Je m'occupe principalement de l'interface graphique. C'est-à-dire une interface à la fois esthétique et ergonomique, qui permettra à l'utilisateur de jouer au sudoku même s'il ne sait pas programmer ou utiliser un terminal.
Pour ce faire, je me suis aidé tout au long de ce travail, du "Livre de Java premier langage" d'Anne Tasso ainsi que de forums où d'autres rencontraient les mêmes problèmes que moi.

Dans un premier temps, j'ai cherché à rédiger un bref pseudo code qui me permettrait de mieux visualiser les méthodes et classes dont j'aurai besoin en Java afin d'afficher une grille, des boutons et un titre sans qu'aucun n'offre de moyen de communication avec la machine : il ne s'agit pour l'instant que d'une esquisse esthétique, mais en aucun cas les boutons ne seront actifs pour l'instant.
Afin d'afficher une grille, le plus simple me paraissait de tracer des lignes verticales et horizontales. Pour cela, j'ai dû imaginer rapidement et sur papier, la taille de ma fenêtre, puis calculer les marges que je voudrais appliquer entre les traits les plus à l'extérieur, et les bordures de la fenêtre. Ensuite, j'ai récupéré sa taille. En supposant que la fenêtre ferait 500x500 (pixels), je voudrais une marge de 50 pixels. Puis, entre chaque trait vertical, je voudrais un pas de 45 pixels, ce qui signifie que les cases tracées par les traits feront 45 pixels de large. Il en est de même pour les traits horizontaux. Ainsi, en traçant 10 traits verticaux et 10 traits horizontaux, on se retrouve avec 9x9 cases de taille par 45x45 pixels (voir figure 3).

(Figure 3)

Dans la pratique, pour avoir le premier trait vertical il faudrait utiliser les coordonnées suivantes :
• xmin=marge et ymin=marge
• xmax=marge et ymax=marge+9*pas
D'où pour chaque trait verticale (supposons n allant de 0 à 10) on aurait alors :
• xmin=marge+n*pas et ymin=marge
• xmax=marge+n*pas et ymax=marge+9*pas
De même, pour les n traits horizontaux (n allant de 0 à 10) on a besoin des coordonnées :
• xmin=marge et ymin=marge+n*pas
• xmax=marge+9*pas et ymax=marge+n*pas
Ensuite, j'ai cherché à voir plus large : une fenêtre dans laquelle serait contenu un panneau alloué à accueillir la grille de sudoku, un panneau qui accueillerait les boutons et un panneau destiné à accueillir le titre du jeu. Là aussi, il s'agit de calculer des coordonnées pour que l'aspect esthétique soit respecté. Ainsi, j'aimerai que les trois sous fenêtre soient agencées comme suit (voir figure 4).

(Figure 4)

On pourrait alors imaginer les données suivantes :
• La fenêtre principale ferait 800x800pixels
• La grille ferait 500x500pixels
– Les cases ferait 45x45pixels
– De sorte que la grille commence à 50pixels des bords
• Le rectangle contenant le titre ferait 800x150pixels
• Le rectangle contenant les boutons ferait 200x100pixels
Pour l'esthétique, je pense remplir le rectangle de titre avec une image sur laquelle serait écrit « Sudoku Game » avec une police spécifique.
Le fond serait également une image, sûrement épurée.

Ma stratégie pour ensuite programmer cette interface, est d'y aller étape par étape :
1. Faire une grille dans un panel.
2. Faire une fenêtre et y ajouter le panel.
3. Ajouter deux autres panels temporairement vierge, destinés à accueillir les boutons et le titre.
4. Ajuster ces trois panels de sorte à ce qu'ils concordent au schéma ci-dessus.
5. Ajouter des boutons au panel de boutons, notamment un bouton pour que l'utilisateur puisse créer son propre sudoku, et un pour que l'ordinateur

en génère un. Ces boutons ne seront, à cette étape là, pas effectif.
6. Maintenant que la première fenêtre de « démarrage » semble correspondre aux attentes (je laisse les détails esthétiques comme le titre pour plus tard), j'essaye de refactorer le code de façon à avoir une méthode pour la fenêtre de démarrage, puis d'autres méthodes pour chaque fenêtre possibles (c'est-à-dire celle correspondant aux boutons créés, et celle correspondant aux boutons générés). Le refactoring permet aussi de bien définir chaque classe du package et, par la suite, lier plus simplement le code aux solveurs et générateur de sudoku de mes camarades.

