DÉVELOPPEMENT D'UN SUDOKU EN JAVA

Projet tutoré - *DUT Informatique*

Sommaire

Introduction

Pour ce projet de 1ère année en DUT Informatique à Fontainebleau, nous avions pour consignes de développer de A à Z un jeu de Sudoku en Java...

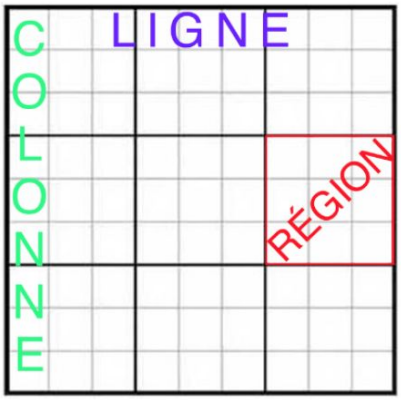
I/ Historique du Sudoku

L'histoire du Sudoku remonte à un jeu de mathématicien suisse du XVIIIe siècle appelé "latin squares". Quelques-uns des premiers numéros de puzzles à apparaître dans les journaux ont été publiés en France en 1895. Mais le jeu moderne de Sudoku tel que nous le connaissons aujourd'hui a été inventé en 1979 par **Howard Garns** (un inventeur de puzzle freelance de Connersville, Indiana, USA), et fut publié dans le magazine *Dell Pencil Puzzle* et *World Game*.

II/ Règles du jeu

Le Sudoku est un jeu présenté sous la forme d'une grille de 81 cases, plus ou moins préalablement complétées selon le niveau de difficulté, le principe étant de la remplir selon différentes contraintes que voici :

- Il ne doit pas y avoir deux fois le même chiffre sur la même ligne
- Il ne doit pas y avoir deux fois le même chiffre sur la même colonne
- Il ne doit pas y avoir deux fois le même chiffre dans la même région



Une grille de Sudoku

(Pour éviter de prendre trop de place, nous ne verrons ici que la version sous Ubuntu de l'interface graphique.

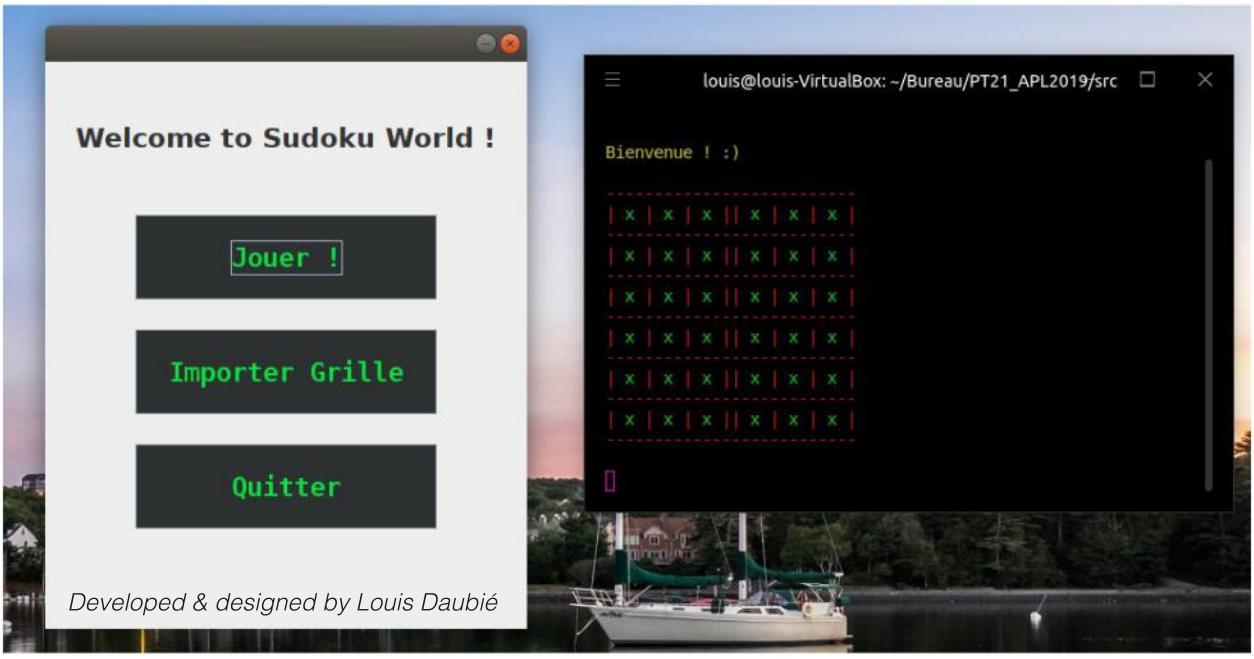
Vous retrouverez les captures d'écran pour MacOS et Ubuntu directement dans le dépôt git

