Chapitre II : Premières applications avec SDL

par Loka

Date de publication : 30/03/2006

Dernière mise à jour : 17/04/2006

C'est ici que débute notre apprentissage de SDL avec la création de nos deux premiers programmes.

II - Débuts avec SDL

Bien, maintenant que SDL est bien installé, nous allons pouvoir commencer à rentrer dans le vif du sujet :

créer une application SDL.

Pour se familiariser avec les différentes fonctions que nous offrent SDL et ses extensions, nous allons procéder pas à pas.

Pour commencer, dans le chapitre II-A, nous allons apprendre à créer une fenêtre SDL vierge, ce qui nous apprendra à initialiser SDL et à quitter SDL correctement.

Ensuite, dans le chapitre II-B, nous allons apprendre à charger des images sur la fenêtre SDL et de les appliquer les unes sur les autres, cela s'appelle le *blitting*

Par la suite, dans les chapitres IV (Gestion des événements) et XV (gestion des événements avancés), je vais tenter de vous expliquer comment marche la gestion des événements sous SDL et ainsi que vous puissiez créer une application qui puisse réagir lorsque l'utilisateur appuie sur une touche, clique à un endroit précis, etc..

Le chapitre V quant à lui vous permettra d'apprendre à utiliser une fonction particulière de SDL : SDL_SetColorKey(), qui vous sera très utile si vous souhaitez créer un jeu ou même pour plein d'autres applications concrètes dans lesquels vous avez besoin de manipuler des images.

Nous finirons notre familiarisation de SDL avec la création d'une feuille de sprite et son utilisation dans une application SDL (chapitre VI).

II-A - Créer une fenêtre SDL

Voici le code source vous permettant de créer une fenêtre SDL, je vais vous l'expliquer point par point dans la suite.

fenêtre SDL

```
#include <stdlib.h>
#include <stdio.h>
#include <SDL/SDL.h>

int main( int argc, char *argv[ ] )
{
    SDL_Surface *screen;
    if( SDL_Init( SDL_INIT_VIDEO ) == -1 )
    {
        printf( "Can't init SDL:  %s\n", SDL_GetError( ) );
        return EXIT_FAILURE;
    }

    atexit( SDL_Quit );
    screen = SDL_SetVideoMode( 640, 480, 16, SDL_HWSURFACE );
```

fenêtre SDL

```
        if( screen == NULL )
        {
            printf( "Can't set video mode: %s\n", SDL_GetError( ) );
            return EXIT_FAILURE;
        }

        SDL_Delay( 3000 );

        return EXIT_SUCCESS;
    }
```

Dans les 2 premières lignes, on inclut le fichier d'en-tête nécessaire pour l'utilisation de la SDL (SDL.h) et la librairie stdlib.h pour l'utilisation de exit() et atexit().

Ensuite vient la fonction bien connue int main() avec ces 2 arguments obligatoire pour un programme SDL (nous y reviendrons dans le chapitre suivant).

Une ligne plus bas on peut déjà trouver le premier type inconnu, **SDL_Surface**, qui est utilisé ici pour une surface d'écran.

Ensuite viens la première fonction inconnue, elle est nommée: **int SDL_Init(Uint32 flags)** et est utilisée pour initialiser la SDL où le paramètre flags est(sont) le(s) subsystem(s) qui doit(-vent) être initialisé(s).

La combinaison de plusieurs flags est possible en introduisant des '|' entre chaque subsystem, donc peut être de la forme : *SDL_Init(SDL_INIT_VIDEO* | *SDL_INIT_AUDIO)*, qui initialisera le subsystem de la vidéo et de l'audio.

Les flags possibles sont:

SDL_INIT_AUDIO

SDL_INIT_VIDEO

SDL_INIT_CDROM

SDL_INIT_TIMER

SDL_INIT_JOYSTICK

SDL_INIT_EVENTTHREAD

Nous n'initialiserons que le sous-système de la vidéo puisque nous n'utilisons pas encore d'autres fonctions que celle de la vidéo.

