

Les événements avec la SDL 2.0

Par [Alexandre Laurent](#) 

Date de publication : 22 avril 2014

DÉBUTANT

Dans le **tutoriel précédent**, vous avez appris à afficher votre premier sprite. Il est temps d'apprendre à le déplacer !

Navigation.....	4
I - Introduction.....	4
II - Les événements.....	4
II-A - Le type SDL_Event.....	4
II-A-1 - Récupérer les événements.....	7
II-A-1-a - SDL_WaitEvent().....	7
II-A-1-a-i - Le paramètre de SDL_WaitEvent().....	7
II-A-1-a-ii - La valeur de retour de SDL_WaitEvent().....	7
II-A-1-b - SDL_WaitEventTimeout().....	7
II-A-1-b-i - Les paramètres de SDL_WaitEventTimeout().....	7
II-A-1-b-ii - La valeur de retour de SDL_WaitEventTimeout().....	7
II-A-1-c - SDL_PollEvent().....	8
II-A-1-c-i - Le paramètre de SDL_PollEvent().....	8
II-A-1-c-ii - La valeur de retour de SDL_PollEvent().....	8
II-A-1-d - Conseils d'utilisation.....	8
II-A-1-e - Remplissage de la queue d'événements.....	9
II-A-2 - Analyse des événements.....	9
II-A-2-a - Événement de la fenêtre.....	9
II-A-2-b - Événement du clavier.....	10
II-A-2-b-i - SDL_Keysym.....	10
II-A-2-c - Événements de la souris.....	11
II-A-2-c-i - Déplacement de la souris.....	11
II-A-2-c-ii - Appui sur les boutons de la souris.....	11
II-A-2-c-iii - Utilisation de la roulette.....	12
II-A-2-d - Événement de joystick.....	12
II-A-2-d-i - Connexion/Déconnexion de joystick.....	12
II-A-2-d-ii - Déplacement d'un joystick.....	12
II-A-2-d-iii - Appui d'un bouton de joystick.....	12
II-A-2-d-iv - Déplacement de trackball d'un joystick.....	13
II-A-2-d-v - Déplacement d'un chapeau d'un joystick.....	13
II-B - Traitement de SDL_Event.....	13
II-C - Code d'exemple.....	14
III - Statut des périphériques.....	17
III-A - Fonctionnement.....	17
III-B - Les informations de la souris.....	17
III-B-1 - SDL_GetMouseState().....	17
III-B-1-a - Les paramètres de SDL_GetMouseState().....	17
III-B-1-b - La valeur de retour de SDL_GetMouseState().....	17
III-B-1-b-i - La macro SDL_BUTTON.....	18
III-B-2 - SDL_GetRelativeMouseState().....	18
III-C - Les informations du clavier.....	18
III-C-1 - SDL_GetKeyboardState().....	18
III-C-1-a - Le paramètre de SDL_GetKeyboardState().....	18
III-C-1-a-i - La valeur de retour de SDL_GetKeyboardState().....	18
III-C-2 - SDL_GetModState().....	19
III-C-2-a - La valeur de retour de SDL_GetModState().....	19
III-C-3 - Les informations des joysticks.....	19
III-C-3-a - SDL_NumJoysticks().....	19
III-C-3-a-i - La valeur de retour de SDL_NumJoysticks().....	19
III-C-3-b - SDL_JoystickOpen().....	20
III-C-3-b-i - Le paramètre de SDL_JoystickOpen().....	20
III-C-3-b-ii - La valeur de retour de SDL_JoystickOpen().....	20
III-C-3-c - SDL_JoystickName().....	20
III-C-3-c-i - Le paramètre de SDL_JoystickName().....	20
III-C-3-c-ii - La valeur de retour de SDL_JoystickName().....	20
III-C-3-d - SDL_JoystickNumAxes().....	20
III-C-3-d-i - Le paramètre de SDL_JoystickNumAxes().....	20
III-C-3-d-ii - La valeur de retour de SDL_JoystickNumAxes().....	20

III-C-3-e - SDL_JoystickNumButtons()	20
III-C-3-e-i - Le paramètre de SDL_JoystickNumButtons()	21
III-C-3-e-ii - La valeur de retour de SDL_JoystickNumButtons()	21
III-C-3-f - SDL_JoystickNumHats()	21
III-C-3-f-i - Le paramètre de SDL_JoystickNumHats()	21
III-C-3-f-ii - La valeur de retour de SDL_JoystickNumHats()	21
III-C-3-g - SDL_JoystickNumBalls()	21
III-C-3-g-i - Le paramètre de SDL_JoystickNumBalls()	21
III-C-3-g-ii - La valeur de retour de SDL_JoystickNumBalls()	21
III-C-3-h - SDL_JoystickGetAxis()	21
III-C-3-h-i - Les paramètres de SDL_JoystickGetAxis()	21
III-C-3-h-ii - La valeur de retour de SDL_JoystickGetAxis()	22
III-C-3-i - SDL_JoystickGetButton()	22
III-C-3-i-i - Les paramètres de SDL_JoystickGetButton()	22
III-C-3-i-ii - La valeur de retour de SDL_JoystickGetButton()	22
III-C-3-j - SDL_JoystickGetHat()	22
III-C-3-j-i - Les paramètres de SDL_JoystickGetHat()	22
III-C-3-j-ii - La valeur de retour de SDL_JoystickGetHat()	22
III-C-3-k - SDL_JoystickGetBall()	22
III-C-3-k-i - Les paramètres de SDL_JoystickGetBall()	23
III-C-3-k-ii - La valeur de retour de SDL_JoystickGetBall()	23
III-C-3-l - SDL_JoystickUpdate()	23
III-C-3-m - SDL_JoystickEventState()	23
III-C-3-m-i - Le paramètre de SDL_JoystickEventState()	23
III-C-3-m-ii - La valeur de retour de SDL_JoystickEventState()	23
III-C-3-n - SDL_JoystickClose()	23
III-C-3-n-i - Le paramètre de SDL_JoystickClose()	23
III-C-4 - Code d'exemple	24
IV - Filtrer des événements	26
IV-A - SDL_SetEventFilter()	26
IV-A-1 - Les paramètres de SDL_SetEventFilter()	27
IV-B - SDL_EventState()	27
IV-C - Les paramètres de SDL_EventState()	27
IV-D - La valeur de retour de SDL_EventState()	28
V - Conclusion	28
VI - Remerciements	28
Navigation	28

Navigation

Tutoriel
précédent :
**[afficher son
premier
sprite](#)**

Sommaire

I - Introduction

Maintenant que vous savez afficher des sprites, il est temps de voir comment le joueur, en appuyant sur son clavier ou sur sa manette, déplace ce sprite.

