

Julie AUSSEIL
Mateus SHIRLAW

Université Paris Ouest Nanterre La Défense.
L3 MIAGE

Projet du module Harmonisation des acquis

Le Sudoku

Sommaire

1. Contexte
2. Objectif
3. Présentation du Sudoku
4. Notre projet
 - 4.1. Découpage du projet
 - 4.2. Choix d'architecture
 - 4.3. Fonctionnalités implémentées
 - 4.4. Fonctionnalités prévues mais non implémentées
 - 4.5. Les raisons des différents échecs
 - 4.6. Les difficultés rencontrées
 - 4.7. Est-ce que le langage C était adapté pour ce projet ?
 - 4.8. Perspectives et améliorations
 - 4.9. Bilan personnel
 - 4.10. Installation et utilisation du projet

1. Contexte

Ce projet intervient dans le cadre du cours d'Harmonisation des acquis, visant à nous familiariser davantage avec le langage c.

Plus précisément, cela a pour but de nous impliquer dans un projet afin d'améliorer nos capacités à concevoir et développer une application.

Le travail en groupe est également un point important car il nécessite une bonne organisation et communication au sein des membres du groupe ainsi qu'une attitude sérieuse et professionnelle.

2. Objectif

Notre projet consiste à concevoir et créer, en c, un générateur et solveur d'une grille de Sudoku (9x9) avec plusieurs niveaux de difficulté disponibles.

De plus, il faudra fournir un rapport complet contenant notamment des informations techniques sur le projet.

3. Présentation du Sudoku

Le Sudoku est un jeu de logique se présentant sous la forme d'une grille 9x9, c'est-à-dire constituée de 9 lignes et 9 colonnes. La grille est également constituée de 9 carrés de 9 cases appelés régions.

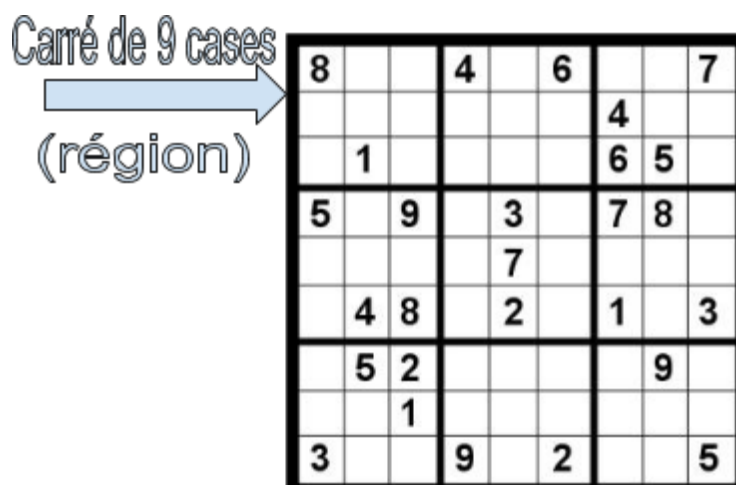


Figure 1 : présentation d'une grille de Sudoku

Le but du jeu est de remplir cette grille de chiffres allant de 1 à 9, sachant qu'un même chiffre ne doit figurer qu'une seule fois par ligne, une seule fois par colonne et une seule fois par région. Autrement dit, chaque ensemble doit contenir tous les chiffres de 1 à 9.

Au début du jeu, une vingtaine de chiffres sont déjà placés dans la grille comme ci-dessus et il suffit de remplir la grille comme il convient. Une grille de Sudoku ne possède qu'une seule solution possible.

Voici 3 exemples d'erreurs :

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

Figure 2 : différents exemples d'erreurs

Remplir une grille de Sudoku n'est qu'une question de logique et non d'arithmétique. Plusieurs niveaux de difficulté existent dans la résolution d'une grille.

Cette difficulté varie en fonction de 3 principaux facteurs :

- la taille de la grille
- le nombre de cases pré-remplies
- la disposition des chiffres

Ci-dessous voici une grille de Sudoku remplie :

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

Figure 3 : grille de Sudoku entièrement remplie

4. Notre projet

Une bibliothèque graphique (SDL.h) sera utilisée pour créer la grille du sudoku, ainsi qu'une interface avec un menu permettant de choisir le niveau de difficulté de la résolution du Sudoku.

Les outils utilisés pour la réalisation de ce projet sont VirtualBox avec un environnement Ubuntu.

