Documents supplémentaires :
http://www.ltam.lu/cours-c//prg-c_c.htm
https://openclassrooms.com/courses/apprenez-a-programmer-en-c/

API : SDL2

2016

Projets langage C

jls
iut
23/06/2016

Table des matières

programmation SDL 2

Introduction

Ce document permet d'apprendre à utiliser les bibliothèques graphiques SDL2 et les bibliothèques fournies sur github et surtout propose à la fin des projets permettant d'aborder tous les concepts de la programmation avancée dans un projet ludique.

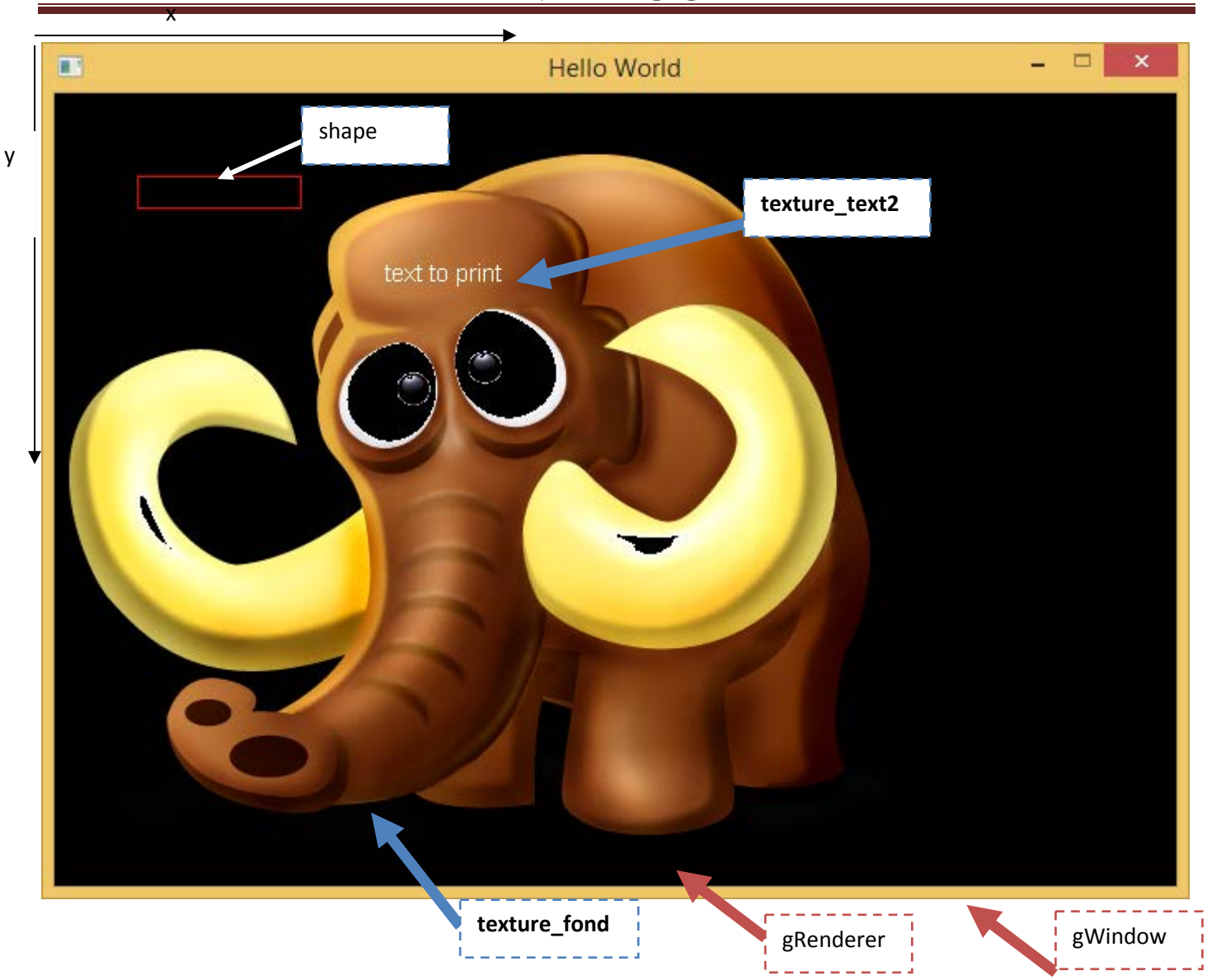
Bibliographie

https://github.com/RyanSwann1/SDL2-Space-Invaders-V1
http://lazyfoo.net/tutorials/SDL/
http://www.meruvia.net/index.php/big-tuto-sdl-2-nouveau
http://programmersranch.blogspot.fr/2014/02/sdl2-keyboard-and-mouse-movement-events.html
https://deptinfo-ensip.univ-poitiers.fr/ENS/doku/doku.php/stu:progc:sdls

Première application :

Le code que nous allons analyser est donné dans la documentation (main\docs\html\index.html)

```
int main(int argc, char ** argv){
SDLS_init("Hello World");
// open image and copy to window
SDL_Texture * texture_fond =SDLS_loadImage("images//elephant.png"); // open image
SDLS_copyTexture(texture_fond, 0,0);// copy in video cache buffer in the top left
// create shape and copy to video cache buffer (automatic)
SDLS_setColor(255,0,0); // red color
SDLS_drawLineRect(50,50,100,20); // rectangle in 50,50 size 100x20
// open Font and copy to video cache buffer
TTF_Font * ttfFont= SDLS_loadFont("fonts\\arial.ttf",15);// load font in RAM, size 15
SDL_Texture *texture_text2= SDLS_loadText(ttfFont,"text to print", 255,255,255); // copy text to
print (in white color) to texture
SDLS_copyTexture(texture_text2,200,100); // copy texture to window on position 200,100
// copy video cache buffer in video buffer (double buffering)
SDLS_displayAll(); // copy cached Video Buffer to Video buffer
SDL_Delay(5000); // wait 5s to see the window
SDLS_cleanup(); // erase memory}
```



Cette première application permet d'afficher une image de fond, un texte et une forme (shape) pendant 5s.

