Bonjour,

Aujourd’hui je vais donc vous présenter ma solution de Sudoku Checker et solver et comment j’ai pu utiliser le software testing pour valider ma solution.

Tout d’abord, j’ai commencé par créer le sudoku checker. Le sudoku checker consiste à tester une grille de sudoku pour qu’elle soit donc valide.

Qu’est-ce donc une grille valide ?

Une grille valide est une grille de taille 9 colonnes et 9 lignes présentant donc 81 chiffres allant de 1 à 9.

Ces chiffres ne peuvent pas être présent deux fois sur la même ligne, la même colonne ou encore la même sous-grille. Une sous-grille n’est rien d’autre qu’une grille de 3 colonnes pour 3 lignes avec des chiffres allant de 1 à 9.

Voici le code que je vais vous expliquer du sudoku checker.

Tout d’abord, l’on commence avec de simple vérification des paramètres d’entrée. On vérifie donc que le paramètre grid soit bien une liste car nous utiliserons ici une matrice 2d pour représenter une grille entière. Nous vérifions aussi que la longueur de cette grille soit bien de 9 colonnes et juste en dessous, nous vérifions cela aussi pour les lignes. Ensuite, on vérifie bien que chaque chiffre de la grille soit entre 0 et 9. 0 étant la valeur pour une case dite vide.

Nous utiliserons des sets pour traquer nos chiffres dans la colonne et dans les sous grilles.

D’ailleurs, comme vous pouvez le voir, nous créons ici 9 sets qui représentent les 9 sous-grille de la grille.

Ensuite, nous allons procéder à un scan horizontal de la totalité de la grille et voir s’il n’y a donc pas le chiffre vérifier dans cette colonne, ligne ou sous-grille. Si c’est le cas, on ajoute notre chiffre vérifier à notre traqueur de ligne, colonne et sous-grille.

Si la boucle arrive à itérer toute la grille c’est que la grille est valide et est donc une grille de sudoku valide.

Ensuite, il a fallu créer un sudoku solver qui résolvait donc les grilles auparavant validées.

Pour cela, un algorithme avec du backtracking a été utilisé.

L’algorithme se comporte de la manière suivante,

Tout d’abord, il crée une copie de la grille d’origine, ensuite, on a une stack pour backtracking. Là, encore une fois on va tester chaque ligne/colonne/sous-grille pour trouver des cases vides et ensuite, on va remplir ces cases vides avec l’entier supérieur testé. Donc, par exemple, on va d’abord tester avec 1, si ça ne va pas, on fait marche arrière grâce au backtracking et on teste avec 2 et ainsi de suite. Ce système fonctionne très bien, mais il a un petit bémol, c’est qu’il peut être gourmand en ressources si le niveau du sudoku est très compliqué.

Place au code,

Donc, tout d’abord, on vérifie donc que grille a résoudre soit bien une grille valide avec notre fonction précédente.

Ensuite, nous faisons une copie de notre grille originale pour que celle-ci ne soit pas modifiée par nos actions futures.

On va ensuite commencer à chercher les cellules vides de la grille et incrémenter la cellule d’un à chaque passage de la boucle. Si après incrémentation, la grille est encore valide, l’on considère comme une réponse temporairement valide, et on passe à la prochaine case vide. Si en revanche ça n’est pas le cas, on incrémente jusqu’à trouver une réponse qui pourrait être considérée de valide. Si l’on ne trouve pas de réponse, alors on revient en arrière et on vide la case à nouveau avant de revenir à notre précédente case qui avait une solution pour ensuite la changer et essayer une autre solution, si toujours aucune solution, on revient en arrière jusqu’à trouver une autre possibilité viable. On fait cela pour toutes les cellules vides et ensuite on retourne la grille résolue.

Nous passons ensuite au random tester de cette solution. Pour tester tout cela, il fallait créer un grand nombre de sudoku valides et de manière aléatoire. C’est là que la fonction de génération entre en jeu. Le générateur va se comporter un peu comme le solver, mais cette fois-ci, les chiffres seront donnés aléatoirement dans un set de 1 à 9 qui seront tenus dans une liste de chiffres valides encore à ajouter. Ce qui fait que au fur et à mesure, on se retrouve avec de moins en moins de chiffre à ajouter à chaque ligne et donc les lignes sont toutes différentes et respectent les condition d’un sudoku valide. La grille générée est donc déjà résolue.

A ce moment-là, le random tester entre en jeu, il prend la grille déjà résolue et va lui retirer des éléments de manière aléatoire et d’un nombre aléatoire. Chaque grille à résoudre pour le solver sera donc totalement aléatoire et aura des difficultés aléatoires. Le tester peut imprimer les résolutions par rapport à la grille originale générée si on lui passe la valeur « True » dans son deuxième paramètre. Il se trouve cependant qu’afficher les grilles ralentisse le programme.