Rapport de projet - Sudoku 2023

Sommaire

xii. Swordfish
    1. Illustration
    2. Pseudo-code
xiii. Jellyfish
    1. Illustration
    2. Pseudo-code
xiv. Squirmbag
    1. Illustration
    2. Pseudo-code
xv. Burma
    1. Illustration
    2. Pseudo-code
xvi. Coloré
    1. Illustration
    2. Pseudo-code
xvii. Turbotfish
    1. Illustration
    2. Pseudo-code
xviii. XY-Chain
    1. Illustration
    2. Pseudo-code
xix. XY-Colored
    1. Illustration
    2. Pseudo-code
xx. Coloring XY
    1. Illustration
    2. Pseudo-code
xxi. Medusa
    1. Illustration
    2. Pseudo-code
xxii. Candidat forcé en chaîne
    1. Illustration
    2. Pseudo-code
xxiii. Nishio
    1. Illustration
    2. Pseudo-code
xxiv. Test de chaque possibilité
    1. Illustration
    2. Pseudo-code

4. La vue
    a. Le lanceur du logiciel
    b. Le menu du logiciel
        i. Jeu
            1. Nouvelle partie

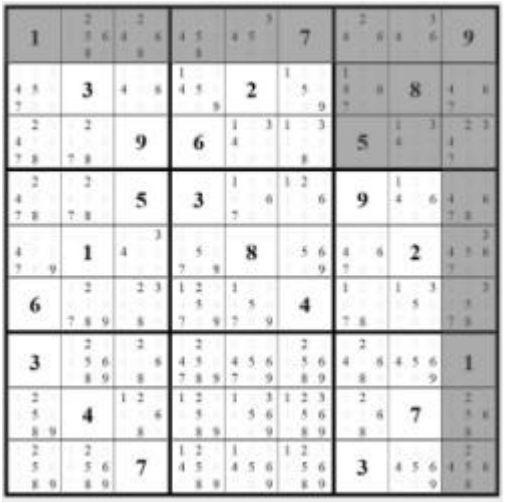
1. Introduction

Nous allons donc expliquer dans un premier temps ce qu'est le jeu du sudoku et comment nous avons implémenter le jeu. Ensuite nous expliciterons le modèle et son contenu dont les différentes heuristiques. Et nous viendrons mentionner la vue ainsi on expliquera le contenu de la fenêtre, également le procédé de l'affichage de la grille, les différents items du menu. Puis nous nous attarderons sur la notion de contrôleur qui nous permet de gérer les différentes actions des différents éléments. Et enfin nous allons parler de nos difficultés ainsi que des améliorations que l'on aurait pu apporter au projet.

2. Théorie

a. Explication

Le sudoku est un jeu de logique dont le but est de remplir une grille entièrement avec des symboles qui peuvent être des chiffres, des lettres, et même des couleurs… Il s'agit d'un jeu de logique car il faut faire attention à ne pas faire apparaître de doublons sur une ligne, une colonne et une région.



Classiquement une grille de sudoku est de taille 9x9 et elle contient donc :

- 9 lignes
- 9 colonnes
- 9 régions

Ici sur la capture de droite, montre avec la zone grisée, la ligne, la colonne ainsi que la région.

Le modèle est donc créé comme suit :

3. Le modèle

a. La grille

La grille est composée de cases. La taille d'une grille est fixée à sa création et elle correspond au nombre de cases qu'elle possède par ligne,colonne et région.

b. La case

Les cases sont soit fixées soit non fixées, si elles sont non fixées, alors on peut récupérer une liste de candidats qui est une Set<Integer>. La fixation d'une case est soit une fixation par l'utilisateur, donc pas un clic droit, soit une fixation par la grille,

donc au début du jeu. La différence entre les 2 est qu'on peut retirer la fixation d'une case fixée par l'utilisateur mais pas retirer la fixation faite par la grille.

c. Les régions

Une région n'est pas toujours un carré de n par n, elle peut être de taille 3 par 2, on peut récupérer le nombre de colonne par région et le nombre de ligne par région grâce à des types énumérés qu'on retrouve dans la classe Size.

d. Les heuristiques

i. Candidat unique

1. Illustration

Si on ne retrouve qu'un seul candidat pour la valeur n, dans une unité d'un sudoku, alors on peut affecter la valeur n à la case.

Ici, on retrouve un seul candidat pour la valeur 4 dans la ligne alors la case où il se trouve doit prendre la valeur 4.

Cette technique marche dans toutes les unités(ligne, région et colonne).

2. Pseudo-code

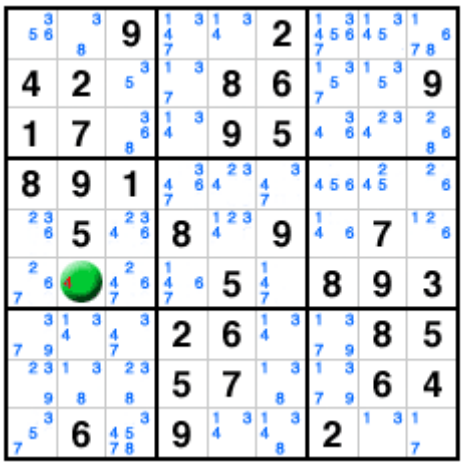
```
Pour chaque ligne, colonne et région :
    Compter le nombre d'occurrence de chaque candidat
    Si un entier à une occurrence de 1:
        Le placer dans la cellule où il est présent
```

ii. Un seul Candidat

1. Illustration

Si un case possède un seul candidat, alors ce candidat est forcément la valeur que la case va prendre.

Ici, on retrouve dans la case verte, un candidat qui est seul, c'est le 4 alors on peut mettre le 4 dans la case directement.



2. Pseudo-code

```
Pour chaque case du sudoku:
    Si la liste de candidat est de taille 1:
        mettre le candidat dans la case
```

iii. Les triplets et jumeaux

1. Illustration

Si dans une région, on retrouve un candidat qui se trouve sur une seule ligne ou une seule colonne, alors si ce candidat se trouve sur les autres régions de cette ligne ou de cette colonne alors on peut supprimer ces candidats de ces régions.

Dans cet exemple, on retrouve le 2 dans une seule ligne de la région de gauche alors qu'on met le 2 dans la 1ère case ou dans la 2ème case, ça retire le 2 du reste de la ligne donc on peut retirer le 2 du reste de la ligne.

2. Pseudo-code

```
Pour chaque région:
    Pour chaque ligne (ou colonne) de la région:
        Compter le nombre d'occurence de chaque candidat
        Si un entier à une occurrence d'au moins 2:
            Si vérifier qu'il n'apparaît pas dans le reste de la région
&& Vérifier qu'il apparaît sur le reste de la ligne (ou colonne) du
sudoku:
```

supprimer le candidat du reste de la ligne ou de la colonne du sudoku or de la ligne de la région

iv. Interactions entre régions

1. Illustration

Si dans 2 régions alignées, l'on peut constater qu'un candidat n'est pas présent dans une ligne (ou colonne dans l'autre variante). C'est qu'il se trouve dans cette ligne dans la 3ème région.
Dans cet exemple, dans la ligne 2, on remarque que la valeur 2 n'apparaît pas en tant que candidat dans les deux premières régions mais apparaît dans la 3ème région. On peut donc retirer ce candidat partout ailleurs dans la 3ème région.

2. Pseudo-code

```
Pour chaque ligne et colonne,
     Pour chaque valeur possible de la grille
            Si cette valeur n'est pas fixé dans cette ligne/colonne
                Si cette  valeur apparaît sur une seule région au sein de la ligne/colonne
                     Supprimer ces valeurs dans les candidats
      candidats dans cette région où elle apparaît sauf sur la
      ligne/colonne
```

v. Candidats identiques

1. Illustration

Cette méthode donne la possibilité de supprimer des candidats indésirables. Car en effet, si deux cases ne contiennent que deux fois les mêmes candidats.

On peut être certain que ces deux candidats vont finalement terminer dans l'une et dans l'autre. Et donc supprimer ces candidats des autres cases de la ligne

Cette méthode s'applique dans les cas suivants:
- avec 2 candidats dans 2 cases
- avec 3 candidats dans 3 cases
- avec 4 candidats dans 4 cases
...
- avec N candidats dans N cases

2. Pseudo-code

```
Pour chaque ligne,colonne,région
     Pour chaque case C de la grille non résolue
          NumberOfCandidates =  nombre de candidats de C
          NombreOfCases =  nombre de cases de la ligne/col/région
courante qui ont exactement  les mêmes candidats
          Si NombredeCandidats = NomberOfCases
                Retirer les candidats de C dans les cases de cette
ligne/col/région  qui n'ont pas exactement les mêmes candidats que C
```

vi. Groupes isolés

1. Illustration

Si 3 candidats se retrouvent seuls dans 3 cases, il est possible de savoir qu'ils finiront bien dans ces 3 cases. Même si les 3 candidats ne sont pas présents dans les 3 cases!