Quelques explications :

`SDLS_init("Hello World");`
cette fonction permet d'initialiser créer le contexte graphique, la structure fenêtre (gWindow) et le renderder (palette) permettant le dessin sur la fenêtre. Cette fonction SDLS_init se trouve dans le fichier SDLS.c, les variables globales gWindow et surtout gRenderer sont utilisés dans de nombreuses fonctions de dessins et ne sont donc pas passés en paramètres. Pour cette raison , la bibliothèque de fonction sdls.c ne permet de travailler que sur une seule fenêtre.

SDL_Texture * texture_fond =`SDLS_loadImage("images//elephant.png");`
Ouverture du fichier image et placement dans une texture texture_fond. Toutes les images et les fonts (qui sont aussi des images de lettres) doivent automatiquement être placé dans une texture pour être affichées sur le renderer. Une texture est un élément graphique en RAM .

`SDLS_copyTexture`(texture_fond, 0,0);
Copie de la texture sur le renderer à la position 0,0. Ici l'éléphant est de taille importante et donc sera position sur presque toute la fenêtre. Tout ce qui est blanc dans l'image, ou bien transparent devient transparent.

```
SDLS_setColor(255,0,0); // red color
SDLS_drawLineRect(50,50,100,20); // rectangle in 50,50 size 100x20
```
Ces 2 lignes, permettent de choisir une couleur, et d'appliquer cette couleur à la forme (shape) que l'on va afficher sur le renderer. Il est possible d'afficher des rectangles, lignes, pointillés, pixels, mais aussi grace à la bibliothèque gfx_primitives des courbes de bézier, des triangles, des lignes épaisses, des cercles, des ellipses, des arcs de cercles. Toutes ces formes n'utilisent pas de texture mais sont directement appliquées sur le renderer.

```
TTF_Font * ttfFont= SDLS_loadFont("fonts\\arial.ttf",15);// load font in RAM, size 15
```
Chargement de la font et de sa taille en mémoire RAM.

```
SDL_Texture *texture_text2= SDLS_loadText(ttfFont,"text to print", 255,255,255);
```
Chargement de la font dans une texture, cette texture est prête à être copiée sur le renderer

```
SDLS_copyTexture(texture_text2,200,100);
```
Copie de la texture dans le renderer

```
SDLS_displayAll();
```
Copie du renderer (VRAM non affichée) dans la VRAM d'affichage ; cette technique est classique pour l'affichage vidéo et notamment pour les jeux vidéos. La préparation de l'image se fait sur une zone de la mémoire non utilisée par la carte graphique. C'est le renderer qui est la structure qui pointe vers cette zone mémoire. La fonction SDLS_displayAll copie alors l'image vers la VRAM d'affichage, ce qui permet une fluidité dans l'affichage.

L'affichage en x,y se fait en partant du bord haut et gauche. La fenêtre a une taille de 640x480 pixels.

Premiers exercices de compréhension

Pour cette première partie, copier le code dans le projet testForUser.cbp et faites les modifications directement dans le fichier testForUser.c

1. Déplacer le texte à l'intérieur du cadre rouge. Tester
2. Ajouter la fonction `SDLS_eraseWithBackgroundColor(255,255,255);` juste après la fonction init. Que se passe-t-il ? Mettre le fond en blanc.
3. Ajouter un rectangle plein vert, et une ligne Bleu après 2 s. Tester.
4. Vous ajoutez `SDLS_eraseWithBackgroundColor(255,255,255);` après la fonction displayAll puis afficher un rectangle plein vert, et une ligne Bleu
5. Ouvrez le fichier testGFX.c, en vous aidant de la documentation de gfx primitive, afficher un triangle et un rond plein, en plus des autres formes. Que se passe-t-il si l'on affiche 2 formes l'une sur l'autre ?
6. Copier ce code avant SDLS_displayAll(). Tester.