Présentation du programme

I/ Présentation de l'interface graphique

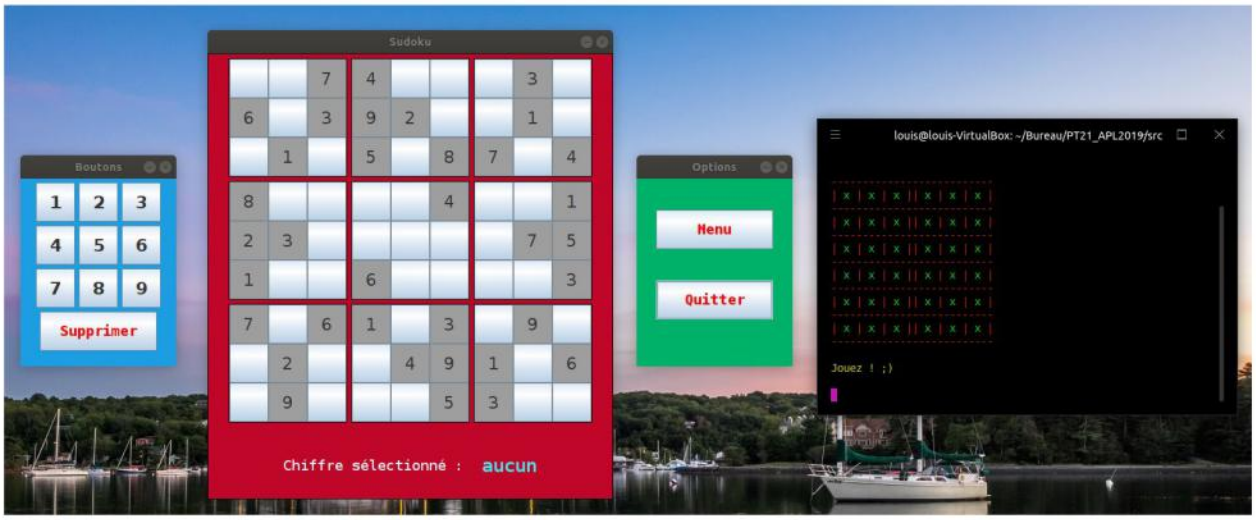
Au lancement du programme, le joueur se trouve face au menu principal. Il a alors le choix entre 3 options :

- Lancer la grille par défaut
- Importer une grille
- Quitter (si finalement il n'a plus envie de jouer, ou s'il est dérangé !)



Menu principal au lancement du jeu

Lorsque le joueur clique sur le bouton "*Jouer*", une grille définie par défaut se lance afin qu'il puisse commencer à jouer directement. Ces 3 fenêtres s'ouvrent alors :



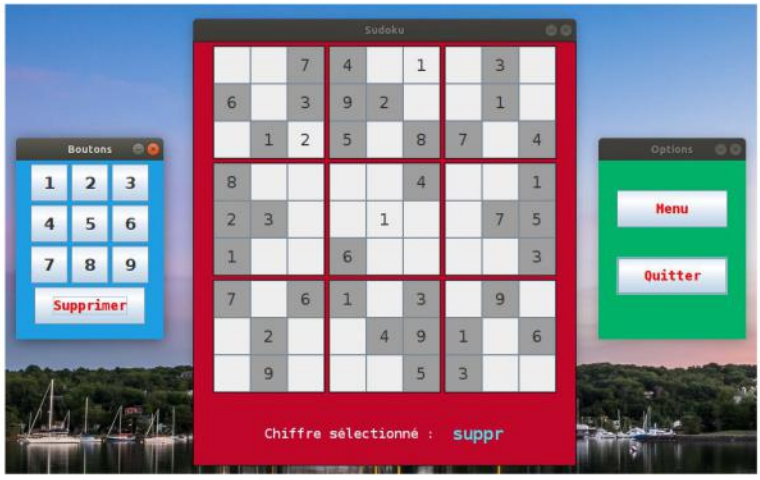
Affichage de la grille par défaut

Cependant, nous devons observer que l'apparence de l'interface graphique diffère selon les systèmes d'exploitation (cf. captures d'écran / git). Les cases grisées représentent les valeurs préalablement remplies par le programme, non-éditables par le joueur.

