

La gestion du joystick avec la SDL

Par François Sellier (François.S)



www.openclassrooms.com

*Licence Creative Commons 6 2.0
Dernière mise à jour le 2/01/2012*

Sommaire

Sommaire	2
La gestion du joystick avec la SDL	3
Qu'est-ce qu'un joystick ?	3
De quels éléments est formé un joystick ?	3
Et la SDL dans tout ça ?	4
Obtenir des informations sur le joystick	5
Obtenir les informations générales sur le joystick	5
Obtenir les informations avancées sur le joystick	7
La gestion des événements du joystick	10
Les boutons	10
Les axes	11
Les chapeaux	12
Les trackballs	13
Savoir sur quel joystick se produit l'évènement	14
Gestion avancée des événements du joystick	15
La fonction initialiserInput()	16
La fonction détruireInput()	17
La fonction updateEvent()	17
En résumé	18
Comment s'en servir ?	20
Améliorations	23
Correction	23
Comment gérer tout ça ?	28
Améliorations, encore et toujours !	30
Q.C.M.	31
Partager	32



La gestion du joystick avec la SDL

Par [François Sellier \(François.S\)](#)


Mise à jour : 02/01/2012

Difficulté : Facile  Durée d'étude : 1 jour



Bonjour à tous !

Vous avez envie de créer un jeu vidéo qui utilise une manette ? Un joystick ? Eh bien... vous êtes au bon endroit. Nous allons voir comment utiliser un joystick avec la SDL.

Bon, n'attendons pas plus longtemps. A l'attaque ! 



Il faut avoir lu le tuto de [M@teo21](#) au moins jusqu'à la première partie des chapitres sur les événements de la SDL.

Sommaire du tutoriel :




- [Qu'est-ce qu'un joystick ?](#)
- [Obtenir des informations sur le joystick](#)
- [La gestion des événements du joystick](#)
- [Gestion avancée des événements du joystick](#)
- [Améliorations](#)
- [Q.C.M.](#)


Qu'est-ce qu'un joystick ?

Tout d'abord, je tiens à dire qu'un joystick peut être :

- un **joystick** ;
- une **manette** ;
- un **trackball**.

Donc le mot "joystick" n'est en fait pas tellement approprié étant donné que cela prend en compte plusieurs objets différents, mais la SDL parle de joystick pour tout désigner.

Donc ne vous inquiétez pas si vous avez une manette, ça marchera aussi. 

J'ai décidé de commencer le cours par quelques explications théoriques puis nous commencerons à programmer par la suite. 

De quels éléments est formé un joystick ?

Je pense qu'une image vaut mieux qu'un long discours :



(Vous l'avez peut-être remarqué, c'est une manette d'Xbox 360)

Un joystick est donc composé de :

- **Boutons** (*buttons* en anglais) ;
- **Axes** (*axis* en anglais) ;
- **Chapeaux** (*hats* en anglais) ;
- **TrackBalls** (là il n'y en a pas sur ma manette).

En voyant l'image, vous devriez pouvoir deviner quel type de signaux nous serons envoyés...
Bon alors là, j'espère que vous suivez toujours sinon...

Et la SDL dans tout ça ?

Maintenant, nous allons voir ce que la SDL peut faire pour gérer tous ces éléments.

Les boutons

Pour gérer les boutons, c'est très simple. En fait c'est comme pour le clavier : SDL nous dit quand un bouton est appuyé, et quand il est relâché. 😊

Les axes

Là encore ce n'est pas très compliqué. SDL nous donne donc l'inclinaison du stick et également sur quel axe. Nous verrons comment bien gérer tout ça par la suite.

Les chapeaux

Les chapeaux peuvent avoir plusieurs directions qui sont gérées par des énumérations (UP;DOWN...).

Les trackballs

Nous verrons également la gestion des trackballs, qui nous diront de combien ils ont bougé et dans quelle direction.
Voici à quoi ressemble un trackball :



Le trackball est la boule du milieu

C'est bon, pas trop difficile ?

Obtenir des informations sur le joystick

Bon, on va attaquer la programmation. Créez un projet SDL et c'est parti ! 🎉🧙

Obtenir les informations générales sur le joystick

Initialiser le sous-programme joystick

Pour effectuer des opérations de base avec le joystick, vous devez l'initialiser. Pour ce faire, il faut juste rajouter `SDL_INIT_JOYSTICK` quand on appelle la fonction `SDL_Init()`.

Cela donne :

Code : C

```
if(SDL_Init(SDL_INIT_VIDEO | SDL_INIT_JOYSTICK) < 0) //  
    initialisation du joystick et du mode vidéo  
    return EXIT_FAILURE;
```



Dans la suite de cette partie, nous allons récupérer des informations sur les joysticks que nous devons écrire. Donc si vous êtes sous Linux, vous aurez la chance de ne pas avoir à regarder dans un fichier, et de simplement faire un `printf()`. Pour les autres, pas de bol, il faudra coder l'ouverture et la fermeture d'un fichier que nous lirons par la suite. J'ai donc choisi de coder l'ouverture/fermeture d'un fichier pour que tout le monde puisse lire ce tutoriel sans se casser la tête à programmer la gestion des fichiers. 😊



Si vous êtes sous Visual, vous pouvez tout de même activer la console lors de l'exécution du programme et ainsi faire un `printf()` et lire directement dans la console. Voici la marche à suivre :

Allez dans les propriétés de votre projet,

--> Propriété de configuration

-> éditeur de lien

> système

et remplacez "`Windows/subsystem:windows`"
par "`console/subsystem:console`"

Un grand merci à [ghighigames](#) pour ses explications. 😊

Compter les joysticks

Pour utiliser un joystick, il faudra vérifier qu'il y en a bien au moins un de branché. Pour cela, il faut savoir combien il y en a... Pour les compter, on utilise la fonction `SDL_NumJoysticks()`. Le prototype n'est pas compliqué car la fonction ne prend aucun paramètre :

Code : C

```
int SDL_NumJoysticks(); // retourne le nombre de joysticks
```

Nous allons donc voir combien il y a de joysticks :

Code : C

```
#include <stdlib.h>
#include <SDL/SDL.h>

int main(int argc, char **argv)
{
    if(SDL_Init(SDL_INIT_JOYSTICK) < 0) // initialise juste le
        joystick (pas besoin d'une fenêtre pour nos tests)
        return EXIT_FAILURE;

    FILE* fichier = NULL;
    fichier = fopen("Joystick.txt", "w+"); // on crée un fichier
    Joystick.txt

    if(fichier != NULL)
    {
        fprintf(fichier, "Il y a %d joysticks.", SDL_NumJoysticks());
        // on écrit combien il y a de joysticks
        fclose(fichier); // on referme le fichier
    }

    else
        return EXIT_FAILURE;

    SDL_Quit();
    return EXIT_SUCCESS;
}
```

Pour mon compte, j'obtiens dans mon fichier texte :

Code : Console

```
Il y a 1 joysticks.
```

Citation : Le fou du français

Secret (cliquez pour afficher)

Au secours !

