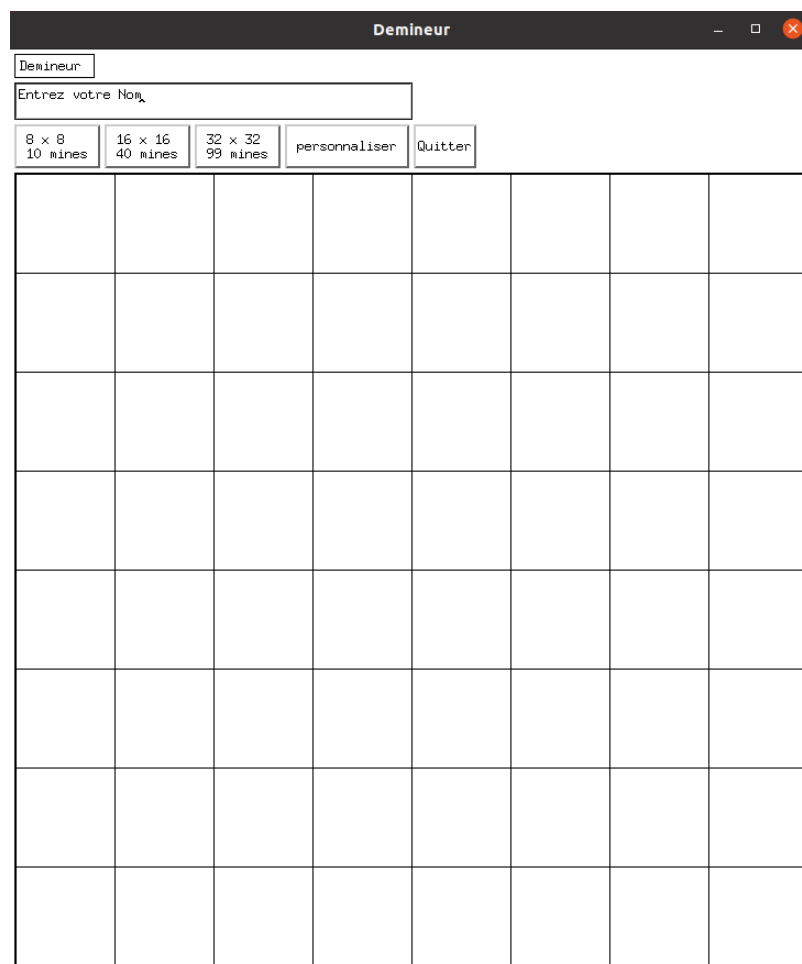


# Projet du démineur

*Matière : Langage C*



The screenshot shows a graphical user interface for a Minesweeper game. The window has a title bar labeled 'Démineur'. Below the title bar, there is a text input field labeled 'Entrez votre Nom'. Underneath the input field, there are five buttons: '8 x 8 10 mines', '16 x 16 40 mines', '32 x 32 99 mines', 'personnaliser', and 'Quitter'. The main area of the window is a large grid of 8 columns and 8 rows, representing the game board. The grid is currently empty, with all cells being white.

Pierre Harold SIGNING - Vincent DE SOUSA

## Table des matières

<b>Introduction</b> .....	2
<b>Partie 1 : Présentation générale du projet</b> .....	2
1-) Installation et utilisation de notre jeu.....	2
2-) Les différents fichiers présents dans notre projet .....	3
<b>Partie 2 : Création du projet</b> .....	3
<b>1-) Première étape : Initialisation de notre champ de mines</b> .....	3
A) <i>Choix de l'initialisation</i> .....	3
B) <i>Initialisation de la grille à partir de pointeur sur caractère</i> .....	4
C) <i>Définition des Widgets</i> .....	4
D) <i>Description des procédures appelées suivant la difficulté</i> .....	5
<b>2-) Deuxième étape : Affichage du champ de mines</b> .....	10
A) <i>Fonctionnement de la grille</i> .....	10
B) <i>Quitter le jeu</i> .....	13
<b>Conclusion</b> .....	13

## Table des figures

Figure 1 - Fonctionnement de l'architecture libsx.....	4
Figure 2 – Fonction init_display .....	5
Figure 3 – Suite fonction init_display .....	5
Figure 4 – Menu du jeu.....	5
Figure 5 - Fonction MakeButton.....	6
Figure 6 – Fonction de rappel GridGame_easyCD .....	6
Figure 7 - Fonction ChoiceDim .....	6
Figure 8 - Définition de la structure dim .....	6
Figure 9 - Fonction NbCase.....	7
Figure 10 - Définition de la structure GameState .....	7
Figure 11 - Fonction InitSetting .....	7
Figure 12- Fonction grilleVide.....	8
Figure 13 - Fonction grilleDeMine .....	8
Figure 14 - Fonction *mineProche.....	8
Figure 15 - Fonction DrawGrille.....	9
Figure 16 – Fonction de rappel GridGame_averageCB .....	9
Figure 17 – Grille de jeu.....	10
Figure 18 – Fonction de rappel button_down .....	11
Figure 19 - Fonction ActionAfterClick .....	11
Figure 20 - Fonction EmptyCase .....	12
Figure 21 - Fonction scoreInFile.....	12
Figure 22 - Fonction GameOver .....	12
Figure 23 - Widget Exit.....	13
Figure 24 - Fonction de rappel QuitGame.....	13

## Introduction

Le Démineur est un jeu vidéo de réflexion dont le but est de localiser des mines cachées dans une grille représentant un champ de mines virtuel, avec pour seule indication le nombre de mines dans les zones adjacentes. Chaque case de la grille peut soit cacher une mine, soit être vide, si la case est vide alors un coefficient de proximité avec les mines aux alentours s'affiche indiquant les dangers du terrain. En comparant les différentes informations récoltées, le joueur peut ainsi progresser dans le déminage du terrain. S'il se trompe et clique sur une mine, il a perdu.

Ce jeu est très connu car il est fourni par défaut avec le système d'exploitation Microsoft Windows. D'autant plus que de nombreuses autres versions du jeu sont disponibles gratuitement sur internet.

Notre projet consiste à réaliser le jeu expliqué précédemment, en utilisant la librairie *libsx* pour l'affichage. Notre jeu doit comporter trois niveaux de difficultés. Il doit également permettre à l'utilisateur de visualiser ses scores précédents grâce à une sauvegarde, ainsi que de lui permettre de choisir le nombre de mines qu'il souhaite mettre dans la grille.

A travers les différentes parties de ce rapport, nous allons dégrossir le projet en partant de l'installation générale du jeu pour terminer sur une explication de plus en plus détaillée du code.