Pour ceux qui utilisent tous les sous-systèmes peuvent par contre simplement déclarer **SDL_INIT_EVERYTHING** qui les initialisent tous.

Pour les curieux la SDL donne également la possibilité d'utiliser la fonction *int SDL_InitSubSystem(Uint32 flags)* qui est peut être très utile lorsque vous ne voulez pas initialiser un subsystem que lorsque celui-ci est choisi en option, ou bien que vous ne l'utilisez que plus tard dans le programme.

Un bon exemple pour ça est par exemple le joystick, car on sait rarement dès le début du jeu que l'utilisateur veut utiliser son joystick (s'il en possède un). Donc on place cela dans les options et si l'utilisateur décide de l'utiliser, le sous-système sera initialisé.

Pour fermer les sous-systèmes il faut utiliser **int SDL_Quit( )** qui ferme tous les sous-systèmes initialisés, mais si vous ne voulez en fermer qu'un seul vous pouvez utiliser **void SDL_QuitSubSystem(Uint32 flags)**.

Il est tout de même possible que l'initialisation retourne *-1* et donc rate, c'est pour cela que nous faisons un test.

Quand cela se passe, on met un message dans la console d'erreur de la SDL, avec **SDL_GetError()** qui retourne l'erreur qui s'est produite. Dans le cas où il y a une erreur, nous terminons le programme en retournant 1

Si aucune erreur n'est intervenue, on appelle la fonction atexit() avec en paramètre SDL_Quit, ce qui veut dire que quand le programme est terminé il doit utiliser SDL_Quit. Puisque l'initialisation s'est bien passée et que tout notre programme se trouve en mémoire, avec SDL_Quit on libère tout lors de l'arrêt du programme.

Maintenant, donnons le paramétrage vidéo avec la fonction **SDL_Surface *SDL_SetVideoMode(int width, int height, int bpp, Uint32 flags)** qui prend comme paramètres la largeur de l'écran, sa hauteur, son nombre de bits par pixel et les flags.

Les trois premiers devraient normalement être bien connus, tandis que le quatrième est tout nouveau, puisqu'il n'accepte pas du tout les mêmes flags que SDL_Init. Ne nous attardons pas trop là-dessus pour l'instant.

Comme vous avez pu le constater dans le prototype de la fonction SDL_SetVideoMode, celle-ci retourne un pointeur qui pointe vers l'adresse de la plate-forme vidéo. C'est d'ailleurs pour cela que nous avons mis notre pointeur *screen* en tant que SDL_Surface.

Nous devons encore une fois tester si notre pointeur pointe bien quelque part. Car il est bien possible que certains modes vidéo ne sont pas acceptés pas certaines cartes graphiques ou système#

Quand quelque chose tourne mal avec la fonction SDL_SetVideoMode, ici *screen* vaudra NULL, donc aucune adresse n'est retournée et le pointeur *screen* reste non valide.

Si tout s'est bien passé, nous devrions maintenant avoir un écran noir qui s'ouvre lors de l'exécution de l'application, pour que cette fenêtre ne disparaisse pas tout de suite, nous utilisons la fonction **void SDL_Delay(Uint32 ms)** qui, dans notre cas, fait attendre 3000 millisecondes à l'application avant de quitter (soit 3 secondes).

Et voilà, maintenant vous savez créer une fenêtre avec SDL.

Maintenant que nous savons faire ça, on va essayer de mettre quelques trucs dans notre fenêtre comme des images par exemple.

