

Sujet :

Atelier de Programmation

GTK

Filière d'ingénieur :

Ingénierie Logicielle et Intégration des Systèmes Informatiques



Encadrant :

Abdelkrim BEKKHOUCHA

Réalisé par :

KOMI Adama.

MADHOUR Souhaib.

Sawadogo Wamanegba Yacouba.

BAILLA Hamza.

Liste des figures

Figure 1 : Le premier interface	9
Figure 2 : Le deuxième interface	10
Figure 3 : Les gladiateurs et les objets	11
Figure 4 : L'arène, sauvegarde et charge.....	12
Figure 5 : Exemple de fichier de sauvegarde	14

I-INTRODUCTION :

Dans notre projet, nous avons développé un jeu de gladiateurs où chaque combattant possède des caractéristiques uniques telles que l'agressivité, la force, les armes et la protection. Les gladiateurs se battent dans une arène hostile, et leur comportement en combat est influencé par leur niveau d'agressivité. L'utilisateur a le rôle de sélectionner et de personnaliser les gladiateurs avant de les envoyer dans l'arène. L'arène comprend également des éléments de décor, comme une maison, un arbre et un rocher, qui offrent des refuges aux gladiateurs.

Chapitre I :

Dossier d'Analyse

I-1 Principe du jeu :

Dans notre projet, nous avons développé un jeu mettant en scène des gladiateurs dotés de caractéristiques telles que l'agressivité, la force, les armes et la protection. Ces gladiateurs évoluent dans une arène hostile où leurs semblables tentent de les tuer. L'agressivité est un paramètre clé qui détermine le comportement d'un gladiateur : un gladiateur agressif attaquera ceux qu'il croise, tandis qu'un gladiateur non agressif évitera d'attaquer les autres, mais se défendra s'il est attaqué. S'il possède des armes et des armures supérieures, il peut remporter des combats.

Dans l'arène, il y a également des objets non combatifs utilisés comme décors, qui permettent aux gladiateurs de s'y réfugier. Ces objets sont une **maison**, un **arbre** et un **rocher**.

I-2 Description du jeu :

Notre jeu se déroule dans un univers appelé Arène, Elle possède donc des dimensions et aucun gladiateur créé dans cet espace ne peut franchir les limites de l'arène. Pour éviter qu'ils ne s'échappent nous avons limités leur déplacement et lorsqu'ils se déplacent nous effectuons des calculs pour vérifier leur proximité avec les limites de l'arène, une fois qu'ils sont vers les bordures, nous leur affectons un déplacement opposé à leur déplacements courant. Pour marquer le décor de l'arène, nous avons utilisé une image pour la recouvrir.

L'entité gladiateur est créée avec plusieurs spécifications.

Pour créer des entités nous les avons attribués un niveau de vie égale à **1000** correspondant à **100%** de sa vie.

Ensuite nous lui spécifions, l'arme qu'il utilisera pour combattre. Dans notre projet nous avons utilisés **5 types** d'armes, à savoir les **Dagues**, les **sabres**, les **marteaux**, les **chaines**, les **Haches**. Chaque type d'arme dispose d'un niveau prédéfini de dégât qu'elle peut engendrer.

Nous passons par la suite en lui spécifiant son armure.

L'armure peut être en **Or**, en **Bronze** ou en **Fer**, chaque peut posséder en supplément un bouclier ou pas. Ainsi la protection de chaque gladiateur reviendra de la capacité d'amortissent de son armure.

En plus de ces spécifications, nous avons d'autres paramètre comme Le niveau d'expérience calculé en fonction du nombre de combat gagnés, ainsi le niveau d'expérience s'augmente de 5 points à chaque combat gagné.

I-3 Scenarios :

Les gladiateurs créés se déplacent dans l'arène :

Situation de non combat :

Lorsque deux gladiateurs non agressifs, se croisent dans l'arène, il n'y a pas de combat, chaqu'un continue sa route.

Situation de combat :

Lorsque deux gladiateurs se croisent que l'un d'eux est agressif, automatiquement, ils déclenchent un combat, aucun autre ne peut interférer dans leur combat.

Dans le cas d'un combat la probabilité de victoire d'un combattant est donnée par la différence de des armes de combat, le niveau d'expérience ainsi que l'armure pour la protection. Ainsi un gladiateur avec une arme en Marteau avec une armure en Or et un bouclier est plus proche de la victoire que son adversaire avec d'autre arme.

Pendant le combat lorsque les gladiateurs s'infligent des dégâts, nous avons leurs niveaux de vie qui diminuent. Nous associerons donc les gladiateurs avec des frames de niveau de vie pour présenter leur niveau de santé.

Dans notre analyse, lors d'un combat, le gladiateur perçoit de l'expérience lorsqu'il gagne un combat, il recouvre aussi sa santé et récupère 50% de la force de son adversaire défait.

Situation de cachette :

Lorsqu'un gladiateur dans sa course croise un objet (**Arbre**, **Maison**, **Rocher**), il se cache dans celui-ci, il attend ainsi, 15 mis à jour de l'horloge avant de sortir de sa cachette et de recontinuer sa progression.

Présentation des interfaces :

Nous avons développé deux interfaces de jeu. La première, destinée à l'accueil du jeu, comporte deux boutons : l'un permet d'accéder à l'arène de jeu et aux gladiateurs, et l'autre sert à quitter le jeu et fermer l'interface. La deuxième interface comprend des boutons pour gérer les gladiateurs, accéder à l'arène de combat, et choisir les forces des gladiateurs.

Démonstration :

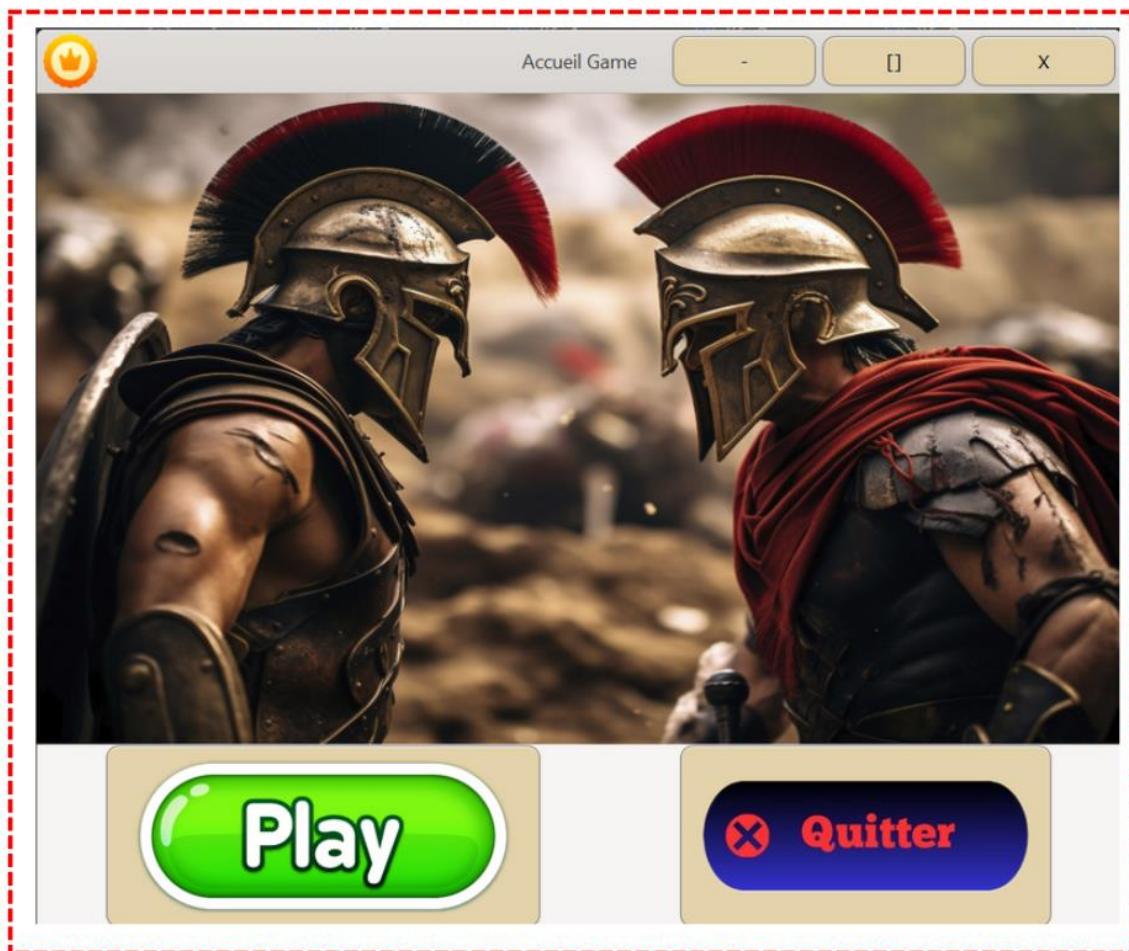


figure 1: Le premier Interface.

