Chapitre XV : Gestion des événements avancés avec SDL

par Loka

Date de publication : 22/10/2007

Dernière mise à jour : 25/10/2007

XV - Gestion des événements avancés

Dans ce tutoriel, nous allons apprendre à gérer les événements autre que les événements claviers vu dans le chapitre IV : Gestion des événements avec SDL.

Nous allons voir en particulier la gestion des événements suivant :

- Les événements souris.

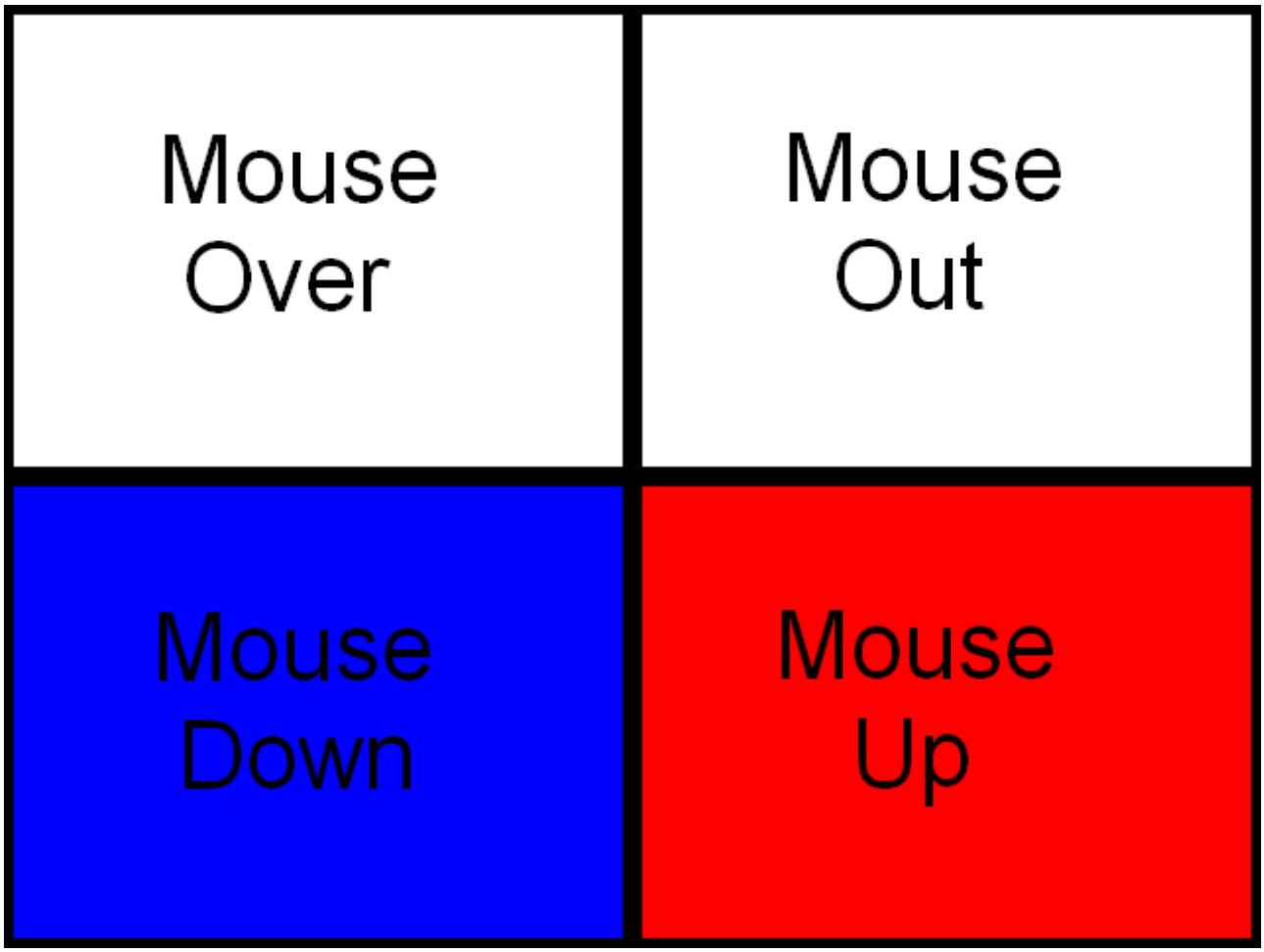
- Les événements joysticks.

- Les événements fenêtre.

XV-1 - Événements souris

Pour apprendre à gérer les événements souris, on va faire un programme d'exemple contenant un simple bouton cliquable.

