Chapitre VII : Jouer du son

par Loka

Date de publication : 01/05/2006

Dernière mise à jour : 01/05/2006

Le son est l'une des caractéristiques essentielles des applications d'aujourd'hui. Voyons comment SDL se débrouille de ce côté là.

VII - Jouer du son
- VII-A - Première approche
  - VII-A-1 - Initialisation audio de SDL
  - VII-A-2 - Initilaliser la couche audio
  - VII-A-3 - Charger un échantillon au format WAV
  - VII-A-4 - Jouer un échantillon audio au format WAV
  - VII-A-5 - Convertir des échantillons
- VII-B - Premier programme avec SDL_mixer
- VII-C - Jouer les pistes Audio d'un cd
  - VII-C-1 - Accéder et utiliser un lecteur de cd-rom
  - VII-C-2 - Jouer les pistes d'un cd audio
- Remerciements

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

VII - Jouer du son

VII-A - Première approche

La bibliothèque de base SDL permet de lire facilement les fichiers au format WAV, mais uniquement ceux-ci.

Afin de jouer d'autres formats de son avec SDL, il existe deux extensions :

- SDL_sound
- SDL_mixer

Les deux permettent à peu près la même chose.

L'avantage de SDL_mixer sur SDL_sound est le fait de pouvoir facilement jouer plusieurs sons en même temps.

SDL_mixer est un module de plus haut niveau que SDL pour le son, cependant à première vue ce module n'est pas d'une utilisation très simple.

La raison est simple, SDL_mixer utilise deux approches différentes pour travailler sur les fichiers sonores : music et mixer.

En fait dans le premier cas, vous chargez un fichier audio qui occupera le périphérique à lui seul tant qu'il sera actif, ce qui peut-être dérangeant lorsque l'on désire jouer plusieurs sons en même temps.

C'est là que la partie mixer intervient, dans ce cas vous définissez un format (fréquence d'échantillonnage...) pour le périphérique, puis vous pouvez charger et jouer plusieurs sons simultanément sur celui-ci, par le système des canaux.

Attention si le format est trop différent du fichier original, vous obtiendrez des effets étranges et nuisibles.

Donc pour résumer, on va faire un peu d'anglais et lire le Readme de SDL_mixer : *"Due to popular demand, here is a simple multi-channel audio mixer. It supports 8 channels of 16 bit stereo audio, plus a single channel of music, mixed by the popular MikMod MOD, Timidity MIDI and SMPEG MP3 libraries"*.

En gros cela dit que SDL_mixer supporte 8 canaux sur 16 bits et un canal pour le type music.

Donc on peut mélanger simultanément mixer et music.

Nous allons dans cette partie apprendre à utiliser les fonctions définies dans **SDL_audio.h**.

VII-A-1 - Initialisation audio de SDL

Afin d'initialiser correctement la bibliothéque, il faudra tout d'abord veiller à appeler la fonction SDL_Init() avec le flag **SDL_INIT_AUDIO** :

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

SDL_Init
```
SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO);
```

Ceci est fait bien évidemment lorsque nous appelons SDL_Init avec le flag **SDL_INIT_EVERYTHING** qui nous permet d'initialiser tous les sous-systèmes de SDL.

L'initialisation de SDL avec le flag **SDL_INIT_AUDIO** a pour effet de démarrer le thread chargé de la gestion audio.

Pour ouvrir le périphérique proprement dit, il faut recourir à la fonction :

```
SDL_OpenAudio(SDL_AudioSpec *desired, SDL_AudioSpec *obtained);
```

Cette fonction prend en argument deux structures **SDL_AudioSpec** qui permettent de décrire le format audio désiré.

La structure pointée par *desired* doit être remplie de la manière suivante :

SDL_AudioSpec
```
typedef struct {
    int freq;
    Uint16 format;
    Uint8  channels;
    Uint8  silence;
    Uint16 samples;
    Uint32 size;
    void (*callback)(void *userdata, Uint8 *stream, int len);
    void  *userdata;
} SDL_AudioSpec;
```

- *freq* indique la fréquence en échantillons par seconde (par exemple, 44100 pour une qualité CD).

- *format* indique le format des échantillons, parmi les valeurs décrites ci-dessous :

- AUDIO_U8 : Echantillons 8 bits non signés,
- AUDIO_S8 : Echantillons 8 bits signés,
- AUDIO_U16LSB : Echantillons 16 bits non signés (non encore supportés, car peu répandus),
- AUDIO_S16LSB : Echantillons 16 bits signés (les plus répandus),
- AUDIO_U16MSB, AUDIO_S16MSB : idem que ci-dessus pour les architectures big endian (pratiquement tout ce qui n'est pas Intel, comme les PowerPC ou les Alpha).

- *channels* le nombre de canaux : 1 pour un son mono, 2 pour la stéréo.

- *silence* la valeur de silence pour le tampon (attention, il est calculé par SDL, vous ne devez pas le remplir !).