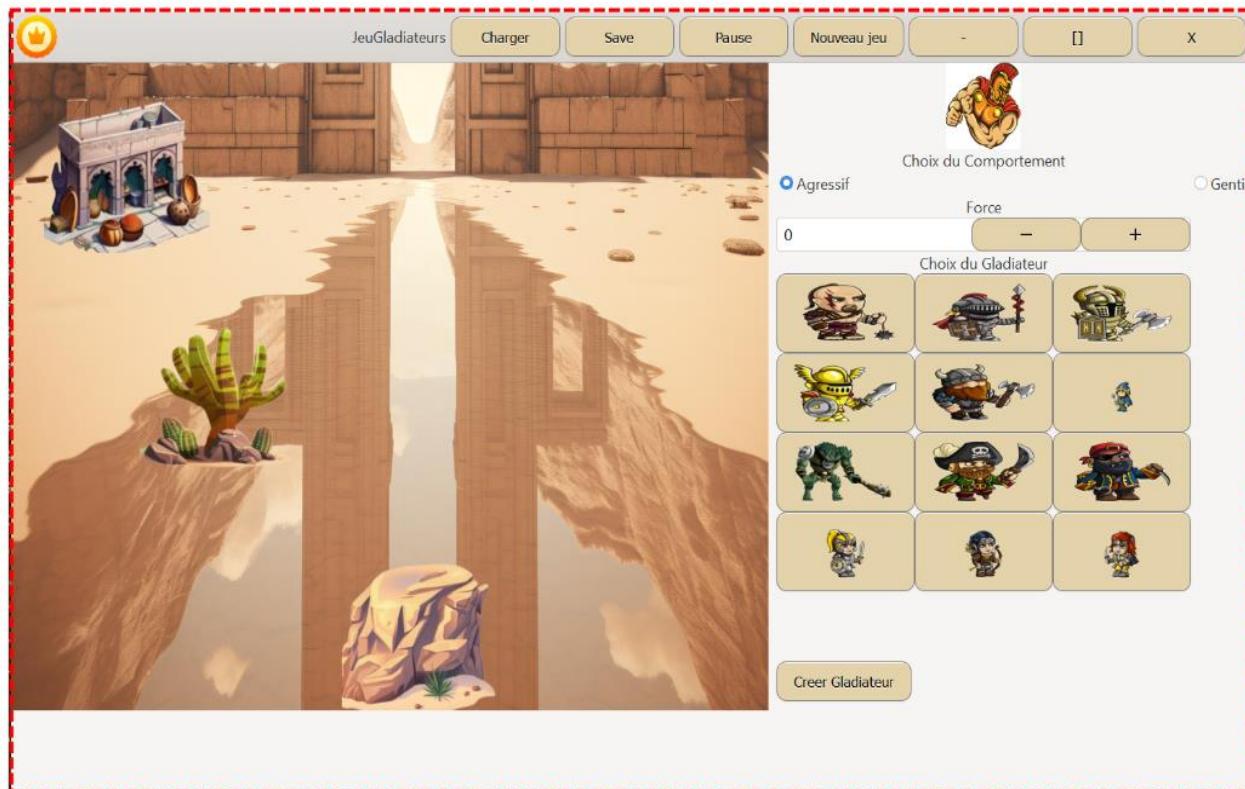


figure 2: Le deuxième interface

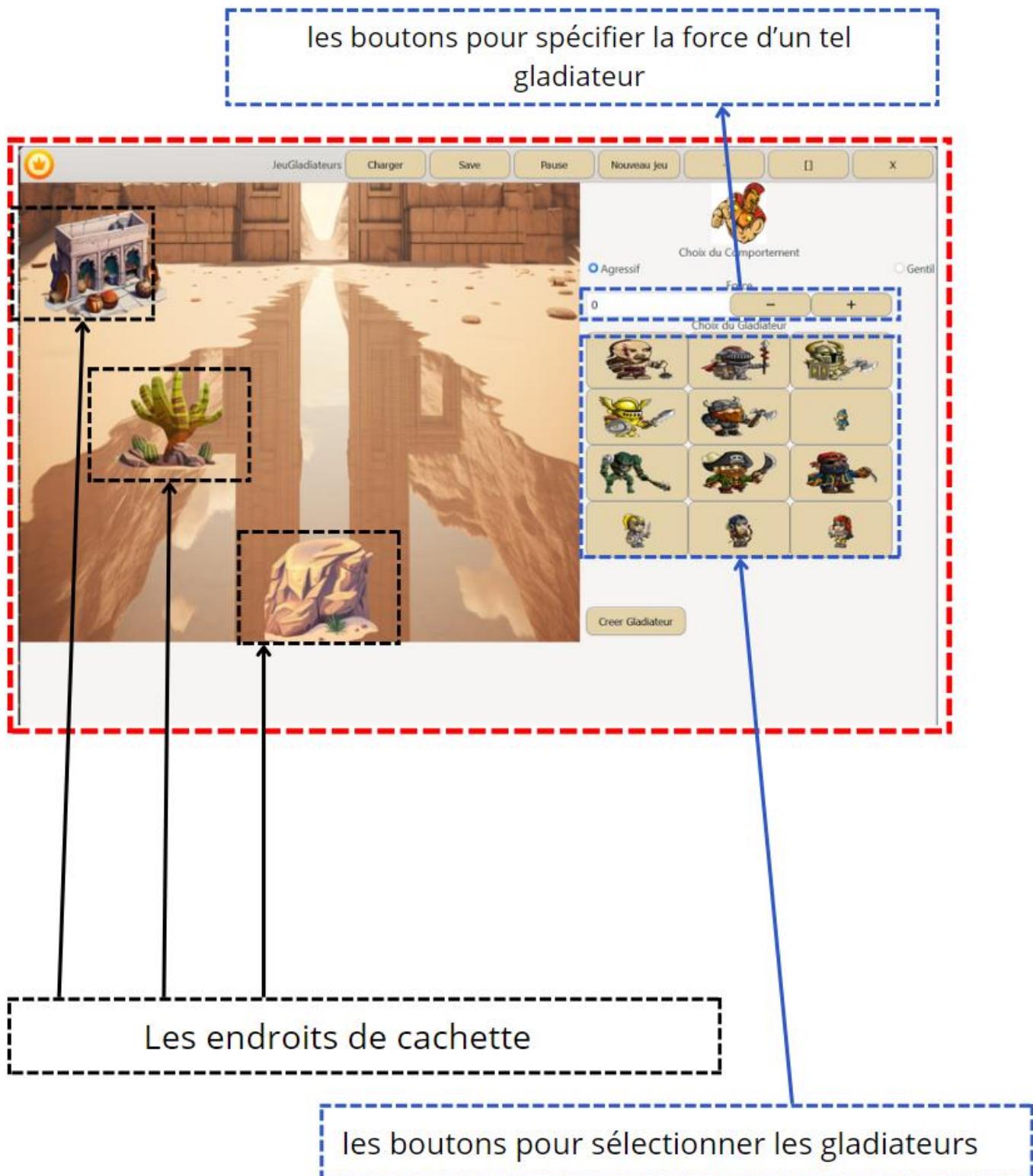


figure 3: Les gladiateurs et les objets

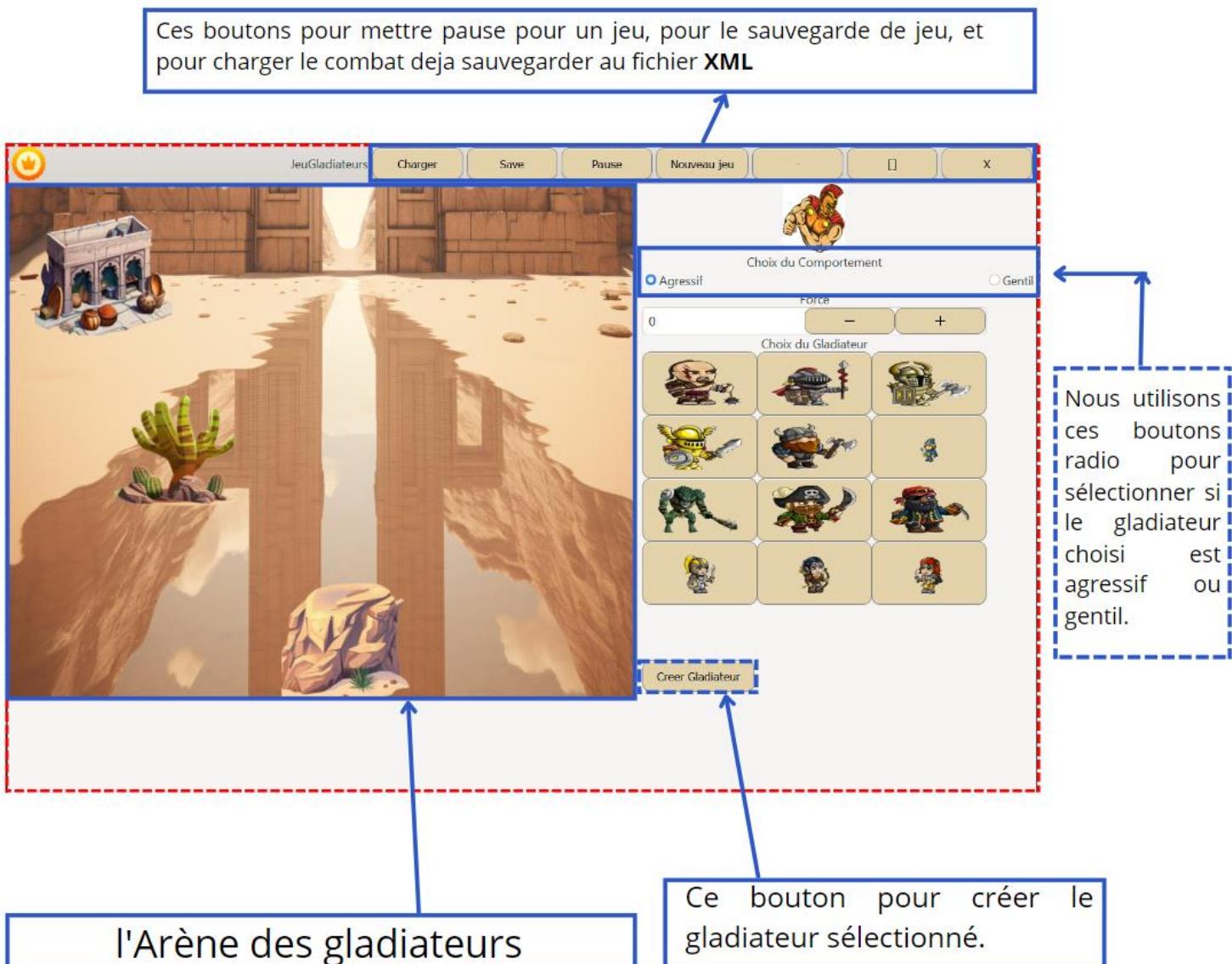


figure 4: l'arene , sauvegarde et charge

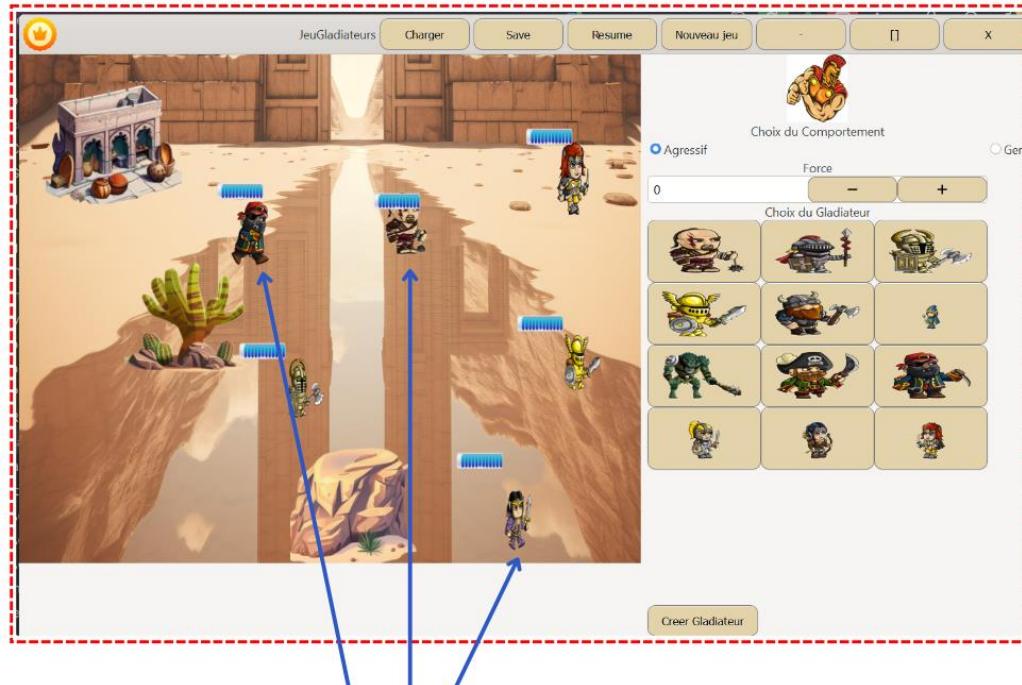
Les Boutons Utilisées :

1-Nouveau jeu : Ce bouton permet de réinitialiser le jeu. Il vide l'arène de tous les gladiateurs et, en back-end, appelle la fonction **New_game()**.

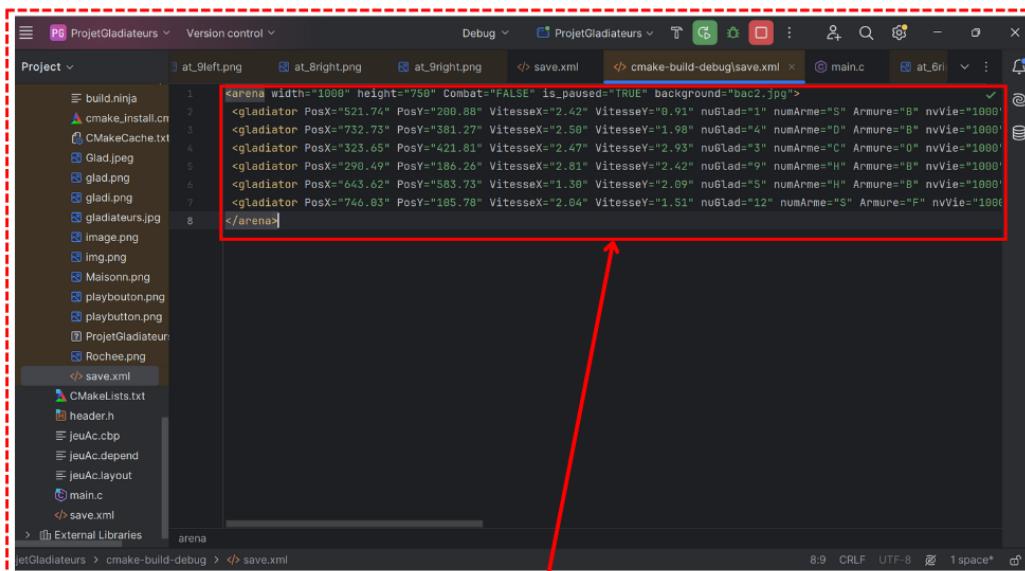
2-Pause : Ce bouton permet de mettre le jeu en pause. Le jeu reste en pause jusqu'à ce qu'un nouveau clic sur le même bouton le reprenne.

3-Save : Ce bouton permet de sauvegarder l'état actuel du jeu. En principe, il appelle la fonction **save_game()**, qui génère un fichier **XML** contenant toutes les caractéristiques de l'arène. Ce fichier comprend la balise "aréna" contenant les balises de chaque gladiateur avec leurs attributs représentant leurs caractéristiques dans l'arène, tels que la position et la force.

Exemple :



Nous avons 6 gladiateurs dans l'arène



```

<arena width="1000" height="750" Combat="FALSE" is_paused="TRUE" background="bac2.jpg">
    <gladiador PosX="521.74" PosY="200.88" VitesseX="2.42" VitesseY="0.91" nu6lad=1 numArme="S" Armure="B" nvVie="1000" />
    <gladiador PosX="732.73" PosY="381.27" VitesseX="2.50" VitesseY="1.98" nu6lad=4 numArme="D" Armure="B" nvVie="1000" />
    <gladiador PosX="323.65" PosY="421.81" VitesseX="2.47" VitesseY="2.93" nu6lad=3 numArme="C" Armure="O" nvVie="1000" />
    <gladiador PosX="290.49" PosY="186.26" VitesseX="2.81" VitesseY="2.42" nu6lad=0 numArme="H" Armure="B" nvVie="1000" />
    <gladiador PosX="643.62" PosY="583.73" VitesseX="1.38" VitesseY="2.09" nu6lad=5 numArme="H" Armure="B" nvVie="1000" />
    <gladiador PosX="746.03" PosY="185.78" VitesseX="2.04" VitesseY="1.51" nu6lad=12 numArme="F" Armure="F" nvVie="1000" />
</arena>

```

Ce code XML représente l'arène et les gladiateurs avec leurs positions qu'on déjà sauvegarder dans le fichier **save.xml**

figure-5:Exemple de sauvegarde

4-Charger : Ce bouton permet de charger le fichier XML préalablement sauvegardé de l'arène. Il fait appel à la fonction **parsefile()** pour lire le fichier au format XML et le traduire dans l'arène.

5-Créer Gladiateur : Ce bouton permet de lancer le gladiateur sélectionné dans l'arène. Il appelle la fonction **create_gladiator()** pour créer le gladiateur choisi, puis le déployer dans l'arène.

