entrée [24]:

```
def print_sudoku(random_grid):
    for i, row in enumerate(random_grid):
        print(*row[:3], "|", *row[3:6], "|", *row[6:])

        if (i + 1) % 3 == 0 and i != 8:
            print("-" * 21)

print_sudoku(random_grid)
```

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

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

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

Sudoku checker



Objectifs

1. Vérifier que toutes les lignes possèdent les chiffres 1 à 9 sans doublon
2. Vérifier que toutes les colonnes possèdent les chiffres 1 à 9 sans doublon
3. Vérifier que tous les carrés possèdent les chiffres 1 à 9 sans doublon



1.2.1 Ligne checker

Ce code vérifie si chaque ligne d'une grille de sudoku contient tous les nombres de 1 à 9.

```
Entrée [7]: def ligne_checker(good_grid):
    for i in range(0, 9):                      #Parcours chaque ligne
        ligne = good_grid[i]                     #Sélectionne une ligne
        if sorted(ligne) != [1, 2, 3, 4, 5, 6, 7, 8, 9]: #Vérifie si la ligne contient tous les nombres de 1 à 9
            return False
        else:
            return True

print(ligne_checker(good_grid))
```

True

1.2.2 Colonne checker

Ce code vérifie si chaque colonne d'une grille de sudoku contient tous les nombres de 1 à 9.

```
Entrée [9]: def colonne_checker(good_grid):
    for colonne in range(9):      # Parcours chaque colonne
        valeurs_colonne = []       # Initialise une liste pour stocker les valeurs de la colonne actuelle

        for ligne in range(9):      # Parcours chaque ligne
            valeurs_colonne.append(good_grid[ligne][colonne])      # Ajoute la valeur de la cellule actuelle de la colonne à la liste

        if sorted(valeurs_colonne) != [1, 2, 3, 4, 5, 6, 7, 8, 9]: # Vérifie si la colonne contient tous les nombres de 1 à 9
            return False
    return True

print(colonne_checker(good_grid))
True
```

1.2.3 Carrée checker

Ce code vérifie si chaque bloc 3x3 d'une grille de sudoku contient des valeurs uniques comprises entre 1 et 9.

```
Entrée [11]: def carre_checker(good_grid):
    for i in range(0, 9, 3):                      # Parcours les blocs de 3x3
        for j in range(0, 9, 3):
            valeurs_carre = set()                    # Initialise un ensemble pour stocker les valeurs uniques du bloc actuel

            for ligne in range(i, i + 3):             # Parcours chaque cellule du bloc actuel
                for colonne in range(j, j + 3):
                    valeur = good_grid[ligne][colonne]  # Enregistre la valeur de la cellule actuelle

                    if valeur in valeurs_carre or valeur < 1 or valeur > 9: # Vérifie si la valeur est déjà présente dans le bloc
                        return False
                    valeurs_carre.add(valeur) # Ajoute la valeur à l'ensemble pour vérification ultérieure.

    return True

print(carre_checker(good_grid))

```

True

Fonction qui réunit notre checker

▼ 1.2.4 Fonction qui appelle mes trois fonctions

Ce code vérifie si une grille de sudoku est valide en vérifiant simultanément ses lignes, colonnes et blocs 3x3.

```
Entrée [11]: def sudoku_checker(good_grid): # Remplacer 'good_grid' par le nom de la variable à utiliser
    return ligne_checker(good_grid) and colonne_checker(good_grid) and carre_checker(good_grid)

print(sudoku_checker(good_grid))
```

True

Sudoku solver



Entrée [30]:

```
grid_empty = [
    [1, 0, 0, 0, 0, 0, 7, 9, 4],
    [0, 5, 0, 0, 0, 0, 0, 8, 0],
    [3, 0, 0, 0, 0, 0, 6, 0, 0],
    [0, 0, 0, 2, 0, 0, 0, 4, 0],
    [0, 0, 0, 0, 7, 0, 0, 0, 0],
    [0, 4, 0, 0, 0, 6, 0, 0, 0],
    [0, 0, 3, 0, 0, 0, 0, 0, 6],
    [0, 0, 0, 0, 0, 0, 0, 2, 0],
    [2, 6, 0, 0, 0, 0, 0, 0, 8]
]

print_grid(grid_empty)
```

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

Mise en forme de la grille

Entrée [31]:

```
def test_sudoku(grid_empty):
    for i, row in enumerate(grid_empty):
        print(*row[:3], "|", *row[3:6], "|", *row[6:])

        if (i + 1) % 3 == 0 and i != 8:
            print("-" * 21)
```

test_grid = grid_empty #Je réattribue ma variable pour ne pas écraser la précédente
test_sudoku(test_grid)

