Master 1 Informatique
Université de Lyon 2 - ICOM 2022/2023

Module Base de la programmation *C++*

Travaux Pratiques 6

Programmation avec *SDL*

Objectif: Ce TP se situe dans la continuité de la séance précédente où il s'agit de porter l'application de casse brique cette fois non pas en mode ASCII mais en utilisant la bibliothèque graphique *SDL*.

Concept du cours :

Bibliothèque SDL : **S**imple **D**irectMedia **L**ayer est une bibliothèque libre et gratuite, de bas niveau permettant de gérer par exemple l'image, le son, les événements clavier/souris et l'accélération matérielles pour le graphisme. Attention, ne pas confondre avec la Standard Library (std : :). Le site officiel contenant la documentation ou des tutoriels est disponible à cette adresse : `https://www.libsdl.org`. Dans ce TP, nous utiliserons la version 2.0 de SDL.

Utilisation : cette bibliothèque va être utilisée comme pour les TP précédents à travers l'environnement de *Visual Studio*. Cependant, tout ce travail peut être transposé dans un autre environnement y compris sous linux ou windows car cette bibliothèque est multi-plateformes.

Distribution d'une bibliothèque : lorsque l'on récupère une bibliothèque, en général il est possible de récupérer différentes parties :

- **Code source** : Il s'agit du code source de la libraire. C'est utile si vous souhaitez voir en détails comment fonctionne la librairie ou si vous souhaitez contribuer à de nouvelles fonctionnalités. Mais cela sort du contexte de ce cours.
- **Runtime libraries** : ce sont les fichiers nécessaires pour qu'une personne puisse utiliser votre programme qui la librairie. Il s'agit de fichier .dll. Ces fichiers seront nécessaires dans notre cas mais ne permettront pas à eux seuls de construire un executable.
- **Development librairies :** cette distribution intègre à la fois les fichiers nécessaires à la construction de votre programme (fichier headers (.h) et fichier .lib). Ils ne sont pas destinés aux utilisateurs du programme que vous aurez produit.

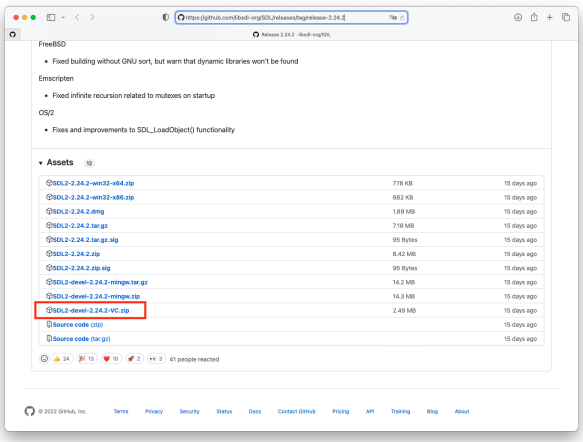
Partie I: Intégration de SDL et première fenêtre (10 ∼ 15 min)

Dans cet exercice il s'agit de récupérer la bibliothèque *SDL* et de configurer un nouveau projet *Visual Studio* qui l'utilisera. Afin de vérifier que l'installation est bien effective, vous afficherez une première fenêtre.

1.1 Récupérez la bibliothèque *SDL* depuis l'adresse suivante :

`https://github.com/libsdl-org/SDL/releases/tag/release-2.24.2`

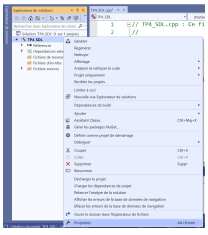
et enregistrez le répertoire source dans un nouveau répertoire `Dev` à la racine du disque principal de votre machine. Pour récupérer la bibliothèque vous devez chercher le kit complet de développement (voir image ci-dessous).



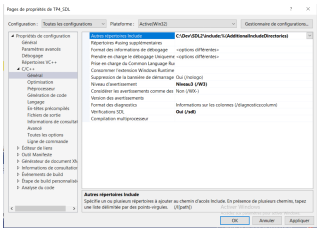
1.2 Comme pour les TP précédents, créer un nouveau projet de type application console C++.