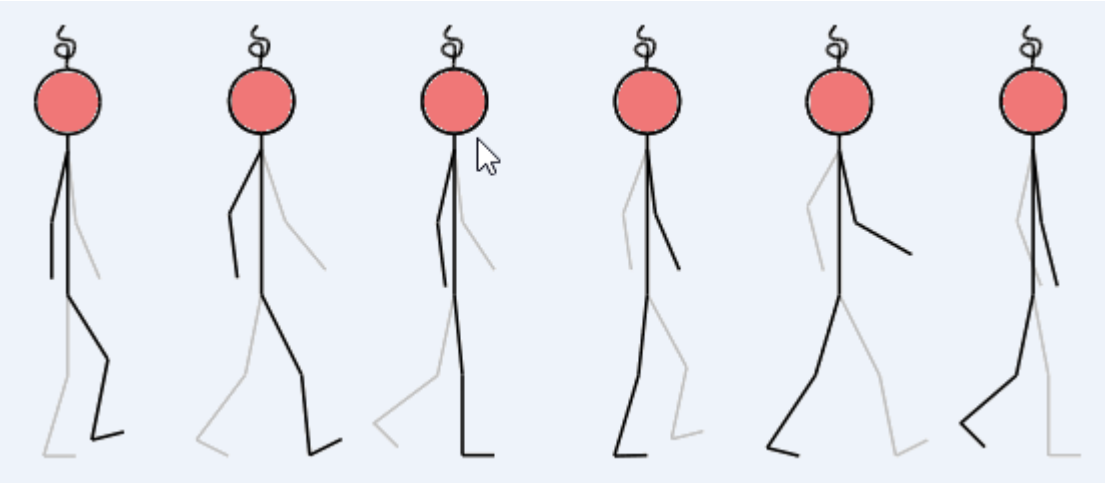
```
SDL_Rect gSpriteSourceBalle[ 4 ]=
    {
        {0,0,25,25}, //balle rouge
        {25,0,25,25}, //balle verte
        {0,25,25,25},  //balle jaune
        {25,25,25,25}
    }; //balle bleu
    SDL_Rect gDestBalle= {62,62,25,25}; // position dans la fen?tre

    SDL_Texture * texture_balles =SDLS_loadImage("images//balles.png"); // copy image
(4 balles of size 25x25) in texture
    SDLS_copyTextureEx(texture_balles,gSpriteSourceBalle[2],gDestBalle); // display
first balle in rectangle gDestBalle of window
```

Explication : le fichier balles.png possède 4 images de taille 25x25. La fonction SDLS_copyTextureEx permet de copier une partie d'une image vers le rectangle de destination dans le renderer. A la différence

avec la fonction SDLS_copyTexture qui copiait toute l'image vers le renderer, il est possible ici de choisir la partie de la texture que l'on veut copier. Dans notre cas, nous utilisons la balle jaune.

7. Modifier le programme pour afficher une balle de chaque couleur aux 4 bords de l'image

8. Ouvrir le fichier marche.png et donner ses dimensions (on pourra utiliser gimp ou paint pour connaitre la taille de l'image)
9. Afficher le troisième personnage dans l'image

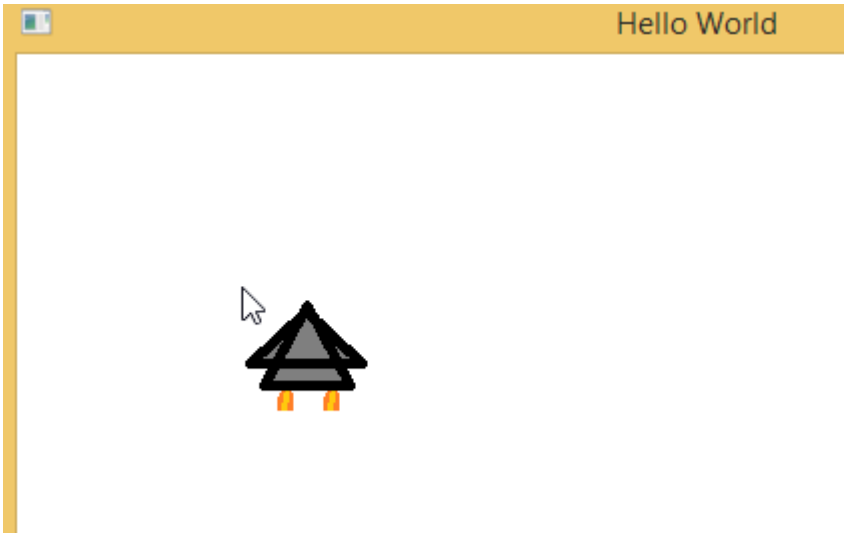


10. Lire la documentation de la fonction SDLS_copyAndRotateTexture et afficher le quatrième personnage avec une rotation de 20° et un déplacement de 50,50.

Deuxième application : les évènements, première animation

Dans ce deuxième programme, nous ajoutons la gestion des évènements clavier ou souris

```
int main(int argc, char ** argv){
int x=100,y=100, quit=0;
SDL_Event event;
// create window with a title
SDLS_init("Hello World");
// open image and copy to window
// load image texture in RAM
SDL_Texture * texture_vaisseau = SDLS_loadImage("images//spaceship.png");
// handle events
while (!quit)
{
SDL_WaitEvent(&event); // c'est bloquant...
switch (event.type){
case SDL_QUIT:
quit = true;
break;
case SDL_MOUSEBUTTONDOWN : // sur le clic de la souris on bouge la forme
if( event.button.button == SDL_BUTTON_LEFT ){
x=event.button.x;
y=event.button.y;
}
break;
case SDL_KEYDOWN: // sur l'utilisation du clavier on bouge la forme
switch (event.key.keysym.sym){
case SDLK_LEFT:--x;break;
case SDLK_RIGHT:++x;break;
case SDLK_UP: --y;break;
case SDLK_DOWN:++y; break;
}
}
SDLS_eraseWithBackgroundColor(255,255,255); // couleur du fond en blanc
SDLS_copyTexture(texture_vaisseau,x,y); // place le vaisseau en x,y
SDLS_displayAll(); // copy cached Video Buffer to Video buffer
}
SDLS_cleanup(); // erase memory
}
```



Explications sur le programme :

Un essai vaut mieux que des explications, lancer le programme et cliquer avec la souris, que se passe-t-il ?
Remarquez que le vaisseau se place en bas à droite du clic de souris.
C'est le code de gestion des évènement et notamment le code ci-dessous qui est activé lors du clic souris :

```
x=event.button.x;
y=event.button.y;
```

La position en x et y de la souris est recopiée dans les variables x,y qui sont ensuite utilisées pour placer la texture du vaisseau, mais ce placement se fait par rapport au bord haut et gauche de la texture.

```
SDLS_copyTexture(texture_vaisseau,x,y);
```

