IFT 3335 - TP1

Paul Godbert 20276212
Mathieu Boumard 20211063

- Comment faire fonctionner notre code :

Notre code est composé de 5 fichiers python situés dans le dossier src.
Le fichier sudoku.py correspond à l'implémentation de Norwig à cette adresse : http://norvig.com/sudoku.html, obtenu via studium.
Les fichiers Q2 à Q5 contiennent notre code, basé sur cette implémentation, pour répondre aux questions 2 à 5 respectivement.
Lancer chaque fichier python exécute une recherche de solution sur un groupe de 95, 100 et 1000 sudokus, et écrit dans la console le nombre de réussite, le temps moyen pris et le temps de la plus longue résolution.

Question 1:

Le programme résout les sudokus avec une fiabilité de 100%, et si rapidement que le temps moyen indiqué est de 0.01 seconde. Le temps maximal pour résoudre un puzzle dans l'ensemble des 100 sudokus est de 0.01 seconde.

```
└─ python3 sudoku.py
All tests pass.
Solved 95 of 95 95sudoku puzzles (avg 0.01 secs (69 Hz), max 0.07 secs).
Solved 100 of 100 100sudoku puzzles (avg 0.00 secs (252 Hz), max 0.01 secs).
Solved 1000 of 1000 1000sudoku puzzles (avg 0.00 secs (241 Hz), max 0.01 secs).
the precision percentage of this method is:  100.0 %
```

Question 2:

Après avoir retiré le choix de la case avec le moins de possibilités, le temps moyen pris par les calculs augmente légèrement. Il faut cependant garder en tête que ces temps peuvent variés étant donné qu'ils dépendent grandement des choix aléatoires réalisés par l'algorithme. Le taux de succès reste de 100%

```
└─ python3 Q2.py
All tests pass.
Solved 95 of 95 95sudoku puzzles (avg 0.03 secs (32 Hz), max 0.15 secs).
Solved 100 of 100 100sudoku puzzles (avg 0.00 secs (242 Hz), max 0.01 secs).
Solved 1000 of 1000 1000sudoku puzzles (avg 0.00 secs (236 Hz), max 0.02 secs).
the precision percentage of this method is:  100.0 %
```

Question 3:

Pour la question 3, nous avons choisi l'heuristique des "naked pairs", qui permet de simplifier la grille de sudoku en éliminant certaines possibilités dès que deux cases ne pouvaient contenir que deux chiffres. Les autres cases vides présentent dans la même "unit" (ligne, bloc ou colonne) éliminent donc ces nombres de leurs valeurs potentielles.

Le temps de résolution diminue lorsque l'on utilise cet algorithme, mais le taux de réussite diminue aussi. Sur les ensembles de 95, 100 et 1000 sudokus utilisés dans nos tests. Cette méthode, couplée à celle de Norwig, ne parvient pas à dépasser un taux de succès d'environ 31%. Le problème semble venir du fait que les deux heuristiques entrent en conflit : les décisions prisent par la solveur en suivant heuristique des "naked pairs" éliminent un bon nombres d'états qui auraient pu mener à la solution, pouvant mener notre solveur dans un sorte de "cul de sac" dont il ne peut sortir, même en retournant en arrière. Dans nos recherches, nous avons aussi testé l'heuristique de hidden pairs, mais les résultats étant encore moins bons, nous avons donc décidé de ne pas garder cette heuristique. Au final, on peut noter que le temps d'exécution est généralement moindre par rapport à la solution de Norwig (269hz contre 241hz sur 1000 puzzle, une plus grande fréquence indiquant un temps total moins long).

```
└─ python3 Q3.py
All tests pass.
Solved 3 of 95 95sudoku puzzles (avg 0.01 secs (119 Hz), max 0.04 secs).
Solved 46 of 100 100sudoku puzzles (avg 0.00 secs (252 Hz), max 0.01 secs).
Solved 451 of 1000 1000sudoku puzzles (avg 0.00 secs (269 Hz), max 0.01 secs).
The precision percentage of this method is: 31.42%
```

Question 4:

Ici, nous avons implémenté un algorithme de Hill-climbing. Nous commençons par noter les cases fixes, et remplir la grille aléatoirement, en respectant la contrainte des 3x3 seulement. Similairement à Lewis, nous définissons un voisin dans l'espace d'état comme une grille de sudoku identique à l'exception de deux valeurs non-fixes faisant partie du même 3x3 ayant été échangées.
L'algorithme étant vorace, il choisit à chaque itération "l'échange" diminuant le plus le nombre de conflits inter-lignes et inter-colonnes. Cependant, cet algorithme se retrouve très vite coincé dans un minimum local, à un score entre 45 et 55 en moyenne (le score étant le nombre de conflits, de 0 à 154).
Comme cet algorithme se retrouve bloqué très rapidement, le taux de succès est extrêmement faible : il est de 0 dans nos tests.

```
└─ python3 Q4.py
Solved 0 of 95 95sudoku puzzles (avg 0.04 secs (26 Hz), max 0.07 secs).
Solved 0 of 100 100sudoku puzzles (avg 0.04 secs (24 Hz), max 0.10 secs).
Solved 0 of 1000 1000sudoku puzzles (avg 0.04 secs (22 Hz), max 0.11 secs).
The precision percentage of this method is: 0.00%
```

Question 5:

Pour cette dernière question, nous avons amélioré notre algorithme de Hill-Climbing en un algorithme de Simulated Annealing. Comme décrit dans l'article de Lewis, nous choisissons un voisin aléatoire, et nous l'acceptons comme nouveau candidat, soit s'il est meilleur que l'itération précédente, soit avec une probabilité proportionnelle à la différence de score entre les deux.
Malheureusement, notre algorithme semble se bloquer lorsqu'il atteint un score d'environ 40, et commence à tourner en boucle à l'infini. Modifier le taux de refroidissement ou la température initiale n'a eu aucun effet sur le taux de réussite de l'algorithme.

Pour éviter de boucler à l'infini, nous avons ajouté un système de "température minimale", qui annonce l'échec de l'essai dès lors que la température passe en dessous.

Nous remarquons tout de même que le score moyen atteint par cette méthode est en moyenne plus faible que celui atteint par le Hill-climbing (35 à 45 pour le simulated annealing contre 45 à 55 pour le hill-climbing), en échange d'un temps d'exécution bien plus long.

```
└─ python3 Q5.py
Solved 0 of 95 95sudoku puzzles (avg 0.16 secs (6 Hz), max 0.24 secs).
Solved 0 of 100 100sudoku puzzles (avg 0.14 secs (7 Hz), max 0.18 secs).
Solved 0 of 1000 1000sudoku puzzles (avg 0.14 secs (7 Hz), max 0.26 secs).
The precision percentage of this method is: 0.00%
```