Le joueur peut alors commencer à jouer ; il a la possibilité de sélectionner un chiffre grâce au pavé numérique situé à gauche du Sudoku. La fenêtre principale affiche en direct quelle valeur est sélectionnée.

Par défaut, aucune valeur n'est sélectionnée. Si l'utilisateur se trompe, il peut à tout moment réinitialiser une ou plusieurs cases, en cliquant sur le bouton "*Supprimer*",

puis en sélectionnant des valeurs dans la grille.

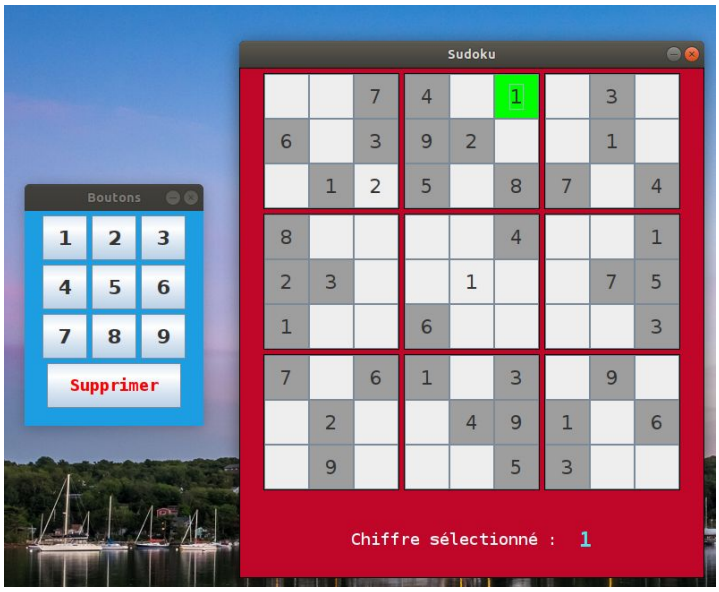


Suppression de valeurs

À droite de la fenêtre du Sudoku se trouve celle des "*Options*", permettant au joueur de revenir au menu principal ou simplement de quitter la partie.

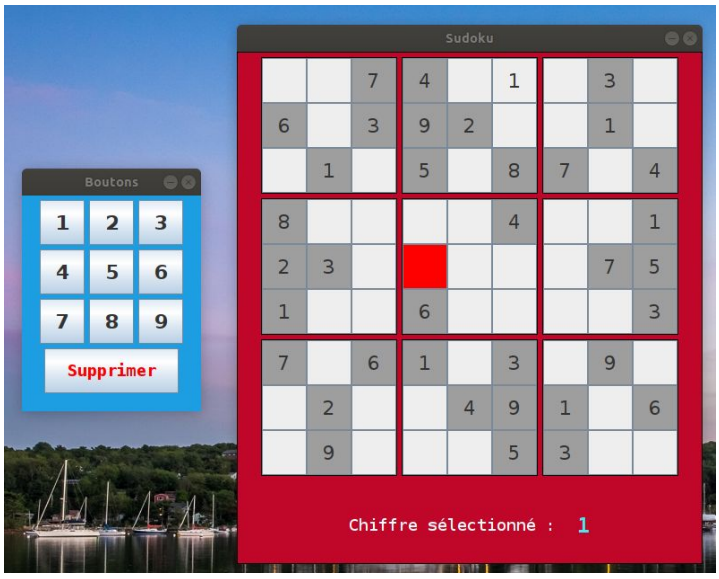
L'algorithme de vérification prend en compte les contraintes du Sudoku (*cf Règles du jeu*), et permet à l'utilisateur de savoir en direct si la valeur sélectionnée peut bien aller dans telle ou telle case.