Donc, pour les besoins de ce tutoriel, on va créer un bouton :



Comme vous pouvez le voir, les différents états de notre bouton sont décomposés dans une feuille de sprite.

Il va donc nous falloir la découper :

set_clips()
```
void set_clips()
{
    //On coupe la partie en haut à gauche (premier sprite)
    clips[ CLIP_MOUSEOVER ].x = 0;
    clips[ CLIP_MOUSEOVER ].y = 0;
    clips[ CLIP_MOUSEOVER ].w = 320;
    clips[ CLIP_MOUSEOVER ].h = 240;

    //On coupe la partie en haut à droite (second sprite)
    clips[ CLIP_MOUSEOUT ].x = 320;
    clips[ CLIP_MOUSEOUT ].y = 0;
    clips[ CLIP_MOUSEOUT ].w = 320;
    clips[ CLIP_MOUSEOUT ].h = 240;

    //On coupe la partie en bas à gauche (troisième sprite)
    clips[ CLIP_MOUSEDOWN ].x = 0;
    clips[ CLIP_MOUSEDOWN ].y = 240;
    clips[ CLIP_MOUSEDOWN ].w = 320;
    clips[ CLIP_MOUSEDOWN ].h = 240;

    //On coupe la partie en bas à droite (quatrième sprite)
    clips[ CLIP_MOUSEUP ].x = 320;
    clips[ CLIP_MOUSEUP ].y = 240;
    clips[ CLIP_MOUSEUP ].w = 320;
    clips[ CLIP_MOUSEUP ].h = 240;
```

set_clips()
```
    }
```

Les variables CLIP_MOUSEOVER, CLIP_MOUSEOUT, CLIP_MOUSEDOWN et CLIP_MOUSEUP sont définies plus haut dans notre code avec pour valeurs respectives 0, 1, 2 et 3.

Si vous ne comprenez pas ce code, je vous renvoi au Chapitre VI : Sprites.

Pour bien travailler sur notre bouton, nous allons créer une classe bouton :

classe bouton
```
   //La classe Button
   class Button
   {
       private:
       //Les attributs du bouton
       SDL_Rect box;

       //La partie de la feuille de sprite qui va être affichée
       SDL_Rect* clip;

       public:
       //Initialisation des variables
       Button( int x, int y, int w, int h );

       //Recuperation des événements et mise en place du bouton
       void handle_events();

       //Affiche le bouton à l'écran
       void show();
   };
```

Dans cette classe, nous avons un rectangle qui définie la position et les dimensions du bouton.

Nous avons aussi un pointeur vers la feuille de sprite qu'on va utiliser.

Ensuite nous avons le constructeur qui nous permet d'initialiser le bouton avec les arguments en paramètre.

Puis nous avons notre fonction de gestion des événements que nous verrons plus en détail par la suite.

Pour finir nous avons la fonction *show()* qui nous permet tout simplement d'afficher le bouton.

Nous allons commencer par voir le constructeur :

constructeur bouton
```
   Button::Button( int x, int y, int w, int h )
   {
       //Initialisation des variables du bouton
       box.x = x;
       box.y = y;
       box.w = w;
       box.h = h;

       //Initialisation du sprite par defaut pour le bouton
       clip = &clips[ CLIP_MOUSEOUT ];
   }
```

Le constructeur est assez simple.

Il initialise les coordonnées x et y du bouton ainsi que sa hauteur et sa largeur (h et w).

Il initialise en plus le sprite par défaut pour le bouton venant de la feuille de sprite qu'on a découpé précédemment.

Nous allons passer à une partie importante : la récupération des événements sur le bouton.

Comme nous nous occupons de récupérer les événements souris sur ce bouton, il nous faut dans un premier temps vérifier si la souris bouge ou non.

handle_events() : mouvement souris

```
void Button::handle_events()
{
    //Les coordonnees de la souris
    int x = 0, y = 0;

    //Si la souris a bougee
    if( event.type == SDL_MOUSEMOTION )
    {
```

Pour vérifier si la souris bouge, il existe un événement SDL approprié : **SDL_MOUSEMOTION**.

Une autre chose intéressante à récupérer de la souris, ce sont ses coordonnées. On va initialiser les variables qui nous serviront pour les coordonnées à 0 comme fait sur le code ci-dessus.