Dans la SDL 2, les fonctionnalités liées aux événements (que ce soit du clavier ou d'une manette de jeu) n'ont subi que très peu de modification. Vous êtes invités à lire le **[guide officiel de migration SDL 1.2 vers SDL 2](#)**. Ainsi, les connaisseurs de la SDL 1.2 ne seront pas dépayés.

II - Les événements

Sous le nom d'événement on regroupera toutes les actions qui peuvent influencer sur votre programme. On appellera aussi bien l'appui sur une touche du clavier, l'utilisation d'un joystick de la manette, le redimensionnement de la fenêtre, le déplacement de la souris, des événements.

Votre application reçoit de nombreux événements qui sont stockés dans une queue en attente de traitement par votre code. Vous pouvez imaginer cette queue, comme une file d'attente dans un parc d'attractions. Le premier arrivé sera le premier à entrer dans l'attraction. Lorsque vous lisez un événement, c'est donc le premier événement reçu (et donc le plus vieux) qui est lu et retiré.

Vous pouvez lire cette queue avec les fonctions **[SDL_WaitEvent\(\)](#)** ou **[SDL_PollEvent\(\)](#)** ce qui entraînera la suppression de l'événement de la queue (si on reprend l'analogie ci-dessus, la personne qui attendait est donc entrée dans l'attraction, donc elle n'est plus dans la file d'attente). Une fois l'événement retiré de la queue, vous pouvez soit l'analyser et agir suivant son type et les informations associés, soit l'ignorer.

La SDL utilise la structure **[SDL_Event](#)** pour stocker les informations relatives à chaque événement. C'est à travers cette structure que vous pourrez récupérer les informations de chaque événement.

De plus, si vous ne souhaitez pas utiliser la queue d'événements, la SDL permet de connaître l'état de vos périphériques. Ainsi, si vous avez besoin de savoir si tel ou tel bouton est appuyé ou la position de la souris, vous pouvez demander à la bibliothèque de vous retourner les informations voulues. Ce second mécanisme est détaillé dans la section **[Statut des périphériques](#)**.

II-A - Le type `SDL_Event`

La structure `SDL_Event` est un conteneur global pour les événements. Dans la SDL, il existe plusieurs types d'événement et chacun peut contenir des informations différentes. En effet, si la SDL doit décrire un clic de souris, alors l'événement doit contenir des informations décrivant ce clic (quel bouton, quelle souris...), mais si la bibliothèque doit décrire un appui sur une touche, alors l'événement doit contenir des informations sur cette touche (quel touche est appuyée...). Que ce soit pour les événements de la souris, ou pour tout autre événement, la SDL utilise la structure `SDL_Event`. Pour ce faire, la structure possède un champ par type d'événement et un champ indiquant le type d'événement reçu. Ainsi, lorsque vous avez un `SDL_Event`, vous devez lire le champ `type` pour savoir quel champ de la structure contient les informations appropriées.

i Chaque champ de `SDL_Event` décrivant un événement est lui-même une structure. En effet, pour décrire complètement un événement, il est nécessaire d'avoir plusieurs données et donc, d'utiliser une structure pour les conserver.


Les types disponibles sont les suivants :

Valeur de type	Structure utilisée pour l'événement	Champ	Description
<code>SDL_CLIPBOARDUPDATE</code>	Aucune structure utilisée pour cet événement.		Mise à jour du presse-papier.
<code>SDL_CONTROLLERAXISMOTION</code>	<code>SDL_ControllerAxisEvent</code>	<code>caxis</code>	Données de déplacement d'un axe d'un contrôleur de jeu.
<code>SDL_CONTROLLERBUTTONDOWN</code> <code>SDL_CONTROLLERBUTTONUP</code>	<code>SDL_ControllerButtonEvent</code>	<code>cbutton</code>	Données d'appui ou de relâchement d'un bouton d'un contrôleur de jeu.
<code>SDL_CONTROLLERDEVICEADDED</code> <code>SDL_CONTROLLERDEVICEREMOVED</code> <code>SDL_CONTROLLERDEVICEREMAPPED</code>	<code>SDL_ControllerDeviceEvent</code>	<code>cdevice</code>	Données de connexion/déconnexion/changement de configuration d'un contrôleur de jeu.
<code>SDL_DOLLARGESTURE</code>	<code>SDL_DollarGestureEvent</code>	<code>dgesture</code>	Données de mouvement sur support tactile.
<code>SDL_DROPFILE</code>	<code>SDL_DropEvent</code>	<code>drop</code>	Données du dépôt d'un fichier dans l'application.
<code>SDL_FINGERDOWN</code> <code>SDL_FINGERUP</code> <code>SDL_FINGERMOTION</code>	<code>SDL_TouchFingerEvent</code>	<code>tfinger</code>	Données de déplacement, appui et