4.1. Découpage du projet

Mateus SHIRLAW

Julie AUSSEIL

Module création et résolution du Sudoku

Structure et fonctions Valeurs:

```
typedef struct Valeur Valeur;  
struct Valeur{  
    int ligne;  
    int colonne;  
    int v_Possible[9];  
};
```

```
int *valeurLigne(int ligne, Sudoku s)  
int *valeurColonne(int colonne, Sudoku s)  
int *valeurRegion(int ligne, int colonne, Sudoku s)  
void melange_tab(int *tab)  
Valeur possible(int ligne, int colonne, Sudoku s,int melange)  
int nb_case_vide(Sudoku s)  
int nb_valeur(Valeur v)  
int fonction_aleatoire(int a)
```

Structure et fonctions Sudoku:

```
typedef struct Sudoku Sudoku;  
struct Sudoku{  
    int grille[9][9];  
};
```

Fonctions de test sur la validité d'une grille de Sudoku:

```
int ligneEstValide(int ligne, Sudoku s)
int colonneEstValide(int colonne, Sudoku s)
int regionEstValide(int ligne,int colonne, Sudoku s)
int aucune_erreur(Sudoku s)
int egale(Sudoku S1,Sudoku S2)
```

Structure et fonctions Arbre:

```
typedef struct Arbre_Sauvegarde Arbre;
struct Arbre_Sauvegarde
{
    Sudoku sudoku;
    Arbre *fils[9];
};
```

```
Arbre *init_abr(Sudoku s)
Void DesalouerArbre(Arbre *abr)
void searchSudoku(Arbre *abr,Arbre *b)
void resolution(Arbre *abr,int *solution)
void creation(Arbre *abr,int *solution)
Arbre *init_null()
Sudoku nettoyer(Sudoku s, int numNiveau)
```

Module interface graphique et affichage

Fonctions affichage fenêtre:

```
void afficherTexte(SDL_Surface *ecran, int x, int y, char *nb, int taille)
void ligneHorizontale(int x, int y, int w, SDL_Surface *ecran)
void ligneVerticale(int x, int y, int h, SDL_Surface *ecran)
void afficher_grille(SDL_Surface *ecran, Sudoku s)
```

Fonctions affichage console:

```
void affiche_sudoku(Sudoku s)
void affiche_valeur(Valeur v)
void affiche_arbre(Arbre* abr)
```

Fonctions gestion des fenêtres :

```
void fenetre_resolution(SDL_Surface *ecran, Sudoku s)
void fenetre_grille(SDL_Surface *ecran, Sudoku sudoku)
void gerer_difficulte(SDL_Surface *ecran, int numNiveau)
void ouverture_fenetre_accueil(SDL_Surface **ecran)
void pause(SDL_Surface *ecran, Sudoku s2)
```

Fonction initialisation:

```
void initialiser_TTF()
```

4.2. Choix d'architecture

De part nos derniers projets, nous avons commencé par une conception très objet. C'est-à-dire un sudoku composé de colonnes, de lignes avec une structure case etc. Mais nous nous sommes rendu compte que cela nous complexifierait la tâche. Donc nous nous sommes tournés vers une conception plus simple avec une structure Sudoku avec uniquement un tableau d'entier de taille 9x9 et une structure Valeur qui conserverait les valeurs possibles d'une case ainsi que les lignes et colonnes correspondantes à celle-ci.

```
typedef struct Sudoku Sudoku;
struct Sudoku{
    int grille[9][9];
};
```

```
typedef struct Valeur Valeur;
struct Valeur{
    int ligne;
    int colonne;
    int v_Possible[9];
};
```

Ensuite, nous nous sommes demandés comment nous allions résoudre le sudoku. Nous avons une première idée de faire une résolution avec une fonction récursive. Mais nous nous étions dit que nous n'aurions pas le nombre de solutions qui est nécessaire pour créer une grille avec une solution unique.

Nous sommes partis dans l'idée de faire un arbre de "sauvegarde" du sudoku. Il est composé d'un sudoku et de 9 fils.

Donc par exemple, si la première case du sudoku a comme valeur possible le 1 et le 7. On va changer le sudoku du fils[0] et placer à sa première case le 1. Et le fils[6], on placera le 7 à la première case.