Encore une fois, SDL nous simplifie la tache car une fois qu'on a capturé l'événement de mouvement de la souris, il est très simple de récupérer les coordonnées de la souris en mouvement, ce qui est fait dans cette partie du code :

handle_events() : récupération coordonnées souris

```
        //Recuperation des coordonnees de la souris
        x = event.motion.x;
        y = event.motion.y;
```

Maintenant que nous savons quand la souris bouge et comment récupérer ses coordonnées, nous pouvons par exemple vérifier si la souris se trouve ou non dans la zone de notre bouton, chose intéressante pour, par exemple, faire un menu de jeu.

Pour faire cette partie, il suffit de vérifier si les coordonnées de la souris se trouvent bien dans le bouton ou non.

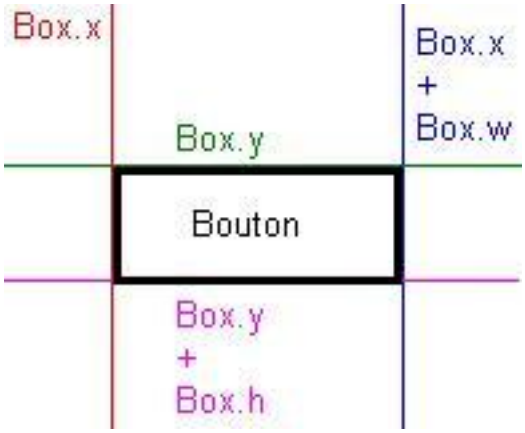
handle_events() : tests souris - bouton

```
        //Si la souris est dans le bouton
        if( ( x > box.x ) && ( x < box.x + box.w ) && ( y > box.y ) && ( y < box.y + box.h ) )
        {
            //Mise à jour du sprite du bouton
            clip = &clips[ CLIP_MOUSEOVER ];
        }
        //Si non
        else
        {
            //Mise à jour du sprite du bouton
            clip = &clips[ CLIP_MOUSEOUT ];
        }
    }
```

x et y sont les coordonnées de la souris, on vérifie que box.x < x < box.x + box.w et que box.y < y < box.y + box.h.

box étant notre bouton bien entendu.

Pour mieux comprendre les tests effectués, je vous invite à regarder le schéma ci-dessous :



Voila, nous sommes capable de savoir quand la souris est dans le bouton et quand elle n'y est pas.

Pour bien montrer dans notre programme que nous savons le faire, on va changer l'affichage du bouton en changeant de sprite comme c'est écrit dans le code ci-dessus.

Une autre chose intéressante lorsqu'on veut faire un menu de jeu par exemple, est de savoir quand est-ce qu'on clique sur un bouton du menu.

Cette partie est faites dans le code ci-dessous :

handle_events() : test clique souris - bouton

```
        //Si un bouton de la souris est pressee
        if( event.type == SDL_MOUSEBUTTONDOWN )
        {
            //Si le bouton gauche de la souris est pressee
            if( event.button.button == SDL_BUTTON_LEFT )
            {
                //Recuperation des coordonnees de la souris
                x = event.button.x;
                y = event.button.y;

                //Si la souris est dans le bouton
                if( ( x > box.x ) && ( x < box.x + box.w ) && ( y > box.y ) && ( y < box.y + box.h ) )
                {
                    //Mise à jour du sprite du bouton
                    clip = &clips[ CLIP_MOUSEDOWN ];
                }
            }
        }
        //Si un bouton de la souris est relache
        if( event.type == SDL_MOUSEBUTTONUP )
        {
            //si c'est le bouton gauche de la souris qui est relache
            if( event.button.button == SDL_BUTTON_LEFT )
            {
                //Recuperation des coordonnees de la souris
                x = event.button.x;
                y = event.button.y;

                //Si la souris est dans le bouton
                if( ( x > box.x ) && ( x < box.x + box.w ) && ( y > box.y ) && ( y < box.y + box.h ) )
                {
                    //Mise à jour du sprite du bouton
                    clip = &clips[ CLIP_MOUSEUP ];
```

handle_events() : test clique souris - bouton
```
                }
            }
        }
    }
```

Pour savoir si un bouton de la souris est appuyé, SDL nous aide grâce à **SDL_MOUSEBUTTONDOWN**.

De même pour savoir si un bouton de la souris est relâche avec **SDL_MOUSEBUTTONUP**.

Une fois qu'on sait si un bouton est appuyé, on peut aussi vérifier lequel est appuyé. Dans notre cas, on va vérifier le clique gauche de la souris et donc vérifier si l'événement clique de la souris est bien **SDL_BUTTON_LEFT**.

Pour savoir si on a bien fais un clique gauche sur le bouton, on vérifie donc si on clique sur un bouton de la souris, si c'est le clique gauche de la souris qui a été pressé et si on se trouve bien dans le bouton.

