Suratteau Maxime

Projet LP25 P19

Projet 2 : Programmer le jeu « Smart Sudoku ».

Énoncé du sujet :

Le jeu « Sudoku » consiste à compléter une grille carrée divisée en 9 régions de 9 cases (une région étant alors des carrés de 3 × 3), en partie remplie avec des chiffres, de façon que dans chaque ligne, chaque colonne et chaque région les chiffres de 1 à 9 apparaissent une et une seule fois

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

Figure 1: Grille de Sudoku non complétée

Dans le jeu « Smart Sudoku » une option est ajoutée qui consiste à, quand l'utilisateur choisi une case, le jeu affiche les chiffres possibles à rentrer, c'est-à-dire les chiffres non figurés dans la ligne, la colonne et la région associées à la case choisie.

Le travail demandé est de programmer le jeu « Smart Sudoku ». Des options peuvent être ajoutées comme (i) l'effacement d'un chiffre rentré et (ii) avoir plusieurs modèles (grilles) initiales.

Pourquoi ce choix :

J'ai choisi le sujet du « Smart Sudoku » tout d'abord car c'est le jeu que je connais le plus. Le projet m'a aussi semblé plus intéressant que le jeu de dames comportant seulement une dizaine de solutions pré-définies. Le sudoku peut être joué sur un immense nombre de grilles.

Les technologies utilisées :

Ce projet à été réalisé en C et développé sur un ordinateur utilisant un noyau Linux. Le programme est néanmoins cross-platform et fonctionne sans issues sous Windows.

Pour la librairie graphique, j'ai choisi d'utiliser GTK pour sa modernité, son accessibilité et sa compatibilité avec les principaux systèmes d'exploitation.
Toutes les autres fonctions sont soit importées de stdio, soit ont été créées spécifiquement pour ce programme.
Pour plus de clarté et pour une meilleure réutilisation de ce programme, les fonctions ont été séparées en trois fichiers. Sudoku.c contenant les fonctions principales du programme, tables.c contenant les différentes tables utilisées pour le jeu et enfin itoa.c qui est une simple fonction visant à convertir un entier en caractère. Chaque fonction a aussi évidemment son header contenant les prototypes et les structure nécessaires.

Les fonctions importantes :

Le code source est commenté en détail dans son intégralité, pour faciliter sa lecture et sa réutilisation. Je vais néanmoins détailler quelques fonctions principales du Smart Sudoku.

La fonction main() n'est pas très intéressante. Elle sert principalement à placer les différents éléments de l'application graphique avant de passer la main à gtk pour la gestion des événements.

Intéressons-nous plutôt à la fonction verifierValeurs() :

```
void verifierValeurs(GtkWidget *widget, gpointer window)
{//On vérifie la grille
     int i,j,retenue=0;
     const gchar *string;
     GtkWidget *dialog;
```

Dessin 1: verifierValeurs (Partie 1)

```
          //On copie les valeurs présentes dans la table dans
une matrice de valeurs
     for(i=0;i<9;i++){
               for(j=0;j<9;j++){
                    string =
gtk_entry_get_text(GTK_ENTRY(tableau[i][j]));
                    valeurs[i][j]=atoi(string);
               }
     }

     for(i=0;i<9;i++){
               retenue += ligne(i);
               retenue += carre(i);
               retenue += colonne(i);
     }
     //On compare la somme des valeurs des retours. Si différent
de 0, alors il y a une erreur.
     if(retenue == 0){
          //Texte de la boîte de dialogue, à droite de l'icone
          dialog =
gtk_message_dialog_new(window,GTK_DIALOG_DESTROY_WITH_PARENT,
GTK_MESSAGE_INFO,GTK_BUTTONS_OK,"Les valeurs introduites sont
possible");

          //Titre de la boîte de dialogue
          gtk_window_set_title(GTK_WINDOW(dialog), "Sudoku
OK!");
     }
     else{
          //Texte de la boîte de dialogue, à droite de l'icone
          dialog =
gtk_message_dialog_new(window,GTK_DIALOG_DESTROY_WITH_PARENT,
GTK_MESSAGE_WARNING,GTK_BUTTONS_OK,"Les valeurs introduites ne
sont pas possible");

          //Titre de la boîte de dialogue
          gtk_window_set_title(GTK_WINDOW(dialog), "Verifier les
valeurs");
     }
     // Afficher le texte dans la boîte de dialogue
     gtk_dialog_run(GTK_DIALOG(dialog));
     //Fermer la boîte de dialogue si on clique sur OK ou X
     gtk_widget_destroy(dialog);
}
```

Dessin 2: verifierValeurs (Partie 2)

Cette fonction sert à vérifier que toutes les valeurs entrées actuellement dans la table sont possibles. C'est-à-dire qu'il y ait qu'une seule fois chaque chiffre dans chaque ligne, colonne ou carré.