Remarque : une texture est forcément intégré dans un rectangle (type SDL_Rect) comme nous l'avions vu lors de l'utilisation de la fonction `copyTextureEx`.

```
SDL_WaitEvent(&event);
```

La fonction SDL_WaitEvent est une fonction bloquante. Le programme lors que l'appel à cette fonction redonne la main à windows. Le code reste donc bloqué à l'arrivé d'un évènement associé à la fenêtre. Un clic de souris, l'appuie sur une touche du clavier par exemple, va débloquer cette fonction et exécuter le reste du code.

```
SDLS_eraseWithBackgroundColor(255,255,255); // couleur du fond en blanc
SDLS_copyTexture(texture_vaisseau,x,y); // place le vaisseau en x,y
SDLS_displayAll(); // copy cached Video Buffer to Video buffer
```

Remarquez que l'affichage est fait dans la boucle infini, il est évident qu'il faut afficher le vaisseau à chaque fois qu'un évènement est apparu.

Pour finir appuyez sur les touches UP/DOWN/LEFT/RIGHT en continu ou non, et regarder évoluer la position du vaisseau.

Deuxièmes exercices de compréhension

11. Modifier le programme pour qu'à chaque appui sur la touche A la vitesse de déplacement augmente et sur Q pour diminuer. Tester
12. Ouvrir le fichier marche.png et faite avancer le personnage à chaque appui sur la flèche RIGTH. A chaque déplacement allez chercher l'image suivante afin que le personnage se déplacement avec une animation.

Troisième application : animation automatique

Utilisation d'un timer pour animer des formes

```
//this function is called each 100ms and send an event to the main...
Uint32 my_callbackfunc(Uint32 interval, void *param)
{
SDL_Event event;
SDL_UserEvent userevent;
/* In this example, our callback pushes an SDL_USEREVENT event
* into the queue */
userevent.type = SDL_USEREVENT;
userevent.code = 0;
userevent.data1 = NULL;
userevent.data2 = NULL;
event.type = SDL_USEREVENT;
event.user = userevent;
SDL_PushEvent(&event);
return(interval);
}
int main(int argc, char ** argv){
int quit=0,angle=0;
SDL_Event event;
// create window with a title
SDLS_init("Hello World");
// open image and copy to window
SDL_Texture * texture_vaisseau = SDLS_loadImage("images//spaceship.png"); // load image texture
in RAM
Uint32 delay = 100; /* To round it down to the nearest 100 ms */
SDL_TimerID my_timer_id = SDL_AddTimer(delay, my_callbackfunc, NULL); //init timer
// handle events
while (!quit)
{
SDL_WaitEvent(&event); // c'est bloquant...
if(event.type==SDL_QUIT)
quit = true;
if(event.type== SDL_USEREVENT){
angle+=4;
if(angle>360)
angle=0;
}
SDLS_eraseWithBackgroundColor(255,255,255); // couleur du fond en blanc
SDLS_copyAndRotateTexture(texture_vaisseau,SDLS_getScreenWidth()-100,50,angle);
SDLS_displayAll(); // copy cached Video Buffer to Video buffer
}
SDLS_cleanup(); // erase memory
}
```

Explications :

```
Uint32 delay = 100; /* To round it down to the nearest 100 ms */
SDL_TimerID my_timer_id = SDL_AddTimer(delay, my_callbackfunc, NULL); //init timer
```

Notez l'appel à la fonction AddTimer, qui génère un appel automatique à la fonction my_callbackfunc toutes les 100ms dans notre exemple. Nous avons donc 2 fonctions qui sont exécutées en parallèle : la fonction main qui gère l'affichage et le entrées clavier et souris et la fonction my_callbackfunc qui est appelée toutes les 100ms.

Question : comment synchroniser ces 2 fonctions ? Rappelons nous que la fonction waitEvent est une fonction bloquante en attente d'évènement extérieur. Justement, la fonction my_callbackfunc peut générer un évènement qui sera vu par le main au travers de la fonction waitEvent. Le tour est joué.

```
SDL_PushEvent(&event);
```

C'est cette fonction qui permet d'envoyer un évènement de type SDL_USEREVENT qui sera vu dans le main grace à waitEvent. Cet évènement est donc envoyé toutes les 100ms.

```
if(event.type== SDL_USEREVENT){
angle+=4;
if(angle>360)
angle=0;
```

Il ne reste donc plus qu'à incrémenter l'angle à chaque évènement puis à insérer cet angle dans la fonction **SDLS_copyAndRotateTexture** qui copie la texture vaisseau dans le renderer en faisant une rotation de cette image d'un angle calculé précédemment.

Troisièmes exercices de compréhension

13. Reprendre l'exercice précédent et faire l'animation automatique du personnage à la place du vaisseau. Le personnage avancera de gauche à droite puis de gauche à droite
14. afficher un vaisseau et le faire se déplacer avec les flèches gauche et droite en bas de l'écran. L'appui sur A lance des boulets vers le haut de l'écran en ligne droite (vertical). Cette boule sera animée via un timer.

