LA GESTION DE L'AUDIO EN SDL2

Introduction

Salutation ! après de longs jours/mois d'absence je vous annonce ce nouveau chapitre, ou nous allons pouvoir **jouer des sons/musiques dans nos programmes SDL**.

Souvenez-vous de la fonction `SDL_Init()`, vous savez maintenant qu'elle prend un paramètre de type `Uint32` qui est un `flag` qui va nous permettre d'utiliser telle ou telle fonction de la bibliothèque SDL.

Et pour pouvoir utiliser le module `AUDIO` de la SDL, il faudra dans ce cas utiliser le flag `SDL_INIT_AUDIO` !

Dans ce chapitre, nous verrons donc :

- La théorie (Signal périodique, Fréquence, Période, Fourier)
- La mise en pratique en SDL
- la bibliothèque `SDL_Mixer`

La théorie

Reprenons de zéro

Cette partie théorie, peut-être un peu compliquée à comprendre mais sachez juste qu'elle n'est en aucun nécessaire pour pouvoir **jouer des sons avec** `SDL_Mixer`, cependant si vous souhaitez jouer/créer des sons en SDL, il faut au préalable comprendre la partie théorique.

Maintenant votre question légitime est *"Comment jouer un son ou une musique ?"*. Déjà, avant de commencer à coder je vous invite à utiliser le logiciel libre Audacity et de produire un son avec votre microphone ! Si vous ne pouviez pas vous enregistrer, alors téléchargez le fichier sifflement.wav et ouvrez le avec le logiciel audacity. Une fois ouvert vous tombez sur le résultat suivant :

Ceci peut vous faire peur, ça ressemble un peu à rien, mais essayons de comprendre cela. Nous avons déjà en notre présence un dessin bleu et nous ne savons pas trop à quoi cela peut correspondre.

Alors regardez bien, la première chose qu'on peut dire c'est que nous avons un son qui se joue lorsqu'on appuie sur le bouton "play" d'audacity. On peut remarquer que notre son dure un certain temps (ici 4 secondes). Puis nous percevons aussi des indicateurs allant de -1 à 1.

Dorénavant, je vais apporter une approche plus scientifique :

- Sur l'axe des ordonnées nous avons le temps.
- Sur l'axe des abscisses nous avons l'amplitude.
- Enfin, la courbe représente un signal.

Mon professeur de mathématiques aime différencier une fonction d'un signal. En effet, un signal est censé représenter quelque chose de physique.

Définition

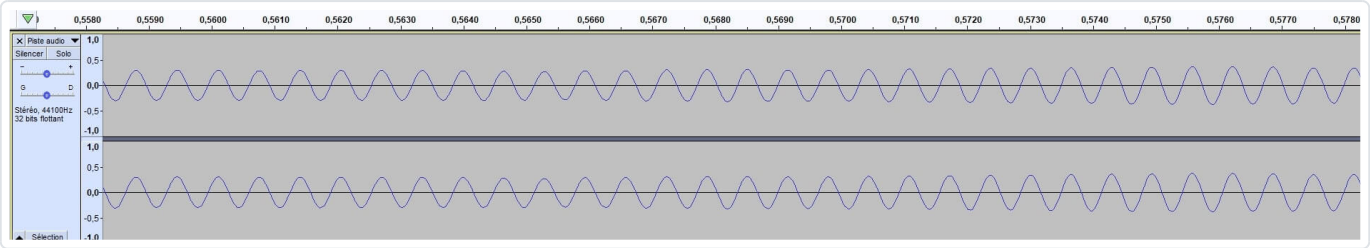
> *(Adjectif) Du latin physicus (« physique, naturel, des sciences naturelles ») tiré du grec ancien ???????, phusikós (« physique, naturel »). (Nom 1) Du latin physica (« physique, science de la nature »), du grec ancien ??????, phusik? (« science de la nature ») dérivé de ?????, phúsis (« nature »)*
> *Wikipedia*

Nous avons ici le mot physique qui désigne science de la nature, ce qui est le cas de mon sifflement car c'est un son sorti de ma bouche et je vous rassure il est bien naturel .

Tandis que la fonction

$$f(x) = \frac{1}{x}$$

décrit à ma connaissance aucun phénomène physique dans la nature. Si vous effectuez un zoom à partir de 0.5 secondes, vous apercevrez quelque chose de stupéfiant ! Soit :



Information

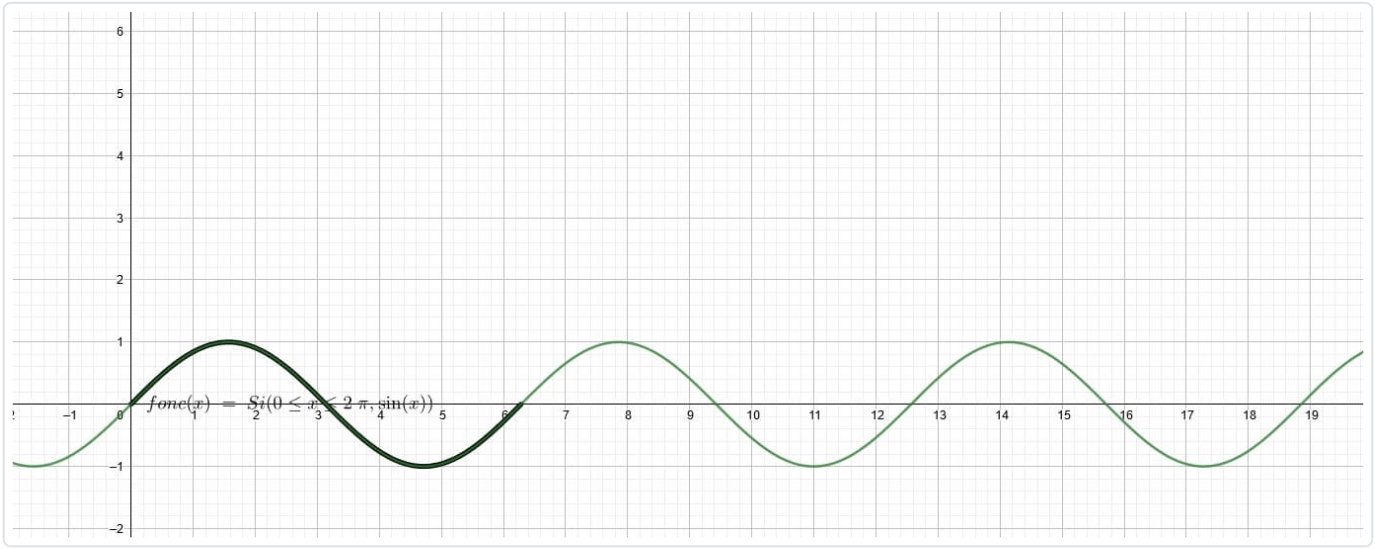
Par chance j'ai sifflet dans mon micro et j'ai constaté que ça a formé une sinusoïde (sans doute Fourier qui se cache derrière, nous allons parler de lui plus tard)