Après avoir obtenu cette fenêtre de « démarrage », j'ai cherché à imaginer à quoi ressemblerait les différentes fenêtres lors d'un appui sur un bouton, et surtout les schémas à avoir pour les boutons car certains ne devront être affichés que dans des cas spécifiques. Par exemple, on ne va plus proposer à l'utilisateur de générer une grille s'il a déjà cliqué sur le bouton qui lui permettrait d'en créer une.
Voici un schéma des boutons souhaités :
• Générer Résoudre soi- même / Résoudre par l'ordinateur / Redémarrer le jeu
• Créer Résoudre par l'ordinateur / Recommencer une grille / Redémarrer le jeu
• Résoudre soi- même Tester la grille / Résoudre par l'ordinateur / Recommencer la grille /Redémarrer le jeu
• Résoudre par l'ordinateur Redémarrer le jeu
• Recommencer la grilleAucun boutons n'est effacé, ni modifié, ni ajouté, seule la grille change
• Tester la grille :
– Si la grille est fausse Aucun boutons ne change
– Si la grille est correcte Redémarrer le jeu
• Redémarrer le jeu Générer / Créer
Pour chaque bouton, voici la fonction qu'il devra emmener, en plus de changer les boutons accessibles à l'utilisateur :
• Générer : Un tableau est généré et affiché sur la grille.
• Créer : Un tableau vide remplissable par l'utilisateur est affiché sur la grille, et l'accord de changer des valeurs est accordé à celui-ci.
• Résoudre soi-même : L'utilisateur à l'accord pour changer des valeurs dans la grille et peut donc la remplir à sa guise.
• Résoudre par l'ordinateur : L'utilisateur voit la grille qu'il a créée ou généré se résoudre et obtient une solution s'il y en a une.
• Recommencer la grille :
– Si la grille est une grille générée : toutes les valeurs que l'utilisateur à entrer sont supprimées et il peut recommencer à zéro sur la grille générée, sans changer celle-ci.
– Si la grille est une création : la grille est réinitialisée avec toutes ses valeurs à zéro.
• Tester la grille : L'ordinateur teste la validité de la grille entrée par l'utilisateur et lui indique si elle est juste ou fausse.
• Redémarrer le jeu : Le jeu repart sur la fenêtre de démarrage, supprimant toute grille en cours.
De là, on s'aperçoit qu'il y a une base commune à chaque grille : le tableau. On peut donc créer plusieurs fonctions relatives aux tableaux, comme pour récupérer une valeur spécifique, en modifier une, copier un tableau dans un autre…

On pose alors une base en imaginant que si une case du sudoku est vide, dans la matrice que comprend l'ordinateur, cela est représenté par un zéro. Toutes les autres valeurs sont identiques au réel, pour les nombres de 1 à 9.
A la fenêtre de démarrage, on souhaite donc avoir une grille vide, remplie de zéro. De même lorsque l'utilisateur clique sur le bouton « créer », avant d'avoir entré d'autres valeurs. Dans le cas d'une grille générée, il faut pouvoir récupérer les valeurs du générateur d'Isabelle faisant une grille aléatoire ayant au moins une solution ; et dans le cas du solveur, récupérer les grilles successives produites par le programme d'Antoine.

