Chapitre XIII : Sauvegarde

par Loka

Date de publication : 10/04/2007

Dernière mise à jour : 10/04/2007

XIII - Sauvegarde

Que vous souhaitiez sauvegarder des paramétres ou progresser dans un jeu, vous aurez besoin de savoir manipuler les entrées/sorties.

Dans ce tutoriel, on va sauvegarder un type de fond d'écran ainsi que la position (les coordonnées) d'un point.

Ce que nous souhaitons ici, c'est que quand nous lançerons le programme, le fond d'écran et la position de notre point sera la même que lorsqu'on aura fermé le programme.

Ce tutoriel va vous apprendre les bases de la sauvegarde et du chargement de données dans un fichier.

XIII-A - Introduction

Pour les besoins de ce tutoriel, nous allons réutiliser notre classe *Point* du chapitre X : Mouvements.

Nous allons ajouter deux méthodes très classiques à cette classe : des accesseurs *get_x* et *get_y* pour récuperer les coordonnées x et y du point.

accesseurs
```
	//Recupere la coordonnees x du point
	int &get_x();

	//Recupere la coordonnees y du point
	int &get_y();
```

Il n'y a besoin d'ajouter rien d'autre dans cette classe, elle est maintenant fonctionnelle pour les sauvegardes.

XIII-A-1 - Entrées/Sorties

Comme dis précédemment, nous allons travailler avec les entrées/sorties.

Nous allons donc inclure une bibliothèque d'entrées/sorties : *fstream*

entete
```
   //Les fichiers d'entete
   #include <SDL/SDL.h>
   #include <SDL/SDL_image.h>
   #include <string>
   #include <fstream>
```

fstream va nous permettre de lire/écrire un fichier.

C'est une bibliothèque standard C++ et non une partie de SDL, tout comme *string*, c'est pourquoi je ne décrirais pas plus son utilisation.

XIII-A-2 - Structure du fichier de sauvegarde

Pour sauvegarder et charger des données, nous allons utiliser un fichier qui nous servira de fichier de sauvegarde.

Avant de se lancer dans le code, il nous faut savoir ce qu'on va stocker dans ce fichier.

Ici, ce dont on aura besoin de savoir est la position de notre point, les coordonnées donc qu'on peut maintenant récupérer grace aux accesseurs, ainsi que la couleur du fond d'écran de notre fenêtre SDL.

Pour les coordonnées, on va les mettre sur la première ligne de notre fichier.

Pour la couleur du fond, plutôt que de mettre les composante rgb (ce qu'on aurait très bien pu faire), on va plutot mettre la couleur de ce qu'on va appeler le niveau (white level, green level, etc.).

Au final, notre fichier texte contiendra des données telles que celle-ci :

```
1   290 200
2   Red Level
```

XIII-B - Chargement des données

Dans ce tutoriel, le chargement des données se fera au lancement du programme.

Nous allons donc faire ce chargement au moment du chargement des fichiers et donc dans la méthode *load_files()*, après chargement des images.

Nous allons, pour charger un fichier, créer un objet *ifstream*.

load_file
```
bool load_files( Point &thisPoint, Uint32 &bg )
{
    //Chargement du point
    point = load_image( "dot.png" );

    //S'il y a eu un probleme au chargement du point
    if( point == NULL )
    {
        return false;
    }

    //Ouverture d'un fichier a lire
    std::ifstream load( "game_save" );
```

ifstream vous permet de récupérer ce qui est dans un fichier. Lorsque vous passez un nom de fichier au constructeur, il l'ouvre pour le lire.

Comme vous pouvez le voir, on passe en paramètre notre point *thisPoint* et la couleur du fond *bg* car on va en avoir besoin dans cette fonction.

Le code qui suit va nous permettre de récupérer les coordonnées de notre point dans le fichier :

load_file
```
    //Si le fichier est charge
    if( load != NULL )
    {
```

load_file
```
            //Le nom du niveau
            std::string level;

            //Recuperation des coordonnees
            load >> thisPoint.get_x();
            load >> thisPoint.get_y();

            //Si la coordonnee x est invalide
            if( ( thisPoint.get_x() < 0 ) || ( thisPoint.get_x() > SCREEN_WIDTH - POINT_WIDTH ) )
            {
                return false;
            }

            //Si la coordonnee y est invalide
            if( ( thisPoint.get_y() < 0 ) || ( thisPoint.get_y() > SCREEN_HEIGHT - POINT_HEIGHT ) )
            {
                return false;
            }
```

Lorsqu'il y a un problème lors du chargement du fichier, l'objet *ifstream* est *NULL*, c'est pourquoi nous testons la valeur de retour de celui-ci.

Si le fichier est bien chargé, nous déclarons notre niveau (*level*) qui va nous servir pour mettre le fond d'écran de notre fenêtre SDL à la bonne couleur.