Vous avez une courbe qui est une sinusoïde qui semble être périodique de période T et de fréquence F.

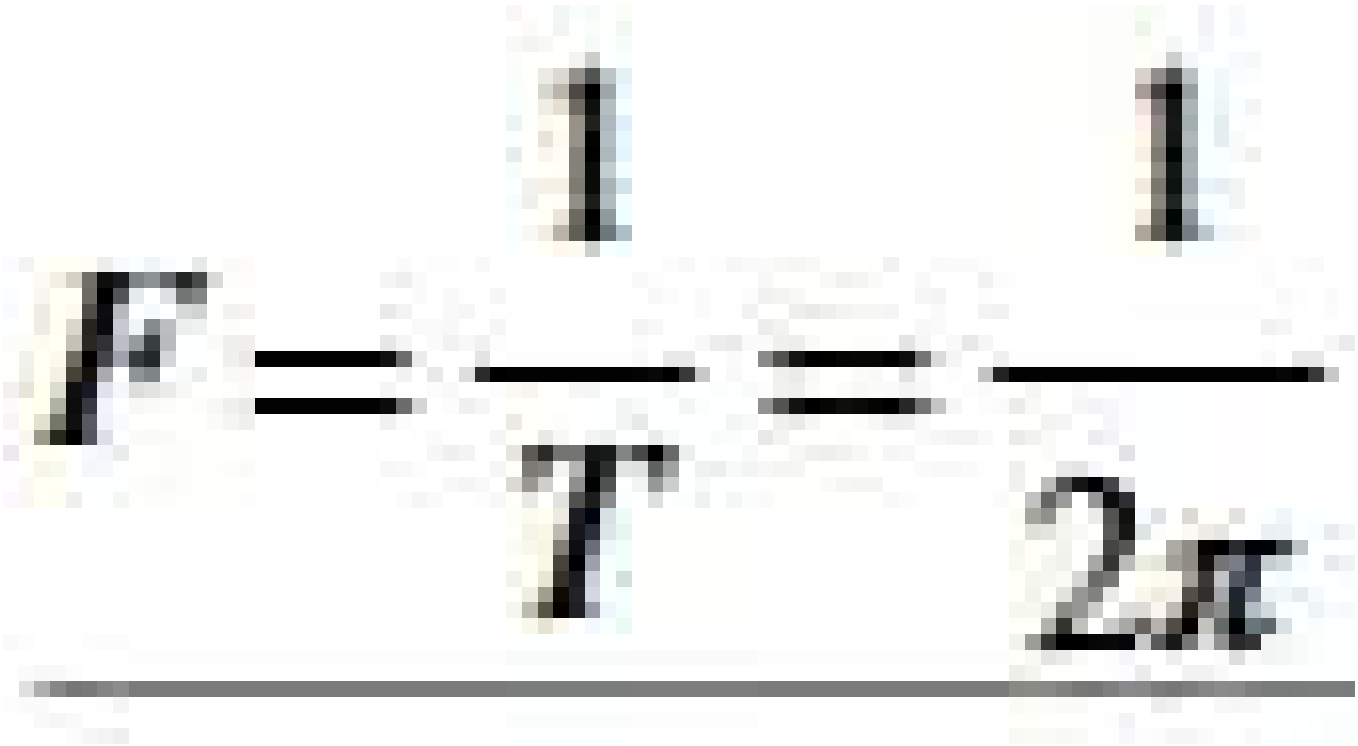
Qu'est-ce la fréquence et la période ? La fréquence est le nombre de répétitions du motif élémentaire en une seconde. La période c'est le temps que met votre motif élémentaire à être répété. Je vous donne

$$f(x) = \sin(x)$$

. Voici donc une courbe qui reprend notre signal à la forme sinusoïde afin de mieux le décrire : Vous aurez alors la courbe suivante :



En vert foncé c'est votre **motif élémentaire**. Vous constater qu'il est **périodique** car il (le motif élémentaire) se répète tous les 2□, donc une fois que nous avons la**période**, nous pouvons alors calculer la fréquence de ce signal qui est donc de



.

L'amplitude est donc de sin(□/2), ici ça vous donnera approximativement 1 (dédicace à mon professeur de physique de BTS SNIR) 1 comme sur le schéma que je vous ai fourni.

Bon la question légitime est *"pourquoi je vous parle de tout ça ?"* C'est juste pour votre culture générale ! Car quand vous allez utiliser `SDL_Mixer` la plupart du temps vous utiliserez un fichier son déjà fait par un musicien qui s'est enregistré. Cependant si vous souhaitez par exemple réaliser vos propres sons comme un synthétiseur vocal, il vous saura indispensable de reconnaître les notions que nous aborderons dans un instant.

De plus la gestion de l'audio en SDL est assez difficile à gérer. C'est pour cela qu'on utilisera plus tard `SDL_Mixer`. Cependant, je souhaite tout de même vous montrer au début comment gérer l'audio sans la bibliothèque SDL_Mixer, afin de comprendre en profondeur la gestion du son, et de me différencier aussi des autres tutos disponibles dans le web.

Maintenant vous savez à quoi ressemble un son en réalité. Pour information vous pouvez même produire votre propre son en programmation, nous coderons cette partie en SDL plus loin dans cet article.

