I.U.T. Montpellier-Sète – Département informatique – BUT 1^{re} année

Développement initiatique

SAE 1 : Jeu de Sudoku (4/12/2023)

1 Présentation du jeu de Sudoku

Le Sudoku est un jeu à un seul joueur. Le joueur doit compléter une grille de Sudoku imcomplète, trouvée par exemple dans un magazine, en une grille de Sudoku complète.

Une *grille de Sudoku complète* est une grille carrée 9x9 partitionnée en 9 carrés 3x3 appelés les *carrés* de la grille, telle que chaque ligne, chaque colonne et chaque carré de la grille contient exactement les entiers de 1 à 9. La table 1 donne un exemple de grille de Sudoku complète.

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

TABLE 1 – Une grille de Sudoku complète

Une *grille de Sudoku incomplète* est une grille dont certaines valeurs sont absentes, et qui peut être complétée en une grille de Sudoku complète. Les cases sans valeur sont appelées les *trous* de la grille. Par exemple, la grille de la table 2 est une grille de Sudoku incomplète (puisqu'elle peut se compléter en la grille de la table 1) qui possède 49 trous.

On considère ici une variante du jeu de Sudoku qui se joue à 2 joueurs. Chacun des 2 joueurs choisit en secret une grille de Sudoku complète, en enlève certaines valeurs et donne la grille de Sudoku incomplète obtenue à l'autre joueur, qui doit ensuite compléter *sa* grille, c'est-à-dire celle donnée par l'autre joueur. Pour équilibrer les chances des 2 joueurs, le nombre de trous est le même pour les 2 grilles. Ce nombre, appelé $nbTrous$, est fixé au début de la partie. Chaque joueur joue un coup à tour de rôle. Un coup consiste, soit à remplir l'un des trous de sa grille, soit à utiliser un joker, c'est-à-dire demander à l'autre joueur de lui révéler

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

TABLE 2 – Une grille de Sudoku incomplète

la valeur d'un trou, qu'il peut ainsi combler. La position du trou est choisie par le demandeur. Quand un joueur prend un joker, il reçoit un point de pénalité. Si le joueur a rempli un trou avec une valeur différente de celle prévue par l'autre joueur, il reçoit un point de pénalité, même si sa valeur permettrait aussi de compléter la grille en une grille de Sudoku complète, ce que l'ordinateur est incapable de déterminer dans le cadre de cette SAE. Dans ce cas, le joueur rejoue immédiatement jusqu'à ce qu'il remplisse un trou avec la bonne valeur ou prenne un joker. Chaque joueur remplit ainsi exactement un trou à chaque tour, de sorte qu'il complète sa grille en exactement $nbTrous$ tours. Le gagnant est celui qui a le moins de points de pénalité à la fin de la partie, avec « match nul » s'ils en ont autant.

On vous demande de programmer une partie de ce jeu entre l'ordinateur et un joueur humain. On vous fournit un « squelette » de code contenant les noms et les spécifications de toutes les fonctions de la version de base, que vous avez à compléter. Vous pourrez ensuite, si vous le souhaitez et s'il vous reste suffisamment de temps, programmer au choix une ou plusieurs des extensions proposées.

Nous insistons sur le fait que les spécifications données dans le squelette de code soivent être **scrupuleusement** respectées, faute de quoi vos fonctions ne passeront pas les tests automatiques qui seront utilisés pour l'évaluation, et vous obtiendrez une note qui ne reflètera pas la qualité de votre travail, même si l'exécution est tout à fait conforme à celle demandée.

2 Version de base (11 points)

2.1 Déroulement de la partie

Pour la version de base, la partie se déroule de la façon suivante, qui sera améliorée dans les extensions. C'est le joueur humain qui choisit la valeur de $nbTrous$. La grille de Sudoku complète choisie par l'ordinateur est pré-définie dans le code et l'ordinateur choisit les $nbTrous$ de cette grille de façon aléatoire. Noter qu'une grille ainsi définie est bien une grille de Sudoku incomplète, mais peut éventuellement être complétée en une grille de Sudoku complète de plusieurs façons.