De même qu'avant, on va changer le sprite du bouton pour montrer qu'on a cliqué dessus.

Le traitement est assez similaire pour le relâchement du bouton gauche de la souris.

Voila, vous êtes maintenant capable de capter les événements de votre souris.

On aurait pu traiter cette partie de code plus efficacement ou plus intelligemment, par exemple en utilisant **SDL_GetMouseState()** et en ne refaisant pas les tests pour savoir si la souris est bien dans le bouton ou non lors du clique (avec par exemple l'enregistrement des différents états par lequel passe le bouton).

Je vous laisse vous débrouiller pour améliorer ce code, vous devriez en être capable.

Il existe bien évidemment d'autres événements souris, comme la gestion de la molette de la souris avec les événements **SDL_BUTTON_WHEELDOWN** et **SDL_BUTTON_WHEELUP**.

Attention, car ces deux événements se récupèrent avec l'événement **SDL_MOUSEBUTTONDOWN** comme le suggère le code suivant :

molette souris
```
    if( event.type == SDL_MOUSEBUTTONDOWN ){
            if( event.button.button == SDL_BUTTON_WHEELDOWN ){
                    ...
            }
    }
```

On pourrait envisager par exemple de gérer un zoom/dezoom avec la molette de la souris en utilisant la librairie SDL_gfx.

Il existe aussi de nombreuses fonctions sympathiques comme la fonction **SDL_ShowCursor(bool show)** qui

permet de masquer ou non le curseur de la souris (ce qui peut être pratique).

Télécharger les sources du chapitre XV-1 (110 ko)

XV-2 - Événements joysticks

Dans cette partie, nous allons voir comment gérer les événements joystick.

Pour illustrer la capture et l'utilisation des événements joystick, nous allons revenir sur le chapitre X et utiliser le joystick pour bouger le point au lieu du clavier.

Gérer les événements joystick est légèrement différent de la gestion des autres événements.

Tout d'abord, les joysticks ont leur propres types de données dans SDL qui est **SDL_Joystick**. Il nous faut donc déclarer notre joystick au début de notre programme :

déclaration du joystick
```
//Le joystick qu'on va utiliser
        SDL_Joystick *stick = NULL;
```

La classe Point ne va pas changer, elle reste exactement la même, la seule chose qui change c'est la façon dont on va récupérer les événements et donc le contenu de la fonction **handle_input()**.

Cependant, avant de plonger dans le vif du sujet, il nous faut apprendre à mettre en place correctement notre joystick à l'initialisation :

init()
```
bool init()
{
    //Initialisation de tous les sous-système de SDL
    if( SDL_Init( SDL_INIT_EVERYTHING ) == -1 )
    {
        return false;
    }

    //Mise en place de l'écran
    screen = SDL_SetVideoMode( SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_BPP, SDL_SWSURFACE );

    //S'il y a eu une erreur lors de la mise en place de l'écran
    if( screen == NULL )
    {
        return false;
    }

    //Verification s'il y a un Joystick
    if( SDL_NumJoysticks() < 1 )
    {
        std::cout << "Erreur : pas de joystick branché" << std::endl;
        return false;
    }

    //Ouverture du Joystick
    stick = SDL_JoystickOpen( 0 );

    //S'il y a un problème à l'ouverture du Joystick
    if( stick == NULL )
```

init()
```
        {
            return false;
        }

        //Mise en place de la barre caption
        SDL_WM_SetCaption( "bouge le Point", NULL );

        //Si tout s'est bien passé
        return true;
    }
```

En effet, ce qui change entre récupérer des événements clavier par exemple et récupérer des événements joystick, c'est qu'il nous faut initialiser le joystick.

Dans un premier temps, il nous faut vérifier s'il y a bien au moins un joystick de branché grâce à la fonction **SDL_NumJoysticks()**.

Si effectivement il y a bien au moins un joystick de branché, nous ouvrons le premier disponible en utilisant **SDL_JoystickOpen()**.

Le premier joystick disponible est le joystick 0.

S'il y a un problème pour l'ouverture du joystick, notre variable **stick** retourne **NULL**.

De même que l'initialisation change avec l'ajout du joystick, notre fonction de nettoyage change dans le même sens :

clean_up()
```
    void clean_up()
    {
        //Libèration des surfaces
        SDL_FreeSurface( point );

        //Fermeture du Joystick
        SDL_JoystickClose( stick );

        //On quitte SDL
        SDL_Quit();
    }
```

Il nous faut fermer le joystick qu'on a ouvert grâce à la fonction **SDL_JoystickClose()**.