Quatrième programme

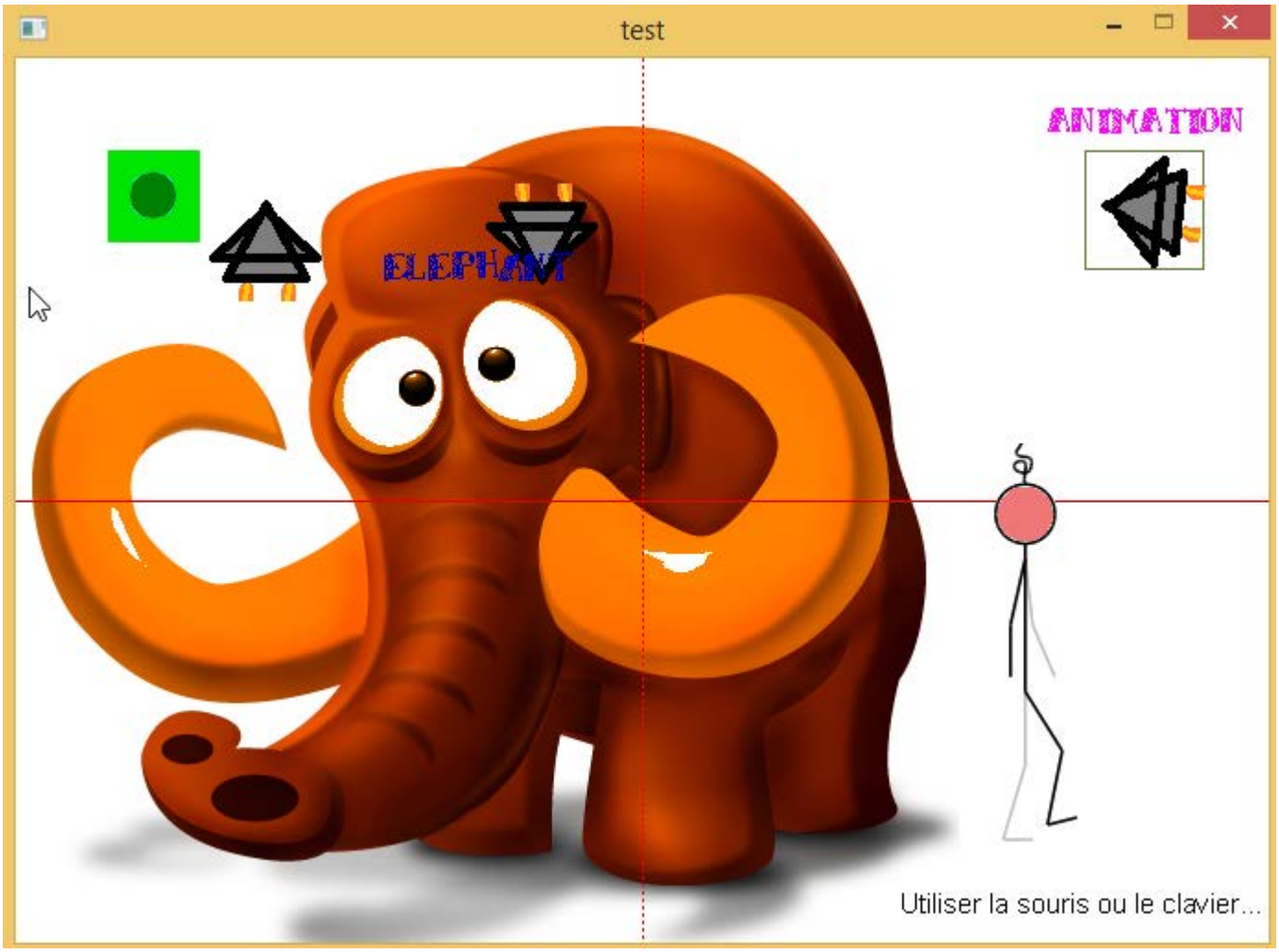
Ouvrir le projet testSDLS.cbp et le lancer. Cliquez sur la souris, à chaque clic, vous déplacer les 2 vaisseaux et la boule au centre du carré vert (en haut à gauche) change de couleur.

15. Quel est le code qui permet faire changer de couleur cette boule ?
16. Avec les flèches, essayez de déplacer les vaisseaux vers le carré vert. Que se passe-t-il ?

Le code permettant par exemple de déplacer (par rapport à l'appui sur la flèche bas) est soumis à la vérification que les 2 rectangles (rectA est la structure permettant de donner la taille et la position du carré vert et rectB est le rectangle dans lequel se trouve la texture du vaisseau. La fonction SDL_HasIntersection permet de savoir si les formes se chevauche, ce qui est important lorsque l'on veut faire un jeu 2D.

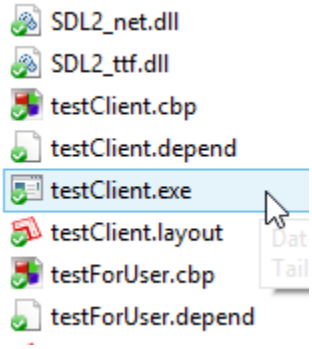
```
        rectB.y=--y;

        if(SDL_HasIntersection(&rectA, &rectB))

          y++;
```