Et donc supprimer ces candidats des autres cases

Cette méthode s'applique dans les cas suivants:
- avec 2 candidats dans 2 cases
- avec 3 candidats dans 3 cases
- avec 4 candidats dans 4 cases
...

- avec N candidats dans N cases

2. Pseudo-code

```
Pour chaque ligne,colonne,région
 Pour chaque case C de la grille non résolue
     NumberOfCandidates =  nombre de candidats de C
     NombreOfCases =  nombre de cases de la  ligne/col/région courante
qui ont au moins les mêmes candidats
     Si NombredeCandidats = NomberOfCases
          Retirer les candidats de C dans les cases de cette
ligne/col/région  qui n'ont pas au moins les mêmes candidats que C
```

vii. Groupes mélangés

1. Illustration

Si on ne retrouve 3 candidats, que dans 3 cases, on est certain qu'ils vont terminer dans ces 3 cases. Il est donc possible de supprimer les autres candidats de ces 3 cases.

Et donc

2. Pseudo-code

```
Pour chaque ligne, colonne, région
             Créer un Set Integer  pour chaque valeur i
              Si une case de la grille contient cette valeur en
candidat
                           Ajouter l'indice de cette case dans le Set
Integer correspondant
              Créer un Set Integer union qui fait l'union de tous les
couples possibles des Set Integer
              Créer un Set Integer mixedGroups
              Pour chaque Set Integer inclue dans l'union
                       Ajouter la valeur i dans mixedGroups
              Si le nombre de valeur de mixedGroups == taille de
l'union
                       Supprimer les candidats autres que mixedGroups
des cases qui contiennent des candidats inclus dans mixedGroups
```

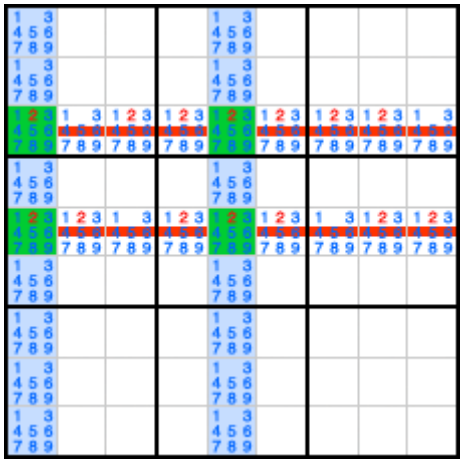
viii. X-Wing

1. Illustration

La principale caractéristiques des heuristiques, qu'on peut qualifier comme "fish", X-Wing compris, c'est que leur exécution repose sur le verrouillage de candidats par unités.
L'unité est soit la colonne, soit la ligne, soit la région.
Pour faire simple, il faut trouver deux unités contenant deux fois ce candidat uniquement et trouver deux autres unités qui vont relier ces premières unités aux secondes.

Ésotérique comme propos. Voici un exemple :

Prenez deux colonnes qui contiennent deux fois le candidat 2, et celles-ci ont des lignes qui relient les candidats entre eux. Ainsi, les candidats sont doublement verrouillés, comme montré ci-après:



Ainsi, toutes autres occurrences de cette valeur dans les unités qui ne sont pas verrouillées ne peuvent être gardées.

2. Pseudo-code

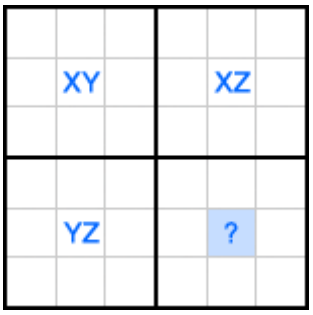
```
X-WingHeuristic
  Pour chaque valeur i de la grille
    Chercher colonne contenant i 2 fois ou plus
   Si nombre colonne >= 2
     Chercher lignes communes à i sur colonnes
     Si colonnes communes = 2
      Renvoyer i des autres lignes de colonne1 et colonne2
     FinSi
   FinSi
  FinPour
  Renvoyer "Pas de solution"
Fin
```

ix. XY-Wing

1. Illustration

XY-Wing, ou autrement appelé Y-Wing, a une géométrie semblable au X-Wing mais ne fonctionne pas du tout sur le même principe.
En effet, X-Wing fait apparaître quatre cases, définis par diagonales, là où le XY-Wing en fait apparaître trois, bien que la cible soit une quatrième.

L'heuristique repose sur trois choses : Un pivot et deux pinces.
Le pivot contient deux candidats, nommons les X et Y.
Les pinces, elles se trouveront soit sur la même ligne, soit sur la même colonne et auront, la valeur X ou Y, et une valeur supplémentaire, Z, comme montré ci après :



Ainsi, la case diagonale à notre pivot, qui est le croisement de nos pinces, ne pourra jamais contenir le candidat Z.

2. Pseudo-code

```
XY-WingHeuristic
  Chercher case ayant 2 candidats
    Pour chaque pivot trouvé
      Chercher sur ligne pince avec 1 seul candidat commun
      Chercher sur colonne pince avec 1 seul candidat commun
      Si pincev et pinceh trouvée
        Si candidat commun pincev pinceh != candidat commun pincev
pivot != candidat commun pinceh pivot
         Renvoyer candidat commun pinceh pincev sur ligne pinceh et sur
colonne pincev
        FinSi
      FinSi
    FinPour
    Renvoyer "Pas de solution"
Fin
```

x. XYZ-Wing

1. Illustration

XYZ-Wing, est une version complémentaire de XY-Wing, en effet, le fonctionnement est le même, à la différence que le pivot contiendra une troisième valeur, la valeur Z, commune aux deux pinces.
Ce qui fait que la case cible pour le retrait du candidat Z ne sera pas la diagonale du pivot, mais les cases séparant les pinces du pivot :

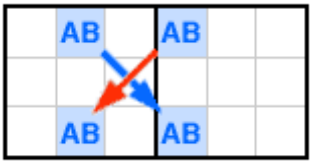
2. Pseudo-code

```
XYZ-WingHeuristic
  Chercher case ayant 3 candidats
    Pour chaque pivot trouvé
      Chercher sur ligne pince avec seulement 2 candidats commun
      Chercher sur colonne pince avec seulement 2 candidats commun
      Si pincev pinceh trouvée
        Si candidat commun pivot pincev != pivot pinceh != candidat
commun pivot pincev pinceh
          Renvoyer candidat commun pivot pincev pinceh sur colonne
entre pincev pivot et sur ligne entre pinceh pivot
        FinSi
      FinSi
    FinPour
    Renvoyer "Pas de solution"
Fin
```

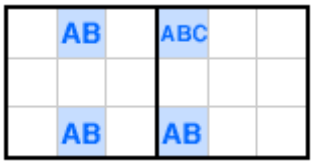
xi. Unicité

1. Illustration

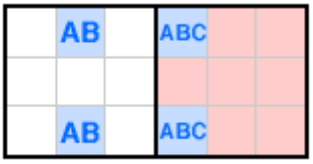
Un des règles principales du sudoku est qu'une grille de sudoku ne comporte qu'une seule solution. Alors, il est impossible de se retrouver dans ce cas.



Puisque dans ce cas, on peut obtenir plusieurs solutions.
Alors, quand on est dans un cas proche de celui ci, pour conserver l'unicité de la solution d'une grille, on va pouvoir soit mettre une valeur directement à une case dans ce cas ci.



Sinon, on peut retirer des candidats du reste de la région dans ce cas ci.



Puisque dans ce cas ci, pour garder l'unicité, on est obligé de mettre le candidat C dans une des 2 cases.(C n'est pas forcément un candidat unique, il faut juste que ce soit des candidats qui se trouvent dans les 2 cases en même temps)

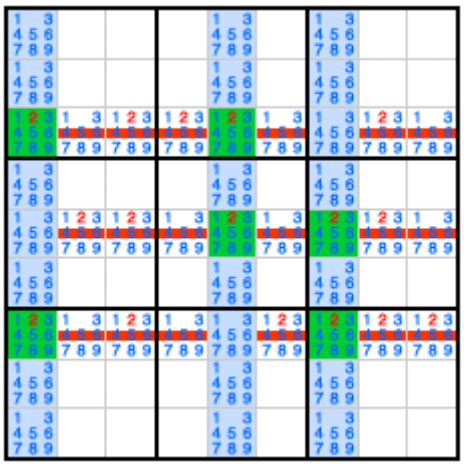
2. Pseudo-code

```
Pour chaque 4 uplet de case qui forme un "carré" dans 2 régions:
    Compter le nombre d'occurence de chaque nombre
    Si exactement 2 entiers a et b possèdent 4 occurrences && un autre
entier au moins une occurrence:
        Mettre les cases qui sont dans la même région dans la même
paire
        Si une paire possède 2 cases avec uniquement a et b:
            Si la 2nde paire possède une case avec uniquement a et b:
                On retire a et b des candidats de la dernière case
            Sinon retirer l'intersection des candidats des candidats de
la 2nde paire aux autres cases de la région possédant la 2nde paire
```