Maintenant que nous savons comment bien initialiser notre joystick, nous allons entrer dans le vif du sujet avec la fonction **handle_input()** :

handle_input() : partie 1
```
    void Point::handle_input()
    {
        //Si un axe a changé
        if( event.type == SDL_JOYAXISMOTION )
        {
            //Si le joystick 0 a bougé
            if( event.jaxis.which == 0 )
            {
                //Si l'axe X a changé
                if( event.jaxis.axis == 0 )
                {
```

handle_input() : partie 1
```
                    //Si l'axe X est neutre
                    if( ( event.jaxis.value > -8000 ) && ( event.jaxis.value < 8000 ) )
                    {
                        xVel = 0;
                    }
                    //Si non
                    else
                    {
                        //Ajustement de la vitesse
                        if( event.jaxis.value < 0 )
                        {
                            xVel = -POINT_WIDTH / 2;
                        }
                        else
                        {
                            xVel = POINT_WIDTH / 2;
                        }
                    }
                }
```

Lorsque le joystick bouge, un événement **SDL_JOYAXISMOTION** apparaît.

Nous vérifions dans un premier temps que le joystick qui a bougé est bien le joystick 0. Vous allez sans doute trouver ça pointilleux sachant que le seul joystick initialisé ici est le 0, mais c'est une bonne habitude à prendre de vérifier quoi qu'il arrive.

Ensuite, nous vérifions sur quel axe il a bougé. Sur la plupart des joysticks moderne, l'axe X est 0 et l'axe Y est 1.

Enfin, nous vérifions si la valeur de l'axe X du joystick est entre -8000 et +8000, dans ce cas, nous considérons qu'il est neutre et que donc le point ne doit pas bouger.

Vous devez penser que c'est une plage très large pour le considérer comme neutre, il faut cependant savoir que la valeur de l'axe du joystick se trouve entre les valeurs -32768 et +32768.

Essayez avec des valeurs plus petites pour vous faire une idée.

Si le joystick n'est pas considéré comme neutre, alors on change la vitesse du point.

Il en est de même pour l'axe Y :

handle_input() : partie 2
```
                //Si l'axe Y a changé
                else if( event.jaxis.axis == 1 )
                {
                    //Si l'axe Y est neutre
                    if( ( event.jaxis.value > -8000 ) && ( event.jaxis.value < 8000 ) )
                    {
                        yVel = 0;
                    }
                    //Si non
                    else
                    {
                        //Ajustement de la vitesse
                        if( event.jaxis.value < 0 )
```

handle_input() : partie 2
```
                        {
                            yVel = -POINT_HEIGHT / 2;
                        }
                        else
                        {
                            yVel = POINT_HEIGHT / 2;
                        }
                    }
                }
            }
        }
    }
```

Vous voila prêt à gérer les événements joystick, le reste ne changeant pas.

Télécharger les sources du chapitre XV-2 (195 ko)

Gérer les événements **SDL_JoyAxisEvent** est ce qu'il y a de plus difficile à gérer lorsqu'on gère les événements d'un joystick.

Récupérer les autres événements comme **SDL_JoyBallEvent**, **SDL_JoyHatEvent** ou encore **SDL_JoyButtonEvent** est assez facile à faire en regardant rapidement la documentation de SDL.

Je vais cependant en parler rapidement.

Pour ce qui est des boutons, après avoir vérifié que l'événement **SDL_JoyButtonEvent** est récupéré, on peut utiliser les événements **SDL_JOYBUTTONDOWN** ou **SDL_JOYBUTTONUP** afin de savoir si le bouton est pressé ou relâché.

Ensuite pour savoir quel bouton est pressé (ou relâche), on utilise *event.jbutton.button* comme ceci :

bouton 0 pressé
```
    // Boutons du Joystick
    case SDL_JOYBUTTONDOWN:
        if ( event.jbutton.button == 0 )
        {
            // Code pour le bouton 0
        }
        break;
```

Pour ce qui est des chapeaux, après avoir vérifié que l'événement **SDL_JoyHatEvent** est récupéré, on peut connaître la direction en utilisant les événements suivant :

événements direction chapeau
```
    SDL_HAT_CENTERED
    SDL_HAT_UP
    SDL_HAT_RIGHT
    SDL_HAT_DOWN
    SDL_HAT_LEFT

    SDL_HAT_RIGHTUP
    SDL_HAT_RIGHTDOWN
    SDL_HAT_LEFTUP
    SDL_HAT_LEFTDOWN
```

