

Programmation de Jeux 2D: Un morpion en SDL, Première partie

par [Jean Christophe Beyler](#)

Date de publication : 22/03/2006

Dernière mise à jour : 03/04/2006

Beaucoup de débutants tentent de programmer des jeux mais ne savent pas trop par où commencer. En effet, les choses à gérer sont nombreuses et il est souvent facile de se perdre et donc de laisser tomber. Cet article et les suivants aideront, je l'espère, à montrer les fondements de l'élaboration d'un petit moteur 2D utilisant la bibliothèque SDL.

- 1 - Introduction
 - 1.1 - SDL
 - 1.2 - Motivation
 - 1.3 - Plan
- 2 - Ouvrir une fenêtre SDL
 - 2.1 - Initialisation
 - 2.2 - Boucle globale
 - 2.3 - Compilation
- 3 - Conclusion
- 5 - Téléchargements

1 - Introduction

Bienvenue aux tutoriels sur la programmation de jeux en 2D et en utilisant la bibliothèque SDL. Beaucoup de personnes commencent par faire de la 2D avant de passer au 3D. Certes, la troisième dimension permet de faire beaucoup de choses qui risquent de surprendre vos amis et votre famille, mais si vous n'êtes pas encore capables de faire un petit morpion ou un pong, il sera difficile de faire un remake de Half-Life...

C'est donc dans cette optique que je vais présenter comment faire des jeux en 2D, comment gérer l'affichage, le clavier, la souris et le son. Nous allons commencer par des présentations générales pour ensuite travailler sur des sujets de plus en plus spécifiques.

1.1 - SDL

Ces tutoriels utiliseront la bibliothèque portable SDL.

La SDL, Simple DirectMedia Layer, est une bibliothèque destinée à permettre l'accès au matériel graphique pour faire, par exemple, des jeux en plein écran (ou en fenêtre), et ça de manière portable. La SDL peut notamment fonctionner sous Linux (où elle utilise X11), comme sous Windows (où elle utilise DirectX).

Il existe beaucoup d'extensions pour permettre d'inclure du son, faire des transformations de base sur des surfaces, charger des images PNG, TGA... Nous allons voir ici comment créer un moteur 2D avec cette bibliothèque.

La SDL est disponible en licence libre LGPL. Cela veut dire que vous pouvez donc librement utiliser cette bibliothèque dans votre application, qu'elle soit libre ou commerciale. La seule obligation est que la bibliothèque SDL doit rester en liaison dynamique.

Où télécharger la bibliothèque SDL: <http://www.libsdl.org/index.php>

Comment compiler avec la bibliothèque SDL?

Sous Linux/Unix, on utilise l'outil sdl-config pour compiler un programme SDL.

Par exemple:

```
g++ -o main main.cpp `sdl-config --cflags --libs`
```

Pour une présentation plus en détail de la bibliothèque SDL ou pour l'installation sous windows, le tutoriel d'Anomaly explique comment installer SDL avec Dev-C++ (voir [Programmation graphique portable en C avec la SDL](#))

1.2 - Motivation

Souvent lorsqu'on veut faire un jeu, on se jette sur la programmation et on se perd tellement rapidement que le projet tombe à l'eau. Bien que certains projets arrivent à décoller avec cette technique appelée l'Extreme Programming, pour un débutant, c'est presque cause perdue. La motivation de ces tutoriels est donc de présenter comment créer un moteur 2D pas à pas qui permet de faire tourner un jeu sans perdre ses cheveux.

Lorsqu'on commence à faire un projet, il ne faut jamais tenter l'impossible. Il est inutile de tenter de faire un moteur 3D si on ne sait pas faire un morpion en 15 minutes. Ces tutoriels vont servir comme préparation à la création d'un tel moteur. Nous allons donc expliquer comment faire ce morpion :-). Mais aussi comment créer une fenêtre, puis comment créer un petit morpion. Une fois que nous avons réussi à faire cela, nous allons compliquer le moteur pour qu'il puisse gérer un [pong](#). Finalement, nous allons montrer comment faire tourner un casse-brique.

Une chose importante à savoir, c'est que ces petits jeux ne sont pas vraiment le but ultime de ces tutoriels. Arriver à faire un morpion n'est pas en soit une chose compliqué (ni vraiment un pong) mais arriver à créer un code robuste, réutilisable est une autre paire de manches. Donc pour chaque jeu, je vais tenter de proposer des extensions qui compliqueront le jeu de base mais ajouteront le petit plus qu'on cherche tous.

1.3 - Plan

Ce groupe de tutoriels va donc montrer les techniques de base que j'utilise pour créer des petits jeux en 2 dimensions en utilisant la bibliothèque SDL. Je vais commencer par un des jeux les plus faciles à implémenter: le morpion.