## Partie 1 : Présentation générale du projet

### 1-) Installation et utilisation de notre jeu

Afin d'installer et d'utiliser notre jeu, veuillez suivre les étapes suivantes :

- Télécharger le fichier « demineur-SIGNING-DESOUSA.tgz »
- Décompresser le à un endroit significatif pour vous.
- Aller dans votre terminal et placez-vous dans le fichier

Exemple : `user@PNS-VirtualBox:~$ cd ~/Téléchargements/demineur-SIGNING-DESOUSA/`

- Lancer l'exécutable « Demineur »

Exemple : `user@PNS-VirtualBox:~/Téléchargements/demineur-SIGNING-DESOUSA$ ./Demin`

⚠ En cas d'apparition de l'erreur suivante : ⚠

```
user@PNS-VirtualBox:~/Téléchargements/demineur-SIGNING-DESOUSA$ ./Demin  
bash: ./Demin: Aucun fichier ou dossier de ce type
```

- Créer l'exécutable à l'aide de la commande « make all »

Exemple : `user@PNS-VirtualBox:~/Téléchargements/demineur-SIGNING-DESOUSA$ make all  
gcc -Wall -g -lsx -o Demineur main.o modele.o callbacks.o vue.o`

- Lancer l'exécutable « Demineur ».

Exemple : `user@PNS-VirtualBox:~/Téléchargements/demineur-SIGNING-DESOUSA$ ./Demin`

## 2-) Les différents fichiers présents dans notre projet

Notre projet est composé de 7 fichiers différents mais communiquant entre eux afin de mener à bien le fonctionnement du jeu.

On retrouve : 1 fichier **main.c**, 2 fichiers « **modele** », 2 fichiers « **callbacks** » et 2 fichiers « **vue** »; avec pour chaque couple un **fichier .c** qui correspond à nos fonctions et à nos procédures ainsi qu'un **fichier .h** qui permet d'exporter ses fonctions et procédures dans d'autres programmes.

- **Les fichiers modele.c et modele.h**

Le fichier **modele.c** permet la création du contenu de la grille ainsi que de ses attribues via une structure « dim » qui informe sur les dimensions de la grille et son nombre de mine.

Le fichier **modele.h** contient l'ensemble des définitions des constantes, des structures et des fonctions du modele.c que l'on souhaite utiliser ailleurs dans les programmes.

- **Les fichiers callbacks.c et callbacks.h**

Le fichier **callbacks.c** contient un ensemble de fonctions de rappel des événements traiter par vue.c.

Le fichier **callbacks.h** désigne les fonctions du callback.c qu'on souhaite utiliser ailleurs dans le programme.

- **Les fichiers vue.c et vue.h**

Le fichier **vue.c** gère l'affichage du jeu grâce aux fonctions de libsx.

Le fichier **vue.h** contient l'ensemble des définitions des constantes liées à l'affichage et des fonctions de vue.c que l'on souhaite utiliser ailleurs dans les programmes.

- **Le fichier main.c**

Le fichier **main.c** comporte une seule fonction qui est la fonction principale permettant de lancer le jeu.

## Partie 2 : Création du projet

### 1-) Première étape : Initialisation de notre champ de mines

#### *A) Choix de l'initialisation*

Notre champ de mines est représenté par une grille pouvant avoir 3 configurations différentes soit 8x8, 16x16 ou 32x32 en fonction du choix de l'utilisateur.

La première phase de notre projet consistait donc à discuter de nos différentes idées concernant le moyen que nous allions utiliser afin d'initialiser cette grille.

Deux choix s'offrait à nous :

- initialiser à partir d'une matrice de caractère (plus simple à implémenter)
- Initialiser à partir de pointeur sur caractère (plus difficile à implémenter)

Malgré le fait que le choix numéro 1 semble plus judicieux de par sa mise en place plus simple et sa compréhension plus logique car la matrice représente un tableau à deux dimensions ce qui correspond parfaitement à une grille, nous avons décidé d'initialiser cette grille à partir de pointeur sur caractère.

En effet, malgré le fait que ce choix soit plus difficile à implémenter car le pointeur sur caractère représente un tableau unidimensionnel nous avons choisis de l'utiliser afin d'améliorer nos connaissances pratiques sur ce type de données.

### *B) Initialisation de la grille à partir de pointeur sur caractère*

Dans notre programme, on déclare notre pointeur sur caractère dans le fichier main.c qui prendra les valeurs de chaque élément de notre grille. Ce pointeur sur caractère sera passé dans la fonction `init_display(argc, argv[v], *grid)` qui est une fonction d'affichage créer dans le fichier `vue.c`, ce fichier fera appel aux fonctions créer dans `callbacks.c` et `callbacks.c` fera appel aux fonctions de `modele.c` ainsi la transmission et la modification des valeurs de notre grille sera possible à travers tous les programmes.

Cette transmission est possible grâce à l'inclusion des fichiers d'en tête (`vue.h`, `callbacks.h`, `modele.h`) dans les différents programmes.

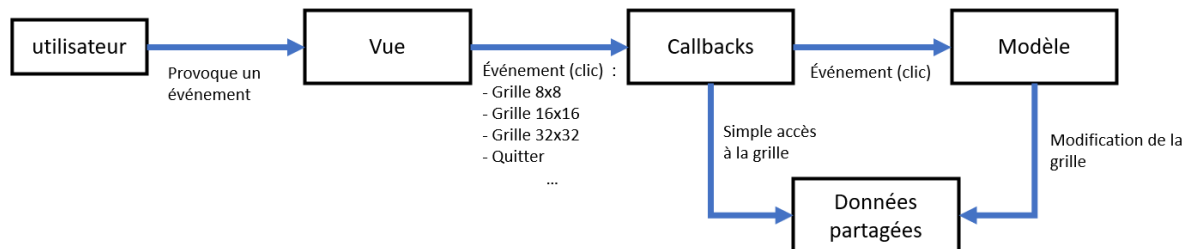


Figure 1 - Fonctionnement de l'architecture libsx

Par la suite, il sera conseillé de se référer à ce schéma pour comprendre ce qui est énoncé car par le biais des différents événements que l'utilisateur peut provoquer nous allons expliquer le fonctionnement de nos programmes.

