Chapitre XII : Animation

par Loka

Date de publication : 10/04/2007

Dernière mise à jour : 21/04/2007

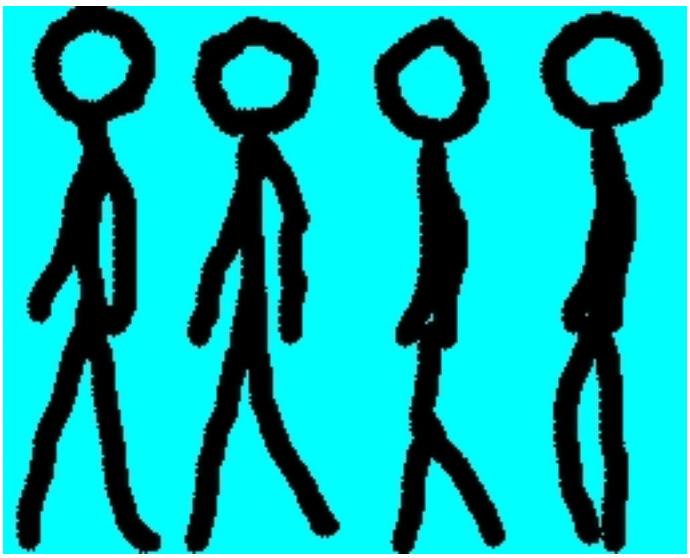
XII - Mouvement

Jusqu'à maintenant, nous avons toujours travaillé juste avec des images fixes.

Ce tutoriel va vous apprendre à mettre en place un petit moteur d'animations basiques.

XII-A - Introduction

Le principe basique d'une animation est de prendre une série d'images qui se suivent comme sur cette feuille de sprites :



Afficher ces images les unes après les autres dans l'ordre créé une illusion de mouvements (si vous ne voyez pas l'animation, cliquez Ici) :

Donc quand vous animez des mouvements en SDL, vous affichez simplement une séquence de SDL_surface.

Pour les besoins de ce tutoriel, nous allons utiliser cette feuille de sprite libre de droits :

Cette feuille de sprite représente la décomposition en 3 phases du mouvement du personnage chat vers la droite et vers la gauche.

Une animation de mouvement peut être décomposée en autant de phases que vous en avez besoin afin d'avoir un mouvement plus naturel ou plus fluide.

Ici nous nous contenterons de 3 phases.

XII-B - Découpage

Nous avons à notre disposition une feuille de sprites contenant les différentes phases du mouvement de notre chat pour l'animation.

Pour l'utiliser correctement, il va nous falloir la découper.

Voici le code de la fonction qui nous permet de découper la feuille de sprites en sprites individuels :

decoupage
```
   void set_clips()
   {
       //On coupe la feuille de sprite
       clipsRight[ 0 ].x = 0;
       clipsRight[ 0 ].y = 0;
       clipsRight[ 0 ].w = CAT_WIDTH;
       clipsRight[ 0 ].h = CAT_HEIGHT;

       clipsRight[ 1 ].x = CAT_WIDTH;
       clipsRight[ 1 ].y = 0;
       clipsRight[ 1 ].w = CAT_WIDTH;
       clipsRight[ 1 ].h = CAT_HEIGHT;
```

decoupage
```
        clipsRight[ 2 ].x = CAT_WIDTH * 2;
        clipsRight[ 2 ].y = 0;
        clipsRight[ 2 ].w = CAT_WIDTH;
        clipsRight[ 2 ].h = CAT_HEIGHT;

        clipsLeft[ 0 ].x = 0;
        clipsLeft[ 0 ].y = CAT_HEIGHT;
        clipsLeft[ 0 ].w = CAT_WIDTH;
        clipsLeft[ 0 ].h = CAT_HEIGHT;

        clipsLeft[ 1 ].x = CAT_WIDTH;
        clipsLeft[ 1 ].y = CAT_HEIGHT;
        clipsLeft[ 1 ].w = CAT_WIDTH;
        clipsLeft[ 1 ].h = CAT_HEIGHT;

        clipsLeft[ 2 ].x = CAT_WIDTH * 2;
        clipsLeft[ 2 ].y = CAT_HEIGHT;
        clipsLeft[ 2 ].w = CAT_WIDTH;
        clipsLeft[ 2 ].h = CAT_HEIGHT;
    }
```

Pour mieux comprendre ce code, je vous invite à revoir ce chapitre :

Chapitre VI : Sprites

Il faut voir qu'on a ici deux sets de sprites, les sprites pour l'animation du mouvement vers la droite et les sprites pour l'animation du mouvement vers la gauche.