Déjà, physiquement un son n'est rien d'autre qu'une onde caractérisée par son signal propre, qui est en fait une vibration de l'air.

> **Information**
>
> Les sons ne sont pas tous sinusoïdale (heureusement).

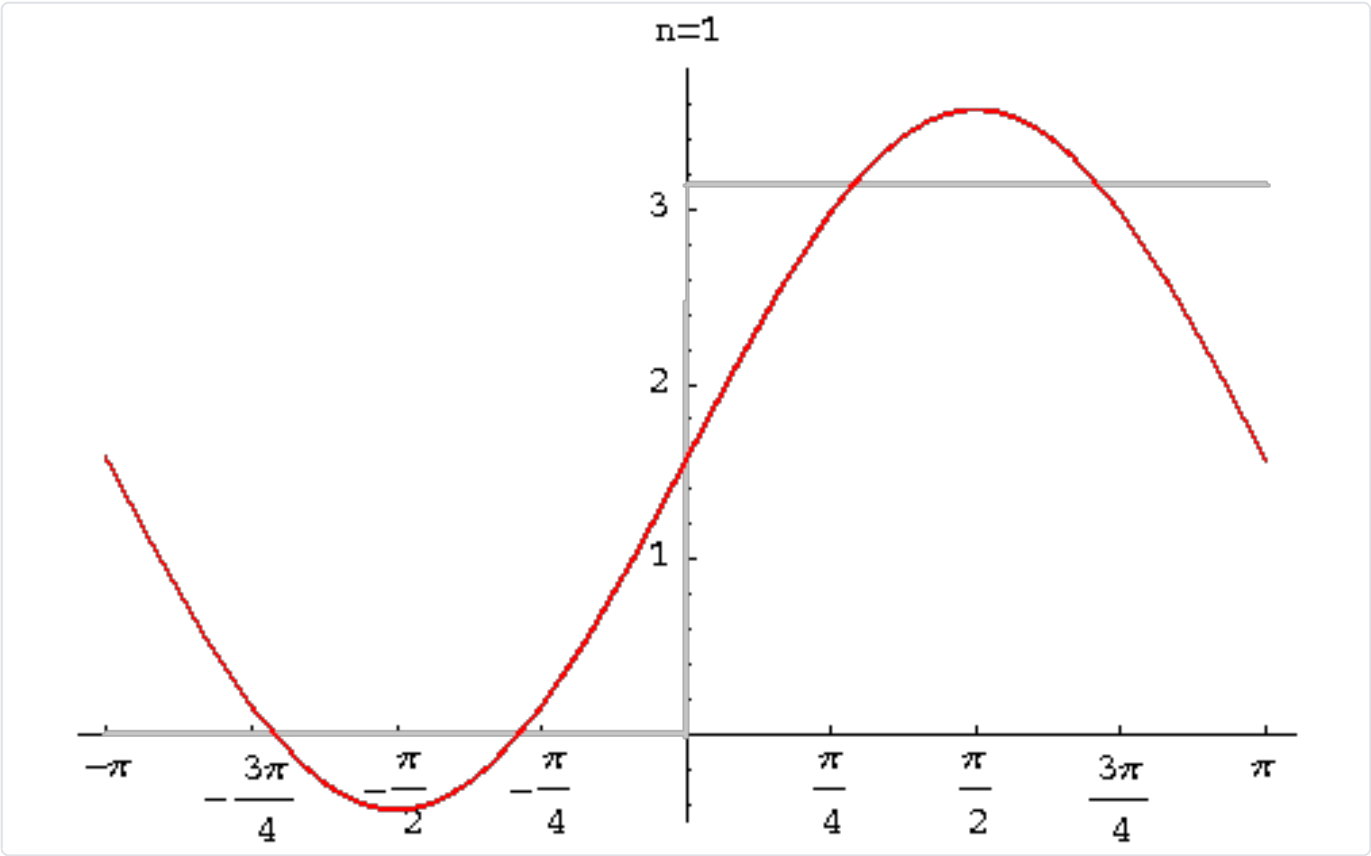
La série de fourier

La série de Fourier est une somme de sinusoïdes de fréquences et d'amplitudes, sous la formule suivante :

- $$P(t) = a_0 \sum_{n=1}^{\infty} (a_n \cos(nx) + b_n \sin(nx))$$
- $$a_0 = \frac{1}{2\pi} \int_{t_0}^{t_0+T} f(x)\,dx$$

- $$a_n = \frac{2}{T} \int_{t_0}^{t_0+T} f(x) \cdot \cos(n\omega t)\, dx$$
- $$b_n = \frac{2}{T} \int_{t_0}^{t_0+T} f(x) \cdot \sin(n\omega t)\, dx$$

Le polynôme de Fourier permet de faire des approximations de courbe périodique mais nous utiliserons aussi pour mettre des fréquences et des amplitudes différentes. Plus simplement l'approximation va nous servir à avoir un signal proche du signal original qu'on souhaite reproduire et dans le cas d'utilisation de fréquences différentes cela nous permettra de produire des sons différents, on utilisera aussi des amplitudes différentes pour avoir un volume différent.



Plus la valeur de est grande plus l'approximation est proche du correcte. Nous remarquons aussi le signe ? cela signifie la pulsation du signal

$$\omega = \frac{2\pi}{T} = 2\pi F$$

.