			relâchement sur le périphérique tactile.
SDL_KEYDOWN SDL_KEYUP	SDL_KeyboardEvent	key	Données d'appui ou relâchement d'une touche du clavier.
SDL_JOYAXISMOTION	SDL_JoyAxisEvent	jaxis	Données d'axe d'une manette.
SDL_JOYBALLMOTION	SDL_JoyBallEvent	jball	Données de trackball d'une manette.
SDL_JOYHATMOTION	SDL_JoyHatEvent	jhat	Données du chapeau d'une manette.
SDL_JOYBUTTONDOWN SDL_JOYBUTTONUP	SDL_JoyButtonEvent	jbutton	Données d'appui ou relâchement d'une manette.
SDL_MOUSEMOTION	SDL_MouseMotionEvent	motion	Données de déplacement de la souris.
SDL_MOUSEBUTTONDOWN SDL_MOUSEBUTTONUP	SDL_MouseButtonEvent	button	Données d'appui ou relâchement d'un bouton de la souris.
SDL_MOUSEWHEEL	SDL_MouseWheelEvent	wheel	Données de la roulette de la souris.
SDL_MULTIGESTURE	SDL_MultiGestureEvent	mgesture	Données de touché multiples.
SDL_QUIT	SDL_QuitEvent	quit	Données de demande de fermeture de l'application.
SDL_SYSWMEVENT	SDL_SysWMEvent	syswm	Données dépendantes du gestionnaire de fenêtres.
SDL_TEXTEDITING	SDL_TextEditingEvent	edit	Données de l'édition de texte.
SDL_TEXTINPUT	SDL_TextInputEvent	text	Données de l'entrée d'un texte.
SDL_USEREVENT	SDL_UserEvent	user	Données fournies par l'utilisateur.
SDL_WINDOWEVENT	SDL_WindowEvent	window	Données de l'événement de la fenêtre.

De plus, il existe six événements spécifiques aux appareils mobiles :


Valeur de type	Description
SDL_APP_TERMINATING	Le système termine l'application.
SDL_APP_LOWMEMORY	Le système manque de mémoire ; libérez-en.
SDL_APP_WILLENTERBACKGROUND	L'application est envoyée en tâche de fond.
SDL_APP_DIDENTERBACKGROUND	L'application est en tâche de fond.
SDL_APP_WILLENTERFOREGROUND	L'application est envoyée en premier plan.
SDL_APP_DIDENTERFOREGROUND	L'application est en premier plan.

 Ces événements doivent être traités en priorité sinon le système tue votre application. Pour cela, utilisez un **filtre d'événement**.

II-A-1 - Récupérer les événements

Pour récupérer un `SDL_Event` de la queue, trois fonctions sont proposées :

- `SDL_WaitEvent()` ;
- `SDL_WaitEventTimeout()` ;
- `SDL_PollEvent()`.

 Chacune de ces fonctions récupèrent le plus vieux éléments présents dans la queue d'événements.

II-A-1-a - `SDL_WaitEvent()`

`SDL_WaitEvent()` est une fonction bloquante. Cela signifie que lorsque vous l'utilisez, votre programme (du moins, votre thread courant) est bloqué jusqu'à ce que la fonction finisse sa tâche. Cela veut dire que s'il n'y a aucun événement disponible, la fonction attend et ne vous redonnera la main qu'après l'arrivée d'un événement dans la queue.

Son utilisation est simple. Elle prend un pointeur sur un `SDL_Event` afin de le remplir avec l'événement récupéré :

```
SDL_Event ev;  
SDL_WaitEvent(&ev);  
// Ici, vous devez analyser le contenu de ev
```

II-A-1-a-i - Le paramètre de `SDL_WaitEvent()`

Un pointeur sur `SDL_Event`. La variable `SDL_Event` pointée sera remplie avec l'événement récupéré de la queue.

II-A-1-a-ii - La valeur de retour de `SDL_WaitEvent()`

La fonction retourne zéro si une erreur a eu lieu. Vous pouvez utiliser `SDL_GetError()` pour plus d'informations sur l'erreur.

II-A-1-b - `SDL_WaitEventTimeout()`

`SDL_WaitEventTimeout()` est similaire à `SDL_WaitEvent()` et bloquera aussi l'exécution de votre programme. Toutefois, avec cette fonction vous pouvez spécifier un temps au bout duquel la fonction vous rendra la main, même si aucun événement n'est arrivé.

II-A-1-b-i - Les paramètres de `SDL_WaitEventTimeout()`

Le premier paramètre est un pointeur sur `SDL_Event`. La variable `SDL_Event` pointée sera remplie avec l'événement récupéré de la queue.

Le second paramètre est un entier représentant un temps en milliseconde au bout duquel la fonction rendra la main au programme et cela même s'il n'y a pas d'événement.

II-A-1-b-ii - La valeur de retour de `SDL_WaitEventTimeout()`

La fonction retourne zéro si une erreur a eu lieu ou si le temps passé en paramètre s'est écoulé. Vous pouvez utiliser `SDL_GetError()` pour plus d'informations sur l'erreur. La fonction retourne 1, lorsqu'un événement a été récupéré.

II-A-1-c - SDL_PollEvent()

La fonction `SDL_PollEvent()` effectue la même chose que les fonctions `SDL_WaitEvent*()`. Contrairement aux autres fonctions `SDL_WaitEvent*()`, `SDL_PollEvent()` n'est pas bloquante et ne bloquera donc pas votre programme même lorsqu'il n'y a aucun événement à récupérer.

```
SDL_Event ev;
if ( SDL_PollEvent(&ev) == 1 )
{
    // Analyse de l'événement
}
```

II-A-1-c-i - Le paramètre de SDL_PollEvent()

Un pointeur sur `SDL_Event`. La variable `SDL_Event` pointée sera remplie avec l'événement récupéré de la queue.

II-A-1-c-ii - La valeur de retour de SDL_PollEvent()

La fonction retourne 1, si un événement a été retourné, zéro s'il n'y en a pas.

II-A-1-d - Conseils d'utilisation

En théorie, les fonctions `SDL_WaitEvent*()` ne sont pas très intéressantes car l'exécution du programme sera bloquée (ou, si ce n'est pas du programme, c'est de la boucle principale). La solution dans laquelle vous utiliseriez un thread n'est pas valable car cette fonction doit être appelée dans le thread initialisant le mode vidéo. Pour des raisons de facilité, il est donc préférable d'utiliser `SDL_PollEvent()`.