Remarquez qu'il y a une façon plus simple de gérer le découpage ici :

decoupage amélioré
```
    void set_clips()
    {
        //On coupe la feuille de sprite
        for(i=0;i<3;i++) {
            clipsRight[ i ].x = CAT_WIDTH * i;
            clipsRight[ i ].y = 0;
            clipsRight[ i ].w = CAT_WIDTH;
            clipsRight[ i ].h = CAT_HEIGHT;

            clipsLeft[ i ].x = CAT_WIDTH * i;
            clipsLeft[ i ].y = CAT_HEIGHT;
            clipsLeft[ i ].w = CAT_WIDTH;
            clipsLeft[ i ].h = CAT_HEIGHT;
    }
```

XII-C - La classe Cat

Notre feuille de sprites contenant un chat, j'ai nommé ma classe Cat mais j'aurais très bien pu la nommer Personnage vu qu'elle peut être utilisée pour n'importe quelle autre feuille de sprites (avec quelques légères modifications).

Voici donc notre classe Cat qui va nous permettre de faire bouger notre chat tout en donnant l'illusion de mouvements :

classe Cat
```
    //Notre personnage
    class Cat
    {
        private:
        //coordonnées
```

classe Cat
```
        int offSet;

        //sa vitesse de deplacement
        int velocity;

        //sa frame courante
        int frame;

        //Son statut d'animation
        int status;

        public:
        //Le constructeur permettant l'initialisation des variables
        Cat();


        void handle_events();

        //montrer le personnage
        void show();
    };
```

Je vais maintenant vous expliquer un peu plus en détail ce code.

Premièrement nous avons les variables *offset* et *velocity* représentant respectivement les coordonnées de notre chat et la vitesse de notre chat.

Vu qu'on va simplement bouger notre chat de droite à gauche, nous avons juste besoin de la composante x des coordonnées.

Ensuite nous avons les variables *frame* et *status*.

frame permet de garder la trace de l'image de l'animation du mouvement à montrer.

status permet de garder la trace du sens du mouvement de notre chat (gauche ou droite).

Ensuite, bien sûr, nous avons le construteur, le gestionnaire d'événements et la fonction qui permet de bouger et d'afficher notre chat.

XII-C-1 - Le constructeur

Voici le code, très simple, du constructeur de notre classe Cat :

constructeur
```
    Cat::Cat()
    {
        //Initialisation des variables de mouvement
        offSet = 0;
        velocity = 0;

        //Initialisation des variables d'animation
        frame = 0;
        status = CAT_RIGHT;
    }
```

Dans le constructeur, on initialise les differentes variables.

On met la coordonnée x à 0, donc le premier sprite de notre chat va se positionner à la position 0 de l'écran.

On met ensuite la vitesse à 0, donc notre chat sera immobile.

Ensuite nous initialisons la variable *frame* à 0 pour signifier qu'on part de l'animation 0.

Pour finir, nous initialisons la variable *status* à *CAT_RIGHT*, ainsi l'animation par defaut est notre chat marchant vers la droite.

XII-C-2 - La méthode handle_events

Voici notre méthode pour récupérer les événements clavier affectant notre chat :

handle_events()
```
void Cat::handle_events()
{
    //Si une touche est préssée
    if( event.type == SDL_KEYDOWN )
    {
        //Mise à jour de la velocité
        switch( event.key.keysym.sym )
        {
            case SDLK_RIGHT: velocity += CAT_WIDTH / 4; break;
            case SDLK_LEFT: velocity -= CAT_WIDTH / 4; break;
            default: break;
        }
    }
    //Si une touche est relachée
    else if( event.type == SDL_KEYUP )
    {
        //Mise à jour de la velocité
        switch( event.key.keysym.sym )
        {
            case SDLK_RIGHT: velocity -= CAT_WIDTH / 4; break;
            case SDLK_LEFT: velocity += CAT_WIDTH / 4; break;
            default: break;
        }
    }
}
```

Pour mieux comprendre ce code, je vous invite à revoir ce chapitre :

Chapitre X : Mouvements

Tout est expliqué dans le chapitre dont le lien se trouve ci-dessus.