- *sample* la taille du tampon audio ; il ne s'agit pas de la taille maximale des échantillons que vous pourrez charger par la suite, mais plutôt de la taille du tampon utilisé de manière interne. D'une façon générale, un grand tampon demandera moins de temps CPU mais introduira un décalage plus grand entre l'arrivée des échantillons et la sortie sur les haut-parleurs. D'un autre côté une taille de tampon trop petite peut entraîner des déformations du signal audio sur les machines peu puissantes, qui auraient du mal à suivre.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

- *size* la taille en octets du tampon, calculée automatiquement par SLD_OpenAudio().

- **callback* : l'adresse d'une fonction appelée automatiquement par SDL à chaque fois que le tampon est vide. Cette fonction est chargée de remplir à nouveau le tampon.

- **userdata* : cette valeur sera passée comme premier argument de la fonction callback décrite ci-dessus. Sa signification est complètement laissée à la charge de l'utilisateur.

Il est possible que le format audio obtenu soit différent de celui demandé, notamment en cas de contraintes matérielles.

C'est pourquoi il faut toujours vérifier le contenu de la structure pointée par *obtained* qui décrit les caractéristiques du format audio effectivement alloué.

Il est également possible de forcer un format audio, en appelant la fonction avec un second argument NULL.

Dans ce cas, SDL fera son possible pour émuler le format audio en fonction des contraintes matérielles, si cela s'avère nécessaire.

Il est important de retenir que l'architecture audio de SDL repose sur des fonctions callbacks.

Il n'y a pas de fonction simple pour jouer un son dans l'API (bien que cela puisse être implanté relativement aisément).

Au lieu de cela, il faut écrire une fonction chargée de fournir à la demande de SDL les échantillons du son, par tranche correspondant à la taille du tampon audio alloué (le champ *samples* de la structure SDL_AudioSpec)...

Pour permettre à l'utilisateur d'initialiser correctement toutes les structures avant que SDL ne se mette à appeler des fonctions callback, l'appel à SDL_OpenAudio() met initialement SDL en mode "pause".

La restitution et donc le remplissage des tampons audio ne commencent que lorsque le programme appelle explicitement SDL_PauseAudio(0), ce qui a pour effet de sortir du mode pause.

Il est possible d'utiliser les fonctions SDL_LockAudio() et SDL_UnlockAudio() pour garantir que la fonction callback n'est pas appelée dans la thread audio (exclusion mutuelle).

Cela peut être utile si vous devez fournir des données audio depuis une thread séparée (i.e. en dehors de la fonction callback qui est toujours exécutée dans la thread audio de SDL)...

VII-A-2 - Initilaliser la couche audio

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Pour initialiser la couche audio de SDL, il va vous falloir renseigner les propriétés de l'objet **SDL_AudioSpec** vu ci-dessus.

Créons donc une fonction d'initialisation de la couche sonore :

audio_Init
```
SDL_AudioSpec audioSortie;
void audioCallback(void *udata, Uint8 *stream, int len);

int audio_Init(void)
{
    // Définition des propriétés audio
    audioSortie.freq = 22050;
    audioSortie.format = AUDIO_S16;
    audioSortie.channels = 2;
    audioSortie.samples = 1024;
    audioSortie.callback = audioCallback;
    audioSortie.userdata = NULL;

    // Initialisation de la couche audio
    if (SDL_OpenAudio(&audio, NULL) < 0)
    {
        fprintf(stderr, "Erreur d'ouverture audio: %s\n", SDL_GetError());
        return (-1);
    }

    return 0;
}
```

On commence par déclarer l'objet **SDL_AudioSpec** qui servira de conteneur à toutes les propriétés utilisées pour le son.

On déclare également une fonction dite "callback" (nous verrons cette partie plus en détail par la suite), après quoi on attaque la fonction d'initialisation.

Dans cette fonction on se contente de définir les propriétés audio de l'objet audioSortie, puis d'initialiser la couche audio avec SDL_OpenAudio() qui prend en argument les spécifications audio désirées et une adresse vers celles réellement allouées (nous n'utiliserons pas ce dernier argument ici).

Les différentes propriétés de audioSortie possibles sont expliquées dans la partie 1 ci-dessus.

VII-A-3 - Charger un échantillon au format WAV

SDL fournit une fonction pour charger un fichier son au format WAV en mémoire.

Actuellement, seuls les fichiers WAV de base sont supportés (données brutes ou au format compressé ADPCM).

Voici la fonction en question :

```
SDL_AudioSpec *SDL_LoadWAV(const char *file, SDL_AudioSpec *spec, Uint8 **audio_buf, Uint32
*audio_len);
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Son utilisation est la suivante :

file est le chemin d'accès du fichier à charger.

On fournit l'adresse d'une structure SDL_AudioSpec qui sera remplie par la fonction avec les données décrivant le format du fichier chargé (cette adresse sera également retournée par la fonction).

La fonction se charge d'allouer un tampon en mémoire et d'y charger le son.

Les arguments *audio_buf* et *audio_len* sont retournés par SDL, afin d'indiquer la longueur et l'adresse dudit tampon.

En cas d'erreur, la fonction renvoie **NULL**.

Le tampon alloué doit, par la suite, être explicitement libéré par l'utilisateur à l'aide de la fonction **SDL_FreeWAV()**.

Voici un exemple de chargement d'échantillon au format WAV :

chargement WAV
```
SDL_AudioSpec audioBufferSpec;
Uint8 *audioBuffer;
Uint32 audioBufferLen;