C'est le joueur humain qui joue le premier coup. Quand c'est au tour de l'ordinateur de jouer, s'il existe dans sa grille un trou *évident*, c'est-à-dire tel qu'il existe un unique entier de 1 à 9 n'apparaissant ni dans la ligne, ni dans la colonne, ni dans le carré contenant ce trou, il choisit un tel trou et le remplit avec cette valeur unique. Par exemple, la case de coordonnées $(4, 9)$ de la grille de la table 2 est un trou évident de cette grille, avec 6 comme unique valeur. S'il y a plusieurs trous évidents, il choisit le premier dans l'ordre des lignes (de gauche à droite et de haut en bas). S'il n'y a pas de trou évident, il prend un joker, en choisissant le premier trou dans l'ordre des lignes.

On suppose que le joueur humain ne triche pas : il choisit bien une grille de Sudoku incomplète et répond correctement à l'ordinateur : il lui dit qu'il a rempli un trou avec une mauvaise valeur seulement quand c'est le cas et lui donne la bonne valeur quand il prend un joker.

2.2 Tableaux et matrices utilisés

Une grille de Sudoku est représentée par une matrice 9x9. Dans cette matrice, les valeurs d'une grille complète sont des entiers de 1 à 9. La matrice correspondant à une grille incomplète contient des valeurs de 0 à 9, les trous étant représentés par la valeur 0. Pour augmenter l'efficacité de ses calculs, l'ordinateur mémorise, pour chacun des trous de sa grille courante, l'ensemble des valeurs *possibles* de ce trou, ainsi que leur nombre. Une valeur *possible* d'un trou est un entier de 1 à 9 n'apparaissant ni dans la ligne, ni dans la colonne, ni dans le carré contenant ce trou.

Par exemple, pour le trou de coordonnées $(3, 6)$ de la grille de Sudoku de la table 2 (aux indices $(2, 5)$ de la matrice), l'ensemble des valeurs possibles est $\{3, 7\}$. Noter qu'il s'agit d'une possibilité « locale » car elle ne tient compte que de la ligne, la colonne et le carré contenant le trou. On pourrait donner une définition plus restrictive, et donc plus intéressante, de la possibilité d'une valeur en tenant compte de l'ensemble de la grille. C'est ce qui sera fait dans certaines extensions.

Un ensemble d'entiers de 1 à 9 est représenté par un tableau de booléens indicé de 0 à 9. La case d'indice 0 n'est pas utilisée et sa valeur est indéterminée. Pour tout indice i de 1 à 9, la case d'indice i a la valeur `vrai` si l'entier i appartient à l'ensemble, et `faux` sinon. Par exemple, l'ensemble $\{3, 7\}$ est représenté par le tableau de booléens $[?, F, F, V, F, F, F, V, F, F]$. Ainsi, pour compléter sa grille, l'ordinateur utilise 3 matrices 9x9 :

- une matrice $gOrdi$ d'entiers de 0 à 9 représentant sa grille courante ;
- une matrice $valPossibles$ contenant pour chaque trou de sa grille courante, c'est-à-dire correspondant à une valeur nulle de $gOrdi$, un tableau de booléens indicé de 0 à 9 représentant l'ensemble des valeurs possibles de ce trou ;
- une matrice $nbValPoss$ contenant pour chaque trou de sa grille courante le nombre de valeurs possibles de ce trou.

Par exemple, si la grille courante de l'ordinateur est la grille de la table 2 $gOrdi[2][5] = 0$, $valPossibles[2][5]$ est le tableau de booléens $[?, F, F, V, F, F, F, V, F, F]$ et $nbValPoss[2][5] = 2$.

Les cases des matrices $ValPossibles$ et $nbValPoss$ ne correspondant pas à un trou de $gOrdi$ ne sont pas utilisées et leur valeur est indéterminée.

Ainsi, un trou évident est un trou ayant une seule valeur possible, c'est-à-dire correspondant

à un singleton dans la matrice $valPossibles$ et à la valeur 1 dans $nbValPoss$. Sa valeur est alors égale à cette unique valeur possible. Sur l'exemple de la grille de la table 2, pour le trou évident de coordonnées $(4, 9)$, on a $valPossibles[3][8] = \{6\}$ et $nbValPoss[3][8] = 1$.