Imaginons la boucle principale de votre programme comme étant la suivante :

```
while (1) // Tant que l'on veut que le jeu fonctionne
{
    // Ici, nous devrions traiter les événements
    afficheMonde();
    deplacementMonde(); // On gère ici, tout ce qui est de la mise à jour du monde
}
```

Si nous utilisions `SDL_WaitEvent()`, il faudrait attendre un événement de l'utilisateur pour pouvoir afficher notre monde (ou la prochaine image de notre monde).

Il serait possible d'utiliser `SDL_WaitEventTimeout()` et d'indiquer un temps court, afin d'afficher et de mettre à jour le monde même s'il n'y a pas d'événement. Si vous essayez et que l'utilisateur effectue énormément d'événements, vous n'allez traiter qu'un seul événement par tour de boucle.

Il faut donc insérer `SDL_WaitEventTimeout()`, ou `SDL_PollEvent()`, dans une boucle afin de traiter tous les événements présents dans la queue. Ainsi, nous sommes certains de gérer tout ce que l'utilisateur a voulu faire avant d'afficher et de mettre à jour le monde.



La solution avec `SDL_WaitEventTimeout()`, n'est pas parfaite une fois dans une boucle.

En effet, si nous plaçons `SDL_WaitEventTimeout()` dans une boucle comme suit :

```
while ( SDL_WaitEventTimeout(&ev,16) ) // Nous traitons les événements de la queue
{
    // Traitement de l'événement
}
```



il serait possible de complètement bloquer (freeze) le programme. Cela se produirait si le programme recevait un événement toutes les N millisecondes, où N est inférieur au temps passé à la fonction `SDL_WaitEventTimeout()`. À chaque fois qu'un événement est reçu, le temps est remis à zéro et le programme doit attendre 16 millisecondes (pour ce cas) avant de retourner 0 (et de provoquer la sortie de la boucle).

Si vous pensez que cela arrive rarement, testez par vous-même avec cette boucle et pendant l'exécution du programme bougez rapidement et en continu la souris.

Donc, nous utilisons `SDL_PollEvent()`. Voici le code avec `SDL_PollEvent()` :

```
while (1) // Tant que l'on veut que le jeu fonctionne
{
    SDL_Event ev;
    while ( SDL_PollEvent(&ev) ) // Nous traitons les événements de la queue
    {
        // Traitement de 'ev' (nous allons voir après comment traiter un SDL_Event)
    }

    afficheMonde();
    deplacementMonde(); // On gère ici, tout ce qui est de la mise à jour du monde
}
```

 Pour des raisons de simplicité du code et de fluidité d'un jeu vidéo les fonctions `SDL_WaitEvent()` ou `SDL_WaitEventTimeout()` sont déconseillées.

II-A-1-e - Remplissage de la queue d'événements

Lors des appels à `SDL_WaitEvent()`, `SDL_WaitEventTimeout()` et `SDL_PollEvent()`, la fonction `SDL_PumpEvents()` est appelée. Cette fonction permet d'analyser les différents périphériques (clavier, souris...) afin de récupérer tous les nouveaux événements et les mettre dans la queue d'événements. Comme la fonction est implicitement appelée, vous n'avez pas besoin de le faire vous-même.

`SDL_PumpEvents()` ne peut être appelée que dans le thread initiateur du mode vidéo et la documentation conseille même de ne l'appeler que dans le thread principal. Les fonctions `SDL_WaitEvent()`, `SDL_WaitEventTimeout()` et `SDL_PollEvent()` sont contraintes par cette même règle, car elles appellent `SDL_PumpEvents()`. Cette limitation est due aux détails de l'implémentation de la fonction `SDL_PumpEvents()` (certains systèmes ne peuvent récupérer les événements autrement).

La fonction ne prend et ne retourne aucune valeur :

```
SDL_PumpEvents();
```

II-A-2 - Analyse des événements

II-A-2-a - Événement de la fenêtre

Lorsque vous recevez un événement de la fenêtre, dont le type est `SDL_WINDOWEVENT`, vous avez accès aux informations suivantes dans le champ `window` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
Uint32	windowID	La fenêtre sur laquelle l'événement a eu lieu.
Uint8	event	Un identifiant de l'événement parmi SDL_WindowEventID .
Sint32	data1	Données dépendantes à l'événement.
Sint32	data2	

II-A-2-b - Événement du clavier

Lorsque vous recevez un événement du clavier, dont le type est `SDL_KEYDOWN` ou `SDL_KEYUP`, vous avez accès aux informations suivantes dans le champ `key` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
Uint32	windowID	La fenêtre sur laquelle l'événement a eu lieu.
Uint8	state	L'état de la touche (soit <code>SDL_PRESSED</code> lorsqu'enfoncée, soit <code>SDL_RELEASED</code> lorsque relâchée).
Uint8	repeat	Différent de zéro, si c'est une répétition de touche.
<code>SDL_Keysym</code>	keysym	Une structure <code>SDL_Keysym</code> identifiant la touche.

II-A-2-b-i - `SDL_Keysym`

La structure `SDL_Keysym` est utilisée pour identifier une touche. En effet, il existe deux moyens d'identifier la touche initiatrice de l'événement (et cela est quelque peu différent de la SDL 1.2). Voici le contenu de la structure :

Type	Nom	Description
<code>SDL_Scancode</code>	scancode	Code de la touche physique .
<code>SDL_Keycode</code>	sym	Code de la représentation virtuelle de la touche.
Uint16	mod	Une combinaison des touches spéciales du clavier (Ctrl/Shift/...). Voir SDL_Keymod .
Uint32		Non utilisé

La différence entre la représentation physique de la touche et la représentation virtuelle est importante. En effet, si vous développez un jeu et que vous souhaitez utiliser QZSD sur votre clavier azerty, un joueur américain aura du mal, car les touches ne sont pas à la même place. Ceci est dû à la représentation virtuelle, qui elle, dépend de la langue. Plus précisément, un clavier américain n'est physiquement pas (ou très peu) différent d'un clavier français, il y a bien six lignes de touches. D'ailleurs, la tabulation est toujours à la même place. Par contre, la configuration du système dit : « pour la touche à droite de la tabulation, je vais afficher 'a' » (si le système est français). Mais si le système est configuré autrement, il peut dire : « pour la touche à droite de la tabulation, je vais afficher 'q' ou encore '» (clavier dvorak) ». La représentation physique ne changera donc pas, même si la configuration du système change.

