TD1 Anis Bougherbal 2A

Projet Sudoku

Objectif : Mise en place d'une application permettant de jouer au sudoku

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

- 1. Introduction

Ce rapport a pour objectif de présenter de manière détaillée mon travail sur la création d'une application de sudoku complète avec le language de programmation python et le module tkinter. Malgré les fonctionnalités assez limité de tkinter en design, j'ai réussi a créer une application de sudoku,allant de l'authentification des joueurs, à la génération de la grille jusqu'à la gestion du classement des joueurs.

L'objectif principal de cette application est de fournir une expérience utilisateur fluide et agréable, avec plusieurs fonctionnalités clés :

- **Authentification des joueurs** : Permet aux utilisateurs de créer un compte et de se connecter pour suivre leurs progrès.
- **Génération de grilles** : Proposer des grilles de sudoku de différents niveaux de difficulté.
- **Sauvegarde et reprise** : Offrir la possibilité aux joueurs de sauvegarder leur progression et de reprendre une partie plus tard.
- **Gestion du classement** : Mettre en place un système de classement pour encourager la compétition amicale entre les joueurs.

Fonctionnalités Principales

L'application propose les fonctionnalités suivantes :

- **Inscription et connexion** : Un système sécurisé d'authentification des utilisateurs.
- **Sélection de la difficulté** : Les joueurs peuvent choisir entre plusieurs niveaux de difficulté (facile, moyen, difficile).
- **Interface utilisateur** : Une interface intuitive pour saisir et modifier les chiffres de la grille.
- **Vérification des solutions** : Un mécanisme pour vérifier si la solution proposée par le joueur est correcte.
- **Sauvegarde automatique** : Enregistrement automatique de la progression du joueur pour permettre une reprise facile de la partie.
- **Classement des joueurs** : Un tableau de classement basé sur les performances des joueurs, notamment le temps de résolution des grilles et la précision.

Cette application vise à offrir une plateforme pour les amateurs de sudoku, en leur permettant de suivre leurs progrès via le score, de rivaliser avec d'autres joueurs et de continuer à améliorer leurs compétences en résolution de sudoku 9x9. En facilitant l'accès à des grilles de sudoku variées et en proposant des fonctionnalités de sauvegarde et de classement.

- 2. Algorithme de résolution

Pour la résolution de la grille de sudoku, j'ai utilisé un algorithme de backtracking, inspiré par cette vidéo explicative. Le backtracking est une méthode de recherche systématique qui explore toutes les possibilités pour trouver une solution, tout en respectant les règles du jeu. L'algorithme commence par identifier les cellules vides dans la grille. Pour chaque cellule vide, il détermine les valeurs possibles qui peuvent y être placées sans provoquer de doublons dans la ligne, la colonne ou la sous-grille 3x3.

Pour connaître toutes les valeurs pouvant être placées dans une case donnée, j'ai créé une fonction `valeurs_possibles` qui vérifie l'absence de doublons dans la ligne, la colonne et la sous-grille 3x3 correspondantes. Cette fonction renvoie une liste de toutes les valeurs possibles pour une case donnée. Ensuite, l'algorithme place la première valeur possible dans la cellule et passe à la cellule suivante. Si aucune valeur valide ne peut être placée dans une cellule donnée, l'algorithme revient en arrière (backtracking) pour essayer une autre valeur. Ce processus récursif continue jusqu'à ce que toutes les cellules soient remplies correctement.

Les principales méthodes de la classe `Solver` incluent les vérifications des doublons dans les lignes, colonnes et sous-grilles, ainsi que la méthode principale de résolution qui implémente l'algorithme de backtracking. L'algorithme de backtracking étant récursif, il ne peut pas renvoyer directement la grille résolue ni le nombre de solutions pour cette grille. Cela implique que pour générer une grille jouable, il nous faudra générer la grille déjà complétée puis retirer un certain nombre de chiffres tout en vérifiant que cette grille ne possède qu'une seule solution.

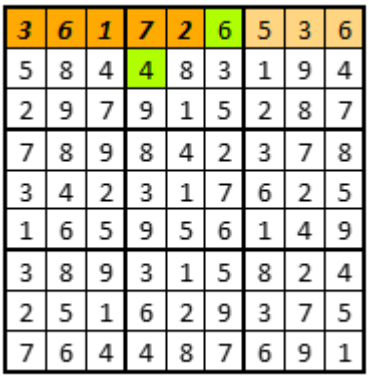
Pour pouvoir extraire le nombre de solutions d'une grille tout en gardant mon algorithme récursif, j'ai mis en place une variable globale dans l'objet Solver qui est incrémentée de 1 chaque fois que l'algorithme de backtracking trouve une solution pour la grille. Un "getter" permet ensuite de récupérer cette variable `nb_solu`. Cette variable est indispensable pour la génération de grilles jouables car elle permet de savoir si une grille ne possède qu'un seule solution..

