Rapport de projet

# SUDOKU

en C++

M'hamad Abbas
El Khamis Sadouioui
Mohamed Iheb Faiza

28 avril 2016

sous la direction de

Mr. Michel Meynard

UNIVERSITÉ DE MONTPELLIER
département informatique

# Table des matières

## Remerciements

Nous adressons nos remerciements aux personnes qui nous ont aidé dans la réalisation de ce projet.
En premier lieu, nous remercions M. Michel Meynard en tant que tuteur de projet il nous a guidé dans notre travail et nous a aidé à trouver des solutions pour avancer.
Nous remercions aussi l'université de Montpellier et la faculté des sciences pour l'environnement de travail qu'elles proposent et qui nous a permis de mener à bien ce projet.

# Introduction

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

Dans le cadre de notre L2 informatique, nous avons eu à réaliser en C++ un jeu de sudoku. Ce projet - réalisé tout au long du second semestre - doit permettre de mettre en oeuvre nos connaissances en toute autonomie, d'acquérir de nouvelles compétences et de travailler en groupe puisque nous étions trois étudiants. Ce rapport présente nos travaux.

Dans une première partie nous analyserons le sujet et présenterons les solutions algorithmiques que nous avons définies. Une deuxième partie détaillera l'ensemble des moyens mis en œuvre pour la réalisation de ce projet. Enfin, une partie sera consacrée à la gestion de projet qui nous a permis de mener à terme la réalisation du jeu.

# 1 Objectifs et enjeux

## 1.1 Cahier des charges

Le cahier des charges imposait la réalisation d'un jeu **utilisable**, se présentant sous la forme d'une **interface graphique** élégante et permettant de jouer avec **plusieurs niveaux de difficulté** sur des grilles carrés de **4x4, 9x9 ou 16x16 cases**. Nous devions également proposer une fonction permettant de guider l'utilisateur dans la résolution de la grille et d'une fonction permettant de sauvegarder une partie en cours.
La première étape consistait à créer une grille jouable. Pour ce faire, on devait générer une grille pleine puis vider ("*trouer*") certaines cases. Le choix de ces cases était un enjeu crucial. En effet, les cases trouées doivent laisser la solution unique, le nombre de cases trouées dans ces conditions modifieras la difficultés de la partie.
Un barème vraiment relatif a été choisi pour délimiter ces niveaux. Pour une grille de 9x9, le niveau facile correspond à 25 cases trouées sur 81, le niveau moyen correspond à 40 cases trouées et le niveau difficile à 55 cases trouées. Le choix de ce barème a un impact certain sur la satisfaction de l'utilisateur. Pour des sudoku de 4x4, les niveaux correspondent respectivement à 4,8 et 12. Pour les sudoku de 16x16, ils correspondent à 50,100 et 150 cases sur 256.

## 1.2 Conception algorithmique

### 1.2.1 Résolution d'une grille

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

Dans un premier temps, nous avons réalisé une fonction permettant de résoudre une grille. Celle-ci s'appuie sur la méthode dite de *backtracking*. Elle consiste à parcourir la grille de gauche a droite et de haut en bas et de tester les valeurs possibles.

Pour chaque case visitée, la fonction teste séquentiellement les chiffres de 1 à 9 et vérifie si la valeur est déjà présente sur la ligne, sur la colonne ou dans le bloc. Pour se faire nous avons créer trois méthodes *verifHorizontale,*

*verifVerticale et verifBloc* qui renvoyaient toutes un booléen, *vrai* si le chiffre testé était présent dans la zone de test et *faux* s'il était absent.

Lorsque la valeur créé un conflit, alors la fonction incrémente la valeur testée. Si aucun chiffre n'est disponible (*toutes les possibilités testées*) alors la fonction teste la case précédente en reculant et en lui affectant un nouveau chiffre. Une fois ces tests terminés, notre grille est résolue.