II-A-2-c - Événements de la souris

II-A-2-c-i - Déplacement de la souris

Lorsque vous recevez un événement de déplacement de la souris, dont le type est `SDL_MOUSEMOTION`, vous avez accès aux informations suivantes dans le champ `motion` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
Uint32	windowID	La fenêtre sur laquelle l'événement a eu lieu.
Uint32	which	Identifiant de la souris (pour les événements venant d'une surface tactile).
Uint32	state	Combinaison suivant les boutons enfoncés de la souris.
Sint32	x	Coordonnée en X relative à la fenêtre.
Sint32	y	Coordonnée en Y relative à la fenêtre.
Sint32	xrel	Déplacement sur l'axe des X.
Sint32	yrel	Déplacement sur l'axe des Y.

II-A-2-c-ii - Appui sur les boutons de la souris

Lorsque vous recevez un événement d'appui d'un bouton de la souris, dont le type est `SDL_MOUSEBUTTONDOWN` ou `SDL_MOUSEBUTTONUP`, vous avez accès aux informations suivantes dans le champ `button` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
Uint32	windowID	La fenêtre sur laquelle l'événement a eu lieu.
Uint32	which	Identifiant de la souris (pour les événements venant d'une surface tactile).
Uint8	button	Le bouton donc le changement d'état a initié l'événement.
Uint8	state	<code>SDL_PRESSED</code> si le bouton est enfoncé ou <code>SDL_RELEASED</code> si le bouton est relâché.
Uint8	clicks	1 pour un clic simple, 2 pour un double clic, etc. (\geq SDL 2.0.2).
Sint32	x	Coordonnée en X relative à la fenêtre.
Sint32	y	Coordonnée en Y relative à la fenêtre.

II-A-2-c-iii - Utilisation de la roulette

Lorsque vous recevez un événement d'utilisation de la roulette de la souris, dont le type est `SDL_MOUSEWHEEL`, vous avez accès aux informations suivantes dans le champ `wheel` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
Uint32	windowID	La fenêtre sur laquelle l'événement a eu lieu.
Uint32	which	Identifiant de la souris (pour les événements venant d'une surface tactile).
Sint32	x	Valeur de défilement horizontal, positive lorsque le défilement est vers la droite, sinon négative.
Sint32	y	Valeur de défilement vertical, positive lorsque le défilement est vers le haut, sinon négative.

II-A-2-d - Événement de joystick

II-A-2-d-i - Connexion/Déconnexion de joystick

Lorsque vous recevez un événement de connexion ou de déconnexion d'un joystick, dont le type est respectivement `SDL_JOYDEVICEADDED` ou `SDL_JOYDEVICEREMOVED`, vous avez accès aux informations suivantes dans le champ `jdevice` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
Uint32	which	Lors d'une connexion : l'indice du périphérique. Lors d'une déconnexion : l'identifiant de l'instance.

II-A-2-d-ii - Déplacement d'un joystick

Lorsque vous recevez un événement de déplacement d'un joystick, dont le type est `SDL_JOYAXISMOTION`, vous avez accès aux informations suivantes dans le champ `jaxis` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
SDL_JoystickID	which	L'indice du joystick.
Uint8	axis	L'indice de l'axe qui a bougé.
Sint16	value	La position actuelle de l'axe (entre -32768 et 32767).

II-A-2-d-iii - Appui d'un bouton de joystick

Lorsque vous recevez un événement d'appui sur un bouton d'un joystick, dont le type est `SDL_JOYBUTTONDOWN` ou `SDL_JOYBUTTONUP`, vous avez accès aux informations suivantes dans le champ `jbutton` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
SDL_JoystickID	which	L'indice du joystick.
Uint8	button	L'indice du bouton initiant l'événement.
Uint8	state	L'état du bouton (SDL_PRESSED lorsqu'enfoncé et SDL_RELEASED lorsque relâché).

II-A-2-d-iv - Déplacement de trackball d'un joystick

Lorsque vous recevez un événement de déplacement de trackball d'un joystick, dont le type est `SDL_JOYBALLMOTION`, vous avez accès aux informations suivantes dans le champ `jball` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
SDL_JoystickID	which	L'indice du joystick.
Uint8	ball	L'indice du trackball initiant l'événement.
Sint16	xrel	Le déplacement sur l'axe des X.
Sint16	yrel	Le déplacement sur l'axe des Y.

II-A-2-d-v - Déplacement d'un chapeau d'un joystick

Lorsque vous recevez un événement de déplacement d'un chapeau d'un joystick, dont le type est `SDL_JOYHATMOTION`, vous avez accès aux informations suivantes dans le champ `jhat` de `SDL_Event` :

Type	Nom	Description
Uint32	timestamp	Marqueur temporel.
SDL_JoystickID	which	L'indice du joystick.
Uint8	hat	L'indice du chapeau initiant l'événement.
Uint8	value	La nouvelle position du chapeau.

II-B - Traitement de `SDL_Event`

Maintenant que nous savons récupérer les `SDL_Event` de la queue des événements, il faut analyser les valeurs reçues.

Le procédé pour ce faire est simple. Vous recevez un `SDL_Event` de la queue des événements, celui-ci va contenir les informations pour un événement d'un type défini. Il suffit donc de :

- regarder quel type d'événement vous avez ;
- lire les informations associées à ce type.