- 3.L'algorithme de génération

Pour la génération de la grille complète, j'ai utilisé l'algorithme trouvé par mfgravesjr.
et codé en java.Voici comment fonctionne cet algorithme :

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

Je commence par créer un tableau de longueur 9x9 avec des subdivisions 3x3 contenant chacune des chiffres aléatoires de 1 à 9 . Ainsi, toutes les sous-grilles sont pré-remplies de manière aléatoire, mais chaque ligne et chaque colonne contiennent encore des valeurs en double, ce qui n'est pas autorisé dans un jeu de Sudoku.



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

Je vais donc essayer de supprimer tous les doublons comme suit : je vais commencer par parcourir la première ligne en mémorisant les chiffres parcouru via une liste. Si je tombe sur un chiffre que j'ai déjà croisé, je vais chercher dans sa subdivision 3x3 un chiffre qui n'a pas été croisé dans la ligne. Si j'arrive a trouver un chiffre qui n'a pas été croisé, je peux l'échanger avec le doublon. Je vais ensuite faire la même
chose avec les colonnes.

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

Nous rencontrons un cas exceptionnel : nous essayons de trier la colonne 4, mais il y a un doublon de 8. Le problème est que le 8 fait partie de la ligne 4, qui a déjà été triée. Il est crucial de ne pas désorganiser cette ligne en échangeant le doublon avec une quelconque valeur de la subdivision 3x3, car cela risquerait de la déstructurer à nouveau. Par conséquent, il faut échanger les doublons des lignes ou colonnes déjà triées avec leurs voisins (ici, 4 ou 2) afin de préserver l'organisation de la ligne. On remarque que 4 n'a jamais été croisé, donc nous pouvons échanger 8 avec 4.

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

Mais que se passe-t-il si tous les chiffres dans la subdivision d'un doublon ont déjà été croisés ? C'est le cas ici : 7 est un doublon et sa colonne n'est pas encore triée. Nous cherchons donc à échanger ce doublon avec un chiffre de sa subdivision 3x3 qui n'a pas encore été triée (8, 4 ou 3). Attention, nous ne pouvons pas échanger 7 avec les chiffres au-dessus de lui car cette ligne a déjà été triée.

Cependant, aucun chiffre échangeable dans la subdivision 3x3 n'a été croisé. Dans ce cas, nous cherchons dans la subdivision de la première occurrence du doublon, toujours en veillant à ne pas échanger le doublon avec un chiffre d'une colonne déjà triée. Les possibilités d'échange sont ici 5 ou 6. Nous remarquons que 5 n'a jamais été croisé, donc nous pouvons l'échanger avec 7. La ligne est maintenant triée.

*

Il arrive parfois de rencontrer un doublon qui ne possède aucun chiffre non croisé dans sa subdivision 3x3, et dont la première occurrence ne possède également aucun chiffre non croisé échangeable dans sa propre subdivision. Dans ce cas, nous devons forcer les échanges jusqu'à obtenir un chiffre non croisé. Voici comment nous procédons : nous revenons à la première occurrence et échangeons ce chiffre avec son voisin direct, qui est logiquement un chiffre déjà croisé, devenant ainsi un nouveau doublon. Ensuite, nous revenons à la première occurrence de ce nouveau doublon et vérifions si son voisin direct a déjà été croisé. Si c'est le cas, nous les échangeons et recommençons.

Lorsque l'algorithme trouve un chiffre qui n'a jamais été croisé, il effectue l'échange et s'arrête.

Il y a un dernier problème lors de la génération de la grille, il arrive rarement que l'algorithme soit incapable d'échanger des cellules uniques sans entrer dans une boucle infinie, un problème qui se produit rarement mais qui arrive tout de même. Pour éviter cela, après 81 itérations infructueuses de la boucle, une dernière stratégie est utilisée. Cette stratégie consiste à passer à la colonne suivante pour effectuer les tris nécessaires avant de revenir à la colonne initiale pour réessayer les stratégies de tri classiques. Vous remarquerez que je ne parle ici que de colonnes car j'utilise cette dernière stratégie uniquement sur ces dernières.