1	0	0		0	0	0		7	9	4
0	5	0		0	0	0		0	8	0
3	0	0		0	0	0		6	0	0

0	0	0		2	0	0		0	4	0
0	0	0		0	7	0		0	0	0
0	4	0		0	0	6		0	0	0

0	0	3		0	0	0		0	0	6
0	0	0		0	0	0		0	2	0
2	6	0		0	0	0		0	0	8

Entrée [32]:

```
#Etape 1
def get_empty(grid_empty):
    cases_vides = []
    for ligne in range(9):
        for colonne in range(9):
            if grid_empty[ligne][colonne] == 0:
                cases_vides.append((ligne, colonne))
    return cases_vides

cases_vides = get_empty(grid_empty)
print("Cases vides détectées : ", cases_vides)
print("\n")

#Etape 2
def get_possibility(grid_empty, ligne, colonne):
    chiffres_posibles = set(range(1, 10))
    valeur_case = grid_empty[ligne][colonne]
    for y in range(9):
        chiffres_posibles.discard(grid_empty[ligne][y])
    for x in range(9):
        chiffres_posibles.discard(grid_empty[x][colonne])

    start_ligne, start_colonne = 3 * (ligne // 3), 3 * (colonne // 3)
    for x in range(start_ligne, start_ligne + 3):
        for y in range(start_colonne, start_colonne + 3):
            chiffres_posibles.discard(grid_empty[x][y])
    return list(chiffres_posibles)

cases_vides = get_empty(grid_empty)
for case in cases_vides:
    ligne, colonne = case
    possibilites = get_possibility(grid_empty, ligne, colonne)
    print("Case vide, ligne & colonne : {}, Possibilités : {}".format(case, possibilites))
```

1.3.2 Déetecter, lister et déterminer

Visualiser les cases vides et leur proposition

- Etape #1 : détecter et me lister les cases vides
- Etape #2 : détermine les chiffres possibles

Cases vides détectées : [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 0), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 8), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 7), (2, 8), (3, 0), (3, 1), (3, 2), (3, 4), (3, 5), (3, 6), (3, 8), (4, 0), (4, 1), (4, 2), (4, 3), (4, 5), (4, 6), (4, 7), (4, 8), (5, 0), (5, 2), (5, 3), (5, 4), (5, 6), (5, 7), (5, 8), (6, 0), (6, 1), (6, 3), (6, 4), (6, 5), (6, 6), (6, 7), (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (7, 8), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7)]

Case vide, ligne & colonne : (0, 1), Possibilités : [2, 8]
Case vide, ligne & colonne : (0, 2), Possibilités : [2, 6, 8]
Case vide, ligne & colonne : (0, 3), Possibilités : [3, 5, 6, 8]
Case vide, ligne & colonne : (0, 4), Possibilités : [2, 3, 5, 6, 8]
Case vide, ligne & colonne : (0, 5), Possibilités : [2, 3, 5, 8]
Case vide, ligne & colonne : (1, 0), Possibilités : [4, 6, 7, 9]
Case vide, ligne & colonne : (1, 2), Possibilités : [2, 4, 6, 7, 9]
Case vide, ligne & colonne : (1, 3), Possibilités : [1, 3, 4, 6, 7, 9]
Case vide, ligne & colonne : (1, 4), Possibilités : [1, 2, 3, 4, 6, 9]
Case vide, ligne & colonne : (1, 5), Possibilités : [1, 2, 3, 4, 7, 9]
Case vide, ligne & colonne : (1, 6), Possibilités : [1, 2, 3]
Case vide, ligne & colonne : (1, 8), Possibilités : [1, 2, 3]

1.3.3.1 Trouver la première case vide

Trouver la 1ère case vide

```
Entrée [33]: def find_empty_location(test_grid):
    for l in range(9):
        for c in range(9):

            if test_grid[l][c] == 0: # Vérifie si la case est vide (valeur égale à 0)
                return l, c
    return None, None
```