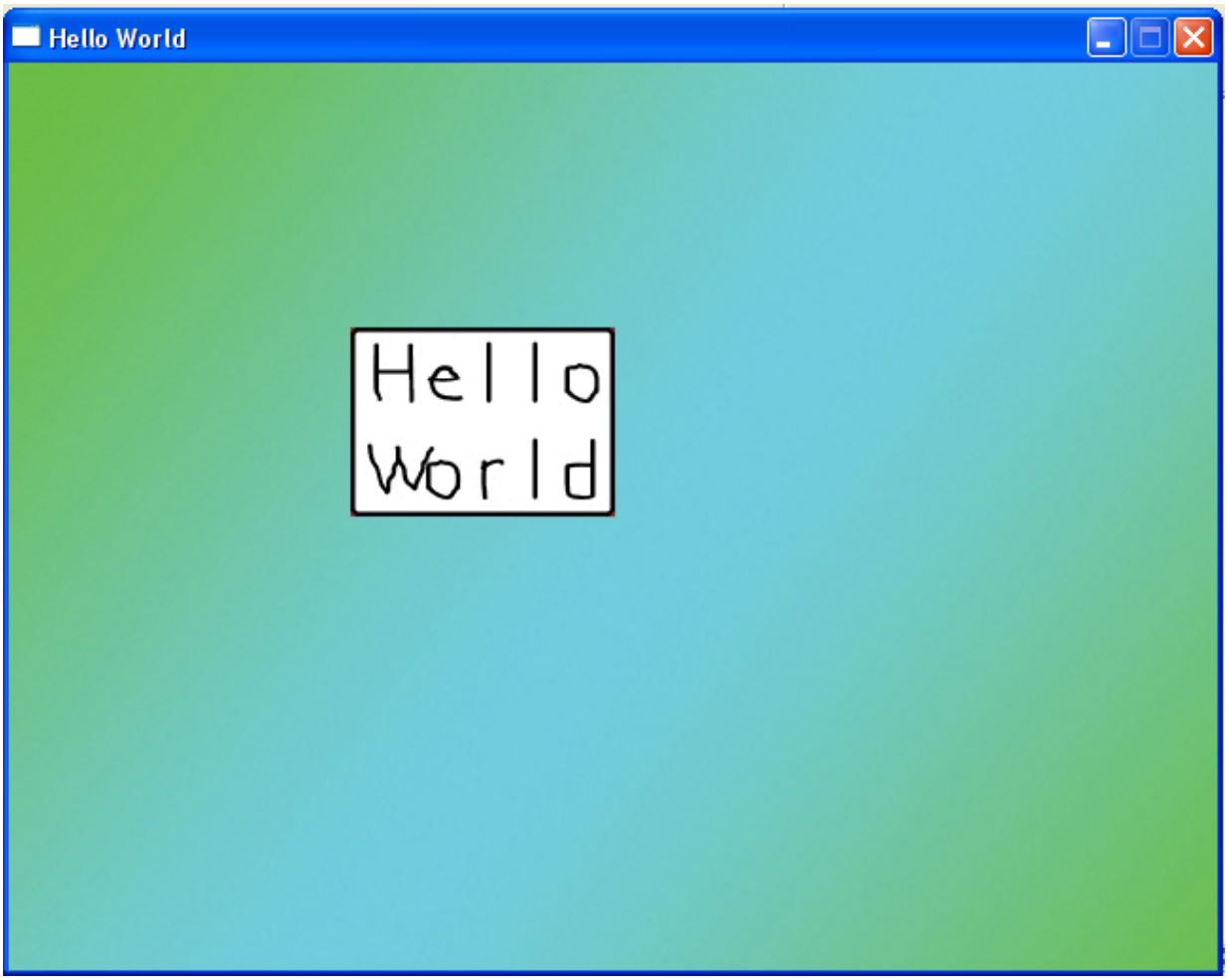
C'est l'objet de la partie ci-dessous.

II-B - Hello World - Blitting

Dans cette partie, à travers un exemple de programme simple (Hello World), nous allons apprendre à charger des surfaces et à les appliquer les unes sur les autres (**Blitting**).

Une surface est juste un mot extravagant pour une image. Blitter une surface signifie juste d'appliquer une surface sur une autre.

Voila en gros ce que nous cherchons à obtenir :



Nous allons commencer par include les en-têtes

header
```
#include "SDL/SDL.h"
#include <string>
```

SDL.h est inclu parce que bien sur nous allons utiliser des fonctions SDL ^^

string quand à lui ben... c'est juste que je préfère std::string plutôt que char*.

