Vos premières fenêtres SDL 2

Par Alexandre Laurent

Date de publication : 3 octobre 2013

Dernière mise à jour : 25 juillet 2018

DÉBUTANT

Après avoir créé votre premier projet avec la bibliothèque SDL 2, l'étape logique faisant suite est la création d'une première fenêtre dans laquelle vous allez afficher votre jeu.

Commentez

Navigation

Tutoriel précédent : **installation**

Sommaire

Tutoriel suivant : **afficher son premier sprite**

I - Introduction

Après avoir **installé et configuré la SDL**, vous pouvez maintenant commencer à programmer votre jeu. La première chose à effectuer pour utiliser la bibliothèque est **de l'initialiser**. Ensuite, vous pourrez utiliser ses fonctions et, entre autres, ouvrir une **fenêtre pour votre application**.

*Autant que possible, ce tutoriel explique l'utilisation des fonctions. Si vous avez un doute, ou que le tutoriel n'est pas clair sur un point, vous pouvez vous référer à la **documentation officielle**. Si cela ne suffit pas, n'hésitez pas à demander **sur le forum**.*

II - Initialisation de la bibliothèque

Le code du **tutoriel précédent**, copié ci-dessous, initialisait et ouvrait déjà une fenêtre avec la bibliothèque. Nous allons simplement maintenant l'analyser.

```
#include <SDL2/SDL.h>

#include <stdio.h>

int main(int argc, char** argv)
{
    /* Initialisation simple */
    if (SDL_Init(SDL_INIT_VIDEO) != 0 )
    {
        fprintf(stdout,"Échec de l'initialisation de la SDL (%s)\n",SDL_GetError());
        return -1;
    }

    {
        /* Création de la fenêtre */
        SDL_Window* pWindow = NULL;
        pWindow = SDL_CreateWindow("Ma première application SDL2",SDL_WINDOWPOS_UNDEFINED,
                                                                  SDL_WINDOWPOS_UNDEFINED,
                                                                  640,
                                                                  480,
                                                                  SDL_WINDOW_SHOWN);


        if( pWindow )
        {
            SDL_Delay(3000); /* Attendre trois secondes, que l'utilisateur voie la fenêtre */

            SDL_DestroyWindow(pWindow);
        }
        else
        {
            fprintf(stderr,"Erreur de création de la fenêtre: %s\n",SDL_GetError());
        }
    }

    SDL_Quit();

    return 0;
}
```

La section permettant l'initialisation de la bibliothèque est :

```
if (SDL_Init(SDL_INIT_VIDEO) != 0 )
{
    fprintf(stdout,"Échec de l'initialisation de la SDL (%s)\n",SDL_GetError());
    return -1;
}
```

Vous remarquerez donc l'utilisation de la fonction SDL_Init() permettant de « démarrer » la bibliothèque.

Afin de démarquer les fonctions de la bibliothèque SDL de vos fonctions ou des fonctions de la bibliothèque C standard, les développeurs de la bibliothèque ont nommé toutes les fonctions de la SDL en commençant par SDL_.

II-A - Le paramètre de SDL_Init()

L'unique paramètre de la fonction permet de définir les composants que la bibliothèque doit initialiser. Ceux-ci peuvent être :

SDL_INIT_TIMER	Initialise le sous-système des chronomètres
SDL_INIT_AUDIO	Initialise le sous-système pour l'audio
SDL_INIT_VIDEO	Initialise le sous-système pour le rendu
SDL_INIT_JOYSTICK	Initialise le sous-système pour les joysticks
SDL_INIT_HAPTIC	Initialise le sous-système pour le retour de force
SDL_INIT_GAMECONTROLLER	Initialise le sous-système pour les contrôleurs de jeux
SDL_INIT_EVENTS	Initialise le sous-système pour les événements
SDL_INIT_EVERYTHING	Initialise tous les sous-systèmes nommés ci-dessus.
SDL_INIT_NOPARACHUTE	Avec cette option, la SDL ne mettra pas en place de fonction de nettoyage lors de la réception de signaux fatals (par exemple : erreur de segmentation).

II-B - La valeur de retour de SDL_Init()

Dans le code mis en avant ci-dessus, la valeur de retour de la fonction est vérifiée afin de quitter le programme lors d'un cas d'erreur. Lorsqu'une erreur se produit, la valeur est différente de 0. Afin d'informer l'utilisateur (et, bien souvent le programmeur), un message clair est affiché sur la sortie standard d'erreur. La fonction SDL_GetError() permet de récupérer un descriptif de l'erreur rencontrée par la bibliothèque.