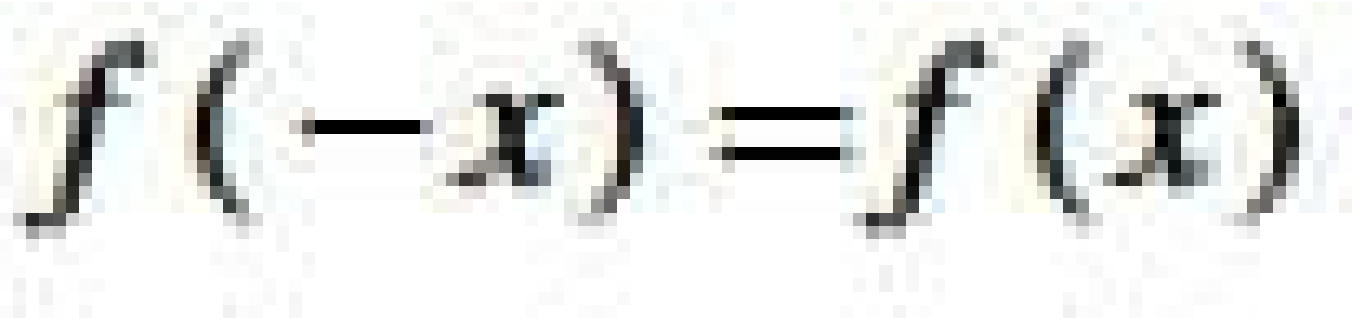
Fonction paire et fonction impaire

Je tiens à parler un peu des fonctions paires et impaires pour votre culture générale.

Vous avez sûrement connu, cette façon de vérifier un nombre pair ou impair :

```
int a = 32;
bool estImpair = ((a % 2) == 0);
```

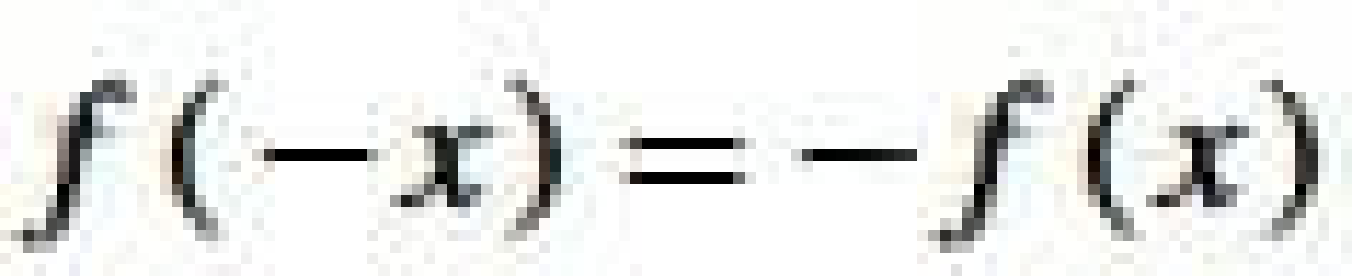
Oubliez là, car ici on parle de fonction paire et impaire, une fonction est dite paire si



Par exemple, pour x = 4, dans le cas d'une fonction paire, on obtiendra le même résultat que ça soit pour f(4) ou f(-4). On dit qu'ils ont a les mêmes images.

> **Information**
>
> L'image d'une fonction est le résultat produit par cette fonction.

Une fonction est dite impaire si



Toujours dans l'exemple x = 4 mais dans le cas d'une fonction impaire, nous obtiendrons f(-4)=-7 et f(4)=7. On dira alors qu'ils possèdent des images opposées.

Conclusion

La parité, d'une fonction permet de simplifier les calculs de la série de Fourier.

- Dans une fonction qui n'est pas paire ni impaire, vous serez alors obligé de calculer *an* et *bn* dans la série de fourier.
- Une fonction qui est paire (symétrique par rapport à l'axe des ordonnées), vous aurez alors tous vos *bn* (b1, b2, ..., bn) à nul !
- Une fonction qui est impaire (symétrique par rapport a l'origine), vous aurez alors tous vos *an* (a1, a2, ..., an) à nul !

Si vous prenez sin(x) sachant qu'on sait que cette fonction est impaire donc symétrique par rapport à l'origine ! (Rappel de la propriété : -sin(4) = sin(-4)). vous aurez donc vos *an* tous à 0 donc il n'y aura pas besoin de calculer an !

La mise en pratique en SDL

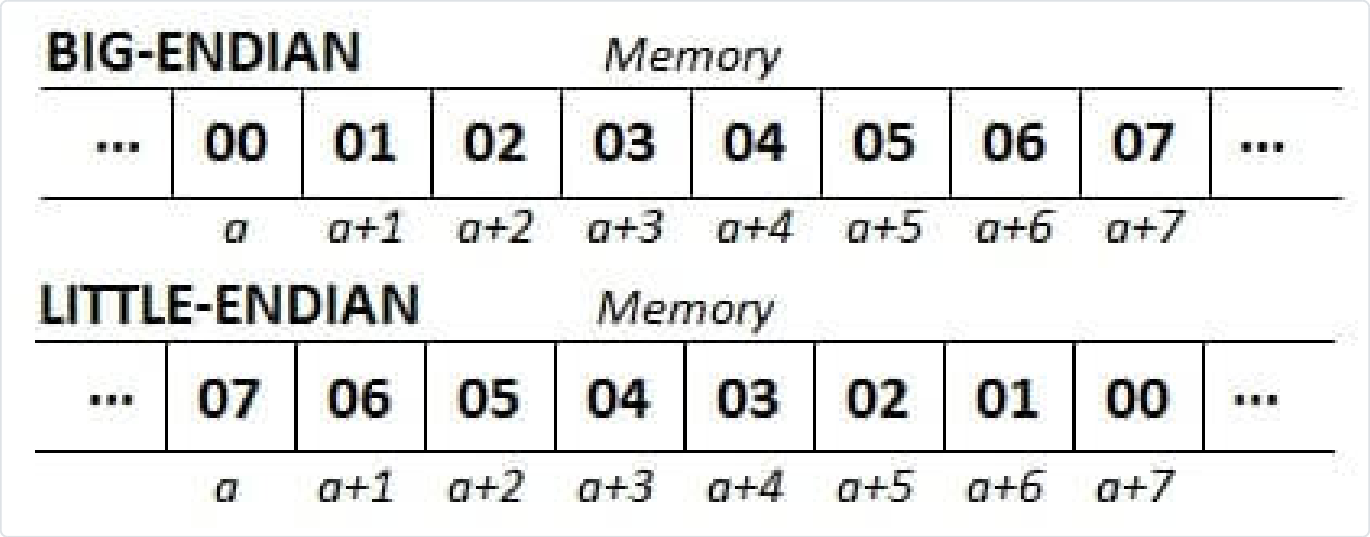
Pour cette section, nous allons charger que le module audio de la SDL.

```
SDL_Init(SDL_INIT_AUDIO);
```