Afin de palier à des problèmes de boutons, nous avons créé une variable booléenne qui indique à l'ordinateur si la grille en cours est une grille de création ou une grille générée. Cela permet, en appuyant sur le bouton « recommencer », de savoir quelle grille récupérer et recharger dans l'affichage.
Par ailleurs, on crée aussi un tableau intermédiaire 'tableauEnCours' qui permet de ne pas modifier les valeurs des tableaux initiaux, que l'on veut garder intact.
Enfin, il faut que l'utilisateur puisse entrer des valeurs dans certaines grilles. Afin qu'il n'y ait pas d'abus et que l'utilisateur entre un nombre à n'importe quel moment, j'ai choisi d'implémenter une variable booléene 'peutRemplir' qui permet, si elle est « true », à l'utilisateur de rentrer des valeurs, et si elle est « false », d'afficher un message d'erreur. Cette variable reste sur « false » lors de la génération d'une grille, de sa résolution par l'ordinateur, dans la fenêtre de démarrage, etc. Et passe sur « true » lorsque l'utilisateur clique sur « créer » ou sur « résoudre moi- même ».

Pour ajouter une valeur, l'utilisateur n'a qu'à cliquer sur la case de son choix, un JTextField apparaît alors sur la case en question. L'utilisateur peut entrer une valeur comprise entre 0 et 9, auquel cas la valeur sera prise par l'ordinateur et ajoutée dans le tableau et à l'affichage. Si l'utilisateur entre un autre caractère, un message d'erreur s'affiche. Un test de validité du code d'Antoine est lancé pour vérifier que la valeur rentrée n'est pas déjà dans ligne, colonne ou bloc, si c'est le cas un message d'erreur s'affiche également.

A chaque modification du 'tableauEnCours', que ce soit par l'ordinateur (générer, résolution...) ou par l'utilisateur lors d'un clic, une méthode s'occupe de rafraîchir l'affichage et donc d'afficher la grille affectée par la modification.

Pour le prochain rendu, outre régler les bugs actuels, j'aimerais pouvoir refactorer encore un peu le code (notamment sur la classe 'Grille' qui est selon moi trop chargée et mérite que l'on créer une nouvelle classe relative à l'utilisation / modification des tableaux ; implémenter des switchs au lieu des if trop importants...) afin de l'optimiser et peaufiner l'aspect esthétique.

== MAJ

Pour ce rendu, j'ai principalement optimisé des méthodes, mais je n'ai pas encore ajouté de classe afin de soulager la classe 'Grille'. Certaines méthodes en revanche se sont vu attribuer de nouvelles boucles ou tests booléens afin de les alléger.

Nous avons également résolu deux problèmes d'affichages :

• Lorsque l'utilisateur rentrait sa grille et que celle-ci était testée correct par l'algorithme, une case contenant le dernier chiffre rentré par l'utilisateur apparaissait entre le titre et la grille. Pour résoudre ce problème, le programme ne propose plus de redémarrer le jeu, mais le redémarre d'office en affichant un message de félicitations au joueur. Il peut alors choisir à nouveau de créer une grille, en générer une, ou quitter le jeu. Nous en avons profité pour supprimer le bouton "vérifier le sudoku" et lancer un test de vérification à chaque fois qu'un chiffre est rentré dans la table.

• Le second consistait à déporter les messages liés à l'algorithme de résolution sur l'écran de l'utilisateur et qu'ils n'apparaissent plus dans le terminal. Un tri a été réalisé pour ne plus afficher les messages superflus et la méthode d'affichage dans une nouvelle fenêtre qui avait déjà été appliqué pour les messages d'erreurs et de félicitations a été utilisée pour informer l'utilisateur du déroulement de la résolution.

Enfin, une mise à jour graphique a été effectuée :

• Le titre a été revu en blanc au lieu du noir, avec une police différente;
• Le fond est devenu gris;
• La grille et les chiffres de bases seront maintenant blancs et en gras;
• Les chiffres entrés lors de la résolution seront beige et en police normale.