### 1.2.2 Génerer des grilles de jeu aléatoires

La seconde étape de ce projet consiste à réaliser une grille de sudoku. Dans un premier temps nous avions pensé à procéder de la manière suivante : Nous avions conçu une méthode nommée *Générer*, cette méthode parcourait notre Sudoku de la même manière que le *backtracking*, c'est à dire à partir des cordonnées (0,0) de gauche à droite et de haut en bas, et sa stratégie était la suivante.
Pour chaque case visitée il génèrait une valeur de manière aléatoire entre 1 et 9 et il verifiait que cette valeur n'apparaîsse ni dans la ligne ni dans la colonne ni dans le bloc grâce aux méthodes *verifLigne verifBloc et verifHorizon*. Si aucune valeur entre 1 et 9 n'était accepté, alors il reculait d'une case et il affectait une nouvelle valeur à la case précédente.
Cependant cette manière de faire ne fonctionnait pas, en effet les sudoku générés contiennent beaucoup trop souvent de "schémas" identiques comme dans l'exemple suivant. Ces schémas amènent plusieurs problèmes, le premier étant que les sudokus n'étaient pas agréable à jouer et qu'au bout de quelques chiffres placés, le schéma récursif devenait évident et la grille n'avait plus d'intérêt.

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

Le second était que ces grilles étaient peu trouables, nous avons rarement atteint plus de 35 trous pour un sudoku de taille 9, ce problème etait dû au fait que la génération se faisait de manière pseudo-aléatoire. Un ordinateur étant une machine déterministe il est très difficile de lui faire générer des nombres totalement aléatoire.

Pour palier à ce soucis nous avons augmenter le nombre de facteur aléatoire. Au tout début de notre algorithme nous générions donc *25* valeurs aléatoires que nous placions à *25* endroits aléatoires, pour un sudoku de taille 9, tout en vérifiant qu'elles ne créaient pas de conflit *(exemple deux fois le même chiffre sur la même ligne)*. Une fois cette partie faite, nous faisions appel à notre algorithme de résolution de sudoku qui résolvait cette grille et par conséquent générer un sudoku à solution unique.

### 1.2.3 Trouer une grille complète

Dès lors que nous étions capable de générer une grille pleine et de résoudre un sudoku, il restait à le trouer.. Trouer un sudoku c'est enlever la valeur d'une case pour laisser le joueur la remplir lui même. La plus grande difficulté de cette partie était de garder l'unicité de la solution. Pour cela nous travaillions avec principalement deux méthodes, *Trouer* et *ResoudreBêtement* que nous allons expliquer maintenant.

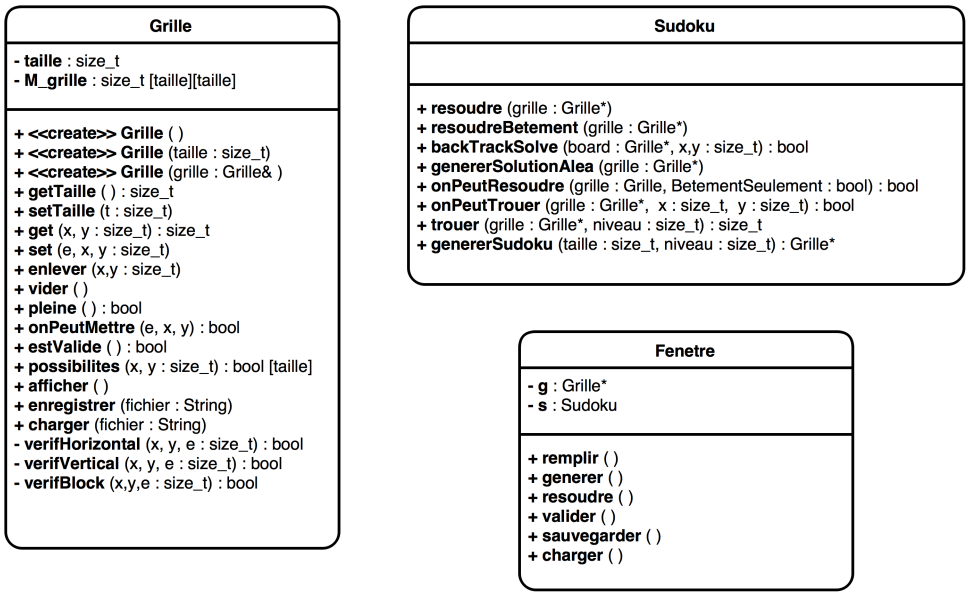
Notre première méthode, *ResoudreBêtement*, remplissait la grille avec, pour chaque case, la seule valeur disponible. Si plusieurs possibilités existaient pour cette case, cela impliquait donc que la solution n'était pas unique.
La seconde méthode, *trouer*, parcourait notre sudoku case par case de manière aléatoire et à chaque case elle faisait appel à *ResoudreBêtement* afin de savoir lors du retrait de la case concernée, si d'autres solutions seraient créées.
Pour éviter de repasser plusieurs fois sur la même case *(car le trouage de certaines cases empêche l'unicité de la solution)*, nous utilisions donc un tableau à double entrée de Booléen nommé *Vu*. Le tableau etait entièrement initialisé à *faux* et lorsque l'on essaiyait de trouer une case, la case correspondante du tableau *Vu* à *vrai*.
Pour résumer on a choisit des cases de façon aléatoire, que l'on a trouées puis on a vérifié par la méthode *RésoudreBêtement* si la solution générée etait identique au sudoku de base pour conserver ou non le trou effectué.