... enfin bon, si vous relisez le code, vous comprendrez que même s'il y a 1, 2 ou 0 joysticks, il y aura toujours un "s". 😊

Oula je suis en train de virer au cours de français, mais retenez tout de même qu'on ne met pas de "s" à la fin d'un mot si celui-ci est au singulier (enfin pour la plupart). 😊

Si vous voulez programmer pour éviter la faute... et bien faites le.

En résumé : je n'ai qu'un seul joystick connecté.

Un utilisateur qui aura 0 joysticks devra faire avec son clavier.

Les noms des joysticks

On s'en sert moins souvent, mais c'est parfois utile de savoir comment récupérer le nom du joystick utilisé. Par exemple pour que l'utilisateur ait le choix entre deux noms de joysticks plutôt que entre "joystick n°1" et "joystick n°2".

Pour ce faire, on utilise la fonction **SDL_JoystickName()** dont voici le prototype :

Code : C

```
const char* SDL_JoystickName(int numeroJoystick); // retourne le
nom du joystick choisi
```

Cette fonction retourne un **const char*** car le nom du joystick est **constant** (il ne va pas changer de nom au beau milieu de l'exécution 🤖).

Si on reprend l'exemple précédent et qu'on y rajoute un peu de code, on a le nom des joysticks :

Code : C

```
#include <stdlib.h>
#include <SDL/SDL.h>

int main(int argc, char **argv)
{
    if(SDL_Init(SDL_INIT_VIDEO | SDL_INIT_JOYSTICK) < 0)
        return EXIT_FAILURE;

    FILE* fichier = NULL;
    fichier = fopen("Joystick.txt", "w+"); // on crée un fichier
    Joystick.txt

    if(fichier != NULL)
    {
        fprintf(fichier, "Il y a %d joysticks.", SDL_NumJoysticks());
        // on écrit combien il y a de joysticks
        for(int i=0; i<SDL_NumJoysticks(); i++)
        // tant qu'il y a un joystick non-traité
            fprintf(fichier, "Nom du joystick numero %d :
%s", i, SDL_JoystickName(i)); // on écrit les noms des joysticks
        fclose(fichier); // on referme le fichier
    }

    else
        return EXIT_FAILURE;

    SDL_Quit();
    return EXIT_SUCCESS;
}
```

J'obtiens :

Code : Console

```
Il y a 1 joysticks.
Nom du joystick numero 0 : Controller (XBOX 360 For Windows)
```

Si vous n'avez pas de joystick connecté, le nom du joystick 0 est (NULL). Eh... ou plutôt vous n'avez rien. 🤖

Obtenir les informations avancées sur le joystick

Nous allons voir comment obtenir davantage d'informations sur notre joystick :

- nombre de **boutons** ;
- nombre d'**axes** ;
- nombre de **chapeaux** ;
- nombre de **trackballs**.

Les fonctions permettant d'accéder à ces informations ne demandent plus le numéro du joystick mais une structure de type `SDL_Joystick`.

Il faut également dire à la structure quel joystick utiliser. Pour cela, il y a la fonction **`SDL_JoystickOpen()`**.

On résume tout ça dans ce code :

Code : C

```
SDL_Joystick *joystick; // attention, c'est bien un pointeur !
joystick = SDL_JoystickOpen(0); // prend en paramètre le numéro du
joystick, ici 0
```

Mettez cela juste après l'initialisation du joystick.

Et qui dit ouvrir... dit fermer. 🤖

A la fin du programme, il faut également fermer le joystick à l'aide de **`SDL_JoystickClose()`**.

Code : C

```
SDL_JoystickClose(joystick);
```

Cette fonction prend en paramètre votre structure `SDL_Joystick`. Mettez ça à la fin du programme, juste avant **`SDL_Quit()`**.

Bon, maintenant, on va pouvoir en connaître plus sur notre joystick. 😊

Les boutons

Pour avoir le nombre de boutons du joystick, on utilise la fonction **`SDL_JoystickNumButtons()`** :

Code : C

```
int SDL_JoystickNumButtons(SDL_Joystick *joystick); // retourne le
nombre de boutons sur le joystick
```

Les axes

Pour avoir le nombre d'axes du joystick, on utilise la fonction **`SDL_JoystickNumAxis()`** :

Code : C

```
int SDL_JoystickNumAxis(SDL_Joystick *joystick); // retourne le
nombre d'axes sur le joystick
```

Les chapeaux

Pour connaître le nombre de chapeaux du joystick, on utilise la fonction **`SDL_JoystickNumHats()`** :

Code : C

```
int SDL_JoystickNumHats(SDL_Joystick *joystick); // retourne le
nombre de chapeaux sur le joystick
```

Les trackballs

Et enfin, pour avoir le nombre de trackball du joystick, on utilise la fonction **`SDL_JoystickNumBalls()`** :

Code : C


```
int SDL_JoystickNumBalls(SDL_Joystick *joystick); // retourne le
nombre de trackballs sur le joystick
```

Toutes les fonctions précédentes prenaient en paramètre un pointeur sur une structure `SDL_Joystick`.

En résumé

Maintenant, on va rajouter tout ça dans notre code :

Code : C

```
#include <stdlib.h>
#include <SDL/SDL.h>

int main(int argc, char **argv)
{
    if(SDL_Init(SDL_INIT_VIDEO | SDL_INIT_JOYSTICK) < 0)
        return EXIT_FAILURE;

    SDL_Joystick *joystick; // on crée le joystick
    joystick = SDL_JoystickOpen(0); // on l'assigne au numéro 0

    FILE* fichier = NULL;
    fichier = fopen("Joystick.txt", "w+"); // on crée un fichier
    Joystick.txt

    if(fichier != NULL)
    {
        fprintf(fichier, "Il y a %d joysticks.", SDL_NumJoysticks());
        // on écrit combien il y a de joysticks
        for(int i=0; i<SDL_NumJoysticks(); i++)
            fprintf(fichier, "Nom du joystick numero %d :
%s", i, SDL_JoystickName(i)); // on écrit les noms des joysticks

        fprintf(fichier, "Nombre de boutons :
%d", SDL_JoystickNumButtons(joystick)); // nombre de boutons
        fprintf(fichier, "Nombre d'axes :
%d", SDL_JoystickNumAxis(joystick)); // nombre d'axes
        fprintf(fichier, "Nombre de chapeaux :
%d", SDL_JoystickNumHats(joystick)); // nombre de chapeaux
        fprintf(fichier, "Nombre de trackballs :
%d", SDL_JoystickNumBalls(joystick)); // nombre de trackballs
        fclose(fichier); // on referme le fichier
    }

    else
        return EXIT_FAILURE;

    SDL_JoystickClose(joystick);
    SDL_Quit();
    return EXIT_SUCCESS;
}
```

Résultat :

Code : Console

```
Il y a 1 joysticks.
Nom du joystick numero 0 : Controller (XBOX 360 For Windows)
Nombre de boutons : 10
Nombre d'axes : 5
Nombre de chapeaux : 1
Nombre de trackballs : 0
```

Comme vous le voyez, je n'ai pas de trackball. Une manette n'a généralement pas de trackball, un trackball est plutôt une sorte de "souris alternative".



Hein !? Tant d'axes ? Je crois que ton truc bug !

Non, non. 😊

En fait un "stick" possède 2 axes : gauche/droite et haut/bas. Mais nous verrons ça plus tard.