Ensuite, nous devons créer une structure `SDL_AudioSpec`, voici les différents champs de notre structure :

int	**freq**	DSP frequency (samples per second); see Remarks for details
SDL_AudioFormat	**format**	audio data format; see Remarks for details
Uint8	**channels**	number of separate sound channels; see Remarks for details
Uint8	**silence**	audio buffer silence value (calculated)
Uint16	**samples**	audio buffer size in samples (power of 2); see Remarks for details
Uint32	**size**	audio buffer size in bytes (calculated)
SDL_AudioCallback	**callback**	the function to call when the audio device needs more data; see Remarks for details
void*	**userdata**	a pointer that is passed to **callback** (otherwise ignored by SDL)

- `freq` : correspond la fréquence d'échantillonnage du signal, en effet plus le signal est grand plus, plus la qualité du son est précise.
- `format` : C'est le format de notre son en type int ou float qui représente le nombre de bits pour une donnée de notre son et le le endianness c'est-à-dire l'ordre dans lequel ces octets sont organisés en mémoire. Nous verrons les flags possibles.
- `channels` : Nombre de canaux audio où l'on envoie le son. 1 (mono), 2 (stéréo), 4 (quad) et 6 (5.1).
- `samples` : la taille du buffer qui stockera les valeurs du son. Cette valeur doit être une puissance de 2.
- `callback` : on précisera ici la fonction afin de jouer les données de notre audio, depuis cette fonction. Le callback doit avoir cette forme `void SDL_AudioCallback(void* user data, Uint8* stream, int Len)`.



endianness (big-endian vs little-endian)

Ce sont les seuls champs qui nous intéressent, on peut ignorer le reste. Le champ `silence` et `size` sont automatiquement calculés et `userdata` peut-être à `nullptr`, ou `this` si vous l'utilisez dans une class par exemple.

Pour le champ `format` voici les valeurs possibles :

8-bit support	
AUDIO_S8	signed 8-bit samples
AUDIO_U8	unsigned 8-bit samples
16-bit support	
AUDIO_S16LSB	signed 16-bit samples in little-endian byte order
AUDIO_S16MSB	signed 16-bit samples in big-endian byte order
AUDIO_S16SYS	signed 16-bit samples in native byte order
AUDIO_S16	AUDIO_S16LSB
AUDIO_U16LSB	unsigned 16-bit samples in little-endian byte order
AUDIO_U16MSB	unsigned 16-bit samples in big-endian byte order
AUDIO_U16SYS	unsigned 16-bit samples in native byte order
AUDIO_U16	AUDIO_U16LSB
32-bit support (new to SDL 2.0)	
AUDIO_S32LSB	32-bit integer samples in little-endian byte order
AUDIO_S32MSB	32-bit integer samples in big-endian byte order
AUDIO_S32SYS	32-bit integer samples in native byte order
AUDIO_S32	AUDIO_S32LSB
float support (new to SDL 2.0)	
AUDIO_F32LSB	32-bit floating point samples in little-endian byte order
AUDIO_F32MSB	32-bit floating point samples in big-endian byte order
AUDIO_F32SYS	32-bit floating point samples in native byte order
AUDIO_F32	AUDIO_F32LSB

En gros vous avez un buffer qui peut être soit sur : 8 bits, 16 bits des 32 bits. Dans le cas des 16 bits et des 32 bit, vous avez le choix de choisir le type entier ou flottant. Vous avez aussi LSB (little-endian) et MSB (big-endian) qui correspond à endianness. Nous allons choisir le format de buffer en utilisant le flag `AUDIO_F32SYS` et le flag `AUDIO_F32SYS` afin que notre endianness soit

automatiquement choisi par notre machine.

Remplir la structure

```
SDL_AudioSpec spec;

SDL_memset(&spec, 0, sizeof(spec));

spec.freq = 96000; // 4 100 Hz, 48 000 Hz, 96 000 Hz, 192 000 Hz (standard)
spec.format = AUDIO_F32SYS;
spec.channels = 1; // mono
spec.samples = 4096; // Oublier pas que ce sa doit être en puissance de deux 2^n
spec.callback = [](void* param, Uint8* stream, int len)
{
    // Envoyez les données dans notre buffer...
};
```

Ensuite nous devons sélectionner la carte son que nous allons utiliser, pour cela on utilisera la fonction suivante :

```
SDL_AudioDeviceID SDL_OpenAudioDevice(const char*          device,
                                      int                  iscapture,
                                      const SDL_AudioSpec* desired,
                                      SDL_AudioSpec*       obtained,
                                      int                  allowed_changes)
```

- Le premier paramètre est le nom de la carte son.
- Le deuxième paramètre autorise ou non l'enregistrement d'un son.
- Le troisième définit les caractéristiques du son désiré.
- Le quatrième définit la caractéristique du son final. La différence avec le champ `desired` est que, lorsque nous désirons une configuration audio que votre carte son ne peut pas gérer (comme l'échantillonnage ou la fréquence), SDL va automatiquement vous renvoyer la configuration possible dans le champ `obtained`.

- Le cinquième paramètre définit la fonctionnalité qu'on souhaite effectuer sur la carte son pour modifier par exemple une voix et la rendre de plus en plus grave. Voici les valeurs possibles :

SDL_AUDIO_ALLOW_FREQUENCY_CHANGE
SDL_AUDIO_ALLOW_FORMAT_CHANGE
SDL_AUDIO_ALLOW_CHANNELS_CHANGE
SDL_AUDIO_ALLOW_ANY_CHANGE

Voici un exemple d'utilisation :

```
SDL_AudioDeviceID dev = SDL_OpenAudioDevice(nullptr, 0, &spec, &spec, SDL_AUDIO_ALLO
```

Mettre nullptr au premier paramètre va choisir tout seul le périphérique. Ensuite effectuer une boucle infinie :

Ensuite nous devons mettre en lecture notre périphérique audio. En utilisant :

```
void SDL_PauseAudioDevice(SDL_AudioDeviceID dev,
                          int               pause_on)
```

- Le premier paramètre donné est l'identifiant de notre périphérique.
- Le deuxième peut prendre deux valeurs soit : `SDL_FALSE` (ou 0) pour ne pas mettre en pause `SDL_TRUE` (ou 1) pour mettre en pause.