Pour ce faire, on récupère tout d'abord toutes les valeurs de la table en parcourant chacune des cases et en demandant à GTK leur valeur avant de les convertir en entier pour les rentrer dans un tableau appelé « valeurs » (voir les deux lignes de code ci-dessous).

```
string =
gtk_entry_get_text(GTK_ENTRY(tableau[i][j]));
valeurs[i][j]=atoi(string);
```

Dessin 3: verifierValeurs (Extrait)

Une fois ceci effectué, nous appelons 3 sous-programmes qui vérifieront respectivement le respect des règles pour les lignes, les colonnes, et les carrés.
Étudions le sous-programme ligne() :

```
int ligne(int J){ //Vérifier une ligne
     int marqueur[9]={0,0,0,0,0,0,0,0,0},i,d;

     for(i=0;i<9;i++){ //Parcourt un vecteur matriciel à la
recherche de nombres répétés
          d = valeurs[i][J];
          if(marqueur [d-1]==1){
               return 1;
          }
          else
               marqueur[d-1]=1;
     }
     return 0;
}
```

Dessin 4: ligne

Ce sous-programme simple, va, pour une ligne donnée, vérifier si le chiffre d'une des cases est déjà apparue avant, ou est nouveau, dans quel cas il ajoutera un marqueur pour s'en rappeler avant de passer à la case suivante.

Ligne() est très similaire aux sous-programmes colonne() et carré(). Une fois ces trois exécutés, verifierValeurs() va ensuite dessiner une nouvelle boîte de dialogue à l'aide de GTK pour nous informer si des erreurs sont présentes, ou le cas échéant, pour nous dire que la grille est correcte.

Regardons maintenant comment sont stockées les différentes tables de sudoku :

Elles sont rangées dans le fichier tables.c. Séparées du reste du code, il est ainsi plus facile d'en ajouter même si cela reste fastidieux.

Voici en exemple la troisième table proposée dans le jeu :

```
void table3(){
     //3ème table de sudoku ecrite en dur.
     gtk_entry_set_text( GTK_ENTRY(tableau[0][0]), ITOA(1) );
     gtk_entry_set_text( GTK_ENTRY(tableau[8][0]), ITOA(3) );
     gtk_entry_set_text( GTK_ENTRY(tableau[2][1]), ITOA(7) );
     gtk_entry_set_text( GTK_ENTRY(tableau[3][1]), ITOA(2) );
     gtk_entry_set_text( GTK_ENTRY(tableau[4][1]), ITOA(6) );
     gtk_entry_set_text( GTK_ENTRY(tableau[6][1]), ITOA(4) );
     gtk_entry_set_text( GTK_ENTRY(tableau[7][1]), ITOA(8) );
     //Le reste de la fonction est similaire...
}
```

Dessin 5: table3

Le programme est très simple, dans chaque ligne, nous éditons une boîte de texte de la grille de sudoku et nous y affectons une valeur qui va être transformé en caractère par la fonction ITOA().

Terminons cette étude des fonctions avec ecrireVal(). Cette fonction va récupérer une position dans la grille et une valeur à y écrire.

```
void ecrireVal(GtkWidget *widget, gpointer user_data){ //Ecrit la
valeur choisie dans sa case
     //On traite le pointeur sur la structure pos
     struct Pos *pos = user_data;

     //On récupère le parent du widget qui a appellé la fonction
     GtkWidget *parent = gtk_widget_get_toplevel(widget);

     //On récupère la valeur à ecrire
     gint valeur =
GPOINTER_TO_INT(g_object_get_data(G_OBJECT(widget), "valeur"));

     //On ecrit la valeur
     gtk_entry_set_text( GTK_ENTRY(tableau[pos->i][pos->j]),
ITOA(valeur) );

     //On détruit le parent
     gtk_widget_destroy(parent);
}
```

Dessin 6: ecrireVal

Un pointeur user_data est passé à la fonction, on le traite de façon à ce que le programme comprenne que c'est un pointeur vers une structure. La valeur à écrire dans la case définie dans la structure est elle stockée dans le bouton qui à servi à appeler cette fonction. On la récupère et on écrit la valeur dans la case choisie en utilisant la même méthode que pour l'affectation dans le sous-programme table3().