1.3.3.2 Vérifier la validité

Vérifie si le numero inscrit est True ou False, pour savoir si c'est le bon numéro ou pas en checkant : ligne, colonne et carrée

```
Entrée [34]: def is_valid(test_grid, row, col, num):

    for l in range(9):
        if test_grid[row][l] == num: #Vérifier si le chiffre 'num' est déjà présent dans la ligne
            return False

    for c in range(9):
        if test_grid[c][col] == num: #Vérifier si le chiffre 'num' est déjà présent dans la colonne
            return False

    start_row, start_col = 3 * (row // 3), 3 * (col // 3)
    for l in range(3):
        for c in range(3):
            if test_grid[l + start_row][c + start_col] == num: #Vérifier si le chiffre 'num' est déjà présent dans le carré
                return False

    return True
```

L'algorythme va aller chercher la première case vide, la remplir avec les chiffres que l'algo estime possible. Ensuite elle va mettre en oeuvre la récursivité. Si tout fonctionne il continue, sinon il effectue un backtraking. Pour voir l'évolution de l'algorythme, on affiche chaque étape pour voir son fonctionnement. A l'issue, on affiche le sudoku terminé.

```
Entrée [56]: def solve_sudoku(test_grid):
    row, col = find_empty_location(test_grid)

    if row is None:
        return True

    for num in range(1, 10):
        if is_valid(test_grid, row, col, num):
            test_grid[row][col] = num      #Place le chiffre 'num' dans la case vide
            print("chiffre", num, "en", (row, col))
            test_sudoku(test_grid)        #Affichage de l'étape
            print("\n")

            if solve_sudoku(test_grid):  #Récursivité & Backtracking
                return True

            print("Retour en", (row, col), "pour tester le chiffre suivant.")
            test_sudoku(test_grid)      #Affichage de l'étape
            print("\n")
            test_grid[row][col] = 0      #Remplace la valeur précédente par la suivante

    return False

grid_reso = test_grid

print("Sudoku initial :")
test_sudoku(grid_reso)
print("\n")

print("Sudoku résolu :")
print("\n")
solve_sudoku(grid_reso)
test_sudoku(grid_reso)
```

Sudoku initial :

1	0	0		0	0	0		7	9	4
0	5	0		0	0	0		0	8	0
3	0	0		0	0	0		6	0	0

0	0	0		2	0	0		0	4	0
0	0	0		0	7	0		0	0	0
0	4	0		0	0	6		0	0	0

0	0	3		0	0	0		0	0	6
0	0	0		0	0	0		0	2	0
2	6	0		0	0	0		0	0	8

Sudoku résolu :

chiffre 2 en (0, 1)

1	2	0		0	0	0		7	9	4
0	5	0		0	0	0		0	8	0
3	0	0		0	0	0		6	0	0

0	0	0		2	0	0		0	4	0
0	0	0		0	7	0		0	0	0
0	4	0		0	0	6		0	0	0

0	0	3		0	0	0		0	0	6
0	0	0		0	0	0		0	2	0
2	6	0		0	0	0		0	0	8

chiffre 6 en (0, 2)

1	2	6		0	0	0		7	9	4
---	---	---	--	---	---	---	--	---	---	---

Retour en (1, 4) pour tester le chiffre suivant.

1	2	6		3	5	8		7	9	4
4	5	7		1	2	0		0	8	0
3	0	0		0	0	0		6	0	0

0	0	0		2	0	0		0	4	0
0	0	0		0	7	0		0	0	0
0	4	0		0	0	6		0	0	0

0	0	3		0	0	0		0	0	6
0	0	0		0	0	0		0	2	0
2	6	0		0	0	0		0	0	8

chiffre 6 en (1, 4)

1	2	6		3	5	8		7	9	4
4	5	7		1	6	0		0	8	0
3	0	0		0	0	0		6	0	0

0	0	0		2	0	0		0	4	0
0	0	0		0	7	0		0	0	0
0	4	0		0	0	6		0	0	0

0	0	3		0	0	0		0	0	6
0	0	0		0	0	0		0	2	0
2	6	0		0	0	0		0	0	8

chiffre 2 en (1, 5)

1	2	6		3	5	8		7	9	4
4	5	7		1	6	2		0	8	0
3	0	0		0	0	0		6	0	0

0	0	0		2	0	0		0	4	0
---	---	---	--	---	---	---	--	---	---	---

Vérifie si la grille est bien résolu ou non

```
Entrée [36]: def sudoku_checker(grid_reso):
    return ligne_checker(grid_reso) and colonne_checker(grid_reso) and carre_checker(grid_reso)

print(sudoku_checker(grid_reso))
```

True

Sudoku creator



Génère une grille de sudoku valide + définition de suppression des nombres en fonction du niveau de difficulté choisi.