Et j'ai 5 axes car j'ai 2 sticks et le dernier axe correspond à LT et RT pour les manettes d'*xbox360* ou L2 et R2 pour les manettes de *PS3*.

Bon, tout ce qu'on vient de faire, c'était de la rigolade. 😂 Je pense que vous êtes en train de lire ceci car vous voulez savoir comment gérer les événements pas pour compter les joysticks. 😊

Alors je pense que vous trouverez la suite plus intéressante...

La gestion des événements du joystick

Bon, maintenant, on entre dans le cœur du sujet : la gestion des événements. 😊

Mais avant tout, il y a encore une initialisation(une seule je vous rassure), pour pouvoir gérer les événements du joystick.

Code : C

```
SDL_JoystickEventState(SDL_ENABLE);
```

Cette instruction se lit littéralement : statut des événements du joystick : activé.
Ce code active les événements du joystick que nous allons utiliser.



Peut-on également désactiver les événements du joystick ?

Oui, il suffit de passer en argument `SDL_IGNORE` pour les désactiver. 😊

Les boutons

Bon... Alors là, c'est vraiment très simple : un bouton peut générer deux événements :

- `SDL_JOYBUTTONDOWN` quand on appuie sur le bouton
- `SDL_JOYBUTTONUP` quand on relâche le bouton

Pour savoir quel bouton est pressé, on récupère la variable `evenements.jbutton.button`.

Par exemple :

Code : C

```
switch(evenements.type)
{
    case SDL_JOYBUTTONDOWN:
        switch(evenements.jbutton.button)
        {
            case 0 : // appui sur bouton 0
                quitter = 1; // on quitte
                break;
            case 7 : // appui sur bouton 7
                quitter = 1; // on quitte aussi
                break;
        }
}
```

```
        break;
    }
```

Comme vous avez pu le voir, la gestion des boutons est très facile. 😊

Bon, on passe à la suite ! 🧑🏻

Les axes

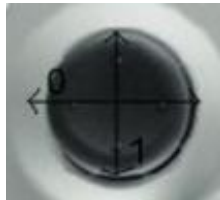
Un axe ne peut générer qu'un seul évènement : quand on change son **inclinaison**. Pour savoir si l'axe a bougé, on utilise l'évènement `SDL_JOYAXISMOTION`.

Pour la gestion des axes, il y a deux variables à prendre en compte :

- `evenements.jaxis.value`, c'est la valeur de l'axe
- `evenements.jaxis.axis`, c'est l'axe concerné

Revenons au problème posé par les axes : pourquoi y avait-il tant d'axes ?

Voici la réponse :



Vous voyez donc qu'il y a deux axes : gauche/droite et haut/bas. 😊

Sachez qu'un axe peut avoir une valeur positive ou négative : quand l'axe 0 est négatif, il est vers la gauche, et quand il est positif, il est vers la droite.

En combinant intelligemment ces deux variables on peut arriver à gérer la direction du stick :

Code : C

```
switch(evenements.type)
{
    case SDL_JOYAXISMOTION:
        if(evenements.jaxis.axis == 0 && evenements.jaxis.value < -
2000) // si la valeur de l'axe 0 est inférieure à -2000

        // donc si il est vers la gauche (voir paragraphe ci dessus)
            quitter = 1; // alors on quitte
        break;
}
```

Ce programme s'arrête quand on pousse le stick vers la gauche.



Et si je veux gérer plusieurs sticks ?

Eh bien, il suffit de voir si un autre axe peut être utilisé (par exemple le 2;3... En fait, on peut tous les utiliser 🤖).

Un autre exemple :

Code : C

```
switch(evenements.type)
{
    case SDL_JOYAXISMOTION:
        if(evenements.jaxis.axis == 3 && evenements.jaxis.value < -
2000) // si la valeur de l'axe 3 est inférieure à -2000
            quitter = 1; // alors, on quitte
```

```
        break;  
    }
```

Les chapeaux

Comme je le disais au début, les chapeaux ont plusieurs directions. Donc, logiquement, ils ne génèrent qu'un seul évènement : `SDL_JOYHATMOTION`.
`SDL_JOYHATMOTION` signifie "mouvement du chapeau".

Si vous avez compris, essayez de deviner ce que fait ce code :

Code : C

```
switch(evenements.type)  
{  
    case SDL_JOYHATMOTION:  
        quitter = 1;  
        break;  
}
```

La réponse est :

Secret (cliquez pour afficher)

Le programme s'arrête lorsque l'utilisateur bouge le chapeau de son joystick. 🤔

C'est bien beau de savoir qu'un chapeau a bougé, mais on ne sait ni lequel, ni dans quelle direction !

Déjà, pour savoir quel chapeau est utilisé, on utilise la variable `evenements.jhat.hat`.
Ce nombre est donc le numéro du chapeau utilisé. 🤔

Et puis pour savoir dans quel direction il est, on récupère une valeur dans `evenements.jhat.value` dont voici les valeurs possibles :

Valeur	Signification
<code>SDL_HAT_UP</code>	en haut
<code>SDL_HAT_DOWN</code>	en bas
<code>SDL_HAT_RIGHT</code>	à droite
<code>SDL_HAT_LEFT</code>	à gauche
<code>SDL_HAT_RIGHTDOWN</code>	en bas à droite
<code>SDL_HAT_RIGHTUP</code>	en haut à droite
<code>SDL_HAT_LEFTDOWN</code>	en bas à gauche
<code>SDL_HAT_LEFTUP</code>	en haut à gauche
<code>SDL_HAT_CENTERED</code>	au milieu(n'est dans aucune direction)

Et maintenant que vous savez tout ça, on peut savoir quel chapeau a bougé et dans quelle direction. 🤔

Maintenant, devinez ce que fait ce code, en vous aidant du tableau ci-dessus.

Code : C

```
switch(evenements.type)
```

```

{
    case SDL_JOYHATMOTION:
        if (evenements.jhat.hat == 0 && evenements.jhat.value ==
            SDL_HAT_LEFTDOWN)
            quitter = 1;
        break;
}

```

Secret (cliquez pour afficher)

Ce programme s'arrête quand le chapeau est orienté en bas à gauche. 😊

Les trackballs

Tout comme les axes, les trackballs ne génèrent qu'un seul type d'évènements : ils bougent. Pour savoir s'ils ont bougé, on utilise la variable `SDL_JOYBALLMOTION`.

Donc si on fait ceci :

Code : C

```

switch (evenements.type)
{
    case SDL_JOYBALLMOTION :
        quitter = 1;
        break;
}

```

... le programme s'arrête lorsque un mouvement du trackball se produit.

Maintenant, on va voir comment savoir dans quel direction il a bougé : pour cela on utilise les coordonnées relatives que voici :

- `evenements.jball.xrel` pour les coordonnées relatives de l'axe x
- `evenements.jball.yrel` pour les coordonnées relatives de l'axe y



Les coordonnées relatives permettent en quelque sorte de savoir à quelle vitesse le trackball a bougé. La souris a également des coordonnées relatives.

Maintenant, on va voir comment savoir quel trackball est utilisé. Pour ce faire, nous devons utiliser cette variable : `evenements.jball.ball`.

