MOOC software testing

Bonjour, je vais vous faire une présentation de ce que j'ai fait pour le problem set n°3 proposé dans la formation de Udacity, autrement di le testing d'un solveur de soduku.

Voici comment je vais procéder. Tout d'abord j'expliquerai brièvement concrètement en quoi consiste le problem set, je vais ensuite parler de la partie programmation du solveur, puis je vais parler des différentes stratégies de tests que j'ai utilisées pour tester ce solveur, et enfon je vais parler des différents problèmes que j'ai rencontrés et de la façon dont je les ai résolus si c'est le cas.

Tout d'abord, Ce problem set consiste en la création d'un programme qui permet de résoudre une grille de sudoku donné en entrée du programme, pour cela, il faut déjà que la grille passée en entrée respecte les préconditions, c'est-à-dire, qu'elle doit être une liste de 2 dimensions qui contient des entiers compris entre 1 et 9. La fonction check_sudoku.

Comment fonctionne mon code. Il s'agit tout simplement d'un algorithme qui utilise la force brute avec un système de backtracking. C'est-à-dire qu'elle va essayer de mettre tous les entiers 1 par 1 et si la grille n'est pas valide, elle va recommencer à l'entier supérieur tant que l'entier est inférieur à 9. La complexité de ce code est particulièrement élevée, notamment dans le worst case, (une grille qui contient que des 0).

Maintenant je vais vous parler de la stratégie de test que j'ai adoptée pour tester ce solveur. Tout d'abord, j'ai effectué du blackbox testing, c'est-à-dire que j'ai mis des grilles en entrée dont je connaissais déjà le comportement attendu de la fonction. J'ai ensuite envoyé ce code à un ami pour qu'il le teste sur sa machine et qu'il en fasse un review avec des grilles qu'il a de son côté. Par la suite, j'ai effectué un random_testing, des grilles sont générées aléatoirement un certain nombre de fois. Pour générer ces nombres, j'ai fait appel à la librairie random qui fonctionne à l'aide d'un algorithme beaucoup utilisé qu'on appelle Twister Mersenne. Cet algorithme n'est pas particulièrement rapide ni extrêmement efficace face à certains tests statistiques mais étant donné qu'il utilise un certain nombre premier assez élevé, l'algorithme stocke ce nombre dans un tableau 32 bits de 623 dimensions, ces dimensions sont ensuite permutées et on obtient un chiffre de 32 bits. Ensuite j'ai créé une sorte de fuzzer basé sur l'algorithme de Charlie Miller, notamment avec la methode de réécriture du contenu du sudoku (c'est-à-dire des permutations des éléments). Des statistiques de réussites du programme ont été stockées notamment ceux du random tester. Les grilles sont affichées ainsi que leurs solutions afin de vérifier le bon fonctionnement du programme.

Pour conclure, je vais parler des différents problèmes rencontrés lors du debug et du testing du solveur. Il y a eu les problèmes de conversion de code de Python2 vers Python3 mais ces problèmes ont été réglés très rapidement, Lors de la génération de grilles aléatoire, j'ai rencontré quelques difficultés pour générer des grilles, notamment certaines fois où une ligne était générée et toutes les autres étaient les mêmes, fatalement le solveur échoue. Dans les débuts je ne créais pas

de copie des grilles, ce qui faisait que je rentrais des grilles complétées dans le solveur. J'ai tenté de stocker les évènements qui ont eu lieu lors des tests mais je ne suis pas parvenu à logger le contenu des grilles donc je me suis contenté de les afficher dans le terminal.

Donc voilà c'est à peu près tout ce que j'avais à dire à propos de ce problem set. Merci beaucoup d'avoir écouté.