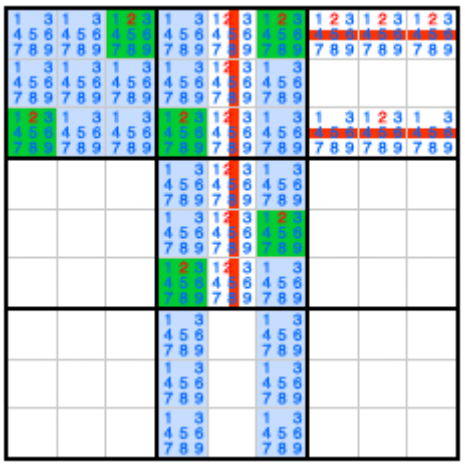
xii. Swordfish

1. Illustration

Il existe plusieurs cas pour cette heuristique, mais les 6 cas fonctionnent sur le même principe. On cherche 3 lignes (ou colonnes) où on trouve 2 candidats pour la même valeur et qu'on retrouve cette correspondance pour 3 colonnes (ou lignes) . Ici, on retrouve le 2 dans 3 colonnes et en même temps on retrouve cette correspondance dans les 3 lignes en rouge. Par exemple, si on choisit le 2 le plus en haut à gauche, on sera obligé de prendre les deux autres 2 dans sa diagonale pour prendre un maximum de 2. Et si on prend dans l'autre diagonale, on sera obligé de prendre les 2 autres de la diagonale, alors on ne sait pas qui prend parmi les 2 diagonales mais on sait qu'un 2 par ligne sera pris donc on peut supprimer les 2 des autres colonnes sur ces lignes.



Cela fonctionne pareil pour les autres cas. Ici, pour 3 colonnes qui contiennent chacune deux fois le 2 avec cette correspondance dans 2 lignes et 1 région.



Ici pour le cas où on retrouve deux fois le 2 dans 2 colonnes et 1 région et cette correspondance dans 2 lignes et 2 régions.

Cette méthode s'applique dans les cas suivants, avec **2 candidats**:
- dans **3 colonnes**, en supprimant les candidats dans **3 lignes**
- dans **3 colonnes**, en supprimant les candidats dans **2 lignes et 1 région**
- dans **2 colonnes et 1 région**, en supprimant les candidats dans **2 lignes et 2 régions**
- dans **3 lignes**, en supprimant les candidats dans **3 colonnes**
- dans **3 lignes**, en supprimant les candidats dans **2 colonnes et 1 région**
- dans **2 lignes et 1 région**, en supprimant les candidats dans **2 colonnes et 2 régions**

2. Pseudo-code

```
1er cas (2ème cas): Pour chaque n de 1 à SIZE
```

```
            Pour chaque combinaison de 3 lignes (ou colonnes):
                Si chaque lignes à au moins 2 occurrences de n
                    Récupérer les indices dans chaque lignes (ou
colonnes) où se trouve n
                    Faire une intersection des indices pour trouver 3
indices où se situe n
                    Supprimer n du reste des colonnes (ou lignes) où se
situe n sauf dans les lignes (ou colonnes) de départ

3ème cas (4ème cas): Idem mais on fait l'intersection pour trouver 2
colonnes (ou lignes) et 1 région ou se situe n au moins 2 fois dans
chaque

5ème cas (6ème cas):Pour chaque n de 1 à SIZE
            Pour chaque combinaison de 2 lignes (ou colonnes) et 1
région:
                Si chaque unité à au moins 2 occurrences de n:
                Récupérer les indices dans chaque lignes (ou colonnes)
où se trouve n
                Faire une intersection des indices pour trouver 2
colonnes(ou lignes) et 2 régions où se situe n
                Supprimer n du reste des 2 colonnes (ou lignes) et 2
régions où se situe n sauf dans les unités de départ
```

xiii. Jellyfish

1. Illustration

Vous vous souvenez de X-Wing et du double verrouillage de candidat ? Jellyfish, c'est la même chose.
Là où X-Wing est un verrouillage de 2 candidats dans 2 types d'unités.
JellyFish est un verrouillage de 2 à 4 candidats dans 4 unités :

2. Pseudo-code

```
JellyFishHeuristic
  Pour chaque candidat i
    Chercher colonnes avec i présent entre 2 et 4 fois
      Si colonnes >= 4
        Chercher 4 lignes communes pour i dans colonnes
        Renvoyer i dans lignes restantes pour suppression
      FinSi
  FinPour
  Renvoyer "Pas de solution"
Fin
```

xiv. Squirmbag

1. Illustration

Squirmbag est l'avant dernier représentant des heuristiques "fish", avant Burma.
Le fonctionnement est exactement le même que X-Wing, ou SwordFish, ou JellyFish.
On doit isoler entre 2 et 5 fois le même candidat dans deux unités différentes, 5 fois.

2. Pseudo-code

```
SquirmBagHeuristic
  Pour chaque candidat i
    Chercher colonnes avec i présent entre 2 et 5 fois
      Si colonnes >= 5
        Chercher 5 lignes communes pour i dans colonnes
        Renvoyer i dans lignes restantes pour suppression
      FinSi
  FinPour
  Renvoyer "Pas de solution"
Fin
```

xv. Burma

1. Illustration

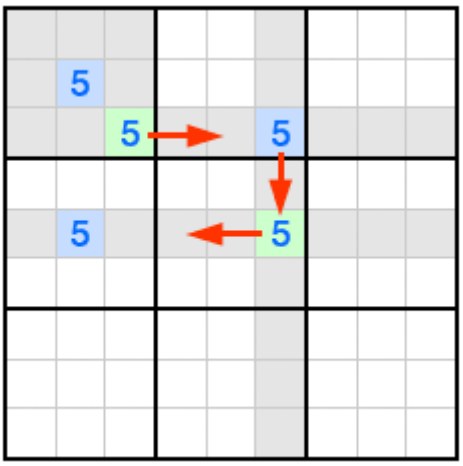
Il s'agit d'une autre façon de faire le XWing, or la il s'agit de trouver trois colonnes ou trois lignes avec exactement 3 mêmes candidats dans chacun de ces derniers. Avec en plus respectivement 3 lignes et 3 colonnes qui eux comportent au moins 3 candidats identiques dans chacun de ces derniers.

2. Pseudo-code

xvi. Coloriage

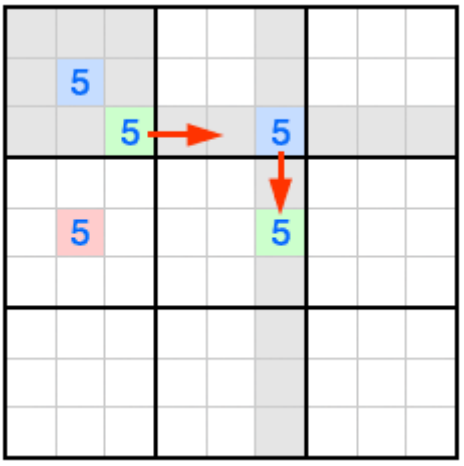
1. Illustration

Cette méthode permet de mettre en lumière les choix alternatifs. Le principe est simple, on cherche une unité où il y a 2 cases qui possèdent un candidat, on colorie l'un en bleu et l'autre en vert. Si on trouve une case qui possède ce même candidat dans la région de bleu, on le colorie en vert et inversement pour vert.
Si on obtient 2 cases bleues (ou vertes) dans la même unité, on peut supprimer tous les candidats des cases bleues(ou vertes) car on ne peut avoir qu'une seule occurrence d'un candidat par ligne.

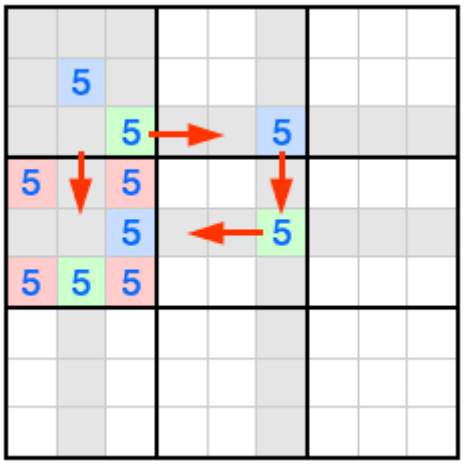


Ici, on a trouvé deux fois le 5 dans la région en haut à gauche, alors on colorie l'un en bleu et l'autre en vert et on colorie chaque case qui se trouve dans la même unité qu'une couleur et qui possède le 5 dans la couleur opposée. Alors, on retrouve 2 cases bleues dans la même colonnes, or c'est impossible donc on peut supprimer le candidat 5 de chaque case bleu.