Ensuite nous allons définir les différents attributs de notre écran.

attributs de l'écran
```
//Les attributs de notre écran
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_BPP = 32;
```

Je pense que vous comprenez ce que définissent SCREEN_WIDTH et SCREEN_HEIGHT.

SCREEN_BPP définit le nombre de bits par pixel. Dans cet exemple, nous utilisons des couleurs 32 bits.

Après avoir défini les attributs pour notre écran, il nous faut définir les surfaces que nous allons utiliser.

Nous allons avoir besoin de 3 surfaces :

- la surface de l'écran qu'on appellera screen
- la surface qui servira de fond d'écran (background)
- la surface qui servira à afficher notre "Hello World" qui sera une image bitmap avec écrit Hello World dessus

Voici la définition de ces 3 surfaces :

surfaces
```
//Les surfaces que nous allons utiliser
SDL_Surface *message = NULL;
SDL_Surface *background = NULL;
SDL_Surface *screen = NULL;
```

Remarque : c'est une bonne idée de toujours mettre ses pointeurs à NULL s'ils pointent sur rien.

Nous avons terminé la partie définition des différentes variables dont nous aurons besoin, maintenant passons dans le vif du sujet : **le chargement d'image**.

Nous allons construire une fonction qui chargera une image à partir de son nom.

Voici le prototype de la fonction :

protoype fonction load_image
```
SDL_Surface *load_image( std::string filename )
```

Cette fonction retourne un pointeur vers la version optimisé de l'image chargée.

Pourquoi la version optimisée ?

Comme nous avons définit notre écran en 32bits, si le bitmap est en 24bits, il va nous falloir faire quelques petits changements sur l'image.

En effet, bien que SDL soit capable d'appliquer une image d'un format différent sur une autre, il faut pour cela qu'il change de lui même le format "*à l'arrache*" ce qui peut causer quelques ralentissements.

Comme vous le voyez, ce n'est pas une bonne idée d'essayer de blitter une surface sur une autre si les 2 sont dans des formats différents.

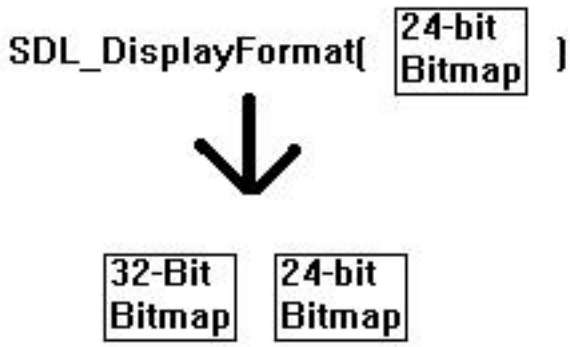
Voici comment nous allons donc procéder :

Tout d'abord il va nous falloir définir 2 variables, une variable qui servira de tampon pour stocker l'image chargée temporairement et une autre où on mettra l'image optimisée.

Il va nous falloir charger l'image en utilisant la fonction **SDL_LoadBMP(char *)** qui s'occupera de charger l'image au format Bitmap et de la mettre dans notre variable tampon.

Si le chargement s'est bien passé, il nous faut créer l'image optimisée.

La fonction **SDL_DisplayFormat(SDL_Surface *)** va nous permettre de créer une image dans le même format que notre écran.



Nous allons donc, comme le montre le petit schéma ci-dessus, nous retrouver avec 2 surfaces : l'ancienne surface et la version optimisée.

Il va donc nous falloir faire un peu de ménage.

On va se servir de la fonction **SDL_FreeSurface(SDL_Surface *)** afin de libérer proprement l'ancienne surface

pour finir, comme notre fonction retourne la version optimisée de l'image, il ne faut pas oublier... de la retourner ^^.