Vous aurez ainsi le code suivant :

```
int main(int argc, char* argv[])
{
    if (SDL_Init(SDL_INIT_AUDIO) < 0)
```

```
        return -1;

    SDL_AudioSpec spec;

    SDL_memset(&spec, 0, sizeof(spec));

    spec.freq = 96000; // 4 100 Hz, 48 000 Hz, 96 000 Hz, 192 000 Hz (standard)
    spec.format = AUDIO_F32SYS;
    spec.channels = 1;
    spec.samples = 4096; // Oublier pas que ce sa doit être en puissance de deux 2^n
    spec.callback = [](void* param, Uint8* stream, int len)
    {
        // Envoyez les données dans notre buffer...
    };

    SDL_AudioDeviceID dev = SDL_OpenAudioDevice(nullptr, 0, &spec, &spec, SDL_AUDIO_A

    SDL_PauseAudioDevice(dev, SDL_FALSE);

    for (;;); // boucle infinie

    SDL_Quit();

    return 0;
}
```

Ensuite si vous lancez votre programme, il n'y aura rien qui se passe car on n'a pas encore envoyé de données à notre buffer `stream` ! Pour ce faire, voici ce que vous devez rentrer dans votre callback :

```
spec.callback = [](void* param, Uint8* stream, int len)
{
    // Envoyez les données dans notre buffer...
    int samples = len / sizeof(float); // 4096

    for (auto i = 0; i < samples; i++)
    {
        reinterpret_cast<float*>(stream)[i] = 0.5 * SDL_sinf(2 * M_PI * i / 1000);
    }
};
```

Si vous exécutez maintenant votre programme vous entendrez un son. Vous pouvez utiliser soit la fréquence soit la période, soit :

- Asin(2 * pi * x/T) => utilise la période

- Asin(2 * pi * f*x) => utilise la fréquence

Vous pouvez vous amuser à faire des do-ré-mi-fa-so-la-si-do en utilisant leur fréquence si vous êtes doué en musique et donc composez votre propre musique.

SDL_Mixer

Bon, on a vu une partie de la théorie d'un son via par exemple la série de Fourier, nous avons vu aussi comment produire un son sous SDL. Maintenant, ce qu'on aimerait faire c'est déjà jouer des fichiers .wav, .mp3, etc ...

Mais avant de coder avec SDL_Mixer il faut que vous installiez, SDL_Mixer via à ce lien https://www.libsdl.org/projects/SDL_mixer/, comme nous l'avions fait pour le module SDL_TTF ! Il faut que vous soyez autonome , juste pensez à modifier votre configuration du projet sous visual studio, ou votre ligne de commande lors de la compilation sous g++.

Donc, pour pouvoir utiliser SDL_Mixer il faut d'abord l'initialiser tout comme SDL voici le prototype de la fonction a appelé :

```
extern DECLSPEC int SDLCALL Mix_OpenAudio(int frequency, Uint16 format, int channels,
```

On connait déjà quelque paramètre

- Le premier paramètre est la fréquence d'échantillonnage (qualité du son)
- Le deuxième paramètre est le format de données (comme pour le champ `format` de la structure `SDL_AudioSpec`)
- Le troisième représente les canaux mono, stéréo, 5.1
- Le quatrième est mon taux d'échantillons soit la taille du buffer.

Voici comment l'initialiser :

```
if (Mix_OpenAudio(96000, MIX_DEFAULT_FORMAT, MIX_DEFAULT_CHANNELS, 1024) < 0)
{
    SDL_Log("Erreur initialisation SDL_mixer : %s", Mix_GetError());
    SDL_Quit();
    return -1;
}
```

Ensuite pensez à libérer la mémoire, en utilisant la fonction `Mix_CloseAudio()`.

```
extern DECLSPEC void SDLCALL Mix_CloseAudio(void);
```

Ce qui nous donne le code suivant :

```
#include <SDL.h>
#include <SDL_mixer.h>

int main(int argc, char* argv[])
{
    if (SDL_Init(SDL_INIT_VIDEO) < 0) // Pas besoin de SDL_INIT_AUDIO
        return -1;

    if (Mix_OpenAudio(96000, MIX_DEFAULT_FORMAT, MIX_DEFAULT_CHANNELS, 1024) < 0) //
    {
        SDL_Log("Erreur initialisation SDL_mixer : %s", Mix_GetError());
        SDL_Quit();
        return -1;
    }

    Mix_CloseAudio();
    SDL_Quit();

    return 0;
}
```

Ensuite nous devons créer un pointeur sur une structure `Mix_Music` afin de récupérer le buffer audio de notre fichier son. cela ce fait en appelant la fonction `Mix_LoadMUS` qui a pour signature :

```
extern DECLSPEC Mix_Music * SDLCALL Mix_LoadMUS(const char *file);
```

- Elle retourne notre pointeur sur une structure Mix_Music
- Elle prend pour unique paramètre le chemin de votre fichier son

```
Mix_Music* music = Mix_LoadMUS("ouman.mp3");

if (music == nullptr)
{
    SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Erreur chargement de la musique : %s"
    Mix_CloseAudio();
    SDL_Quit();
    return -1;
}
```

Ensuite nous pouvons jouer notre musique en utilisant la fonction `Mix_PlayMusic` voici sa signature :

```
extern DECLSPEC int SDLCALL Mix_PlayMusic(Mix_Music *music, int loops);
```

- Le premier paramètre est le pointeur sur la `Mix_Music`.
- Le dernier paramètre est pour savoir combien de fois nous devons jouer la musique (-1 jouer en boucle).
- Retourne -1 s'il y a une erreur et 0 s'il n'y en a pas.