### *C) Définition des Widgets*

Les Widgets sont définis dans la fonction `init_display` qui initialise tous les éléments liés à l'affichage de notre jeu. Cette fonction initialise 8 Widgets à l'aide des fonctions présentes dans la librairie `libsx` :

- **Title** : représente le titre de notre jeu il est réalisé avec la fonction `MakeLabel`.
- **EnterName** : représente une zone de texte à compléter du nom de l'utilisateur, cette zone de texte est réalisée avec la fonctionnalité `MakeStringEntry`.
- **BEasyMode, BAverageMode, BHighMode, BCustomMode** : représentent les boutons du menu à travers lesquels on peut choisir la difficulté du jeu ils sont réalisés à l'aide de la fonction `MakeButton` de `libsx`.
- **Exit** : représente un bouton qui permet à l'utilisateur de quitter le jeu.
- **Game** : représente un espace dans le lequel on peut dessiner des formes ou des textes, cet espace est conçu grâce à la fonction `MakeDrawArea`.

```

void init_display (int argc, char *argv[], char *grid){
    /*Définition des widgets */
    Widget BEasyMode,BAverageMode,BHighMode,BCustomMode ;
    Widget Title,EnterName, Exit;
    char *player ;

    //Entête de fenêtre
    Title = MakeLabel("Demineur ") ;

    //Zone de texte pour nom de joueur et validation
    EnterName = MakeStringEntry ("Entrez votre Nom" ,SCREENVIEW ,NULL ,NULL) ;

    //Mode de Jeu
    BEasyMode = MakeButton (" 8 x 8 \n 10 mines " , GridGame_easyCB , grid) ;
    BAverageMode = MakeButton (" 16 x 16 \n 40 mines " , GridGame_averageCB , grid) ;
    BHighMode = MakeButton (" 32 x 32 \n 99 mines " , GridGame_highCB , grid) ;
    BCustomMode = MakeButton (" personnaliser ",GridGame_editCB,grid) ;

    //Fin de jeu
    Exit = MakeButton("Quitter", QuitGame, NULL);
    Game = MakeDrawArea(GRID_X,GRID_Y,NULL,NULL);
}

```

Figure 2 – Fonction init\_display

Ensuite, on définit la taille et la position de l'ensemble des Widgets sur la fenêtre avec les fonctions SetWidgetPos et SetWidgetSize.

```

/*Positionnement des widgets */

SetWidgetSize(Title, TBOX_X, TBOX_Y) ;
SetBorderColor(Title, RED) ;

SetWidgetPos(EnterName, PLACE_UNDER, Title, NO_CARE, NULL) ;
SetWidgetSize(EnterName, SCREENVIEW, 1.6*TBOX_Y) ;

SetWidgetPos(BEasyMode, PLACE_UNDER, EnterName, NO_CARE, NULL) ;
SetWidgetSize(BEasyMode, TBOX_X, 2*TBOX_Y) ;

SetWidgetPos(BAverageMode,PLACE_UNDER,EnterName ,PLACE_RIGHT,BAEasyMode) ;
AttachEdge(BAverageMode,LEFT_EDGE,ATTACH_TOP) ;
SetWidgetSize(BAverageMode, TBOX_X, 2*TBOX_Y) ;

SetWidgetPos(BHighMode, PLACE_UNDER, EnterName, PLACE_RIGHT, BAverageMode) ;
SetWidgetSize(BHighMode, TBOX_X, 2*TBOX_Y) ;

SetWidgetPos(BCustomMode, PLACE_UNDER, EnterName, PLACE_RIGHT, BHighMode) ;
AttachEdge(BCustomMode,LEFT_EDGE,ATTACH_LEFT) ;
SetWidgetSize(BCustomMode, TBOX_X, 2*TBOX_Y) ;

SetWidgetPos(Exit, PLACE_UNDER, EnterName, PLACE_RIGHT, BCustomMode) ;
SetWidgetSize(Exit, TBOX_X, 2*TBOX_Y) ;

Game = MakeDrawArea(GRID_X,GRID_Y,NULL,NULL);
SetWidgetPos(Game, PLACE_UNDER, BCustomMode, NO_CARE, NULL) ;

GetStandardColors() ;
ShowDisplay(); // pour afficher l'interface et son contenu
SetButtonDownCB(Game,button_down) ;
}

```

Figure 3 – Suite fonction init\_display

#### D) Description des procédures appelées suivant la difficulté

Au début du jeu, plusieurs choix de jeu s'offrent à vous :

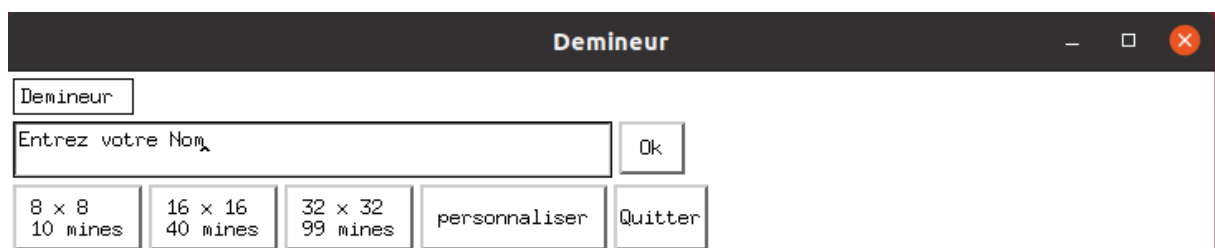


Figure 4 – Menu du jeu

Vous pouvez cliquer sur différentes tailles de grille (8x8, 16x16, 32x32).

Admettons que vous cliquez sur le bouton facile soit la grille 8x8, ce bouton est en réalité une fonction qui renvoie un type widget :

**Widget MakeButton(char \*label, ButtonCB func, void \*data);**

Lorsqu'on l'appelle, cette fonction prend plusieurs paramètres :

- **Char \*label** : correspond au nom que l'on souhaite donner au bouton
- **ButtonCB func** : est la fonction de rappel (callback) qui va être appelée lorsqu'on appuie dessus.
- **Void \*data** : correspond à la donnée partagée ici ce sera notre grille.

C'est ainsi nous avons appelé la fonction **MakeButton** qui crée un bouton permettant de choisir la difficulté, tout en prenant soin de définir la variable **BEasyMode** en tant que Widget, par conséquent l'appuie sur le bouton appelle la fonction de rappel **GridGame\_easyCB** situé dans le fichier callbacks.c.

```
//Mode de Jeu
BEasyMode = MakeButton (" 8 x 8 \n 10 mines " , GridGame_easyCB , grid) ;
```

Figure 5 - Fonction MakeButton

La fonction de rappel **GridGame\_easyCD** comprend les fonctionnalités suivantes qui seront expliquées par la suite.

```
void GridGame_easyCB(Widget * w, void *grid ){
    d = ChoiceDim(1) ;
    grid = malloc(NbCase(d)*sizeof(char)) ;
    MarkedGrid = malloc(NbCase(d)*sizeof(char)) ;
    FlagGrid = malloc(NbCase(d)*sizeof(char)) ;
    InitSetting(&G, d) ;
    grilleVide(grid, d) ;
    grilleVide(MarkedGrid, d) ;
    grilleVide(FlagGrid, d) ;
    grilleDeMine(grid, d.mine_Number) ;
    g=mineProche(grid, d) ;
    DrawGrille(grid, d) ;
}
```

Figure 6 – Fonction de rappel GridGame\_easyCD

#### • La Fonction ChoiceDim

La fonction **ChoiceDim** contenu dans modele.c permet d'initialiser la variable dim d qui à partir du choix de difficulté donné permet de définir la longueur, la largeur et le nombre de mines dans la grille.

```
dim ChoiceDim(int choice){
    dim d ;
    if(choice ==1){
        d.width = M1 ;
        d.height = M1 ;
        d.mine_Number = NB_MINE1 ;
    }
    if(choice ==2){
        d.width = M2 ;
        d.height = M2 ;
        d.mine_Number = NB_MINE2 ;
    }
    if(choice ==3){
        d.width = MX3 ;
        d.height = MY3 ;
        d.mine_Number = NB_MINE3 ;
    }
    return d ;
}
```