Un exemple pour mieux comprendre :

Code : C

```

switch (evenements.type)
{
    case SDL_JOYBALLMOTION:
        if (evenements.jball.ball == 0 && evenements.jball.xrel > 10)
            // si le trackball n°0 est utilisé et bouge vers la droite
            quitter = 1; // on quitte
        break;
}

```



Si le trackball a sa valeur `xrel` supérieure à 0, c'est qu'il a bougé vers la droite, sinon c'est vers la gauche.

Un autre exemple :

Code : C

```

switch(evenements.type)
{
    case SDL_JOYBALLMOTION: // si un mouvement du trackball s'est
produit
        /* on ajoute les valeurs qui ont changé */
        posSourisX += evenements.jball.xrel;
        posSourisY += evenements.jball.yrel;

        /* on vérifie que le curseur ne sort pas de l'écran */
        posSourisX = (posSourisX < 0) ? (0) : (posSourisX); // les
conditions sont en ternaire
        posSourisX = (posSourisX > LARGEUR_FENETRE) ?
(LARGEUR_FENETRE) : (posSourisX);
        posSourisY = (posSourisY < 0) ? (0) : (posSourisY);
        posSourisY = (posSourisY > HAUTEUR_FENETRE) ?
(HAUTEUR_FENETRE) : (posSourisY);
        break;
}

```

Rappel sur les conditions ternaires :

(expression 1) ? (expression 2) : (expression 3);



se lit :

si l'expression 1 est vrai, exécuter l'expression 2, sinon exécuter l'expression 3

Savoir sur quel joystick se produit l'évènement

Jusqu'à maintenant, nous gérons les évènements du joystick en pensant **qu'un seul** joystick était utilisé. Mais si vous voulez créer un jeu en écran splitté avec donc plusieurs manettes, vous devrez savoir quel joystick est utilisé..

Pour cela, la SDL nous donne le numéro du joystick dans plusieurs variables :

- `evenements.jaxis.which`
- `evenements.jbutton.which`
- `evenements.jhat.which`
- `evenements.jball.which`

En fait, ce sont tous les *which* qui sont dans les structures en rapport avec un des éléments d'un joystick. 🤖

Donc maintenant que vous savez ça, vous saurez me dire ce que fait ce code :

Code : C

```

switch(evenements.type)
{
    case SDL_JOYHATMOTION:
        switch(evenements.jhat.which)
        {
            case 0:
                switch(evenements.jhat.hat)
                {
                    case 0:
                        switch(evenements.jhat.value)
                        {
                            case SDL_HAT_UP:
                                avancerPerso(&perso1, HAUT);
                                break;
                            case SDL_HAT_DOWN:
                                avancerPerso(&perso1, BAS);
                                break;
                            case SDL_HAT_RIGHT:
                                avancerPerso(&perso1, DROITE);

```

```

        break;
        case SDL_HAT_LEFT:
            avancerPerso (&perso1, GAUCHE) ;
            break;
    }
    break;
}
break;

case 1:
    switch (evenements.jhat.hat)
    {
        case 0:
            switch (evenements.jhat.value)
            {
                case SDL_HAT_UP:
                    avancerPerso (&perso2, HAUT) ;
                    break;
                case SDL_HAT_DOWN:
                    avancerPerso (&perso2, BAS) ;
                    break;
                case SDL_HAT_RIGHT:
                    avancerPerso (&perso2, DROITE) ;
                    break;
                case SDL_HAT_LEFT:
                    avancerPerso (&perso2, GAUCHE) ;
                    break;
            }
            break;
        }
        break;
    }
    break;
}
}
}

```

Secret (cliquez pour afficher)

Le joystick n°0 dirige le personnage 1 avec son chapeau n°0 et le joystick n°1 dirige le personnage 2 avec son chapeau n°0.



Mais il est fou celui là à utiliser autant de switches.

Non, je suis pas fou (jusqu'à preuve du contraire), et la prochaine partie va nous permettre d'éviter une noyade dans ce genre de code. 😊

Gestion avancée des évènements du joystick

Passons maintenant à notre partie sur la gestion avancée. 😎

Nous allons utiliser la technique qu'utilise [Fvirtman](#) dans son tutoriel, donc je vous conseille de le lire pour comprendre ce qui suit. Cette technique consiste à tout mettre dans un tableau et d'accéder aux case avec des if(...) pour gérer les évènements. Je peux vous dire qu'une fois programmée, cette gestion est beaucoup plus simple et le code est moins lourd qu'avec nos switches.



- [Simplifier les events avec SDL](#)

J'attends...

...

...

...

C'est parti !

Bon alors, je commence par quelques explications : nous allons créer des pointeurs que nous allouerons par la suite avec une fonction **initialiserInput()** et détruits par une fonction **détruireInput()**. Une fonction **updateEvent()** vas nous permettre de récupérer les évènements de notre joystick.

Voici le fichier **.h** où on y trouve la structure Input et les fonctions qui vont avec :

Code : C

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

typedef struct InputTrackball // je crée une structure car les
trackballs ont 2 variables à prendre en compte
{
    int xrel; // valeur x réelle
    int yrel; // valeur y réelle
};

typedef struct Input
{
    SDL_Joystick *joystick; // le joystick
    char *boutons; // les boutons
    int *axes; // les axes
    int *chapeaux; // les chapeaux
    InputTrackball trackballs; // les trackballs

    int numero; // le numero
};

void initialiserInput(Input *input,int numeroJoystick); //
initialise en fonction du numéro du joystick
void détruireInput(Input *input); // libère la mémoire allouée
void updateEvent(Input *input); // récupère les évènements
```

La fonction **initialiserInput()**

Voici le but de cette fonction : allouer et mettre à zéro tous les éléments de la structure.

Code : C

```
void initialiserInput(Input *input,int numeroJoystick)
{
    if(numeroJoystick < SDL_NumJoysticks()) // on vérifie qu'il y a
    bien un bon numéro de joystick
    {
        input->joystick = SDL_JoystickOpen(numeroJoystick); // on
        met le joystick à numéro correspondant
        input->numero = numeroJoystick; // je pense que vous
        comprenez cette ligne...

        /* on alloue chaque éléments en fonctions de combien il y
        en a */
        input->boutons = (char*)
        malloc(SDL_JoystickNumButtons(input->joystick) * sizeof(char));
        input->axes = (int*) malloc(SDL_JoystickNumAxis(input->
        joystick) * sizeof(int));
        input->chapeaux = (int*) malloc(SDL_JoystickNumHats(input->
        joystick) * sizeof(int));
        input->trackballs = (InputTrackball*)
        malloc(SDL_JoystickNumBalls(input->joystick) *
        sizeof(InputTrackball));

        for(int i=0;i<SDL_JoystickNumButtons(input->joystick);i++)
        // tant qu'on a pas atteint le nombre max de boutons
        input->boutons[i] = 0; // on met les valeurs à 0
```