Chapitre II :

Dossier de Programmation

```

● ● ●

#include <gtk/gtk.h>
#include <cairo.h>
#include <math.h>
#include <stdio.h>
#define WIN32_LEAN_AND_MEAN           //biblio pour ajouter le son
#include <windows.h>                //biblio pour ajouter le son
#include <mmsystem.h>               //biblio pour ajouter le son
#include <gdk-pixbuf/gdk-pixbuf.h>   //biblio pour ajouter le son
#include <stdlib.h>
#include<string.h>
#include "header.h"                  //pour inclure l'Analyseur XML
#define Talloc(x) (x *)malloc(sizeof(x))
#define NUM_FRAMES 12

/******************
 * structure pour la taille
*****************/
typedef struct
{
    int height; ///longueur
    int width; ///largeur
}Taille;

/******************
 * structure d'une fenetre
*****************/
typedef struct
{
    char titre[15]; ///titre du widget
    GtkWidget *window; ///la fenetre
    char icon[15]; ///chemin de l'icone de la fenetre gboolean iconexist;
    Taille dim; ///dimension de la fenetre
}fenetre;

/******************
 * structure d'un bouton
*****************/
typedef struct
{
    char titre[15]; ///titre du widget
    GtkWidget *button;/// le bouton
    char type; /// 1 :normale/2 :radio/3 :check/4 :spin
    gboolean active;///TRUE active FALSE desactive
}Bouton;

/******************
 * structure d'une image
*****************/
typedef struct
{
    GtkWidget *image; /// Widget d'image
    char path[20]; /// Chemin vers le fichier d'image
    Taille dim;/// Taille de l'image
} MyImage;

/******************
 * structure d'une box
*****************/
typedef struct
{
    GtkWidget *box; ///initialisation de widget box;
    gchar *orient; ///l'orientation dans la fenetre {horizontal,vertical};
}Box;

```

```

● ● ●

*****  

* structure d'un label  

*****  

typedef struct  

{  

    GtkWidget* label; //initialisation de widget label;  

    gchar *text; //le texte a ecrire;  

}Label;

*****  

* structure d'une grid  

*****  

typedef struct  

{  

    GtkWidget *grid;//pointeur sur le grid  

}Mongrid;

*****  

* structure de positionnement  

*****  

typedef struct  

{  

    double X;///Sur l'axe des absisses  

    double Y;///Sur l'axe des ordonnée  

}Dep;

*****  

* Structure pour les objets du jeu  

*****  

typedef struct  

{  

    Dep *position;///Position des objets  

    GdkPixbuf *image;///Image objet  

    GdkPixbuf *scaled;///Image objet  

}Objet;

///Les Objets
Objet *Tab[3];

*****  

*Structure de données pour un gladiateur
*****  

typedef struct  

{  

    int nuGlad;///Numero Gladiateur  

    int nvforce;///Niveau de force  

    int nvVie;///Niveau de vie  

    Dep *Pos;///Position du gladiateur  

    Dep *Vitesse;///Vitesse du gladiateur  

    gboolean Agressif ;///TRUE:Agressif ; FALSE:Gentil //Agressif du Gladiateur  

    char numArme;///Arme du gladiateur  

    char Armure;///Armure du gladiateur  

    gboolean bouclier;///Bouclier  

    int nvExp;///Niveau d'experience en fonction des combats  

    gboolean isControlled;///Qui le controle (user ou automatique)  

    gboolean isfighting;///En combat  

    gboolean iswalking;///Entrain de marché  

    gboolean isattaking;///Attaque ou defense  

    int nbCgagne;///Nombre de combat gagné  

    int current_frame; // Indice de l'image de saut actuelle  

    gboolean isMovingRight; //pour savoir le sens de direction  

    gboolean hidden;///Montre si le gladiateur est caché  

    int nbhid;///Compteur pour cacher le gladiateur  

    int hidway;///Endroit ou est caché le gladiateur  

}Gladiateur;

```

```

● ● ●

/*
*Structure de données pour les
*mouvements du gladiateur
*/
typedef struct
{
    GdkPixbuf *FramesR[12];///deplacement a droite
    GdkPixbuf *FramesL[12];///deplacement a gauche
    GdkPixbuf *FramesAR[12];///attaque a droite
    GdkPixbuf *FramesAL[12];///attaque a gauche
}Fram;

/*
*Structure de données pour les frames des 13 gladiateurs
*/
Fram *TabFr[13];
///le niveau de vie des gladiateurs
GdkPixbuf *Vie[7];

///Liste des gladiateurs
GList *gladiators = NULL;

///Gladiateur contrôlé par l'utilisateur (NULL s'il n'y en a pas)
Gladiateur *cont_gladiator = NULL;

/*
*Structure de l'interface
*/
typedef struct
{
    fenetre *F;///la fenetre
    Bouton *B1;///bouton radio pour la selection de l'agressivité
    Bouton *B2;///bouton radio pour la selection de l'agressivité
    MyImage *Image;/// image
    Box *Boxe;/// structure de box
    Mongrid *grid;///structure d'une grid
    Label *lab;///structure d'un label
    Gladiateur *G; ///structure d'un gladiateur
    gboolean combat; ///pour gerer le combat
    int choix; /// choix du gladiateur selectionné
    gboolean is_paused;///si l'interface est en pause
    /// Dimensions de l'arène
    int ARENA_WIDTH;///largeur de l'arène
    int ARENA_HEIGHT;///hauteur de l'arène
    GtkWidget *arena;/// l'arene
    GdkPixbuf *image;/// image
    GdkPixbuf *scaled;///Image objet
    char Arenabac[30];///chemin de l'image de fond de l'arène
}Interface;

/*
Fonction: TesterAllocation: une fonction definie qui permet de tester
si l'allocation d'un element a reussi. Le principe est de
pouvoir passer n'importe quel type d'element en parametre
Entree: elem: L'element dont on veut tester
Sortie: affiche un message d'echec si l'allacation echoue
*/
void TesterAllocation(elem)
{
    if (!elem) {
        printf("\nErreur d'allocation");
        exit(-1);
    }
}

```



```

*****  

Fonction: init_image: pour initialiser une image  

Entree: char path[50]: le chemin vers l'image  

        int width: largeur de l'image  

        int height: hauteur de l'image  

Sortie: retourne la structure de l'image initialisée  

*****  

MyImage *init_image( char path[50], int width, int height)  

{  

    ///allocation d'espace  

    MyImage *info =Talloc(MyImage);  

    ///tester l'allocation  

    TesterAllocation(info);  

    ///remplir les champs  

    strcpy( info->path, path );///le chemin de l'image  

    info->dim.width = width;/// largeur de l'image  

    info->dim.height = height; ///hauteur de l'image  

    return(MyImage *)info;  

}  

*****  

Fonction: creer_image : pour creer une image  

Entree: MyImage *I : la structure de l'image initialisée  

Sortie: rien  

*****  

void creer_image(MyImage *I)  

{  

    ///Charge une image depuis un fichier spécifié par le chemin  

    GdkPixbuf *src_pixbuf = gdk_pixbuf_new_from_file(I->path, NULL);  

    ///Redimensionne l'image chargée aux dimensions spécifiées  

    GdkPixbuf *resized_pixbuf =  

        gdk_pixbuf_scale_simple(src_pixbuf,I->dim.width,I->dim.height,  

    GDK_INTERP_BILINEAR);  

    ///Libère la mémoire occupée par le pixbuf source  

    g_object_unref(src_pixbuf);  

    ///Crée un widget GTK image à partir du pixbuf redimensionné et l'associe à l'image  

    I->image= gtk_image_new_from_pixbuf(resized_pixbuf);  

}  

*****  

Fonction: liberer_gladiateur : pour liberer un gladiateur  

Entree: Gladiateur *gladiator : le gladiateur à liberer  

Sortie: rien  

*****  

void liberer_gladiateur(Gladiateur *gladiator)  

{  

    if (!gladiator) return;  

    /// Libérer les ressources spécifiques du gladiateur  

    if (gladiator->Vitesse)///liberer l'espace memoire de la vitesse  

    {  

        free(gladiator->Vitesse);  

    }  

    if (gladiator->Pos)///liberer l'espace memoire de la position  

    {  

        free(gladiator->Pos);  

    }  

    /// Libérer la structure du gladiateur elle-même  

    free(gladiator);  

} //FIN DE LA FONCTION.

```

```

● ● ●

*****  

Fonction: new_game : pour reinitialiser le jeu
Entree: GtkWidget *widget : widget déclenchant la réinitialisation du jeu
        gpointer user_data : pointeur vers les données utilisateur (structure Interface)
Sortie: rien
*****  

void new_game(GtkWidget *widget, gpointer user_data)
{
    ///Cast de user_data en structure Interface, contenant les informations de l'interface
    Interface *interface = (Interface *)user_data;
    GList *iter = gladiators;///la liste des gladiateurs
    /// Libérer chaque gladiateur dans la liste
    while (iter)
    {
        Gladiateur *gladiateur = (Gladiateur *)iter->data;///l'element courant
        /// Avancer l'itérateur avant de supprimer l'élément
        iter = g_list_next(iter);
        ///liberer l'espace memoire des paramètres du gladiateur
        liberer_gladiateur(gladiateur);
    }
    /// Libérer la liste elle-même
    g_list_free(gladiators);
    gladiators = NULL;
    interface->combat=FALSE;///Mettre l'état de combat à FALSE(plus de combat en cours)
    /// Redessiner l'arène après la suppression des gladiateurs
    gtk_widget_queue_draw(interface->arena);
}

*****  

Fonction: on_pause_clicked : pour mettre en pause le jeu
Entree: GtkWidget *widget : widget déclenchant la mise en pause du jeu
        gpointer user_data : pointeur vers les données utilisateur (structure Interface)
Sortie: rien
*****  

void on_pause_clicked(GtkWidget *widget, gpointer user_data)
{
    ///Cast de user_data en structure Interface, contenant les informations de l'interface
    Interface *interface = (Interface *)user_data;
    interface->is_paused = !interface->is_paused;///changer l'état boolean du bouton
    if (interface->is_paused)
    {///changer le bouton Pause en Resume s'il est en pause
        gtk_button_set_label(GTK_BUTTON(widget), "Resume");
    } else
    {///changer le bouton Resume en Pause s'il est en Resume
        gtk_button_set_label(GTK_BUTTON(widget), "Pause");
    }
}

*****  

Fonction: init_fenetre : pour initialiser une fenêtre
Entree: char titre[15] : le titre de la fenêtre
        char icon[15] : chemin de l'icone de la fenêtre
        int w : largeur de la fenêtre
        int h : hauteur de la fenêtre
Sortie: retourne la structure de l'image initialisée
*****  

fenetre *init_fenetre(char titre[15],char icon[15],int w,int h)
{
    ///Allocation de l'espace memoire
    fenetre *Nf=Talloc(fenetre);
    ///si l'allocation echoue
    TesterAllocation(Nf);
    ///Remplir les champs
    strcpy(Nf->titre,titre);///titre de la fenêtre
    strcpy(Nf->icon,icon);/// icon sous forme de chemin
    Nf->dim.height=h;///largeur de la fenêtre
    Nf->dim.width=w;///hauteur de la fenêtre

    return(fenetre*)Nf;
}

```

```

● ● ●

*****Fonction: load_game_state : pour charger le jeu*****
Entree: GtkWidget *w : widget déclenchant le chargement
        gpointer user_data : pointeur vers les données
                    utilisateur (structure Interface)
Sortie: rien
*****void load_game_state(GtkWidget *w, gpointer data)
{
    GList *iter = gladiators;
    /// Libérer chaque gladiateur dans la liste
    while (iter != NULL)
    {
        Gladiateur *gladiateur = (Gladiateur *)iter->data;///element courant
        iter = g_list_next(iter); // Avancer l'itérateur avant de supprimer l'élément
        liberer_gladiateur(gladiateur);///liberer l'espace
    }
    /// Libérer la liste elle-même
    g_list_free(gladiators);
    gladiators = NULL;
    /** Cast de user_data en structure Interface,
    contenant les informations de l'interface */
    Interface *I=(Interface *)data;
    //charger le fichier de sauvegarde
    parsefile("save.xml",I);
}

*****Fonction: save_game : pour sauvegarder le jeu*****
Entree: GtkWidget *w : widget déclenchant la sauvegarde
        gpointer user_data : pointeur vers les données
                    utilisateur (structure Interface)
Sortie: rien
*****void save_game(GtkWidget *w, gpointer data)
{
    /** Cast de user_data en structure Interface, contenant
    les informations de l'interface */
    Interface *I=(Interface*)data;
    FILE *file = fopen("save.xml", "w");///ouvrir le fichier de sauvegarde
    if (!file)//si l'ouverture a échoué
    {
        fprintf(stderr, "Failed to open file for writing: %s\n", "save.xml");
        return;
    }
    /// Ouvrir la balise root
    fprintf(file,
            "<arena width=\"%d\" height=\"%d\" Combat=\"%s\" is_paused=\"%s\" background=\"%s\">\n"
            , I->ARENA_WIDTH, I->ARENA_HEIGHT, I->combat? "TRUE": "FALSE",
            I->is_paused? "TRUE": "FALSE", I->Arenabac);

    /// Parcourir la liste des gladiateurs et écrire leurs informations dans le fichier
    for (GList *iter = gladiators; iter != NULL; iter = g_list_next(iter))
    {
        Gladiateur *gladiateur = (Gladiateur *)iter->data;
        fprintf(file,
                " <gladiator PosX=\"%2f\" PosY=\"%2f\" VitesseX=\"%2f\" VitesseY=\"%2f\""
                nuGlad=%"d"
                numArme=%"c" Armure=%"c" nvVie=%"d" nvforce=%"d" nvExp=%"d" nbCgagne=%"d"
                current_frame=%"d" nbhid=%"d" hidway=%"d" isControlled=%"s" isMovingRight=%"s"
                isfighting=%"s" iswalking=%"s" isattaking=%"s" Agressif=%"s" bouclier=%"s"
                hidden=%"s"/>\n",
                gladiateur->Pos->X, gladiateur->Pos->Y, gladiateur->Vitesse->X, gladiateur->Vitesse-
                >Y, gladiateur->nuGlad,
                gladiateur->numArme, gladiateur->Armure, gladiateur->nvVie, gladiateur->nvforce, gladiateur-
                >nvExp,
                gladiateur->nbCgagne, gladiateur->current_frame, gladiateur->nbhid, gladiateur->hidden?
                gladiateur->hidway:-1,
                gladiateur->isControlled ? "TRUE" : "FALSE", gladiateur->isMovingRight? "TRUE" : "FALSE",
                gladiateur->isfighting ? "TRUE" : "FALSE", gladiateur->iswalking ? "TRUE" : "FALSE",
                gladiateur->isattaking ? "TRUE" : "FALSE", gladiateur->Agressif? "TRUE" : "FALSE",
                gladiateur->bouclier? "TRUE" : "FALSE", gladiateur->hidden? "TRUE" : "FALSE");
    }

    /// Fermer la balise root
    fprintf(file, "</arena>");

    fclose(file);
} //FIN DE LA FONCTION.