En effet, les lignes étant triées en premier, je n'ai pas besoin d'en arriver à cette dernière stratégie car elles sont déjà correctement triées avec l'avant dernière stratégie et ne nécessitent pas de réarrangement supplémentaire.

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

Une fois toutes ces stratégies effectuées, nous devrions obtenir une grille complète avec une seule solution.

Cet algorithme m'a pris énormément de temps à mettre en place, car le code source initial était en Java et retrouver les différentes stratégies dans son code s'est avéré difficile. J'ai souvent dû tout reprendre à zéro. Cependant, j'ai opté pour cet algorithme de génération car il semblait être très rapide et moins gourmand en ressources que simplement utiliser un algorithme de force brute. Essayons maintenant de déterminer combien de temps nous mettons à générer 1000 grilles solvables.

```python
import time
from Generator import Generator

if __name__ == "__main__":
    generator = Generator()
    start_time = time.time()

    # Génération de 1000 grilles
    for _ in range(1000):
        grille = generator.generer(9)

    end_time = time.time()
    duration = end_time - start_time
    print("Temps nécessaire pour générer 1000 grilles de Sudoku :", duration, "secondes")
```

```
Temps nécessaire pour générer 1000 grilles de Sudoku : 0.17868566513061523 secondes
```

On peut voir que notre algorithme génère 1000 grilles en 0.17 seconde, ce qui nous donne environ 5882 grilles par seconde. Cet algorithme est donc très rapide et peut donc tout à fait être utilisé pour entraîner une intelligence artificielle, par exemple.

- 3. Algorithme d'authentification

Pour l'authentification, j'ai adapté des systèmes d'authentification trouvés sur YouTube, comme celui de Technologie informatique Parvat

Une fois le programme lancé, l'utilisateur a le choix entre se connecter ou créer un nouvel utilisateur. Un fichier stocke toutes les informations de l'utilisateur créé : le nom, le mot de passe, son score, et toutes les grilles qu'il a déjà commencées. Une fois toutes les informations inscrites dans ce fichier au nom de l'utilisateur, nous allons créer le dictionnaire permettant de gérer toutes ces informations dans le jeu lui-même.

Voici comment se présente le dictionnaire :

- `"User"` : nom du joueur
- `"Pass"` : le mot de passe
- `"Score"` : le score
- `1… i` : un tableau avec les informations de chaque grille.

Le tableau se présente comme suit :

- le niveau (1 à 3)
- la dimension de la grille
- la grille dans l'état actuel
- la réponse de cette grille
- la grille initiale
- le temps écoulé

Ce dictionnaire va nous permettre de connaître toutes les informations du joueur et sera souvent mis à jour en fonction des actions de l'utilisateur grâce aux fonctions 'file_in_dico', qui permet de mettre à jour le dictionnaire en fonction des informations dans le fichier source, et 'Supprimer_grille', qui supprime une grille du dictionnaire.

- 4. Le menu

Le menu, qui prend en paramètre le dictionnaire de l'utilisateur et qui possède donc toutes les informations concernant ce dernier, nous offre plusieurs possibilités. Il permet bien évidemment de créer une grille de sudoku tout en choisissant la difficulté de cette dernière. Une fois la grille lancée, le menu se ferme et se rouvre après la fermeture de la grille. Il en va de même pour les grilles déjà entamées, ce qui permet de ne pas avoir à rafraîchir le menu à chaque fois via le menu déroulant en haut à gauche de la page.

Pour ouvrir une grille déjà entamée, le menu utilise un système d'indices. En effet, chaque grille sauvegardée est indexée dans le dictionnaire des informations de l'utilisateur, et le menu affiche un bouton pour chaque grille existante, indiquant le niveau de difficulté et le pourcentage de complétion. Ce pourcentage est calculé grâce à l'état de la grille la dernière fois que l'utilisateur l'a sauvegardée et grâce à l'état initial de la grille, tous deux stockés dans le dictionnaire. L'utilisateur peut alors sélectionner une grille pour la reprendre ou choisir de la supprimer en utilisant un menu contextuel accessible par un clic droit. Il est important de noter qu'il n'est pas possible d'avoir plus de cinq grilles entamées pour un utilisateur et que le menu adapte automatiquement les dimensions de la page en fonction du nombre de grilles entamées. Ce système permet une gestion fluide et efficace des différentes parties en cours, tout en offrant un retour immédiat sur l'avancement de chaque grille.

Le menu offre également un classement de tous les utilisateurs en fonction de leur score. Cela permet aux utilisateurs de se situer par rapport aux autres et d'instaurer un esprit de compétition nécessaire à l'amélioration. Une fois le classement demandé, l'application va chercher chaque nom d'utilisateur via les fichiers Users, stocke le nom et le score dans une liste, trie cette liste en fonction du score et affiche le classement sur la page.