Il faut comprendre qu'une vitesse négative va faire déplacer notre chat vers la gauche et qu'une vitesse positive le fera déplacer vers la droite.

XII-C-3 - La méthode show

Voici la fonction qui va nous permettre de déplacer et d'afficher notre chat.

Tout d'abord il faut penser à garder notre chat dans l'écran afin de l'avoir toujours à l'oeil (un chat c'est malicieux...).

show
```
void Cat::show()
{
    //Mouvement
    offSet += velocity;

    //On garde le personnage dans les limites de la fenêtre SDL
    if( ( offSet < 0 ) || ( offSet + CAT_WIDTH > SCREEN_WIDTH ) )
    {
        offSet -= velocity;
    }
```

Le code n'est pas très difficile à comprendre, si on atteint le bord gauche (coordonnée X égal à zero), on annule la vitesse. De même, si on arrive sur le bord droit, on fait de même.

Une autre chose à voir c'est que la coordonnée x du sprite qu'on traite est le bord haut gauche du sprite. C'est pour cela qu'on est obligé d'ajouter la taille du sprite pour le test avec le bord droit et non avec le bord gauche.

Une autre façon de faire serait de mettre simplement la position du personnage à jour (qu'on colle donc aux bords de l'écran) :

show - autre façon de faire
```
void Cat::show()
{
    //Mouvement
    offSet += velocity;

    //On garde le personnage dans les limites de la fenêtre SDL
    if( offSet < 0 )
    {
        offSet = 0;
    }
    if( offSet + CAT_WIDTH > SCREEN_WIDTH )
    {
        offset = SCREEN_WIDTH - CAT_WIDTH;
    }
```

Vous pouvez utiliser les deux façons, elles marchent aussi bien l'une que l'autre bien que la deuxième est plus souvent utilisée

Après avoir bougé le sprite et vérifié la collision avec les bords de l'écran, il est temps de changer de phase du mouvement.

En effet, à chaque déplacement, il va falloir changer le sprite pour simuler le mouvement.

show - suite
```
    //Si Cat bouge à gauche
    if( velocity < 0 )
    {
        //On prend le personnage de profil gauche
        status = CAT_LEFT;

        //On bouge à la prochaine image de l'animation
        frame++;
    }
    //Si Cat bouge à droite
    else if( velocity > 0 )
    {
        //On prend le personnage de profil droit
        status = CAT_RIGHT;

        //On bouge à la prochaine image de l'animation
        frame++;
    }
```

show - suite
```
        //Si Cat ne bouge plus
        else
        {
            //Restart the animation
            frame = 1;
        }
```

La première chose à faire est de vérifier de quel côté notre chat est en train de bouger en testant tout simplement le signe de la vitesse.

Comme dit précédemment, si la vitesse est négative alors notre chat bouge à gauche et si la vitesse est positive alors notre chat bouge à droite.

Ensuite, si notre chat bouge à gauche, il faut mettre la variable *status* à *CAT_LEFT*, ce qui nous servira pour l'affichage plus tard, et incrémenter notre compteur *frame* afin que le sprite suivant soit affiché.

De même, si notre chat bouge à droite, il faut mettre la variable *status* à *CAT_RIGHT* et incrémenter le compteur *frame*.

Si notre chat ne bouge pas, on met la variable *frame* à 1 afin de redémarrer l'animation depuis le sprite où le chat est immobile (image du milieu sur la feuille de sprite selon notre découpage).

Nous aurions très bien pu mettre la variable *frame* à 0 mais notre chat se serait trouvé avec une patte en l'air lorsqu'il est immobile...

Il faut aussi penser à faire boucler l'animation, en effet notre compteur ne doit ici pas depasser 3 (on incremente le compteur avant le test) car il n'y a que 3 phases du mouvement (0, 1 et 2) :

show - suite
```
        //Boucle l'animation
        if( frame >= 3 )
        {
            frame = 0;
        }
```

Donc si on arrive à une valeur du compteur *frame* égale à 3, on le met à 0 afin que l'animation boucle.

Ainsi on aura une suite du genre :

0, 1, 2, 0, 1, 2, ...

Enfin, il nous reste à afficher le bon sprite sur l'écran.