Si l'emplacement est validé par l'algorithme, la case sélectionnée se colore en vert, et le chiffre y est inscrit.



Vérification en direct / valeur acceptée

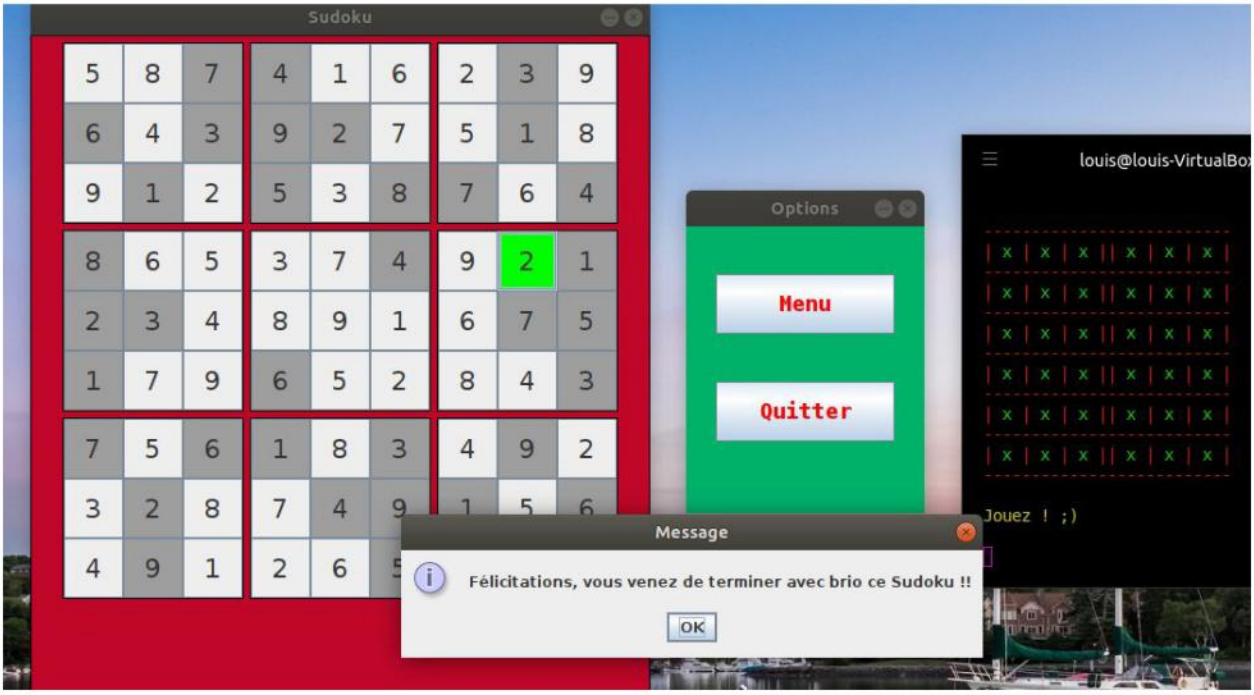
Sinon, la case rejette le chiffre, elle se colore en rouge et reste vide.



Vérification en direct / valeur refusée

Néanmoins, dans le cas où la grille est encore trop clairsemée, par exemple lorsque le joueur choisit une grille de niveau "Diabolique", il se peut qu'il puisse entrer une valeur dans une case qui s'avérera plus tard invalide. Ceci est dû au fait que l'algorithme vérifie au cas par cas, par rapport aux valeurs déjà présentes dans la grille. L'utilisateur devra alors revenir sur ses pas pour corriger ses erreurs...

Une fois que l'utilisateur a terminé sa grille, la phrase notifiant le chiffre sélectionné disparaît, laissant place à une fenêtre "pop-up" avec le message "*Félicitations, vous venez de terminer avec brio ce Sudoku !!*". Il ne reste plus au joueur qu'à cliquer sur "OK" pour quitter la partie.

