

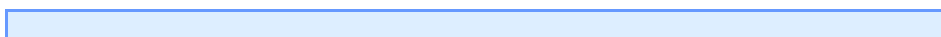
# Chapitre X : Mouvement



par [Loka](#)

Date de publication : 07/11/2006

Dernière mise à jour : 07/11/2006



X - Mouvement

  X-1 - La classe point

  X-2 - Mouvements du point

  X-3 - Affichage du mouvement

Téléchargements

Remerciements

## X - Mouvement

Il est maintenant temps de faire un programme un petit peu plus interactif.

Ce tutoriel a pour but de couvrir les concepts de base du mouvement d'un objet sur un écran.

### X-1 - La classe point

Afin d'illustrer ce tutoriel, j'ai décidé d'utiliser comme objet à déplacer un simple point :



Ce point est un objet comme un autre, j'aurais très bien pu choisir un bonhomme ou une voiture...

Nous allons maintenant construire notre classe point.

Afin d'afficher notre objet point sur l'écran, nous allons avoir besoin de ses coordonnées et d'une méthode pour afficher notre image.

J'ai aussi décidé que le point se déplacera grâce à la pression de certaines touches.

Afin de déplacer notre objet point, il nous faut donc récupérer aussi les événements clavier (pression des touches associés aux déplacements).

Nous avons donc aussi besoin d'une méthode qui nous permet de récupérer ces événements qui affectent notre point.

Voilà donc à quoi va ressembler notre classe point :

#### classe Point

```
class Point
{
    private:
        //Les coordonnées x et y du point
        int x, y;

        //La vitesse du point
        int xVel, yVel;

    public:
        //Initialisation des variables
        Point();

        //Recupere la touche pressee et ajuste la vitesse du point
        void handle_input();

        //Montre le point sur l'ecran
        void show();
};
```

Comme vous le voyez, il n'y a rien de compliqué dans notre classe point.

Vous pouvez aussi tout aussi bien appeler cette classe bonhomme et charger une image de bonhomme au lieu de notre image de point, cela marche tout aussi bien.

Je donnerai à la fin de ce tutoriel, en plus des sources pour déplacer le point sur l'écran, des sources pour déplacer un personnage sur de l'herbe.

Revenons à notre classe Point.

Nous avons donc les coordonnées x et y en variable privée ainsi que les vitesses xVel et yVel qui nous permettront de déplacer le point comme vous allez le voir dans la prochaine partie.

Le constructeur nous sert à initialiser les variables, la méthode *handle\_input()* à récupérer les événements et la méthode *show()* à afficher notre objet.

## X-2 - Mouvements du point

Avant de déplacer notre point, il nous faut le placer.

Le constructeur nous permet d'initialiser la position du point sur l'écran ainsi que ses vitesses de déplacement horizontales et verticales (xVel et yVel).

### constructeur

```
Point::Point()
{
    //Initialisation des coordonnees
    x = 0;
    y = 0;

    //Initialisation de la vitesse
    xVel = 0;
    yVel = 0;
}
```

Comme vous le voyez, ce que fait simplement le constructeur est de placer notre objet point dans le coin haut gauche de notre écran SDL.

Maintenant, afin que notre objet point puisse se mouvoir, il nous faut récupérer les événements clavier.

Nous allons donc voir à quoi ressemble notre méthode *handle\_input()*

### handle\_input : touche pressée

```
void Point::handle_input()
{
    //Si une touche a ete pressee
    if( event.type == SDL_KEYDOWN )
    {
        //ajustement de la vitesse
        switch( event.key.keysym.sym )
        {
            case SDLK_UP: yVel -= POINT_HEIGHT / 2; break;
            case SDLK_DOWN: yVel += POINT_HEIGHT / 2; break;
            case SDLK_LEFT: xVel -= POINT_WIDTH / 2; break;
            case SDLK_RIGHT: xVel += POINT_WIDTH / 2; break;
            default: break;
        }
    }
}
```

Voici la partie de code de la méthode `handle_input()` correspondant à la pression d'une touche.

Vous pensez sans doute que tout ce qu'il y aurait eu à faire c'est `x++`(ou `x--`) ou `y++`(ou `y--`) à la pression d'une touche.

Le problème avec ça c'est que le point bougera seulement quand une touche sera pressée.

Ce qui veut dire que vous devez presser la touche, relâcher la touche, puis presser à nouveau la touche encore afin qu'il continue de bouger.

Une façon pas très intelligente de faire serait d'appeler **SDL\_EnableKeyRepeat** car ça poserait plusieurs problèmes : impossibilité de donner une vitesse à notre objet, surcharge de la gestion des événements, etc.

Donc ce que nous faisons à la place est de changer la vitesse (horizontale ou verticale) de notre point.