Pour cela, on va utiliser ce code que je vais expliquer par la suite :

show - suite
```
        //Affichage
        if( status == CAT_RIGHT )
        {
```

show - suite
```
            apply_surface( offSet, SCREEN_HEIGHT - CAT_HEIGHT, cat, screen, &clipsRight[ frame ] );
        }
        else if( status == CAT_LEFT )
        {
            apply_surface( offSet, SCREEN_HEIGHT - CAT_HEIGHT, cat, screen, &clipsLeft[ frame ] );
        }
```

Si notre chat bouge vers la droite (*CAT_RIGHT*), on affiche le bon sprite du chat marchant vers la droite.

De même, si notre chat bouge vers la gauche (*CAT_LEFT*), on affiche le bon sprite du chat marchant vers la gauche.

Plus en détail, on va afficher à l'écran *screen* (4ème argument) à la position x *offSet* (1er argument) et à la position y = *SCREEN_HEIGHT* - *CAT_HEIGHT* (2ème argument), n'oublions pas que le point que nous traitons pour les coordonnées du sprite est son bord haut gauche, notre chat *cat* (3ème argument, c'est notre feuille de sprite entière).

Le dernier argument nous permet de déterminer quelle partie découpée de la feuille de sprite on va afficher.

Une meilleure façon de faire qui nous aurait évité de faire ce test aurait été de faire un tableau à deux dimensions avec CAT_RIGHT = 0 et CAT_LEFT = 1.

En ajoutant les variables CAT_DIRECTIONS et CAT_NBR_FRAMES (respectivement égales à 2 et 3 dans notre cas), nous aurions ainsi pu définir un tableau de clips de cette façon :

tableau de clips
```
    SDL_Rect clips[CAT_DIRECTIONS][CAT_NBR_FRAMES];
```

XII-C-4 - La boucle principale

Le gros du travail est fait maintenant, il ne nous reste plus qu'à assembler tout ça.

Comme dans tout programme SDL, avant la boucle principale, on va commencer par initialiser SDL, charger les fichiers puis ici on va avoir besoin de découper notre feuille de sprites ainsi que de "construire" notre chat.

Viens donc ensuite la boucle principale où tout va se passer :

boucle principale
```
        //Tant que l'utilisateur n'a pas quitte
        while( quit == false )
        {
            //Mise en route du timer
            fps.start();

            //Tant qu'il y a un evenement
            while( SDL_PollEvent( &event ) )
            {
                //Recuperation des evenements pour notre "Cat"
                walk.handle_events();

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

boucle principale
```
                quit = true;
            }
        }

        //Remplissage de l'ecran avec du blanc
        SDL_FillRect( screen, &screen->clip_rect, SDL_MapRGB( screen->format, 0xFF, 0xFF, 0xFF ) );

        //Affichage du "Cat" sur l'ecran
        walk.show();