1.3 Pour utiliser la bibliothèque *SDL* il est nécessaire d'indiquer à *Visual Studio* où se trouvent les fichiers relatifs à la bibliothèques (à la fois les fichiers headers, la librairies. Pour cela vous devez suivre les trois étapes suivantes :

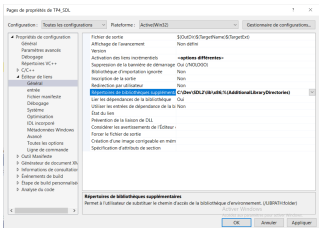
- Lancez le **panneau de configuration** lié à votre projet. Il s'obtient simplement à partir d'un clic droit de votre projet (voir image (a) ci dessous).
- Dans la section *C/C++/Général* du panneau précédent, dans le champ *Autres répertoires include* rajoutez le répertoire `include` qui est présent dans votre répertoire récupéré (normalement dans `C:\Dev\SDL2\include` (voir image (b)).
- Dans la section *Editeur de liens/Général*, dans le champ *Répertoire de bibliothèques supplémentaires*, rajoutez le répertoire de votre librairie en sélectionnant celui associé à votre architecture (voir image (c)).
- Toujours dans la section *Editeur de liens* mais dans sous section *entrée*, rajoutez les deux noms des librairies suivantes `SDL2.lib` et `SDL2main.lib` (voir image (d).



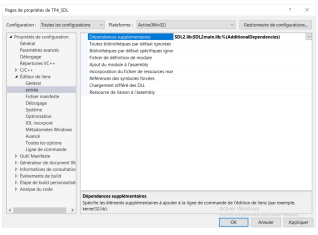
(a) panneau de conf



(b) include



(c) lien lib



(d) lien (input)

1.4 Maintenant, la configuration est presque terminée et il ne reste plus qu'à copier le fichier `C:\Dev\SDL2\lib\x86\SDL2.dll` dans le même répertoire où est situé votre fichier source `.cpp`.

1.5 Recopiez le code (a) représenté dans la figure suivante et vérifier que votre programme fonctionne toujours.

```cpp
#include <SDL.h>
int main(int argc, char* argv[])
{
    SDL_Init(SDL_INIT_EVERYTHING);
    return 0;
}
```

(a)

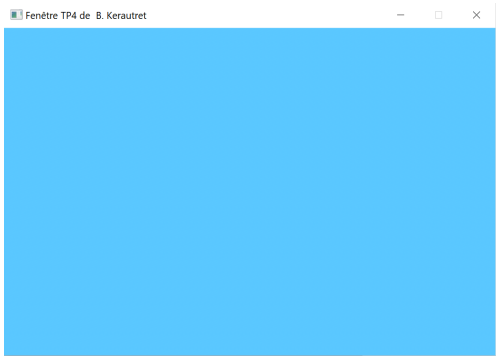
```cpp
#include <iostream>
#include "SDL.h"

int main(int argc, char* argv[])
{
    SDL_Init(SDL_INIT_EVERYTHING);
    SDL_Window* win = SDL_CreateWindow("Fenêtre TP4 de  B. Kerautret", SDL_WINDOWPOS_CENTERED,
                                                                      SDL_WINDOWPOS_CENTERED,
                                                                      600, 400, SDL_WINDOW_SHOWN);
    SDL_Renderer* renderer = SDL_CreateRenderer(win, -1, SDL_RENDERER_ACCELERATED);
    SDL_SetRenderDrawColor(renderer, 90, 200, 255, 255);
    SDL_RenderClear(renderer);
    SDL_RenderPresent(renderer);
    SDL_Delay(10000);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(win);
    SDL_Quit();
    return 0;
}
```

(b)

1.6 Recopiez le code (b) ci-dessus. Essayez de comprendre où se trouvent les éléments permettant de régler les caractéristiques suivantes :

- Titre de la fenêtre.
- Dimensions et position de la fenêtre.
- La couleur de fond.
- Délais d'attente pour terminer l'application



Si tout se passe correctement vous devriez obtenir le résultat de l'image ci-dessus.

