Cahier des charges

I- Algorithmes

Pour l'algorithme de Backtracking, on parcourt les cases une par une. À chaque fois qu'une case est vide, on regarde le premier chiffre, compris entre 1 et n*k, qui peut y être placé. Ensuite, on passe à la case vide suivante.
Si l'algorithme ne peut plus rien remplir et que la grille n'est pas complète, alors il retourne à la dernière case modifiée, et place le chiffre suivant. Si le chiffre qui est mis est déjà n*k, alors il remonte à la case qui la précède, et réitère l'opération.

Pour l'algorithme de l'Intelligence Artificielle, on souhaite avoir un algorithme qui effectue des déductions. On souhaite parcourir la grille case par case, et regarder si un numéro peut être facilement placé. Pour cela, on regarde si l'ArrayList solutions de la case ne contient qu'une seule valeur.

II- Objectifs

Dans un premier temps, nous utiliserons une base de données pour stocker des grilles de Sudoku complètes. Il y aura des grilles de tailles différentes, pour adapter notre sujet à diverses grilles.
Ensuite, nous essaierons de créer nous-mêmes nos grilles de Sudoku de façon aléatoire, en ne fournissant que les dimensions de la grille.

Lors du lancement du jeu, nous pouvons choisir les dimensions de la grille que nous désirons. Ensuite, on choisit si l'on désire utiliser l'algorithme de Backtracking, ou laisser l'Intelligence Artificielle résoudre la grille.
Si on choisit la deuxième option, alors deux options s'offrent à nous. Soit on laisse l'Intelligence Artificielle jouer, soit on intervient à l'aide de deux boutons. Le premier permet de mettre fin au jeu en affichant directement la solution de la grille. Le deuxième est un bouton "Suivant", qui permet de compléter une case avec la bonne valeur.
Une fois la grille complétée, nous pouvons passer le curseur sur les différentes cases, pour voir les raisons qui ont poussé l'Intelligence Artificielle à placer ce nombre et pas un autre.

Nous souhaitons également instaurer un niveau de difficulté. Celui-ci se traduira soit par une difficulté Facile/Normal/Difficile, soit par un entier qui définit le nombre de cases visibles à la création de la grille. Nous pourrons comparer le temps que met l'Intelligence Artificielle à compléter une grille avec beaucoup de case dévoilées, et le temps qu'elle met à en compléter une avec peu de case.

III- Les classes

Pour le projet, nous prévoyons deux classes. Ces dernières sont la classe Grille et la classe Case.
La classe Grille a comme attributs "Case[][] grille" représentant la grille de Sudoku, et deux entiers "int n" et "int k" représentant les dimensions d'un bloc de cette grille.
La classe Case quant-à-elle, possède trois attributs. Le premier est "int valeur", qui indique le numéro se trouvant dans la case. Ce numéro est compris entre 0 et n*k, où 0 est la valeur d'une case considérée vide. Puis, le deuxième attribut est "ArrayList<Integer> solutions", qui contient les valeurs que peut contenir la case. Enfin, le dernier attribut est "ArrayList<String> raisons'', qui contient les raisons qui ont poussé l'Intelligence Artificielle à mettre une valeur au sein d'une case.

Au niveau de l'interface graphique, nous aurons également deux classes. Celles-ci seront VueInterface et Model.

IV- Les méthodes

Dans la classe Grille, les méthodes sont :

- Premier constructeur Grille(File fichier) : Générer une grille de Sudoku grâce à un fichier
- Deuxième constructeur Grille(int n, int k) : Générer une grille de Sudoku aléatoire, de dimensions nk*nk
- Algorithme de Backtracking
- Algorithme de l'Intelligence Artificielle
- afficher() : Affiche la grille de Sudoku
- estFull() : Indique si la grille de Sudoku est complète
- appartientLigne(int n, int ligne) : Indique si l'entier n est déjà présent sur la même ligne
- appartientColonne(int n, int colonne) : Indique si l'entier n est déjà présent sur la même colonne

- appartientBloc(int n, int bloc) : Indique si l'entier n est déjà présent au sein du même bloc
- LigneFull() : Indique si la ligne est complète ou s'il ne manque qu'un numéro
- ColonneFull() : Indique si la colonne est complète
- BlocFull() : Indique si le bloc est complet
- LignePresqueFull() : Indique s'il manque un seul numéro pour que la ligne soit complète
- ColonnePresqueFull() : Indique s'il manque un seul numéro pour que la colonne soit complète
- BlocPresqueFull() : Indique s'il manque un seul numéro pour que le bloc soit complet
- Fonction qui enregistre le temps mis par l'Intelligence Artificielle pour faire une grille de Sudoku

Dans la classe Case, les méthodes sont :

- Constructeur Case(int n) : Crée une case de valeur n
- estVide() : Indique si la case est "vide"
- Fonction qui indique s'il n'y a qu'une valeur possible pour la case
- Fonction qui modifie l'ArrayList des solutions
- Fonction qui modifie l'ArrayList des raisons