C'est aussi ici qu'est géré la difficulté, en effet l'une des conditions d'arrêt de notre algorithme est le nombre de trous poduit,qui dépend de la difficulté choisit.

# 2 Partie Programmation

## 2.1 UML

Voici notre diagramme UML.
L'UML(Unified Modeling Language) est un langage de modélisation graphique universel apparut au début des années 1980. Il permet de simplifier grandement les programme orienté objet car il résume toutes les classes, méthodes et attributs nécessaires sous forme graphique.
Ainsi ici chacune de nos classes est représentées par un bloc, en haut nous avons nos attributs et en bas nos méthodes et fonctions. Notre programme se décompose en trois classes distinctes *Grille*, *sudoku*, et *Fenetre*. La classe Grille génère des grilles vides de tailles 4, 9 ou 16 et contient aussi toutes les méthodes relatives à l'entrée de données par l'utilisateur.
La classe sudoku contient les méthodes utilisées pour créer le sudoku toutes les méthodes utilisent et/ou renvoient une instance de grille. Nous avons par exemple de quoi générer et résoudre un sudoku. La classe Fenêtre elle, gère toute la partie graphique lié à Qt.



## 2.2 C++

Le langage de programmation **C++** est un langage normalisé pour la première fois en 1989, depuis plusieurs autres standards sont parus. Le dernier, celui que nous utilisons, date de 2011. Ce langage n'appartient à personne ainsi nous pouvons l'utiliser librement.

Nous avons utilisé le paradigme objet, car cela nous semblait indispensable pour ce projet, il nous a permis de facilement décomposer notre programme en plusieurs grandes classes qui travaillent entre elles.

Le choix de ce langage pour écrire notre programme s'explique d'abord par le fait que ce langage nous est familier, en effet nous l'utilisons depuis plusieurs années maintenant au sein de nos études, mais c'est aussi le premier langage de programmation objet que nous avons appris.
Les nombreux avantages qu'offre le C++ nous ont aussi poussé à le choisir. Un outil important est le **GDB**, cet outil nous permet de savoir d'où vient le problème en exécutant notre programme instruction par instruction, ainsi il trouve facilement quelles lignes provoquent un bogue. Et lorsque le **GDB** ne suffit pas à repérer toutes les erreurs, la sortie terminal qu'offre ce langage et qui est très facile d'utilisation, prend le relais. Bien utilisée, elle permet facilement d'identifier la ligne du programme à laquelle il bogue *(par exemple au niveau des boucles infinies)*.
Le **C++** nous offre un large panel de types, il nous permet même de créer nos propres types, cela améliore grandement le coût en mémoire et par conséquent les performances de notre algorithme. Au sein de ce projet, pour remplir nos grilles nous avions besoin de chiffres ne dépassant jamais 16, ainsi au lieu d'utiliser un *short int* qui coûte en mémoire deux octets, nous pouvions créer nos propres entiers qui aurait coûtés seulement quatre bits.

## 2.3 Qt

La raison principale nous poussant à utiliser le framework Qt pour travailler sur l'interface graphique était la large documentation disponible le concernant, facilitant ainsi son utilisation. De plus Qt est portable et nous permettait de passer aisément d'un système Windows à un système Linux.

