

TD Outils 4 : SDL2

I. Visualiseur d'Image

Dans le [dépôt git de l'UE](#), vous trouverez plusieurs codes pour vous aider à comprendre SDL2. Le code `SDL_Simple` qui ouvre une fenêtre et affiche une image et le code d'un embryon du jeu Pacman dans le répertoire du même nom. L'objectif du TD est de vous familiariser avec SDL2, et de l'intégrer à votre module Image. Prenez donc du temps pour explorer ces codes, et plus particulièrement `SDL_Simple`, et dans le Pacman les classes `Jeu`, `sdlJeu` et `txtJeu`.

Il y a également un code minimaliste qui intègre `ImGUI` qui est une bibliothèque d'interface utilisateur (UI) pour le C++. Elle offre de nombreux widgets et est beaucoup plus légère que Qt. A garder dans un coin de votre tête pour le projet. [Regardez ici](#).

1. Pour le module Image à rendre, vous devez ajouter une classe dans les fichiers `ImageViewer.h` et `ImageViewer.cpp`.

```
Classe ImageViewer
Privé
    win : SDL_Window
    ren : SDL_Renderer
    // imSDL : SDL_Surface    // pas obligatoire
    // tex : SDL_Texture      // pas obligatoire
    // éventuellement d'autres données
public
    // Constructeur qui initialise tout SDL2 et créer la fenêtre
    // et le renderer
    Constructeur ImageViewer()

    // Détruit et ferme SDL2
    Destructeur ImageViewer()

    // Affiche l'image passée en paramètre et permet le (dé)zoom
    Procédure Display(im : ref const sur une Image )
```

La procédure `Display` affiche une image passée en paramètre. Tout le problème est de rendre une image de type *Image* « compatible avec SDL2 ». L'image sera placée au centre d'une fenêtre SDL2 de taille 200x200 pixels de fond gris clair et vous devrez permettre de zoomer/dézoomer sur votre image grâce aux touches T et G et quitter la procédure `afficher()` grâce à la touche `ESCAPE`.

Pour afficher une *Image* avec SDL2, trois solutions sont envisageables.

- Sauver l'image dans un fichier « .ppm », puis charger l'image dans SDL2 avec `IMG_Load`. Ça marche, mais ce n'est pas très satisfaisant de passer par un fichier.
- Faire une double boucle sur tous les pixels de l'Image et afficher un carré avec SDL2 pour chaque pixel. Ça marche, mais ce n'est pas très efficace. Il faut trouver comment gérer le zoom.
- Convertir le tableau de pixel de la classe *Image* en une `SDL_Surface`, puis créer la `SDL_Texture` qui est affichée par SDL2. Par exemple, ChatGPT propose ce code pour accéder à un pixel :

```
// Accès au pixel à la position (x, y)
int x = 10;
int y = 10;
Uint32* pixels = (Uint32*)surface->pixels;
Uint32 pixel = pixels[y * surface->w + x];

// Modification du pixel (par exemple, changer la couleur en rouge)
pixels[y * surface->w + x] = SDL_MapRGB(surface->format, 255, 0, 0);
```

2. Dans votre module Image, créez un 3e programme principal nommé `mainAffichage.cpp` qui contient impérativement le code suivant, sans le modifier pour le script de notation.

`mainAffichage.cpp` (produisant l'exécutable `bin/affichage`) :

```
#include "Image.h"

int main (int argc, char** argv) {

    Image image (11,11);

    Pixel gris (226, 226, 226);
    Pixel vert (1, 130, 0);
    Pixel noir (0, 0, 0);
    Pixel beigeC (210, 188, 154);
    Pixel beigeF (168, 105, 8);
    Pixel marron (102, 65, 8);

    image.effacer(gris);
    image.dessinerRectangle(3,0,7,10,vert);
    image.dessinerRectangle(0,1,10,1,vert);
    image.setPix(2,2,vert);
    image.setPix(8,2,vert);
    image.setPix(3,2,gris);
    image.setPix(7,2,gris);
    image.setPix(4,2,noir);
    image.setPix(6,2,noir);
    image.dessinerRectangle(1,5,9,6,vert);
    image.setPix(5,10,gris);
    image.dessinerRectangle(9,7,9,10,marron);
    image.dessinerRectangle(2,5,8,6,beigeC);
    image.dessinerRectangle(3,7,7,9,beigeC);
    image.dessinerRectangle(4,5,6,6,beigeF);
    image.setPix(5,7,beigeF);

    ImageViewer imview;
    imview.display( image );

    return 0;
}
```

II. Valgrind et les bibliothèques : SDL2

Si vous lancez valgrind sur un exécutable utilisant une bibliothèque comportant des fuites mémoire, vous obtiendrez ces fuites même si votre code libère bien toute la mémoire allouée que vous allouez. Elles apparaissent pour la plupart en *still reachable*, ce qui correspond à des zones de mémoire sur lesquelles on dispose encore de pointeurs et que l'on aurait pu désallouer si on l'avait voulu. Ces fuites (qui n'en sont pas vraiment...) relèvent du fonctionnement interne des bibliothèques et sont indépendantes de votre volonté. Elles ne risquent pas de faire planter votre programme, mais gênent la recherche de vrais bugs. Heureusement, pour ne voir que vos fuites mémoires, il est possible d'indiquer à valgrind d'ignorer ces erreurs, en utilisant le fichier de suppression valgrind_lif7.supp fourni dans l'archive, et en spécifiant les options adéquates lors de l'exécution de valgrind :

```
$> valgrind --leak-check=full --num-callers=50 --suppressions=./valgrind_lif7.supp  
--show-reachable=yes -v NOM_DE_L'EXECUTABLE
```

Les fuites mémoire propres à SDL2 sont toujours dénombrées, mais apparaissent désormais en *suppressed*. Documentez-vous sur valgrind pour comprendre le sens des options.