En effet, les chapeaux ne peuvent être dirigés que dans une seule direction à la fois et pour savoir si le chapeau a bougé, on utilise l'événement **SDL_JOYHATMOTION** :

coordonnées de depart du fond
```
   // Mouvement du chapeau
   case SDL_JOYHATMOTION:
       if ( event.jhat.hat & SDL_HAT_UP )
       {
           // Vers le haut
       }
       else if ( event.jhat.hat & SDL_HAT_RIGHTDOWN )
       {
           // Vers la diagonale droit-bas
       }
       break;
```

Pour le mouvement de la boule, après avoir vérifié que l'événement **SDL_JoyBallEvent** est récupéré, on utilise **SDL_JOYBALLMOTION**. La récupération du mouvement en x et y se fait à l'aide de *event.jball.xrel* et *event.jball.yrel*.

Ces derniers nous donnent une position relative et non absolue contrairement à la souris.

SDL_JOYBALLMOTION : tiré de la doc SDL
```
   case SDL_JOYBALLMOTION:
       printf ("Trackball axis %i ",event.jball.ball);
       printf ("on joystick %i ", event.jball.which);
       printf ("moved to (%i,%i)\n", event.jball.xrel, event.jball.yrel);
       break;
```

La librairie SDL nous permet aussi de connaître les caractéristiques du matériel, surtout pour les joysticks, grâce aux fonctions suivantes : **SDL_JoystickNumAxes** , **SDL_JoystickNumButtons**, **SDL_JoystickNumBalls**, **SDL_JoystickNumHats**... qui retournent des int et prennent l'élément **SDL_Joystick** comme argument.

XV-3 - Événements fenêtre

Jusqu'à présent, nos fenêtres SDL étaient de taille fixe, n'étaient pas redimensionnables et n'étaient pas forcement bien adaptés à l'écran.

Le but de cette partie du tutoriel est de pouvoir capturer les événements fenêtre afin de gérer le redimensionnement, le passage en mode plein écran, et d'autres choses que vous pourriez trouver utiles dans le futur.

Commençons par construire une classe fenêtre (Window en anglais) :

classe Window
```
   // Notre classe fenetre
   class Window
   {
       private:
       //variable qui indique si on est en mode fenétré ou non
       bool windowed;

       //variable qui dit si notre fenêtre est ok
       bool windowOK;

       public:
```

classe Window
```
        //Constructeur
        Window();

        //Recuperation des événements de la fenêtre
        void handle_events();

        //plein écran ou non
        void toggle_fullscreen();

        //Verifie si quelque chose ne va pas avec la fenetre
        bool error();
    };
```

Nous avons tout d'abord deux variables booléennes.

La première va nous permettre de savoir si la fenêtre est en mode fenêtré ou en mode plein écran.

La seconde nous indique si la fenêtre est opérationnelle tout simplement.

Ensuite nous avons un constructeur, un récupérateur d'événements sur notre fenêtre, une fonction pour le passage en mode plein écran ou en mode fenêtré et pour finir une fonction de vérification d'erreurs.

Nous allons revoir notre fonction d'initialisation car nous allons devoir mettre tout ce qui concerne la fenêtre dans le constructeur de la fenêtre.

fonction init()
```
    bool init()
    {
        //Initialisation de tout les sous-systemes
        if( SDL_Init( SDL_INIT_EVERYTHING ) == -1 )
        {
            return false;
        }

        //si tout s'est bien passé
        return true;
    }
```

Ainsi, maintenant dans la fonction *init()*, on ne s'occupe que d'initialiser SDL.

Toute la mise en place de la fenêtre se passe dans le constructeur que voici :

constructeur de la classe Window
```
    Window::Window()
    {
        //Mise en place de l'écran
        screen = SDL_SetVideoMode( SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_BPP, SDL_SWSURFACE |
    SDL_RESIZABLE );

        //S'il y a une erreur
        if( screen == NULL )
        {
            windowOK = false;
            return;
        }
        else
        {
            windowOK = true;
        }
```

constructeur de la classe Window

```
        //on met en place la barre caption de la fenetre
        SDL_WM_SetCaption( "Test Evenements Fenetre", NULL );

        windowed = true;
    }
```

Dans cette classe, nous créons une fenêtre qui peut être redimensionnable.

Ainsi, vous pouvez voir apparaître le **SDL_RESIZABLE** en paramètre à la fonction **SDL_SetVideoMode()**.

Ensuite vient le test habituel afin de savoir si la fenêtre a bien été créée ou non, cependant on met à jour la variable **windowOK** qui va nous dire qu'il y a un problème.