// A inclure dans votre code pour charger un fichier sonore dans audioBuffer
if(!SDL_LoadWAV("essai.wav", &audioBufferSpec,
    &audioBuffer, &audioBufferLen))
{
    printf("Erreur lors du chargement du fichier WAV.\n");
    return 1;
}
        ...
SDL_FreeWAV(&audioBuffer);
```

Dans ce code nous utilisons un tampon Uint8 *audioBuffer et sa taille Uint32 audioBufferLen.

Maintenant que nous savons charger un fichier WAV, il va nous falloir apprendre à le jouer, ceci est le sujet de la prochaine partie.

VII-A-4 - Jouer un échantillon audio au format WAV

La fonction callback, que nous avons décrite légérement plus haut, est la fonction qui est appellée à chaque cycle de votre programme pour remplir le buffer audio.

En résumé, elle contient toutes les instructions à effectuer à chaque passage dans votre code pour gérer le son.

Une fois le son initialisé avec le code de la partie ci-dessus, et un fichier WAV chargé dans le buffer audio, nous

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

allons utiliser le code ci-dessous pour lire le fichier :

audioCallBack
```
Uint32 audioLen, audioPos;

void audioCallBack(void *udata, Uint8 *stream, int len)
{
    // On ne lit que s'il reste des données à jouer
    if ( audioLen == 0 )
        return;

    // Remise à zéro du tampon de sortie
    memset(stream, 0, len);

    // Lecture du buffer audio
    if (audioPos < audioBufferSpec.len) {
        if (audioPos+len > audioBufferSpec.len)
            len = audioBufferSpec.len = audioPos;
        SDL_MixAudio(stream, audioBuffer + audioPos,
            len, SDL_MIX_MAXVOLUME);
        audioPos += len;
    }

    // Décrémentation de ce qu'il reste à lire
    audioLen -= len;
}
```

A savoir tout d'abord, la fonction **SDL_MixAudio()** est **la** fonction qui permet de lire vos buffers audios.

Ici, *audioLen* correspond à la longueur de piste restante à jouer (s'il ne reste rien à jouer on quitte).

stream correspond au tampon audio de sortie (ce qui sera envoyé à la carte son en gros) et *len* la taille du tampon.

On remet ce tampon à zéro avant de le remplir avec les données sonores à jouer.

En effet, à chaque cycle de votre programme seul un octet (Uint8) est ici envoyé au matériel audio, et la suite de ces octets joués par la carte formera le son entendu par l'utilisateur.

Donc, si *audioLen* est inférieure à la longueur du buffer à jouer on envoie le son (si ce n'est pas le cas il y a un problème), et ce avec la fonction **SDL_MixAudio()**.

Cette fonction prend en argument le tampon de sortie, l'octet à envoyer (*audioBuffer* est un pointeur sur un Uint8, décalé de *audioPos* pour avoir l'octet actuel à jouer), la longueur du buffer et le volume.

Ce dernier est compris entre **0** et **SDL_MIX_MAXVOLUME**, et vous pourrez le faire varier, par exemple, en pourcentage :

```
75% du volume = (0.75 * SDL_MIX_MAXVOLUME)
```

Une fois le son joué on incrémente la position dans le buffer audio de la taille de la séquence jouée (qui est celle du tampon).

Tout ceci est assez complexe c'est vrai mais dites vous que ça reste plus simple que du directMusic.

Une fois que le buffer audio est lu, on décrémente la longueur d'échantillon à jouer de la longueur jouée dernièrement.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Tout cela est bien joli, on a maintenant un module de son initialisé, un fichier WAV chargé et la fonction adéquate pour lire ce fichier, il nous reste donc à lancer la lecture.

lecture
```
// Lancement de la lecture
SDL_PauseAudio(0);

// Attendre que la lecture du son soit terminée
while ( audioLen > 0 )
    SDL_Delay(100);

// Fermeture du module
SDL_CloseAudio();
```

SDL_PauseAudio() est la fonction servant à mettre ou à enlever la pause. En effet, de base le son est sur pause, passer 0 en argument à cette fonction relance la lecture, 1 réenclenchant la pause.

N'oubliez pas de terminer le processus audio par **SDL_CloseAudio()** à la fin de votre code.

VII-A-5 - Convertir des échantillons

L'un des aspects les plus puissants de la partie audio de SDL est la possibilité de convertir aisément des échantillons d'un format à l'autre, par l'intermédiaire de blocs de conversion audio.

Concrètement, un bloc de conversion audio est un objet **SDL_AudioCVT**, dont la structure est la suivante :

SDL_AudioCVT
```
typedef struct SDL_AudioCVT {
         int needed;         // Valeur non-nulle si la conversion est possible
         Uint16 src_format;  // Format audio source
         Uint16 dst_format;  // Format audio cible
         double rate_incr;   // Rapport de conversion pour la fréquence
         Uint8 *buf;         // Tampon contenant les données audio
         int    len;         // Longueur du tampon d'origine ci-dessus
         int    len_cvt;     // Longueur du tampon de destination
         int    len_mult;    // Coefficient: le tampon doit avoir une taille de len*len_mult
         double len_ratio;   // Rapport de taille entre le tampon original et le nouveau
         void (*filters[10])(struct SDL_AudioCVT *cvt, Uint16 format);
         int filter_index;   // Filtre de conversion en cours
 } SDL_AudioCVT;
```

Rassurez-vous, une bonne partie des champs de cette structure est remplie pour vous par SDL.

Connaissant les caractéristiques des formats d'origine et de destination, il faut faire appel à la fonction **SDL_BuildAudioCVT()** afin de préparer le bloc de conversion :