Voici le code de notre fonction de chargement d'image :

protoype fonction load_image
```
   SDL_Surface *load_image( std::string filename ) {
           //Surface tampon qui nous servira pour charger l'image
           SDL_Surface* loadedImage = NULL;

           //L'image optimisée qu'on va utiliser
           SDL_Surface* optimizedImage = NULL;

           //Chargement de l'image
           loadedImage = SDL_LoadBMP( filename.c_str() );

           //Si le chargement se passe bien
           if( loadedImage != NULL ) {
                   //Création de l'image optimisée
                   optimizedImage = SDL_DisplayFormat( loadedImage );

                   //Libération de l'ancienne image
                   SDL_FreeSurface( loadedImage );
           }

           //On retourne l'image optimisée
           return optimizedImage;
   }
```

Par la suite, il nous faut blitter la surface, nous allons donc créer une nouvelle fonction qui se chargera de cette tâche.

Notre fonction prendra en paramètre les coordonnées où l'on veut blitter notre surface, la surface que nous allons

appliquer ainsi que la surface sur laquelle nous allons appliquer notre surface. Voici à quoi cela va ressembler au début :

debut fonction apply_surface
```
void apply_surface( int x, int y, SDL_Surface* source, SDL_Surface* destination ) {
        SDL_Rect offset;

        offset.x = x;
        offset.y = y;
```

Voila donc le début de notre fonction de blitting.

Nous faisons ainsi car SDL_BlitSurface() accepte seulement les offsets dans un **SDL_Rect**.

Un SDL_Rect est un type de donnée représentant un rectangle. Il possède 4 membres représentant les offsets X et Y, la largeur (Width) et la hauteur (Height) du rectangle.

Ici nous avons seulement besoin des données membres X et Y.

Ensuite il nous faut donc blitter la surface (c'est tout de même la finalité de cette fonction), pour cela nous allons utiliser la fonction **SDL_BlitSurface()** de cette façon :

SDL_BlitSurface
```
        //On blitte la surface
        SDL_BlitSurface( source, NULL, destination, &offset );
}
```

Comme vous pouvez le remarquer, ce bout de code est la suite directe, et la fin, de celui mit juste avant.

Notre fonction est donc déjà terminé.

Le premier argument de SDL_BlitSurface est la surface que nous allons utiliser.

Ne vous souciez pas encore du second argument, laissez le à NULL pour le moment.

Le troisième argument est la surface sur laquelle nous allons appliquer la surface que nous allons utiliser.

Le quatrième et dernier argument contient l'offset d'où notre surface sera appliqué sur la surface de destination.

Bon nous avons fini avec les fonctions dont nous allons avoir besoin pour faire notre petit programme, maintenant nous allons passer au main().

Tout d'abord, lorsque vous utilisez SDL, vous devrez toujours utiliser :

main avec SDL
```
int main( int argc, char* args[] )

ou
```

main avec SDL
```
int main( int argc, char** args )
```

Sous windows, n'utilisez jamais int main(), void main(), ou autre chose qui ne marchera pas.

Bien, maintenant que l'on sait tout ça, on va se lancer dans le main en commençant par initialiser SDL :

init SDL
```
int main( int argc, char* args[] ) {
        //Initialisation de tous les sous-systèmes de SDL
        if( SDL_Init( SDL_INIT_EVERYTHING ) == -1 ) {
                return EXIT_FAILURE;
        }
```

Ici nous initialisons SDL en utilisant **SDL_Init()**.

Nous donnons comme paramètre à SDL_Init() **SDL_INIT_EVERYTHING**, qui démarre tous les sous-systèmes SDL(Vidéo, Audio, etc...).

Les sous-systèmes SDL sont les moteurs individuels utilisés pour fabriquer un jeu.

Nous n'allons pas utiliser tous ces sous-systèmes, mais cela ne nous génera pas si nous les initialisons tout de même.

Si SDL ne peut pas s'initialiser, il retourne -1. Dans ce cas, nous attrapons l'erreur en retournant EXIT_FAILURE, qui termine le programme.

Après l'initialisation de SDL, il est temps de fabriquer notre écran et de récupérer un pointeur sur cet écran afin de pouvoir blitter des surfaces dessus.

SDL_SetVideoMode
```
        //Mise en place de l'écran
        screen = SDL_SetVideoMode( SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_BPP, SDL_SWSURFACE );
```

Vous connaissez déjà les 3 premiers arguments si vous avez suivi normalement. Le quatrième argument crée la surface écran dans la mémoire système.