```

```

/*
*****
Fonction: toggle_maximize : pour maximiser la taille de la fenetre
Entree: GtkWidget *button : le bouton maximisation
        gpointer user_data : pointeur vers les données utilisateur
Sortie: rien
*****
void toggle_maximize(GtkButton *button, gpointer data)
{
    /// Cast de user_data en structure window, contenant les informations de la fenetre
    GtkWidget *window = GTK_WINDOW(data);
    ///si la fenetre est deja a la taille maximale on la minimise
    if (gtk_window_is_maximized(window))
        gtk_window_unmaximize(window);
    else//si la taille de la fenetre n'est pas maximale on la maximise
        gtk_window_maximize(window);
}

/*
*****
Fonction: on_reduce_button_clicked : pour reduire la taille de la fenetre
Entree: GtkWidget *button : le bouton maximisation
        gpointer user_data : pointeur vers les données utilisateur
Sortie: rien
*****
void on_reduce_button_clicked(GtkButton *button, gpointer user_data)
{
    /// Recuperer la fenetre associee au widget button
    GtkWidget *window = gtk_widget_get_toplevel(GTK_WIDGET(user_data));

    /// Vérifier que window est une fenetre valide
    if (GTK_IS_WINDOW(window))
        /// Reduire la fenetre
        gtk_window_iconify(GTK_WINDOW(window));
    else
        /// Gerer le cas ou window n'est pas une fenetre valide
        g_warning("Le widget associe au bouton n'est pas une fenetre valide.");
}

/*
*****
Fonction: creer_fenetre : pour la creation de la fenetre
Entree: Interface *I : la structure de l'interface contenant
        les informations de la fenetre
        gboolean edit : definit si la fenetre est editable ou pas
Sortie: rien
*****
void creer_fenetre(Interface *I, gboolean edit)
{
    GtkWidget *button;///pour la creation des boutons
    MyImage *im=init_image(I->F->icon,50,50);///initialisation de l'icone de la fenetre
    ///Si le type est Toplevel
    I->F->window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
    ///Mettre la taille a la fenetre
    gtk_window_set_default_size(GTK_WINDOW(I->F->window),
                                I->F->dim.width,I->F->dim.height);

    /// Creez une barre de titre personnalisée
    GtkHeaderBar *header_bar = GTK_HEADER_BAR(gtk_header_bar_new());
    GtkWidget *box=gtk_box_new(GTK_ORIENTATION_HORIZONTAL,2);
    gtk_box_set_homogeneous(box,FALSE);
    gtk_container_add(GTK_CONTAINER(header_bar),box);
    ///creation de l'icone de la fenetre
    creer_image(im);
    gtk_container_add(GTK_CONTAINER(box),im->image);
    ///creation du label de la fenetre
    GtkWidget *title_label = gtk_label_new(I->F->titre);
    /// Set the label as the custom title
    gtk_header_bar_set_custom_title(GTK_HEADER_BAR(header_bar), title_label);
    /// Ajoutez un bouton de fermeture a la barre de titre
    GtkWidget *close_button = gtk_button_new_with_label("X");
    gtk_header_bar_pack_end(GTK_HEADER_BAR(header_bar),close_button);
    g_signal_connect_swapped(close_button, "clicked",
                             G_CALLBACK(gtk_widget_destroy),I->F->window);

    /// Ajoutez un bouton d'agrandissement a la barre de titre
    GtkWidget *maximize_button = gtk_button_new_with_label("[+]");
    gtk_header_bar_pack_end(GTK_HEADER_BAR(header_bar), maximize_button );
    /** Connectez le signal "clicked" du bouton d'agrandissement a
    une fonction de gestion personnalisee ***/
    g_signal_connect(maximize_button, "clicked",
                     G_CALLBACK(toggle_maximize), I->F->window);

    /// Ajoutez un bouton de reduction a la barre de titre
    GtkWidget *reduce_button = gtk_button_new_with_label("-");
    gtk_header_bar_pack_end(GTK_HEADER_BAR(header_bar), reduce_button);
    /**Connectez le signal "clicked" du bouton de reduction a
    la fonction de reduction de la fenetre ***/
    g_signal_connect(reduce_button, "clicked",
                     G_CALLBACK(on_reduce_button_clicked),GTK_WINDOW(I->F->window));
    if(edit)
    {
        /// Ajoutez un bouton de Nouveau jeu
        button = gtk_button_new_with_label("Nouveau jeu");
        gtk_header_bar_pack_end(GTK_HEADER_BAR(header_bar),button);
        g_signal_connect(button, "clicked", G_CALLBACK(new_game), I);
        /// Ajoutez un bouton de pause
        button = gtk_button_new_with_label("Pause");
        g_signal_connect(button, "clicked", G_CALLBACK(on_pause_clicked), I);
        gtk_header_bar_pack_end(GTK_HEADER_BAR(header_bar),button);
        /// Ajoutez un bouton de sauvegarde
        button = gtk_button_new_with_label("Save");
        g_signal_connect(button, "clicked", G_CALLBACK(save_game), I);
        gtk_header_bar_pack_end(GTK_HEADER_BAR(header_bar),button);
        /// Ajoutez un bouton de chargement
        button = gtk_button_new_with_label("Charger");
        g_signal_connect(button, "clicked", G_CALLBACK(load_game_state), I);
        gtk_header_bar_pack_end(GTK_HEADER_BAR(header_bar),button);
    }
    ///Attachez la barre de titre a la fenetre
    gtk_window_set_titlebar(GTK_WINDOW(I->F->window), GTK_WIDGET(header_bar));
    /// definir la position de la fenetre
    gtk_window_set_position(GTK_WINDOW(I->F->window),GTK_WIN_POS_CENTER);
    ///le signal de la fenetre
    g_signal_connect(I->F->window, "destroy", G_CALLBACK(gtk_main_quit), NULL);
}

```

```

● ● ●

/****************************************************************************
Fonction: Mongrid* Init_grid : pour initialiser une grid
Entree: rien
Sortie: retourne la structure de la grid initialisée
****************************************************************************/
Mongrid* Init_grid()
{
    Mongrid *g;
    //allocation d'espace memoire pour la structure
    g =Talloc(Mongrid);
    //tester l'allocation memoire
    TesterAllocation(g);
    //initialiser la grid
    g->grid = gtk_grid_new();
    return(Mongrid*)g;
}

/****************************************************************************
Fonction: void Ajouter_elem_grid : pour ajouter un element dans la grid
Entree: Mongrid *g: la structure de la grid initialisée
        GtkWidget* elem : l'element qui sera ajouté dans la grid
        int x : coordonnee en abscisse de l'element dans la grid
        int y : coordonnee en abscisse de l'element dans la grid
Sortie: rien
****************************************************************************/
void Ajouter_elem_grid(Mongrid *g, GtkWidget* elem,int x,int y)
{
    //ajouter l'elem child dans le grid selon les coordonnees et la taille du grid
    gtk_grid_attach(GTK_GRID(g->grid), elem, x,y, 1,1);
}

/****************************************************************************
Fonction: Label *initLabel : pour initialiser un label
Entree: Label *label: la structure du label a initialiser
        char *text : le texte du label
Sortie: retourne la structure du label initialisée
****************************************************************************/
Label *initLabel(Label *label,char *text)
{
    label = Talloc(Label); //allocation memoire du label
    TesterAllocation(label); //tester l'allocation
    //Convertir le texte en encodage local (locale) vers UTF-8 et l'assigner à label->text
    label->text =g_locale_to_utf8(text,-1,NULL, NULL,NULL);
    return label;
}

/****************************************************************************
Fonction: : pour creer un label
Entree: Label *label: la structure du label initialisée
Sortie: rien
****************************************************************************/
void CreerLabel(Label *label)
{
    //creation du widget label
    label->label = gtk_label_new(label->text);
}

/****************************************************************************
Fonction: Bouton *init_bouton : pour initialiser un bouton
Entree: char Label[30] : le texte du bouton
        boolean active : definit si le bouton est actif ou pas
        char type : definit le type de bouton
Sortie: retourne la structure du bouton initialisée
****************************************************************************/
Bouton *init_bouton(char Label[30],gboolean active,char type)
{
    // Allocion de l'espace memoire
    Bouton *NB=Talloc(Bouton);
    //Si l'allocation echoue quitter
    TesterAllocation(NB);
    //Sinon remplir les champs
    strcpy(NB->titre,Label);
    NB->active=active;
    NB->type=type;

    return(Bouton*)NB;
} //FIN DE LA FONCTION

```

```

● ● ●

/****************************************************************************
Fonction: Interface *radio_button_toggled : Fonction de rappel pour le
signal "clicked" d'un bouton
radio
Entree: GtkToggleButton *togglebutton : le bouton
gpointer user_data : pointeur vers les données utilisateur
Sortie: un pointeur vers la structure Interface
*/
Interface *radio_button_toggled(GtkToggleButton *togglebutton, gpointer user_data)
{
    /// Récupérer la structure de l'interface à partir des données utilisateur
    Interface *interface = (Interface *)user_data;
    /// Récupérer le gladiateur actuellement sélectionné dans l'interface
    Gladiateur *D = interface->G;
    /// Vérifier si le bouton radio est activé
    gboolean active = gtk_toggle_button_get_active(togglebutton);
    /// Si le bouton radio est activé
    if (active)
    {
        /// Récupérer le label du bouton radio
        gchar *label = gtk_button_get_label(GTK_BUTTON(togglebutton));
        /// Comparer le label pour déterminer le type de gladiateur (Agressif ou Gentil)
        if (!strcmp("Agressif", label))
        {
            /// Si le label est "Agressif", définir le gladiateur comme agressif
            D->Agressif = TRUE;
        }
        else if (!strcmp("Gentil", label))
        {
            /// Si le label est "Gentil", définir le gladiateur comme non agressif
            D->Agressif = FALSE;
        }
    }
    /// Retourner la structure de l'interface
    return (Interface *)interface;
}//FIN DE LA FONCTION

/****************************************************************************
Fonction: Value_SpineBoutton : Fonction de rappel pour le signal d'un bouton spin
Entree: GtkButton *button : le bouton GtkButton
gpointer user_data : pointeur vers les données utilisateur
Sortie: rien
*/
void Value_SpineBoutton(GtkButton *button, gpointer user_data)
{
    /// Récupérer la structure de l'interface à partir des données utilisateur
    Interface *interface = (Interface *)user_data;
    /// Récupérer le gladiateur actuellement sélectionné dans l'interface
    Gladiateur *D = interface->G;
    /// Vérifier si l'objet passé est un GtkSpinButton
    if (GTK_IS_SPIN_BUTTON(button))
    {
        /// Récupérer la valeur du GtkSpinButton
        int value = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(button));
        /// Mettre à jour la force du gladiateur avec la valeur obtenue
        D->nvforce = value;
    }
    else
    {
        /// Afficher un message d'erreur si l'objet passé n'est pas un GtkSpinButton
        g_print("Erreur : objet invalide passé\n");
    }
}//FIN DE LA FONCTION

/****************************************************************************
Fonction: creer_button_RadioPere : Créer un bouton radio
Entree: Interface *I : la structure de l'interface
Sortie: rien
*/
void creer_button_RadioPere(Interface *I)
{
    /// Créer un nouveau bouton radio avec le label spécifié dans I->B1->titre
    I->B1->buton = gtk_radio_button_new_with_label(NULL, I->B1->titre);
    /// Rendre le bouton radio sensible (activé)
    gtk_widget_set_sensitive(I->B1->buton, TRUE);
    /// Définir l'état actif du bouton radio en fonction de I->B1->active
    gtk_toggle_button_set_active(GTK_RADIO_BUTTON(I->B1->buton), I->B1->active);
    /** Connecter le signal "toggled" du bouton radio à la
    fonction de rappel radio_button_toggled ***/
    g_signal_connect(I->B1->buton, "toggled", G_CALLBACK(radio_button_toggled), I);
}//FIN DE LA FONCTION

```



```

*****+
void creer_button_radiofils(Interface *I, GtkWidget *pere)
{
    /*** Créer un nouveau bouton radio qui appartient
    au même groupe que le bouton parent ***/
    I->B2->buton =
    gtk_radio_button_new_with_label_from_widget(GTK_RADIO_BUTTON(pere), I->B2->titre);
    /// Rendre le bouton radio enfant sensible (activé)
    gtk_widget_set_sensitive(I->B2->buton, TRUE);
    /// Activer le bouton radio enfant en fonction de I->B2->active
    gtk_toggle_button_set_active(GTK_RADIO_BUTTON(I->B2->buton), I->B2->active);
    /*** Connecter le signal "toggled" du bouton radio enfant à
    la fonction de rappel radio_button_toggled***/
    g_signal_connect(I->B2->buton, "toggled", G_CALLBACK(radio_button_toggled), I);
} //FIN DE LA FONCTION.

*****+
Fonction: init_box : Initialise une structure Box avec une orientation spécifiée
Entree: Box* box : pointeur vers la structure Box
        char opt : option pour l'orientation ('0' pour horizontal, autre pour vertical)
Sortie: pointeur vers la structure Box initialisée
*****+
Box* init_box(Box* box, char opt)
{
    /// Allocation mémoire pour la structure Box
    box = Talloc(Box);
    /// Vérifier si l'allocation a échoué
    TesterAllocation(box);
    /// Définir l'orientation du box en fonction de l'option 'opt'
    if (opt == '0')
    {
        /// Si l'option est '0', définir l'orientation comme "horizontal"
        box->orient = g_strdup("horizontal");
    }
    else
    {
        /// Sinon, définir l'orientation comme "vertical"
        box->orient = g_strdup("vertical");
    }
    /// Retourner le pointeur vers la structure Box initialisée
    return box;
} //FIN DE LA FONCTION

*****+
Fonction: creer_Box : Crée une boîte GTK en fonction de l'orientation spécifiée
Entree: Box *box : pointeur vers la structure Box
Sortie: rien
*****+
void creer_Box(Box *box)
{
    /// Vérifier si l'orientation de la boîte est "vertical"
    if (strcmp(box->orient, "vertical") == 0)
    {
        /// Créer une nouvelle boîte GTK avec une orientation verticale
        box->box = gtk_box_new(GTK_ORIENTATION_VERTICAL, 1);
    }
    /// Vérifier si l'orientation de la boîte est "horizontal"
    else if (strcmp(box->orient, "horizontal") == 0)
    {
        /// Créer une nouvelle boîte GTK avec une orientation horizontale
        box->box = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 1);
    }
} //FIN DE LA FONCTION.