Entrée [79]:

```
def generate_sudoku():
    nums = random.sample(range(1, 10), 9)           #Liste contenant tous les chiffres mélangé de 1 à 9

    grid = []
    for r in range(9):
        row = []
        for c in range(9):
            index = (3 * (r % 3) + r // 3 + c) % 9   #Calcul de l'indice dans la liste mélangée pour placer un chiffre dans la case
            row.append(nums[index])                   #Ajout du chiffre à la ligne
        grid.append(row)                           #Ajout de la ligne à la grille
    return grid

grid = generate_sudoku()
display(grid)
```

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

Ce code retire aléatoirement un nombre spécifié de chiffres d'une grille de sudoku en fonction du niveau de difficulté donné. Il stocke temporairement la valeur de chaque cellule, la remplace par zéro pour la retirer de la grille, puis arrête lorsqu'il a retiré le nombre spécifié de chiffres selon le niveau de difficulté.

```
Entrée [26]: def remove_numbers(grid, difficulty):
    # Retirer des chiffres en fonction de la difficulté
    if difficulty == 'débutant':
        to_remove = 30
    elif difficulty == 'moyen':
        to_remove = 40
    elif difficulty == 'avancé':
        to_remove = 50
    else:
        print("Difficulté non reconnue. Utilisation de la difficulté moyenne par défaut.")
        to_remove = 40
    cells = [(i, j) for i in range(9) for j in range(9)]
    random.shuffle(cells)
    for i, j in cells:
        if to_remove == 0:
            break
        if grid[i][j] != 0:
            grid[i][j] = 0
            to_remove -= 1
    return grid

def print_grid(grid):
    for i, row in enumerate(grid):
        print(*row[:3], "|", *row[3:6], "|", *row[6:])
        if (i + 1) % 3 == 0 and i != 8:
            print("-" * 21)

difficulty = input("Choisissez la difficulté (débutant, moyen, avancé) : ").lower()
grid = generate_sudoku()
remove_numbers(grid, difficulty)
print("Voici votre grille de sudoku :")
print_grid(grid)
```

Choisissez la difficulté (débutant, moyen, avancé) : débutant
Voici votre grille de sudoku :

1	6	2	0	0	5	8	3	7
0	4	0	8	3	0	1	6	0
8	3	0	0	6	0	9	4	5

6	0	9	4	5	8	3	7	0
0	0	0	3	0	1	6	2	9
0	7	1	6	0	0	4	0	0

2	9	0	0	8	3	0	0	6
0	0	3	7	1	6	0	9	4
7	1	6	0	9	0	0	8	3

Choisissez la difficulté (débutant, moyen, avancé) : moyen
Voici votre grille de sudoku :

0	7	0	2	0	5	0	3	0
0	0	0	6	3	1	0	7	0
0	7	1	6	0	0	4	0	0

2	9	0	0	8	3	0	0	6
0	0	3	7	1	6	0	9	4
7	1	6	0	9	0	0	8	3

0	0	0	0	5	0	0	1	0
0	5	6	3	0	0	7	0	0
3	1	8	0	9	0	4	0	6

9	2	0	0	0	0	0	0	7
0	6	3	1	8	0	9	0	4
1	8	7	9	2	0	5	0	3

3	2	9	0	0	7	0	0	0
1	6	0	0	0	5	0	2	0
0	4	5	0	0	0	0	6	7

2	0	1	0	0	0	4	5	0
6	7	0	0	5	3	0	0	0
4	0	0	0	9	0	0	0	8

choisissez la difficulté (débutant, moyen, avancé) : avancé
Voici votre grille de sudoku :

0	3	0	0	1	0	0	0	4
0	0	0	0	8	0	0	3	0
7	0	4	0	3	0	0	0	0

3	2	9	0	0	7	0	0	0
1	6	0	0	0	5	0	2	0
0	4	5	0	0	0	0	6	7

2	0	1	0	0	0	4	5	0
6	7	0	0	5	3	0	0	0
4	0	0	0	9	0	0	0	8

Choisissez la difficulté (débutant, moyen, avancé) : test
Difficulté non reconnue. Utilisation de la difficulté moyenne par défaut.