Il nous faut ensuite vérifier si notre pointeur sur surface screen n'est pas NULL, ce qui voudrait dire que ça s'est mal passé.

test ecran
```
        //S'il y a une erreur dans la création de l'écran
        if( screen == NULL ) {
                return EXIT_FAILURE;
        }
```

On va agrémenter notre fenêtre en ajoutant une légende sur le haut de la fenêtre :

legende ecran
```
        //Mise en place de la barre caption
```

legende ecran
```
SDL_WM_SetCaption( "Hello World", NULL );
```

Ici la légende indiquera "Hello World" comme sur le schéma ci-dessous

Ensuite nous allons charger les images :

chargement images
```
//Chargement des images
message = load_image( "hello_world.bmp" );
background = load_image( "background.bmp" );
```

Nous utilisons comme vous le voyez notre fonction load_image().

il est maintenant temps d'utiliser notre deuxième fonction qui va se charger d'appliquer l'image de fond sur l'écran :

background
```
//On applique le fond sur l'écran
apply_surface( 0, 0, background, screen );
```

Avant d'appliquer notre fond, l'écran ressemblait à ça :

Maintenant que nous avons appliquer notre surface, il ressemble à ça en mémoire :

Quand vous appliquez une surface, vous copiez les pixels de l'un sur l'autre surface.

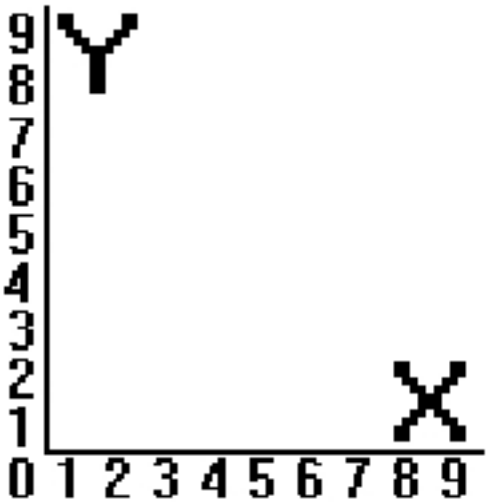
Donc maintenant l'écran ressemble à l'image background que nous venions de charger.

Nous allons donc continuer en appliquant maintenant la surface qui contient notre message "Hello World"

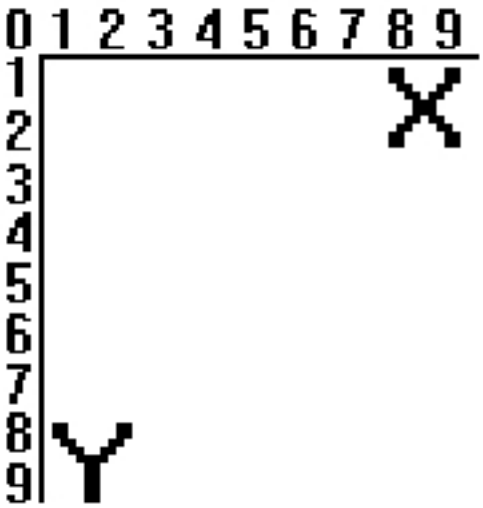
message
```
          //On applique le message sur l'écran
          apply_surface( 180, 140, message, screen );
```

Comme vous le voyez sur le code ci-dessus, nous appliquons notre surface sur l'écran en X=180 et Y=140.