```

```

/*
*****
Fonction: gboolean on_draw : Fonction de rappel pour dessiner sur un widget
Entrée: GtkWidget *widget : le widget sur lequel dessiner
        cairo_t *cr : le contexte de dessin Cairo
        gpointer user_data : données utilisateur
Sortie: retourne TRUE si le dessin est réussi
*****
*/
gboolean on_draw(GtkWidget *widget, cairo_t *cr, gpointer user_data)
{
    /// Convertir les données utilisateur en structure Interface
    Interface *interface = (Interface *)user_data;
    /// Variables pour l'image et son échelle
    GdkPixbuf *pixbuf;
    /// Dessiner l'image de fond de l'arène
    interface->image= gdk_pixbuf_new_from_file("bac.jpg", NULL);
    /// Redimensionner l'image à la taille spécifiée dans la structure Interface
    interface->scaled= gdk_pixbuf_scale_simple(interface->image, interface->ARENA_WIDTH,
                                                interface->ARENA_HEIGHT, GDK_INTERP_BILINEAR);
    /// Utiliser l'image redimensionnée comme source pour Cairo
    gdk_cairo_set_source_pixbuf(cr, GDK_PIXBUF(interface->scaled), 0, 0);
    /// Appliquer l'image à toute la zone du widget et remplir
    cairo_paint(cr);
    /// Libérer la mémoire utilisée par l'image originale
    g_object_unref(GDK_PIXBUF(interface->image));
    /// Libérer la mémoire utilisée par l'image redimensionnée
    g_object_unref(GDK_PIXBUF(interface->scaled));

    /// Dessiner les objets dans l'arène
    for (int i = 0; i < 3; i++)
    {
        gdk_cairo_set_source_pixbuf(cr, Tab[i]->scaled, Tab[i]->position->X, Tab[i]->position-
        >Y);
        cairo_paint(cr);
    }

    /// Dessiner les gladiateurs avec animation
    for (GList *iter = gladiators; iter != NULL; iter = g_list_next(iter))
    {
        Gladiateur *gladiator = (Gladiateur *)iter->data;
        ///si le gladiateur n'est pas caché
        if(!gladiator->hidden)
        {
            ///si l'interface est en pause
            if(interface->is_paused)
                gladiator->current_frame=0;
            ///si le gladiateur est en marche
            if(gladiator->isWalking)
            {
                ///si le gladiateur se déplace à droite
                if(gladiator->isMovingRight)
                    pixbuf = TabFr[gladiator->nuGlad]->FramesR[gladiator->current_frame];
                else ///si le gladiateur se déplace à gauche
                    pixbuf = TabFr[gladiator->nuGlad]->FramesL[gladiator->current_frame];
            }
            else
                ///Chargement des frames de combat
                if(gladiator->isMovingRight) //si le gladiateur combat à droite
                    pixbuf = TabFr[gladiator->nuGlad]->FramesAR[gladiator->current_frame];
                else //si le gladiateur combat à gauche
                    pixbuf = TabFr[gladiator->nuGlad]->FramesAL[gladiator->current_frame];
            ///gerer la position du gladiateur
            gdk_cairo_set_source_pixbuf(cr, pixbuf, gladiator->Pos->X, gladiator->Pos->Y);
            cairo_paint(cr);
            ///gerer le niveau de vie
            if((gladiator->nvVie==0) && (gladiator->nvVie<200))
                pixbuf=Vie[0];///0-19%
            else
                if((gladiator->nvVie==200) && (gladiator->nvVie<400))
                    pixbuf=Vie[1];///20-39%
                else
                    if((gladiator->nvVie>=400) && (gladiator->nvVie<600))
                        pixbuf=Vie[2];///40-59%
                    else
                        if((gladiator->nvVie>=600) && (gladiator->nvVie<800))
                            pixbuf=Vie[3];///60-79%
                        else
                            if((gladiator->nvVie>=800) && (gladiator->nvVie<1000))
                                pixbuf=Vie[4];///80-99%
                            else
                                pixbuf=Vie[5];///100%
            ///Dessiner le niveau de vie
            gdk_cairo_set_source_pixbuf(cr, pixbuf, gladiator->Pos->X, gladiator->Pos->Y);
            cairo_paint(cr);
            ///Incrémente le frame actuel du gladiateur pour l'animation
            gladiator->current_frame = (gladiator->current_frame + 1) % NUM_FRAMES;
        }
        else//si le gladiateur est caché
        {
            ///Le cacher durant 15 Mis à jour
            if(gladiator->nhid>15)
                gladiator->nhid++;
            else
            {
                ///faire sortir le gladiateur de sa cachette
                gladiator->current_frame=0;
                gladiator->hidden=FALSE;
                gladiator->nhid=0;///remettre le compteur à 0 pour la duree de sa cachette
                switch(gladiator->hidway)
                {
                    ///si la cachette est la maison
                    case 0: ///il sort de la maison avec de nouvelles coordonnées
                        gladiator->Pos->X=Tab[0]->position->X+100;///abscisse
                        gladiator->Pos->Y=Tab[0]->position->Y+60;///ordonnées
                        gladiator->Vitesse->X = -gladiator->Vitesse->X;///la vitesse change
                        /// Changer l'animation
                        gladiator->isMovingRight = !gladiator->isMovingRight;
                        break;
                    ///si la cachette est l'arbre
                    case 1: ///il quitte l'arbre avec de nouvelles coordonnées
                        gladiator->Pos->X=Tab[1]->position->X+100;///abscisse
                        gladiator->Pos->Y=Tab[1]->position->Y+100;///ordonnées
                        break;
                    ///si la cachette est le rocher
                    default: ///il quitte le rocher avec de nouvelles coordonnées
                        gladiator->Pos->X=Tab[2]->position->X+100;///abscisse
                        gladiator->Pos->Y=Tab[2]->position->Y-100;///ordonnées
                        gladiator->Vitesse->X = -gladiator->Vitesse->X;
                        /// Changer l'animation
                        gladiator->isMovingRight = !gladiator->isMovingRight;
                }
            }
        }
    }
    return FALSE;
} //FIN DE LA FONCTION.

```

```

*****  

Fonction: void creer_button : creer un bouton  

Entree: Interface *I : structure de l'interface  

Sortie: rien  

*****  

void creer_button(Interface *I)  

{  

    GtkWidget *img;///le widget image  

    switch(I->B1->type)  

    {  

        ///Cas d'un bouton normal  

        case '1':  

            I->B1->button=gtk_button_new_with_label(I->B1->titre);  

            break;  

        ///Cas d'un check_box  

        case '3':  

        {  

            I->B1->button=gtk_check_button_new_with_label(I->B1->titre);  

            ///Activez le bouton  

            gtk_toggle_button_set_active(GTK_CHECK_BUTTON(I->B1->button),I->B1->active);  

        }  

        break;  

        ///Cas d'un bouton spin  

        case '4':  

        {  

            ///(valeur minimale,Valeur Maximale,ecart)  

            I->B1->button=gtk_spin_button_new_with_range(0,100,1);  

            g_signal_connect(I->B1->button, "value-changed",  

                            G_CALLBACK(Value_SpineButton), I);  

        }  

        break;  

    }  

} //FIN DE LA FONCTION  

*****  

Fonction: on_mouse_click:Fonction pour gerer les clics de la souris  

Entrée:GtkWidget *widget:l'élément qui resposable du signal  

        GdkEventButton *event:la position de l'évènement où le signal est émis  

        gpointer user_data:Données passées en paramètre dans la fonction signal  

Sortie: la fonction retourne TRUE à la fin du traitement  

*****  

gboolean on_mouse_click(GtkWidget *widget, GdkEventButton *event, gpointer user_data)  

{  

    if (event->button == GDK_BUTTON_PRIMARY)  

    { // Botton gauche de la souris  

        /// Parcourir la liste des gladiateurs pour vérifier si un gladiateur a été cliqué  

        for (GLList *iter = gladiators; iter != NULL; iter = g_list_next(iter))  

        {  

            ///Recuperer la structure du gladiateur  

            Gladiateur *gladiateur = (Gladiateur*)iter->data;  

            ///Vérifier si le clic a été effectué tous proche du gladiateur  

            if( (event->x >= gladiateur->Pos->X) && (event->x <= gladiateur->Pos->X + 70) &&  

                (event->y >= gladiateur->Pos->Y) && (event->y <= gladiateur->Pos->Y + 70)  

            )  

            {  

                /**Si on a clic proche d'un gladiateur  

                 Si aucun gladiateur n'est pas sélectionné*/  

                if (!cont_gladiator)  

                {  

                    /// attribuer ce gladiateur à l'utilisateur pour le contrôler  

                    cont_gladiator = gladiateur;  

                    gladiateur->isControlled= TRUE;  

                    return TRUE;  

                }  

                else  

                {  

                    /// Si un gladiateur est déjà sélectionné, le désélectionner  

                    cont_gladiator->isControlled= FALSE;  

                    cont_gladiator = NULL;  

                    return TRUE;  

                }  

            }  

        }  

    }  

    return TRUE;  

} //FIN DE LA FONCTION.

```

```

● ● ●

*****
Fonction:on_mouse_motion:Fonction pour gérer les evenements de mouvement de la souris
Entrée:GtkWidget *widget:l'élément qui responsable du signal
GdkEventMotion *event:la position de l'évènement où le signal est émis
gpointer user_data:Données passées en paramètre dans la fonction signal
Sortie: la fonction retourne TRUE à la fin du traitement
*****
gboolean on_mouse_motion(GtkWidget *widget, GdkEventMotion *event, gpointer user_data)
{
    ///Recuperer les données de l'interface
    Interface *I=(Interface*)user_data;
    ///Si on a un gladiateur qui est contrôlé
    if (cont_gladiator)
    {
        ///Faire suivre le gladiateur avec les mouvements du curseur
        cont_gladiator->Pos->x = event->x - 70;
        cont_gladiator->Pos->y = event->y - 70;
        /// Limiter le déplacement du gladiateur à l'intérieur de l'arène
        if (cont_gladiator->Pos->x < 0) cont_gladiator->Pos->x = 0;
        if (cont_gladiator->Pos->x > I->ARENA_WIDTH - 150) cont_gladiator->Pos->x = I->ARENA_WIDTH - 150;
        if (cont_gladiator->Pos->y < 0) cont_gladiator->Pos->y = 0;
        if (cont_gladiator->Pos->y > I->ARENA_HEIGHT - 150) cont_gladiator->Pos->y = I->ARENA_HEIGHT - 150;
        /// Redessiner l'arène pour afficher les nouveaux emplacements des gladiateurs
        gtk_widget_queue_draw(widget);
    }
    return TRUE;
}

*****
Fonction:calculate_Exp:Fonction pour calculer l'experience gagnée par le gladiateur
Entrée:int nbcombat:Représente le nombre de combat gagné par le gladiateur
Sortie: la fonction retourne la valeur de l'expéience en entier
*****
int calculate_Exp(int nbcombat)
{
    return (int)nbcombat*5;
}

*****
Fonction:Calculate_Protect:Fonction pour calculer la protection du gladiateur
Entrée:char armure:type de l'armure
gboolean bouclier:la présence ou l'absence du bouclier
Sortie:la fonction retourne la valeur de la protection en entier
*****
int Calculate_Protect(char armure,gboolean bouclier)
{
    switch(armure)
    {
        case 'F' :return bouclier? 4:5; ///Armure en fer
        case 'B' :return bouclier? 6:7; ///Armure en Bronze
        case 'O': return bouclier? 8:9; ///Armure en Or
    }
}

*****
Fonction:calculate_damage:Fonction pour calculer les domages reçus par un gladiateur
Entrée:int force:Force du gladiateur
int nvExp:Experience du gladiateur
char arme:type de l'arme
Sortie:la fonction retourne la valeur de domage en entier
*****
int calculate_damage(int force, int nvExp, char arme)
{
    float a,b;///Variables de calcul
    b=(float)(force+nvExp)/100;///ratio de la force et de l'experience

    switch (arme)
    {
        case 'S':/// Si l'arme est un Sabre
            a=(float)17 / 20;///ratio du sabre
            a=(a+b)*10;
            return (int)a;
        case 'H':/// Si l'arme est une Hache
            a=(float)15 / 20;///ratio de la hache
            a=(a+b)*10;
            return (int)a;
        case 'C':/// Si l'arme est une Chaîne
            a=(float)14 / 20;///ratio de la chaîne
            a=(a+b)*10;
            return (int)a;
        case 'M':/// Si l'arme est un Marteau
            a=(float)18 / 20;///ratio du Marteau
            a=(a+b)*10;
            return (int)a;
        case 'D':/// Si l'arme est une Dague
            a=(float)12 / 20;///ratio d'une dague
            a=(a+b)*10;
            return (int)a;
        default:
            return 0;
    }
}

} //FIN DE LA FONCTION calculate_damage()