Figure 7 - Fonction ChoiceDim

```
//Structure définissant la dimension d'une grille
typedef struct {
    unsigned int width ; // dim_y
    unsigned int height ; // dim_x
    unsigned int mine_Number ; // mine_number = nb_m
}dim ;
```

Figure 8 - Définition de la structure dim

- **Fonction NbCase**

A présent, connaissant les dimensions de la grille nous pouvons allouer l'espace dynamique à nos pointeurs sur caractère grâce à la fonction **NbCase** du modele.c qui renvoie un entier correspondant à la multiplication de la longueur par la largeur de la grille :

```
// Antécédent : une variable de type dimension
// Rôle : Renvoie le nombre de cellule dans la grille
unsigned int NbCase (const dim d){
    return (d.height) * (d.width) ;
}
```

Figure 9 - Fonction NbCase

**grid** : contiendra les valeurs que peuvent prendre les cases de la grille

**MarkedGrid** : permettra de connaître pour chacun des éléments de la grille s'il a été affiché ou non.

**FlagGrid** : permettra de savoir pour chaque case de la grille si un drapeau a été posé dessus.

N.B : Un drapeau correspond à une case que l'on suspecte être une mine pour éviter de cliquer dessus malencontreusement.

- **Fonction InitSetting**

La fonction **InitSetting** du modele.c initialise les éléments de la variable GameState \*G :

**EndGame** : Indique si la partie est finie (1) ou non (0).

**OpenException** : Condition d'ouverture d'une case dans laquelle une mine est présente.

**FixFlag** : Indique l'autorisation de poser un drapeau (0) ou non (1).

**FlagNumber** : Indique le nombre de drapeau que l'utilisateur possède, il est naturellement égal au nombre de mine présentes dans la grille.

**Score** : Indique le score de l'utilisateur, initialisé à 0.

```
void InitSetting (GameState *G, const dim d){
    G -> EndGame = 0 ;
    G -> OpenException = 0 ;
    G -> FixFlag = 0 ;
    G -> GameResume = 1 ;
    G -> FlagNumber = d.mine_Number ;
    G -> Score = 0 ;
}
```

Figure 11 - Fonction InitSetting

```
typedef struct {
    unsigned int GameResume ; // signalant si une partie est en cours ou pas
    unsigned int EndGame ; // Indiquant si l'on est à la fin du jeu
    unsigned int OpenException ; // exception d'ouverture pour l'effet générique de la cellule zéro
    unsigned int FixFlag ; // L'autorisation de poser un drapeau
    unsigned int FlagNumber ; // le nombre de drapeau que je possède
    unsigned int Score ; // Le score du joueur
} GameState ;
```

Figure 10 - Définition de la structure GameState

- **Fonction grilleVide**

Par la suite, on initialise nos pointeurs sur caractère précédemment alloué d'un espace mémoire à '0' grâce à la fonction **grilleVide** du modele.c. Cette fonction permet de d'initialiser toutes les valeurs pointées à '0'.



```
//Antécédent : gridrille allouer dynamiquement et dimensions définies par l'utilisateur
//Rôle : initialise une gridrille vide (pleine de 0) à l'aide d'un pointeur sur caractère
void grilleVide(char *grid, const dim d){

    for(int i=0; i<=NbCase(d); i++){
        *(grid+i)='0';
    }
}
```

Figure 12- Fonction grilleVide

- **Fonction grilleDeMine**

Puis la fonction **grilleDeMine** du fichier modele.c permet d'initialiser les mines dans la grille à des valeurs pointées aléatoires.

```
//Antécédent : dimensions définies et gridrille vide initialisée
//Rôle : initialise les emplacements aléatoire des bombes sur la gridrille
void grilleDeMine(char *grid, const int nb_m){
    int k=0, rang;

    while(k!=nb_m+1){
        rang=rand()%(strlen(grid)-1);
        printf("rang: %d\n",rang);
        *(grid+rang)='M';
        k++;
    }
}
```