Voici votre grille de sudoku :

7	0	0	0	0	9	0	1	5	8
0	3	9	0	1	5	0	7	0	4
1	0	0	0	0	0	4	0	0	0

0	4	2	0	0	1	0	5	0	7
3	0	0	0	8	7	0	0	2	0
5	0	0	0	0	0	0	3	9	0

4	2	3	0	9	1	0	0	8	7	6
0	0	5	0	0	7	6	0	4	0	3
8	0	0	0	0	0	0	3	0	0	5

Appel du sudoku solver que l'on a crée précédemment

```
Entrée [81]: def find_empty_location(grid):  
  
    #Fonction pour trouver la première case vide dans le tableau.  
  
    for l in range(9):  
        for c in range(9):  
  
            if grid[l][c] == 0:  
                return l, c  
  
    return None, None # Retourne None si aucune case vide n'est trouvée
```

```
Entrée [82]: def is_valid(grid, row, col, num):  
  
    for l in range(9):  
        if grid[row][l] == num:  
            return False  
  
    for c in range(9):  
        if grid[c][col] == num:  
            return False  
  
    start_row, start_col = 3 * (row // 3), 3 * (col // 3)  
    for l in range(3):  
        for c in range(3):  
            if grid[l + start_row][c + start_col] == num:  
                return False  
  
    return True
```

```
Entrée [83]: def solve_sudoku(grid):  
  
    row, col = find_empty_location(grid)  
  
    #Si aucune case vide n'est trouvée  
    if row is None:  
        return True  
  
    #S'il trouve un 0, le remplace par la 1ere valeur  
    for num in range(1, 10):  
        if is_valid(grid, row, col, num):  
            grid[row][col] = num  
            print("Chiffre", num, "en", (row, col))  
            print_sudoku(grid)  
            print("\n")  
  
            if solve_sudoku(grid): #Récursivité  
                return True  
  
    #Si la valeur ne fonctionne pas, reviens en arrière pour tenter avec la seconde valeur  
    #algo de Backtraking  
    print("Retour en", (row, col), "pour tester le chiffre suivant.")  
    print_sudoku(grid)  
    print("\n")  
    grid[row][col] = 0 #Annule la 1ere valeur et la remplace par la suivante  
  
    return False  
  
solve()
```

Lancement du sudoku avec la difficulté choisie et si on veut ou non le résoudre

```
Entrée [85]: def create_sudoku(grid):
    print("Choisissez un niveau de difficulté : débutant, moyen ou avancé.")
    while True:
        difficulty = input("Niveau de difficulté : ").strip().lower()
        if difficulty in {"débutant", "moyen", "avancé"}:
            break
        else:
            print("Niveau de difficulté invalide. Veuillez choisir parmi débutant, moyen ou avancé.")

    grid_creator = generate_sudoku()
    grid_creator = remove_numbers(grid_creator, difficulty)
    print("Voici votre grille de Sudoku à résoudre :")
    print_sudoku(grid_creator)

    while True:
        choice = input("Résoudre le Sudoku ? (oui/non) : ").strip().lower()
        if choice == "oui":
            solve_sudoku(grid_creator)
            print("Voici la solution :\n")
            print_sudoku(grid_creator)
            break
        elif choice == "non":
            print("Pas de sudoku du coup. Ciao !")
            break
        else:
            print("Bad réponse, réponds par 'oui' ou par 'non'.")
```

```
create_sudoku(grid)
```

Choisissez un niveau de difficulté : débutant, moyen ou avancé.

Niveau de difficulté :

Choisissez un niveau de difficulté : débutant, moyen ou avancé.

Niveau de difficulté : débutant

Voici votre grille de Sudoku à résoudre :

3	8	6	0	2	0	0	0	9
4	2	0	0	7	0	3	8	6
1	0	0	3	8	0	4	2	5

0	6	0	0	5	1	7	9	0
2	5	1	7	9	3	8	6	0
0	9	0	8	0	4	0	0	1

6	4	0	5	1	0	9	0	8
0	1	7	9	0	8	6	4	2
0	3	8	6	0	2	0	0	0

Résoudre le Sudoku ? (oui/non) :

Résoudre le Sudoku ? (oui/non) : non

Pas de sudoku du coup. Ciao !

chiffre 5 en (0, 5)

3	8	6	1	2	5	0	0	9
4	2	0	0	7	0	3	8	6
1	0	0	3	8	0	4	2	5

0	6	0	0	5	1	7	9	0
2	5	1	7	9	3	8	6	0
0	9	0	8	0	4	0	0	1

6	4	0	5	1	0	9	0	8
0	1	7	9	0	8	6	4	2
0	3	8	6	0	2	0	0	0

Retour en (0, 5) pour tester le chiffre suivant.

