**SDL 2 What is that ?**

**Partie 1**

# Qu’est-ce que la SDL2 ? (10 points)

Simple DirectMedia Layer est une bibliothèque de développement multiplateforme conçue pour fournir un accès de bas niveau au matériel audio, clavier, souris, joystick et graphique via OpenGL et Direct3D.

Wiki mis à part, SDL ou Simple DirectMedia Layer est une bibliothèque de développement portable sur la plupart des plateformes sous licence zlib.

# Que peut faire la SDL 2 ? (10 points)

La SDL 2 permet d’accéder au matériel audio et de contrôle, ainsi qu’offrir un accès graphique avec les bibliothèques OpenGL et Direct3D.

Concrètement elle permet de gérer les évènements et les inputs utilisateurs ainsi que tous les retours graphiques et audio.

# Sur quelles plateformes fonctionne la SDL2 ? (10 points)

SDL prend officiellement en charge Windows, Mac OS X, Linux, iOS et Android.  
La prise en charge d'autres plates-formes peut être trouvée dans le code source.

# Comment obtenir et installer la SDL2 sur un EDI ? (10 points)

* Télécharger la dernière version de SDL2 depuis le site officiel <https://www.libsdl.org/>
* Choisir dans Development Librairies le lien pour windows si vous êtes sous windows : <https://www.libsdl.org/release/SDL2-devel-2.0.10-mingw.tar.gz>
* Ensuite il faut lancer un projet codeblocks, ajouter le fichier SDL2.dll du dossier i686-w64-mingw32\bin à la racine du projet.
* Après dans les options de construction du projet, dans les options de linker il faut ajouter les .lib que l’on trouve dans le dossier i686-w64-mingw32\lib, soit les fichiers libSDL2main.a et libSDL2.dll.a aux librairies et ajouter -lmingw32 dans les autres options de linker.
* Puis dans les répertoires de recherche, pour le compiler il faut ajouter le dossier include, et pour le linker il faut ajouter le dossier lib.

Ainsi on a configuré la SDL2 pour Codeblocks en C.

# Quelles sont les différences entre SDL 1 et SDL 2 ? (5 points)

La SDL2 est une nouvelle version de la SDL1.2 :  
-Elle permet en plus de tout ce que faisait la SDL1, d’afficher de la 3D.  
-Elle ajoute du support pour OpenGL 3, permet à présent d’avoir plusieurs fenêtres, moniteurs et périphériques audio.  
-Elle ajoute un support pour Android et iOS.  
-Elle permet un meilleur support pour les inputs : manette, clavier, tactile, …  
-Elle passe sous licence zlib à la place de la LGPL.

# Pour utiliser la SDL2 dans votre projet, vous devez initialiser l’instance SDL2, quelles sont les étapes d’initialisation à réaliser pour utiliser la lib SDL2, donnez le code correspondant et expliquez précisément chaque instruction. (15 points)

Pour initialiser l’instance SDL2, il faut d’abord inclure la SDL2 avec l’instruction : #include "SDL2/SDL.h".  
Ensuite il faut modifier le int main() tel que : int main(int argc, char \*argv[]).  
Enfin il faut initialiser la SDL avec la fonction SDL\_Init(), pour initialiser tous les sous-systèmes on utilise SDL\_Init(SDL\_INIT\_EVERYTHING).  
Il ne faut pas oublier de quitter la SDL à la fin avec la fonction SDL\_Quit().

Pour l’exemple, voir le code du programme ci-joint.

# Comment créer une fenêtre avec la SDL 2, donnez et expliquez le code correspondant à la création d’une fenêtre avec la SDL2. (10 points)

Pour créer une fenêtre avec la SDL 2 on utilise la fonction SDL\_CreateWindow() qu’on attribue à notre pointeur de type SDL\_Window.  
Cette fonction prend pour paramètre dans l’ordre : Le titre de la fenêtre, sa position initiale en x, sa position initiale en y, sa largeur en pixel, sa hauteur en pixel et divers instructions rassemblés dans les SDL\_WindowFlags.  
Ce qui donne l’instruction exemple telle que : pWindow=SDL\_CreateWindow("SDL2WhatIsThat",SDL\_WINDOWPOS\_CENTERED,SDL\_WINDOWPOS\_CENTERED,1024,780,SDL\_WINDOW\_SHOWN);  
L’instruction SDL\_WINDOW\_SHOWN rend la fenêtre visible à son apparition.