La grille courante du joueur humain est représentée par une matrice 9x9 $gHumain$ d'entiers de 0 à 9, qui est égale à la fin de la partie à la grille de Sudoku complète secrète de l'ordinateur appelée $gSecret$.

Remarque importante : Les tableaux et matrices seront initialisés et modifiés par des fonctions ayant ces tableaux et matrices en paramètres. Donc, *contrairement à ce qui était souvent fait en cours et en TD*, vous allez définir des fonctions *modifiant* les tableaux (ou matrices) donnés en paramètres. En Java, un tableau transmis en paramètre d'une fonction f_2 et modifié dans f_2 est aussi modifié dans la fonction f_1 appelant f_2, ce qui est justement l'effet souhaité ici.

2.3 Fonctions utiles

Les premières fonctions à programmer sont rassemblées dans la rubrique *Fonctions utiles*. Il s'agit de fonctions qui seront utilisées dans la suite du programme, mais qui ne sont pas spécifiques au jeu de Sudoku et pourraient être utilisées dans des contextes différents. Ces fonctions pourraient être insérées dans le fichier `Ut.java`, que vous pouvez d'ailleurs utiliser. On vous demande cependant de n'utiliser que le fichier `Ut.java` fourni sur Moodle, et on vous demande de ne pas le rendre.

2.4 Initialisation des matrices

Au début de la partie, le joueur humain saisit la valeur de $nbTrous$, qui doit être comprise entre 0 et 81 (même si vous doutez de l'intérêt des valeurs 0 et 81 !). L'ordinateur crée alors sa grille secrète $gSecret$ (donnée dans le code) et calcule la grille $gHumain$ (que le joueur humain devra compléter) en choisissant $nbTrous$ trous dans la grille $gSecret$ de façon aléatoire. Le joueur humain saisit la grille $gOrdi$ que l'ordinateur devra compléter. On suppose qu'il ne triche pas : la grille $gOrdi$ saisie par le joueur humain est une grille de Sudoku incomplète et le reste pendant la partie. Le programme n'a pas besoin de le vérifier. Pour éviter les erreurs du joueur, il vérifie cependant que les valeurs saisies sont comprises entre 1 et 9 et que la grille saisie a exactement $nbTrous$ trous. Si ce n'est pas le cas, le joueur doit re-saisir les valeurs jusqu'à ce que ces conditions soient vérifiées. Ainsi, les grilles des 2 joueurs sont des grilles de Sudoku incomplètes pendant toute la partie, et à la fin chacun des joueurs a complété sa grille en la grille secrète de l'autre joueur.

Compléter les fonctions de la rubrique « *Initialisation* ».

2.5 Partie

Compléter les fonctions des rubriques « *Tour du joueur humain* », « *Tour de l'ordinateur* » et « *Parties* », en suivant scrupuleusement les spécifications données (au risque de nous répéter !).

Aspects pratiques (MODIFICI)

Sauf si votre enseignant vous donne des instructions contraires ou vous demande un rendu dans un projet Gitlab, vous devez créer un dossier contenant tous les fichiers de votre projet. Le nom de ce répertoire commence par `sudoku` et comprend ensuite le nom de famille des membres du groupe, par exemple `sudoku_Mozart_Schubert`.

Ce dossier contient le fichier de votre version de base `SudokuBase.java` (le squelette rempli, avec pas ou peu de méthodes en plus) et autant de fichiers que vous souhaitez pour ajouter des extensions à une copie de votre version de base. Il y faut ajouter un unique fichier `README.txt` indiquant :

- les contributions des différents membres du groupe,
- le contenu des différents fichiers,
- les stratégies utilisées ou les raisonnements effectués dans certaines extensions,
- un mode d'emploi si ce n'est pas évident.

Une fois ce dossier récupéré, il suffira à votre enseignant de se placer dedans, de compiler tous les fichiers en ligne en tapant `javac *.java` par exemple...