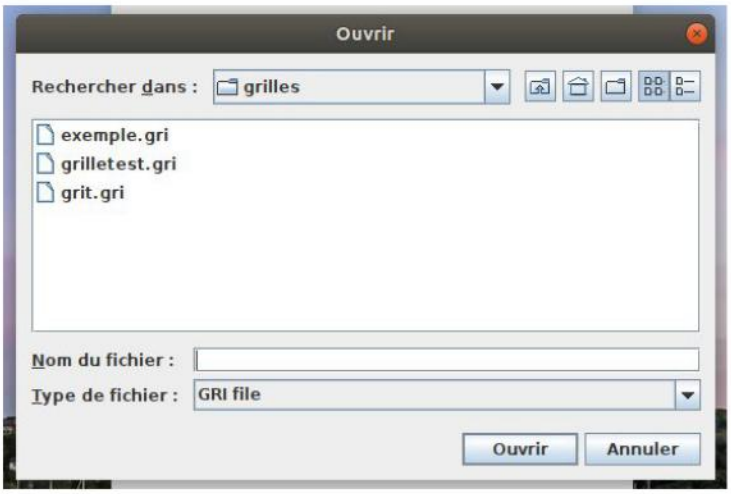


Fin de partie

Revenons au menu principal !

Le joueur doit pouvoir importer une grille avec des valeurs différentes de la grille par défaut.

Au clic, une fenêtre de sélection apparaît, lui permettant de choisir un fichier "*.gri*" contenant une suite de chiffres. Une grille est alors générée avec ces valeurs.



Fenêtre d'importation

II/ Structure du programme

Notre programme est constitué de 6 classes :

- *Main* : le coeur du programme, servant à son exécution.

- *Menu* : servant à la création et l'affichage du menu principal. Nous avons utilisé des JLabels pour afficher le contenu textuel, et des JButtons pour l'interaction avec l'utilisateur. La première partie permet l'affichage d'un message de bienvenue en console. Puis nous avons les caractéristiques de la fenêtre. Nous appelons également l'ObservateurMenu afin de faire fonctionner les boutons. La fonction custom() regroupe les caractéristiques graphiques des composants (polices, taille, couleurs...)

- *ObservateurMenu* : servant à définir les fonctions des boutons du menu principal. Pour cela, nous utilisons un ActionListener pour “observer” les événements au clic. À l’aide de conditions, nous définissons la fonction de chacun des boutons. L’itération de la variable “regulatorFenetre” sert à réinitialiser l’affichage de la fenêtre. Cependant, un bug persiste toujours, nous n’avons pas réussi à trouver une solution...

- *Grille* : servant à la création et l’affichage de la grille de Sudoku avec les valeurs stockées dans la classe Tableau. Nous avons fait le choix de créer l’interface graphique de notre grille en initialisant 81 boutons, répartis dans les 9 régions. Pour délimiter graphiquement ces dernières, nous avons utilisé l’objet Border, disponible dans le package swing fourni en standard. La structure globale du programme est la même que pour la classe Menu.

Nous y créons physiquement nos 81 boutons du Sudoku, dans lesquels nous récupérons les valeurs contenues dans la classe Tableau à l’aide de getText(), ainsi que les 10 boutons de sélection situés à gauche (les 9 chiffres + le bouton “Supprimer”).

Nous utilisons un ActionEvent pour déterminer quelle valeur a été sélectionnée par le joueur à l’aide de getSource() et de setText(), ce qui lui permet de vérifier en direct si la valeur placée dans la grille l’est correctement ou non.

Ensuite nous passons à la partie algorithmique du programme, pour cela nous effectuons des vérifications booléennes ;

* pour la fonction “isFinalStep()”, nous avons créé deux boucles for() imbriquées qui tournent tant que la grille n’est pas complètement remplie. Tant qu’il existe des cases vides dans la grille, elle retourne “false”, et une fois que toutes les cases sont remplies, elle retourne “true”, et la boucle cesse de tourner.

* pour la fonction “isGood()”, nous vérifions simplement si la valeur est déjà présente dans une ligne ou une colonne. Elle retourne “false” dans ce cas.

Pour finir, dans la dernière partie du programme, nous avons défini des "switchcases". Nous regardons au cas par cas pour chaque bouton d'un carré de 9.

- *ObservateurGrille* : servant à définir les fonctions des boutons du panneau "*Options*". C'est le même principe que pour "ObservateurMenu", expliqué plus haut. Malheureusement un bug persiste toujours et nous ne sommes pas parvenus à le régler...

- *Tableau* : contenant les valeurs affichées dans la grille par défaut, ainsi que les getters et les setters.