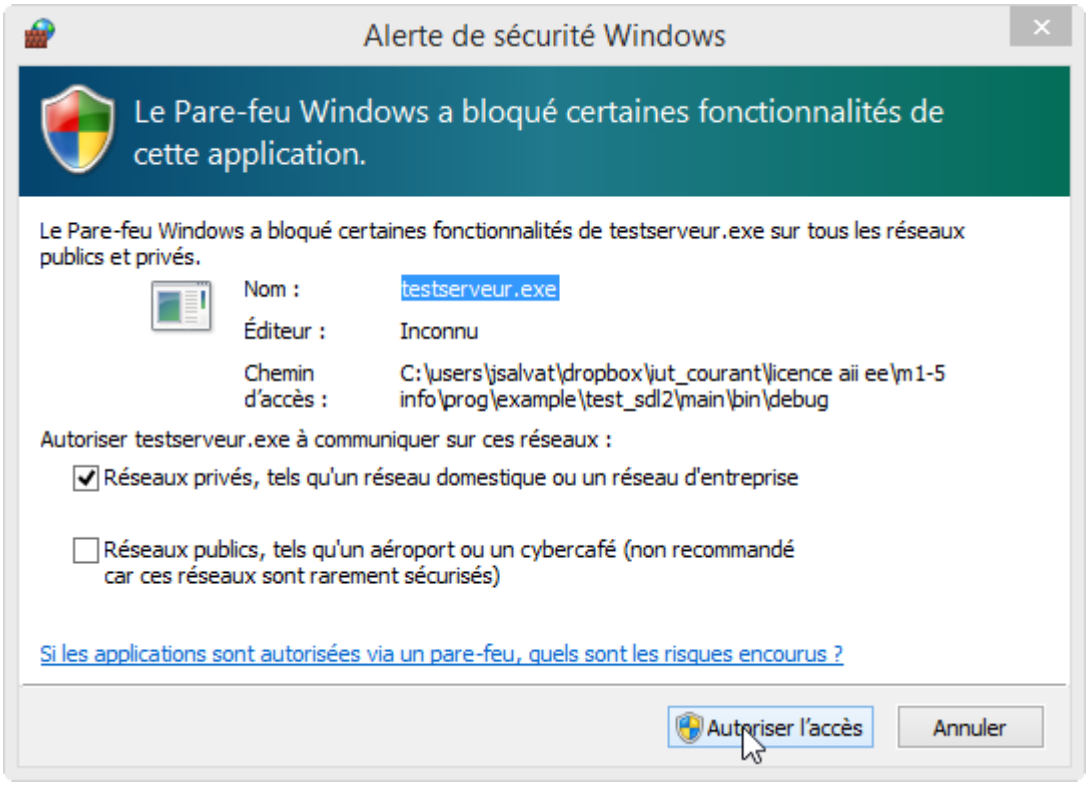
17. Positionner la souris sur le carré vert et cliquer. Que se passe-t-il ? Les vaisseaux ont-ils bougé ? Quel est le code qui bloque le déplacement des vaisseaux ?
18. Enlever le code SDLS_changeColor(texture_fond, 255, 130,0); que se passe-t-il ? Expliquer
19. Une ligne verticale au centre est faite de pointillé, quel est le code qui permet cela ? Et pour la ligne horizontale au centre ?

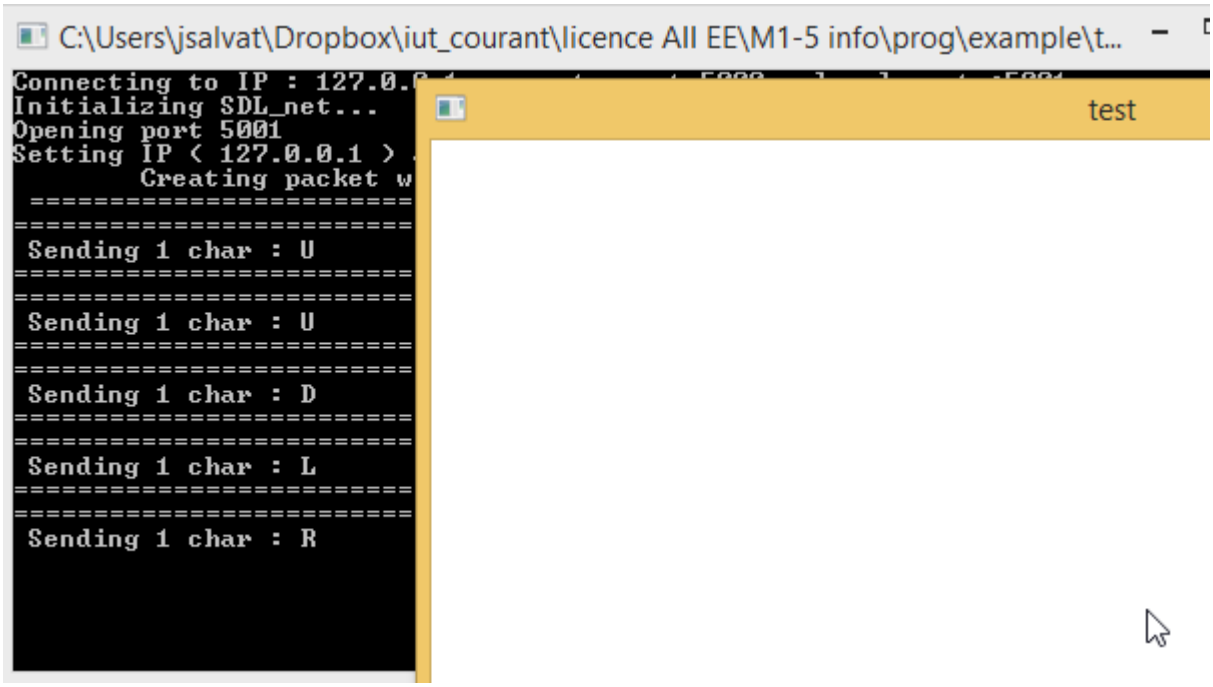


Client/serveur

Ouvrir le projet testserveur.cbp. Lancer le programme testClient.exe qui est un client UDP qui envoie en localhost la touche sur laquelle on vient de taper.