```
int SDL_BuildAudioCVT(SDL_AudioCVT *cvt, Uint16 src_format, Uint8 src_channels,
                      int src_rate, Uint16 dst_format, Uint8 dst_channels, int dst_rate);
```

Son utilisation est simple :

On fournit l'adresse d'un objet **SDL_AudioCVT** non initialisé, ainsi que les caractéristiques des formats audio : *format* (voir les constantes définies dans la partie 1), nombre de canaux (1 ou 2 pour mono ou stéréo) et fréquence d'échantillonnage.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

La fonction renvoie une valeur nulle, si tout s'est bien déroulé.

Le bloc de conversion ainsi obtenu n'est cependant pas complet.

Avant de pouvoir effectuer la conversion par le biais de la fonction **SDL_ConvertAudio()**, il faut remplir les champs *buf* et *len* de la structure.

Attention, la taille effective du tampon est souvent supérieure à la taille des données d'origine, SDL effectuant la conversion in situ.

La marche à suivre est donc d'allouer un tampon (par exemple, à l'aide de malloc) d'une taille de *len* (la taille des données d'origine) multipliée par *len_mult* (fourni par **SDL_BuildAudioCVT**), puis de charger les données au début du tampon ainsi alloué.

Il est à noter que la valeur de *len* doit bien être la taille des données d'origine, sans tenir compte du facteur *len_mult*.

La conversion des données proprement dite est ensuite fort simple.

Il suffit de passer l'objet **SDL_AudioCVT** ainsi construit à la fonction **SDL_ConvertAudio()**.

Maintenant vous souhaitez pouvoir jouer des échantillons sonore d'un autre format, nous allons voir cela dans la partie suivante en utilisant l'extension SDL : **SDL_mixer**

VII-B - Premier programme avec SDL_mixer

Comme nous l'avons vu, SDL_mixer est une extension de SDL permettant une utilisation plus aisé des fichiers sonores ainsi que la lecture de formats audio autre que WAV.

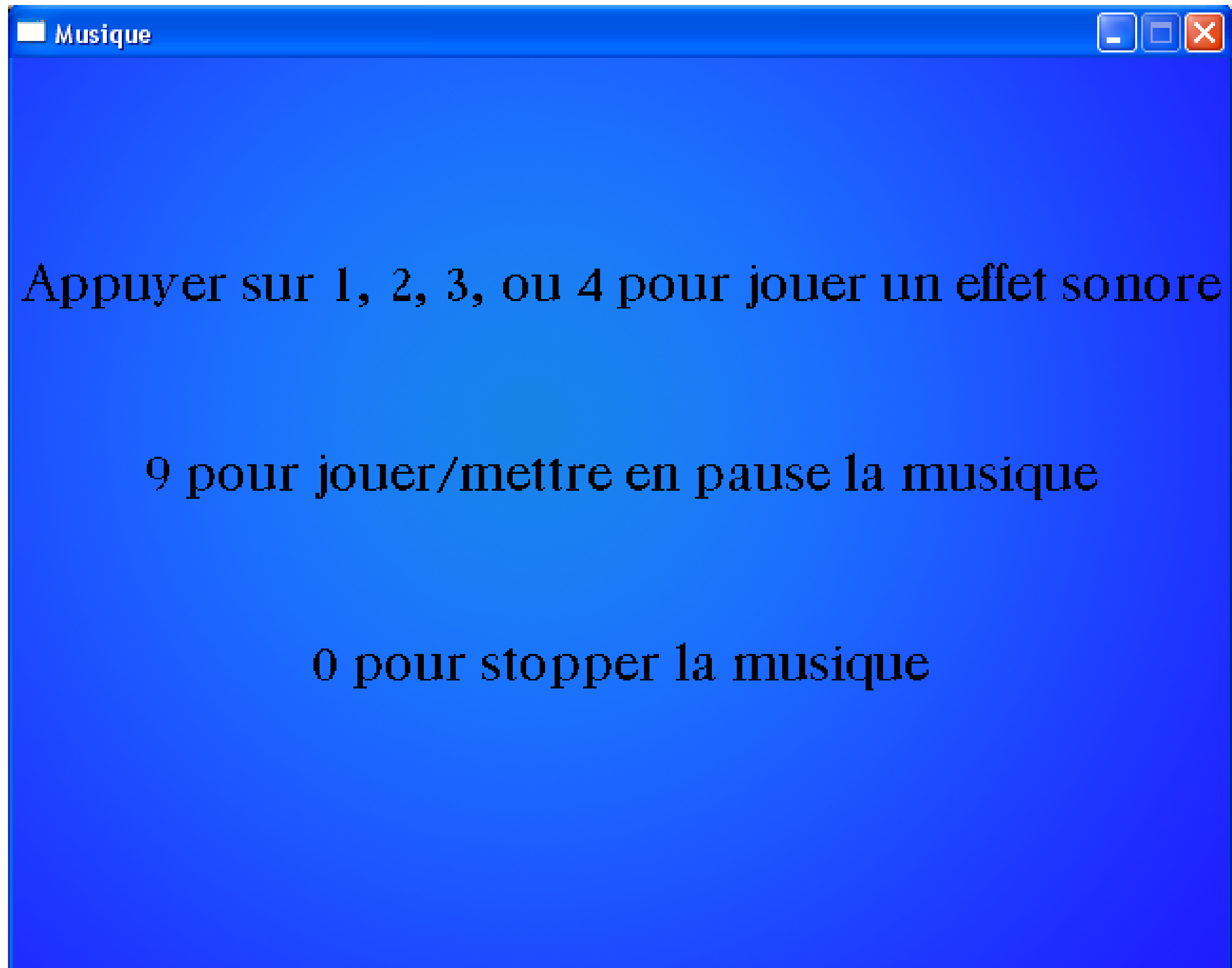
Vous pouvez télécharger SDL_mixer ici

L'installation est très simple, n'oubliez pas de rajouter la librairie SDL_mixer dans votre linker.

Nous apprendrons à jouer des effets sonores, jouer une musique et avoir la possibilité de la mettre en pause voir de la stopper en utilisant SDL_mixer.

Au final nous aurons quelquechose de ce genre :

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.



Commençons notre programme par la déclaration de nos variables :

music variables