La première étape pour créer l'interface était bien evidemment de l'imaginer clairement, afin de mieux organiser l'implémentation. Aprés concertation avec le groupe, nous avons convenu de proposer à l'utilisateur cinq bouttons :

- Générer *(créé une grille en fonction d'une taille et d'un niveau de difficulté)*
- Résoudre *(Résout la grille courante)*
- Valider *(Teste les valeurs entrées par l'utilisateur)*
- Sauvegarder *(Sauvegarde la dernière instance de la grille)*
- Charger *(Charge la grille précédemment sauvegarder)*

Ainsi que deux CheckBox à choix unique pour choisir la taille du sudoku et le niveau de difficulté. Tout ces objets sont initialisés dans la classe *fenêtre* sous la forme de *Widgets*.

L'utilisation d'un *QGridLayout* pour mettre en forme la grille de sudoku, semblait être la meilleure solution. En effet il s'agit d'un tableau *virtuel* de taille *dynamique* dans lequelle on place des Widgets. Le principe était donc de remplir ce *QGridLayout* avec deux *Widgets* différents selon s'il s'agit d'une case d'ores et déjà remplie ou d'une case que l'utilisateur doit remplir. Cependant son initialisation devait dépendre du sudoku courant, ainsi elle était effectuée par le biais d'une méthode *remplir* à chaque génération *(ou chargement)* de grille, mais toujours précédée de la destruction de sa précédente instance pour éviter une superpostion des différentes grilles.

Pour conclure, la première qualité de Qt n'est pas son intuitivité mais sa facilité d'utilisation aprés la longue periode de prise en main.

# 3 Gestion de projet

Pour notre projet nous étions initialement quatre, mais l'un de nos camarade, Léo NORMAND, nous a quitté suite à l'arrêt de ses études. Pour commencer nous avions décidé que Léo NORMAND et Mohamed IHEB devaient s'occuper de la partie génération de sudoku et des heuristiques *(Qui sont finalement absents dans le programme)*. El khamis SADOUIOUI et M'hamad ABBAS devaient eux, s'occuper du trouage de sudoku.

La génération étant finie avant que le trouage ne soit fonctionnel il nous fallait nous organiser différement pour le bon déroulement du projet. De ce fait et après une rencontre avec notre encadrant Monsieur Michel MEYNARD, nous avons décidé d'affecter M'hamad ABBAS à la partie graphique. Mohamed IHEB et El khamis SADOUIOUI eux, devait se documenter et modifier la génération et le trouage afin de les rendre totalement fonctionnels.

Nous faisions régulièrement des points d'avancement afin de rester sur la même ligne directive malgré les différentes tâches que nous nous étions confié. Quant au rapport, une première version fût rédigée par Mohamed IHEB puis El Khamis SADOUIOUI s'est joint à la rédaction. Une fois la partie graphique achevée, M'hamad ABBAS s'est attelé à mettre le rapport au format LaTeX. **Google drive** nous a tous permis de travailler sur le même fichier.

Afin de mener notre projet à bien nous utilisions l'éditeur de texte **Atom**, nous compilions avec **GNU Compiler Collection**.
Nous mettions notre travail en commun à distance à l'aide de **Google Drive**. Cet outil nous permettait d'avoir un suivit sur les modifications apporter à un fichier et le système de commentaire qu'il offre pour pouvoir modifier ou faire une remarque sans rendre le fichier illisible était trés utile.
Pour rendre compte de l'avancement de notre travail à notre encadrant nous étions tenu d'utiliser **GitLab**.

# Conclusion et apport personnel

Notre collaboration régulière nous a permis de tous nous intéresser aux différents aspects de se projet. Ce fut donc l'occasion pour nous d'approfondir et consolider nos connaissances et de les mettre en pratique. Le travail en groupe dans un projet collaboratif est une première vraie expérience impliquant de s'intéresser sérieusement à la gestion de projet et aux outils disponibles (Gitlab, google drive). Enfin, un projet de cette envergure impose également une certaine constance dans les efforts réalisés.
Nous gardons un ressenti très positif à l'égard de cette expérience et avons pleinement conscience de l'apport qu'elle représente pour nous, tant au niveau académique (connaissances, mise en œuvre) que professionnelle (première expérience très valorisante). Nous tenons également à souligné que l'aspect ludique d'un tel projet aide largement les étudiants à en tirer profit.