Le but de cette première partie est de se familiariser avec cette bibliothèque. Nous verrons dans les parties suivantes que, finalement, l'implémentation du morpion n'est qu'un détail par rapport au but recherché: programmer un jeu. Ces tutoriels supposeront une petite connaissance en C++ (de toute façon, je suis plutôt un programmeur C, donc la partie C++ restera relativement simple).

Ce premier tutoriel sera divisé en huit parties:

- 1 Ouvrir une fenêtre SDL;
- 2 Lier la souris à la fenêtre et afficher des ronds à l'endroit cliqué;
- 3 Ajouter les règles du jeu et le finaliser
- 4 Ajouter une classe Objet pour rendre le jeu plus souple
- 5 Ajouter un menu dans un jeu
- 6 Ajouter une Intelligence Artificielle
- 7 Ajouter du son
- 8 Extensions

Attention, cet article ne parlera pas tout le temps d'optimisation de code ou de technique pour rendre l'affichage le plus rapide possible. Nous commencerons d'abord par faire un code qui marche et, seulement après, nous nous intéresserons aux optimisations possibles.

2 - Ouvrir une fenêtre SDL

2.1 - Initialisation

Puisque le code de cette première partie tient sur un seul fichier, je vais simplement expliquer chaque partie du code de façon linéaire... Lorsque le code commencera à être plus compliqué, ne vous inquiétez pas, les explications seront plus thématiques. Pour l'instant, intéressons-nous à la création d'une fenêtre SDL. Le programme commence par de simples inclusions de fichiers d'entête.

Inclusions

```
#include <SDL.h>
#include <iostream>
```

Ensuite, bonne ou mauvaise habitude, lorsque je crée une fenêtre, j'utilise des constantes pour définir la taille de la fenêtre. Pourquoi? Parce que si le code est en fonction de ces constantes, il sera plus facile de passer à une version qui a une taille de fenêtre modifiable. Mais nous n'en sommes pas encore là.

Définitions de constantes

```
const int WIDTH=640;
const int HEIGHT=480;
```

Finalement, nous définissons la fonction main, elle contiendra l'initialisation SDL, la création de la fenêtre et la boucle événementielle.

Déclarations des variables locales

```
int main(int argc, char **argv)
{
    //Déclaration des variables
    SDL_Event event;
    SDL_Surface *screen;
    int done = 0;
```

L'initialisation de SDL se fait avec la fonction `SDL_Init`. Puisque nous voulons ouvrir une fenêtre, nous passons le flag `SDL_INIT_VIDEO`. Comme tout bon programmeur, on teste le retour de `SDL_Init`. Il faut toujours tester le retour des fonctions SDL (et les autres d'ailleurs).

Initialisation de SDL

```
//Initialisation de SDL
if(SDL_Init(SDL_INIT_VIDEO)!=0) {
    std::cerr << "Problème pour initialiser SDL\n" << SDL_GetError() << std::endl;
    return 1;
}
```

Nous pouvons si nous le désirons, donner un titre à la fenêtre obtenue:

Mise en place d'un titre pour la fenêtre

```
//Mettre un titre à la fenêtre
SDL_WM_SetCaption("Programme SDL de base", NULL);
```

Je mets cet appel avant la création du programme parce que, si vous voulez aller plus loin, vous pouvez appeler la fonction `SDL_WM_SetIcon` qui elle doit être appelée avant le premier appel de la fonction qui crée une surface de fenêtre: `SDL_SetVideoMode`. La fonction `SDL_WM_SetIcon` permet de définir quelle icône sera utilisée par la fenêtre. Ensuite on récupère une surface pour la fenêtre avec la fonction `SDL_SetVideoMode`.

Création de la fenêtre

```
//Ouverture d'une surface pour l'affichage de la fenêtre
screen = SDL_SetVideoMode(WIDTH,HEIGHT, 32,
    SDL_DOUBLEBUF | SDL_HWSURFACE);
if(screen==NULL)
    done = 1;
```

Les paramètres de cette fonction sont:

- 1 La largeur de la fenêtre;
- 2 La hauteur de la fenêtre;
- 3 Le nombre de bits par pixel (32 veut donc dire 3 couleurs et un octet pour l'alpha);
- 4 Quel genre de fenêtre... Nous demandons un double buffer et on demande de le mettre dans la mémoire video (Voir la [FAQ](#) pour plus de détails).

2.2 - Boucle globale

Finalement, nous avons une boucle infinie (du moins tant que le programme doit continuer). Le programme sortira de la boucle lorsque le programme n'a plus besoin de la fenêtre. Cela se traduit par: lorsque done==1, nous sortons de la boucle et le programme se terminera.