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

        //On rend la main tant qu'on en a pas besoin
        while( fps.get_ticks() < 1000 / FRAMES_PER_SECOND )
        {
            //Attente...
        }
    }
```

C'est à peu près toujours la même chose que sur les tutoriels précédents, notamment celui sur le mouvement (Chapitre X : Mouvements).

Donc comme vous pouvez le voir pour un petit moteur d'animation, tout ce dont on a besoin c'est de savoir quelle animation on va utiliser et quelle image, quelle phase de l'animation, on va afficher.

Je rajouterais que j'ai fait une attente active mais qu'il aurait été plus judicieux d'utiliser SDL_framerate comme on peut le voir dans le tutoriel de fearyourself ici

XII-D - Améliorations

Dans l'état actuel de notre programme, notre variable **FRAMES_PER_SECOND** agit sur le comportement du jeu lui même, ce qu'il ne devrait pas faire car comme son nom l'indique, il s'agit seulement du nombre d'images à afficher par secondes.

En effet, si vous changez la valeur de cette variable dans le programme actuel, vous remarquerez que la vitesse de déplacement de notre chat baissera ou augmentera selon que vous aurez baissé ou augmenté la valeur.

Dans un jeu, ce type de comportement ne doit pas être influencé par le nombre d'images par seconde qu'on affiche.

Pour bien faire, le déplacement de notre chat doit se faire en pixel par unité de temps et non pas en pixel par frame comme c'est le cas ici.

Afin de remedier à ce problème, on va creer une nouvelle variable qu'on va nommer **CAT_VITESSE** qui contiendra la vitesse de déplacement de notre chat en pixel par secondes :

CAT_VITESSE
```
//Vitesse de marche du chat (en pixel par seconde)
const int CAT_VITESSE = 120;
```

Il nous suffit ensuite de changer le code de notre fonction *handle_events()* :

handle_events() amélioré
```
void Cat::handle_events()
{
    //Si une touche est préssée
    if( event.type == SDL_KEYDOWN )
    {
        //mise à jour de la velocité
        switch( event.key.keysym.sym )
        {
            case SDLK_RIGHT: velocity += (CAT_VITESSE / FRAMES_PER_SECOND); break;
            case SDLK_LEFT: velocity -= (CAT_VITESSE / FRAMES_PER_SECOND); break;
            default: break;
        }
    }
    //si une touche est relachée
    else if( event.type == SDL_KEYUP )
    {
        //Mise à jour de la velocité
        switch( event.key.keysym.sym )
        {
            case SDLK_RIGHT: velocity -= (CAT_VITESSE / FRAMES_PER_SECOND); break;
            case SDLK_LEFT: velocity += (CAT_VITESSE / FRAMES_PER_SECOND); break;
            default: break;
        }
    }
}
```

Ainsi la vitesse de notre chat sera maintenant constante et ne dépendra plus du nombre de frames par secondes.

Il subsiste un autre problème, la vitesse d'animation du personnage change aussi en fonction de la valeur de notre variable **FRAMES_PER_SECOND**.

En effet, on voit bien dans le code de la méthode *show()* qu'elle est lié directement au nombre de frames par seconde.

Pour remedier à ce problème, on va ajouter un timer pour régler la vitesse d'animation du personnage et ne plus le lier au nombre de frames par seconde.

Commençons par définir notre vitesse d'animation :

nouvelles variables
```
const int PERSO_DELAI_FRAME = 120; // délais entre deux frames (en ms)
const int PERSO_NB_ANIM = 3; // Nombre de frame d'animation
```

Ensuite, il nous faut donc notre timer, sur notre personnage donc on va ajouter celui-ci dans notre classe **Cat** :

Classe Cat avec timer
```
class Cat
{
    private:
    int offSet;
    Timer anim;
```

La suite de notre classe reste la même.

Attention à bien définir la classe Timer avant la classe Cat, sinon votre compilateur ne va pas aimer.

On va démarrer ce timer lorsqu'on va créer notre chat, donc dans le constructeur :

construteur amélioré
```
Cat::Cat()
{
    //Initialisation des variables de mouvement
    offSet = 0;
    velocity = 0;

    //Initialisation des variables d'animation
    frame = 0;
    status = CAT_RIGHT;

    //Départ du timer
    anim.start();
}
```

Il nous reste à changer l'animation lorsqu'il est temps de le faire, donc lorsque notre timer atteint la valeur qu'on a fixé au début :

show() amélioré
```
void Cat::show()
{
    [...]

    //Si Cat bouge à gauche
    if( velocity < 0 )
    {
        //On prend le personnage de profil gauche
        status = CAT_LEFT;

                //S'il est l'heure, on change l'animation
                if( this->anim.get_ticks() >= CAT_DELAI_FRAME ) {

                        // On remet le timer à 0
                        anim.start();

                        // On passe à la frame suivante
                        frame++;
                }
    }
    //Si Cat bouge à droite
    else if( velocity > 0 )
    {
        //On prend le personnage de profil droit
        status = CAT_RIGHT;

                //S'il est l'heure, on change l'animation
                if( this->anim.get_ticks() >= CAT_DELAI_FRAME ) {

                        // On remet le timer à 0
                        anim.start();

                        // On passe à la frame suivante
                        frame++;
                }
    }
        [...]
```

Le début du code ne change pas, et la fin non plus. Le code change lorsqu'on incrémentait le nombre de frames.

Maintenant, avant d'incrémenter ce nombre, on va vérifier si il est temps de changer l'animation du personnage.

Si c'est le cas, on change l'animation, on redémarre notre timer *anim* et on incrémente le nombre de frame.

Voila, cette fois le comportement de notre programme est conforme au comportement d'un jeu dans la gestion des animations et des mouvements.

Téléchargements

Télécharger les sources du chapitre XII (112 ko)

Télécharger les sources du chapitre XII - version améliorée par FabaCoeur (101 ko)

Version pdf (149 ko - 11 pages) 

Remerciements

Je remercie fearyourself pour ses corrections.

Je remercie FabaCoeur pour avoir soulever un point important pour ce tutoriel et pour avoir fournit la deuxième version des sources.