Dans le cas où nous n'avons aucune valeur possible, les fils pointeront sur NULL. De cette manière, le fils dont le sudoku est plein sera donc la grille résolue.

```
typedef struct Arbre_Sauvegarde Arbre;
struct Arbre_Sauvegarde
{
    Sudoku sudoku;
    Arbre *fils[9];
};
```

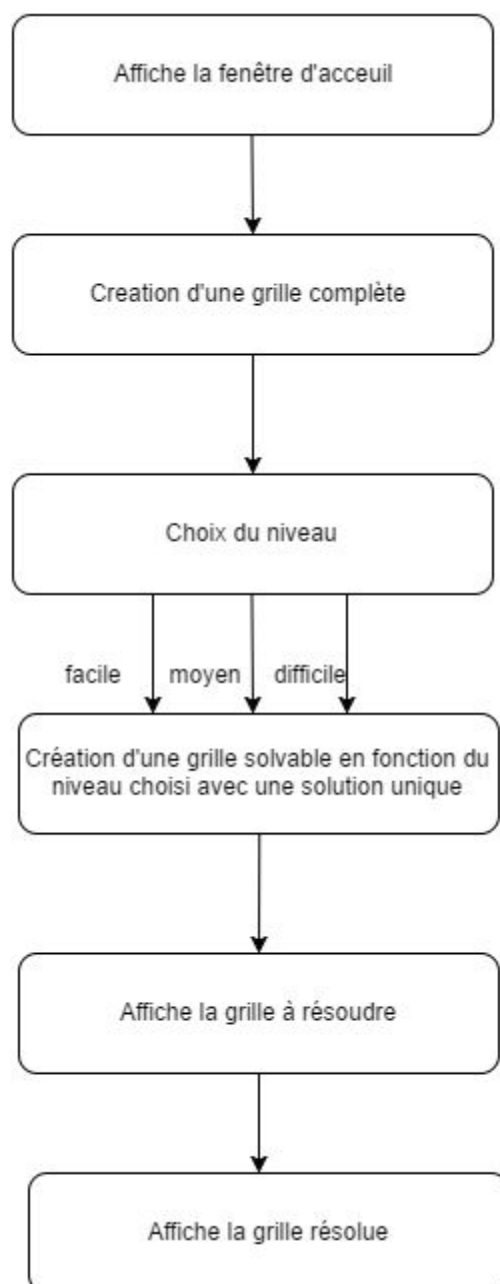


4.3. Fonctionnalités implémentées

Nous avons implémenté **la création d'une grille de Sudoku avec une solution unique** à partir d'une grille composée que de zéros en fonction de la difficulté choisie, ainsi que sa résolution.

Nous avons créé une interface graphique pour l'utilisateur qui affiche la grille dans une fenêtre et qui propose à l'utilisateur de choisir un niveau de difficulté pour la résolution du Sudoku.

Nous affichons également les différentes étapes de la création et de la résolution de la grille dans la console.



4.4. Fonctionnalités prévues mais non implémentées

Nous avions prévu de proposer à l'utilisateur de jouer en remplissant lui-même la grille du Sudoku à l'aide d'une saisie à l'écran.

4.5. Les raisons des différents échecs

Nous n'avons pas implémenté l'interface de jeu pour l'utilisateur principalement par manque de temps.

Aussi, pour proposer cette fonctionnalité, il aurait fallu créer nous-même en dur dans le code une grille de Sudoku puis enlever des valeurs afin que l'utilisateur les remplisse. Or ce n'est pas la solution que nous avons choisi.

4.6. Les difficultés rencontrées:

Nous avons rencontré des difficultés pour faire la conception du projet avant de commencer à coder. Nous avons d'abord proposé une solution avec plusieurs structures (Ligne, Colonne, Région...) qui était trop complexe, il a donc fallu refaire une conception simplifiée.

-Nous avons eu du mal à installer la librairie SDL car nous avons tout d'abord essayé sur Windows et cela nécessitait beaucoup de configurations. Ensuite nous avons décidé d'utiliser Linux, il a donc fallu installer une virtual Box et installer SDL.

-Nous n'avions jamais utilisé SDL avant, il a donc fallu apprendre à l'utiliser.

-Nous avons eu des difficultés sur l'implémentation de l'arbre

4.7. Est ce que le langage C était adapté pour ce projet ?

Le langage C dans ce projet était adapté dans le sens où nous étions à l'aise techniquement. Nous maîtrisons le langage qui nous a facilité l'implémentation et l'utilisation de structures telle que l'arbre pour résoudre le sudoku. Le langage C pour un petit projet comme celui-là convient amplement.