Boucle générale

```
//Boucle générale
while(!done)
{
```

Dans cette boucle englobante, se trouve une boucle événementielle, nous allons d'abord regarder les événements à gérer. Comme beaucoup de bibliothèques du genre, SDL fonctionne avec une pile d'événements (comme l'API Windows par exemple). Chaque événement (clic souris, frappe de clavier, changement de l'état de la fenêtre, etc...) provoque un appel à la fonction

Prototype de fonction

```
int SDL_PushEvent(SDL_Event *event);
```

Logiquement, il faut vider cette pile et nous le faisons avec une deuxième boucle. Pour récupérer ce qui se trouve dans la pile, nous utilisons la fonction

Prototype de fonction

```
int SDL_PollEvent(SDL_Event *event);
```

La boucle interne la plus simple (avec aucune gestion des événements) ressemble donc à:

Boucle événementielle générale

```
//Traiter les événements
while(SDL_PollEvent(&event))
{
    //Pour chaque événement, on regarde d'abord le type
    switch(event.type)
    {
        //Gestion de l'événement en fonction de son type

        //Si on ne veut pas le gérer, on ne fait rien
        default:
            break;
    }
}
```

Donc dans notre cas, nous avons:

```

Notre Boucle événementielle
//Traiter les évènements
while(SDL_PollEvent(&event))
{
    switch(event.type)
    {
        //Si on veut quitter, done=1 suffira
        case SDL_QUIT:
            done=1;
            break;
        //Si on vient de relâcher une touche de clavier
        case SDL_KEYUP:
            //Et si c'est la lettre q
            if(event.key.keysym.sym==SDLK_q)
                //On met done a 1 pour quitter
                done=1;
            break;
        //Sinon on ne fait rien
        default:
            break;
    }
}
    
```

Les commentaires pourraient suffire pour la compréhension de cette boucle mais je vais prendre quelques minutes pour mieux expliquer. Pour chaque événement, on peut affecter un comportement particulier. Pour distinguer le comportement, on utilise donc un switch avec la valeur de event.type.

Ainsi, si l'événement est de type SDL_QUIT, cela veut dire que quelque part dans le code, on a demandé de quitter le programme, il semble naturel de terminer cette boucle. En positionnant done à 1, on assure que la boucle englobante ne fera pas une autre itération mais on terminera quand même les événements restantes sur la pile. Cela semble tout de même plus propre.

Le deuxième type d'événement est SDL_KEYUP qui est lancé lorsque l'utilisateur a appuyé sur une touche et la relâche. Lorsque nous recevons ce type d'événement, il semble également logique de vérifier quelle touche a été relâchée. Si c'est le 'q', on va demander de quitter (en positionnant done à 1) Il a été porté à mon attention que ce bout de code n'est pas tout à fait portable. En effet, ce code ne fonctionnera pas correctement sur certains Systèmes d'exploitation avec certains claviers. Pour le rendre portable, il faudrait passer par les codes unicode. Mais cela sort du cadre de ce premier tutoriel. Remarquez simplement que pour sortir, il faudrait peut-être utiliser la lettre 'a' ou alors mettre *SDLK_a* à la place de *SDLK_q*. Si nous voulions faire les choses dans les règles, une solution élégante serait de poser sur la pile un événement SDL_QUIT pour n'avoir le code de fin de programme à un seul endroit. Ce sera une amélioration pour la suite.

Après notre boucle externe, on quitte avec SDL_Quit.

```

Nettoyage mémoire
SDL_Quit();
    
```

2.3 - Compilation

Finalement, pour compiler le programme SDL, il suffit de faire ceci:

```

Compilation
g++ -o main Main.cpp `sdl-config --cflags --libs`
    
```

3 - Conclusion

Voilà pour cette première partie, nous avons fait une introduction de l'utilisation de la bibliothèque SDL et ce n'est sûrement pas inutile.

Par la suite, nous verrons comment lier la souris à l'application et comment associer les clics de souris à des zones de la fenêtre.

Jc

- [Sommaire du tutoriel](#);
- [Ouvrir une fenêtre SDL](#);
- [Lier la souris à la fenêtre et afficher des ronds à l'endroit cliqué](#);
- [Ajouter les règles du jeu et le finaliser](#);
- [Ajouter les classes Objet et Moteur pour rendre le jeu plus souple](#);
- [Ajouter un menu dans un jeu](#);
- [Ajouter une Intelligence Artificielle \(Min-Max\)](#);
- [Ajouter une Intelligence Artificielle \(Alpha-Beta\)](#);

5 - Téléchargements

Voici le code source pour ce premier tutoriel: [\(3 Ko\)](#).

Voici la version pdf de cet article: [\(37 Ko\)](#).

Si vous avez des suggestions, remarques, critiques, si vous avez remarqué une erreur, ou bien si vous souhaitez des informations complémentaires, n'hésitez pas à me contacter !