Ainsi, il suffit de faire un `switch` du type et dans chacun des cas, le code de traitement des informations pour cet événement :

```

SDL_Event event;
if ( SDL_PollEvent(&event) )
{
    switch(event.type)
    {
        case SDL_WINDOWEVENT: // Événement de la fenêtre
            if ( event.window.event == SDL_WINDOWEVENT_CLOSE ) // Fermeture de la fenêtre
            {
                // Action à faire lorsque l'utilisateur clique sur la croix
            }
            break;
        case SDL_KEYUP: // Événement de relâchement d'une touche clavier
            if ( event.key.keysym.sym == SDLK_ESCAPE ) // C'est la touche Échap
            {
                // Action à faire lorsque l'utilisateur relâche la touche Échap
            }
            break;
    }
}

```

II-C - Code d'exemple

Le mieux pour présenter l'utilisation de tous ces événements est de créer un programme qui se contentera d'afficher un message suivant la touche appuyée.

Tous les événements ne sont pas traités, mais il suffit de regarder la documentation et de suivre le même procédé afin de gérer les événements manquants.

Voici le code source de ce programme :

```

#include <SDL2/SDL.h>

#include <stdio.h>

int main(int argc, char** argv)
{
    /* Initialisation simple */
    if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_JOYSTICK) != 0 )
    {
        fprintf(stdout, "Échec de l'initialisation de la SDL (%s)\n", SDL_GetError());
        return -1;
    }

    {
        /* Création de la fenêtre */
        SDL_Window* pWindow = NULL;
        pWindow = SDL_CreateWindow("Ma première application SDL2", SDL_WINDOWPOS_UNDEFINED,
                                   SDL_WINDOWPOS_UNDEFINED,
                                   640,
                                   480,
                                   SDL_WINDOW_SHOWN);

        if (pWindow)
        {
            char cont = 1; /* Détermine si on continue la boucle principale */
            SDL_Event event;
            SDL_Joystick* pJoy=NULL;

            SDL_JoystickEventState(SDL_ENABLE); // On indique que l'on souhaite utiliser le système d'événements pour les j

            while ( cont != 0 )
            {
                while ( SDL_PollEvent(&event) )
                {
                    /* Traitement de l'événement */
                    switch (event.type) /* Quel événement avons-nous ? */

```

```

{
    case SDL_KEYDOWN:
        fprintf(stdout, "Un appuie sur une touche :\n");
        fprintf(stdout, "\trépétition ? : %d\n", event.key.repeat);
        fprintf(stdout, "\tscancode : %d\n", event.key.keysym.scancode);
        fprintf(stdout, "\tkey : %d\n", event.key.keysym.sym);
        if ( event.key.keysym.scancode == SDL_SCANCODE_ESCAPE )
        {
            cont = 0;
        }
        break;
    case SDL_KEYUP:
        fprintf(stdout, "Un relachement d'une touche :\n");
        fprintf(stdout, "\trépétition ? : %d\n", event.key.repeat);
        fprintf(stdout, "\tscancode : %d\n", event.key.keysym.scancode);
        fprintf(stdout, "\tkey : %d\n", event.key.keysym.sym);
        break;

    case SDL_MOUSEMOTION:
        fprintf(stdout, "Un déplacement de la souris :\n");
        fprintf(stdout, "\tfenêtre : %d\n", event.motion.windowID);
        fprintf(stdout, "\tsouris : %d\n", event.motion.which);
        fprintf(stdout, "\tposition : %d;%d\n", event.motion.x, event.motion.y);
        fprintf(stdout, "\tdelta : %d;%d\n", event.motion.xrel, event.motion.yrel);
        break;
    case SDL_MOUSEBUTTONDOWN:
        fprintf(stdout, "Un relachement d'un bouton de la souris :\n");
        fprintf(stdout, "\tfenêtre : %d\n", event.button.windowID);
        fprintf(stdout, "\tsouris : %d\n", event.button.which);
        fprintf(stdout, "\tbouton : %d\n", event.button.button);
#if SDL_VERSION_ATLEAST(2,0,2)
        fprintf(stdout, "\tclics : %d\n", event.button.clicks);
#endif

        fprintf(stdout, "\tposition : %d;%d\n", event.button.x, event.button.y);
        break;
    case SDL_MOUSEBUTTONDOWN:
        fprintf(stdout, "Un appuie sur un bouton de la souris :\n");
        fprintf(stdout, "\tfenêtre : %d\n", event.button.windowID);
        fprintf(stdout, "\tsouris : %d\n", event.button.which);
        fprintf(stdout, "\tbouton : %d\n", event.button.button);
#if SDL_VERSION_ATLEAST(2,0,2)
        fprintf(stdout, "\tclics : %d\n", event.button.clicks);
#endif

        fprintf(stdout, "\tposition : %d;%d\n", event.button.x, event.button.y);
        break;
    case SDL_MOUSEWHEEL:
        fprintf(stdout, "Roulette de la souris :\n");
        fprintf(stdout, "\tfenêtre : %d\n", event.wheel.windowID);
        fprintf(stdout, "\tsouris : %d\n", event.wheel.which);
        fprintf(stdout, "\tposition : %d;%d\n", event.wheel.x, event.wheel.y);
        break;

    case SDL_JOYDEVICEADDED:
        fprintf(stdout, "Connexion de joystick :\n");
        fprintf(stdout, "\tjoystick : %d\n", event.jdevice.which);

        if ( pJoy == NULL )
        {
            pJoy = SDL_JoystickOpen(event.jdevice.which);
        }
        else
        {
            fprintf(stdout, "Ce nouveau joystick ne sera pas ouvert (le programme ne gère qu'un joystick)\n");
        }
        break;
    case SDL_JOYDEVICEREMOVED:
        fprintf(stdout, "Déconnexion de joystick :\n");
        fprintf(stdout, "\tjoystick : %d\n", event.jdevice.which);

        if ( pJoy != NULL )
        {