```
//La musique qui sera jouée
Mix_Music *music = NULL;

//Les effets sonores que nous allons utiliser
Mix_Chunk *scratch = NULL;
Mix_Chunk *high = NULL;
Mix_Chunk *med = NULL;
Mix_Chunk *low = NULL;
```

Voici donc les nouvelle données avec lesquelles nous allons travailler.

Mix_Music est le type de données pour la musique et Mix_Chunk celui pour les effets sonores.

Nous allons maintenant retoucher notre fonction d'initialisation *init()* afin d'initialiser les fonctions audio de SDL_mixer.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

init

```
bool init()
{
    //Initialisation de tous les sous-systèmes de SDL
    if( SDL_Init( SDL_INIT_EVERYTHING ) == -1 )
    {
        return false;
    }

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

    //S'il y a eu une erreur lors de la mise en place d el'écran
    if( screen == NULL )
    {
        return false;
    }

    //Initialisation de SDL_TTF
    if( TTF_Init() == -1 )
    {
        return false;
    }

    //Initialisation de SDL_mixer
    if( Mix_OpenAudio( 22050, MIX_DEFAULT_FORMAT, 2, 4096 ) == -1 )
    {
        return false;
    }

    //Mise en place de la barre caption
    SDL_WM_SetCaption( "Musique", NULL );

    //Si tout s'est bien passé
    return true;
}
```

Comme vous le voyez, nous appelons **Mix_OpenAudio()** pour initialiser les fonctions audio de SDL_mixer.

Le premier argument de **Mix_OpenAudio()** est la fréquence du son que nous allons utiliser.

Le second argument est le format du son utilisé que nous mettons par défaut (MIX_DEFAULT_FORMAT).

Le troisième argument est le nombre de channels que nous souhaitons utiliser (ici 2 : stéréo).

Le dernier argument est une taille d'échantillon, que nous mettons à 4096.

Après avoir changer notre fonction **init()**, il va nous falloir changer notre fonction **load_files()** afin de charger la musique et les effets sonores.

load_files

```
bool load_files()
{
    //Chargement du fond
    background = load_image( "background.png" );

    //Ouverture du Font
    font = TTF_OpenFont( "CaslonBold.ttf", 30 );

    //S'il y a eu un problème au chargement du fond
    if( background == NULL )
    {
        return false;
    }
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

load_files
```
        //S'il y a eu un problème à l'ouverture du Font
        if( font == NULL )
        {
            return false;
        }

        //Chargement de la musique
        music = Mix_LoadMUS( "beat.wav" );

        //S'il y a eu une erreur au chargement de la musique
        if( music == NULL )
        {
            return false;
        }

        //Chargement des effets sonores
        scratch = Mix_LoadWAV( "scratch.wav" );
        high = Mix_LoadWAV( "high.wav" );;
        med = Mix_LoadWAV( "medium.wav" );;
        low = Mix_LoadWAV( "low.wav" );;

        //S'il y a eu un problème au chargement des effets sonore
        if( ( scratch == NULL ) || ( high == NULL ) || ( med == NULL ) || ( low == NULL ) )
        {
            return false;
        }

        //Si tout s'est bien chargé
        return true;
    }
```

Pour charger la musique, nous utilisons **Mix_LoadMUS()**.

Mix_LoadMUS() prend le nom du fichier de musique et retourne les données de la musique.

La fonction retourne NULL s'il y a une erreur.

Pour charger les effets sonores, nous utilisons **Mix_LoadWAV()**.

Cette fonction charge le fichier son (*.wav) dont le nom est passé en argument et retourne une donnée de type **Mix_Chunk** ou NULL s'il y a une erreur.