Enfin, si tout se passe bien, nous mettons cette variable **windowOK** à **true**, ainsi que la variable **windowed** qui nous indique qu'on démarre en mode fenêtré.

Le changement de mode va se faire avec la fonction **toggle_fullscreen()** :

Fonction toggle_fullscreen()

```
    void Window::toggle_fullscreen()
    {
        //Si on est en mode fenetré
        if( windowed == true )
        {
            //On met en plein écran
            screen = SDL_SetVideoMode( SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_BPP, SDL_SWSURFACE |
    SDL_RESIZABLE | SDL_FULLSCREEN );

            //S'il y a une erreur
            if( screen == NULL )
            {
                windowOK = false;
                return;
            }

            //Mise à jour de la variable
            windowed = false;
        }
        //Si on est en plein écran
        else if( windowed == false )
        {
            //On passe en mode fenetré
            screen = SDL_SetVideoMode( SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_BPP, SDL_SWSURFACE |
    SDL_RESIZABLE );

            //S'il y a une erreur
            if( screen == NULL )
            {
                windowOK = false;
                return;
            }

            //Mise à jour de la variable
            windowed = true;
        }
    }
```

La possibilité de passer en mode plein écran ou en mode fenêtré n'est possible seulement grâce au fait qu'on est avec une fenêtre redimensionnable.

Pour le passage en mode plein écran ou en mode fenêtré, on vérifie dans un premier temps dans quel mode nous nous trouvons grâce à la variable **windowed** qui est tenu à jours.

Si on est en mode fenêtré, alors on passe en mode plein écran en appelant la fonction **SDL_SetVideoMode()** avec comme paramètre supplémentaire **SDL_FULLSCREEN**.

Si on est déjà en mode plein écran, on revient en mode fenêtré en faisant appel à la même fonction **SDL_SetVideoMode()** sans le paramètre **SDL_FULLSCREEN**.

Le passage d'un mode à l'autre va se faire en capturant un événement clavier (la pression de la touche entrée) comme nous allons le voir dans la fonction de récupération des événements de la fenêtre.

handle_events : redimmension

```
void Window::handle_events()
{
    //Si quelque chose ne va pas avec notre fenetre
    if( windowOK == false )
    {
        return;
    }

    //Si la fenetre est redimmensionnee
    if( event.type == SDL_VIDEORESIZE )
    {
        //Redimmensionnement de la fenetre
        screen = SDL_SetVideoMode( event.resize.w, event.resize.h, SCREEN_BPP, SDL_SWSURFACE |
SDL_RESIZABLE );

        //S'il y a une erreur
        if( screen == NULL )
        {
            windowOK = false;
            return;
        }
    }
```

Voici le début de notre fonction **handle_events()**.

Dans un premier temps, on vérifie si notre fenêtre est bien opérationnelle.

Nous allons ensuite attraper et gérer la redimension de la fenêtre.

L'événement SDL correspondant au redimensionnement de la fenêtre est **SDL_VIDEORESIZE**. Si la fenêtre est redimensionnée, on récupère les nouvelles dimensions grâce à **event.resize.w** pour obtenir la nouvelle largeur de la fenêtre et **event.resize.h** pour la hauteur.

Comme vous pouvez le voir, quand nous redimensionnons la fenêtre, on fait de nouveau appel à **SDL_SetVideoMode()**, il nous faut donc de nouveau vérifier si notre fenêtre est bien construite.

Dans la suite de notre fonction, on va faire le passage en mode plein écran ou en mode fenêtré en récupérant un événement clavier (la pression de la touche entrée) :

handle_events : toggle_fullscreen()

```
    //Si la touche entrée a été préssée
```

handle_events : toggle_fullscreen()
```
        else if( ( event.type == SDL_KEYDOWN ) && ( event.key.keysym.sym == SDLK_RETURN ) )
        {
            //mode plein écran on/off
            toggle_fullscreen();
        }
```

Nous avons déjà vu les événements clavier, donc si quelque chose pose problème, vous pouvez toujours revenir au chapitre IV : gestion des événements avec SDL.

Un autre type d'événement fenêtre fourni par SDL est **SDL_ACTIVEEVENT**.