Il y a aussi des cas où on retrouve une intersection entre 2 couleurs.



Ici, on retrouve à l'intersection d'une case verte et d'une case bleue donc on peut supprimer le 5 qui se trouve à l'intersection.



Même principe pour celui-ci, on retrouve une intersection entre les cases vertes et les cases bleues donc on peut supprimer les 5 dans les cases rouges.

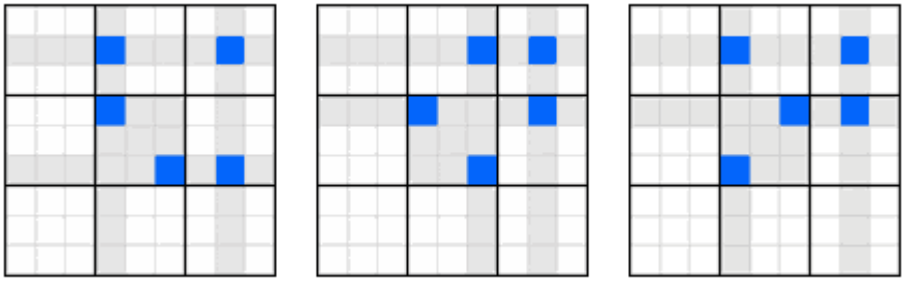
2. Pseudo-code

```
Pour chaque n de 1 à SIZE:
    Pour chaque case du sudoku:
        Si case a pour candidat n
            Rechercher une case2 dans une des ses unités qui possède n
            Si case2 existe:
                Colorier case en bleu et case 2 en vert
                Appel récursif:
                Si case de la même unité qu'une case bleu est aussi une
case qui est dans la même unité qu'une case verte:
                    Supprimer n de la case qui fait l'intersection
                Colorier en bleu toutes les cases qui possède n et qui
sont dans la même unité que les cases vertes
                Colorier en vert toutes les cases qui possède n et qui
sont dans la même unité que les cases bleues
                Si case bleu est dans la même unité qu'une autre case
bleu:
                    Supprimer tous les n des cases bleu
                Si case verte est dans la même unité qu'une autre case
verte:
                    Supprimer tous les n des cases vertes
                Si pas de nouvelle case colorier:
                    changer de case de départ
                Sinon:
                    Retourner à appel récursif
```

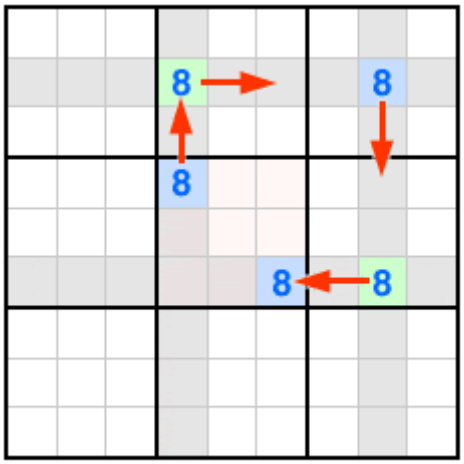
xvii. Turbotfish

1. Illustration

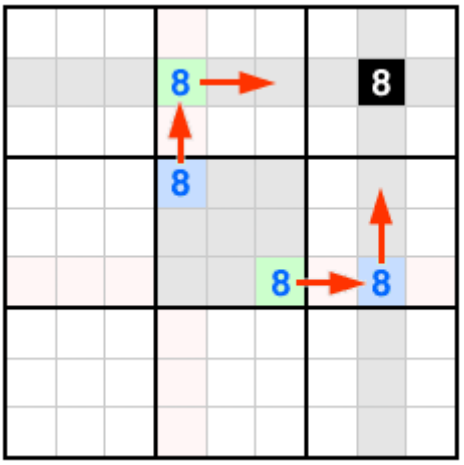
Pour cette heuristique, on va utiliser le principe du coloriage mais sur 5 cases uniquement, si les 5 cases sont dans une disposition particulière : 2 lignes, 2 colonnes, et 1 région et que les 5 cases possèdent un même candidat.



On peut alors utiliser la logique du coloriage sur ces 5 cases.
Ici, on retrouve 2 cases bleues dans la même région, alors on peut supprimer le 8 de toutes les cases bleues.



Ici, on retrouve l'intersection, alors on peut supprimer le 8 de la case noire d'après le principe d'un coloriage car la case noire est à l'intersection entre une case verte et une case bleue.



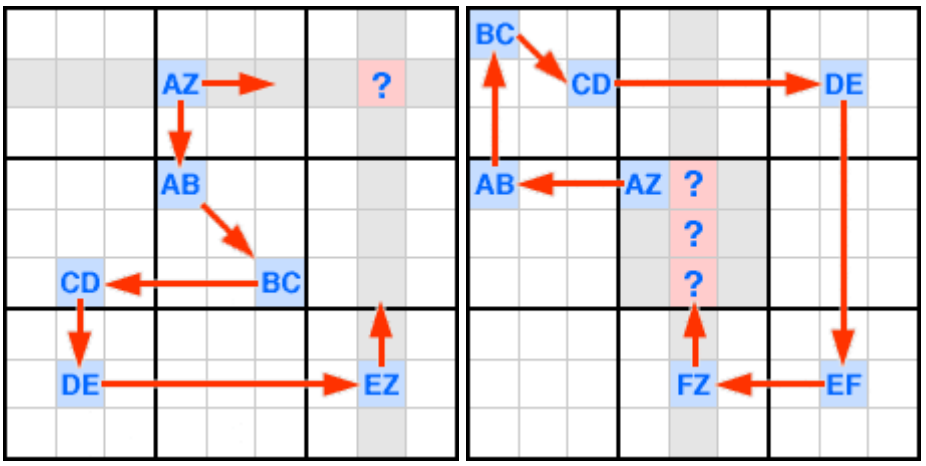
2. Pseudo-code

```
Pour chaque n de 1 à SIZE:
    Pour chaque ensemble de 5 cases dans le sudoku:
        Si on retrouve 2 lignes avec 2 cases chacune && 2 colonnes avec
2 cases chacune et 1 région avec 2 cases && les 5 cases ont n comme
candidats:
            (Appliquer le principe du coloriage sur ces 5 cases
uniquement)
            Prendre 2 cases dans une même unité (case1 et case2)
            Colorier case en bleu et case 2 en vert
                Appel récursif:
                Si case de la même unité qu'une case bleu est aussi une
case qui est dans la même unité qu'une case verte:
                    Supprimer n de la case qui fait l'intersection
                Colorier en bleu toutes les cases qui possède n et qui
sont dans la même unité que les cases vertes
                Colorier en vert toutes les cases qui possède n et qui
sont dans la même unité que les cases bleues
                Si case bleu est dans la même unité qu'une autre case
bleu:
                    Supprimer tous les n des cases bleu
                Si case verte est dans la même unité qu'une autre case
verte:
                    Supprimer tous les n des cases vertes
                Si pas de nouvelle case colorier:
                    changer de case de départ
                Sinon:
                    Retourner à appel récursif
```

xviii. XY-Chain

1. Illustration

XY-Chain ne démérite pas son nom, pour une fois, puisque le principe fondamental de son heuristique est le même que pour XY-Wing.
En effet, deux candidats et cases sont liés par une sorte de relation de Chasles, cette relation créant une chaîne.
Ainsi, prenons une case contenant seulement deux candidats, A et Z.
Nous souhaitons montrer que Z ne peut appartenir aux voisins de notre cas.
Nous allons donc chercher une case dans la même unité que la nôtre contenant A pour candidat… Du genre AB…
Nous allons donc réitérer cette démarche jusqu'au blocage ou… jusqu'à obtenir une case EZ… La boucle est bouclée et les unités de AZ et EZ ne pourront contenir Z, ce seront donc des candidats à éliminer :



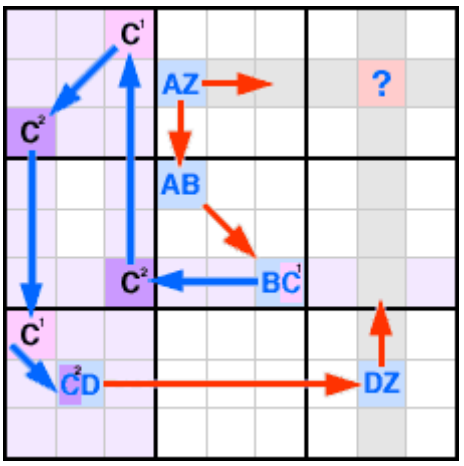
2. Pseudo-code

```
XY-ChainHeuristic
  Chercher case ayant 2 candidats
  Pour chaque maillon
    Chercher maillon avec 1 candidat commun
    Si maillon trouvé
      Si candidat commun entre maillon et debutChaine
       Renvoyer candidat commun entre maillon et debutChaine sur ligne
et colonne maillon et ligne et colonne debutChaine
      Sinon
        Chercher maillon suivant
      FinSi
    FinSi
  FinPour
  Renvoyer "Pas de solution"
Fin
```

xix. XY-Colored

1. Illustration

Vous vous rappelez de XY-Chain ? Vous vous rappelez du coloriage ?
Imaginez, maintenant que vous appliquez l'heuristique XY-Chain, mais que vous vous retrouviez bloqué ? Il vous suffirait juste de colorier le maillon de votre chaîne et une fois débloqué, de continuer la chaîne jusqu'à arriver au bout ou, jusqu'à ce que même le coloriage se retrouve bloqué. Voilà le but de XY-Colorié.
On applique XY-Chain et en cas de blocage, on applique le coloriage jusqu'au déblocage.