Il nous reste à changer notre fonction de nettoyage et nous pourrons passer au main.

nettoyage
```
    void clean_up()
    {
        //Libération des surfaces
        SDL_FreeSurface( background );

        //Libération des effets sonores
        Mix_FreeChunk( scratch );
        Mix_FreeChunk( high );
        Mix_FreeChunk( med );
        Mix_FreeChunk( low );

        //Libération de la musique
        Mix_FreeMusic( music );

        //Fermeture du Font
        TTF_CloseFont( font );

        //On quitte SDL_mixer
        Mix_CloseAudio();
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

nettoyage

```
        //On quitte SDL_ttf
        TTF_Quit();

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

Nous faisons appel à **Mix_FreeChunk()** pour se débarrasser des effets sonores et **Mix_FreeMusic()** pour libérer la musique.

Ensuite, une fois que nous en avons fini avec SDL_mixer, nous appelons **Mix_CloseAudio()**.

Dans notre fonction main, après avoir initialisé, chargé les fichiers, que le fond et les messages ont bien été *blittés*, nous entrons dans notre boucle principale.

effets sonores

```
        //Tant que l'utilisateur n'a pas quitté
        while( quit == false )
        {
            //Tant qu'il y a un événement
            while( SDL_PollEvent( &event ) )
            {
                //Si une touche a été pressée
                if( event.type == SDL_KEYDOWN )
                            {
                    //Si 1 a été préssé
                    if( event.key.keysym.sym == SDLK_1 )
                    {
                        //On joue l'effet scratch
                        if( Mix_PlayChannel( -1, scratch, 0 ) == -1 )
                        {
                            return 1;
                        }
                    }
                    //Si 2 a été préssé
                    else if( event.key.keysym.sym == SDLK_2 )
                    {
                        //On joue l'effet hight
                        if( Mix_PlayChannel( -1, high, 0 ) == -1 )
                        {
                            return 1;
                        }
                    }
                    //Si 3 a été préssé
                    else if( event.key.keysym.sym == SDLK_3 )
                    {
                        //On joue l'effet med
                        if( Mix_PlayChannel( -1, med, 0 ) == -1 )
                        {
                            return 1;
                        }
                    }
                    //Si 4 a été préssé
                    else if( event.key.keysym.sym == SDLK_4 )
                    {
                        //On joue l'effet low
                        if( Mix_PlayChannel( -1, low, 0 ) == -1 )
                        {
                            return 1;
                        }
                    }
```

Lorsque nous vérifions si une touche a été pressée, nous vérifions dans un premier temps si 1,2,3 ou 4 ont été pressé.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Ce sont les touches pour jouer les effets sonores dans notre programme.

Si une de ces touches a été pressée, nous faisons appel à **Mix_PlayChannel()** pour jouer l'effet sonore associé à la touche pressée.

Le premier argument de **Mix_PlayChannel()** est le canal dans lequel nous allons jouer le son.

Vu que nous l'avons mis à -1, **Mix_PlayChannel()** regardera juste le premier channel valide qu'il trouvera et jouera le son.

Le second argument est le Mix_Chunk que l'on va jouer.

Le troisième argument est le nombre de fois que l'effet sonore sera joué en boucle.

Donc si nous le mettons à 0, l'effet sonore sera joué qu'une seule fois.

Quand il y a un problème, **Mix_PlayChannel()** retourne -1.

Passons à la musique maintenant :

musique non jouée

```
            //Si 9 a été pressé
            else if( event.key.keysym.sym == SDLK_9 )
            {
                //S'il n'y a pas de musique jouée
                if( Mix_PlayingMusic() == 0 )
                {
                    //On lance la musique
                    if( Mix_PlayMusic( music, -1 ) == -1 )
                    {
                        return 1;
                    }
                }
```

Nous vérifions si la touche 9 a été pressée, ce qui devrait avoir pour effet de jouer/mettre en pause la musique.

Premièrement nous vérifions si la musique est jouée avec **Mix_PlayingMusic()**.

Si la musique n'est pas jouée, nous appelons **Mix_PlayMusic()** afin de la jouer.

Le premier argument de **Mix_PlayMusic()** est la musique que nous allons jouer.

Le second argument est le nombre de fois que la musique sera jouée en boucle.

Ici, le fait de mettre -1 aura l'effet de jouer la musique en boucle jusqu'à ce qu'elle soit stoppée manuellement.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.
http://loka.developpez.com/tutoriel/sdl/sons/

S'il y a un problème, **Mix_PlayMusic()** retourne -1.

Le code suivant va nous permettre de réagir si la musique est déjà jouée :

musique jouée
```
                //Si la musique est déjà lancée
                else
                {
                    //Si la musique est en pause
                    if( Mix_PausedMusic() == 1 )
                    {
                        //On enlève la pause (la musique repart où elle en était)
                        Mix_ResumeMusic();
                    }
                    //Si la musique est en train de jouer
                    else
                    {
                        //On met en pause la musique
                        Mix_PauseMusic();
                    }
                }
            }
```

Nous vérifions si la musique est en pause grâce à **Mix_PausedMusic()**.

Si c'est le cas, nous relançons la musique avec **Mix_ResumeMusic()** (la musique sera jouée non pas depuis le debut, mais depuis le moment où elle s'est *"pausée"*).