Figure 13 - Fonction grilleDeMine

- **Fonction mineProche**

Enfin on appelle la fonction **mineProche** qui initialise à chaque valeur pointée de la grille les coefficients de proximité des mines en prenant en compte tous les cas envisageables.

```
//Antécédent : dimensions définies et grille de bombe initialisée
//Rôle : initialise autour des emplacements des bombes les coefficients de proximité
char *mineProche(char *grid, const dim d){

    unsigned int taille = NbCase(d);
    for(int i=0; i<taille; i++){
        if(*(grid+i)!='M'){
            //Partie gridauche de la gridrille
            if((i%d.width)==0){

                //Case en haut à gridauche de la gridrille
                if(i==0){
                    //((condition) ? instruction si vrai : instruction si faux
                    if (*(grid+i-1)!='M') (*(grid+i-1))++;
                    if (*(grid+i-d.width)!='M') (*(grid+i-d.width))++;
                    if (*(grid+i+d.width)!='M') (*(grid+i+d.width))++;
                    if (*(grid+i+d.width+1)!='M') (*(grid+i+d.width+1))++;
                }

                //Case en bas à gridauche de la gridrille
                else if(i==taille-d.width){
                    if (*(grid+i-d.width)!='M') (*(grid+i-d.width))++;
                    if (*(grid+i-d.width+1)!='M') (*(grid+i-d.width+1))++;
                    if (*(grid+i+1)!='M') (*(grid+i+1))++;
                }

                //Le reste des cases à gridauche de la gridrille
                else{
                    if (*(grid+i-d.width)!='M') (*(grid+i-d.width))++;
                    if (*(grid+i-d.width+1)!='M') (*(grid+i-d.width+1))++;
                    if (*(grid+i-1)!='M') (*(grid+i-1))++;
                    if (*(grid+i+1)!='M') (*(grid+i+1))++;
                    if (*(grid+i+d.width)!='M') (*(grid+i+d.width))++;
                    if (*(grid+i+d.width+1)!='M') (*(grid+i+d.width+1))++;
                }
            }

            //Partie droite de la gridrille
            else if(((i%d.width)==d.width-1)){

                //Case en haut à droite de la gridrille
                if(i==d.width-1){
                    if (*(grid+i-1)!='M') (*(grid+i-1))++;
                    if (*(grid+i-d.width)!='M') (*(grid+i-d.width))++;
                    if (*(grid+i+d.width)!='M') (*(grid+i+d.width))++;
                }

                //Case en bas à droite de la gridrille
                else if(i==NbCase(d)-1){
                    if (*(grid+i-d.width)!='M') (*(grid+i-d.width))++;
                    if (*(grid+i-d.width+1)!='M') (*(grid+i-d.width+1))++;
                    if (*(grid+i-1)!='M') (*(grid+i-1))++;
                }
            }
        }
    }
}
```

```
//Le reste des cases à droite de la gridrille
else{
    if (*(grid+i-d.width-1)!='M') (*(grid+i-d.width-1))++;
    if (*(grid+i-d.width)!='M') (*(grid+i-d.width))++;
    if (*(grid+i-1)!='M') (*(grid+i-1))++;
    if (*(grid+i+d.width-1)!='M') (*(grid+i+d.width-1))++;
    if (*(grid+i+d.width)!='M') (*(grid+i+d.width))++;
}

//Partie supérieur de la gridrille sans les coins
else if(i>=1 && i<=d.width){
    if (*(grid+i-1)!='M') (*(grid+i-1))++;
    if (*(grid+i+1)!='M') (*(grid+i+1))++;
    if (*(grid+i-d.width-1)!='M') (*(grid+i-d.width-1))++;
    if (*(grid+i+d.width-1)!='M') (*(grid+i+d.width-1))++;
    if (*(grid+i+d.width)!='M') (*(grid+i+d.width))++;
    if (*(grid+i+d.width+1)!='M') (*(grid+i+d.width+1))++;
}

//Partie inférieure de la gridrille sans les coins
else if(i>=NbCase(d)-d.width-1 && i<=NbCase(d)-2){
    if (*(grid+i-d.width-1)!='M') (*(grid+i-d.width-1))++;
    if (*(grid+i-d.width)!='M') (*(grid+i-d.width))++;
    if (*(grid+i-d.width+1)!='M') (*(grid+i-d.width+1))++;
    if (*(grid+i-1)!='M') (*(grid+i-1))++;
    if (*(grid+i+1)!='M') (*(grid+i+1))++;
}

//Partie centrale de la gridrille
else{
    if (*(grid+i-d.width-1)!='M') (*(grid+i-d.width-1))++;
    if (*(grid+i-d.width)!='M') (*(grid+i-d.width))++;
    if (*(grid+i-d.width+1)!='M') (*(grid+i-d.width+1))++;
    if (*(grid+i-1)!='M') (*(grid+i-1))++;
    if (*(grid+i+1)!='M') (*(grid+i+1))++;
    if (*(grid+i+d.width-1)!='M') (*(grid+i+d.width-1))++;
    if (*(grid+i+d.width)!='M') (*(grid+i+d.width))++;
    if (*(grid+i+d.width+1)!='M') (*(grid+i+d.width+1))++;
}

return grid;
}
```

Figure 14 - Fonction \*mineProche

- **Fonction DrawGrille**

Finalement on appelle la fonction **DrawGrille** du fichier vue.c permettant de dessiner la grille à partir des dimensions passées en paramètres à l'aide de la fonction DrawBox de libsx. Pour utiliser ces fonctionnalités il a fallu créer un espace où l'on peut dessiner avec la fonction MakeDrawArea.

```
void DrawGrille(char *grid, dim d){
    unsigned int Box = CaseSize (d) ;
    ClearDrawArea();
    for(int i=0 ; i<d.width ; i++){
        for(int j=0 ; j<d.height ; j++){
            DrawBox(i*Box, j*Box,Box,Box) ;
        }
    }
}
```

Figure 15 - Fonction DrawGrille

En ce qui concerne les autres modes de difficulté possibles, le schéma de fonctionnement qui vient d'être expliqué se réitère, les seuls éléments qui vont changer seront les données de dimension affectée à la variable dim d qui permettront de modifier toutes les fonctions utilisées par la suite.

```
void GridGame_averageCB(Widget * w, void *grid ){
    d = ChoiceDim(2) ;
    grid = malloc(NbCase(d)*sizeof(char)) ;
    flag = malloc(NbCase(d)*sizeof(char)) ;
    except = malloc(sizeof(int)) ;
    *except = 0 ;
    grilleVide(grid, d) ;
    grilleVide(flag ,d) ;
    grilleDeMine(grid, d.mine_Number) ;
    g=mineProche(grid, d) ;
    DrawGrille(grid, d) ;
}

void GridGame_highCB(Widget * w, void *grid ){
    d = ChoiceDim(3) ;
    grid = malloc(NbCase(d)*sizeof(char)) ;
    flag = malloc(NbCase(d)*sizeof(char)) ;
    except = malloc(sizeof(int)) ;
    *except = 0 ;
    grilleVide(grid, d) ;
    grilleVide(flag ,d) ;
    grilleDeMine(grid, d.mine_Number) ;
    g=mineProche(grid, d) ;
    DrawGrille(grid, d) ;
}
```

Figure 16 – Fonction de rappel GridGame\_averageCB

Pour récapituler, la fonction **GridGame\_easyCD** permet d'initialiser les valeurs de chaque case de la grille et de dessiner la grille dans l'espace DrawArea à travers les fonctions expliquées précédemment.

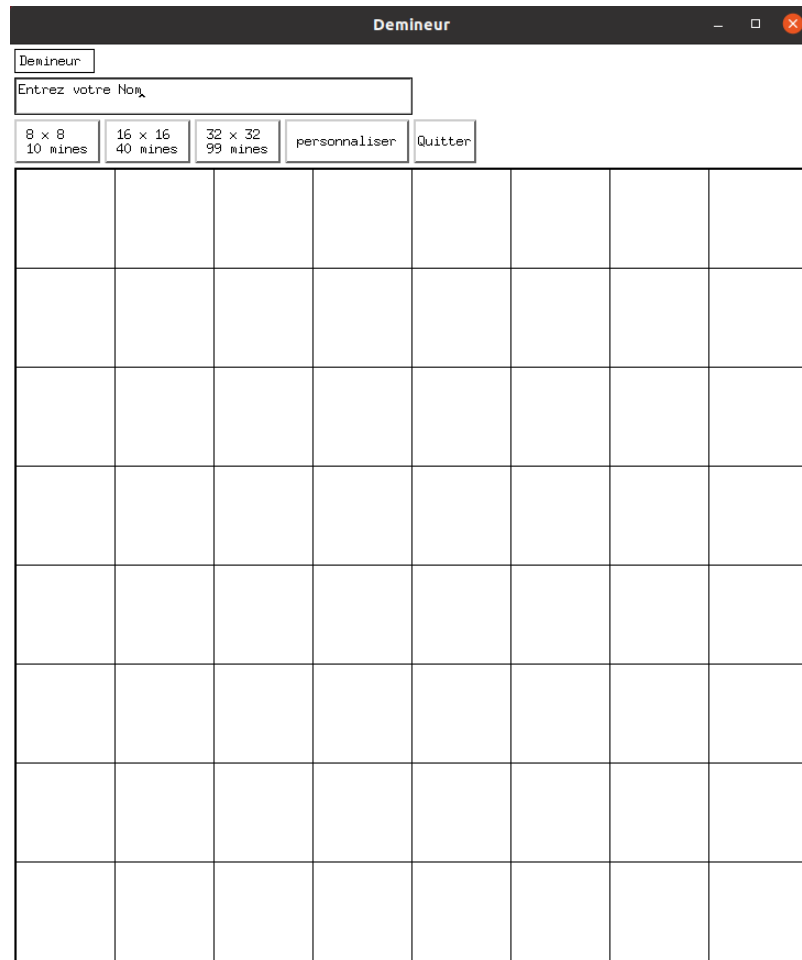


Figure 17 – Grille de jeu

## 2-) Deuxième étape : Affichage du champ de mines

### A) Fonctionnement de la grille

Après avoir initialisé la grille, nous allons expliquer son fonctionnement, c'est à dire les actions qui vont se dérouler lorsqu'on clique sur une case.