```

        for(int i=0;i<SDL_JoystickNumAxes(input->joystick);i++) //
        tant qu'on a pas atteint le nombre max d'axes
            input->axes[i] = 0; // on met aussi les valeurs à 0
        for(int i=0;i<SDL_JoystickNumHats(input->joystick);i++) //
        tant qu'on a pas atteint le nombre max de chapeaux
            input->chapeaux[i] = SDL_HAT_CENTERED; // on dit que les
            chapeaux son centrés
        for(int i=0;i<SDL_JoystickNumBalls(input->joystick);i++) //
        tant qu'il y a des trackballs
        {
            // on met à zéro
            input->trackballs[i]->xrel = 0;
            input->trackballs[i]->yrel = 0;
        }
    }

    else
    {
        // si le numéro du joystick n'était pas correct
        // on met tout à NULL
        input->joystick = NULL;
        input->boutons = NULL;
        input->axes = NULL;
        input->chapeaux = NULL;
        input->trackballs = NULL;
    }
}

```



J'ai utilisé *malloc()* puis une boucle *for* pour mettre à zéro, mais j'aurais pu utiliser la fonction *calloc()* qui alloue la mémoire et qui met tout à zéro par défaut dans les cases de mémoire allouées.

La fonction détruireInput()

Cette fonction a pour but de libérer la mémoire allouée ainsi que de fermer le joystick.

Code : C

```

void detruireInput(Input *input)
{
    if(input->joystick != NULL) // on vérifie que le joystick
    existe bien
    {
        input->numero = 0; // on le remet à zéro

        // on libère tout
        free(input->boutons);
        free(input->axes);
        free(input->chapeaux);
        free(input->trackballs);
        SDL_JoystickClose(input->joystick);
    }
}

```

La fonction updateEvent()

Maintenant on s'attaque à la fonction la plus importante qui récupère tous les événements du joystick : la fonction *updateEvent()*.

Elle est chargée de récupérer les événements qui se produisent, mais **uniquement** s'ils se produisent sur le joystick en question. Une condition s'effectue donc à chaque début de la boucle.

Les événements sont ensuite traités dans un switch comme ci-dessous :

Code : C

```

void updateEvent(Input *input)
{
    static SDL_Event evenements; // en statique car appelle
    plusieurs fois par seconde

    for(int i=0; i<SDL_JoystickNumBalls(input->joystick); i++)
    {
        // on met à zéro
        input->trackballs[i]->xrel = 0;
        input->trackballs[i]->yrel = 0;
    }

    while(SDL_PollEvent(&evenements)) // tant qu'il y a des
    évènements à traiter
    {
        if(input->joystick != NULL //si le
        joystick existe
        &&(evenements.jbutton.which == input->numero
        || evenements.jaxis.which == input->numero // et si c'est
        bien je joystick a gérer
        || evenements.jhat.which == input->numero
        || evenements.jball.which == input->numero))
        {
            switch(evenements.type)
            {
                case SDL_JOYBUTTONDOWN:
                    input->boutons[evenements.jbutton.button] = 1;
                    // bouton appuyé : valeur du bouton : 1
                    break;

                case SDL_JOYBUTTONUP:
                    input->boutons[evenements.jbutton.button] = 0;
                    // bouton relâché : valeur du bouton : 0
                    break;

                case SDL_JOYAXISMOTION:
                    input->axes[evenements.jaxis.axis] =
                    evenements.jaxis.value;
                    break;

                case SDL_JOYHATMOTION:
                    input->chapeaux[evenements.jhat.hat] =
                    evenements.jhat.value;
                    break;

                case SDL_JOYBALLMOTION:
                    input->trackballs[evenements.jball.ball]->xrel =
                    evenements.jball.xrel;
                    input->trackballs[evenements.jball.ball]->yrel =
                    evenements.jball.yrel;
                    break;

                default:
                    break;
            }
        }
    }
}

```

En résumé

Voici toute les fonctions précédentes dans un seul code :

Code : C

```
#include <sdtlib.h>
```

```

#include <SDL/SDL.h>

#include "Input.h" // mettez le nom du fichier où est la structure
et ses fonctions

void initialiserInput(Input *input,int numeroJoystick)
{
    if(numeroJoystick < SDL_NumJoysticks()) // on vérifie qu'il y a
    bien un bon numéro de joystick
    {
        input->joystick = SDL_JoystickOpen(numeroJoystick); // on
        met le joystick à numéro correspondant
        input->numero = numeroJoystick; // je pense que vous
        comprenez cette ligne...

        /* on alloue chaque éléments en fonctions de combien il y
        en a */
        input->boutons = (char*)
        malloc(SDL_JoystickNumButtons(input->joystick) * sizeof(char));
        input->axes = (int*) malloc(SDL_JoystickNumAxis(input->
        joystick) * sizeof(int));
        input->chapeaux = (int*) malloc(SDL_JoystickNumHats(input->
        joystick) * sizeof(int));
        input->trackballs = (InputTrackball*)
        malloc(SDL_JoystickNumBalls(input->joystick) *
        sizeof(InputTrackball));

        for(int i=0;i<SDL_JoystickNumButtons(input->joystick);i++)
        // tant qu'on a pas atteint le nombre max de boutons
        input->boutons[i] = 0; // on met les valeurs à 0
        for(int i=0;i<SDL_JoystickNumAxes(input->joystick);i++) //
        tant qu'on a pas atteint le nombre max d'axes
        input->axes[i] = 0; // on met aussi les valeurs à 0
        for(int i=0;i<SDL_JoystickNumHats(input->joystick);i++) //
        tant qu'on a pas atteint le nombre max de chapeaux
        input->chapeaux[i] = SDL_HAT_CENTERED; // on dit que les
        chapeaux son centrés
        for(int i=0;i<SDL_JoystickNumBalls(input->joystick);i++) //
        tant qu'il y a des trackballs
        {
            // on met à zéro
            input->trackballs[i]->xrel = 0;
            input->trackballs[i]->yrel = 0;
        }
    }

    else
    {
        // si le numéro du joystick n'était pas correct
        // on met tout à NULL
        input->joystick = NULL;
        input->boutons = NULL;
        input->axes = NULL;
        input->chapeaux = NULL;
        input->trackballs = NULL;
    }
}

void detruireInput(Input *input)
{
    if(input->joystick != NULL) // on vérifie que le joystick
    existe bien
    {
        input->numero = 0; // on le remet à zéro

        // on libère tout
        free(input->boutons);
        free(input->axes);
        free(input->chapeaux);
        free(input->trackballs);
        SDL_JoystickClose(input->joystick);
    }
}

```

```

    }
}

void updateEvent(Input *input)
{
    static SDL_Event evenements; // en statique car appelle
    plusieurs fois par seconde

    for(int i=0;i<SDL_JoystickNumBalls(input->joystick);i++)
    {
        // on met à zéro
        input->trackballs[i]->xrel = 0;
        input->trackballs[i]->yrel = 0;
    }

    while(SDL_PollEvent(&evenements)) // tant qu'il y a des
    évènements à traiter
    {
        if(input->joystick != NULL //si le
        joystick existe
        &&(evenements.jbutton.which == input->numero
        || evenements.jaxis.which == input->numero // et si c'est
        bien je joystick a gérer
        || evenements.jhat.which == input->numero
        || evenements.jball.which == input->numero))
        {
            switch(evenements.type)
            {
                case SDL_JOYBUTTONDOWN:
                    input->boutons[evenements.jbutton.button] = 1;
                    // bouton appuyé : valeur du bouton : 1
                    break;

                case SDL_JOYBUTTONUP:
                    input->boutons[evenements.jbutton.button] = 0;
                    // bouton relâché : valeur du bouton : 0
                    break;

                case SDL_JOYAXISMOTION:
                    input->axes[evenements.jaxis.axis] =
                    evenements.jaxis.value;
                    break;

                case SDL_JOYHATMOTION:
                    input->chapeaux[evenements.jhat.hat] =
                    evenements.jhat.value;
                    break;

                case SDL_JOYBALLMOTION
                    input->trackballs[evenements.jball.ball]->xrel =
                    evenements.jball.xrel;
                    input->trackballs[evenements.jball.ball]->yrel =
                    evenements.jball.yrel;
                    break;

                default:
                    break;
            }
        }
    }
}

```

Comment s'en servir ?

Maintenant, ~~nous allons~~ vous allez voir comment se servir de tout ça. 😊

Il faut initialiser la structure input.

Pour ce faire, nous allons tout d'abord remplacer ça :

Code : C