N'ayant malheureusement pas eu le temps de faire fonctionner l'importation de grilles comme prévu dans les consignes, nous avons écrit plusieurs grilles "en dur" ; pour tester l'algorithme avec ces dernières, il vous suffit simplement de vous rendre dans le fichier *Tableau*, de décommenter celle que vous souhaitez et commenter les autres *(cf. Annexe 1)*.
(Attention : initTab[][] ne doit être présent qu'une seule fois).

Un fichier Makefile permet la compilation du jeu (par la commande **make**) ainsi que son exécution (par la commande **make run**). Nous y avons transcrit toutes les dépendances entre nos fichiers dans les règles. La commande javac fait déjà (très mal) un travail similaire à **make**, ce qui peut créer des interférences. Pour éviter cela, nous lui avons passé l'option -implicit:none.

Dans un souci de clarté et d'une organisation optimale, les fichiers **.class* sont automatiquement placés dans le répertoire *class* lors de l'exécution du Makefile. La commande **make clean** supprime ce répertoire.

Annexe 1

Valeurs du tableau que l'on peut changer manuellement

```
public class Tableau {
    public Tableau() { }

    // Tableau initial

    // int[][] initTab = { { 0,0,0,0,9,5,0,0,4 },
    //                     { 5,3,0,4,0,8,7,0,2 },
    //                     { 0,0,0,7,0,0,6,0,3 },
    //                     { 9,0,0,0,3,4,0,8,0 },
    //                     { 0,4,0,0,1,0,0,7,0 },
    //                     { 0,2,0,5,7,0,0,0,6 },
    //                     { 4,0,9,0,0,2,0,0,0 },
    //                     { 6,0,7,9,0,3,0,2,1 },
    //                     { 2,0,0,6,5,0,0,0,0 } };

    ////// Nouvelles grilles tests //////

    int[][] initTab = { { 0,0,7,4,0,0,0,3,0 },    // Facile
                        { 6,0,3,9,2,0,0,1,0 },
                        { 0,1,0,5,0,8,7,0,4 },
                        { 8,0,0,0,0,4,0,0,1 },
                        { 2,3,0,0,0,0,0,7,5 },
                        { 1,0,0,6,0,0,0,0,3 },
                        { 7,0,6,1,0,3,0,9,0 },
                        { 0,2,0,0,4,9,1,0,6 },
                        { 0,9,0,0,0,5,3,0,0 } };

    // int[][] initTab = { { 0,0,8,0,5,0,0,9,0 },    // Moyen
    //                     { 0,0,0,4,8,3,0,2,7 },
    //                     { 0,0,0,6,0,0,5,3,0 },
    //                     { 3,8,0,0,1,0,0,7,6 },
    //                     { 6,0,0,7,0,4,0,0,1 },
    //                     { 2,1,0,0,6,0,0,4,5 },
    //                     { 0,7,3,0,0,1,0,0,0 },
    //                     { 8,5,0,3,9,7,0,0,0 },
    //                     { 0,2,0,0,4,0,7,0,0 } };

    // int[][] initTab = { { 4,0,0,0,0,0,9,0,5 },    // Difficile
    //                     { 0,0,8,9,0,1,0,0,0 },
    //                     { 0,0,9,4,0,0,0,8,3 },
    //                     { 0,0,0,8,0,0,4,0,0 },
    //                     { 0,8,6,3,0,4,5,2,0 },
    //                     { 0,0,5,0,0,7,0,0,0 },
    //                     { 2,5,0,0,0,9,8,0,0 },
    //                     { 0,0,0,1,0,3,6,0,0 },
    //                     { 1,0,7,0,0,0,0,0,2 } };

    // int[][] initTab = { { 3,0,0,8,0,5,0,0,2 },    // Diabolique !
    //                     { 0,7,0,0,2,3,0,0,0 },
    //                     { 0,0,0,0,7,0,0,0,3 },
    //                     { 0,0,9,7,0,0,5,2,0 },
    //                     { 0,0,2,0,0,0,8,0,0 },
    //                     { 0,1,6,0,0,2,3,0,0 },
    //                     { 6,0,0,0,4,0,0,0,0 },
    //                     { 0,0,0,2,9,0,0,4,0 },
    //                     { 9,0,0,5,0,1,0,0,6 } };
```