3	8	6	1	2	5	0	0	9
4	2	0	0	7	0	3	8	6
1	0	0	3	8	0	4	2	5

0	6	0	0	5	1	7	9	0
2	5	1	7	9	3	8	6	0
0	9	0	8	0	4	0	0	1

6	4	0	5	1	0	9	0	8
0	1	7	9	0	8	6	4	2
0	3	8	6	0	2	0	0	0

chiffre 7 en (8, 8)

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

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

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

Voici la solution :

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

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

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

Choisissez un niveau de difficulté : débutant, moyen ou avancé.

Niveau de difficulté : nope

Niveau de difficulté invalide. Veuillez choisir parmi débutant, moyen ou avancé.

Niveau de difficulté : moyen

Voici votre grille de Sudoku à résoudre :

0	0	2	0	0	5	8	0	3
4	1	0	8	7	3	0	9	2
0	0	0	0	0	2	0	0	5

9	0	0	1	0	8	7	0	0
0	5	0	0	0	6	9	2	4
0	3	0	9	2	0	0	5	8

0	0	1	0	8	0	3	0	9
0	0	0	3	6	0	2	4	1
3	0	9	2	4	0	0	8	0

Résoudre le Sudoku ? (oui/non) : nope

Bad réponse, réponds par 'oui' ou par 'non'.

Résoudre le Sudoku ? (oui/non) :



place
ig-c
e ye
ich l
berwe
tencio

Verification du sudoku une fois résolue

```
Entrée [ ]: def sudoku_checker(grid):
    return ligne_checker(grid) and colonne_checker(grid) and carre_checker(grid)

print(sudoku_checker(grid))
```

Pygame

par ChatGPT (qu'on remercie)



```
1> import pygame
2  import sys
3
4  # Initialize Pygame
5  pygame.init()
6
7  # Define colors
8  WHITE = (255, 255, 255)
9  BLACK = (0, 0, 0)
10 RED = (255, 0, 0)
11 GREEN = (0, 255, 0)
12
13 # Define window size
14 width, height = 600, 600
15 cell_size = 60
16
17 # Example Sudoku grid (0 represents an empty cell)
18 > grid = [ ...
29
30 # Create game window
31 window = pygame.display.set_mode((width, height))
32 pygame.display.set_caption("Sudoku")
33
34 # Variable to keep track of win state
35 win = False
```

```
37 # Function to draw Sudoku grid
38 > def draw_grid():
39     for i in range(0, 10):
40         if i % 3 == 0:
41             thickness = 8
42         else:
43             thickness = 3
44         pygame.draw.line(window, BLACK, (0, i * cell_size), (width, i * cell_size), thickness)
45         pygame.draw.line(window, BLACK, (i * cell_size, 0), (i * cell_size, height), thickness)
46
47 # Function to display numbers on the grid
48 > def display_numbers():
49     font = pygame.font.SysFont(None, 40)
50     for i in range(9):
51         for j in range(9):
52             if grid[i][j] != 0:
53                 text = font.render(str(grid[i][j]), True, BLACK)
54                 window.blit(text, (j * cell_size + 20, i * cell_size + 15))
55
56 # Function to check if the grid is correctly solved
57 > def is_solved():
58     for i in range(9):
59         for j in range(9):
60             if grid[i][j] == 0:
61                 return False
62     return True
63
```

```
64 # Function to check if a number can be placed in an empty cell
65 def is_valid(row, col, num):
66     for i in range(9):
67         if grid[row][i] == num or grid[i][col] == num:
68             return False
69
70     square_row = row // 3
71     square_col = col // 3
72     for i in range(square_row * 3, square_row * 3 + 3):
73         for j in range(square_col * 3, square_col * 3 + 3):
74             if grid[i][j] == num:
75                 return False
76
77     return True
```

```
78 # Function to allow user input in an empty cell
79 def input_number(row, col):
80     font = pygame.font.SysFont(None, 40)
81     input_active = True
82     while input_active:
83         for event in pygame.event.get():
84             if event.type == pygame.KEYDOWN:
85                 if event.unicode.isdigit() and is_valid(row, col, int(event.unicode)):
86                     grid[row][col] = int(event.unicode)
87                     input_active = False
88                 elif event.key == pygame.K_BACKSPACE:
89                     grid[row][col] = 0
90                     input_active = False
91                 elif event.key == pygame.K_RETURN:
92                     input_active = False
93
94         pygame.draw.rect(window, RED, (col * cell_size + 5, row * cell_size + 5, cell_size - 10, cell_size - 10))
95         pygame.display.flip()
96
97         window.blit(font.render(str(grid[row][col]), True, BLACK), (col * cell_size + 20, row * cell_size + 15))
98         pygame.display.flip()
99
100        pygame.time.delay(100)
101
102    # Function to display victory message
103 def display_win_message():
104     font = pygame.font.SysFont(None, 50)
105     text = font.render("Victory!", True, GREEN)
106     window.blit(text, (200, 250))
107     pygame.display.flip()
108
```



```
109 # Main game function
110 def play_sudoku():
111     global win
112     while True:
113         for event in pygame.event.get():
114             if event.type == pygame.QUIT:
115                 pygame.quit()
116                 sys.exit()
117             elif event.type == pygame.MOUSEBUTTONDOWN:
118                 x, y = pygame.mouse.get_pos()
119                 row = y // cell_size
120                 col = x // cell_size
121                 if grid[row][col] == 0:
122                     input_number(row, col)
123                     if is_solved():
124                         win = True
125
126                     window.fill(WHITE)
127                     draw_grid()
128                     display_numbers()
129
130             if win:
131                 display_win_message()
132
133             pygame.display.flip()
134
135 # Start the game
136 if __name__ == "__main__":
137     play_sudoku()
```