2. Pseudo-code

```
XY-ColoredHeuristic
Chercher case ayant 2 candidats
  Pour chaque maillon
    Chercher maillon avec 1 candidat commun
    Si maillon trouvé
      Si candidat commun entre maillon et debutChaine
       Renvoyer candidat commun entre maillon et debutChaine sur ligne
et colonne maillon et ligne et colonne debutChaine
      Sinon
        Chercher maillon suivant
      FinSi
    Sinon
      Chercher case à colorier
      Si case non coloriee
        Chercher maillon suivant
      FinSi
    FinSi
  FinPour
  Renvoyer "Pas de solution"
Fin
```

xx. Coloring XY

1. Illustration

2. Pseudo-code

xxi. Medusa

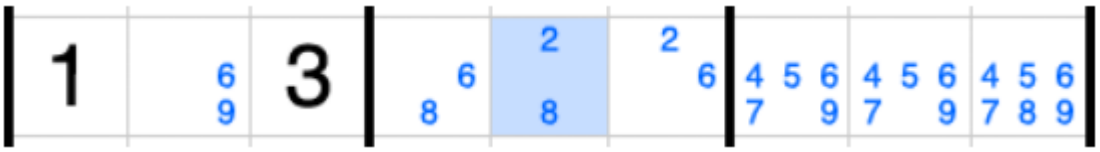
1. Illustration

2. Pseudo-code

xxii. Candidat forcé en chaîne

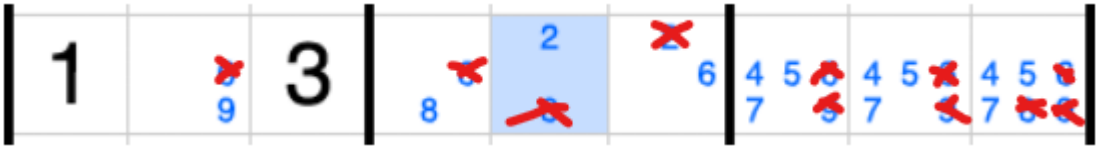
1. Illustration

Dans une situation de blocage, ou l'on se trouve en train d'hésiter entre plusieurs candidats d'une même case. La manière de résolution est très simple, il faut faire autant d'essais qu'il y a de candidats dans la case. Prenons le cas suivant :

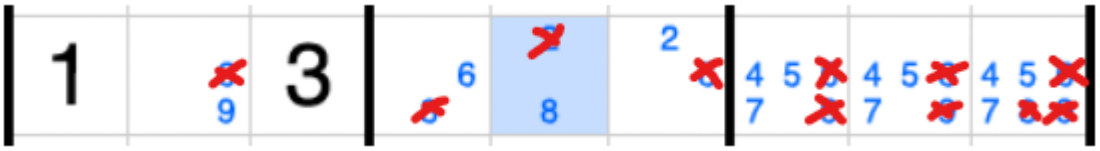


On prend par exemple la case qui est en surbrillance, on résout deux fois la grille:

- une première fois avec le candidat 2 : Voici les actions engendrées après la mise du candidats 2.



- une seconde fois avec le candidat 8 : Voici les actions engendrées après la mise du candidats 8.



On se rend donc compte qu'entre les deux essais que la première ne contient que le candidat 9, et que les trois dernières cases on supprime les mêmes candidats.
Si on constate des changements identiques pour les deux résolutions, on estime alors ces changements alors véridiques.

2. Pseudo-code

Voici le pseudo-code associé à la méthode de candidat forcée en chaîne :

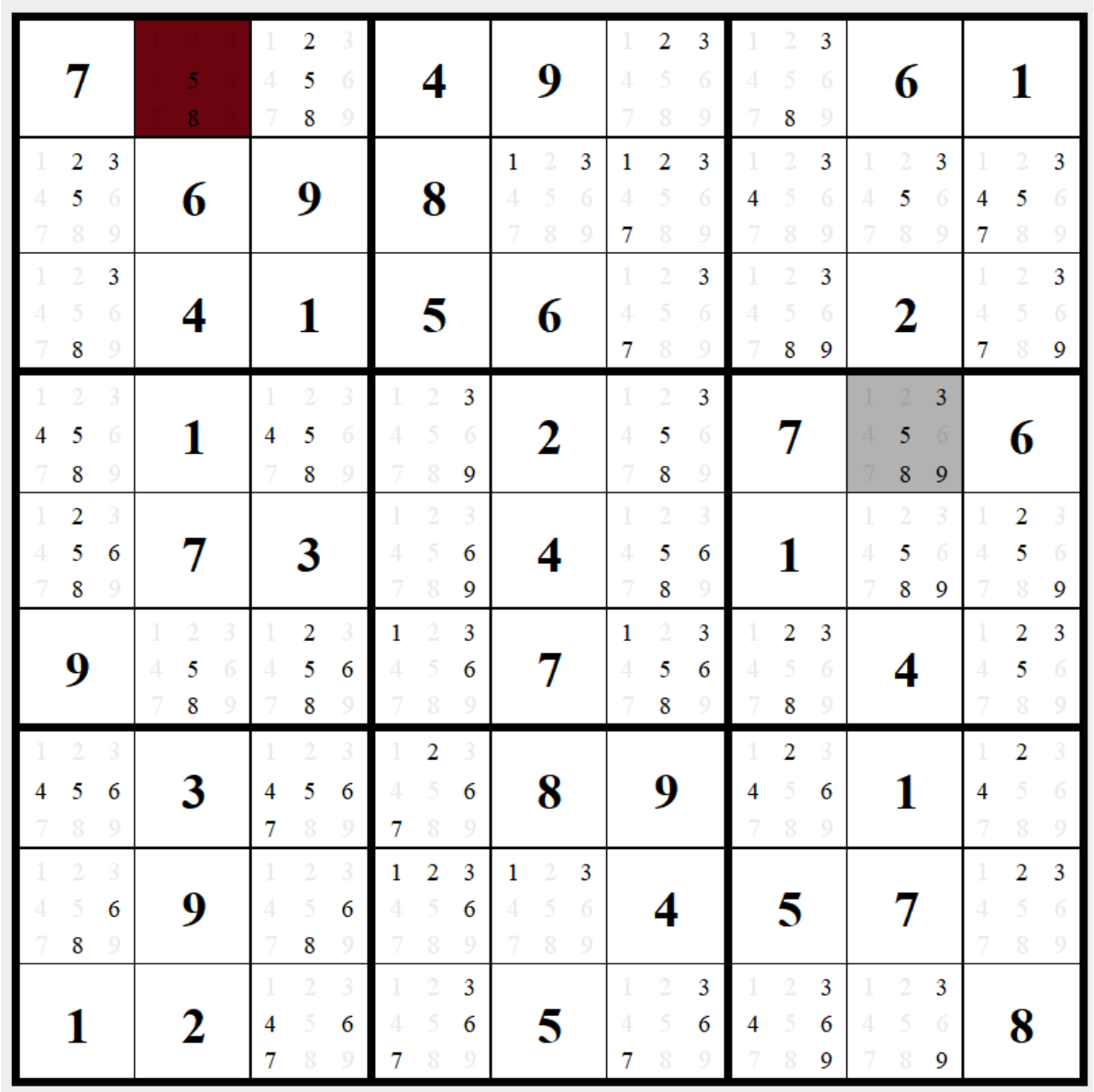
```
Variables :
   - La grille du modèle : g (récupéré à partir du modèle)
   - La liste des cases qui ne contiennent pas de valeurs fixées dans
     la grille : lst (liste non vide)
   - Le tableau de grilles - grids (non initialisé)

# Première étape : lecture et tentative de remplissage

Pour chaque case de lst :
      coordsCase ← Coordonnées de la case courante
      Créer grids qui comptent autant de grilles que de candidats de la
case
```

```
      Pour chaque candidats de case :
            Créer une grille en plaçant le candidat aux coordonnées
coordsCase dans la grille créer et l'ajouter au tableau grids
            Remplir au maximum la grille

      # Seconde étape : Vérification des résultats après remplissage de
grids

      Parcourir la ligne ou se trouve coordsCase,
      Parcourir la colonne ou se trouve coordsCase,
      Parcourir la région ou se trouve coordsCase.

      Pour chaque parcours :
            Pour chaque case au coordonnées différentes de celles de
case :
                  Vérifier si toutes les cases aux coordonnées de
coordsCase de chaque grille de grids sont identiques, donc place le
même candidat dans la case alors trouvé. Dans ce cas renvoie de la case
pour placement du candidat qui a été communément ajouté.
                  Sinon chercher le même candidat qui a été supprimé
entre la grille de départ et toutes les grilles contenues dans grids.
Dans ce cas renvoie de la case pour suppression du candidat qui a été
communément retiré.

                  Sinon continuer avec la case suivante

Fin du parcours de chaque case:
      renvoyer null
```

xxiii. Nishio

1. Illustration

Dans une situation de blocage, on utilise la méthode de Nishio qui est une manière plus simplifiée que l'heuristique qui suit le Nishio. La méthode consiste à supprimer un candidat d'une case dans le cas où ce dernier provoque le non-respect des consignes de résolution d'une grille du sudoku c'est-à-dire par la présence d'un doublon.