Une fois votre dossier stable, vous pourrez en faire un fichier compressé :

```
zip -r sudoku_Mozart_Schubert.zip sudoku_Mozart_Schubert/
```

Il ne vous restera plus alors qu'à envoyer votre fichier zip à votre enseignant par mail et/ou un rendu Moodle avant le jeudi 14/12 à 23 :59.

3 Extensions

Pour coder les extensions, vous pourrez créer un ou plusieurs dossiers en plus du dossier de la version de base. Les `README.txt` dans ces dossiers supplémentaires précisent les extensions qui sont codées (avec leurs numéros, comme 3.1 par exemple), et toute explication qui vous paraît pertinente, par exemple sur les stratégies suivies ou sur les parties du programme qui fonctionnent plus ou moins bien.

3.1 Joker versus choix aléatoire (1 point)

L'ordinateur, qui est un joueur avisé, se rend compte que quand un des trous de sa grille n'a que 2 valeurs possibles, il a intérêt à tenter sa chance en remplissant la case avec une des 2 valeurs plutôt que de prendre un joker. En effet, il aura alors seulement 1 chance sur 2 de recevoir un point de pénalité, plutôt qu'une pénalité certaine.

1. Il se demande alors s'il aurait intérêt à prendre ce risque si le trou a plus de 2 valeurs possibles, mais il a un peu oublié son cours de probabilités vu au lycée. Pouvez-vous l'aider : jusqu'à combien de valeurs possibles, l'ordinateur a-t-il intérêt à prendre le risque de choisir une valeur au hasard plutôt que de prendre un joker ?

   Remarque : à partir d'un nombre de valeurs possibles supérieur à 2, l'ordinateur peut

se tromper plusieurs fois avant de trouver la bonne valeur et donc prendre plusieurs points de pénalité, contre un seul point s'il prend un joker de suite.

2. Modifier le code pour introduire cette amélioration du comportement de l'ordinateur.
   Dans la fonction `tourOrdinateur`, il faudra bien sûr ajouter une validation par l'humain du coup tenté par l'ordinateur, ou l'ajout d'une pénalité.

3.2 Gestion de la tricherie (1 point)

Le joueur humain peut tricher soit en choisissant comme grille secrète une grille qui n'est pas une grille de Sudoku incompléte, soit en donnant des réponses en cours de partie qui ne correspondent pas à sa grille secrète. L'ordinateur s'en apercevra à un moment ou à un autre.

Il va vérifier dès la saisie de $gOrdi$ que la matrice ne contient pas de doublon dans une même ligne, colonne ou carré (condition nécessaire, mais non suffisante pour qu'une grille soit une grille de sudoku incomplète). Si l'ordinateur trouve un tel doublon, le joueur doit re-saisir les valeurs de sa grille jusqu'à ce qu'elle ne contienne plus de tel doublon (et ait exactement $nbTrous$ trous).

S'il n'a pas détecté la tricherie au moment de la saisie de $gOrdi$, il la détectera au cours de la partie,

— soit parce qu'un trou n'aura plus aucune valeur possible,
— soit parce que la valeur donnée par le joueur humain à l'occasion d'une prise de joker de l'ordinateur n'appartient pas à l'ensemble des valeurs possibles de la case,
— soit parce que le joueur humain prétend que la valeur proposée par l'ordinateur est incorrecte (et donne une pénalité à l'ordinateur) alors que c'est la seule valeur possible. Remarque : ce dernier cas de tricherie peut se détecter seulement si on a codé l'extension 3.1.

Dans ces trois cas, la partie se termine immédiatement avec la défaite du joueur humain.

3.3 Optimisation de la gestion des trous évidents (1 point)

Cette extension ne modifie pas la stratégie de l'ordinateur dans le jeu, mais rend son algorithme plus efficace.

Dans la version de base, à chacun de ses tours, l'ordinateur cherche un trou évident en parcourant la matrice $nbValPoss$ dans l'ordre des lignes.