```

```

● ● ●

/****************************************************************************
Fonction:Mise_enCombat:Fonction pour actualisé l'état d'un gladiateur
Entrée:Gladiateur *G:gladiateur
Sortie:la fonction met à jour le nouvel état du gladiateur
****************************************************************************/
void Mise_enCombat(Gladiateur *G)
{
    G->isfighting=TRUE;///Montrer qu'il est en combat
    G->IsWalking=FALSE;///Monter qu'il n'effectue plus de changement de position
    G->current_frame=0;///Remettre ses frames à Zero
    G->IsAttaking=TRUE;///Mettre sa position en attaque pour l'utilisation des frames
    //Annuler sa vitesse
    G->Vitesse->X=0.0;
    G->Vitesse->Y=0.0;
}

/****************************************************************************
Fonction:MINI:Fonction qui retourne le minimum entre 2 valeurs
Entrée:int x : valeur de comparaison
        int y :valeur de comparaison
Sortie:la fonction retourne le plus petit
****************************************************************************/
int MINI(int x,int y)
{
    return(x<y? x:y);
}

/****************************************************************************
Fonction:Arret_Combat:Fonction pour actualisé l'état d'un gladiateur
Entrée:Gladiateur *G:gladiateur
Sortie:la fonction met à jour le nouvel état du gladiateur
****************************************************************************/
void Arret_Combat(Gladiateur *G)
{
    G->isfighting=FALSE;///Arreter le combat
    G->IsWalking=TRUE;///Le remettre en marche s'il n'est pas mort
}

/****************************************************************************
Fonction:Vainqueur:Fonction pour actualisé l'état d'un gladiateur après une victoire
Entrée:Gladiateur *G:gladiateur
Sortie:la fonction met à jour le nouvel état du gladiateur
****************************************************************************/
void Vainqueur(Gladiateur *G1,Gladiateur *G2)
{
    G1->nbCgagne++;///Augmenter son nombre de combat gagner
    G1->nVie = 1000;///Remettre son niveau de vie
    G1->nvExp=calculate_Exp(G1->nbCgagne);///mettre à jour son experience
    G1->nvforce = MINI(G1->nvforce + (G2->nvforce / 2), 100);///Augmenter sa force
}

/****************************************************************************
Fonction:CombatGlad:Fonction pour le clacul des changements pendant le combat
Entrée:Gladiateur *G1:1er Gladiateur
        Gladiateur *G2:2e Gladiateur
        Interface *I :Interface de combat
Sortie:la fonction effectue les calculs des domages et reranche continuellement
        le niveau de vie des gladiateurs
****************************************************************************/
boolean CombatGlad(Gladiateur *G1, Gladiateur *G2, Interface *I)
{

    ///Calcul de la protection des gladiateurs
    int protect1 = Calculate_Protect(G1->Armure, G1->bouclier);
    int protect2 = Calculate_Protect(G2->Armure, G2->bouclier);

    ///Calcul des domages que les gladiateurs peuvent infliger
    int damage1 = calculate_damage(G1->nvforce, G1->nvExp, G1->numArme);
    int damage2 = calculate_damage(G2->nvforce, G2->nvExp, G2->numArme);

    ///Difference entre les domages et la protection
    damage2-=protect1;
    damage1-=protect2;

    ///Les positionner de manière opposé
    G1->isMovingRight = TRUE;
    G2->isMovingRight = FALSE;

    ///Si les gladiateurs sont toujours vivants
    if( (G1->nVie > 0) && (G2->nVie > 0) )
    {
        ///Soustraire les domages
        G1->nVie -=damage2;
        G2->nVie -=damage1;
        if(G1->nVie > 0 && G2->nVie > 0)
            return TRUE; /// Continue the combat
    }

    ///Si l'un est mort
    if (G1->nVie > 0)
        Vainqueur(G1,G2);
    else
        Vainqueur(G2,G1);

    ///Arreter le combat
    Arret_Combat(G1);
    Arret_Combat(G2);

    return FALSE; /// End the comba
}///FIN DE LA FONCTION.

```



```

/*
*****
Fonction:update_positions:Fonction de mise a jour des positions des gladiateurs
Entrée:Interface *I :Interface de combat
Sortie:la fonction effectue les calculs du positionnement des gladiateurs
*****
gboolean update_positions(Interface *I)
{
    /*Si l'interface est en pause ne pas
    changer la position des gladiateurs*/
    if(I->is_paused)
        return G_SOURCE_CONTINUE;

    /// Parcourir la liste des gladiateurs et mettre a jour leurs positions
    for (GList *iter1 = gladiators; iter1 != NULL; iter1 = g_list_next(iter1))
    {
        Gladateur *gladiator1 = (Gladateur *)iter1->data;
        /// Si le gladiateur est contrôlé par l'utilisateur, ne pas le déplacer automatiquement
        if( (!gladiator1->isControlled) &&
            (!gladiator1->isfighting) &&
            (!gladiator1->hidden)
        )
        {
            /// Générer de nouveaux déplacements aléatoires
            /// Ajustement aléatoire de la vitesse horizontale
            gladiator1->Vitesse->X += g_random_double_range(-0.1, 0.1);
            /// Ajustement aléatoire de la vitesse verticale
            gladiator1->Vitesse->Y += g_random_double_range(-0.1, 0.1);
            /// Limiter la vitesse maximale
            if( gladiator1->Vitesse->X > 3.0 ) gladiator1->Vitesse->X = 3.0;
            if( gladiator1->Vitesse->Y > 3.0 ) gladiator1->Vitesse->Y = 3.0;
            /// Limiter la vitesse minimale
            if( gladiator1->Vitesse->X < -3.0 ) gladiator1->Vitesse->X = -3.0;
            if( gladiator1->Vitesse->Y < -3.0 ) gladiator1->Vitesse->Y = -3.0;

            /// Appliquer les déplacements
            gladiator1->Pos->X += gladiator1->Vitesse->X;
            gladiator1->Pos->Y += gladiator1->Vitesse->Y;

            /*Verifier si le gladiateur atteint
            les bords de l'arène et inverser sa direction si nécessaire*/
            if( (gladiator1->Pos->X < 0 ) ||
                (gladiator1->Pos->X > I->ARENA_WIDTH - 200)
            )
            {
                gladiator1->Vitesse->X = -gladiator1->Vitesse->X;
                /// Changer l'animation
                gladiator1->isMovingRight = !gladiator1->isMovingRight;
            }

            if( (gladiator1->Pos->Y < 0 ) ||
                (gladiator1->Pos->Y > I->ARENA_HEIGHT - 200)
            )
            gladiator1->Vitesse->Y = -gladiator1->Vitesse->Y;
        }
    }

    return G_SOURCE_CONTINUE;
}

```

```

● ● ●

*****Fonction:CreeFrame:Fonction qui les frames des gladiateurs
Entrée:char ch1[30]:path des images
        char ch2[30]:path des images
        int T:Taille de l'image
Sortie:la fonction charge les images et les crée
*****Fram *CreeFrame(char ch1[30],char ch2[30],int T)
{
    //Variables
    GdkPixbuf *image;
    Fram *F=talloc(Fram);
    TesterAllocation(F);
    char filename[30];
    int i ;

    //Recuperation des frames du deplacement droit
    for(i=0;i<12;i++)
    {
        sprintf(filename, "%s%dright.png",ch1,i);
        image= gdk_pixbuf_new_from_file(filename, NULL);
        /// Redimensionner l'image de l'objet à la taille souhaitée
        F->FramesR[i]= gdk_pixbuf_scale_simple(image,T, T, GDK_INTERP_BILINEAR);
    }
    //Recuperation des frames du deplacement gauche
    for(i=0;i<12;i++)
    {
        sprintf(filename, "%s%dleft.png",ch1,i);
        image= gdk_pixbuf_new_from_file(filename, NULL);
        /// Redimensionner l'image de l'objet à la taille souhaitée
        F->FramesL[i]= gdk_pixbuf_scale_simple(image,T, T, GDK_INTERP_BILINEAR);
    }
    //Recuperation des frames des attaques droit
    for(i=0;i<12;i++)
    {
        sprintf(filename, "%s%dright.png",ch2,i);
        image= gdk_pixbuf_new_from_file(filename, NULL);
        /// Redimensionner l'image de l'objet à la taille souhaitée
        F->FramesAR[i]= gdk_pixbuf_scale_simple(image,T, T, GDK_INTERP_BILINEAR);
    }
    //Recuperation des frames des attaques gauche
    for(i=0;i<12;i++)
    {
        sprintf(filename, "%s%dleft.png",ch2,i);
        image= gdk_pixbuf_new_from_file(filename, NULL);
        /// Redimensionner l'image de l'objet à la taille souhaitée
        F->FramesAL[i]= gdk_pixbuf_scale_simple(image,T, T, GDK_INTERP_BILINEAR);
    }
    return(Fram*)F;
}//FIN

*****Fonction:initFrameGlad:Fonction qui charge les frames des gladiateurs
Entrée:void
Sortie:la fonction charge les paths pour la création des frames
*****void initFrameGlad()
{
    //Variables
    GdkPixbuf *image;
    int i ;
    char filename[30];
    //Chargement des différentes paths
    TabFr[0]=CreeFrame("G0/run/run_","G0/Attack/at_",100);
    TabFr[1]=CreeFrame("G1/run/run_","G1/Attack/at_",100);
    TabFr[2]=CreeFrame("G2/run/run_","G2/Attack/at_",170);
    TabFr[3]=CreeFrame("G3/run/run_","G3/Attack/at_",170);
    TabFr[4]=CreeFrame("G4/run/run_","G4/Attack/at_",170);
    TabFr[5]=CreeFrame("G5/run/run_","G5/Attack/at_",180);
    TabFr[6]=CreeFrame("G6/run/run_","G6/Attack/at_",180);
    TabFr[7]=CreeFrame("G7/run/run_","G7/Attack/at_",140);
    TabFr[8]=CreeFrame("G8/run/run_","G8/Attack/at_",140);
    TabFr[9]=CreeFrame("G9/run/run_","G9/Attack/at_",140);
    TabFr[10]=CreeFrame("G10/run/run_","G10/Attack/at_",140);
    TabFr[11]=CreeFrame("G11/run/run_","G11/Attack/at_",140);
    TabFr[12]=CreeFrame("G12/run/run_","G12/Attack/at_",140);

    //Creation du niveau de vie
    for(i=0;i<6;i++)
    {
        sprintf(filename, "scorevie/life%d.png",i);
        image= gdk_pixbuf_new_from_file(filename, NULL);
        /// Redimensionner l'image de l'objet à la taille souhaitée
        Vie[i]= gdk_pixbuf_scale_simple(image,70, 30, GDK_INTERP_BILINEAR);
    }
}//FIN DE LA FONCTION;

```



```

/*****************************  

Fonction:CreateObj:Fonction qui crée les frames des Objets de l'interface  

Entrée:char ch[20]:path de l'objet  

    double x:position à l'horizontale de l'objet  

    double y:Position à la verticale de l'objet  

Sortie:la fonction pour crée l'objet avec ses informations  

*****  

Objet *CreateObj(char ch[20],double x,double y)  

{  

    Objet *O=Talloc(Objet);///ALLOCATION  

    TesterAllocation(O);///Tester l'allocation  

    O->position=Talloc(Dep);  

    TesterAllocation(O->position);  

    ///Positionnement  

    O->position->X=x;  

    O->position->Y=y;  

    O->image= gdk_pixbuf_new_from_file(ch, NULL);  

    // Redimensionner l'image de l'objet à la taille souhaitée  

    O->scaled= gdk_pixbuf_scale_simple(O->image,100, 150, GDK_INTERP_BILINEAR);  

    return(Objet*)O;  

}  

/*****************************  

Fonction:InitObject:Fonction qui charge les frames des Objets  

Entrée:void  

Sortie:la fonction charge les paths pour la création des frames  

*****  

void InitObject()  

{  

    ///Chargement des paths  

    Tab[0]=CreateObj("maison.png",40,40);  

    Tab[1]=CreateObj("arbre.png",150,300);  

    Tab[2]=CreateObj("roche.png",400,600);  

}  

/*****************************  

Fonction:defaultValueGlad:Fonction qui definit des valeurs  

    par defaut pour un gladiateur  

Entrée:Interface *I:Interface  

Sortie:void  

*****  

void defaultValueGlad(Interface *I)  

{  

    I->G->Agressif=TRUE;///Agressif par defaut  

    I->G->nvVie=1000;///Niveau de vie plein  

    I->G->nvforce=50;///Force à 50%  

    I->G->numArme='C';///Arme par defaut une chaîne  

    I->G->bouclier=FALSE;///Ne dispose pas de bouclier  

    I->G->nbCgagne=0;///Initialiser son nombre de combat  

    I->G->nvExp=0;///Initialiser son experience  

    I->G->iswalking=TRUE;///En marche au debut  

    I->G->isfighting=FALSE;///Ne combat pas  

    I->G->Armure='B';///Poss-de une armure en Bronze  

    I->choix=0;///Numero du gladiateur est Zero  

    I->G->hidden=FALSE;///le gladiateur n'est pas caché  

    I->G->nbhid=0;///Temps pour se cacher initialiser  

    I->G->nuGlad=0;///Numero du gladiateur est Zero  

    /// Creer un nouveau gladiateur avec des coordonnees et une vitesse aleatoire  

    I->G->Pos->X= 810;///Point de sortie  

    I->G->Pos->Y = 180;///Point de sortie  

    I->G->Vitesse->X = g_random_double_range(1.0, 3.0);  

    I->G->Vitesse->Y= g_random_double_range(1.0, 3.0);  

    I->G->isControlled= FALSE;///Le gladiateur n'est pas contrôlé  

    I->G->current_frame=0;///Initialiser ses frames de deplacement  

    I->G->isMovingRight=TRUE;///le gladiateur se déplace vers droite  

}///FIN DE LA FONCTION.