On n’oublie pas de mettre fin à la fenêtre avec la fonction SDL\_DestroyWindow().

Pour l’exemple voir le code du programme ci-joint.

# Gestion d’une fenêtre SDL 2, donnez les primitives associées à la fenêtre SDL2. (10 points)

Les primitives associées à la fenêtre SDL2 sont :

void SDL\_SetWindowTitle(SDL\_Window\* window, **const** char\* title);

Qui permet de donner un titre à la fenêtre, on y passe comme argument, le pointeur de type SDL\_Window\* et la chaine de caractère qui constitue le titre.

void SDL\_SetWindowIcon(SDL\_Window\* window, SDL\_Surface\* icon);

Qui permet de mettre une icône à la fenêtre, on y passe comme argument, le pointeur de type SDL\_Window et le pointeur de type SDL\_Surface\* pour la future icône de la fenêtre.

**const** char\* SDL\_GetWindowTitle(SDL\_Window\* window);

Qui permet de récupérer le titre de la fenêtre, elle renvoie une chaine après qu’on l’y ait passé comme argument le pointeur de la fenêtre type SDL\_Window\*.

void SDL\_SetWindowPosition(SDL\_Window\* window , int x, int y);

Qui permet de déplacer la fenêtre à la position (x , y)

void SDL\_SetWindowSize(SDL\_Window\* window , int w , int h);

Qui permet de modifier la taille de la fenêtre avec w la largeur et h la hauteur en pixels.

void SDL\_GetWindowPosition(SDL\_Window\* window, int\* x, int\* y);

Qui permet de récupérer la position de la fenêtre dans les pointeurs x et y.

void SDL\_GetWindowSize(SDL\_Window\* window, int\* w, int\* h);

Qui permet de récupérer la taille de la fenêtre dans les pointeurs w et h.

void SDL\_MaximizeWindow(SDL\_Window\* window);

Qui permet d’agrandir la fenêtre au maximum.

void SDL\_MinimizeWindow(SDL\_Window\* window);

Qui permet de réduire la fenêtre au minimum.

void SDL\_RestoreWindow(SDL\_Window\* window);

Qui permet de restaurer la fenêtre.

int SDL\_SetWindowFullscreen(SDL\_Window\* window , SDL\_bool fullscreen);

Qui permet de rentrer et de sortir du mode plein écran en envoyant : SDL\_TRUE pour rentrer en mode plein écran ou SDL\_FALSE pour sortir du mode plein écran.  
Il faut noter que la fonction renvoie 0 en cas de succès et une valeur négative en cas d’erreur.

int SDL\_SetWindowBrightness(SDL\_Window\* window, float luminosite);

Qui permet de définir la luminosité de la fenêtre par rapport à celle de l’écran.  
Le float varie de 0 à 1, à 1 la luminosité vaut celle de l’écran, à 0 la fenêtre est noire.  
Il faut noter que la fonction renvoie 0 en cas de succès et une valeur négative en cas d’erreur.

float SDL\_GetWindowBrightness(SDL\_Window\* window);

Qui permet de récupérer la luminosité de la fenêtre.

# Gérer un rendu : qu’est-ce qu’un renderer, comment est-il associé à la fenêtre SDL 2 ? Donnez et expliquez le code permettant de gérer un rendu, création, affichage, effacement… (15 points)

Le Renderer est l’espace où l’on peut dessiner, écrire, afficher dans la fenêtre.  
Si la fenêtre était un tableau, le Renderer en serait la toile.  
Un Renderer ne peut donc pas exister sans fenêtre.

Avec un pointeur de type SDL\_Renderer\* on crée le rendu en lui affectant le résultat de la fonction SDL\_Renderer\* SDL\_CreateRenderer(SDL\_Window\* window, int index, Uint32 flags).  
SDL\_Window\* est le pointeur de la fenêtre qu’on associe au renderer.  
Il faut mettre l’index à -1, il s’agit en vérité de l’index du pilote pour le rendu à initialiser, mais -1 permet d’initialiser le premier qui correspond aux flags qu’on lui demande.  
Et enfin les flags se trouvant dans : SDL\_RendererFlags, permettant d’utiliser l’accélération matérielle, de synchroniser le rendu avec le rafraichissement, d’associer le renderer à une texture ou d’associer le renderer à une surface.