Il est conseillé de vérifier les valeurs de retour de chaque fonction pouvant rencontrer des erreurs (pas uniquement lors de l'utilisation de la SDL) afin de réagir au mieux par rapport à ces cas qui arrivent plus souvent que l'on ne le croit. De plus, cela vous aidera en tant que programmeur afin de trouver les bogues plus rapidement.

III - Une simple fenêtre

Une fois que la bibliothèque est initialisée, vous pouvez l'utiliser et donc, ouvrir une fenêtre.

Le code du programme ouvrant une fenêtre dans le **précédent tutoriel** était :

```
SDL_Window* pWindow = NULL;
pWindow = SDL_CreateWindow("Ma première application SDL2",SDL_WINDOWPOS_UNDEFINED,
                                                          SDL_WINDOWPOS_UNDEFINED,
                                                          640,
                                                          480,
                                                          SDL_WINDOW_SHOWN);

if( pWindow )
{
    /* Suite du programme */

    SDL_DestroyWindow(pWindow); // Destruction de la fenêtre
}
else
{
        fprintf(stderr,"Erreur de création de la fenêtre: %s\n",SDL_GetError());
}
```

La fonction créant une fenêtre est : SDL_CreateWindow(). Celle-ci retourne un pointeur sur une variable de type SDL_Window (nouvelle structure implémentée à partir de SDL 2).

III-A - Les paramètres de SDL_CreateWindow()

SDL_CreateWindow() attend plusieurs paramètres afin de créer la fenêtre. Ceux-ci sont :

- le nom à afficher ;
- la position en X ;
- la position en Y ;
- la taille en largeur ;
- la taille en hauteur ;
- les options (plein écran, **fenêtre OpenGL**…).

Pour la position, vous pouvez utiliser SDL_WINDOWPOS_UNDEFINED ou SDL_WINDOWPOS_CENTERED, deux valeurs spéciales indiquant que c'est à la bibliothèque de déterminer la position de la fenêtre. La première valeur laisse le libre champ à la bibliothèque de placer la fenêtre où elle le souhaite, la seconde permet de toujours avoir une fenêtre centrée sur l'écran.

III-A-1 - Les options de création de la fenêtre

Le dernier paramètre de la fonction SDL_CreateWindow() permet d'indiquer des options spécifiques pour la fenêtre à créer. Celles-ci sont :

SDL_WINDOW_FULLSCREEN	Crée une fenêtre plein écran
SDL_WINDOW_FULLSCREEN_DESKT(	Crée une fenêtre plein écran à la résolution actuelle du bureau
SDL_WINDOW_OPENGL	Crée une fenêtre pouvant être utilisée pour OpenGL
SDL_WINDOW_SHOWN	Crée une fenêtre et l'afficher
SDL_WINDOW_HIDDEN	Crée une fenêtre cachée
SDL_WINDOW_BORDERLESS	Crée une fenêtre sans bordure
SDL_WINDOW_RESIZABLE	Crée une fenêtre redimensionnable
SDL_WINDOW_MINIMIZED	Crée une fenêtre minimisée
SDL_WINDOW_MAXIMIZED	Crée une fenêtre maximisée
SDL_WINDOW_INPUT_GRABBED	Crée une fenêtre et récupère le focus d'entrée
SDL_WINDOW_INPUT_FOCUS	Crée une fenêtre et donne le focus d'entrée
SDL_WINDOW_MOUSE_FOCUS	Crée une fenêtre et donne le focus de la souris
SDL_WINDOW_FOREIGN	La fenêtre n'est pas créée par la SDL

III-B - La valeur de retour de SDL_CreateWindow()

La fonction SDL_CreateWindow() renvoie un pointeur sur une variable de type **SDL_Window**, représentant la fenêtre. Chaque fois que vous devez agir sur cette fenêtre, vous devez indiquer la variable retournée.

En cas d'erreur, la variable retournée aura la valeur NULL. Comme pour **l'initialisation de la bibliothèque**, vous pouvez utiliser SDL_GetError() pour avoir une chaîne de caractères décrivant l'erreur.

	Encore une fois, il est fortement conseillé de vérifier la valeur retournée par la fonction. En effet, l'utilisation d'un pointeur NULL entrainera des erreurs de segmentation (« segmentation fault »).