Concept du cours :

Structure de base : Dans le code que vous avez recopiez, vous avez pu mettre en place la configuration suivante :

- Initialiser la SDL (ligne 9). Cette étape est indispensable car elle permet d'utiliser ensuite les modules liés à la gestion du rendu, du temps, de l'audio, des joysticks, contrôleurs de jeux ou la gestion des événements. Ici pour allez plus vite, nous avons mis le paramètre pour initier tous les modules.
- Construction d'une fenêtre (ligne 10). Dans les paramètres du constructeur `SDL_Window`, en dehors de la taille vous pouvez par exemple rajouter un paramètre pour changer le type de fenêtre (par exemple en combinant plusieurs propriétés : `SDL_WINDOW_SHOWN | SDL_WINDOW_RESIZABLE`).
- Création d'un `Renderer` (ligne 13). Un `Renderer` est une structure qui permet d'avoir un rendu du contenu de la fenêtre. On peut demander à SDL différents type de rendu, et dans notre exemple nous demandons un rendu accéléré avec la carte graphique et le -1 signifie qu'il utilisera le premier pilote graphique disponible.
- Remplissage du fond (lignes 14, 15) : le `Renderer` s'utilise en définissant une couleur avant d'effectuer une opération telle que le remplissage (`SDL_renderClear()`). Même après avoir été exécutées, l'effet de ces fonctions n'est pas encore visible à l'écran tant que l'instruction n'a pas été donnée par le programmeur.
- Présenter le `Renderer` (ligne 16) : la dernière étape est de rendre visible le résultat final de ce qui a été calculé aux étapes précédentes (avec la fonction `SDL_RenderPresent()`). Cette stratégie en deux étapes principales permet optimiser les opérations et d'éviter que le joueur puisse voir des étapes intermédiaires dans la construction de l'image.

Partie II: Base de casse brique balle et affichage joueur (20 ∼ 30 min)

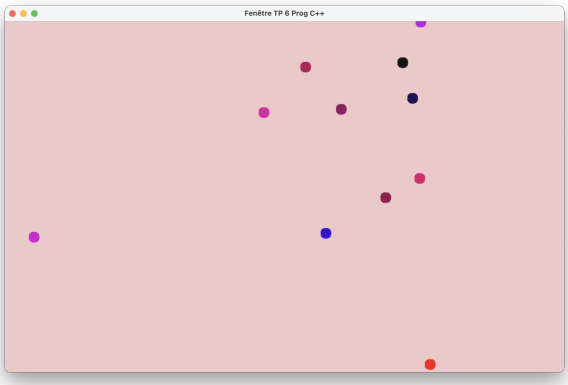
Pour faire le jeu de casse brique, une base a été faite que vous pouvez récupérer sur *moodle*.

2.1 Récupérez les trois fichiers `jeuCasseBrique.cpp`, `PlatformeCasseBrique.cpp` et `PlatformeCasseBrique.h`. Intégrez les deux fichiers précédents dans votre projet et recopiez le contenu de `jeuCasseBrique.cpp` dans votre fichier principal.

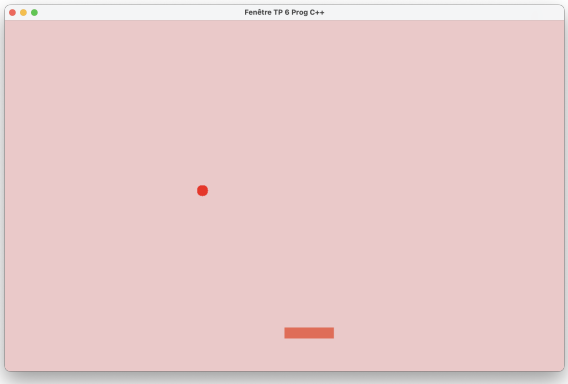
2.2 Lancez votre programme et vérifiez que vous obtenez une balle rebondissant sur les parois de la fenêtre comme sur l'image suivante.