```
SDL_JoystickEventState(SDL_ENABLE);  
SDL_Event evenements;
```

Par ça :

Code : C

```
SDL_JoystickEventState(SDL_ENABLE); // on n'oublie pas d'activer les  
évènements du joystick !  
Input input; // on crée la structure  
initialiserInput(&input,0); // on l'initialise au joystick numéro 0
```

Il faut également détruire la structure à la fin du programme :

Code : C

```
détruireInput(&input); // on détruit à la fin du programme
```

Au début de chaque tours de boucle, il faudra également récupérer les évènements comme ceci :

Code : C

```
updateEvent(&input);
```

... et ainsi enlever ceci :

Code : C

```
SDL_PollEvent(&evenements);
```

Voilà ce que doit donner votre code désormais :

Code : C

```
#include <SDL.h>  
#include <SDL/SDL.h>  
  
#include "Event.h" // mettez le nom de votre header  
  
int main(int argc, char **argv)  
{  
    SDL_Init(SDL_INIT_VIDEO | SDL_INIT_JOYSTICK); // on initialise  
    les sous-programmes vidéo et joystick  
  
    SDL_JoystickEventState(SDL_ENABLE); // on active les évènements  
    du joystick  
    Input input; // on crée la structure  
    initialiserInput(&input,0); // on l'initialise au joystick n°0  
  
    int quitter = 0;  
  
    /* tout la blabla des fenêtre et des diverses variables ... */  
    while(!quitter)  
    {  
        updateEvent(&input); // on récupère les évènements  
  
        /* traitement des évènements (à suivre) ... */  
    }  
}
```

```
    }

    /* diverses destruction ...*/

    detruireInput(&input); // on détruit la structure input
    SDL_Quit(); // on quitte la SDL

    return 0;
}
```

Maintenant que tout est en place, on va voir comment gérer votre joystick.

Les boutons

Rien de plus simple : si le bouton est pressé, sa valeur est de 1 donc si vous faites la condition, ce sera vrai.

Exemple :

Code : C

```
if(input.boutons[0]) // si c'est égal à 1, c'est que le bouton est
appuyé, donc la condition est vraie
{
    // instructions...
}
```

Si vous voulez qu'une instruction ne soit effectuée qu'une seule fois quand on appuie sur un bouton, il suffit de mettre la valeur du bouton à zéro comme ceci :

Code : C

```
boutons[0] = 0;
```

Le joueur devra ainsi relâcher le bouton de re-appuyer pour régénérer un événement `SDL_JOYBUTTONDOWN`.

Les axes

On accède à la valeur de l'axe comme ceci `input.axes[numéro de l'axe]`.

Exemple :

Code : C

```
if(input.axes[0] > 2000) // si l'axe 0 a une inclinaison supérieure
à 2000
{
    // instructions...
}
```

Les chapeaux

Pour savoir dans quelle direction est un chapeau, on procède comme ceci :

`input.chapeaux[numéro du chapeau] == direction.`

Exemple :

Code : C

```
if(input.chapeaux[0] == SDL_HAT_UP) // si le chapeau n°0 est vers le
haut
{
    // instructions...

    input.chapeaux[0] = SDL_HAT_CENTERED; // pareille que pour les
```

```
boutons, mais on fait croire qu'il est retourné au centre
}
```

Les trackballs

Pour les trackballs, on procède de cette façon :

input.trackballs[numéro du trackball].xrel **ou** yrel.

Exemple :

Code : C

```
if(input.trackballs[0].xrel > 0) // si le trackball n°0 est vers la
droite (valeur positive)
{
    // instructions...
}
```

Améliorations

Comme amélioration, vous pourriez ~~en complexifiant énormément le code~~ réussir à gérer plusieurs joysticks avec une seule structure Input, gérant également la souris et le clavier.

Bon, je pense que vous en avez pour un bon bout de temps...

Correction



Le code ci-dessous ne gère pas les trackballs.

Voici mes 2 fichiers :

- Le fichier "Event.h" :

Secret (cliquez pour afficher)

Code : C++

```
#ifndef EVENT_H_INCLUDED
#define EVENT_H_INCLUDED

#include <SDL/SDL.h>

/* structure qui gère un joystick */
typedef struct InputJoystick InputJoystick;
struct InputJoystick
{
    SDL_Joystick *joystick;
    char *boutons;
    int *axes;
    int *chapeaux;
    int numero;
};

/* structure qui gère une souris */
typedef struct InputSouris InputSouris;
struct InputSouris
{
    int *boutons;

    int x;
    int y;
    int xrel;
    int yrel;
}
```

```

};

/* la structure qui gère les événements */
typedef struct Input Input;
struct Input
{
    char *touches; // on alloue les touches par la suite ...

    int quitter; // il est géré automatiquement mais peut
    être demandé à être mis à 1

    InputSouris souris; // une souris
    InputJoystick *joystick; // pointeur car il peut y avoir
    plusieurs joysticks

    int nombreJoysticks; // sert à détruire à la fin
};

void initialiserInput(Input *input, int utiliserJoystick1, int
numeroJoystick1, int utiliserJoystick2, int numeroJoystick2);
void detruireInput(Input *input); // fonction qui libère la
mémoire allouée
void updateEvent(Input *input); // fonction qui récupère les
événements

#endif // EVENT_H_INCLUDED

```

- Et le fichier "Event.cpp" :

Secret ([cliquez pour afficher](#))

Code : C++

```

#include <stdlib.h>
#include <SDL/SDL.h>

#include "Event.h"

void initialiserInput(Input *input, int utiliserJoystick1, int
numeroJoystick1, int utiliserJoystick2, int numeroJoystick2)
{
    input->nombreJoysticks = 0;

    if(utiliserJoystick1 == 1 && SDL_NumJoysticks() > numeroJoystick1)
        input->nombreJoysticks ++;

    if(utiliserJoystick2 == 1 && numeroJoystick2 != numeroJoystick1 &&
SDL_NumJoysticks() > numeroJoystick2)
        input->nombreJoysticks ++;

    /* si il y a des joystick, on les alloue et initialise */
    if(input->nombreJoysticks != 0)
    {
        SDL_JoystickEventState(SDL_ENABLE);

        input->joystick = (InputJoystick*) malloc(input->
nombreJoysticks * sizeof(InputJoystick));

        for(int i=0; i<input->nombreJoysticks; i++)
        {
            input->joystick[i].joystick = SDL_JoystickOpen((i == 0) ?
(numeroJoystick1) : (numeroJoystick2));
            input->joystick[i].numero = (i == 0) ? (numeroJoystick1) :
(numeroJoystick2);

            input->joystick[i].boutons = (char*)
malloc(SDL_JoystickNumButtons(input->joystick[i].joystick) *

```