```

```

        SDL_JoystickClose(pJoy);
        pJoy = NULL;
    }
    break;
case SDL_JOYAXISMOTION:
    fprintf(stdout, "Déplacement joystick :\n");
    fprintf(stdout, "\tjoystick : %d\n", event.jaxis.which);
    fprintf(stdout, "\taxe : %d\n", event.jaxis.axis);
    fprintf(stdout, "\tvaleur : %d\n", event.jaxis.value);
    break;
case SDL_JOYBUTTONDOWN:
    fprintf(stdout, "Appui bouton joystick :\n");
    fprintf(stdout, "\tjoystick : %d\n", event.jbutton.which);
    fprintf(stdout, "\tbutton : %d\n", event.jbutton.button);
    fprintf(stdout, "\tétat : %d\n", event.jbutton.state);
    break;
case SDL_JOYBUTTONUP:
    fprintf(stdout, "Relâchement bouton joystick :\n");
    fprintf(stdout, "\tjoystick : %d\n", event.jbutton.which);
    fprintf(stdout, "\tbutton : %d\n", event.jbutton.button);
    fprintf(stdout, "\tétat : %d\n", event.jbutton.state);
    break;
case SDL_JOYBALLMOTION:
    fprintf(stdout, "Déplacement de trackball :\n");
    fprintf(stdout, "\tjoystick : %d\n", event.jball.which);
    fprintf(stdout, "\ttrackball : %d\n", event.jball.ball);
    fprintf(stdout, "\tdéplacement : %d;%d\n", event.jball.xrel, event.jball.yrel);
    break;
case SDL_JOYHATMOTION:
    fprintf(stdout, "Déplacement de chapeau d'un joystick :\n");
    fprintf(stdout, "\tjoystick : %d\n", event.jhat.which);
    fprintf(stdout, "\tbutton : %d\n", event.jhat.hat);
    fprintf(stdout, "\tvaleur : %d\n", event.jhat.value);
    break;

case SDL_WINDOWEVENT:
    fprintf(stdout, "Un événement de fenêtre, sur la fenêtre : %d\n", event.window.windowID);

    // En théorie, ici, il faudrait faire un autre test ou switch pour chaque type de cet événement
    break;
default:
    fprintf(stdout, "Événement non traité : %d\n", event.type);
}

fprintf(stdout, "\n");
}

/* On a traité les événements, on peut continuer le jeu */
}

if ( pJoy != NULL )
{
    SDL_JoystickClose(pJoy);
    pJoy = NULL;
}

SDL_DestroyWindow(pWindow);
}
else
{
    fprintf(stderr, "Erreur de création de la fenêtre: %s\n", SDL_GetError());
}
}

SDL_Quit();

return 0;
}

```


III - Statut des périphériques

La méthode précédente pour récupérer les événements ne correspond pas à tous les besoins. Dans certaines applications, ou jeux, il est préférable de vérifier si une touche est appuyée et ce, à n'importe quel moment.

Avec la méthode précédente, vous devez créer un tableau, pour conserver l'état appuyé ou non de vos touches. Sachant que la SDL propose un mécanisme similaire, il est préférable de l'utiliser au lieu de le reprogrammer soi-même.

En effet, il est possible de connaître l'état des périphériques et donc de savoir si des touches du clavier sont appuyées, ou encore, l'état de la manette (boutons, joysticks...) et cela, sans avoir besoin d'analyser la queue d'événements.


III-A - Fonctionnement

La mise en place est encore plus simple que pour les `SDL_Event`. Ici, il suffit d'appeler `SDL_PumpEvents()` avant de récupérer l'état des périphériques pour que la SDL mette à jour ses informations sur tous les périphériques.

Voici un exemple pour la position de la souris :

```
int x, y;
Uint32 boutons;

SDL_PumpEvents();
boutons = SDL_GetMouseState(&x, &y);
// Vous n'avez plus qu'à utiliser x, y et/ou boutons
```

 ***SDL_PumpEvents()** ne peut être appelée que dans le thread initiateur du mode vidéo et la documentation conseille même de ne l'appeler que dans le thread principal.*

III-B - Les informations de la souris

III-B-1 - SDL_GetMouseState()

`SDL_GetMouseState()` permet de récupérer la position de la souris en coordonnées de fenêtre (0, 0 correspond au coin supérieur gauche et le coin inférieur droit aura les valeurs de la taille de la fenêtre). De plus, la valeur renvoyée dépend des boutons appuyés. Pour déterminer quels sont les boutons appuyés, vous devez utiliser la macro `SDL_BUTTON(X)` (où X correspond généralement à 1, pour le bouton de gauche, 2 pour le bouton du milieu et 3 pour le bouton de droite), par exemple :

```
SDL_PumpEvents();
if(SDL_GetMouseState(NULL, NULL) & SDL_BUTTON(1))
    printf("Le bouton 1 (gauche) est appuyé.\n");
```

III-B-1-a - Les paramètres de SDL_GetMouseState()

La fonction prend deux pointeurs en paramètre. Les valeurs pointées vont être remplies avec les valeurs X et Y de la position de la souris. Vous pouvez aussi passer `NULL` et dans ce cas, vous n'aurez pas les coordonnées de la souris.

III-B-1-b - La valeur de retour de SDL_GetMouseState()

La valeur de retour est un nombre binaire, où chaque bit correspond à un bouton de la souris. Pour vérifier si un bouton est appuyé, vous devez utiliser un masque afin de filtrer le bit correspondant au bouton voulu. Pour créer ce masque vous pouvez utiliser la macro `SDL_BUTTON()` :

```
SDL_PumpEvents();  
if(SDL_GetMouseState(NULL, NULL) & SDL_BUTTON(1))  
    printf("Le bouton 1(gauche) est appuyé.\n");
```

III-B-1-b-i - La macro SDL_BUTTON

SDL_BUTTON est une macro permettant de créer un entier spécifique pour tester si tel ou tel bouton est appuyé. En effet, la valeur retournée par SDL_GetMouseState() est un nombre binaire où les bits mis à 1 représentent les boutons appuyés. La macro va générer un entier avec un bit à 1, pour le bouton correspondant à l'index que vous passez. Il est ensuite donc possible de faire un ET logique, pour déterminer si ce bit est aussi à 1 dans le nombre retourné par la fonction SDL_GetMouseState().