Par exemple dans la capture précédente, pour la première case on à la présence de deux candidats qui sont 5, et 8. On constate donc que si l'on place un 8 on aurait la grille suivante.

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

Ou l'on constate là où la case était grisée est en fait vide de candidat quand on place le candidat 8 dans la case rouge.

2. Pseudo-code

Voici le pseudo-code associé à la méthode de Nishio :

```
Pour chaque case ne contenant que 2 candidats :
      coordsCase <- coordoonées (x, y) de la case
      Pour chaque candidat de la case :
            Créer une grille avec la mise du candidat aux coordonnées
coordsCase
            Remplir la grille tant que le remplissage n'est pas bloqué
            Si aucune contradiction est présente alors
                  Passer au candidat suivant
```

```
            Sinon
                  Retourner la case ou l'on supprime le candidat qui
            provoque la contradiction
A la fin du parcours :
      Retourner null
```

xxiv. Test de chaque possibilité

1. Illustration

Cette méthode de résolution d'une grille de sudoku se réalise en dernier, et elle renvoie toujours un résultat si la grille possède une solution. Elle teste pour chaque case toutes les possibilités de façon récursive.

2. Pseudo-code

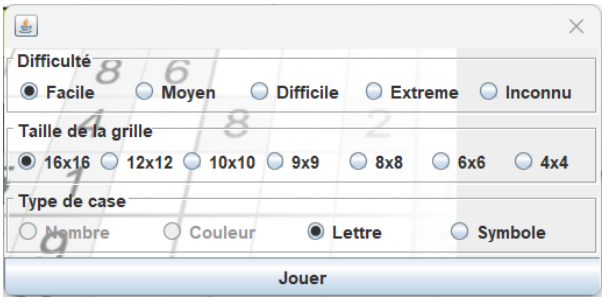
Voici le pseudo-code de l'heuristique qui représente le test de chaque possibilité :

```
Variables :
   -  Liste de case qui n'ont pas de valeur fixé : lst
   -  Position courante dans la liste lst : pos
Initialisation :
   -  lst est une liste après lecture de la grille du modèle contient
      toutes les cases qui n'ont pas de valeur fixées par l'utilisateur
      ou par la grille .
   -  pos est initialisé à 0.
Pour chaque candidat de lst(pos) :
      Placer le candidat
      Supprimer le candidat dans la ligne, la colonne et la région
      Si la grille ne contient pas de case avec 0 candidats :
            Continuer avec la case suivante (par appel récursif) et
attendre la réponse
            Si la résolution avec les autres cases est vaine :
                  Défaire et continuer avec le candidat suivant
            Sinon retourne une case parmi toutes celles qui n'étaient
pas fixées au départ pour placement de la valeur fixé à cette dernière
      Sinon
            Défaire et continuer avec le candidat suivant
      retourne null
```

4. La vue

a. Le lanceur du logiciel

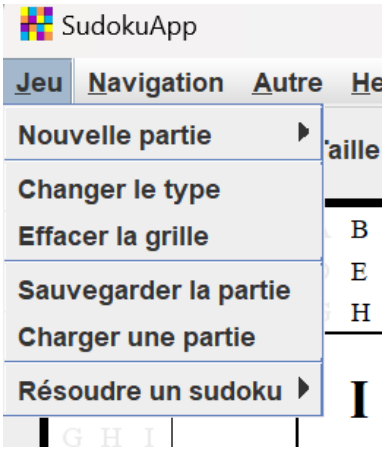
Lorsqu'on démarre le jeu, le lanceur de programme, ou launcher, apparaît à l'écran. L'utilisateur peut ensuite choisir entre commencer une nouvelle partie, en continuer une, ou modifier les options avant de jouer.



En sélectionnant une nouvelle partie, le joueur peut choisir : la taille de la grille, le niveau de difficulté et l'affichage des symboles dans chaque case. Le programme choisit ensuite, selon les critères choisis par l'utilisateur, Une grille au hasard.