Cet événement va nous indiquer s'il y a eu un changement sur le focus. Cet événement arrive lorsque l'écran, la souris ou le clavier deviennent actif ou inactif.

handle_events : changement de focus sur la fenêtre
```
        //Si le focus de la fenetre a changé
        else if( event.type == SDL_ACTIVEEVENT )
        {
            //Si la fenetre a été réduite ou remise en place
            if( event.active.state & SDL_APPACTIVE )
            {
                //Si l'application n'est plus active
                if( event.active.gain == 0 )
                {
                    SDL_WM_SetCaption( "Test Evenements Fenetre : Reduite", NULL );
                }
                else
                {
                    SDL_WM_SetCaption( "Test Evenements Fenetre", NULL );
                }
            }
```

Dans un premier temps, on va regarder comment gérer le changement de focus sur la fenêtre.

On va vérifier si la fenêtre devient active ou inactive en passant du mode réduit ou non. L'événement SDL qui nous indique le changement de focus sur la fenêtre est **SDL_APPACTIVE**.

Lorsque la fenêtre est réduite, **gain** dans la structure **SDL_ActiveEvent** est à 0. Dans le cas contraire (où la fenêtre n'est pas réduite), il est à 1.

Dans ce programme, nous changeons la barre de caption afin de notifier à l'utilisateur le changement.

Toujours dans cette partie, nous allons vérifier s'il y a eu un changement de focus au niveau du clavier ou de la souris :

handle_events : changement de focus sur le clavier ou la souris
```
            //Si quelque chose se passe avec le focus du clavier
            else if( event.active.state & SDL_APPINPUTFOCUS )
            {
                //Si l'application perd le focus du clavier
                if( event.active.gain == 0 )
                {
                    SDL_WM_SetCaption( "Test Evenements Fenetre : Focus du clavier perdu", NULL );
                }
                else
                {
                    SDL_WM_SetCaption( "Test Evenements Fenetre", NULL );
```

handle_events : changement de focus sur le clavier ou la souris
```
            }
        }
        //Si quelque chose se passe avec le focus de la souris
        else if( event.active.state & SDL_APPMOUSEFOCUS )
        {
            //Si l'application perd le focus de la souris
            if( event.active.gain == 0 )
            {
                SDL_WM_SetCaption( "Test Evenements Fenetre : Focus de la souris perdu", NULL );
            }
            else
            {
                SDL_WM_SetCaption( "Test Evenements Fenetre", NULL );
            }
        }
    }
```

SDL_APPINPUTFOCUS nous indique si il y a eu un changement dans le focus pour le clavier.

SDL_APPMOUSEFOCUS fais de même pour la souris.

Dans le cas de la souris, lorsqu'on perd le focus, **gain** est à 0 et il passe à 1 lorsque celui-ci est restauré.

Nous allons terminer cette fonction par un autre événement SDL sur la fenêtre : **SDL_VIDEOEXPOSE**

SDL_VIDEOEXPOSE est un événement qui se passe lorsque l'écran est altéré par quelque chose d'extérieur au programme. Nous allons le récupérer et mettre à jour l'écran si ça arrive :

handle_events : altération de l'écran
```
    //Si l'écran de la fenetre a été altéré
    else if( event.type == SDL_VIDEOEXPOSE )
    {
        //Mise à jour de l'ecran
        if( SDL_Flip( screen ) == -1 )
        {
            //S'il y a une erreur
            windowOK = false;
            return;
        }
    }
}
```

Voila pour la fonction de récupération des événements sur notre fenêtre.

Il ne reste plus grand chose à voir sur cette partie, je vous montre tout de même la fonction error() de notre fenêtre :

fonction error()
```
bool Window::error()
{
    return !windowOK;
}
```

Comme vous le voyez, c'est tout ce qu'il y a de plus simple en utilisant notre variable **windowOK**.

La création de la fenêtre se fait dans le *main()*, comme ceci :

main : création de la fenêtre
```
int main( int argc, char* args[] )
{
    //ce qui va nous permettre de quitter
    bool quit = false;

    //Initialisation
    if( init() == false )
    {
        return 1;
    }

    //Creation d'une fenetre
    Window myWindow;

    //Si la creation échoue
    if( myWindow.error() == true )
    {
        return 1;
    }
```

Rien de compliqué là dedans, on créé la fenêtre et on vérifie s'il y a un problème à la création.

La boucle principale de notre programme est à peu près la même que dans les autres tutoriaux, je vous laisse le soin de regarder le code de plus près dans les sources téléchargeables.

Télécharger les sources du chapitre XV-3 (100 ko)

Sources et pdf

Télécharger les sources du chapitre XV-1 (110 ko)

Télécharger les sources du chapitre XV-2 (100 ko)

Télécharger les sources du chapitre XV-3 (195 ko)

Version pdf (134 ko - 21 pages) 

Remerciements

Je remercie RideKick pour sa relecture.