```

    sizeof(char));
    input->joystick[i].axes = (int*)
malloc(SDL_JoystickNumAxes(input->joystick[i].joystick) *
sizeof(int));
    input->joystick[i].chapeaux = (int*)
malloc(SDL_JoystickNumHats(input->joystick[i].joystick) *
sizeof(int));

    for(int j=0;j<SDL_JoystickNumButtons(input-
>joystick[i].joystick);j++)
        input->joystick[i].boutons[j] = 0;

    for(int j=0;j<SDL_JoystickNumAxes(input-
>joystick[i].joystick);j++)
        input->joystick[i].axes[j] = 0;

    for(int j=0;j<SDL_JoystickNumHats(input-
>joystick[i].joystick);j++)
        input->joystick[i].chapeaux[j] = 0;
    }
}

else
{
    input->joystick = NULL;
}

input->touches = (char*) malloc(SDLK_LAST * sizeof(char)); //
allocation des touches

/* On met les touches à zéro */
for(int i=0;i<SDLK_LAST;i++)
    input->touches[i] = 0;

input->souris.x = 0;
input->souris.y = 0;
input->souris.xrel = 0;
input->souris.yrel = 0;

input->souris.boutons = (int*) malloc(8 * sizeof(int)); //
allocation des boutons de la souris

/* On met les boutons de la souris à zéro */
for(int i=0;i<8;i++)
    input->souris.boutons[i] = 0;

input->quitter = 0;
}

void detruireInput(Input *input)
{
    if(input->joystick != NULL)
    {
        SDL_JoystickEventState(SDL_DISABLE); // ignorer les événements
du joystick

        for(int i=0;i<input->nombreJoysticks;i++) // tant qu'un
joystick existe
        {
            input->joystick[i].numero = 0;

            // on libère la mémoire
            free(input->joystick[i].boutons);
            free(input->joystick[i].axes);
            free(input->joystick[i].chapeaux);
            SDL_JoystickClose(input->joystick[i].joystick);
        }

        free(input->joystick);
        input->joystick = NULL;
    }
}

```

```

    }

    if(input->souris.boutons != NULL) // si les boutons de la souris
    sont alloués
    {
        free(input->souris.boutons); // on libère la mémoire
        input->souris.boutons = NULL;
    }

    if(input->touches != NULL) // si les touches du clavier sont
    allouées
    {
        free(input->touches); // on libère la mémoire
        input->touches = NULL;
    }
}

void updateEvent(Input *input)
{
    static SDL_Event evenements; // en statique car appelle plusieurs
    fois par seconde

    input->souris.xrel = 0;
    input->souris.yrel = 0;

    input->souris.boutons[SDL_BUTTON_WHEELDOWN] = 0;
    input->souris.boutons[SDL_BUTTON_WHEELUP] = 0;

    if(input->nombreJoysticks != 0) // on ne vérifie qu'une seule
    fois qu'un joystick existe
    {
        while(SDL_PollEvent(&evenements))
        {
            switch(evenements.type)
            {
                case SDL_KEYDOWN:
                    input->touches[evenements.key.keysym.sym] = 1;
                    break;

                case SDL_KEYUP:
                    input->touches[evenements.key.keysym.sym] = 0;
                    break;

                case SDL_MOUSEMOTION:
                    input->souris.x = evenements.motion.x;
                    input->souris.y = evenements.motion.y;

                    input->souris.xrel = evenements.motion.xrel;
                    input->souris.yrel = evenements.motion.yrel;
                    break;

                case SDL_MOUSEBUTTONDOWN:
                    input->souris.boutons[evenements.button.button] =
1;

                    break;

                case SDL_MOUSEBUTTONUP:
                    if(evenements.button.button !=
SDL_BUTTON_WHEELDOWN && evenements.button.button !=
SDL_BUTTON_WHEELUP)
                        input->
souris.boutons[evenements.button.button] = 0;
                    break;

                case SDL_WINDOWEVENT:
                    if(evenements.window.event ==
SDL_WINDOWEVENT_CLOSE) // si on clique sur "fermer"

```

```

        input->quitter = 1; // on quitte
        break;

    default:
        break;
}

/* on vérifie qu'un des joysticks est concerné */
if(input->joystick[0].numero == evenements.jbutton.which
|| input->joystick[0].numero == evenements.jaxis.which
|| input->joystick[0].numero == evenements.jhat.which
|| input->joystick[1].numero == evenements.jbutton.which
|| input->joystick[1].numero == evenements.jaxis.which
|| input->joystick[1].numero == evenements.jhat.which)
{
    switch(evenements.type)
    {
        case SDL_JOYBUTTONDOWN:
            input-
>joystick[evenements.jbutton.which].boutons[evenements.jbutton.button]
= 1;

            break;

        case SDL_JOYBUTTONUP:
            input-
>joystick[evenements.jbutton.which].boutons[evenements.jbutton.button]
= 0;

            break;

        case SDL_JOYAXISMOTION:
            input-
>joystick[evenements.jaxis.which].axes[evenements.jaxis.axis] =
evenements.jaxis.value;
            break;

        case SDL_JOYHATMOTION:
            input-
>joystick[evenements.jhat.which].chapeaux[evenements.jhat.hat] =
evenements.jhat.value;
            break;

        default:
            break;
    }
}

}

else // si aucun joystick n'existe, on traite les événements
normalement
{
    while(SDL_PollEvent(&evenements))
    {
        switch(evenements.type)
        {
            case SDL_KEYDOWN:
                input->touches[evenements.key.keysym.sym] = 1;
                break;

            case SDL_KEYUP:
                input->touches[evenements.key.keysym.sym] = 0;
                break;

            case SDL_MOUSEMOTION:
                input->souris.x = evenements.motion.x;
                input->souris.y = evenements.motion.y;

                input->souris.xrel = evenements.motion.xrel;
                input->souris.yrel = evenements.motion.yrel;
                break;
        }
    }
}

```

```

        case SDL_MOUSEBUTTONDOWN:
            input->souris.boutons[evenements.button.button] =
1;
            break;

        case SDL_MOUSEBUTTONUP:
            if(evenements.button.button !=
SDL_BUTTON_WHEELDOWN && evenements.button.button !=
SDL_BUTTON_WHEELUP)
                input-
>souris.boutons[evenements.button.button] = 0;
            break;

        case SDL_WINDOWEVENT:
            if(evenements.window.event ==
SDL_WINDOWEVENT_CLOSE) // si on clique sur "fermer"
                input->quitter = 1; // on quitte
            break;

        default:
            break;
    }
}
}

```

Je pense qu'en regardant les structures et les fonctions vous devinerez comment se servir cette nouvelle structure...

Haha ! Plus sérieusement, je ne pense pas que vous voudriez lire tout ce code, alors vous n'avez qu'à copier/coller le code dans deux fichiers et lire ce qui suit. 🤖

Comment gérer tout ça ?

Pour commencer, on crée la structure Input et on l'initialise :

Code : C++