b. Le menu du logiciel

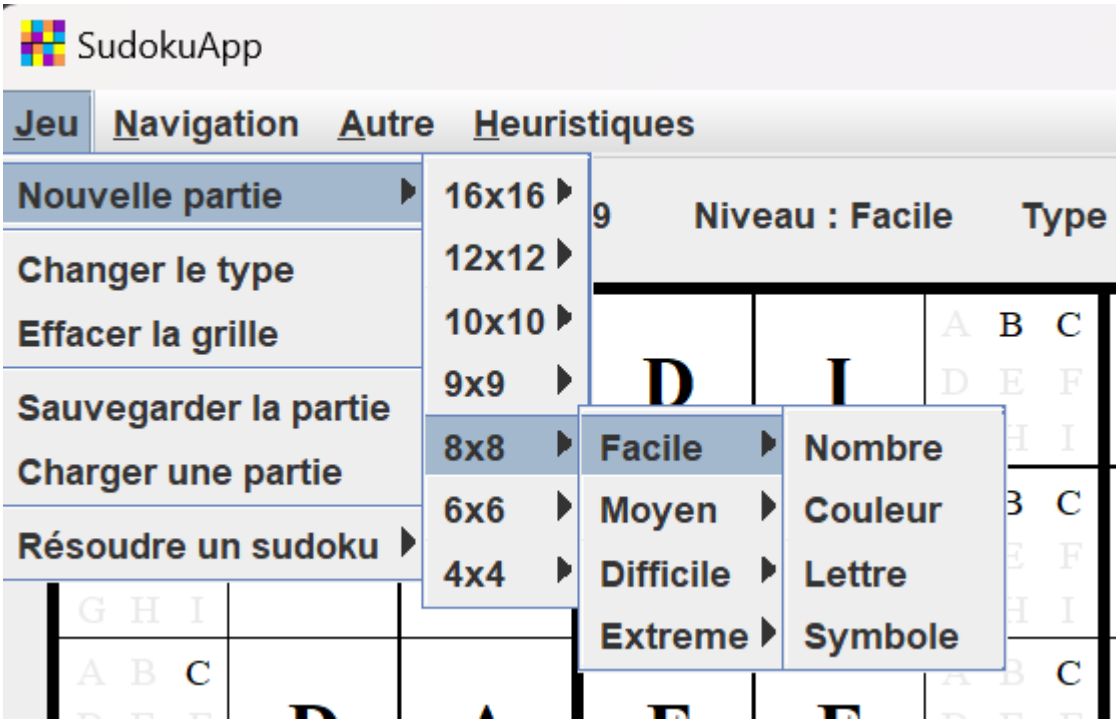
i. Jeu



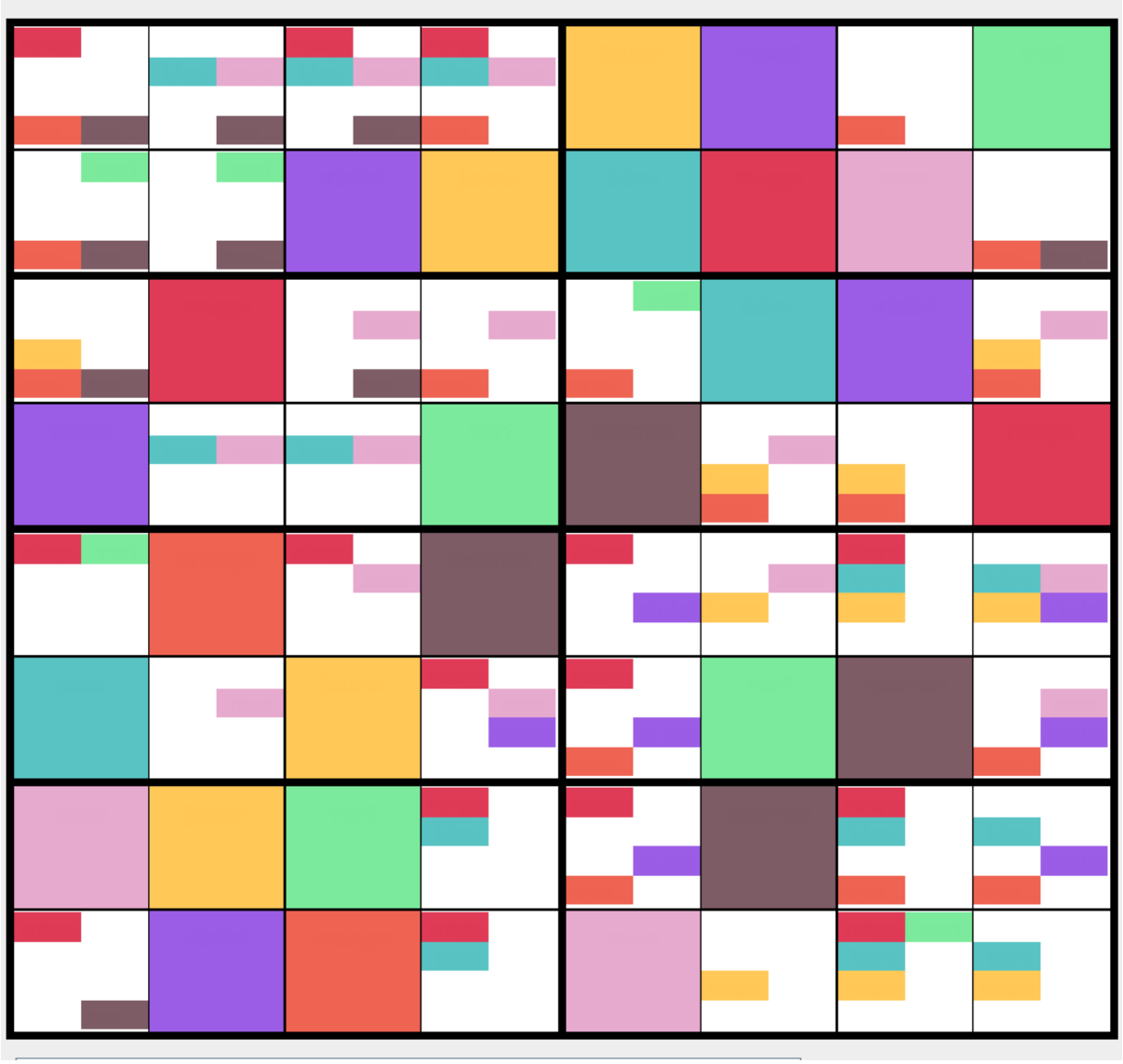
Concernant le jeu, on peut interagir directement avec le logiciel, pour la création d'une nouvelle partie, ou pour en sauvegarder ou pour résoudre un sudoku inexistant. A savoir les commandes qui risquent de toucher à la grille en cours demande une poursuite d'exécution à l'utilisateur.

1. Nouvelle partie

L'item "Nouvelle partie" crée une nouvelle grille de sudoku en fonction de la taille, du niveau et du type de données souhaitées en chargeant une grille de sudoku prédéfinies pour.



Par exemple, nous pouvons demande au logiciel de créer une grille de la taille 8x8, de niveau facile avec le type de données, couleur, ce qui nous donne la grille suivante :

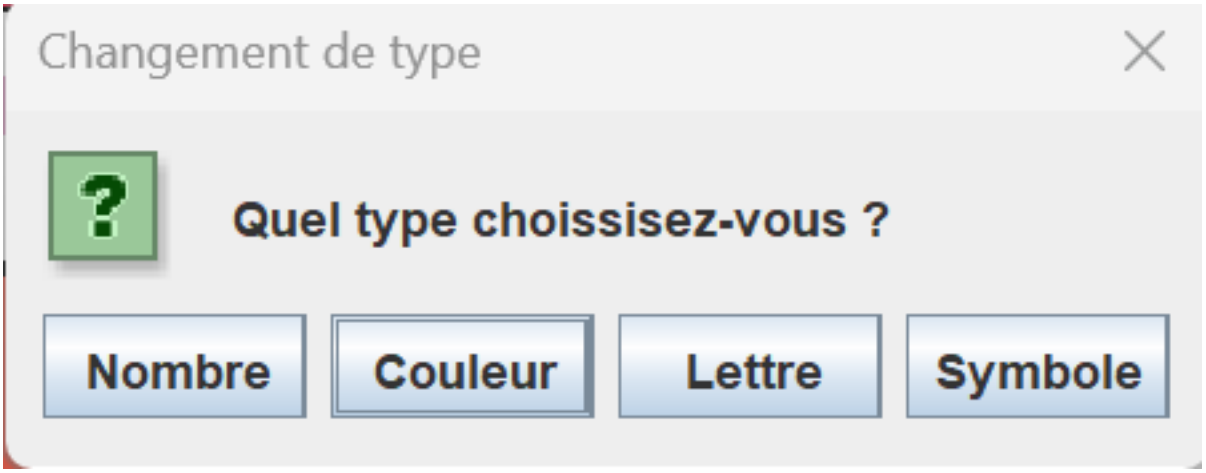


2. Effacer la grille

L'item “Effacer la grille” permet d’effacer toute action faite par l’utilisateur. Cela efface donc tout placement de candidat et revient à la grille de départ.

3. Changer le type

Le changement de type de données contenues dans la grille de sudoku dépend de la taille de notre grille de sudoku. Chaque taille ne permet pas la lisibilité parfaite pour chaque type de données. Mais le changement de type nous permet de passer d’un type couleur, d’un type lettre, et autre type de données.
Le type de données qui est en cours d’utilisation est mis en valeur.



4. **Sauvegarder une partie**

La sauvegarde d'une partie en cours permet de sauver notre partie en cours dans un fichier à part qu'on peut à nouveau la charger.

5. **Charger une partie**

Le chargement d'une partie réalise la lecture d'un fichier préalablement sauvegardé par l'utilisateur du logiciel. Elle permet de reprendre la ou la partie a été arrêtée.

6. **Résoudre une grille**

L'item "Résoudre une grille" permet de créer nous même une nouvelle grille et de permettre de savoir si la grille est soluble ou non. Dans le cas où le résultat est positif cela retourne la grille résolue.

ii. Navigation

Une implémentation d'un historique de commande réalisé par l'utilisateur a été mise en place afin de pouvoir annuler ou refaire une commande.

1. **Refaire**

La commande refaire permet d'aller plus loin dans l'historique des commandes et de réaliser à nouveau la commande qui a été préalablement annulée.

2. **Annuler**

La commande annuler permet de revenir vers le début de l'historique, afin d'annuler la commande qui vient d'être faite.

iii. Autre

1. Préférences

Les préférences sont les différentes options qui sont disponibles à l'utilisateur Elle permettent de modifier les informations suivantes :

- Le mode sombre ou clair
- L'utilisation forcée ou non des candidats, c'est-à-dire, en affirmative, si l'on souhaite placer un candidat dans une case il faut que ce candidat fasse parties des candidats de la case
- Le mode auto-complétion signifie que la grille se complète elle-même dès le début de la partie et lors des changements effectués par le joueur. Lors du placement d'un candidat dans une case, supprimera le même candidat qui se trouve dans la même ligne, la même colonne et la même région que la case. Pareillement pour le retrait d'une valeur fixée dans une case entraîne la remise des candidats. Cela évite de perdre du temps à tout remplir.

2. Comment jouer ?

Il s'agit d'un fichier en format PDF expliquant comment jouer au sudoku et en explicitant bien les différentes règles du sudoku à respecter.

3. Les différentes heuristiques

Il s'agit de ce fichier-ci, et il permet d'expliquer chaque heuristique et leur spécificités. Ces derniers sont expliquées dans la partie précédente.

