BURGY Corentin
PEREIRA Raphaël
DELAUNE-LORIETTE Mathéo
VIALLETON Rémi

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Description des méthodes de création du Sudoku :

La méthode *grilleSolution*

Cette méthode permet de créer une grille de sudoku conforme sous forme de tableau. Elle est utilisée plusieurs fois à chaque fois que l'utilisateur lance une partie et elle permet d'obtenir la grille solution à partir de laquelle on va construire notre grille de sudoku à remplir.

Code :

```
261     def grilleSolution(self, grille):
262         """
263         Algorithme de backtracking permettant de créer une grille de solution. Elle prend en paramètre une grille solution vide d'abord.
264         Le principe de cet algorithme est que pour chaque case, on choisit une valeure qui pourrait convenir, on explore jusqu'au bout si cette
265         valeur convient vraiment, et si elle ne convient pas, on remonte pour la changer.
266         Returns
267         -------
268         False tant qu'aucune solution n'est trouvé, True quand il n'y a plus de case vide (solution trouvée)
269 
270         Params
271         -------
272         Grille
273 
274         """
275         possible = [1, 2, 3, 4, 5, 6, 7, 8, 9]
276 
277         for i in range(9):
278             for case in grille[i]:
279                 if case.valeur == 0:
280                     shuffle(possible)
281                     for nombre in possible:
282                         if self.nombreValide(case, nombre, self.adjacence_solution):
283                             case.valeur = nombre
284                             if self.grilleRemplie(grille):              #Une fois que la grille est remplie (plus de 0), la fonction renvoit True
285                                 return True
286                             else:
287                                 if self.grilleSolution(grille):           #Si la grille n'est pas complète, on répète l'opération jusqu'à ce qu'ell
288                                     return True
289 
290                     break           #Les instructions de break et continue servent à arrêter les deux boucles for en même temps pour re-parcourir l'
291             else:
292                 continue
293             break
294 
295 
296         case.valeur=0
297         #La ligne précèdente est importante pour éviter les solutions où l'algorithme parcours toute la grille en trouvant une solution partielle
298         #(contenant des 0). Lorsque cela se produit la fonction récursive renvoit False et il faut ajouter la ligne précédente pour mettre à 0 la de
299         return False
```

Le fonctionnement de cette méthode est basé sur celui d'un algorithme de backtracking (retour sur trace). Ce type d'algorithme, particulièrement utiles pour la résolution de problèmes de satisfaction de contraintes, permet de trouver une solution en utilisant la récursivité pour construire une solution petit à petit en revenant en arrière dès que les contraintes ne sont plus satisfaites. Cet algorithme est aussi utilisé dans la méthode *resoudreSudoku* décrite en commentaire directement dans le programme.

Quand on appelle la méthode, elle parcourt la grille jusqu'à trouver une case vide. Une fois cette case vide sélectionnée, on essaye de lui entrer une valeur au hasard parmi celles possible selon les règles du sudoku. Si la grille est remplie, on finit l'algorithme, sinon on continue de chercher de nouvelles cases vides en rappelant la fonction. Si, au bout d'un moment, on ne trouve aucune valeur possible pour une case vide, on remonte au choix précédent et on essaye la valeur possible suivante et on remonte ainsi de suite si on ne trouve toujours pas de valeur possible.

La fonction prend en paramètre une grille correspondant à un tableau contenant des instances de la classe *Case*. Cette classe, servant donc à représenter des cases, permet de stocker et de pouvoir récupérer facilement les informations de valeur et de position d'une case. Les informations de position ne seront pas spécialement utiles ici mais le seront dans d'autre parties du projet. Ce tableau est vide au premier appel externe de la méthode.

D'abord, on initialise à la ligne 275 une liste contenant tous les chiffres pouvant être contenues dans une case de sudoku (de 1 à 9).

On entre ensuite dans le parcours du tableau, la variable *i* parcours les lignes et la variable *case* prend la valeur de chaque case de cette ligne, ces deux boucles sont donc destinées à parcours l'entièreté du tableau.

On peut noter la présence à la ligne 291 d'un *else* puis d'un *continue*. Cette structure en for/else/continue sera expliquer plus tard car il faut comprendre le reste de la méthode pour pouvoir l'étudier.

Puis, on teste si la case courante est de valeur nulle ou non. Si elle l'est, on entre dans la condition et on mélange aléatoirement la liste des valeurs possibles, c'est ce qui nous permet d'obtenir des solutions différentes.

La variable *nombre* parcours ensuite la liste des nombres possibles et on teste s'il peut être placé dans la case courante à l'aide de la méthode *nombreValide*, renvoyant True si une valeur peut être placé dans une case en suivant les règles du sudoku, False sinon. Si la valeur peut être placée, on la place (ligne 283) puis on vérifie si la grille est remplie avec la méthode *grilleRemplie*.

C'est cette condition qui va confirmer l'arrêt de notre récursivité lorsque la grille est correctement remplie. En effet, comme on remplie des cases tant qu'il en reste des vides, la grille va forcément se remplir et retourner True. Si la grille n'est pas remplie, on rappelle la méthode avec la nouvelle grille modifiée qui va donc essayer de résoudre cette nouvelle grille. On essaye donc de résoudre les grilles consécutives à chaque fois que l'on fait le choix de modifier une valeur pour vérifier si ce choix peut mener à une solution.

La condition nous faisant conclure que la grille n'a pas de solution est lorsque l'on ne peut plus changer de valeur en respectant les contraintes, c'est-à-dire qu'aucun nombre dans les possibilités ne convient à la case courante. C'est ce qui est traduit par la présence des différents breaks et de la structure *for/else/continue*. En effet, à la ligne 290, lorsque l'on trouve une case vide mais qu'une possibilité ne convient, on sort du parcours de la variable *case*, on n'entre pas dans le *else* et on break donc de l'autre parcours de ligne effectuée par la variable *i*. On remet donc cette case à 0 et on renvoi directement False (ligne 299) lorsqu'on ne trouve pas de valeur pour une case nulle et notre récursivité remonte donc à la case précédente et essaye les possibilités suivantes et s'il n'y en a plus, renvoi False à son tour.

La structure *else : continue* ligne 291 et 292 permet de ne pas sortir du parcours de ligne (et donc renvoyer False) quand on ne trouve pas de case nulle sur une ligne et donc de passer à la ligne suivante.

Finalement, quand on a réussi à remplir correctement la grille, toutes les fonctions renvoient True et la grille passée en paramètre devient la grille solution car la valeur des cases à été modifiée dans la classe *Case*, donc on modifie directement le tableau en paramètre lorsqu'on modifie la valeur des cases. La méthode *grilleSolution* est donc appelé en début de programme (ligne 17) avec comme paramètre *self.grid_solution* la grille destinée à contenir la grille résolu.