```

```

/*
*****
Fonction:initializeGlad:Fonction pour l'allocation de la memoire pour un gladiateur
Entrée:Interface *interface:Interface
Sortie:la fonction retourne l'interface avec un gladiateur et des valeurs par defaut
*****
Interface *initializeGlad(Interface *interface)
{

    //ALLOCATION ET TEST DES ALLOCATIONS
    interface->G=Talloc(Gladiateur);///la structure du gladiateur
    TesterAllocation(interface->G);

    interface->G->Pos=Talloc(Dep);///Les deplacement
    TesterAllocation(interface->G->Pos);

    interface->G->Vitesse=Talloc(Dep); ///La vitesse
    TesterAllocation(interface->G->Vitesse);

    defaultValueGlad(interface);///Initialisation des valeurs par defaut

    return(Interface*)interface;
}

/*
*****
Fonction:creeGlad:Fonction pour remplie les informations du gladiateurs avec des
valeurs defini
Entrée:Interface *I:L'interface
    int force:Force du gladiateur
    char numAr:Type de l'arme
    gboolean Agressif:Agressivité du gladiiteur
    char type_Armure>Type de l'armure
Sortie:void
*****
void creeGlad(Interface *I ,int force,char numAr,gboolean Agressif,char type_Armure)
{
    Gladiateur *G=I->G;
    //Affection des valeurs predefinis
    G->Armure=type_Armure;///Armure
    G->bouclier=TRUE;///la possession de bouclier
    G->Agressif=Agressif;///Agressilité
    G->nvforce=force;///Force
    G->numArme=numAr;///Type de l'arme
    G->nbcGagne=0;///Nombre de combat initialisé
    G->isControlled=FALSE;///Le gladiateur n'est pas controlé
    G->nvVie=1000;///Le niveau de vie
    G->hidden=FALSE;///Le gladiateur n'est pas caché
    G->nbid=0;///Temps pour se cacher renitialiser
    G->nvExp=0; ///Initialisation du niveau d'experience
    G->isfighting=FALSE;///Ne combat pas pour le momment
    G->iswalking=TRUE;///Marche au debut
    G->isMovingRight=TRUE;///Se deplace vers la droite

}

/*
*****
Fonction:InitGladi:Fonction pour crée des gladiateurs avec des valeurs predefinis
Entrée:Interface *I:L'interface
Sortie:void
*****
void InitGladi(Interface *I)
{
    //En fonction du choix de l'utilisateur crée le gladiateur avec ses valeurs
    switch(I->choix)
    {
        case 1:creeGlad(I,80,'S',TRUE,'B');///1er gladiateur
        break;
        case 2:creeGlad(I,50,'M',TRUE,'F');///2e gladiateur
        break;
        case 3: creeGlad(I,70,'C',TRUE,'O');///3e gladiateur
        break;
        case 4:creeGlad(I,50,'D',TRUE,'B');///4e gladiateur
        break;
        case 5:creeGlad(I,50,'H',TRUE,'B');///5e gladiateur
        break;
        case 6:creeGlad(I,60,'S',TRUE,'B');///6e gladiateur
        break;
        case 7:creeGlad(I,80,'C',FALSE,'O');///7e gladiateur
        break;
        case 8:creeGlad(I,60,'C',FALSE,'O');///8e gladiateur
        break;
        case 9:creeGlad(I,80,'H',FALSE,'B');///9e gladiateur
        break;
        case 10:creeGlad(I,50,'M',FALSE,'F');///10e gladiateur
        break;
        case 11:creeGlad(I,70,'H',FALSE,'F');///11e gladiateur
        break;
        default :creeGlad(I,70,'S',FALSE,'F');///12 gladiateur
    }
}

//FIN DE LA FONCTION.
}

```

```

● ● ●

/****************************************************************************
Fonction:ChoixGlad:Fonction qui récupère les données à chaque clique sur le choix
d'un gladiateur
Entrée:GtkWidget *w:l'élément qui émet le signal
gpointer *data: la donnée passée en paramètre
Sortie:void
****************************************************************************/
void ChoixGlad(GtkWidget *w,gpointer *data)
{
    Interface *I=(Interface*)data;
    ///Recuperer le choix
    int choix = GPOINTER_TO_INT(g_object_get_data(G_OBJECT(w), "choix"));
    I->choix = choix;
    I->G->nGlad=choix;
    InitGlad(I);
}

/****************************************************************************
Fonction:create_gladiator: Fonction pour créer un nouveau gladiateur dans l'arène
Entrée:GtkWidget *widget:l'élément qui émet le signal
gpointer user_data: la donnée passée en paramètre
Sortie:void
****************************************************************************/
void create_gladiator(GtkWidget *widget, gpointer user_data)
{
    Interface *I=(Interface*)user_data;

    ///Si la force du gladiateur est inférieure à 40
    if(I->G->nvforce<40)
        I->G->nvforce=40;
    /// Ajouter le nouveau gladiateur à la liste
    gladiators = g_list_append(gladiators, I->G);
    I=initializeGlad(I);

}

/****************************************************************************
Fonction:detect_collisions:Fonction pour détecter les collisions entre les gladiateurs
Entrée:Interface *interface:L'interface
Sortie:void
****************************************************************************/
void detect_collisions(Interface *interface)
{
    int i;
    /// Parcourir la liste des gladiateurs
    for (GList *iter1 = gladiators; iter1 != NULL; iter1 = g_list_next(iter1))
    {
        Gladiateur *gladiatori1 = (Gladiateur *)iter1->data;
        ///Position avec les objets
        ///Verifier si le gladiateur n'est pas proche des objets
        for(i=0;i<3;i++)
        if( (fabs(gladiatori1->Pos->X-Tab[i]->position->X)<50) &&
            (fabs(gladiatori1->Pos->Y-Tab[i]->position->Y)<50)
        )
        {   ///Si il est proche des objet et qu'il n'est pas caché, le caché
            if(!gladiatori1->hidden)
            {
                gladiatori1->hidway=i;///Lieu de cachette
                gladiatori1->hidden=TRUE;
            }
            break;
        }

        for (GList *iter2 = gladiators; iter2 != NULL; iter2 = g_list_next(iter2))
        {
            Gladiateur *gladiatori2 = (Gladiateur *)iter2->data;
            ///Verifier si les 2 gladiateurs n'occupent pas la même position
            if( (gladiatori1 != gladiatori2) &&
                (fabs(gladiatori1->Pos->X - gladiatori2->Pos->X) < 40 ) &&
                (fabs(gladiatori1->Pos->Y - gladiatori2->Pos->Y) < 40 ) &&
                (!gladiatori1->hidden) && (!gladiatori2->hidden)
            )
            ///Verifier si l'un d'eux est Agressif
            if( ( (gladiatori1->Agressif) || (gladiatori2->Agressif) ) &&
                ( (!gladiatori1->isfighting) && (!gladiatori2->isfighting) )
            )
            {
                ///Les préparer pour le combat
                Mise_enCombat(gladiatori1);
                Mise_enCombat(gladiatori2);
                interface->combat=TRUE;///Confirmer la présence d'un combat
                gladiatori1->Pos->Y=gladiatori2->Pos->Y;
            }

        }
    }
}

} //FIN DE LA FONCTION.

```



```

*****
Fonction:combat_animation:Fonction pour lancer le combat entre gladiateur
Entrée:Interface *interface:L'interface
Sortie:void
*****
void combat_animation(Interface *interface)
{

    /// Parcourir la liste des gladiateurs
    for (GList *iter1 = gladiators; iter1 != NULL; iter1 = g_list_next(iter1))
    {
        Gladiateur *gladiator1 = (Gladiateur *)iter1->data;
        for (GList *iter2 = gladiators; iter2 != NULL; iter2 = g_list_next(iter2))
        {
            Gladiateur *gladiator2 = (Gladiateur *)iter2->data;
            //Verifier si les 2 gladiateurs n'occupent pas la même position
            if( (gladiator1 != gladiator2) &&
                ( fabs(gladiator1->Pos->X - gladiator2->Pos->X) < 40 )&&
                ( fabs(gladiator1->Pos->Y - gladiator2->Pos->Y) < 40 ) &&
                (!gladiator1->hidden) && (!gladiator2->hidden)
            )
            {
                ///L'un des deux est Agressif declenché le combat
                if(( (gladiator1->Agressif) || (gladiator2->Agressif) ) &&
                    ( (gladiator1->isfighting) && (gladiator2->isfighting) )
                )
                {
                    /// Collision detecter
                    CombatGlad(gladiator1,gladiator2,interface);

                    /// Supprimer les gladiateurs morts de la liste
                    if (gladiator1->nVie <= 0)
                    {
                        interface->combat=FALSE;
                        gladiators = g_list_remove(gladiators, gladiator1);
                        free(gladiator1); ///libérer la mémoire
                    }
                    if (gladiator2->nVie <= 0)
                    {
                        interface->combat=FALSE;
                        gladiators = g_list_remove(gladiators, gladiator2);
                        free(gladiator2); /// libérer la mémoire
                    }
                }
            }
        }
    }
}

*****

Fonction:main_loop:Fonction principale pour l'animation de l'interface
Entrée:Interface *interface:L'interface
Sortie:TRUE
*****
gboolean main_loop(Interface *interface)
{
    /// Redessiner l'arène
    gtk_widget_queue_draw(interface->arena);

    /// Vérifier les collisions et déclencher les combats si nécessaire
    if(!interface->combat && !interface->is_paused)
        detect_collisions(interface);

    /// Gérer les animations de combat
    if (interface->combat)
        combat_animation(interface);

    /// Mettre à jour les positions des gladiateurs
    update_positions(interface);

    return G_SOURCE_CONTINUE;///TRUE
}//FIN DE LA FONCTION.

```

```

*****Fonction:CreeInterface:Fonction pour créer les élément visuels de l'interface
Entrée:Interface *interface:L'interface
Sortie:la fonction retourne l'interface et ses éléments
*****
Interface *CreeInterface(Interface *interface)
{

    char filename[20];
    GdkPixbuf *pixbuf;
    int row,col ;///Variables pour la representation des elements
    /// Création des boutons avec les images
    GtkWidget *button;
    GtkWidget *image_widget;
    ///Création des Conteneurs
    Mongrid *G1;
    GtkWidget *b1=gtk_box_new(GTK_ORIENTATION_VERTICAL,0);
    G1=Init_grid();
    gtk_box_set_spacing(interface->Boxe->box,10);
    gtk_box_pack_end(GTK_BOX(interface->Boxe->box), b1, FALSE,TRUE, 0);
    gtk_box_pack_start(GTK_BOX(b1), G1->grid, TRUE,TRUE, 0);

    ///Image de l'interface
    MyImage *Img=init_image("glad.png",90,100);
    creer_image(Img);
    Ajouter_elem_grid(G1,Img->image,0,0);

    ///Création des boutons de l'interface radio
    interface->lab=initLabel(interface->lab,"Choix du Comportement");
    CreerLabel(interface->lab);
    Ajouter_elem_grid(G1,interface->lab->label,0,1);

    interface->B1=init_bouton("Agressif",TRUE,'3');
    creer_button_RadioPere(interface);
    Ajouter_elem_grid(G1,interface->B1->buton,0,2);

    interface->B2=init_bouton("Gentil",FALSE,'3');
    creer_button_radiofils(interface,interface->B1->buton);
    Ajouter_elem_grid(G1,interface->B2->buton,1,2);

    ///Zone de choix de la force
    interface->lab=initLabel(interface->lab,"Force");
    CreerLabel(interface->lab);
    Ajouter_elem_grid(G1,interface->lab->label,0,3);

    interface->B3=init_bouton("",TRUE,'4');
    creer_button(interface);
    Ajouter_elem_grid(G1,interface->B3->buton,0,4);

    interface->lab=initLabel(interface->lab,"Choix du Gladiateur");
    CreerLabel(interface->lab);
    Ajouter_elem_grid(G1,interface->lab->label,0,5);

    /// Crédit d'un grid pour les gladiateurs
    Mongrid *grid=Init_grid();
    Ajouter_elem_grid(G1,grid->grid,0,6);

    /// Chargement des images pour les boutons
    for (int i = 1; i <= 12; i++)
    {
        sprintf(filename, "IconsCaracters/glad%d.png", i);
        pixbuf = gdk_pixbuf_new_from_file(filename, NULL);
        image_widget = gtk_image_new_from_pixbuf(pixbuf);
        button = gtk_button_new();
        gtk_button_set_image(GTK_BUTTON(button), image_widget);
        g_object_set_data(G_OBJECT(button), "choix", GINT_TO_POINTER(i));
        g_signal_connect(button, "clicked", G_CALLBACK(ChoixGlad), interface);

        row = (i - 1) / 3;
        col = (i - 1) % 3;
        Ajouter_elem_grid(grid,button,col,row);
    }

    ///bouton pour création des gladiateurs
    Mongrid *G2=Init_grid();
    gtk_box_pack_end(GTK_BOX(b1), G2->grid, TRUE,FALSE, 10);
    interface->B4=init_bouton("Creer Gladiateur",FALSE,'1');
    creer_button(interface);
    Ajouter_elem_grid(G2,interface->B4->buton,0,0);
    g_signal_connect(interface->B4->buton, "clicked", G_CALLBACK(create_gladiator), interface);

    return interface;
} //FIN DE LA FONCTION.

```

```

● ● ●

/*
*****
Fonction: parsefile : charger le fichier XML pour le traitement des
            donnees de l'interface
Entree: const char *filename : le chemin vers le fichier XML
        Interface *I : l'interface traitée
Sortie: rien
*****
void parsefile(const char *filename,Interface *I)
{
    XMLDocument doc;/// Déclaration d'une structure pour le document XML
    int i;/// Variable pour les boucles
    XMLNode *child;/// Pointeur vers un nœud enfant
    /// Chargement du fichier XML dans la structure doc
    if (XMLDOC_Loading(&doc, filename))
    {
        /// Récupération de la racine du document XML
        XMLNode *root = XMLNode_child(doc.root, 0);
        /// Vérifie si la balise racine est "arena"
        if ( !strcmp((char*)root->tag, "arena"))
        {
            if(xmlGetProp(root, (char*)"width"))///recuperer la largeur de l'arene
                I->ARENA_WIDTH= atoi((char*)xmlGetProp(root, (char*)"width"));
            if(xmlGetProp(root, (char*)"height"))///recuperer la hauteur de l'arene
                I->ARENA_HEIGHT= atoi((char*)xmlGetProp(root, (char*)"height"));
            if(xmlGetProp(root,(char*)"background"))///recuperer le background de l'arene
                strcpy(I->Arenabac,(char*) xmlGetProp(root,(char*)"background"));
            ///recuperer l'état du combat
            if(xmlGetProp(root, (char*)"Combat"))
            {
                if( !strcmp("TRUE",(char *)xmlGetProp(root,(char*)"Combat")) )
                    I->combat=TRUE;
                else
                    I->combat=FALSE;
            }

            if(xmlGetProp(root, (char*)"is_paused"))///si l'interface est en pause
            {
                if( !strcmp("TRUE", (char*)xmlGetProp(root,(char*)"is_paused")) )
                    I->is_paused=TRUE;
                else///si l'interface n'est pas en pause
                    I->is_paused=FALSE;
            }

            /// Parcourir tous les enfants de la racine pour traiter les gladiateurs
            for(i=0;i<root->children.size;i++)
            {
                child = XMLNode_child(root,i);
                if ( !strcmp((char*)child->tag, "gladiator"))
                {
                    /// Allocation mémoire d'un nouveau gladiateur
                    Gladateur *gladiator = Talloc(Gladateur);
                    TesterAllocation(gladiator);///tester l'allocation memoire
                    gladiator->Pos = Talloc(Dep);/// Allocation mémoire de la position
                    TesterAllocation(gladiator->Pos);///tester l'allocation memoire
                    gladiator->Vitesse = Talloc(Dep);/// Allocation mémoire de la vitesse
                    TesterAllocation(gladiator->Vitesse);///tester l'allocation memoire
                    /// Extraction et conversion des attributs du gladiateur
                    if(xmlGetProp(child, (char*)"PosX"))///recuperer la position en abscisse
                        gladiator->Pos->X = atof((char*) xmlGetProp(child, (char*)"PosX"));

                    if(xmlGetProp(child, "PosY"))///recuperer la position en ordonnée
                        gladiator->Pos->Y = atof((char*) xmlGetProp(child, (char*) "PosY"));

                    if(xmlGetProp(child, "VitesseX"))///recuperer la vitesse en abscisse
                        gladiator->Vitesse->X = atof((char*) xmlGetProp(child, (char*)"VitesseX"));

                    if(xmlGetProp(child, "VitesseY"))///recuperer la vitesse en ordonnée
                        gladiator->Vitesse->Y = atof((char*) xmlGetProp(child,(char*)"VitesseY"));

                    if (xmlGetProp(child, "nvVie"))///recuperer le niveau de vie
                        gladiator->nvVie = atoi((char *) xmlGetProp(child,(char*)"nvVie"));

                    if (xmlGetProp(child, "nvforce"))///recuperer le niveau de force
                        gladiator->nvforce = atoi((char *) xmlGetProp(child, (char*)"nvforce"));

                    if (xmlGetProp(child, "nvExp"))///recuperer le niveau d'experience
                        gladiator->nvExp = atoi((char *) xmlGetProp(child, (char*)"nvExp"));

                    if (xmlGetProp(child, "nbCgagne"))///recuperer le nombre de combat gagnés
                        gladiator->nbCgagne = atoi((char *) xmlGetProp(child, (char*)"nbCgagne"));
                }
            }
        }
    }
}