Sudoku

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

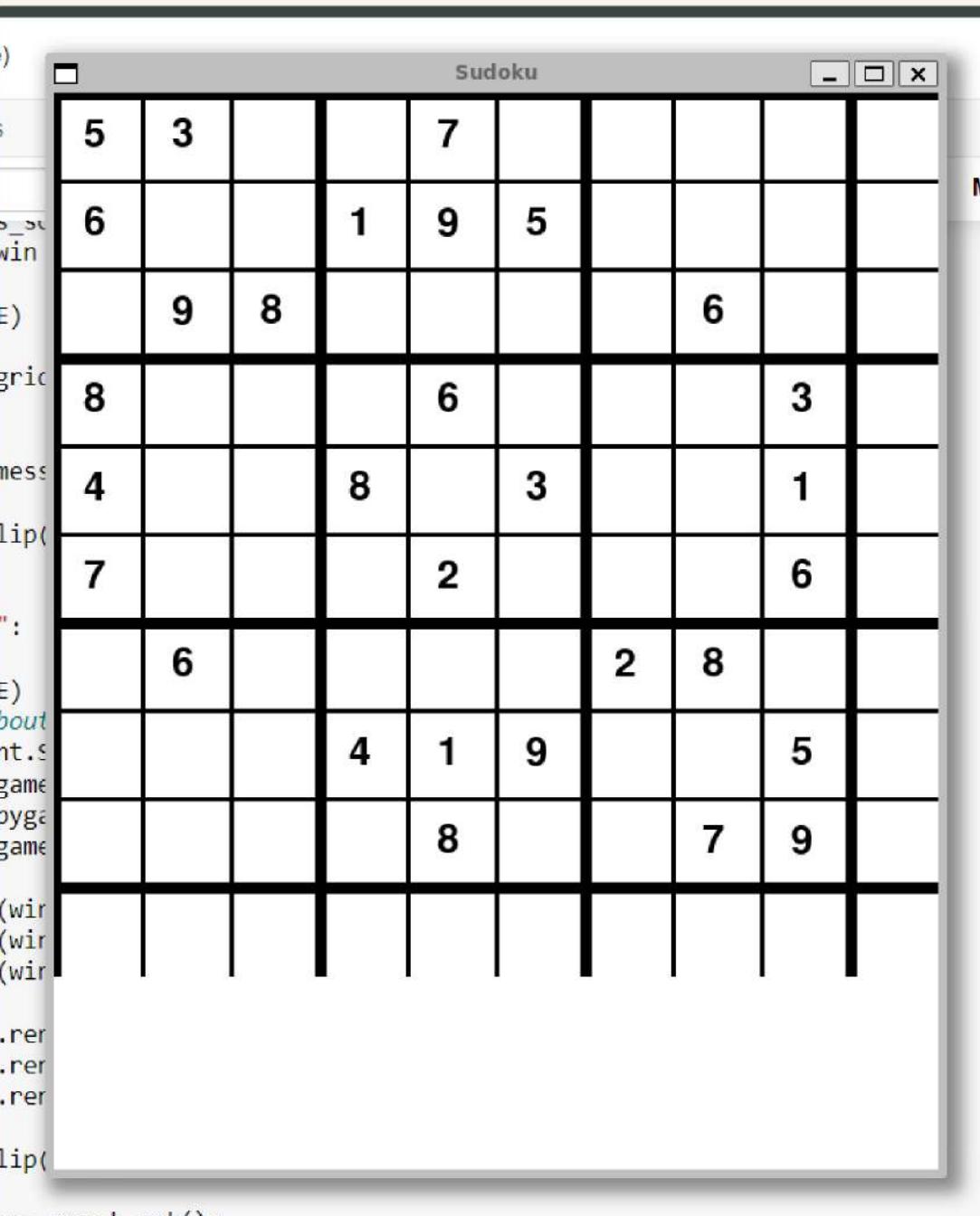
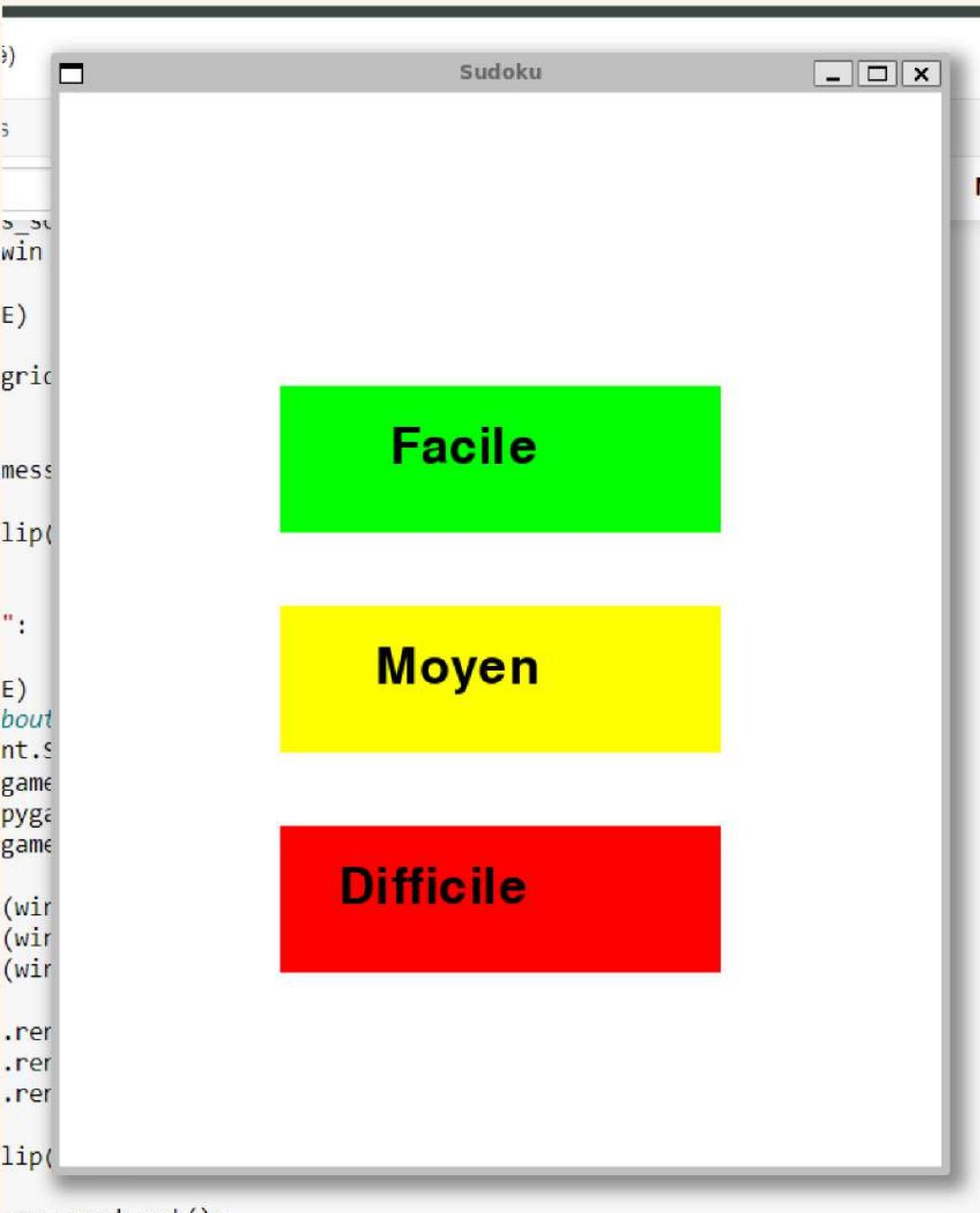
si la grille est correctement résolue

Sudoku

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

Victory!

démo



Thank You

par Anne & Lou