IV - Destruction d'une fenêtre

Comme nous avons pu le voir durant la **création d'une fenêtre** avec la fonction SDL_CreateWindow(), une fenêtre est représentée par une variable de type SDL_Window. Afin de détruire une fenêtre précédemment créée, vous devez passer la variable la représentant à la fonction : SDL_DestroyWindow(), comme suit :

```
SDL_DestroyWindow(pWindow);
```

V - Arrêter la SDL

À la fin du programme (ou, lorsque vous n'avez plus besoin de la SDL), il est nécessaire de faire l'opération inverse à l'initialisation avec la fonction : SDL_Quit(). (Ceci est aussi vrai, si vous rencontrez une erreur lors de l'exécution du programme.)

La fonction SDL_Quit() ne prend pas de paramètre et ne retourne pas de valeur, il suffit de l'appeler de la sorte :

```
SDL_Quit();
```

VI - Plus de détails sur les fenêtres

Maintenant que vous avez vu la création d'une fenêtre avec la SDL 2, vous pouvez maintenant vous intéresser aux détails concernant sa gestion. En effet, la bibliothèque vous permet de créer une fenêtre **compatible avec un rendu effectué par OpenGL**, de créer **plusieurs fenêtres** ou même **d'interagir sur une fenêtre**.

VI-A - Fenêtre OpenGL (3D)

Une fenêtre OpenGL se construit de la même manière qu'une fenêtre normale : avec la fonction SDL_CreateWindow(). Toutefois, pour que OpenGL puisse dessiner dedans, il faut obligatoirement spécifier l'option SDL_WINDOW_OPENGL lors de la création.

VI-B - Gestion de plusieurs fenêtres

Depuis la SDL 2, il est possible d'afficher et de gérer plusieurs fenêtres SDL. Cela est possible grâce à l'introduction de la structure SDL_Window, représentant une fenêtre. En effet, chaque appel à SDL_CreateWindow() retournera un pointeur sur une variable SDL_Window différente, permettant ainsi de les identifier par la bibliothèque. Lorsque vous souhaitez agir sur telle ou telle fenêtre, il suffira d'utiliser la variable correspondante.

VI-C - Interagir avec une fenêtre

De multiples fonctions sont disponibles dans la bibliothèque afin d'interagir avec la bibliothèque. Voici un récapitulatif :

SDL_CreateWindow()	Crée une fenêtre
SDL_CreateWindowFrom()	Crée une fenêtre SDL à partir d'une fenêtre déjà existante
SDL_DestroyWindow()	Détruit une fenêtre
SDL_GetWindowData()	Récupère les données utilisateur associées à la fenêtre
SDL_GetWindowFlags()	Récupère les options actuelles de la fenêtre
SDL_GetWindowGrab()	Détermine si la fenêtre a le focus clavier
SDL_GetWindowPosition()	Récupère la position actuelle de la fenêtre
SDL_GetWindowSize()	Récupère la taille actuelle de la fenêtre
SDL_GetWindowTitle()	Récupère le titre actuel de la fenêtre
SDL_HideWindow()	Cache la fenêtre
SDL_MaximizeWindow()	Agrandit la fenêtre
SDL_MinimizeWindow()	Réduit la fenêtre dans la barre des tâches
SDL_RaiseWindow()	Place la fenêtre devant les autres
SDL_RestoreWindow()	Restaure la taille et la position d'une fenêtre minimisée ou maximisée
SDL_SetWindowData()	(Re)Définit les données utilisateur de la fenêtre
SDL_SetWindowFullscreen()	Passe la fenêtre en plein écran
SDL_SetWindowGrab()	(Re)Donne le focus clavier à cette fenêtre
SDL_SetWindowIcon()	(Re)Définit l'icône de la fenêtre
SDL_SetWindowPosition()	(Re)Définit la position de la fenêtre
SDL_SetWindowSize()	(Re)Définit la taille de la fenêtre
SDL_SetWindowBordered()	(Re)Définit l'affichage des bordures de la fenêtre
SDL_SetWindowTitle()	(Re)Définit le titre de la fenêtre
SDL_ShowWindow()	Affiche la fenêtre

VII - La structure SDL_Window

Il n'est certes pas essentiel de connaître le contenu de cette structure, toutefois, cela peut être intéressant et didactique de le savoir. En sachant de telles choses, vous pouvez comprendre plus facilement comment a été construite la SDL et ainsi mieux comprendre son fonctionnement.