```

```

        if (xmlGetProp(child, "nuGlad"))///recuperer le numero du gladiateur
            gladiator->nuGlad = atoi((char *) xmlGetProp(child, (char*)"nuGlad"));

        if (xmlGetProp(child, "hidway"))///recuperer l'endroit où il est caché
            gladiator->hidway = atoi((char *) xmlGetProp(child,(char*)"hidway"));

        if (xmlGetProp(child, "current_frame"))///recuperer le frame courant
            gladiator->current_frame = atoi((char *) xmlGetProp(child,(char*) "current_frame"));

        if (xmlGetProp(child, "nbhid"))///duree a laquelle le gladiateur reste caché
            gladiator->nbhid = atoi((char *) xmlGetProp(child, (char*)"nbhid"));

        if (xmlGetProp(child, "numArme"))///le numero de l'arme qu'il possède
            gladiator->numArme=(char *)xmlGetProp(child,(char*)"numArme");

        if (xmlGetProp(child, "armure"))///l'armure qu'il porte
            gladiator->Armure=(char *) xmlGetProp(child, (char*) "armure");

        ///le controle du gladiateur par la souris
        if (xmlGetProp(child, (char*) "isControlled") )
        {
            ///s'il est contrôlé
            if (!strcmp("TRUE", (char *) xmlGetProp(child, (char*) "isControlled")))
                gladiator->isControlled = TRUE;
            else///s'il n'est pas contrôlé
                gladiator->isControlled = FALSE;
        }
        ///le mouvement du gladiateur
        if (xmlGetProp(child, (char*) "isMovingRight"))
        {
            ///s'il est en mouvement
            if (!strcmp("TRUE", (char *) xmlGetProp(child, (char*)"isMovingRight")))
                gladiator->isMovingRight = TRUE;
            else///s'il n'est pas en mouvement
                gladiator->isMovingRight = FALSE;
        }
        ///le combat du gladiateur
        if (xmlGetProp(child, (char*)"isfighting"))
        {
            ///s'il est en combat
            if (!strcmp("TRUE", (char *) xmlGetProp(child,(char*)"isfighting")))
                gladiator->isfighting = TRUE;
            else///s'il n'est pas en combat
                gladiator->isfighting = FALSE;
        }
        ///la marche du gladiateur
        if (xmlGetProp(child, (char*)"iswalking") )
        {
            ///s'il est en marche
            if (!strcmp("TRUE", (char *) xmlGetProp(child, (char*) "iswalking")))
                gladiator->iswalking = TRUE;
            else///s'il n'est pas en marche
                gladiator->iswalking = FALSE;
        }
        ///l'attaque du gladiateur
        if (xmlGetProp(child,(char*) "isattaking") )
        {
            ///s'il est attaqué
            if (!strcmp("TRUE", (char *) xmlGetProp(child,(char*) "isattaking")))
                gladiator->isattaking = TRUE;
            else///s'il n'est pas en attaque
                gladiator->isattaking = FALSE;
        }
        ///l'agressivité du gladiateur
        if (xmlGetProp(child, (char*)"Agressif") )
        {
            ///s'il est agressif
            if (!strcmp("TRUE", (char *) xmlGetProp(child,(char*) "Agressif")))
                gladiator->Agressif = TRUE;
            else///s'il n'est pas agressif
                gladiator->Agressif = FALSE;
        }
        ///le bouclier du gladiateur
        if (xmlGetProp(child,(char*) "bouclier") )
        {
            ///s'il a un bouclier
            if (!strcmp("TRUE", (char *) xmlGetProp(child, (char*) "bouclier")))
                gladiator->bouclier = TRUE;
            else///s'il n'a pas un bouclier
                gladiator->bouclier = FALSE;
        }

        if (xmlGetProp(child,(char*) "hidden") )
        {
            ///s'il est caché
            if (!strcmp("TRUE", (char *) xmlGetProp(child, (char*) "hidden")))
                gladiator->hidden = TRUE;
            else///s'il n'est pas caché
                gladiator->hidden = FALSE;
        }
        ///ajouter le gladiateur à la liste des gladiateurs
        gladiators = g_list_append(gladiators, gladiator);
    }
}

//si la racine n'est pas "arena" on affiche un message d'erreur
else g_print("arena not found");
XMLDocument_free(&doc);
}
else // si le chargement du fichier XML a échoué
g_warning("Failed to parse XML file %s", filename);
return ;
}//FIN DE LA FONCTION parsefile();

```



```

*****
Fonction:CreeInterface:Fonction d'initialisation de l'interface utilisateur
Entrée:Interface *interface:L'interface
Sortie:void
*****
void *initialize_interface(Interface *interface)
{
    /// Creer une fenetre principale
    interface=Talloc(Interface);
    TesterAllocation(interface);

    ///Cree la fénêtre de l'interface
    interface->F=init_fenetre("Jeu Gladiateur","img.png",1500,1500);
    interface->ARENA_WIDTH=1400;///Largeur de l'arène
    interface->ARENA_HEIGHT=920;///Hauteur de l'arène
    strcpy(interface->Arenabac,"bac.png");///Background de l'arène
    interface->is_paused=FALSE;///L'interface n'est pas en pause
    interface->combat=FALSE;///il n'y a pas de combat dans l'interface
    creer_fenetre(interface,TRUE);///Création de la fenêtre
    interface=initializeGlad(interface);

    /// Creer une boite pour les widgets
    interface->Boxe=init_box(interface->Boxe,'0');
    creer_Box(interface->Boxe);
    ///Ajouter la box a la fenetre
    gtk_container_add(GTK_CONTAINER(interface->F->window),interface->Boxe->box);

    ///Création de la Zône de dessin
    interface->arena= gtk_drawing_area_new();
    gtk_box_pack_start(GTK_BOX(interface->Boxe->box), interface->arena, TRUE, TRUE, 0);

    ///Dessiner l'interface
    g_signal_connect(interface->arena, "draw", G_CALLBACK(on_draw), interface);

    /// Connecter les signaux pour les évènements de la souris
    gtk_widget_add_events(interface->arena, GDK_BUTTON_PRESS_MASK | GDK_POINTER_MOTION_MASK);
    g_signal_connect(interface->arena, "motion-notify-event", G_CALLBACK(on_mouse_motion),
    interface);
    g_signal_connect(interface->arena, "button-press-event", G_CALLBACK(on_mouse_click),
    interface);

    /// Demarrer le timer pour mettre a jour les positions des gladiateurs a des intervalles
    reguliers
    g_timeout_add(90, (GSourceFunc)main_loop,interface);

    interface=CreeInterface(interface);
}

*****
Fonction:create_game_window:Fonction pour la création de la fenêtre de jeu
Entrée:void
Sortie:void
*****
void create_game_window()
{
    ///Initialiser l'interface
    Interface *interface;
    interface=initialize_interface(interface);

    ///Afficher tous les widgets
    gtk_widget_show_all(interface->F->window);
    gtk_main();
}

```

```

● ● ●

/****************************************************************************
Fonction:on_play_button_clicked:Fonction qui reçoit le signal pour debuter le jeu
Entrée :GtkWidget *widget:l'élément qui émet le signal
        gpointer user_data: la donnée passée en paramètre
Sortie:void
****************************************************************************/
void on_play_button_clicked(GtkWidget *widget, gpointer user_data)
{
    Interface *I=(Interface*)user_data;
    gtk_widget_destroy(I->F->window);
    free(I);
    create_game_window();
}

/****************************************************************************
Fonction:on_quitter_button_clicked:Fonction qui reçoit le signal pour quitter le jeu
Entrée :GtkWidget *widget:l'élément qui émet le signal
        gpointer user_data: la donnée passée en paramètre
Sortie:void
****************************************************************************/
void on_quitter_button_clicked(GtkWidget *widget, gpointer user_data)
{
    gtk_main_quit();
}

/****************************************************************************
Fonction:mainfenetre:Fonction qui crée la fenêtre d'accueil
Entrée :int argc: Nombre d'argument en mode console
        char *argv[]:chaîne de caractère de ses arguments
Sortie:void
****************************************************************************/
void mainfenetre(int argc, char *argv[])
{
    gtk_init(&argc, &argv);///Initialisation de l'interface
    ///Variables et allocation
    GtkWidget *play_button, *quitter_button;
    GtkWidget *play_image, *quitter_image;
    Interface *I=Talloc(Interface);
    TesterAllocation(I);

    ///Créer des boxes pour mes éléments
    GtkWidget *box1 = gtk_box_new(GTK_ORIENTATION_HORIZONTAL,0);
    GtkWidget *box=gtk_box_new(GTK_ORIENTATION_VERTICAL,1);
    gtk_box_pack_end(box,box1,TRUE,TRUE,0);
    ///Créer la fenêtre principale
    I->F=init_fenetre("Accueil Game","Accueil.jpg",1000,700);
    creer_fenetre(I,TRUE);

    ///chargement et création de l'image de l'accueil
    I->Image=Talloc(MyImage);
    TesterAllocation(I->Image);
    I->Image=init_image("Accueil.jpg",1000,600);
    creer_image(I->Image);
    gtk_box_pack_start(box,I->Image->image,TRUE,TRUE,0);

    ///Créer et ajouter le bouton Play avec une image
    play_button = gtk_button_new();
    play_image = gtk_image_new_from_file("bt.png");
    gtk_button_set_image(GTK_BUTTON(play_button), play_image);
    gtk_box_pack_start(box1,play_button,TRUE,TRUE,0);
    g_signal_connect(play_button, "clicked", G_CALLBACK(on_play_button_clicked),I);

    ///Créer et ajouter le bouton Quitter avec une image
    quitter_button = gtk_button_new();
    quitter_image = gtk_image_new_from_file("boutonQuitter.png");
    gtk_button_set_image(GTK_BUTTON(quitter_button), quitter_image);
    gtk_box_pack_end(box1,quitter_button,TRUE,TRUE,0);
    g_signal_connect(quitter_button, "clicked", G_CALLBACK(on_quitter_button_clicked), NULL);

    gtk_container_add(GTK_CONTAINER(I->F->window), box);

    ///Afficher tous les widgets
    gtk_widget_show_all(I->F->window);
    gtk_main();
}

//FIN DE LA FONCTION.

```



```

*****Fonction:play_sound*****
Fonction:play_sound
Entrée :char le chemin de l'audio a jouer.
Sortie:@
*****
void play_sound(const char *sound_file) {
    PlaySound(TEXT(sound_file), NULL, SND_FILENAME | SND_ASYNC);
}

*****Fonction:main:Fonction principale*****
Fonction:main:Fonction principale
Entrée :int argc: Nombre d'argument en mode console
        char *argv[]:chaine de caractère de ses arguments
Sortie:@
*****
int main(int argc,char **argv)
{
    const char *sound_path = "C:\\\\Users\\\\hamza\\\\Downloads\\\\backSound.wav";
    play_sound(sound_path);
    InitObject();///Initialiser les objets de l'interface
    initFrameGlad();///Cree les frames de l'interface
    mainfenetre(argc,argv);
    return 0;
}

```

III-Conclusion :

Ce rapport nous a permis de présenter le travail réalisé dans le cadre du module « Atelier de programmation ». Le projet consistait à développer une application de bureau basée sur la bibliothèque GTK, simulant un jeu de gladiateurs.

Ce projet a été une occasion précieuse d'enrichir nos connaissances et compétences en programmation d'interfaces graphiques, tout en mettant en pratique l'ensemble des acquis de notre formation.

Nous avons également appris l'importance du travail en équipe, de la bonne organisation, et du respect des délais dans la réalisation d'un projet.

Cependant, nous avons rencontré quelques difficultés liées à l'utilisation de la bibliothèque GTK, au manque de temps et à la contrainte du temps durant la réalisation du projet. Néanmoins, nous avons pu mettre à profit les connaissances acquises dans les modules de SDD et d'Algorithmique. Nous sommes convaincus que ce projet basé sur GTK a été une excellente occasion d'appliquer et de renforcer nos compétences en SDD et en Algorithmique.