Mais nous sommes mis d'accord sur le fait que l'utilisation d'un langage de programmation orienté objet comme le c++ aurait été plus agréable et aurait permis une meilleure organisation du code.

4.8. Perspectives et améliorations

Pour améliorer notre projet, nous pourrions ajouter des fonctionnalités pour que l'utilisateur remplisse lui-même le Sudoku.

Nous pourrions aussi rajouter dans l'interface graphique les différentes étapes de résolution de la grille de Sudoku.

4.9. Bilan personnel

Julie :

Ce projet a été intéressant pour moi car il m'a permis de découvrir la librairie SDL et d'apprendre à l'utiliser.

Si je devais refaire le projet, je passerais davantage de temps à coder le sudoku et sa création et résolution car l'interface graphique m'a pris beaucoup de temps.

Mateus :

Ce projet m'a permis de bien comprendre comment fonctionne un arbre et pouvoir utiliser une structure de type arbre autre qu'avec un arbre binaire.

Mon erreur était de penser que faire un arbre était une mauvaise idée et trop compliquée pour le projet. Mais nous avons été vite rectifié lors du rendez-vous du point projet où vous nous avez incité à continuer notre idée sur l'arbre.

Et si je devais refaire le projet, j'essaierais d'optimiser le programme pour une résolution et une création plus rapide et de profiter ainsi de faire un peu de SDL.

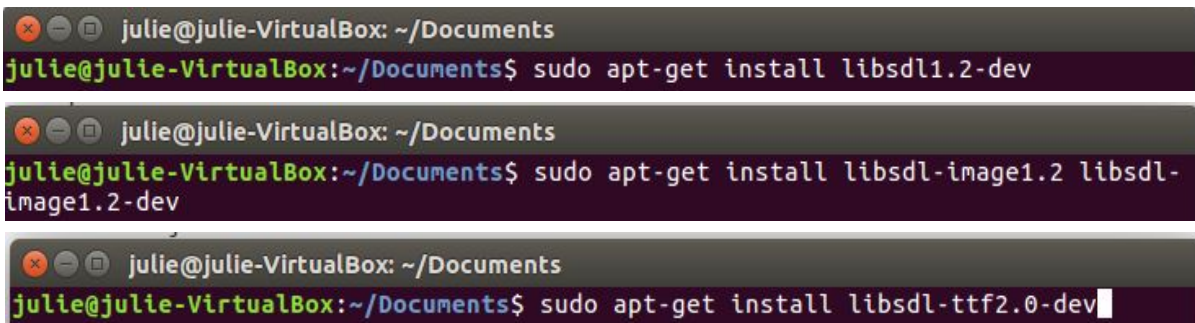
4.10. Installation et utilisation du projet

Installation:

Pour lancer notre projet, il faut tout d'abord installer la librairie graphique SDL, ainsi que SDL_image et SDL_ttf.

Sous Linux, nous avons réalisé une commande bash pour installer, il suffit de taper `bash install.sh` dans le répertoire du projet. Dans cette commande contient les commandes d'installations et exécution du programme (cf voir README)

Ou sinon ces trois commandes ci-dessous dans la console afin d'installer SDL, SDL_image et SDL_ttf :

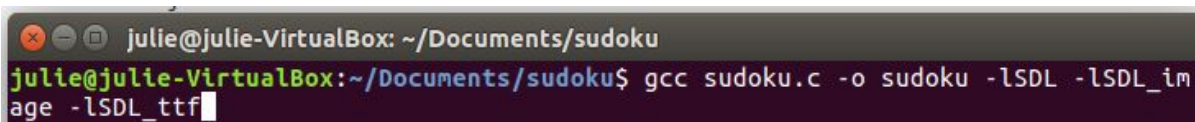


```
julie@julia-VirtualBox: ~/Documents
julie@julia-VirtualBox:~/Documents$ sudo apt-get install libsdl1.2-dev

julie@julia-VirtualBox: ~/Documents
julie@julia-VirtualBox:~/Documents$ sudo apt-get install libsdl-image1.2 libsdl-image1.2-dev

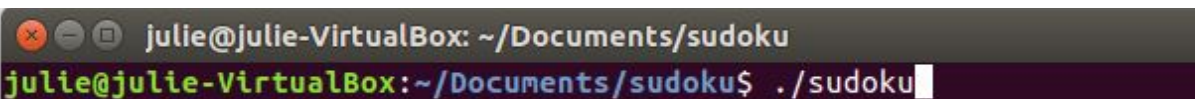
julie@julia-VirtualBox: ~/Documents
julie@julia-VirtualBox:~/Documents$ sudo apt-get install libsdl-ttf2.0-dev
```