La structure est définie dans le fichier d'entête SDL_video.h. Toutefois la seule ligne intéressante de ce fichier est :

```
typedef struct SDL_Window SDL_Window;
```

La ligne définit bien le type SDL_Window, mais ne donne pas d'information sur son contenu. En effet, le programmeur utilisant la bibliothèque n'a pas à avoir accès à ses champs (afin d'éviter de mauvais usages) et de cette façon, il n'y a pas accès. Pour en savoir plus, il faut explorer le code source de la bibliothèque. On trouvera dans le fichier SDL_sysvideo.h la définition suivante :

```
/* Définit la structure de fenêtre SDL, correspondant aux fenêtres de haut niveau */
struct SDL_Window
{
    const void *magic;
    Uint32 id;
    char *title;
    SDL_Surface *icon;
```

```
    int x, y;
    int w, h;
    int min_w, min_h;
    int max_w, max_h;
    Uint32 flags;

    /* Conserve la position et la taille d'une fenêtre non plein écran */
    SDL_Rect windowed;

    SDL_DisplayMode fullscreen_mode;

    float brightness;
    Uint16 *gamma;
    Uint16 *saved_gamma;        /* (un simple décalage du gamma) */

    SDL_Surface *surface;
    SDL_bool surface_valid;

    SDL_WindowShaper *shaper;

    SDL_WindowUserData *data;

    void *driverdata;

    SDL_Window *prev;
    SDL_Window *next;
};
```

Certains pourraient avoir deviné quelque peu son contenu. En effet, il semble logique que cette structure conserve la taille de la fenêtre, le titre et le mode (plein-écran ou non), car ce sont les paramètres que nous donnons lors de la **création de celle-ci**. On remarquera aussi l'utilisation de SDL_Surface (une structure déjà présente dans SDL 1.X) pour conserver l'icône (icon) ainsi que le contenu de la fenêtre. Plus étonnant, la structure garde aussi un pointeur sur la fenêtre précédemment créée (prev) et la fenêtre suivante (si une seconde fenêtre est créée) (next). On devinera alors que la structure SDL_Window est aussi un maillon d'une liste doublement chainée.

VIII - Les fenêtres dans les jeux vidéo

Maintenant que vous savez utiliser les fenêtres avec la SDL 2, il est important de revenir sur quelques aspects de la gestion des fenêtres pour les jeux vidéo. En effet, la fenêtre est le cadre où apparaitra votre jeu. Chaque joueur a ses propres préférences de jeu, certains aiment jouer dans une petite fenêtre, d'autres en plein écran. Pour cela, vous allez devoir mettre en place un **mode plein écran** ou encore, **détecter les résolutions utilisables** sur la machine.

VIII-A - Idées sur la gestion plein écran

Afin de fournir un jeu ou une application qui ne contraint pas l'utilisateur et ne dérange pas ses habitudes (ce qui pourrait entrainer un mécontentement), il est conseillé d'éviter l'imposition du mode plein écran. Toutefois, le proposer en option (par exemple, au démarrage de l'application) est une bonne idée.

Vous pouvez aussi opter pour la mise en place d'une touche permettant de passer à la volée du mode fenêtré au mode plein écran et inversement. Pour cela, vous devrez utiliser la fonction : SDL_SetWindowFullscreen().

VIII-A-1 - Les paramètres de SDL_SetWindowFullscreen

La fonction accepte deux paramètres :

- la fenêtre sur laquelle vous voulez agir ;
- le nouvel état de la fenêtre parmi :
    - 0, pour remettre la fenêtre en mode fenêtré ;

- SDL_WINDOW_FULLSCREEN, pour mettre la fenêtre en plein écran ;
- SDL_WINDOW_FULLSCREEN_DESKTOP, pour mettre la fenêtre en plein écran à la résolution du bureau.

VIII-A-2 - La valeur de retour de SDL_SetWindowFullscreen

La fonction retourne 0 si le changement a été effectué, ou une valeur négative dans le cas contraire. En cas d'erreur, vous pourrez récupérer un message précis sur la cause avec la fonction SDL_GetError().