Pour éviter ces parcours de matrice coûteux, on vous demande de stocker les trous évidents en attente dans un tableau de trous `tabTrous` géré comme une *pile* : le dernier élément entré (*push*) dans `tabTrous` sera le premier élément sorti (*pull*). Un trou est lui-même représenté par un tableau de 2 entiers contenant ses coordonnées. Les trous évidents occupent les premières cases du tableau de trous, à partir de l'indice 1 ; leur nombre est mémorisé à l'indice 0 (plus précisément, le nombre de trous sera égal à `tabTrous[0][0]`, la valeur `tabTrous[0][1]` étant indéterminée). Un nouveau trou évident est ajouté (*push*) à la suite des trous en attente, et c'est le dernier trou qui est pris (*pull*) quand l'ordinateur cherche un trou évident, si la pile

n'est pas vide. Notez que le trou évident choisi n'est plus a priori le premier dans l'ordre des lignes, ce qui n'a aucune importance pour le jeu !

3.4 Construction d'une grille de Sudoku complète (2 points)

Cette extension permet à l'ordinateur de construire lui-même sa grille de Sudoku complète secrète.

Il part d'une grille de Sudoku complète quelconque, par exemple la grille donnée dans le code ou une grille triviale contenant les entiers de 1 à 9 en ordre croissant dans la première ligne, ... à vous de compléter !

Il transforme ensuite cette grille en lui appliquant une succession (composition) de transformations aléatoires conservant l'état de grille de Sudoku complète. Les transformations possibles sont les suivantes :

- rotation de la grille de 90 degrés dans le sens trigonométrique
- symétrie par rapport à l'axe de symétrie horizontal (milieu)
- symétrie par rapport à la diagonale principale
- échange de 2 lignes passant par les mêmes carrés, paramétré par les deux numéros de ligne

3.5 Construction d'une grille incomplète de niveau facile (1.5 point)

Cette extension permet à l'ordinateur de construire une grille de Sudoku incomplète de niveau facile à partir d'une grille de Sudoku complète. Pour cela, il choisit successivement des cases qu'il transforme en trous. Pour sélectionner un trou à chaque étape, il itère sur toutes les cases remplies jusqu'à trouver, si possible, une case qui peut être tranformée en un trou évident. Quand l'algorithme ne trouve plus de tel trou, le processus se termine et la grille courante devient la grille incompléte choisie.

Dans cette extension, c'est donc l'ordinateur qui choisit la valeur de $nbTrous$, égal au nombre de trous de la grille ainsi construite.

A partir des extensions 3.4 et 3.5, vous pouvez programmer une partie de Sudoku où l'ordinateur construit lui-même une grille de Sudoku incomplète de niveau facile.

3.6 Comment trouver plus de trous évidents ? (2 points)

Cette extension améliore la stratégie de l'ordinateur en lui permettant de trouver plus de trous évidents, en donnant une définition plus large de l'évidence d'un trou :

- Pour une ligne i et une valeur val données, si une seule colonne j peut contenir val, le trou (i, j) est évident.
- De même, pour une colonne j et une valeur val données, si une seule ligne i peut contenir val, le trou (i, j) est évident.

Pour implanter cette extension, en plus des matrices $valPossibles$ et $nbValPoss$, vous utiliserez des matrices $colPossibles$, $nbColPoss$, $ligPossibles$ et $nbLigPoss$ similaires :

- $colPossibles[i][val]$ est l'ensemble des colonnes j telles que val est une valeur possible pour le trou $gOrdi[i][j]$, et $nbColPoss[i][val]$ est son cardinal.
- $ligPossibles[j][val]$ est l'ensemble des lignes i telles que val est une valeur possible pour le trou $gOrdi[i][j]$, et $nbLigPoss[j][val]$ est son cardinal.

Modifier toutes les fonctions liées à la stratégie de l'ordinateur pour que celui-ci recherche un trou évident de de $gOrdi$ à l'aide des matrices $nbValPoss$, mais aussi $nbColPoss$ et $nbLigPoss$.

Et les carrés ?

On pourrait aussi maintenir des matrices $carPossibles$ et $nbCarPoss$ pour les carrés.

On suppose que les carrés sont numérotés de 0 à 8 dans l'ordre des lignes, et les cases d'un carré donné sont numérotées de 0 à 8 dans l'ordre des lignes.