Si la musique n'est pas en pause, alors nous la mettons en pause avec **Mix_PauseMusic()**.

Il nous reste la possibilité de stopper la musique :

stop music
```
            //Si 0 a été pressé
            else if( event.key.keysym.sym == SDLK_0 )
            {
                //On stoppe la musique
                Mix_HaltMusic();
            }
```

Nous vérifions donc si 0 a été pressé.

Si c'est le cas, la musique est stoppée avec l'utilisation de **Mix_HaltMusic()**.

Pour conclure cette partie, je vous recommande de télécharger et de garder avec vous cette documentation : SDL_mixer documentation

VII-C - Jouer les pistes Audio d'un cd

L'accès aux périphériques manque cruellement dans la plupart des bibliothèques multimedia disponibles.

Qu'à cela ne tienne, SDL vous propose d'accéder sans problème au(x) lecteur(s) de CD-ROM.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

VII-C-1 - Accéder et utiliser un lecteur de cd-rom

Vous pouvez savoir combien de lecteurs de CD-ROM existent sur les systèmes grâce à la fonction **SDL_CDNumDrives()**, et accéder à l'un d'entre eux avec **SDL_CDOpen()**.

Le lecteur 0 est le lecteur par défaut sur le système. Le lecteur de CD-ROM peut être ouvert même s'il ne contient aucun disque.

La fonction **SDL_CDStatus()** permet de déterminer l'état du lecteur de CD-ROM. Après avoir utilisé le lecteur vous devez le fermer avec **SDL_CDClose()**.

Vous pouvez récupérer le nom du lecteur de CD-ROM au sein du système, en utilisant le fonction **SDL_CDName()**.

Voici un exemple de code permettant d'accéder à un lecteur cd-rom :

Accés lecteur cd-rom

```
    SDL_CD *cdrom;

    if ( SDL_CDNumDrives() > 0 ) {
        cdrom = SDL_CDOpen(0);
        if ( cdrom == NULL ) {
            fprintf(stderr, "Impossible d'ouvrir le lecteur de CD-ROM par défaut %s\n"
SDL_GetError());
            return;
        }

        ...

        SDL_CDClose(cdrom);
    }
```

Comme vous le voyez, c'est plutôt simple.

Nous allons donc voir un programme complet permettant de lister et d'ouvrir les lecteurs de cd-rom.

Pour accéder aux lecteurs CD avec SDL, on passe par les objets SDL_CD.

lister et ouvrir les lecteurs cd-rom

```
#include <SDL.h>

// Objet CD Audio
SDL_CD *AudioCD = NULL;

int main(void)
{
    int iNbrLecteursCD = 0;

    // Initialisation de SDL_CDROM
    if (SDL_Init(SDL_INIT_CDROM) < 0)
        return 0;

    // Récupération du nombre de lecteurs
    iNbrLecteursCD = SDL_CDNumDrives();
    printf("Nombre de lecteurs CD : %d\n",iNbrLecteursCD);

    // Nom du lecteur 0
    printf("Ouverture du lecteur : '%s'\n",
        SDL_CDName(0));
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

lister et ouvrir les lecteurs cd-rom

```
        // Ouverture du lecteur 0
        AudioCD = SDL_CDOpen(0);
        if (AudioCD == NULL) {
            printf("Echec de l'ouverture du lecteur : %s\n",
                SDL_GetError());
            return 0;
        }

        // Lecture des pistes

        // Fermeture du CD
        if (AudioCD!=NULL)
            SDL_CDClose(AudioCD);

        return 0;
    }
```

Dans la fonction principale, on déclare une variable iNbrLecteursCD qui contiendra le nombre de lecteurs CD disponibles.

A partir de cela vous pourrez, plus tard, faire choisir à l'utilisateur quel lecteur il compte utiliser.

On initialise ensuite la partie CD-ROM de SDL et avec la partie **TIMER** (que nous allons voir au chapitre suivant).

En effet, la lecture de CD Audio se fait en arrière plan, d'où le besoin d'utiliser les timers (chapitre VIII) et le multithreading (chapitre XVIII).

Ceci fait, on récupère le nombre de lecteurs disponibles dans iNbrLecteursCD grâce à la fonction **SDL_CDNumDrives()** qui retourne un entier.

On considère ensuite que l'utilisateur a choisi d'ouvrir le lecteur 0.

Tout d'abord on récupère son nom, ce qui est fait facilement via la fonction **const char* SDL_CDName()** qui prend en paramètre le numéro du lecteur et qui retourne une chaîne de caractères (le nom du lecteur).

Une fois le nom affiché en sortie on ouvre le lecteur CD dans l'objet AudioCD via la fonction **SDL_CD SDL_CDOpen()** qui prend en paramètre le numéro du lecteur.

Désormais, le lecteur CD est ouvert et on peut accèder aux pistes audio de celui-ci via l'objet AudioCD (que nous verrons plus en détail dans la partie suivante).

Ne pas oublier de fermer le lecteur CD avec la fonction **void SDL_CDClose(SDL_CD lecteur_a_fermer)** avant la fin de l'exécution du programme.

Maintenant que le lecteur est ouvert et initialisé nous allons apprendre à l'utiliser.

VII-C-2 - Jouer les pistes d'un cd audio

Avant de commencer il nous faut connaître la structure d'une piste audio dans SDL, nommée **SDL_CDtrack**.

Elle comporte les champs suivants :

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

struct CDtrack