Pour compiler le programme on exécute la commande :



```
julie@julia-VirtualBox: ~/Documents/sudoku
julie@julia-VirtualBox:~/Documents/sudoku$ gcc sudoku.c -o sudoku -lSDL -lSDL_image -lSDL_ttf
```

Puis pour exécuter le programme :



```
julie@julia-VirtualBox: ~/Documents/sudoku
julie@julia-VirtualBox:~/Documents/sudoku$ ./sudoku
```

Utilisation:

Le programme est très simple. Il faut choisir le niveau puis cliquez sur lancer et il affichera une grille de sudoku à résoudre en fonction du niveau choisi. Et pour finir appuyez sur Résolution pour afficher la solution du sudoku.

Annexe :

Cahier des charges :

Projet Sudoku	Version 1.0
Cahier des charges : Sudoku	27/09/16
Mateus Shirlaw, Julie Ausseil	

1. Introduction

Ce document décrit le cahier des charges du projet Sudoku du module d'Harmonisation des acquis. Il s'agit de créer un générateur et solveur de Sudoku.

Durant ce projet nous avons programmé en langage c.

Ce projet est effectué en binôme.

2. Pré-requis

Pour mettre en place ce projet, nous avons besoin d'un IDE (Netbeans), d'un logiciel de gestion de version - Github, ainsi que d'une librairie graphique (SDL).

En ce qui concerne les compétences nécessaires, nous devons être aptes à concevoir et développer un projet en c.

Nous devons définir les fonctionnalités nécessaires à la résolution de ce projet.

3. Spécifications fonctionnelles

3.1. Menu

L'utilisateur lance le jeu. Une interface d'accueil s'affiche.

L'utilisateur a la possibilité de choisir le niveau de difficulté (voir 3.2.) ou de lancer le générateur de Sudoku.

Pour lancer le générateur, l'utilisateur clique sur un bouton "Lancer", qui ouvrira une nouvelle fenêtre.

3.2. Choix de niveau

Pour choisir le niveau de difficulté, la demande est accessible via des boutons dans l'interface d'accueil avec 3 niveaux disponibles (facile, moyen, difficile).

3.3. Générateur de grille 9x9

Lors de la création de la grille de Sudoku, l'utilisateur n'a pas de possibilité d'action sur l'application.

La grille se génère automatiquement et pseudo-aléatoirement à l'aide d'une interface graphique. Le nombre de cases pré-remplies et la disposition de ces cases dépendront du niveau de difficulté choisi par l'utilisateur.

Une fois la grille générée, l'utilisateur a deux choix :

- il peut cliquer sur un bouton pour lancer la résolution automatique de la grille.
- il peut cliquer sur un autre bouton pour résoudre lui même la grille de Sudoku

3.4. Résolution de la grille automatique

L'application résout automatiquement la grille de Sudoku.

L'utilisateur n'a pas la main sur cette résolution.

Une fois la grille entièrement remplie, l'utilisateur a la possibilité de retourner sur la page d'accueil.

3.5. Résolution de la grille par l'utilisateur

L'utilisateur doit remplir la grille du Sudoku.

Pour cela, il clique sur une case vide à remplir et effectue la saisie au clavier d'un chiffre allant de 1 à 9.

Si le chiffre renseigné est correct, il s'affiche en noir à l'écran.

Si le chiffre est incorrect, il s'affiche en rouge. L'utilisateur doit alors re cliquer sur la case pour la remplir avec le chiffre attendu.

Une case préalablement remplie (soit par l'algorithme soit par l'utilisateur, si elle est correcte) ne peut pas être modifiée.

Une fois la grille entièrement remplie, l'utilisateur a la possibilité de retourner sur la page d'accueil.

4. Charte graphique

4.1. Menu & Choix de niveau

Projet Sudoku

niveau de difficulté :

facile

moyen

difficile

LANCER

4.2. Générateur de grille 9x9

Facile (0-10)

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

www.olesur.com

RÉSOUTRE

JOUER

4.4. Résolution de la grille automatique

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

QUITTER

4.5. Résolution de la grille par l'utilisateur

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

ajout d'un chiffre de 1 à 9 par l'utilisateur après avoir cliqué sur la case

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

QUITTER