Lorsqu’on modifie le rendu il faut le réactualiser et l’afficher avec la fonction void SDL\_RenderPresent(SDL\_Renderer\* renderer) qui demande notre pointeur de type SDL\_Renderer\* comme argument.

Enfin pour effacer le rendu quand on a finit de l’utiliser, on efface son contenu avec la fonction int SDL\_RenderClear(SDL\_Renderer\* renderer), ou on le détruit avec la fonction void SDL\_DestroyRenderer(SDL\_Renderer\* renderer) ;

# SDL\_Point et SDL\_Rect : Donnez et expliquez le code de déclaration d’un SDL\_Point et SDL\_Rect (5 points)

SDL\_Point est une structure contenant deux entiers (x,y), on déclare une variable de ce type de la manière suivante : SDL\_Point monPoint ={30,40} ; Avec ce code, j’ai créé monPoint de type SDL\_Point de coordonnées x=30 et y=40.

SDL\_Rect est une autre structure contenant l’origine du Rectangle en x,y ainsi que sa largeur vers la droite et sa hauteur vers le bas à partir de l’origine. On déclare une variable de ce type de la manière suivante : SDL\_Rect monRectangle={0,0,300,400} ; Ainsi, mon Rectangle a comme point de départ : 0,0) et comme dimensions : Largeur=300 Hauteur=400.

# Les couleurs avec la SDL2 : donnez les instructions permettant de gérer la couleur avec la SDL2

Pour définir une couleur qu’on va utiliser sur un rendu, on va utiliser la fonction :  
int SDL\_SetRenderDrawColor(SDL\_Renderer\* renderer, Uint8 r, Uint8g, Uint8 b, Uint8 a);  
La fonction renvoie 0 en cas de succès et une valeur négative en cas d’échec.  
Les quatre valeurs r, g, b, a sont de type Uint8, entier sur 8 bits, et représentent dans l’ordre la valeur de 0 à 255 du : rouge, vert, bleu et l’opacité.  
Plus la valeur est proche de 255 plus elle est représentée, et plus elle est proche de 0 moins elle est présente.

# Donnez le code permettant d’afficher un fond rouge dans le rendu.

Le code permettant d’afficher un fond rouge dans le rendu est :

pRenderer = SDL\_CreateRenderer(pWindow, -1, SDL\_RENDERER\_ACCELERATED); //On crée le rendu.

SDL\_SetRenderDrawColor(pRenderer, 255, 0, 0, 255); //On change la couleur du rendu.

SDL\_RenderClear(pRenderer); //On actualise le rendu.

SDL\_RenderPresent(pRenderer); //On affiche le rendu.

Pour l’exemple voir le code du programme ci-joint.

# Dessiner dans le rendu : donnez le code des fonctions permettant de dessiner dans le rendu les formes suivantes : -Carré vide -Carré plein -Cercle vide -Cercle plein

Les codes permettant de dessiner ces formes sont dans les fonctions suivantes :

-Carré vide : void AfficherUnRectangle(SDL\_Renderer\* renderer,SDL\_Rect\* rect) ;

-Carré plein : void RemplirUnRectangle (SDL\_Renderer\* renderer,SDL\_Rect\* rect) ;

-Cercle vide : void DessinerUnCercle(SDL\_Renderer \* pRendu, int nCentreX, int nCentreY, int nRayon) ;

-Cercle plein : void DessinerUnDisque(SDL\_Renderer \* pRendu, int nCentreX, int nCentreY, int nRayon) ;

Pour l’exemple voir le code du programme ci-joint.