```
typedef struct {
    Uint8 id;
    Uint8 type;
    Uint32 lenght;
    Uint32 offset;
} SDL_CDtrack;
```

id contient le numéro de la piste et le type de données (audio, data...).

Dans notre cas il faudra que le type soit égal à **SDL_AUDIO_TRACK**, dans le cas contraire le type serait **SDL_DATA_TRACK**.

lenght contient la longueur de la piste en frames.

En gros, une frame fait 2 ko, mais pour obtenir la longueur de la piste en secondes vous pouvez utiliser la constante **CD_FPS** de SDL.

Ainsi, la durée d'une piste en secondes est égale à la longueur de la piste en frame divisé par la constante CD_FPS (DureePisteSecondes = SDL_CDtrack.lenght/CD_FPS;).

offset contient le numéro de frame où est positionné le début de la piste, ce qui nous servira à déterminer le début et la fin de chaque piste.

Nous allons maintenant tenter de lire une piste audio, mais avant toute chose il faudra s'assurer que le lecteur est prêt à la lire.

Pour cela on utilise les objets de type **CDstatus** et la fonction **CDStatus SDL_CDStatus(SDL_CD *cdrom)**, qui renvoie les status suivants :

- CD_TRAYEMPTY : pas de CD dans le lecteur
- CD_STOPPED : aucune lecture en cours, il y a un CD dans le lecteur
- CD_PLAYING : lecture en cours
- CD_PAUSED : lecture en cours et mise en pause
- CD_ERROR : erreur de lecture du status

Vous pouvez aussi utiliser la macro **CD_INDRIVE** qui prend comme paramètre le status du lecteur CD renvoyé par **SDL_CDStatus**.

Elle renvoie 0 si le lecteur est vide.

Désormais nous pouvons lire une piste, voyons voir la fonction suivante :

PlayCDTrack

```
void PlayCDTrack(SDL_CD *lecteur, int piste)
{
    if (CD_INDRIVE(SDL_CDStatus(lecteur)) == 0)
        printf("Le lecteur %s est vide.\n",SDL_CDName(lecteur->id));

    else if (lecteur->track[piste].type != SDL_AUDIO_TRACK)
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

PlayCDTrack

```
            printf("La piste n'est pas une piste audio!\n");

        else if (SDL_CDPlay(lecteur, lecteur->track[piste].offset, lecteur->track[piste].length) == 0)
            printf("Lecture de la piste %d en cours sur le lecteur %s.\n",
                piste, SDL_CDName(lecteur->id));

        else {
            printf("Erreur de lecture: %s",SDL_GetError());
            exit(0);
        }
    }
```

La fonction prend comme argument l'objet **SDL_CD** à lire et le numéro de la piste à lire.

Pour lire une piste on utilise la fonction SDL_CDPlay() qui prend comme arguments l'objet, l'offset de la piste à lire et sa longueur length (en CD_FPS, pour lire 10 secondes entrez CD_FPS*10 par exemple).

Notez que SDL possède deux fonctions pour jouer un CD.

Vous pouvez soit utiliser SDL_CDPlayTracks() pour jouer une piste précise, soit utiliser SDL_CDPlay() et jouer l'ensemble du CD.

Ici on lit toute la piste en partant du début.

Si le lecteur est vide on affiche un message d'erreur.

Sinon on joue la piste, et si la fonction SDL_CDPlay() ne renvoie pas la valeur 0 (en cas d'erreur) on affiche un message d'erreur avant de quitter.

Je précise que si la longueur de la piste à lire est supérieure à la longueur réelle de cette piste la piste suivante sera jouée.

Par exemple, si la piste 2 dure 2min30 et que vous entrez 2min50 en paramètre length à SDL_CDPlay() 20 secondes de la piste 3 seront jouées.

Il existe également cinq autres fonctions à connaître pour utiliser des CD Audio avec SDL :

- SDL_CDStop(SDL_CD *lecteur) Stoppe la lecture en cours sur le lecteur passé en argument.
- SDL_CDPause(SDL_CD *lecteur) Met la lecture du lecteur passé en argument en pause.
- SDL_CDResume(SDL_CD *lecteur) Reprend la lecture sur le lecteur passé en argument.
- SDL_CDEject(SDL_CD *lecteur) Ouvre le cadie du lecteur CD.
- SDL_CDClose(SDL_CD *lecteur) Ferme l'accès et libère les ressources allouées du descripteur de CD passé en argument.

Voilà, vous en savez désormais assez pour commencer à lire des CD AUdio dans vos programmes avec SDL.

N'oubliez pas de libérer les ressources allouées pour vos objets SDL_CD avec la fonction **SDL_CDClose()**.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.
http://loka.developpez.com/tutoriel/sdl/sons/

Remerciements

Je remercie **tastalian** de m'avoir permis d'utiliser une partie de son tutoriel qui m'avais beaucoup aidé à l'époque où j'ai commencé SDL.

Voici son site : http://www.tastalian.org

Version pdf (366 ko - 21 pages)

Télécharger les sources (553 ko)

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.