Quand la touche *droite* est pressée, nous augmentons la vitesse horizontale (`xVel`) de la moitié de la longueur de notre objet point (donc de 10),

ce qui a pour effet d'augmenter sa position X de 10 à chaque rafraîchissement de l'écran (à chaque frame).

De même, lorsque la touche *gauche* est pressée, nous décrétons la vitesse horizontale de 10 ce qui a pour effet de diminuer la position X de

notre objet de 10 à chaque frame.

Le même principe est appliqué pour les coordonnées Y de notre point.

Cependant, souvenez-vous comment marche les coordonnées Y dans une fenêtre SDL : [première application avec SDL](#)

Donc augmenter les coordonnées Y de notre point le fait descendre, et décrétement le fait monter.

Il nous faut aussi traiter quand la touche est relâchée afin de stopper le mouvement du point :

#### handle\_input : touche relâchée

```
//Si une touche a ete relachee
else if( event.type == SDL_KEYUP )
{
    //ajustement de la vitesse
    switch( event.key.keysym.sym )
    {
        case SDLK_UP: yVel += POINT_HEIGHT / 2; break;
        case SDLK_DOWN: yVel -= POINT_HEIGHT / 2; break;
        case SDLK_LEFT: xVel += POINT_WIDTH / 2; break;
        case SDLK_RIGHT: xVel -= POINT_WIDTH / 2; break;
        default: break;
    }
}
```

Quand on relâche la touche, il nous suffit d'annuler l'effet de la pression de la touche.

Quand on appuie sur la touche *droite*, on augmente sa vitesse `xVel` de 10, donc quand on la relâche, il suffit de décrémenter de 10.

Il est maintenant temps d'appliquer ces vitesses et d'afficher notre point et ses mouvements.

### X-3 - Affichage du mouvement

L'affichage et les mouvements de notre point sont gérés par la méthode `show()` que voici :

```
show()
void Point::show()
{
    //Bouge le point à gauche ou à droite
    x += xVel;

    //Si le point se rapproche trop des limites(gauche ou droite) de l'ecran
    if( ( x < 0 ) || ( x + POINT_WIDTH > SCREEN_WIDTH ) )
    {
        //On revient
        x -= xVel;
    }

    //Bouge le point en haut ou en bas
    y += yVel;

    //Si le point se rapproche trop des limites(haute ou basse) de l'ecran
    if( ( y < 0 ) || ( y + POINT_HEIGHT > SCREEN_HEIGHT ) )
    {
        //On revient
        y -= yVel;
    }

    //Affiche le point
    apply_surface( x, y, point, screen );
}
```

Tout d'abord, nous déplaçons notre point en ajoutant la vitesse à ses coordonnées.

Nous vérifions, à chaque mouvement, si le point arrive aux limites de notre écran SDL.

Si c'est le cas, nous annulons le mouvement en soustrayant sa vitesse à ses coordonnées (comme lors du relâchement de la touche).

En faisant ainsi, le point ne peut donc pas sortir de l'écran.

Cependant, en faisant ainsi, il se peut que vous vous retrouviez avec une situation comme celle-ci :



Ceci arrive lorsque vous affectez la vitesse à quelque chose qui n'est pas divisible par les dimensions de l'écran. Donc le point ne bouge pas jusqu'au *mur*, il bouge seulement jusqu'à sa position juste avant.

Une meilleure façon de faire serait de mettre les coordonnées du point égale aux dimensions de l'écran moins les dimensions du point lorsqu'on arrive en collision avec les parois de l'écran.

Je vous laisse essayer.

A la fin de la méthode *show()* on affiche finalement notre point grâce à notre fonction *apply\_surface()*

Il ne nous reste plus grand chose à faire, voici donc à quoi ressemble notre boucle principale dans le main :

#### boucle principale

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

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

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

    //On remplit l'écran de blanc
    SDL_FillRect( screen, &screen->clip_rect, SDL_MapRGB( screen->format, 0xFF, 0xFF, 0xFF ) );

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

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

#### boucle principale

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

Rien de nouveau par rapport aux tutoriels précédents.

Nous vérifions s'il y a un événement pour notre point, puis nous vérifions si l'utilisateur souhaite quitter.

Ensuite nous affichons le fond et nous bougeons et affichons notre point.

Finalement nous mettons à jour notre écran.

A noter 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](#)

## Téléchargements

[Télécharger les sources du X \(172 ko\)](#)

[Télécharger un autre exemple dans lequel on peut déplacer un Tux ou un personnage \(232 ko\)](#)

[Version pdf \(72 ko - 8 pages\)](#) 

## Remerciements

Je remercie [trinityDev](#) pour sa relecture.

Je remercie [fearyourself](#) pour ses corrections supplémentaires.