Déroulement du programme :

Essayons maintenant de démarrer le programme. Tout d'abord, nous sommes accueillis par une boîte de texte nous expliquant rapidement les différentes actions possibles.

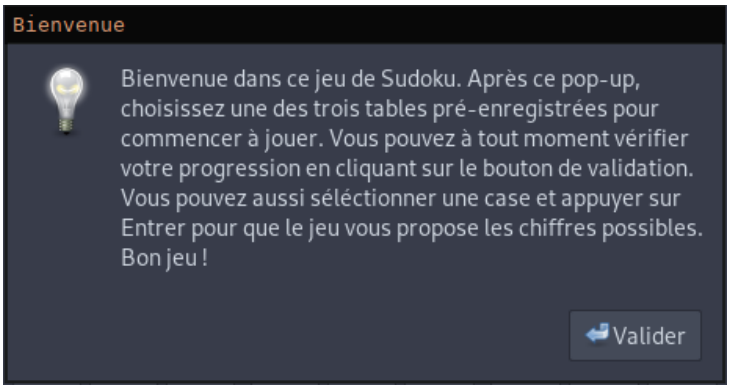

Figure 2: Fenêtre d'accueil

En cliquant sur « Valider », le pop-up disparaît et nous nous retrouvons face à la fenêtre principale du jeu.

Cette fenêtre est composée en premier lieu :

- En haut à gauche de trois boutons labellé « 1 » « 2 » et « 3 » servant à changer de table de Sudoku.
- A leur droite, un bouton servant à vérifier la cohérence de la tables.c
- Et encore plus à droite, un bouton servant à quitter.

En dessous de ces boutons ce trouve la grille de Sudoku. Chaque case est une boîte de texte. Pour entrer des valeurs, il suffit donc de la sélectionner et de rentrer une valeur grâce au clavier. Pour accéder à la partie « Smart Sudoku », il suffit d'appuyer sur Entrée en sélectionnant une case.