# Quelles sont les fonctions permettant de dessiner des points et des lignes ? (5 points)

Pour dessiner des points il y a les fonctions :

int SDL\_RenderDrawPoint(SDL\_Renderer\* renderer,

int x,

int y)

Qui va dessiner un point à ses coordonnées dans le rendu, elle demande le pointeur de rendu, et les coordonnées du point.

int SDL\_RenderDrawPoints(SDL\_Renderer\* renderer,

const SDL\_Point\* points,

int count)

Qui va dessiner des points à leurs coordonnées dans le rendu, elle demande cependant le pointeur de rendu ainsi qu’un tableau de point et le nombre de points à dessiner.

Pour dessiner des lignes il y a les fonctions :

int SDL\_RenderDrawLine(SDL\_Renderer\* renderer,

int x1,

int y1,

int x2,

int y2)

Qui va dessiner une ligne à partie d’un point de départ et d’un point d’arrivée dans le rendu, elle demande le pointeur de rendu, et les coordonnées des deux points.

int SDL\_RenderDrawLines(SDL\_Renderer\* renderer,

const SDL\_Point\* points,

int count)

Qui va dessiner plusieurs lignes qui se suivent à partir d’un tableau de points, elle demande cependant le pointeur de rendu ainsi qu’un tableau de point et le nombre de points, soit le nombre de lignes à dessiner plus un.

# A quoi servent les fonctions SDL\_RenderClear et SDL\_RenderPresent ? (5 points)

La fonction :

int SDL\_RenderClear(SDL\_Renderer\* renderer)

va remplacer la totalité du contenu du rendu par la couleur actuellement sélectionnée.

La fonction :

void SDL\_RenderPresent(SDL\_Renderer\* renderer)

va actualiser le contenu du rendu avec les changements qu’on a effectué depuis le dernier rendu, et l’afficher.

# Expliquez SDL\_Delay. (5 points)

La fonction :

void SDL\_Delay(Uint32 ms)

va faire attendre le programme un certain nombre de millisecondes, passé en argument, avant qu’il ne continue après l’appel de SDL\_Delay.

# Qu’est-ce qu’une surface en SDL2 ? (5 points)

Une surface en SDL2 est une structure comprenant une collection de pixels, et aisément modifiable pixel par pixel. Elle peut par la suite être convertie en texture.

# Donnez le code permettant de créer une surface (10 points)

Pour créer une surface on peut utiliser la fonction :

SDL\_Surface\* SDL\_CreateRGBSurface(Uint32 flags,

int width,

int height,

int depth,

Uint32 Rmask,

Uint32 Gmask,

Uint32 Bmask,

Uint32 Amask)

Qui va renvoyer un pointeur de type SDL\_Surface\* et qui demande en paramètre : les flags qui ne sont pas utiles ici et doivent être mis à 0, la largeur de la surface, la hauteur de la surface, la profondeur en quatrième paramètre est le nombre de bits par pixel, ensuite il y a les masques de couleur rouge, verte, bleue et le masque du composant alpha (la transparence).

Il y a également la fonction :  
SDL\_Surface\* SDL\_CreateRGBSurfaceFrom(void\* pixels,

int width,

int height,

int depth,

int pitch,

Uint32 Rmask,

Uint32 Gmask,

Uint32 Bmask,

Uint32 Amask)

Qui va demander les mêmes paramètres à l’exception du premier où elle souhaite un tableau de pixels pour le mettre dans la surface.

# Dessiner dans une surface : Donnez le code de la fonction SDL\_FillRect (10 points)

Le code de la fonction SDL\_FillRect est :

int SDL\_FillRect(SDL\_Surface\* dst,

const SDL\_Rect\* rect,

Uint32 color)

Pour l’exemple voir le code du programme ci-joint.

# SDL\_BlitSurface(..) : donnez le code pour tester cette méthode. (5 points)

SDL\_BlitSurface permet de copier une surface sur une autre surface en totalité ou en partie.

On peut la tester en copiant notre pSurface déjà créée sur la surface de notre fenêtre.