Ce qu'il faut comprendre c'est que le système de coordonnées de SDL ne fonctionne pas ainsi :



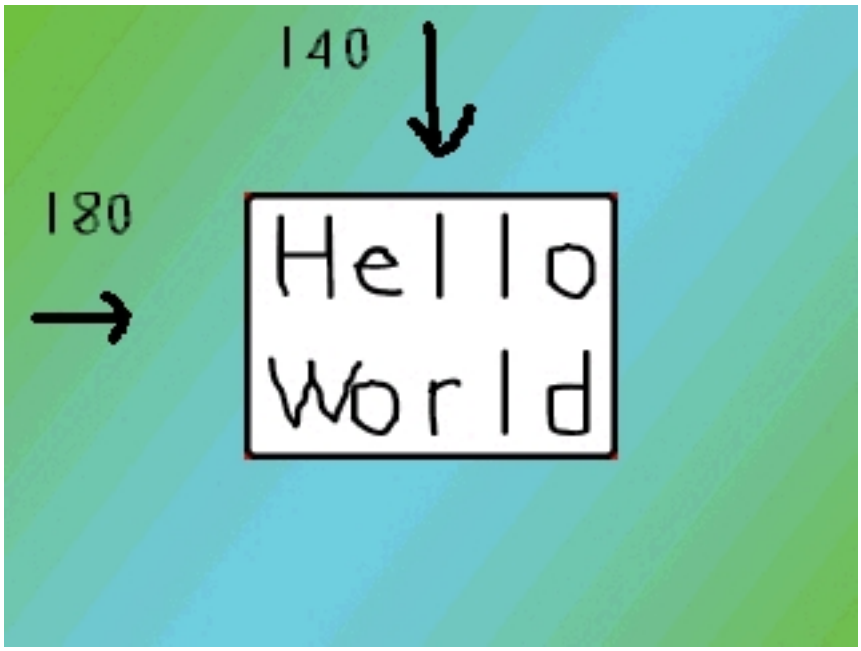
Le système de coordonnées de SDL se présente comme cela :



Donc l'origine se trouve dans le coin en haut à gauche de l'écran.

Ainsi quand vous appliquez la surface message, celle-ci va se placer à 180 pixels sur la droite et 140 pixels vers le bas par rapport au coin haut gauche.

Voici une illustration afin de mieux comprendre :



Le système de coordonnées de SDL peut vous paraître effrayant au début, mais vous vous y ferez vite.

Bien que nous avons appliquer nos surfaces, dans l'état actuel, l'écran restera noir.

En effet, il nous faut mettre à jour l'écran en utilisant la fonction **SDL_Flip()** pour que l'écran ressemble à celui que nous avons en mémoire.

SDL_Flip
```
//Mise à jour de l'écran
if( SDL_Flip( screen ) == -1 ) {
	return EXIT_FAILURE;
}
```

Si il y a une erreur, la fonction retourne -1.

Nous pouvons presque dire que le travail est fini, il nous reste encore 2 choses à faire : laisser l'écran s'afficher quelques secondes et terminer notre application correctement.

En effet, si nous lançons notre programme, celui-ci disparaîtra de votre écran si rapidement que vous n'aurez même pas le temps d'admirer votre travail...

Pour maintenir l'écran quelques temps, nous allons utiliser la fonction **SDL_Delay()** qui prend en paramètre un temps en millisecondes

SDL_Delay
```
//On attend 2 secondes
SDL_Delay( 2000 );
```

Dans l'exemple ci-dessus, l'écran restera affiché pendant 2000 millisecondes soit 2 secondes.

Pour finir, nous allons terminer notre programme proprement.