Ensuite, nous récupérons les deux entiers correspondants aux coordonnées x et y de notre point depuis le fichier de sauvegarde.

Comme vous pouvez le voir, nous récupérons ces entiers avec *ifstream* comme nous le ferions en utilisant *cin*.

C'est parce que ce sont tous les 2 des *istream* (input stream).

Nous vérifions aussi si les coordonnées qu'on récupère ont un sens, si elles sont valides.

En effet, l'utilisateur peut facilement altérer le fichier de sauvegarde.

Maintenant qu'on a récupéré les coordonnées de notre point, on va récupérer la couleur du fond d'écran de notre fenêtre SDL.

load_file
```
            //On passe le caractere suivant ("\n")
            load.ignore();

            //On recupere la ligne suivante
            getline( load, level );

            //Si une erreur se produit pendant la recuperation des donnees
            if( load.fail() == true )
            {
                return false;
            }
```

Nous passons à la ligne suivante contenant la couleur du niveau avec la fonction *ignore()*, qui nous permet de passer le caractère suivant qui est ici un '\n'.

Nous récupérons donc la ligne suivante avec la fonction *getline()* cette fois ci.

La fonction *getline()* est différente de l'utilisation de '>>' comme fait précédemment pour la récupération des coordonnées. Cette fonction permet de récupérer tout jusqu'à la fin de la ligne.

Ensuite nous vérifions juste si il n'y a pas de problèmes durant la lecture du fichier.

La fonction *fail()* retourne *true* si il y a un problème.

Il nous reste maintenant à affecter la bonne couleur à notre variable *bg* représentant la couleur du fond d'écran de notre fenêtre SDL :

load_file
```
            //Si le niveau etait blanc
            if( level == "White Level" )
            {
                //On met le fond en blanc
                bg = SDL_MapRGB( screen->format, 0xFF, 0xFF, 0xFF );
            }
            //Si le niveau etait rouge
            else if( level == "Red Level" )
            {
                //On met le fond en rouge
                bg = SDL_MapRGB( screen->format, 0xFF, 0x00, 0x00 );
            }
            //Si le niveau etait vert
            else if( level == "Green Level" )
            {
                //On met le fond en vert
                bg = SDL_MapRGB( screen->format, 0x00, 0xFF, 0x00 );
            }
            //Si le niveau etait bleu
            else if( level == "Blue Level" )
            {
                //On met le fond en bleu
                bg = SDL_MapRGB( screen->format, 0x00, 0x00, 0xFF );
            }
                else{
                        //Pas de correspondance
                        return false
                }
```

Maintenant qu'on a récupéré notre niveau, il est facile d'affecter la bonne couleur à notre variable *bg* comme vous pouvez le voir.

On a fini de récupérer ce dont on avait besoin, il nous reste à fermer proprement le fichier :

load_file
```
            //Fermeture du fichier
            load.close();
        }

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

Voilà pour le chargement des données depuis notre fichier de sauvegarde.

Ce système ne sert pas que pour les sauvegardes dans un jeu, il peut aussi servir pour pleins de choses telles que

les fichiers de configuration par exemple.

Maintenant qu'on a fini la partie sur le chargement des données, on va attaquer la partie sur la sauvegarde des données.

XIII-C - Sauvegarde des données

Dans ce tutoriel, la sauvegarde des données se fera à la fermeture du programme, donc automatiquement.

C'est pourquoi nous allons implémenter ce mécanisme dans notre fonction de nettoyage *clean_up* qui est appelé à la fermeture de notre fenêtre SDL.

clean_up
```
   void clean_up( Point &thisPoint, Uint32 &bg )
   {
       //Liberation de la surface
       SDL_FreeSurface( point );

       //Ouverture d'un fichier pour ecrire
       std::ofstream save( "game_save" );

        if (!save) {
           std::cout << "Erreur d'ouverture du fichier de sauvegarde : sauvegarde échouée" <<
   std::endl;
       }
       else {

           //Ecriture des coordonnees dans le fichier
           save << thisPoint.get_x();
           save << " ";
           save << thisPoint.get_y();
           save << "\n";
```

Dans cette fonction, on va créer un objet *ofstream* pour écrire dans le fichier.

Comme vous pouvez le voir, puisque l'utilisation de *ifstream* était similaire à celle de *cin*, il est logique que l'utilisation de *ofstream* soit similaire à celle de *cout*

Si l'ouverture de **save** échoue, la sauvegarde échoue aussi.

Dans cette partie du code, nous écrivons dans le fichier les coordonnées x et y de notre point, récupérées grâce aux accesseurs.

Il nous faut maintenant enregistrer la couleur de fond.

clean_up
```
           //Les valeurs rgb de la couleur du fond
           Uint8 r, g, b;