2.3 Dans votre fichier principal, repérez comment fonctionne la fin du jeu et la boucle principale et repérez où se situe la gestion du clavier. Modifiez le code façon à ce que l'utilisateur puisse arrêter proprement une partie en fermant la fenêtre.

2.4 Afin de tester si la gestion multi-balles fonctionne bien, dans la méthode `initJoueurBalle`, modifiez le code de façon à ajouter une dizaine de balles définies avec des positions, des vitesses et des couleurs aléatoires.



2.5 Commentez la code de la question précédente de façon à avoir à nouveau qu'une seule balle, et appelez l'affichage du plateau avec la méthode `render` dans la méthode `tick` pour avoir un rendu comparable à l'image suivante.



Partie III: Déplacement joueur et rebond sur plateau (15 ∼ 20 min)

Dans cette partie, il s'agit de pouvoir gérer correctement les déplacements du joueur et les rebonds de la balle sur le joueur.

3.1 Dans la méthode `tick`, dans la partie de gestion du clavier, modifiez les coordonnées du plateau de façon à pouvoir faire bouger le plateau à droite ou à gauche avec les flèches gauche/droite du clavier.

Comme vous pouvez le voir, l'utilisation de ce mode de déplacement ne parait pas exploitable car trop lent pour un jeu de casse brique. Une meilleure stratégie est de définir une vitesse au plateau et de mettre à jour sa position dans la boucle principale à partir de sa vitesse. La valeur de la vitesse sera alors gérée en fonction du clavier (touche enfoncée = vitesse non nulle, touche non enfoncée = vitesse nulle).

3.2 Pour gérer le déplacement du plateau du joueur en fonction de sa vitesse, rajoutez dans le plateau un attribut `vitesse` de type float initialisé à 2.0 et exploitez cet attribut dans la méthode `miseAjourPosPlateau()` de façon à mettre à jour la position du plateau en fonction de la vitesse.

3.3 Dans la méthode `tick`, rajoutez l'appel à la fonction précédente et modifiez les valeurs de la vitesse du joueur dans la gestion du clavier. Le plateau du joueur doit désormais pouvoir se déplacer beaucoup plus rapidement.

3.4 Complétez la méthode `collision` de la structure `PlateauJeu` de façon à renvoyer vrai si les coordonnées de la balle entre en collision avec du plateau avec et faux sinon. On pourra simplement tester les coordonnées x et y de la balle et du plateau.

3.5 Modifiez la fonction `miseAjourVectVitesse` de façon à simuler le rebond de la balle sur le plateau du joueur.

Partie IV: Génération des briques : affichage et collision (10 ∼ 15 min)

4.1 Dans le construction de la classe `PlateformeCasseBrique`, rajoutez la génération des briques et les stocker dans le vecteur associé (`monVectBriques`).

4.2 Complétez la méthode `render` et l'appelez dans la méthode `tick`.

Enfin il reste à gérer la collision de la balle sur les briques.

4.3 Complétez la fonction `collision` de la structure `Brique`. Pour simplifier vous pouvez utiliser les boites englobantes et la fonction SDL : `SDL_HasIntersection` en prenant en paramètres deux rectangles.

4.4 Intégrez la fonction précédente dans la méthode `miseAjourVectVitesse`. Vous devriez pouvoir voir la balle rebondir sur les briques.

Partie V: Gestion des briques et vies du joueur (10 ∼ 15 min)

5.1 En vous inspirant du TP précédant, finalisez le jeu pour avoir les briques qui disparaissent une fois touchée. Rappel : l'idée peut être simplement de ne pas afficher les briques n'ayant plus de résistance et ensuite de ne pas les prendre en compte pour le rebond.