VIII-A-3 - Exemple

```
SDL_Event event;
while (SDL_PollEvent(&event)) // Récupération des actions de l'utilisateur
{
    switch(event.type)
    {
        case SDL_QUIT: // Clic sur la croix
            quit=1;
            break;
        case SDL_KEYUP: // Relâchement d'une touche
            if ( event.key.keysym.sym == SDLK_f ) // Touche f
            {
                // Alterne du mode plein écran au mode fenêtré
                if ( fullscreen == 0 )
                {
                    fullscreen = 1;
                    SDL_SetWindowFullscreen(pWindow,SDL_WINDOW_FULLSCREEN);
                }
                else if ( fullscreen == 1 )
                {
                    fullscreen = 0;
                    SDL_SetWindowFullscreen(pWindow,0);
                }
            }
            break;
    }
}
```

La gestion des événements sera analysée dans un prochain tutoriel.

Dans une application complète, il est déconseillé de mettre un tel code directement dans la section gérant les événements. En effet, faire de la sorte et pour tous les événements mène à un code difficile à relire et à maintenir.

VIII-B - Recherche des résolutions disponibles

Dans une optique proche de l'option pour le mode fenêtré/plein écran, il est intéressant pour l'utilisateur d'avoir le choix de la résolution de l'application. Vous devrez donc rechercher les résolutions disponibles sur sa machine et lui proposer de choisir parmi l'une d'elles. Le code suivant permet de détecter les résolutions utilisables :

```
// On récupère le nombre de modes d'affichage pour l'écran 0
int modeNumber = SDL_GetNumDisplayModes(0);
if (modeNumber < 0)
{
 fprintf(stdout,"Échec lors de la récupération du nombre de modes (%s)\n",SDL_GetError());
 return -2;
}
fprintf(stdout,"Vous avez %d mode(s) d'affichage\n", modeNumber);
```

```
// Affichage des différents modes
SDL_DisplayMode displayMode;
for (i = 0 ; i < modeNumber ; i++)
{
    error = SDL_GetDisplayMode(0, i, &displayMode);
    if (error < 0)
    {
        fprintf(stdout, "Échec lors de la récupération du mode d'affichage
 (%s)\n", SDL_GetError());
        return -3;
    }

    fprintf(stdout, "Mode %d : %dx%dx%d\n", i, displayMode.w, displayMode.h,
 displayMode.refresh_rate);
}
```

Ce code doit être inséré après l'**initialisation de la SDL** (et avant **son arrêt**).

Pour afficher les résolutions disponibles, il est nécessaire d'utiliser les fonctions SDL_GetNumDisplayMode() et SDL_GetDisplayMode(). La première retourne le nombre de modes d'affichage disponibles, la seconde, remplit une structure avec les informations du mode d'affichage.

VIII-B-1 - Les paramètres de SDL_GetNumDisplayMode()

La fonction SDL_GetNumDisplayMode() accepte un unique paramètre qui est l'index de l'écran. Vous pouvez mettre 0 par défaut (car vous avez normalement, toujours un écran), faisant que vous récupérerez les informations pour le premier écran.

La fonction SDL_GetNumVideoDisplays() vous permet de connaître le nombre d'écrans présents sur votre configuration.

VIII-B-2 - La valeur de retour de SDL_GetNumDisplayMode()

La fonction retourne le nombre de modes d'affichage pour l'écran indiqué. Si le nombre est négatif, c'est qu'une erreur s'est produite.

VIII-B-3 - Les paramètres de SDL_GetDisplayMode()

La fonction accepte trois paramètres :

- l'index de l'écran ;
- l'index du mode d'affichage à récupérer ;
- un pointeur sur la structure SDL_DisplayMode qui sera remplie avec les informations voulues.

VIII-B-4 - La valeur de retour de SDL_GetDisplayMode()

Si la valeur est négative, une erreur s'est produite. Si elle est de zéro, tout va bien et vous devriez pouvoir lire les informations du mode d'affichage dans la variable de type SDL_DisplayMode.

VIII-B-5 - La structure SDL_DisplayMode

Celle-ci est constituée de cinq champs :

- Uint32 **format** : le format des pixels pour ce mode d'affichage (voir **SDL_PixelFormatEnum**) ;
- int **w** : la largeur ;
- int **h** : la hauteur ;
- int **refresh_rate** : le taux de rafraîchissement (ou 0 si non spécifié) ;
- void* **driverdata** : des données spécifiques au pilote.

IX - Remerciements

Merci à **Neckara** et **Kannagi** pour leur relecture et leurs conseils dans l'amélioration de ce tutoriel.

Merci à **ClaudeLELOUP** pour sa relecture orthographique attentive.

Navigation

Tutoriel précédent : **installation**	**Sommaire**	Tutoriel suivant : **afficher son premier sprite**