En plus des matrices précédentes, l'ordinateur pourrait ainsi utiliser ces matrices :
$carPossibles[c][val]$ donne l'ensemble des numéros de cases dans le carré c dont val est une valeur possible, et $nbCarPoss[c][val]$ est son cardinal.

3.7 Elimination de valeurs impossibles (2 points)

Que peut-on déduire du fait que 2 cases j_1 et j_2 d'une même ligne i contiennent la même paire de valeurs possibles (c'est-à-dire $valPossibles[i][j_1]$ et $valPossibles[i][j_2]$ sont de cardinal 2 et contiennent les deux mêmes valeurs) ? En déduire un moyen d'éliminer des valeurs (im)possibles pour les autres cases de la ligne i.

Le même raisonnement s'applique pour 2 cases d'une même colonne ou d'un même carré.

Le raisonnement ci-dessus s'applique à $k = 2$ cases mais se généralise en fait à tout entier k compris entre 2 et 8 : que peut-on déduire du fait que l'union des ensembles de valeurs contenues dans k certaines cases d'une même ligne (resp. colonne, carré) de $valPossibles$ soit de cardinal k ?

Vous pouvez réfléchir par exemple à partir d'une ligne comprenant $k = 3$ trous dont les valeurs possibles sont respectivement $\{2, 6\}$, $\{2, 8\}$, $\{2, 6, 8\}$.

A l'aide de ces raisonnements, améliorer la stratégie de l'ordinateur pour qu'il élimine plus de valeurs impossibles.

Plus précisément, l'extension consiste à :

- écrire les réponses aux questions ci-dessus et écrire les preuves de ces propriétés (dans le fichier Readme.txt), et/ou
- coder l'amélioration pour $k = 2$, et/ou
- coder l'amélioration pour le cas général (+ 1 point).

3.8 Prêcher le vrai pour obtenir le faux (1.5 point)

Cette extension met en œuvre un autre moyen d'éliminer des valeurs impossibles.

L'idée consiste à considérer un trou (i, j) donné, par exemple un trou avec peu de valeurs possibles, dans le cas notamment où l'ordinateur n'a trouvé aucun trou évident. Considérons une valeur possible val de ce trou que l'on va chercher à éliminer d'une étrange manière, en commençant par faire l'hypothèse que val est la valeur solution de (i, j) dans la grille complète :

1. Enlevons (provisoirement) de l'ensemble des valeurs possibles de (i, j) toutes les valeurs autres que val.
2. Appliquons sur cette grille un peu simplifiée toutes les stratégies vues précédemment pour remplir la grille et enlever des valeurs impossibles de certains trous.
3. Deux cas de figure :
   — On échoue en tombant à un moment sur un trou sans valeur possible. C'est en fait une bonne nouvelle : cela prouve par contradiction que val n'est pas une valeur possible de (i, j). Il faut alors repartir de la grille initiale (avant l'étape 1) et on peut retirer val des valeurs possibles de (i, j).
   — On n'échoue pas. C'est une mauvaise nouvelle car on a travaillé pour rien, on ne peut rien en déduire. Il faut aussi repartir avec la grille initiale obtenue avant l'étape 1.

On peut appliquer cette technique de réfutation à toutes les valeurs possibles d'un même trou à tour de rôle, ce qui peut permettre d'éliminer plusieurs valeurs impossibles de (i, j). Et on peut l'appliquer à tous les trous, en commençant plutôt par les trous avec peu de valeurs possibles, notamment celles avec seulement 2 valeurs possibles.

Remarques

Après l'étape 3, on ne peut pas continuer avec la grille obtenue (avec ses valeurs de la matrice $valPoss$) car les valeurs possibles dans les cases ont été déduites à partir de l'hypothèse que la valeur val est solution (et qu'on peut éliminer les autres valeurs de (i, j)). C'est pourquoi il faut sauvegarder l'état des matrices de la grille initiale (avant l'étape 1) pour repartir de celles-ci après l'étape 3.

Après implantation des extensions 3.6, 3.7 et 3.8, votre IA devrait être capable de résoudre le plupart des grilles « diaboliques » que vous lui soumettez... vous ne pourrez plus gagner contre l'ordinateur !