Voici le code résultant :

```
#include <SDL.h>
#include <SDL_mixer.h>

int main(int argc, char* argv[])
{
    if (SDL_Init(SDL_INIT_VIDEO) < 0)
        return -1;
    // Initialisation de SDL_Mixer
    if (Mix_OpenAudio(96000, MIX_DEFAULT_FORMAT, MIX_DEFAULT_CHANNELS, 1024) < 0)
    {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Erreur initialisation SDL_mixer
        SDL_Quit();
        return -1;
    }
```

```
    Mix_Music* music = Mix_LoadMUS("ouman.mp3"); // Charge notre musique

    if (music == nullptr)
    {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Erreur chargement de la musique
        Mix_CloseAudio();
        SDL_Quit();
        return -1;
    }

    Mix_PlayMusic(music, -1); // Joue notre musique

    SDL_Window* pWindow = nullptr;
    SDL_Renderer* pRenderer = nullptr;
    SDL_Event events;
    bool close = false;

    SDL_CreateWindowAndRenderer(800, 600, SDL_WINDOW_SHOWN, &pWindow, &pRenderer);

    while (!close)
    {
        while (SDL_PollEvent(&events))
        {
            if (events.type == SDL_WINDOWEVENT && events.window.event == SDL_WINDOWEV
                close = true;
        }

        SDL_SetRenderDrawColor(pRenderer, 0, 0, 0, 255);
        SDL_RenderClear(pRenderer);
        SDL_RenderPresent(pRenderer);
    }

    SDL_DestroyRenderer(pRenderer);
    SDL_DestroyWindow(pWindow);
    Mix_FreeMusic(music); // Libére en mémoire notre musique
    Mix_CloseAudio(); // Quitter correctement SDL_Mixer
    SDL_Quit();

    return 0;
}
```

Bon maintenant, nous allons voir quelque fonction utilitaire pour changer le volume, reprendre la musique depuis le début etc...

- Mettre en pause la musique avec `Mix_PauseMusic()` :

```
extern DECLSPEC void SDLCALL Mix_PauseMusic(void);
```

- Reprendre la lecture après la pause de la musique `Mix_ResumeMusic()` :

```
extern DECLSPEC void SDLCALL Mix_ResumeMusic(void);
```

- Revenir au début de la musique `Mix_RewindMusic()`

```
extern DECLSPEC void SDLCALL Mix_RewindMusic(void);
```

- Changer le volume de la musique `Mix_VolumeMusic()`

```
extern DECLSPEC int SDLCALL Mix_VolumeMusic(int volume);
```

- Stopper la lecture de la musique `Mix_HaltMusic()` :

```
extern DECLSPEC int SDLCALL Mix_HaltMusic(void);
```

Voici un programme qui résume toutes les fonctions utilitaires vues précédemment :

```
#include <SDL.h>
#include <SDL_mixer.h>

int main(int argc, char* argv[])
{
    if (SDL_Init(SDL_INIT_VIDEO) < 0)
        return -1;

    if (Mix_OpenAudio(96000, MIX_DEFAULT_FORMAT, MIX_DEFAULT_CHANNELS, 1024) < 0)
    {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Erreur initialisation SDL_mixer
        SDL_Quit();
        return -1;
    }

    Mix_Music* music = Mix_LoadMUS("ouman.mp3");

    if (music == nullptr)
    {
```

```
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Erreur chargement de la musique :
        Mix_CloseAudio();
        SDL_Quit();
        return -1;
    }

    Mix_PlayMusic(music, -1);

    SDL_Window* pWindow = nullptr;
    SDL_Renderer* pRenderer = nullptr;
    SDL_Event events;
    bool close = false;

    if (SDL_CreateWindowAndRenderer(800, 600, SDL_WINDOW_SHOWN, &pWindow, &pRenderer)
    {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Erreur creation fenetre et rendue
        Mix_FreeMusic(music);
        Mix_CloseAudio();
        SDL_Quit();
        return -1;
    }

    Uint8 volume = 0;
    Mix_VolumeMusic(volume); // Mets le volume a 0

    while (!close)
    {
        while (SDL_PollEvent(&events))
        {
            if (events.type == SDL_WINDOWEVENT && events.window.event == SDL_WINDOWEV
                close = true;

            if (events.type == SDL_KEYDOWN && events.key.keysym.sym == SDLK_p)
                Mix_PauseMusic(); // Mets en pause la musique
            if (events.type == SDL_KEYDOWN && events.key.keysym.sym == SDLK_r)
                Mix_ResumeMusic(); // Reprend la lecture
            if (events.type == SDL_KEYDOWN && events.key.keysym.sym == SDLK_s)
                Mix_RewindMusic(); // Revient au début de la musique
            if (events.type == SDL_KEYDOWN && events.key.keysym.sym == SDLK_UP && vo
                volume++; // Augmente le volume jusqu'a MIX_MAX_VOLUME
            if (events.type == SDL_KEYDOWN && events.key.keysym.sym == SDLK_DOWN && v
                volume--; // Réduit le volume jusqu'a 0
            if (events.type == SDL_KEYDOWN && events.key.keysym.sym == SDLK_ESCAPE)
                Mix_HaltMusic(); // Arreter la musique

            if (events.type == SDL_KEYDOWN)
                Mix_VolumeMusic(volume); // Applique le volume desirer

        }

        SDL_SetRenderDrawColor(pRenderer, 0, 0, 0, 255);
        SDL_RenderClear(pRenderer);
        SDL_RenderPresent(pRenderer);
    }
```

```
    SDL_DestroyRenderer(pRenderer);
    SDL_DestroyWindow(pWindow);
    Mix_FreeMusic(music);
    Mix_CloseAudio();
    SDL_Quit();

    return 0;
}
```

Maintenant nous allons voir comment jouer plusieurs sons sous différents canaux (buffer dédié pour le son). Nous allons voir que c'est encore très simple sous `SDL_Mixer`.

Pour commencer, il faut définir préalablement combien de canaux seront utilisés à l'aide de la fonction `Mix AllocateChannels()`, voici donc sa signature :

```
extern DECLSPEC int SDLCALL Mix_AllocateChannels(int numchans);
```

- Le premier paramètre c'est le nombre de canaux.
- Elle retourne -1 si erreur et 0 si ça se passe bien.