```

Input input;
initialiserInput(&input, 1, 0, 1, 0); // on utilise les joysticks n°0 et
n°1

```



La fonction **initialiserInput()** active les évènements du joysticks. C'est bien pratique car ça fait une ligne en moins à taper. 😊

Voici donc le prototype de **initialiserInput()** :

Code : C++

```

void initialiserInput(Input *input, int utiliserJoystick1, int
numeroJoystick1, int utiliserJoystick2, int numeroJoystick2);

```

Pour faire en sorte que le programme n'utilise pas les joysticks, il faut faire ceci :

Code : C

```

initialiserInput(&input, 0, 0, 0, 0); // initialisation "par défaut",
avec seulement le clavier et la souris

```

Vous devrez également détruire la structure à la fin du programme comme ceci (pas de changement) :

Code : C++

```
détruireInput(&input);
```

N'oublions pas notre fonction **updateEvent()** dont l'appel ne change pas. Avec la nouvelle gestion, lorsqu'on clique sur "fermer" (la croix), une variable `input.quitter` se met à 1, alors pourquoi ne pas s'en servir ?

Code : C++

```
while(!input.quitter) // tant que la variable input.quitter n'est
pas égale à 1...
{
    updateEvent(&input); // si on clique sur la croix, la variable
input.quitter se met à 1

    // blabla...
}
```

En résumé, voilà à quoi devrait ressembler votre code :

Code : C++

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

int main(int argc, char *argv[])
{
    /* déclaration de diverses variables ... */

    SDL_Init(SDL_INIT_VIDEO | SDL_INIT_JOYSTICK); // on met les
modes vidéo et joystick

    Input input;
    initialiserInput(&input); // on utilise la "configuration de
base"

    /* d'autres instructions ... */

    while(!input.quitter) // tant qu'on n'a pas quitté
    {
        updateEvent(&input); // on récupère les événements

        /* gestion des événements (à suivre) ... */

        /* d'autres instructions ... */

    }

    /* diverses destructions ... */

    détruireInput(&input); // on dés-alloue la mémoire
    SDL_Quit(); // on quitte la SDL

    return 0;
}
```

On est (enfin) prêt à utiliser la nouvelle structure. 😊

Le clavier

On récupère la valeur d'une touche comme ceci : `input.touches[nom de la touche]`.
Et donc :

Code : C++

```

if(input.touches[SDLK_UP]) // si on appuie sur la flèche du haut...
{
    // instructions ...
}

```

Comme vous le voyez, les conditions pour l'appui des touches sont beaucoup plus courtes que si on le faisait à la manière "habituelle". 😊

La souris

Toutes les informations de la souris se trouvent dans `input.souris`.

Exemple :

Code : C++

```

if(input.souris.boutons[2]) // si on appuie sur le bouton 2 de la
souris ...
{
    // instructions ...
}

```

Le(s) joystick(s)

On accède aux informations sur les joysticks comme ceci :

`input.joystick[numéro du joystick]`

Ensuite c'est comme précédemment : pas de changement.

Un exemple est tout de même de mise ici :

Code : C++

```

if(input.joystick[0].hat[0] == SDL_HAT_UP) // si la chapeau numéro 0
du joystick numéro 0 est vers le haut...
{
    // instructions ...
}

```

Le joystick numéro 0 est le premier **dans le tableau**, c'est-à-dire le premier numéro de joystick passé en argument à la fonction `initialiserInput()`.

Exemple :

Code : C++

```

initialiserInput(&input, 1, 1, 1, 0);

```



Dans ce cas :

- Quand on va chercher des informations sur le joystick 0 par l'intermédiaire de la structure `input`, il faut en réalité chercher le 1 dans le tableau ;
- Quand on va chercher des informations sur le joystick 1 par l'intermédiaire de la structure `input`, il faut en réalité chercher le 0 dans le tableau .

Tout ça pour dire qu'il est judicieux de mettre les numéros dans un ordre facile à retenir et ne pas tout inverser. 😊

Améliorations, encore et toujours !

Mon code est loin d'être parfait. Vous pourriez encore l'améliorer :

- ajouter la gestion des trackballs ;
- gérer plus de joysticks en faisant choisir à l'utilisateur combien il en utilise et tout faire passer dans un tableau en paramètres de la fonction ***initialiserInput()***, le prototype de la fonction ressemblerais à ceci :

Code : C

```
void initialiserInput(Input *input, int nombreJoysticks, int
numerosJoysticks[]);
```

Allez hop, hop, hop, vous avez du boulot !

Vous avez intérêt à très bien connaître le fonctionnement de la gestion avancée du joystick, car les questions du *Q.C.M.* ne portent pas dessus. 😊

Q.C.M.

Le premier QCM de ce cours vous est offert en libre accès.
Pour accéder aux suivants

[Connectez-vous](#) [Inscrivez-vous](#)

Quel flag faut-il passer à ***SDL_Init*** pour activer la gestion du joystick ?

- ☐ SDL_INIT_JOYSTICKS
- ☐ SDL_INIT_JOYSTICK
- ☐ SDL_INIT_VIDEO
- ☐ Rien du tout, c'est par défaut.

Ce code est-il correct ?

Code : C++

```
SDL_Joystick joystick;
joystick = SDL_JoystickOpen(0);
```

- ☐ Oui
- ☐ Non

Que fait ce code ?

Code : C++

```
switch(evenements.type)
{
    case SDL_JOYAXISMOTION:
        if(evenements.jaxis.axis == 0 && evenements.jaxis.value >
2000)
            quitter = 1;
        break;
}
```

- ☐ Il s'arrete si on bouge l'axe 0 à droite.
- ☐ Il s'arrete si on bouge l'axe 0 à gauche.
- ☐ Il ne s'arrete pas !

Quelle est la direction du chapeau si celui-ci a une valeur égale à ***SDL_HAT_RIGHTUP*** ?

- ☐ à gauche
- ☐ à droite
- ☐ en haut à droite

- ☐ en haut à gauche
- ☐ en bas

Que se passe-t-il ?

Code : C

```
switch(eventements.type)
{
    case SDL_JOYBALLMOTION:
        quitter = 1;
        break;
}
```

- ☐ Le programme s'arrête instantanément.
- ☐ Le programme s'arrête lorsqu'un mouvement sur un axe se produit.
- ☐ Le programme s'arrête lorsqu'un mouvement d'un trackball se produit.
- ☐ Le programme ne s'arrête pas !

Où se situent les informations sur les événements des trackballs ?

- ☐ eventements.jball
- ☐ eventements.ball
- ☐ eventements.jaxis
- ☐ eventements.axis

Quel fonction faut-il appeler pour détruire une structure joystick ?

- ☐ SDL_JoysticksClose();
- ☐ SDL_JoystickClose();
- ☐ JoystickClose();

Correction !

Statistiques de réponses au QCM

Voilà, vous savez maintenant gérer un joystick sans difficultés.

La gestion avancée est très utile : si vous n'avez pas lu les dernières parties, je vous conseille d'y jeter un œil. Mieux vaut tard que jamais ! 😊



Si vous pensez que des éléments doivent être développés davantage dans ce tutoriel, merci de m'envoyer un MP.

Partager