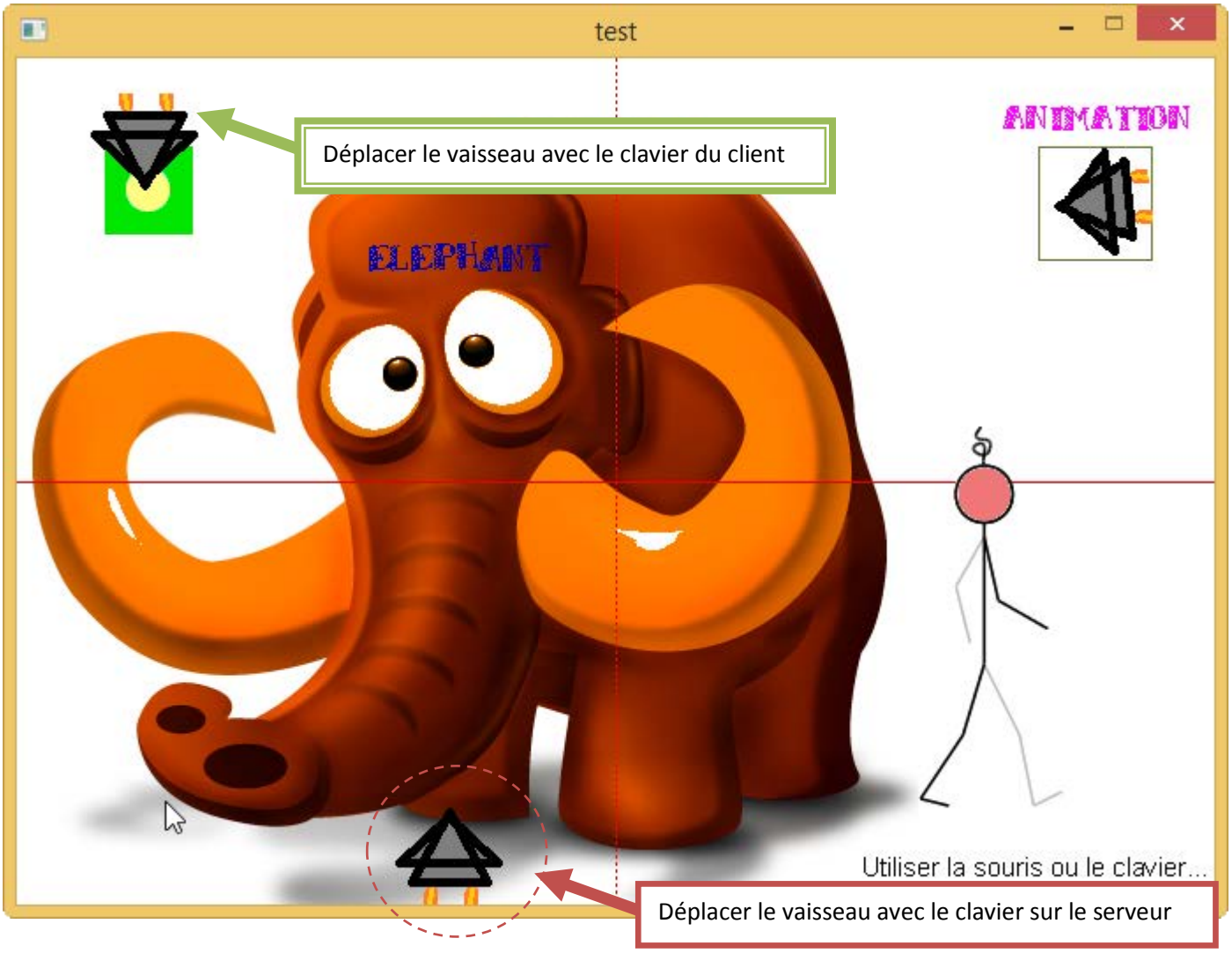


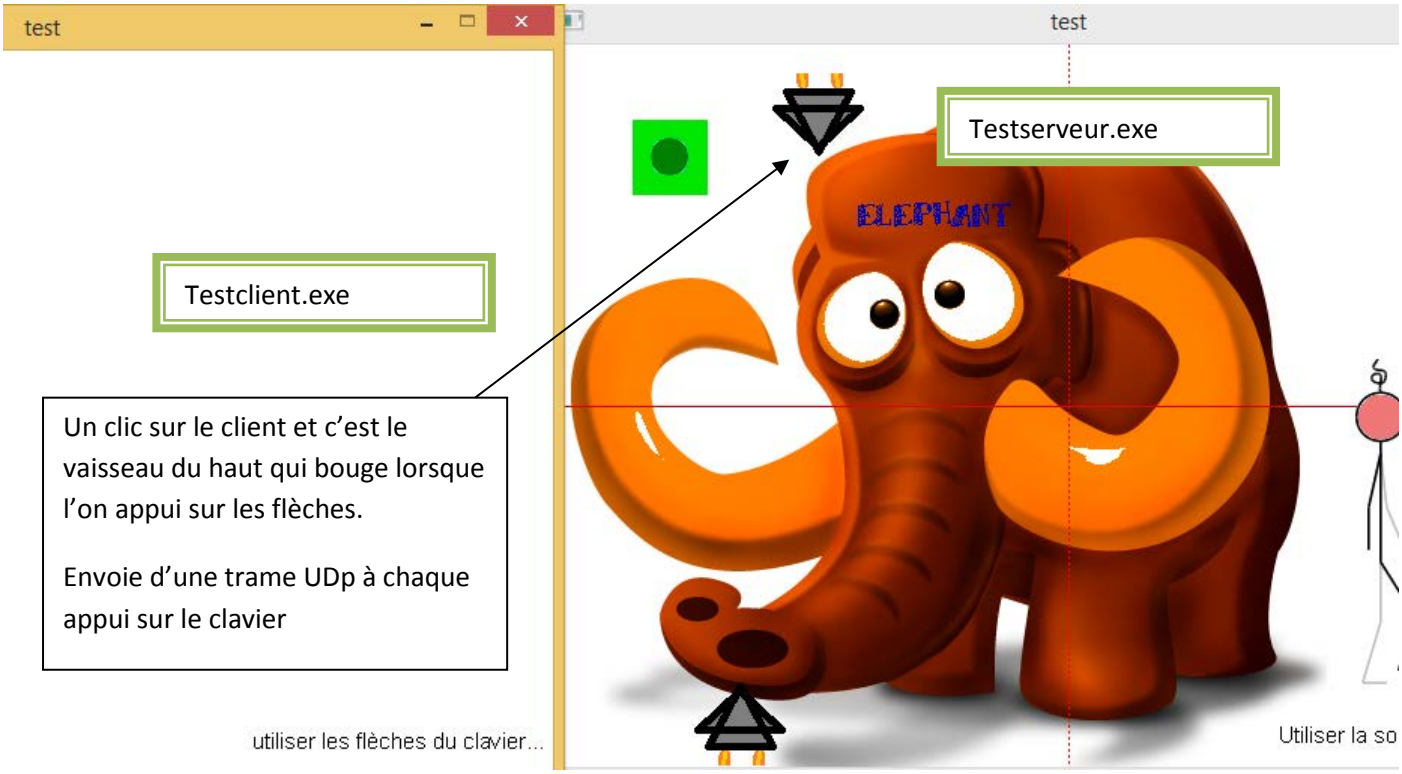




Ci-dessus, on a tapé sur les flèches haut/bas, gauche et droite.

Compiler le serveur et le lancer.


test
ANIMATION
Déplacer le vaisseau avec le clavier du client
ELEPHANT
Utiliser la souris ou le clavier...
Déplacer le vaisseau avec le clavier sur le serveur


test
test
Testserveur.exe
ELEPHANT
Testclient.exe
Un clic sur le client et c'est le vaisseau du haut qui bouge lorsque l'on appui sur les flèches.
Envoie d'une trame UDp à chaque appui sur le clavier
utiliser les flèches du clavier...
Utiliser la so

Quelques explications

Coté serveur graphique :

net_init("127.0.0.1",5001,5000);

Cette ligne de code permet d'ouvrir une connexion UDP (on pourra aller voir le code dans net.c) vers l'adresse 127.0.0.1 c'est-à-dire l'adresse locale. Normal notre serveur se trouve aussi sur notre PC. Mais nous pourrions changer l'adresse de destination vers le pc client… Les paramètres 5001 et 5000 sont les ports de communications sources et destination

Coté client :

net_init("127.0.0.1",5000,5001);

Coté client, on retrouve la même adresse ip de destination, pour l'instant le serveur est en local et les ports source et destination sont inversés par rapport au serveur (normal).

Coté serveur graphique :

```
int threadReceiver( void* data )
{
    char msg[30];
    printf( "Running thread with value = %d\n", (int)data );
    while( gQuit == false )
    {
        //on renvoie la donnée lue

        if(net_CheckForData(msg)>0)
        {
            puts(msg);
            SDL_Event event;
            SDL_UserEvent userevent;

            /* In this example, our callback pushes an SDL_USEREVENT event
            into the queue, and causes our callback to be called again at the
            same interval: */

            userevent.type = SDL_USEREVENT;