Lorsqu'on clique sur une case on souhaite afficher la valeur présente à un rang pointé de notre pointeur sur caractère. Pour ce faire, on utilise la fonction suivante de la librairie libsx:

```
void SetButtonDownCB(Widget w, MouseButtonCB func);
```

Cette fonction prend en paramètre le Widget sur lequel on va cliquer, ici notre Widget affecté à la fonction **MakeDrawArea** et la fonction de rappel.

Enseignant : M. GRANET

Date : 12/06/2022

A présent, lorsque je clique sur la grille je vais appeler la fonction de rappel **button\_down** situé dans le fichier **callbacks.c**. Cette fonction prend comme paramètres : le bouton de la souris sur lequel on a cliqué et les coordonnées x et y du clique. Ce qui va nous permettre de traiter l'action.

Nous souhaitons révéler la valeur de la case seulement si l'utilisateur à cliquer avec le clique gauche de sa souris d'où l'énoncé conditionnel qui suit.

```
void button_down(Widget *w, int which_button, int x, int y, void *data){
    printf("You clicked button %d at (%d,%d)\n", which_button, x,y);
    if(which_button == 1){
        ActionAfterClick(x,y,d,g,flag,except);
    }
}
```

Figure 18 – Fonction de rappel button\_down

Si l'utilisateur à cliquer avec le clique gauche alors la fonction **ActionAfterClick** du fichier **vue.c** est appelée. Cette fonction permet de déterminer à partir des coordonnées du clique la valeur à cet endroit de la grille.

```
void ActionAfterClick(int x, int y, dim d, char *grid, char *MarkedGrid, char *FlagGrid, GameState *G){
    //afficheGrille(grid, d) ;
    int rang=0;
    char *value=calloc(1,sizeof(char));
    score=calloc(1,sizeof(int));

    if(verif(ChoiceDim(1),d)){
        for(int i=0 ; i<GRID_Y; i=i+8*TY){
            for(int j=0 ; j<GRID_X; j=j+8*TX){
                if((x>=i)&&(x<i+8*TX)){
                    if((y>=j)&&(y<j+8*TY)){
                        *value = *(grid+rang) ;
                        printf("%c %c \n",*value,*(MarkedGrid+rang) ) ;
                        if(*value == 'M' && (G->OpenException == 0)){
                            SetColor(GetNamedColor("red")) ;
                            scoreInFile(score);
                            *score=0;
                            GameOver(grid,d) ;
                            GetOkay("Game Over !!") ;
                        }
                        if(*value != 'M' && *value != '0' ){
                            SetColor(GetNamedColor("gray")) ;
                            DrawFilledBox(i+2*TX,j+2*TX,4*TX,4*TY) ;
                            DrawText(value, (2*i+8*TX)/2, (2*j+8*TY)/2);
                            *(MarkedGrid+rang) = '1' ;
                            *score++;
                        }
                        if((*value == '0') && (*(MarkedGrid+rang) == '0')){
                            *(MarkedGrid+rang) = '1' ;
                            G->OpenException = 1 ;
                            SetColor(GetNamedColor("gray")) ;
                            DrawFilledBox(i+2*TX,j+2*TX,4*TX,4*TY) ;
                            DrawText("", (2*i+8*TY)/2, (2*j+8*TX)/2);
                            EmptyCase(x,y,d,grid,MarkedGrid,FlagGrid, &G) ;
                            G->OpenException = 0 ;
                            *score++;
                        }
                    }
                }
            }
            rang++ ;
        }
    }
}
```

Figure 19 - Fonction ActionAfterClick

Un des énoncés conditionnels présent dans la fonction **ActionAfterClick** indique que si l'élément de la grille que l'utilisateur souhaite découvrir est un 0, alors on appelle la fonction **EmptyCase**. Cette fonction permet d'afficher sur la grille les cases adjacentes aux cases qui valent 0.

```
void EmptyCase(const int x,const int y,const dim d,char *grid ,char *flag, int *except){
    unsigned int Box = CaseSize (d) ;
    printf("box : %d", Box);

    if(x>=Box) {
        ActionAfterClick(x-Box, y, d,grid,flag, except);
    }
    if(x>=Box && y>=Box){
        ActionAfterClick(x-Box, y-Box, d, grid ,flag, except);
    }
    if(y>=Box) {
        ActionAfterClick(x, y-Box, d, grid, flag, except);
    }
    if(y>=Box && x<=GRID_X-Box) {
        ActionAfterClick(x+Box, y-Box, d, grid, flag, except);
    }
    if(x<=GRID_X -Box) {
        ActionAfterClick(x+Box, y, d, grid, flag, except);
    }
    if(x<=GRID_X -Box && y<GRID_Y - Box) {
        ActionAfterClick(x+Box, y+Box, d, grid, flag, except);
    }
    if(y<GRID_Y-Box) {
        ActionAfterClick(x, y+Box, d, grid, flag, except);
    }
    if(x>=Box && y<GRID_Y-Box) {
        ActionAfterClick(x-Box, y+Box, d, grid, flag, except);
    }
}
```

Figure 20 - Fonction EmptyCase

Au fil du jeu, l'utilisateur découvre les cases qu'il souhaite jusqu'à gagner s'il découvre toutes les cases sauf les mines. Ou perdre s'il découvre une mine. Ainsi, si l'utilisateur clique sur une mine nous entrons dans l'énoncé conditionnel de la fonction **ActionAfterClick** qui indique que la valeur courante est une mine. Par conséquent, la fonction **GameOver** est appelée, celle-ci permet d'afficher la position de toutes les mines sur la grille afin de prouver que l'utilisateur a perdu. De plus, un message de défaite est affiché sur l'écran à l'aide de la fonction **GetOkay** de libsx. De plus, la fonction **scoreInFile** est appelé afin d'enregistrer le score dans un fichier score.txt

```
void GameOver (char *grid, dim d){
    int rang = 0 ;
    unsigned int Box = CaseSize (d) ;
    char *value=calloc(1,sizeof(char));

    for(int i=0 ; i<GRID_Y; i=i+Box){
        for(int j=0 ; j<GRID_X; j=j+Box){
            *value=*(grid+rang);
            if(*value== 'M'){
                DrawText(value, (2*i+Box)/2, (2*j+Box)/2);
            }
            rang++ ;
        }
    }
    free(value);
}
```