4. Quitter

Il s'agit d'un moyen plus sûr de quitter le logiciel sans avoir oublier de sauvegarder une partie, ou évite d'avoir fait une fausse manipulation. Car elle demande à l'utilisateur une poursuite d'exécution. A l'affirmative l'application se ferme.

c. Le jeu

i. La barre d'information

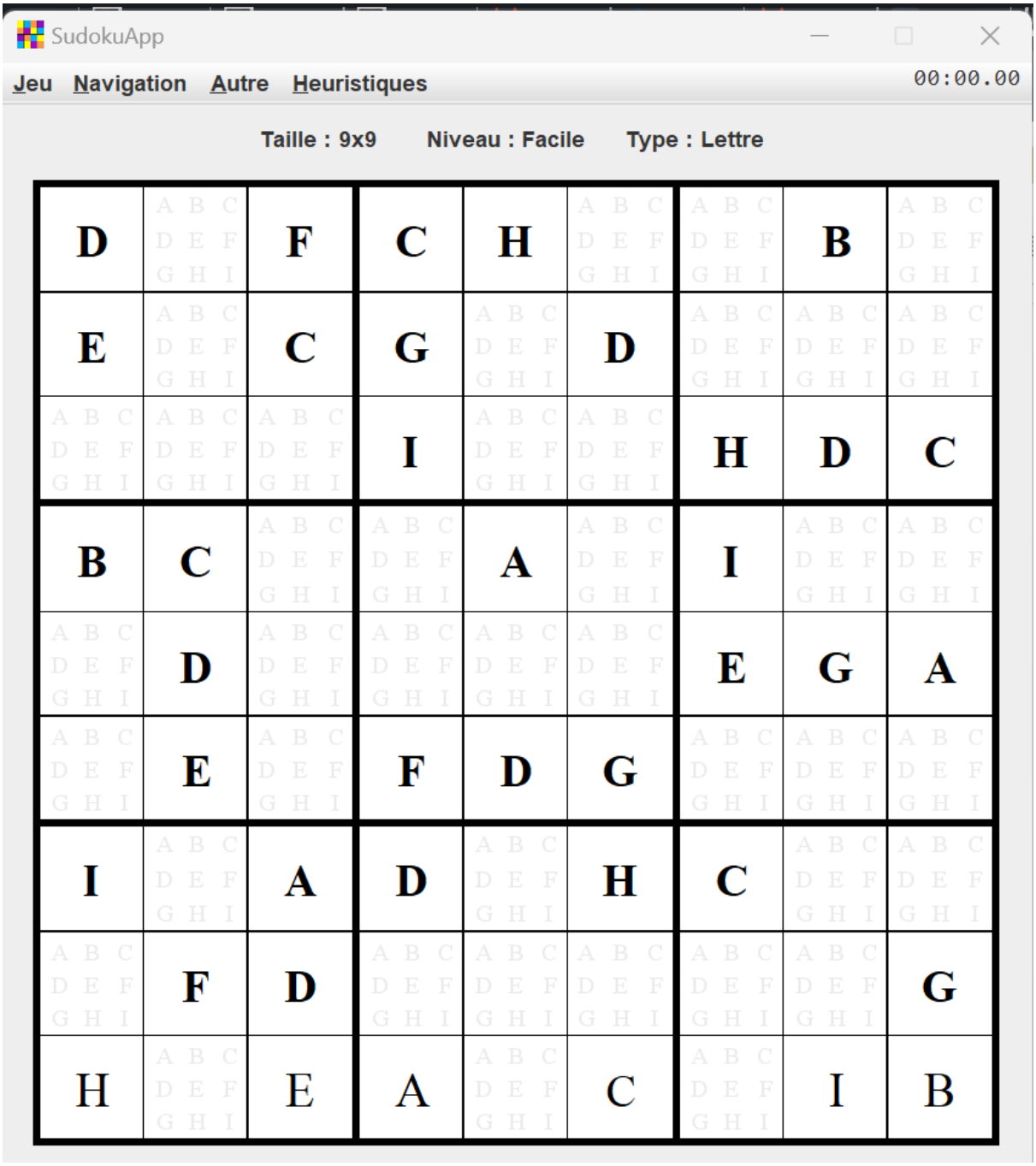
Elle contient trois informations qui définissent la grille qui est en cours d'utilisation qui sont :

- La taille de la grille
- Le niveau de difficulté de la grille
- Le type de données utilisées

Elles sont placé, juste au dessus de la grille, comme affiché sur la capture suivante :

ii. La grille

La grille est affichée au centre de la fenêtre de notre logiciel. Elle contient les différentes cases, et affiche séparément de façon distinctes les différentes régions. Voici un exemple d'affichage.



iii. La barre d’outils

La barre d’outils contient la barre de message d’aide quand l’utilisateur se retrouve bloqué il clique sur le bouton “Aide” et cela parcourt chaque heuristique jusqu’à obtention d’un message retour. Il contient également le bouton “Terminé”, qui termine le jeu si le joueur le souhaite.

Voici l’affichage de la barre d’outils qui se trouve en dessous de la grille sur le fenêtre :

d. Les parties

i. Les parties prédéfinies

Pour permettre aux joueurs de commencer une nouvelle partie, il faut une base de données contenant plusieurs parties de sudoku non commencé. Selon leur niveau de difficulté, Chaque partie est stockée dans un fichier XML, regroupant les parties du même niveau de difficulté. Ensuite, sous forme d'arborescence, Les parties sont mises dans la branche correspondant à leur taille de grille. Le format des grilles est presque identique à celle sauvegardée par le joueur, mais aucun candidat n'est choisi et toute valeur écrite dans une case est seulement fixée par la grille et non par le joueur.

ii. La sauvegarde

Afin de pouvoir continuer une partie ultérieurement, le joueur peut effectuer une sauvegarde de sa partie et choisir dans quel répertoire l’enregistrer. La grille ainsi que le chronomètre sont récupérés et convertis sous format XML. Le temps, divisé en heures, minutes, secondes, du chronomètre est sauvegardé. Pour la grille, chaque case est dans la ligne et dans l’ordre des colonnes dans lequel elle se situe sur la grille. La valeur affichée, son type (si c’est une valeur verrouillée ou écrite par le joueur) et les candidats choisis par le joueur, sont enregistrés.

iii. Le chargement

1. D’une nouvelle partie

Charger une nouvelle partie consiste à choisir le type de grille que l’on veut (taille, difficulté, type de symbole des cases). Le programme se charge ensuite de prendre au hasard une grille (voir les parties prédéfinies) correspondant aux critères

du joueur. Si c'est un changement de partie et non une partie lancée à partir du lanceur de programme : le chronomètre est remis à zéro, les données de la fenêtre de jeu sont rafraîchies et la barre de message d'aide est vidé. Les actions précédemment faites sont supprimés

2. D'une partie d'une partie existante

Après avoir choisi le fichier xml contenant la partie sauvegardé, le fonctionnement est quasiment identique à celui d'une nouvelle partie puisque le format XML d'une grille sauvegardé et celle d'une nouvelle partie est constitué des mêmes attributs. Le chronomètre est calibré au temps qui est sauvegardé, la configuration de chaque case est récupérée. Et comme pour une nouvelle partie, si c'est un changement de partie et non une partie lancée à partir du lanceur de programme : les données de la fenêtre de jeu sont rafraîchies et la barre de message d'aide est vidé. Les actions précédemment faites sont supprimés

e. Le chronomètre

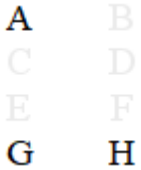
Un chronomètre, directement intégré en haut à droite de l'application, permet au joueur de mesurer le temps qu'il prend à résoudre un Sudoku. Le joueur peut choisir quand le démarrer et l'arrêter. Le chronomètre est sauvegardé et mis à jour en même temps que la grille de Sudoku lors d'une sauvegarde ou chargement d'une partie.

5. Le contrôleur

a. Les valeurs non fixées par l'utilisateur

Les valeurs dites non fixées par l'utilisateur sont des candidats, elle sont représentées par le type JLabel qui sont cliquables d'une manière différentes selon l'action recherchée :
Si l'on souhaite :

- Placer le candidat x dans notre case : On clique une fois sur le bouton droit de la souris
- Retirer le candidat de notre case : On clique deux fois sur le bouton gauche de la souris.
- Ajouter le candidat de notre case : On clique une seule fois sur le bouton gauche.



Chaque case ou une valeur n'est pas fixées ressemblent à ce genre de case, où :

- Un candidat qui ne fait pas partie des candidats de la case est grisé.

- Un candidat qui fait partie des candidats de la case apparaîtra noir ou blanc selon le mode clair ou non.

Chaque candidat va être mis en valeur quand la souris passe par-dessus.

b. Les valeurs fixées par l'utilisateur

Une valeur qui est fixée par l'utilisateur à la même allure que les valeurs fixées par la grille. En revanche à les mêmes aptitudes que les valeurs dites candidates.

6. Bonus

a. Optimisation

Les optimisations qu'on a pas ajouter sont les suivants :

- l'ajout des analyses statistiques de chaque partie jouées.

b. Difficultés

Les difficultés rencontrées sont les suivantes :

- Certaines fois, les explications des heuristiques n'ont pas été très claires d'où le fait que certains ont cherché sur internet d'autre sources.

7. Conclusion

Nous pensons que le projet qui a été réalisé respecte le souhait du client avec nos optimisations qui ont pu être ajoutées comme par exemple les différents types de données et les différentes tailles possibles. Pour le moment nous n'avions pas mis assez de parties prédéfinies et cela sera remédié lors de la soutenance.

8. Bibliographie

Voici les liens que l'on étudié durant nos développement du projet :

- Guide du Sudoku
- SudokuWiki.org
- Sudoku Techniques: Nishio
- Visualiser un fichier PDF avec l'api Acrobat viewer JavaBean
- Manipulation de XML avec JDOM et Java
- Overview (JDOM v2.0.6.1)