On calque notre pointeur pWindowSurface sur la surface de notre fenêtre avec la fonction :

SDL\_Surface\* SDL\_GetWindowSurface(SDL\_Window\* window)

Puis on utilise la fonction qu’on veut tester :

int SDL\_BlitSurface(SDL\_Surface\* src,

const SDL\_Rect\* srcrect,

SDL\_Surface\* dst,

SDL\_Rect\* dstrect)

Et on actualise notre surface de fenêtre avec :

int SDL\_UpdateWindowSurface(SDL\_Window\* window)

pWindowSurface=SDL\_GetWindowSurface(pWindow);

SDL\_BlitSurface(pSurface,NULL,pWindowSurface,NULL);

SDL\_UpdateWindowSurface(pWindow);

Si la fenêtre s’affiche en vert, notre pSurface contenant uniquement un rectangle vert, alors le test est concluant.

# Les textures : Qu’est-ce qu’une texture ? Donnez le code permettant de créer une texture. (10 points)

Une texture est une structure contenant des données pixels efficaces relatives aux pilotes.

Pour créer une texture, on utilise la fonction :

SDL\_Texture\* SDL\_CreateTexture(SDL\_Renderer\* renderer,

Uint32 format,

int access,

int w,

int h)

Qui prend comme argument : Le rendu dans lequel la texture va s’insérée, le format des pixels dans la texture, la variable d’accès de la texture qui permet de savoir si elle va changer rarement, fréquemment ou si elle peut être utilisée comme une cible de rendu, ainsi que la largeur puis la hauteur de la texture.

# Dessinez dans une texture : SDL\_RenderTarget, donnez le code pour tester cette méthode. (10 points)

La fonction :

int SDL\_SetRenderTarget(SDL\_Renderer\* renderer, SDL\_Texture\* texture);

permet de donner une cible de type texture lors de l’utilisation des fonctions de dessins qui font appel au type SDL\_Renderer\*

Le code pour tester cette fonction serait :

SDL\_SetRenderTarget(pRenderer, pTexture);

DessinerUnDisque(pRenderer,155,155,150);

SDL\_RenderPresent(pRenderer);

SDL\_Delay(2000);

Et si rien ne s’affiche et que l’on n’a aucune erreur lors de l’exécution, cela signifie qu’on a bien dessiné dans la texture.

Pour afficher la texture on va voir ça dans la question suivante.

Pour l’exemple voir le code du programme ci-joint.

# SDL\_RenderCopy : A quoi sert cette méthode, donnez le code pour tester cette méthode. (10 points)

La fonction :

int SDL\_RenderCopy(SDL\_Renderer\* renderer,

SDL\_Texture\* texture,

const SDL\_Rect\* srcrect,

const SDL\_Rect\* dstrect)

Va copier la texture source en partie ou en totalité et l’appliquer dans le rendu cible en partie ou en totalité.

Pour la tester on peut garder la texture de la question précédente et cette fois-ci tenter de la copier dans le rendu puis d’actualiser le rendu, si cela fonctionne on aura bien appliquer la texture dans le rendu.

Pour l’exemple voir le code du programme ci-joint.

# SQL\_QueryTexture : A quoi sert cette méthode, donnez le code pour tester cette méthode. (10 points)

La fonction :

int SDL\_QueryTexture(SDL\_Texture\* texture,

Uint32\* format,

int\* access,

int\* w,

int\* h)

Permet à partir d’un pointeur de type SDL\_Texture\* de récupérer les informations de la dite texture et de les appliquer aux variables passées en argument qui sont : format, access, w et h, pour le format des pixels, le type d’accès de la texture, la largeur et la hauteur de cette dernière.

Lors de notre affichage de texture précédent, nous avions mis NULL et NULL pour les rectangles sources et de destination, ce qui prend par défaut la valeur la plus grande des deux et l’applique au deux, ça fonctionne mais on ne conserve ni la taille, ni les ratios.  
Avec cette fonction on peut récupérer la taille et la largeur du rectangle de notre texture ce qui va nous permettre de conserver la taille et les ratios de notre source.

Pour la tester on va :

-Créer un rectangle à partir de la taille de notre texture.  
-Appliquer notre texture au rendu en utilisant les paramètres de ce rectangle à l’origine pour les deux.  
-On se garde de vérifier que notre rectangle texture est plus grand que notre fenêtre et donc notre rendu, car ici ce n’est pas le cas dans nos tests, si le doute se posait, on aurait pu vérifier la taille de notre fenêtre en la comparant avec ce rectangle qu’on a créé.