Figure 22 - Fonction GameOver

```
void scoreInFile(int *score){
    FILE * fp;

    fp = fopen ("score.txt", "w+");
    fprintf(fp, "%s %d", "Score :", *score);

    fclose(fp);
}
```

Figure 21 - Fonction scoreInFile

### *B) Quitter le jeu*

Lorsque vous êtes sur le jeu vous avez aperçu que l'on peut quitter le jeu soit en utilisant la croix rouge en haut à droite, ou bien appuyer sur le bouton Quitter. Comme pour les boutons précédents c'est un Widget qui prend en paramètres :

Char \*label : correspond au nom que l'on souhaite donner au bouton ici « Quitter »

ButtonCB func : est la fonction de rappel (callback) qui va être appelée lorsqu'on appuie dessus soit QuitGame.

Void \*data : correspond à la donnée partager ici nous n'en avons pas l'utilité.

```
//Fin de jeu  
Exit = MakeButton("Quitter", QuitGame, NULL);
```

Figure 23 - Widget Exit

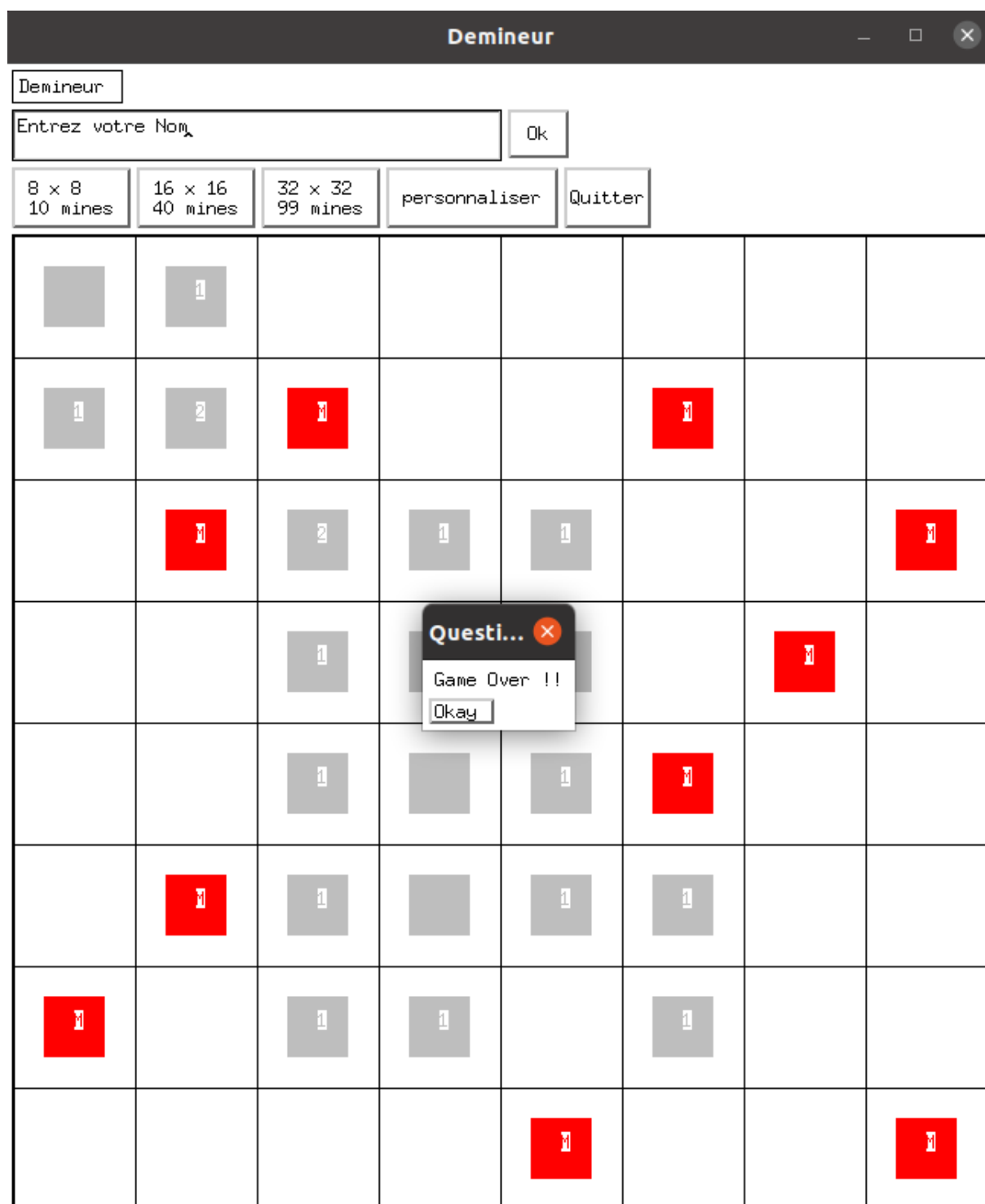
Ainsi en cliquant, sur le bouton on appelle la fonction de rappel QuitGame qui met fin à l'affichage et au programme grâce à la fonctionnalité CloseWindow de libsx.

```
void QuitGame(Widget *w){  
    //Fin de programme  
    CloseWindow();  
}
```