Information

Le nombre de canaux alloués, c'est en fait le nombre de sons que nous pouvons jouer simultanément, comme par exemple la respiration de votre joueur et le son au contact du sol, donc il faudra deux canaux dans cet exemple.

Maintenant nous allons configurer le volume pour nos deux canaux ! Nous allons mettre le premier canal à 100% et 50% pour le second canal, pour ce faire il faut utiliser la fonction `Mix_Volume()`, voici sa signature

```
extern DECLSPEC int SDLCALL Mix_Volume(int channel, int volume);
```

- Le premier paramètre est le canal.
- Le second paramètre est le volume souhaité en pourcentage.
- Elle retourne -1 en cas d'erreur 0 en cas de succès.

Pour jouer vos sons simultanément, il suffit de créer un pointeur sur une structure `Mix_Chunk` pour celà nous appellerons `Mix_LoadWAV()` qui a pour signature :

```
#define Mix_LoadWAV(file)   Mix_LoadWAV_RW(SDL_RWFromFile(file, "rb"), 1)
```

Nous avons à faire avec une Macro qui n'est rien d'autre qu'une utilisation de `SDL_RWFromFile()` nous devons juste passez le chemin du fichier son. Ensuite nous pouvons jouer notre son avec la fonction `Mix_PlayChannel()` voici la signature

```
#define Mix_PlayChannel(channel,chunk,loops) Mix_PlayChannelTimed(channel,chunk,loops
```

Nous avons encore à faire à une macro fonction !

- Le premier paramètre est le numéro du canal.
- Le deuxième paramètre est le pointeur sur le `Mix_Chunck`.
- Le troisième paramètre c'est le nombre de répétition du son (mettre 0 le répète 1 fois, mettre 1 répète deux fois etc ...)

Puis n'oubliez pas de libérer la mémoire des `Mix_Chunk` à la fin du programme en appellant `Mix_FreeChunk()`, voici sa signature :

```
extern DECLSPEC void SDLCALL Mix_FreeChunk(Mix_Chunk *chunk);
```

- Le premier paramètre est un pointeur sur une stucture Mix_Chunk.

Voici maintenant le résultat final :

```
#include <SDL.h>
#include <SDL_mixer.h>

int main(int argc, char* argv[])
{
    if (SDL_Init(SDL_INIT_VIDEO) < 0)
        return -1;

    if (Mix_OpenAudio(96000, MIX_DEFAULT_FORMAT, MIX_DEFAULT_CHANNELS, 1024) < 0)
    {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Erreur initialisation SDL_mixer
        SDL_Quit();
        return -1;
    }

    SDL_Window* pWindow = nullptr;
    SDL_Renderer* pRenderer = nullptr;
    SDL_Event events;
    bool close = false;

    if (SDL_CreateWindowAndRenderer(800, 600, SDL_WINDOW_SHOWN, &pWindow, &pRenderer
    {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Erreur creation fenetre et rendue
        Mix_CloseAudio();
        SDL_Quit();
        return -1;
    }

    Mix_AllocateChannels(2); // Allouer 2 cannaux
    Mix_Volume(0, MIX_MAX_VOLUME); // Mets le son a 100% en volume pour le premier ca
    Mix_Volume(1, MIX_MAX_VOLUME / 2); // Mets le son a 50% en volume pour le deuxièr

    Mix_Chunk* soundA = Mix_LoadWAV("cla.wav");
    Mix_Chunk* soundB = Mix_LoadWAV("boume.wav");

    while (!close)
    {
        while (SDL_PollEvent(&events))
        {
            if (events.type == SDL_WINDOWEVENT && events.window.event == SDL_WINDOWEV
                close = true;
            if(events.type == SDL_KEYDOWN && events.key.keysym.sym == SDLK_a)
                Mix_PlayChannel(0, soundA, 1); // Joue soundA deux fois sur le canal
            if (events.type == SDL_KEYDOWN && events.key.keysym.sym == SDLK_b)
                Mix_PlayChannel(1, soundB, 0);  // Joue soundB une fois sur le canal
            if (events.type == SDL_KEYDOWN && events.key.keysym.sym == SDLK_z)
                Mix_PlayChannel(0, soundB, 3), Mix_PlayChannel(1, soundA, 0); // Joue

        }

        SDL_SetRenderDrawColor(pRenderer, 0, 0, 0, 255);
```

```
            SDL_RenderClear(pRenderer);
            SDL_RenderPresent(pRenderer);
        }

        Mix_FreeChunk(soundA); // Libére la mémoire allouer pour le son
           Mix_FreeChunk(soundB);
        SDL_DestroyRenderer(pRenderer);
        SDL_DestroyWindow(pWindow);
        Mix_CloseAudio();
        SDL_Quit();

        return 0;
    }
```

Voilà nous avons quasiment finit ce tutoriel. Dorénavant, il nous reste qu'à voir encore que certaines fonctions utilitaires pour les sons simultanés :

- Pour stop un canal

```
extern DECLSPEC int SDLCALL Mix_HaltChannel(int channel);
```

- Pour mettre en pause un cannal

```
extern DECLSPEC void SDLCALL Mix_Pause(int channel);
```

- Pour reprendre la lecture d'un cannal

```
extern DECLSPEC void SDLCALL Mix_Resume(int channel);
```

Vous pouvez rentrer en paramètre soit le numéro du canal en question, soit la valeur "-1" afin de stopper tous les canaux.

Bon voilà, nous en avons fini pour ce cours de programmation sous SDL. On aura l'occasion de manipuler SDL pour créer de réels jeux vidéo et nous aurons aussi l'occasion de voir d'autres tutoriels, mais dès à présent vous être prêts pour développer des jeux vidéo intéressants.

REMERCIEMENT

- Mon ami ttatanepvp123 (ton autre pseudo me dégoûte)
- Mon ami Timtim , sans lui il n'y aurait pas eu ces articles sur la SDL et merci énormément pour avoir corrigé mes articles.
- Mon professeur de physique, et mon professeur de mathématiques pour m'avoir enseigné à l'école la série de Fourier
- Remerciement à tous mes lecteurs.