5.2 Même question que précédemment, mais en intégrant les briques bonus qui dupliquent les balles.

5.3 Dans la méthode `tick()`, détectez quand la balle passe sous le plateau du joueur et dans ce cas là, mettez à jour un de ses attribut pour qu'elle ne soit plus intégrée dans le jeu.

5.4 Quand la balle est perdue, ré initialisez le joueur et la balle en utilisant une méthode existante.

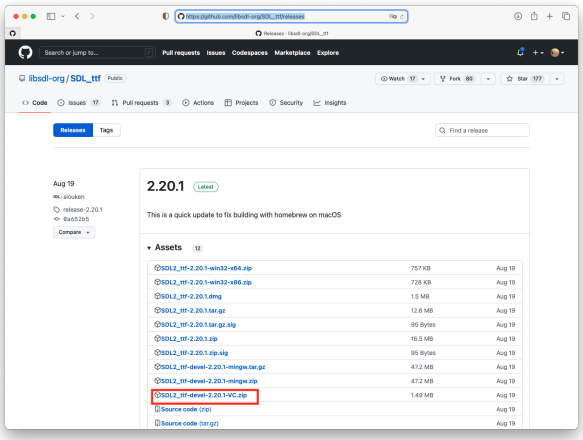
5.5 Fixez un nombre maximun de trois plateaux que vous dessinerez en bas et faire en sorte qu'ils disparaissent à chaque fois que le joueur perd.

Partie VI: Affichage du score et message de fin (20 ∼ 25 min)

6.1 Rajoutez un attribut de classe permettant de d'enregistrer le score du joueur (nombre de briques cassées).

6.2 Pour afficher du texte en SDL, il est nécessaire d'ajouter l'élément *SDL_ttf*. Cette librairie s'installe de la même manière que pour la première partie de cette séance. Elle est disponible à l'adresse suivante :

`https://github.com/libsdl-org/SDL_ttf/releases`



6.3 Testez l'affichage en complétant votre code à partir du code suivant :

```cpp
int main(int argc, char *argv[])
{
    SDL_Init(SDL_INIT_EVERYTHING);
    SDL_Window *win = SDL_CreateWindow("Fenêtre TP6 CPP",
                                SDL_WINDOWPOS_CENTERED,
                                SDL_WINDOWPOS_CENTERED,
                                ReglageJeu::LARGEUR_FENETRE, ReglageJeu::HAUTEUR_FENETRE,
                                    SDL_WINDOW_RESIZABLE);
    if(TTF_Init() < 0) {
        std::cout << "Couldn't initialize TTF lib: " << TTF_GetError() << std::endl;
        return 1;
    }

    SDL_Renderer *renderer = SDL_CreateRenderer(win, -1,
                                            SDL_RENDERER_PRESENTVSYNC);

    PlateformeCasseBrique jeu(renderer);
    while (!jeu.gameOver()) {
        jeu.tick();
    }
    std::string score_text = "GAME OVER! Score:  " + std::to_string(jeu.getScore());
    SDL_Color textColor = { 255, 250, 25, 255 };
    auto font = TTF_OpenFont("FreeSans.ttf", 54);

    SDL_Surface* textSurface =  TTF_RenderText_Solid(font, score_text.c_str(), textColor);
    SDL_Texture* texture = SDL_CreateTextureFromSurface(renderer, textSurface);
    SDL_Rect r; r.x=ReglageJeu::LARGEUR_FENETRE/4; r.y = ReglageJeu::HAUTEUR_FENETRE/2;
    r.w= 300; r.h = 200;
    SDL_QueryTexture(texture, NULL, NULL, &r.w, &r.h);
    SDL_RenderCopy(renderer, texture, NULL, &r);
    SDL_RenderPresent(renderer);
    SDL_Delay(10000);
    return EXIT_SUCCESS;
```

6.4 En vous inspirant du code ci-dessus, modifiez votre code façon à pouvoir afficher le code en direct dans le jeu.