            userevent.code = 1;
            userevent.data1 = msg;
            userevent.data2 = NULL;

            event.type = SDL_USEREVENT;
            event.user = userevent;

            SDL_PushEvent(&event);

        }
        SDL_Delay(50);
    }

}
```

Et dans le main :

```
//Run the thread

    int data = 101;
    SDL_Thread* threadID = SDL_CreateThread( threadReceiver, "Receiver", (void*)data );
```

Coté serveur, un thread est créé. Un thread (`threadReceiver`) est un processus qui s'exécute en parallèle du main. Comme le Timer vu précédemment, à la différence que ce Thread une fois lancée s'exécute tant qu'il est vivant. Dans notre cas, ce Thread vérifie toutes les 50ms si une trame UDP a été reçue et si c'est le cas le message reçu est transféré au main au travers d'un évènement SDL_USEREVENT.

La fonction main pourra alors traitée ce message et déplacer le vaisseau du haut en fonction des ordres reçus.

Projets

Le but ici est de faire un jeu 2D : 3 projets au choix, mais vous pouvez aussi proposer votre projet.

Projet 1 : un seul vaisseau envoie des billes vers un décor à casser. Des billes sont aussi envoyées de façon aléatoire. On pourra ajouter des niveaux, des couleurs de briques qui se cassent en un ou plusieurs coups en fonction de la couleur.

Projet 2 : jeu d'adresse. C'est simple 1 barre par joueur (correspond au palet), le déplacement du joueur 1 se fait par les touches A et Q et pour le deuxième joueur P et M. La balle rebondit sur les bords du jeu et sur le palet du joueur. Le premier qui ne fait pas rebondir la balle sur son palet a perdu (la balle disparait). On pourra créé des niveaux avec augmentaiton de la vitesse de la balle, taille des palet…

Projet 3 : jeu 2 joueurs avec serveur graphique. On pourra proposer une solution pour un jeu en multi-joueur (plus de 2) ;