Afin de différencier les chiffres entrés au début de la grille (donc lors de la génération d'une grille ou ceux entrés par l'utilisateur lors de sa création) de ceux qui sont ajoutés lors de la résolution (qu'ils s'agissent d'une résolution par l'ordinateur ou par l'utilisateur), j'ai créée une méthode de soustraction qui calcule la différence des valeurs pour chaque case de la grille.

Etant donné que l'absence de valeur dans la grille équivaut à un zéro dans le tableau, aucune valeur négative n'est possible. On a donc trois cas :

- Pour une case de tableau de base égale à zéro et une case du tableau en cours égale à zéro, on garde une case vide car 0 - 0 = 0.
- Pour une case de tableau de base égale à zéro et une case du tableau en cours ayant une valeur x, on garde la valeur x car x - 0 = x.
- Pour une case de tableau de base égale à une valeur x, la case du tableau en cours est nécessairement égale à x aussi, alors on garde une case vide car x - x = 0.

Ainsi, on écrit dans un premier temps en blanc les cases du tableau de base, puis seules les cases où la valeur n'est pas nulle après soustraction seront écrites en beige. Les autres cases ne sont pas concernées par l'écriture ou par le changement de pinceau.

Il faut également différencier le cas où l'utilisateur veut lui-même résoudre la grille du cas où c'est l'ordinateur qui le résout, car lorsque l'utilisateur résout une grille on ne sauvegarde aucun tableau de base, nous empêchant ainsi de différencier les chiffres à afficher en blanc de ceux à afficher en beige. Pour cela, on applique un simple test sur un booléen qui est modifié lors d'un click sur le bouton "résoudre par moi-même".

Enfin, nous avons pu régler plusieurs problèmes en une fois : Nous ne voulions pas que l'utilisateur puisse remplir une case avec des nombres à plusieurs chiffres, qu'il faille clicker sur une autre case pour valider le chiffre entré dans la case précédente, nous voulions supprimer des messages d'erreurs intempestifs comme " Entrez un nombre compris entre 0 et 9 " et refuser tout autre touche qu'un chiffre compris entre 0 et 9 car le code de la touche 'a' renvoyait par exemple '10' et le sudoku l'acceptait, ou encore la touche '-' qui rentrait le chiffre '-1' sans que le message d'erreur ne s'affiche.

Pour cela, j'ai déporté l'appel de la méthode qui consistait à effacer et repeindre le graphique juste après avoir appelé la méthode envoyant le nombre saisie dans le tableau. Ainsi, plutôt que de fermer la zone de saisie lors du click suivant, nous la fermons dès lors qu'un chiffre est entré.

Aussi, afin que l'utilisateur ne puisse rentrer qu'un chiffre, nous récupérons le code de la touche tapée dès lors qu'une touche est tapée et non pas lorsque l'utilisateur click sur une autre case. Nous en profitons pour instaurer un test sur une liste de caractères acceptables, c'est-à-dire les caractères correspondant aux chiffres allant de 1 à 9. L'appui sur toute autre touche du clavier revient à taper un 0, et donc ne rien modifier d'une case vide ou vider une case que l'utilisateur voudrait effacer (à condition qu'il ait le droit d'effacer la case en question). Cela m'a indirectement permis de simplifier le switch présent dans la méthode KeyReleased qui prenait à mon sens beaucoup trop de ligne pour pas grand chose. Ceci étant fait, toutes les petites erreurs qui détérioraient la qualité du jeu sont résolues.

Le jeu étant fonctionnel et n'ayant plus d'erreurs qui nous chagrinent, j'espère profiter du temps imparti pour le dernier rendu pour réfactorer encore celui-ci. Notamment en créant des nouvelles méthodes et alléger certaines méthodes qui ont pour moi trop de tâche à réaliser et qui en deviennent "illisibles" pour le programmeur.