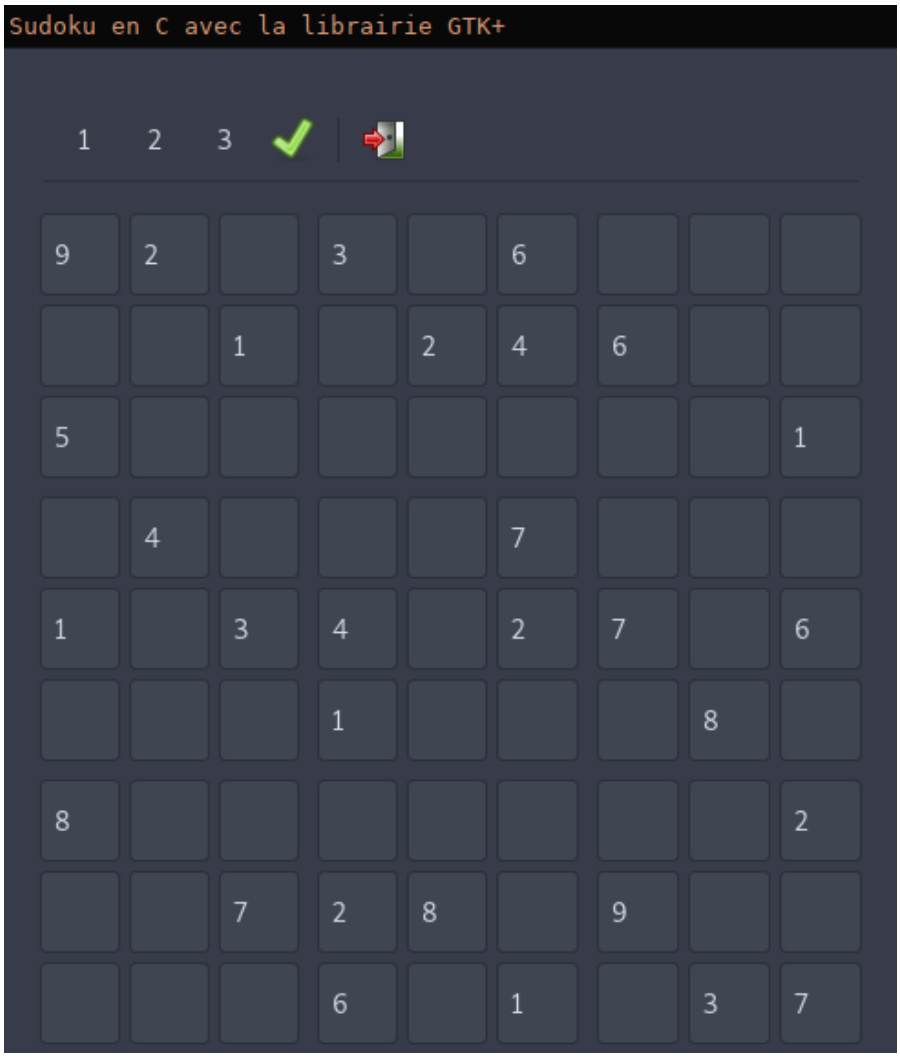

Figure 3: Fenêtre principale

Cela va faire apparaître une nouvelle fenêtre proposant les différents chiffres possible pour cette case.

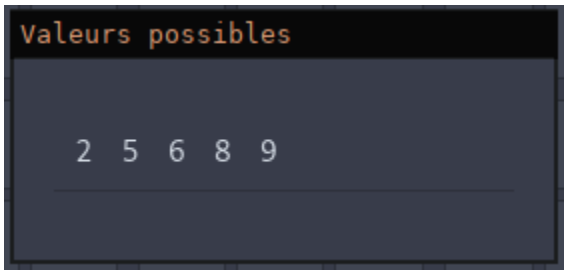

Figure 4: Valeurs possibles

Nous pouvons cliquer sur une des valeurs, et elle sera automatiquement ajoutée à la grille. Si une erreur a été commise, il est très simple de supprimer des valeurs grâce au clavier.

Nous pouvons ensuite vérifier notre travail grâce au bouton de vérification, qui affichera un pop-up différent si la grille est cohérente ou non.

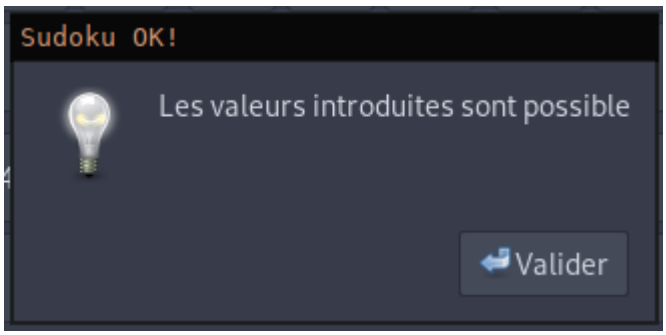

Figure 6: Les valeurs sont correctes

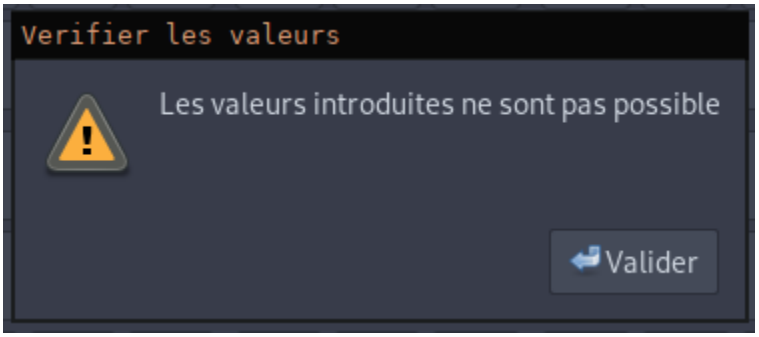

Figure 5: Les valeurs sont incorrectes

Nous pouvons ensuite quitter le programme grâce au bouton prévu à cet effet lorsque nous en avons assez.

Conclusion :

Ce jeu de Sudoku fut une intéressante façon de découvrir la bibliothèque GTK pour programmer des applications graphiques. Les objectifs demandés ainsi que les bonus ont étés implémentées. Il reste tout de même possible d'améliorer ce jeu. On pourrait imaginer un sous-programme qui génère aléatoirement des grilles de Sudoku, ce qui serait plutôt facile à mettre en place de manière itérative, toutes les fonctions nécessaires sont déjà implémentées. On pourrait aussi ajouter des variantes du sudoku comme le Sudoku Samouraï mais là le travail serait nettement plus important.

Annexes :

Le code source, le script de compilation et le programme pré compilé sont tous disponibles dans le fichier compressé qui contenait ce rapport.