- 5. Le l'algorithme de jeu

Concernant le jeu sudoku en lui-même, j'ai décidé de ne pas inclure les grilles de sudoku autres que les grilles de taille 9x9 pour deux raisons. La première est que mon algorithme de génération étant plutôt complexe, il était difficile de le modifier pour générer des grilles non carrées (9x8 par exemple).
La deuxieme est la difficulté de gerer l'équité de l'attribution des points en fonction de la difficulté et de la dimension, comment garantir un classement equitable avec des grilles de dimension différentes ? Faut il faire un classement pour chaque categorie ?

Pour générer la grille une fois créée par notre générateur, nous allons utiliser des widgets `Entry` qui permettent à l'utilisateur d'entrer ou de modifier une ligne de texte. Chaque case de la grille de sudoku est représentée par un widget `Entry`. Voici comment cela fonctionne :

Chaque `Entry` est configuré pour avoir une largeur de 5, une couleur de fond définie par `bgcolor`, un texte centré. Chaque widget est positionné sur une grille en utilisant la méthode `grid`, et un léger espacement (padding) est ajouté autour de chaque case pour une meilleure lisibilité.
Chaque case de cette grille est remplie avec la valeur "val" extrait du dictionnaire de l'utilisateur.
Toutes les cases qui ne sont pas vides dans la grille initial sont configurés en état "`disabled`" car ces dernières sont les cases pré remplis et ne peuvent pas être modifiés.

Vous remarquerez les deux lignes suivantes dans le code :

```
e.bind("<Button-1>", lambda event, r=rowNo+i, c=colNo+j+1:
cell_clicked(cells, c, r, bgcolor))
```

et

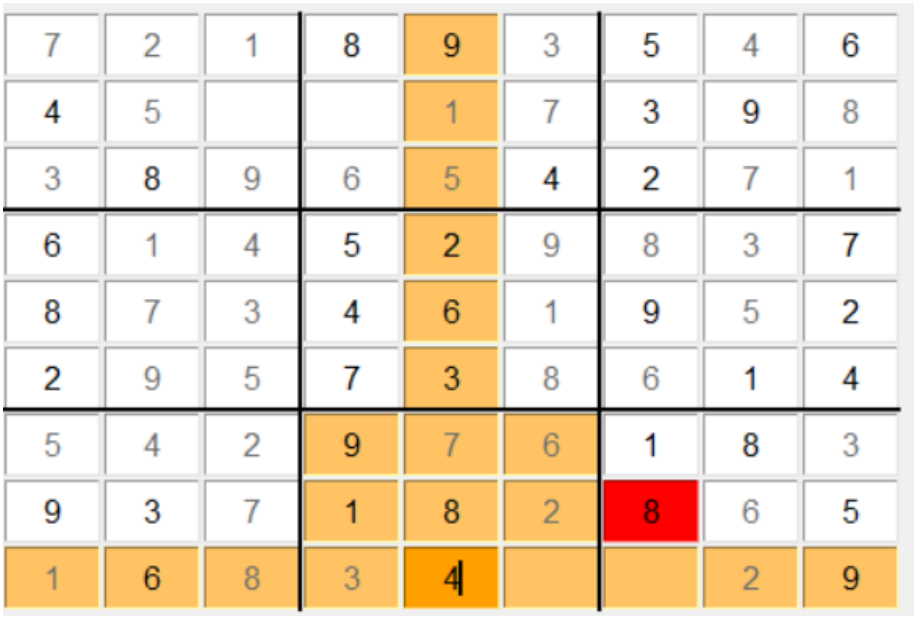
```
e.bind("<KeyRelease>", lambda event, r=rowNo+i, c=colNo+j+1:
color_entry(cells, r, c))
```

La première fonctionnalité permettra de changer la couleur de la case sélectionnée, mais également de changer la couleur de toutes les cases de sa ligne, de sa colonne et de sa subdivision 3x3 pour une bien meilleure lisibilité de la grille. Cette fonctionnalité fonctionne en coloriant les cases concernées et en enregistrant les cases déjà sélectionnées dans une liste appelée "last_click". Ainsi, lorsque je sélectionne une nouvelle case, les cases précédemment coloriées seront re-coloriées en blanc pour colorier les nouvelles cases.