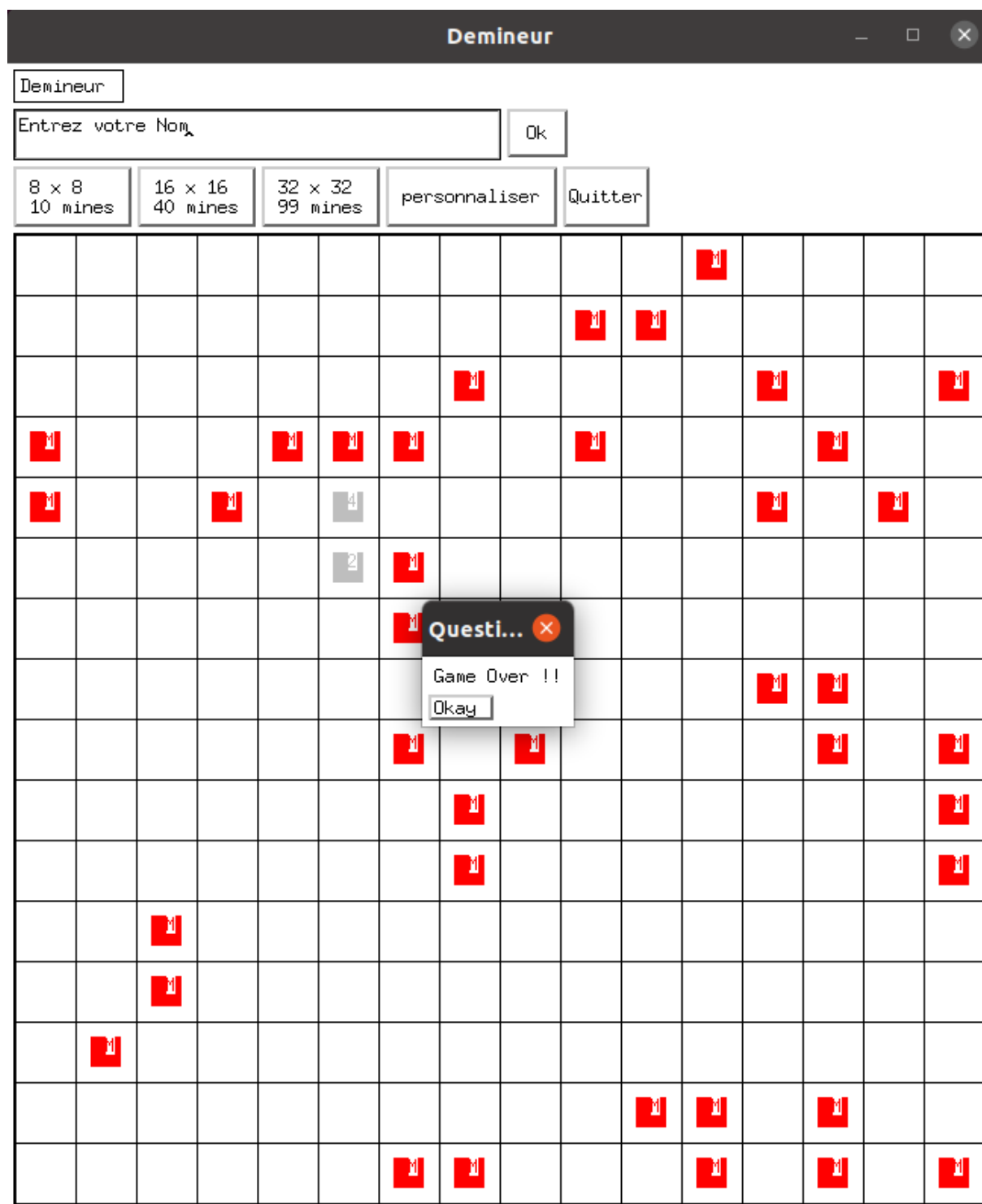
Figure 24 - Fonction de rappel QuitGame

## Conclusion

Ce projet a été intéressant à réaliser, nous avons pu mettre en pratique énormément de notions vues en cours. Ainsi que d'apprendre comment fonctionne l'architecture de la librairie graphique libsx. La compréhension de cette librairie n'a pas été évidente de prime abord mais en cherchant dans la documentation, nous avons pu comprendre son fonctionnement.

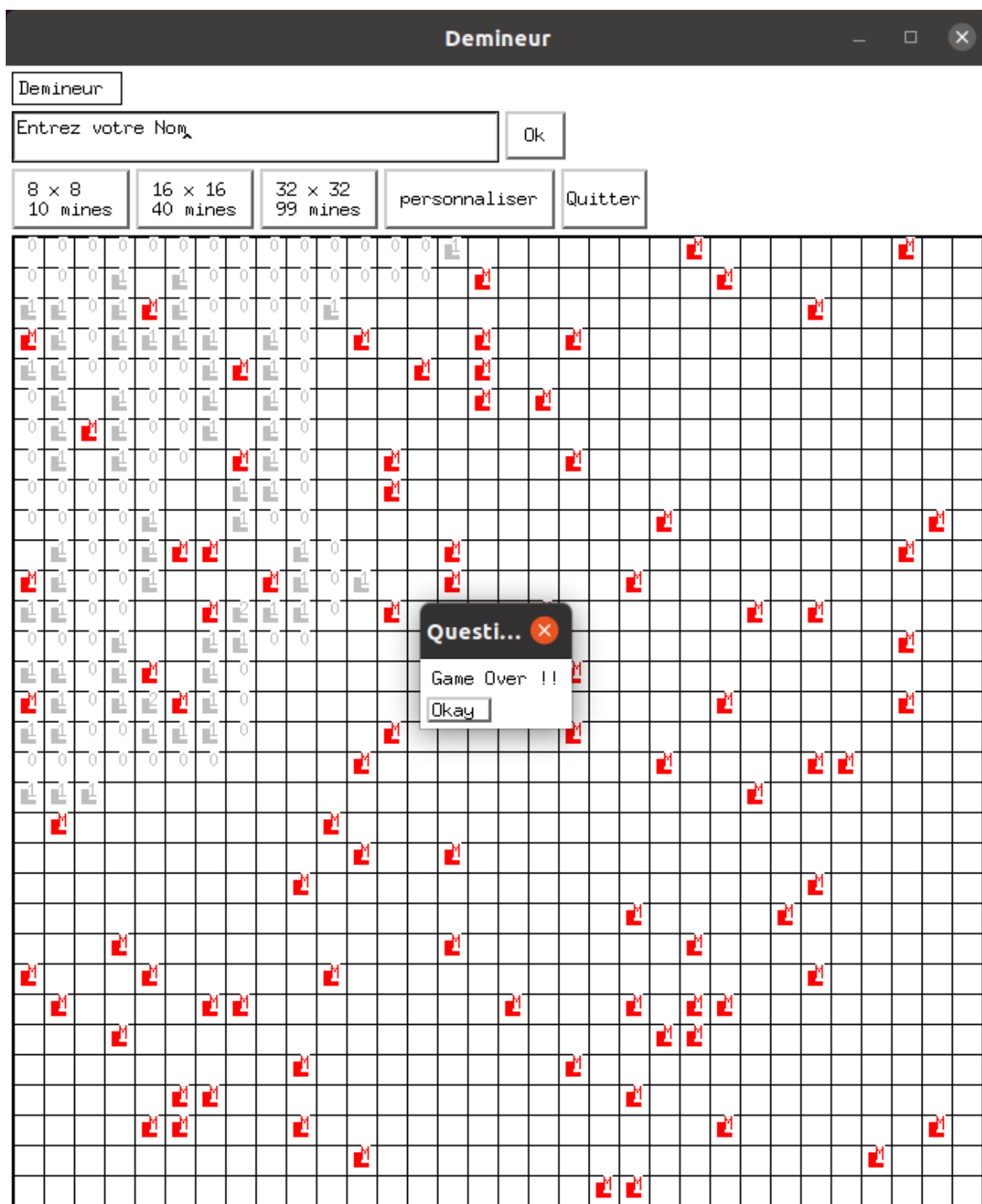
**ANNEXES :**

Exemple de partie en mode facile



Exemple de partie en mode moyen





Exemple de partie en mode difficile