           //Recuperation des valeurs rgb de la couleur du fond
           SDL_GetRGB( bg, screen->format, &r, &g, &b );
```

Nous récupérons donc les valeurs rgb de notre fond en utilisant la fonction *SDL_GetRGB()*.

Il nous reste à écrire dans le fichier la bonne couleur de niveau :

clean_up
```
            //Si le fond etait blanc
            if( ( r == 0xFF ) && ( g == 0xFF ) && ( b == 0xFF ) )
            {
                //On ecrit le type du niveau dans le fichier (blanc)
                save << "White Level";
            }
            //Si le fond etait rouge
            if( r == 0xFF && ( g != 0xFF ) )
            {
                //On ecrit le type du niveau dans le fichier (rouge)
                save << "Red Level";
            }
            //Si le fond etait vert
            if( g == 0xFF && ( b != 0xFF ) )
            {
                //On ecrit le type du niveau dans le fichier (vert)
                save << "Green Level";
            }
            //Si le fond etait bleu
            if( b == 0xFF && ( g != 0xFF ) )
            {
                //On ecrit le type du niveau dans le fichier (bleu)
                save << "Blue Level";
            }
```

Ce qui se fait très naturellement comme vous pouvez le constater.

Il nous reste à fermer proprement le fichier une fois encore ainsi que de quitter SDL proprement aussi :

clean_up
```
            //Fermeture du fichier
            save.close();
        }

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

Voilà pour la sauvegarde des données dans un fichier.

XIII-D - Main et boucle principale

Il nous reste maintenant à assembler tout ça dans la fonction principale, le main.

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

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

        //Le point qu'on va utiliser
        Point monPoint;

        //La couleur du fond
        Uint32 background = SDL_MapRGB( screen->format, 0xFF, 0xFF, 0xFF );
```

main
```
	//Le regulateur
	Timer fps;

	//Chargement des fichiers
	if( load_files( monPoint, background ) == false )
	{
	    return EXIT_FAILURE;
	}
```

Comme vous pouvez le voir, nous utilisons notre fonction *load_files()*.

Je ne pense pas avoir besoin de vous expliquer ce morceau de code, donc nous allons passer à la boucle principale :

boucle principale
```
	//Tant que l'utilisateur n'a pas quitte
	while( quit == false )
	{
	    //On demarre le timer fps
	    fps.start();

	    //Tant qu'il y a un evenement
	    while( SDL_PollEvent( &event ) )
	    {
	        //On recupere l'evenement pour le point
	        monPoint.handle_input();

	        //Si l'utilisateur a appuye sur une touche
	        if( event.type == SDL_KEYDOWN )
	        {
	            //On change le fond selon la touche pressee
	            switch( event.key.keysym.sym )
	            {
	                case SDLK_1: background = SDL_MapRGB( screen->format, 0xFF, 0xFF, 0xFF ); break;
	                case SDLK_2: background = SDL_MapRGB( screen->format, 0xFF, 0x00, 0x00 ); break;
	                case SDLK_3: background = SDL_MapRGB( screen->format, 0x00, 0xFF, 0x00 ); break;
	                case SDLK_4: background = SDL_MapRGB( screen->format, 0x00, 0x00, 0xFF ); break;
	            }
	        }

	        //Si l'utilisateur a clique sur le X de la fenetre
	        if( event.type == SDL_QUIT )
	        {
	            //On quitte le programme
	            quit = true;
	        }
	    }

	    //On remplit l'ecran de blanc
	    SDL_FillRect( screen, &screen->clip_rect, background );

	    //On affiche le point sur l'ecran
	    monPoint.show();

	    //Mise à jour de l'ecran
	    if( SDL_Flip( screen ) == -1 )
	    {
	        return EXIT_FAILURE;
	    }

	    //Tant que le timer fps n'est pas assez haut
	    while( fps.get_ticks() < 1000 / FRAMES_PER_SECOND )
	    {
	        //On attend...
	    }
	}
```

Voici donc la boucle principale.

Pour changer la couleur du fond d'écran de notre fenêtre SDL, vous avez donc simplement à appuyer sur les

touches 1,2,3 et 4.

Le code reste semblable à ce que vous avez déjà pu voir dans mes tutoriels précédents.

A la fin du main, il nous reste plus qu'à lancer notre fonction de nettoyage et de sauvegarde :

boucle principale
```
        //Nettoyage et sauvegarde
        clean_up( monPoint, background );

        return EXIT_SUCCESS;
    }
```

Ainsi, à la fin du programme SDL, nous sauvegardons nos données en plus de nettoyer ce qu'on a utilisé.

Télécharger les sources du chapitre XIII (192 ko)

Version pdf (83 ko - 11 pages)

Remerciements

Je remercie fearyourself pour ses corrections.