La deuxième fonction se nomme `color_entry`, et comme son nom l'indique, elle s'exécute lorsque j'entre un chiffre dans cette case. Cette dernière va vérifier si le chiffre entré est une valeur possible pour cette case, c'est-à-dire s'il y a des doublons dans la ligne, la colonne ou la subdivision. Si c'est le cas, la case concernée s'allume en rouge et le reste. Il est

important de noter qu'ici, il ne s'agit pas de vérifier si la valeur entrée correspond à la bonne réponse, mais plutôt de vérifier si ce chiffre peut être placé là selon les règles du sudoku.

Nous pouvons nous déplacer à travers la grille en utilisant les flèches du clavier grâce à la fonction 'navigate_grid'. Nous obtenons donc cette grille, nous pouvons changer de case grâce a la souris ou aux flèches du clavier.



De plus, un menu déroulant est disponible en haut à gauche avec un bouton 'sauvegarder et quitter' pour ceux qui pourraient être sceptiques. Cependant, la grille et le temps écoulé se sauvegardent automatiquement dans le fichier de l'utilisateur dès qu'il fermera la page de la grille.
Cette sauvegarde automatique est géré par cette ligne:

```
game.protocol("WM_DELETE_WINDOW", lambda: on_close(info_dico, i,
elapsed_time))
```

Elle exécute la fonction 'on_close' lorsque je ferme la page.

Une fois le bouton "Vérifier" pressé, le programme verifie que la grille actuel correspond bien a la réponse et attribue le score en fonction du temps passé a la résolution de la grille.
Voici le barème de point :

Pour les grilles faciles , si le joueur termine la grille en moins de 5 minutes (300 secondes), il obtient un score de 100 points. Entre 5 et 10 minutes (301 à 600 secondes), le joueur reçoit 50 points, et au-delà de 10 minutes (plus de 600 secondes), le score est de 25 points.

Pour les grilles de difficulté moyenne, si le joueur résout la grille en moins de 9 minutes et 10 secondes (550 secondes), il obtient 100 points. Entre 9 minutes et 10 secondes et 12 minutes et 30 secondes (551 à 750 secondes), le score est de 50 points. Au-delà de 12 minutes et 30 secondes (plus de 750 secondes), le joueur reçoit 25 points.

Enfin, pour les grilles difficiles, moins de 10 minutes (600 secondes) rapportent 100 points, entre 10 et 15 minutes (601 à 900 secondes) rapportent 50 points, et au-delà de 15 minutes (plus de 900 secondes) rapportent 25 points.

- Conclusion

Le développement de cette application Sudoku a été un projet enrichissant qui a permis de combiner des compétences en programmation, en conception d'interface utilisateur, et en gestion de données.

Ce projet ouvre également la voie à de futures améliorations et fonctionnalités, telles que des grilles de différentes dimensions, des modes multijoueurs, des défis quotidiens, ou des modes assistés.
En somme, ce projet a été très enrichissant et j'espère qu'elle stimulera comme il se doit l'esprit logique et la capacité de résolution de problème des utilisateurs.

-Bibliographie

https://nsi.xyz/projets/solveur-de-sudoku-en-python/

https://www.developpez.net/forums/d1977902/autres-langages/python/programmation-multimedia-jeux/creer-grille-sudoku/

https://www.youtube.com/watch?v=ibf5cx221hk&pp=ygUTZ3VpIHB5dGhvbiB0a2ludGVyIA%3D%3D

https://www.youtube.com/watch?v=oLxFqpUbaAE&pp=ygUTZ3VpIHB5dGhvbiB0a2ludGVyIA%3D%3D

https://www.youtube.com/watch?v=G_UYXzGuqvM&pp=ygUXcmVzb2x2ZSBhIHN1ZG9rdSBweXRvbiA%3D

https://www.youtube.com/watch?v=eqUwSA0xI-s&t=2s&pp=ygUXcmVzb2x2ZSBhIHN1ZG9rdSBweXRvbiA%3D

https://www.youtube.com/watch?v=LHCHH5siBCg&pp=ygUVZ2VuZXJhdGUgc3Vkb2t1IGdyaWQg

https://www.youtube.com/watch?v=iSdW8OM_b3E&t=236s&pp=ygUVZ2VuZXJhdGUgc3Vkb2t1IGdyaWQg

https://www.youtube.com/watch?v=Xw0xxvyxFuE&t=881s&pp=ygUVZ2VuZXJhdGUgc3Vkb2t1IGdyaWQg

https://www.youtube.com/watch?v=9_0fhHw1l6E&pp=ygUVZ2VuZXJhdGUgc3Vkb2t1IGdyaWQg