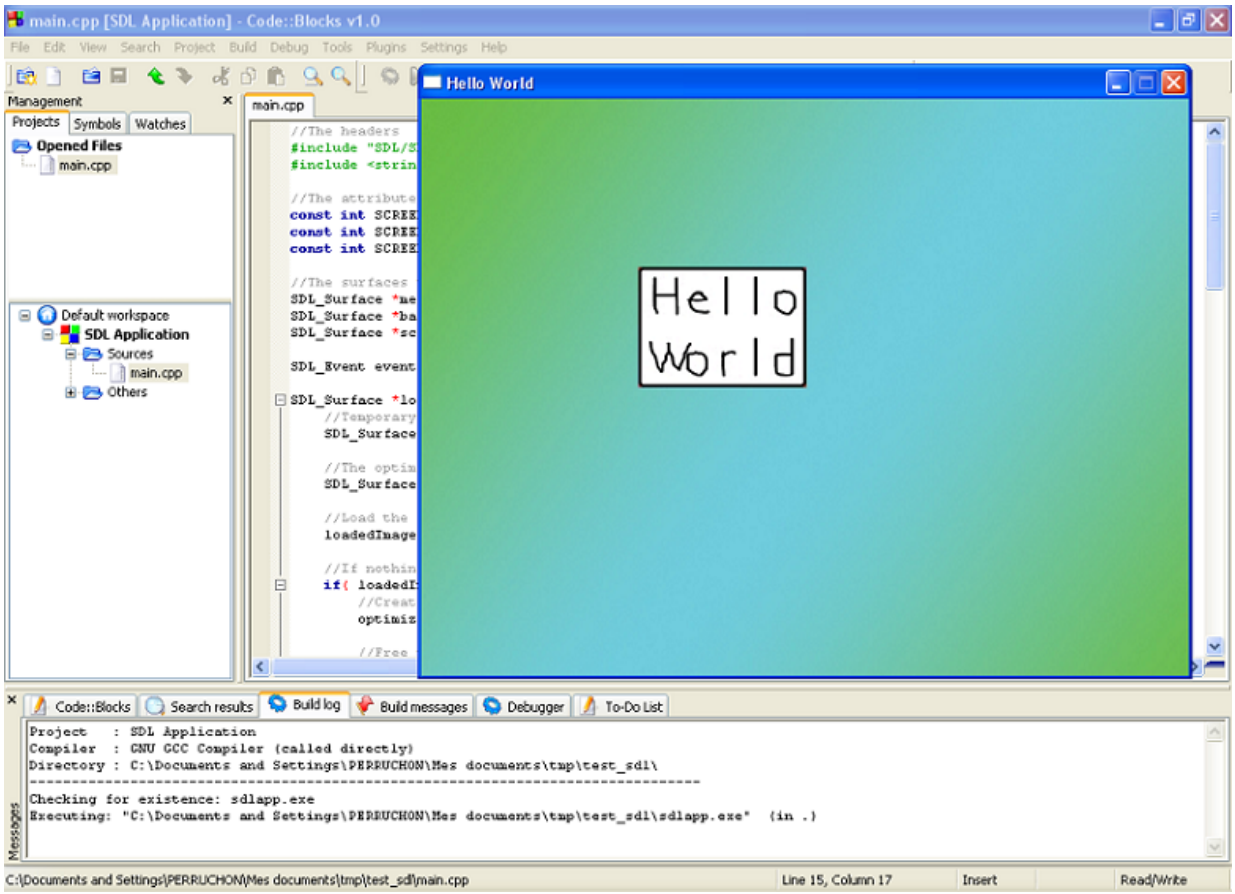
SDL_Delay
```
            //Libération des surfaces
            SDL_FreeSurface( message );
            SDL_FreeSurface( background );

            //On quitte SDL
            SDL_Quit();
    }
```

SDL_FreeSurface() est utilisé pour libérer la surface que nous avons chargé et que nous n'utiliserons plus.

Si vous ne libérez pas la mémoire utilisé, cela provoquera une fuite mémoire.

Ensuite nous appelons **SDL_Quit()** pour quitter SDL. Il nous reste à retourner 0 pour terminer l'application.

Voila ce que nous obtenons à la fin :



Remarque :

Si vous lancez le programme, que les images ne se voient pas, que l'écran de l'application n'apparaît que très brièvement et que vous trouvez dans le fichier stderr.txt : **Fatal signal: Segmentation Fault (SDL Parachute Deployed)**, c'est parce que le programme tente d'accéder à un espace mémoire qui n'est pas supposé être.

Ce phénomène étrange est du au fait qu'il essaye d'accéder à NULL quand apply_surface() est appelé.

Ce qui veut dire que vous devez vérifier que vos fichiers image Bitmap soient bien dans le même répertoire que votre programme.

Version pdf 

Télécharger les sources

<= Retour au chapitre I : Installation de SDL

-=Index Tutos SDL=-

=> Accéder au chapitre III : SDL_TTF