Pour l’exemple voir le code du programme ci-joint.

# Les Images : La SDL2 ne prend en charge nativement que les formats bmp, si vous souhaitez utiliser d’autres formats vous devez associer la lib SDL2\_image à votre projet. SDL\_LoadBMP, est la méthode permettant d’utiliser des bmp, donnez le code permettant de tester cette fonction. (10 points)

La fonction :

SDL\_Surface\* SDL\_LoadBMP(const char\* file)

Nous permet de charger une image format bitmap dans une surface, par ailleurs l’icone de la fenêtre est une surface, on peut donc tester la fonction en appliquant la surface créée à l’icone de la fenêtre avec le code suivant :

pSurface=SDL\_LoadBMP("Assets/BMP/Cirno.bmp");  
SDL\_SetWindowIcon(pWindow,pSurface);

Pour l’exemple voir le code du programme ci-joint.

# Donnez le code permettant de créer une texture à partir d’une image (bmp) (10points)

Comme une image chargée avec SDL\_LoadBMP se stocke dans un pointeur de type SDL\_Surface, pour créer une texture à partir de celle-ci on peut utiliser la fonction :

SDL\_Texture\* SDL\_CreateTextureFromSurface(SDL\_Renderer\* renderer,

SDL\_Surface\* surface)

Ce qui nous donne le code suivant :

pSurface=SDL\_LoadBMP("Assets/BMP/Cirno.bmp");

pTexture=SDL\_CreateTextureFromSurface(pRenderer,pSurface);

Ainsi on a une image chargée dans notre texture.

Pour l’exemple voir le code du programme ci-joint.

# Prenez une image de votre choix au format bmp et testez les fonctions précédentes en ajoutant le code permettant d’afficher l’image dans le rendu. (15 points)

A partir du code précédent on va appliquer notre texture dans notre rendu et l’appliquer :

SDL\_QueryTexture(pTexture,NULL,NULL,&nLargeur,&nHauteur);

monRectangle.w=nLargeur;

monRectangle.h=nHauteur;

SDL\_RenderCopy(pRenderer,pTexture,&monRectangle,&monRectangle);

SDL\_RenderPresent(pRenderer);

On commence par la fonction SDL\_QueryTexture pour appliquer dans les variables nLargeur et nHauteur les dimensions de notre texture et donc de notre image.  
On les applique à notre rectangle qui est toujours positionné à l’origine.  
Puis on va copier notre texture dans notre rendu et l’afficher !

Pour l’exemple voir le code du programme ci-joint.

# Depuis le début nous créons des instances permettant de manipuler la sdl2, cependant nous oublions une étape essentielle dans son utilisation, la destruction des instances, faites un relevé des instances crées et identifier les méthodes que vous devez utiliser pour libérer celles-ci. (15 points)

Les instances qu’on a créées et utilisées depuis le début sont :

SDL\_Window \*pWindow

SDL\_Renderer \*pRenderer

SDL\_Surface \*pSurface

SDL\_Surface \*pWindowSurface

SDL\_Texture \*pTexture

Pour les libérer on doit utiliser :

Pour la texture : SDL\_DestroyTexture(pTexture);

Pour les surfaces : SDL\_FreeSurface(pSurface);  
SDL\_FreeSurface(pWindowSurface);

Pour le rendu : SDL\_DestroyRenderer(pRenderer);

Pour la fenêtre : SDL\_DestroyWindow(pWindow);

Et tout à la fin de l’utilisation de la SDL, il est recommander d’appeler : SDL\_Quit().

Pour l’exemple voir le code du programme ci-joint.

# Vous avez codé votre programme, fonctions et procédures à la volée afin de tester plus précisément les méthodes. A l’aide du type structuré ci-dessus, je vous demande de reprendre votre programme afin de l’améliorer et de proposer suivant les règles de la programmation modulaire et des notions de patterns (MVC) une nouvelle version plus structurée et plus performante que la version précédemment proposée. (30 points)

Typedef struct sdl\_manager

{

SDL\_Window \*pWindow ;

SDL\_Renderer \*pRenderer ;

SDL\_Texture \*pTexture ;

SDL\_Surface \*pSurface ;

}

Voir le projet pour le code final.