III-B-2 - SDL_GetRelativeMouseState()

La fonction SDL_GetRelativeMouseState() est similaire à SDL_GetMouseState(). L'unique différence est que les valeurs de la position de la souris retournées ne sont pas par rapport à la fenêtre, mais par rapport à l'ancienne position de la souris lors du précédent appel à SDL_GetRelativeMouseState(). Avec cette fonction, vous allez donc recevoir le déplacement de la souris effectué entre deux appels à la fonction. Une valeur positive indique un déplacement sur la droite pour l'axe des X et un déplacement vers le bas pour l'axe des Y.


Les paramètres et valeur de retour fonctionnent exactement comme ceux de SDL_GetMouseState().

III-C - Les informations du clavier

III-C-1 - SDL_GetKeyboardState()

La fonction SDL_GetKeyboardState() permet de récupérer l'état de toutes les touches du clavier. Pour cela, la fonction retourne un pointeur sur un tableau où chaque case correspond à une touche. La valeur correspond à l'état de la touche (0 si elle n'est pas appuyée). Ainsi, pour tester si une touche est appuyée, vous pouvez écrire le code suivant :


```
const Uint8 *state = SDL_GetKeyboardState(NULL);  
if (state[SDL_SCANCODE_RETURN]) {  
    printf("La touche entree est appuyée.  
    »);  
}
```

 La fonction ne prend pas en compte si la touche majuscule est utilisée ou non.

III-C-1-a - Le paramètre de SDL_GetKeyboardState()

La fonction SDL_GetKeyboardState() accepte un unique paramètre : un pointeur sur un entier. Ce pointeur permet à la fonction de renseigner la taille du tableau. Vous pouvez passer NULL, dans ce cas, vous n'obtiendrez pas la taille du tableau retourné.

III-C-1-a-i - La valeur de retour de SDL_GetKeyboardState()

La fonction SDL_GetKeyboardState() retourne un pointeur sur un tableau correspondant aux états de chacune des touches du clavier. Les indices à utiliser pour accéder à ce tableau sont les  **SDL_Scancode**. Si la valeur correspondant à tel ou tel indice est 1, alors la touche désignée par l'indice est appuyée. Sinon, la valeur sera 0. Ce tableau est géré par la SDL. Vous ne devez donc pas faire de free() dessus.

III-C-2 - SDL_GetModState()

La fonction `SDL_GetModState()` retourne une combinaison de valeurs dépendantes des touches spéciales du clavier (Ctrl/Shift/...). La liste des valeurs pouvant être retournées est accessible ici : [SDL_Keymod](#). La valeur peut être une combinaison (OU logique) de ces valeurs.

III-C-2-a - La valeur de retour de SDL_GetModState()

La valeur de retour de `SDL_GetModState()` est l'**une des suivantes**, ou une combinaison (OU logique) de celles-ci.

III-C-3 - Les informations des joysticks

Les joysticks, pour être utilisés, doivent être ouverts et fermés (comme un fichier, par exemple). Ainsi, la première étape sera de déterminer le nombre de joysticks disponibles sur la machine avec la fonction `SDL_NumJoysticks()`.

 *N'oubliez pas d'initialiser le système `SDL_INIT_JOYSTICK` lors de l'appel à <http://alexandre-laurent.developpez.com/tutoriels/sdl-2/creer-premieres-fenetres/#LIII> ou avec un appel à `SDL_InitSubSystem()`.*

Une fois que vous êtes certain qu'au moins un joystick est connecté à la machine, vous pouvez « l'ouvrir » avec `SDL_JoystickOpen()`. Vous devez le faire autant de fois qu'il y a de joysticks si vous souhaitez accéder à tous les joysticks connectés.

La fonction vous retournera un pointeur vers un `SDL_Joystick` vous permettant d'identifier ce joystick par la suite (nécessaire pour utiliser les autres fonctions liées aux joysticks).

Vous pouvez obtenir des informations sur le nom du joystick, son nombre d'axes, de boutons, de chapeaux et de trackballs à l'aide de : `SDL_JoystickName()`, `SDL_JoystickNumAxes()`, `SDL_JoystickNumButtons()`, `SDL_JoystickNumHats()`, `SDL_JoystickNumBalls()` respectivement.

En ayant pleine connaissance du joystick accessible, vous pouvez analyser l'état des axes, boutons, chapeaux et trackballs avec les fonctions : `SDL_JoystickGetAxis()`, `SDL_JoystickGetButton()`, `SDL_JoystickGetHat()`, `SDL_JoystickGetBall()` respectivement.

Notez que vous devez utiliser la fonction `SDL_JoystickUpdate()` avant d'essayer de récupérer les informations sur les joysticks afin que la SDL mette à jour les informations sur les joysticks.

Finalement, lorsque vous n'avez plus besoin d'utiliser tel ou tel joystick, vous devez utiliser la fonction `SDL_JoystickClose()`.

III-C-3-a - SDL_NumJoysticks()

La fonction `SDL_NumJoysticks()` retourne le nombre de joysticks présent sur le système.

III-C-3-a-i - La valeur de retour de SDL_NumJoysticks()


`SDL_NumJoysticks()` retourne le nombre de joysticks présents sur le système ou un nombre négatif en cas d'erreur. Vous pouvez récupérer plus de précision sur l'erreur avec la fonction `SDL_GetError()`.

III-C-3-b - SDL_JoystickOpen()

La fonction `SDL_JoystickOpen()` permet d'initialiser un joystick en vue d'une utilisation future. La fonction retourne un pointeur sur la structure `SDL_Joystick*` vous permettant d'identifier le joystick.

III-C-3-b-i - Le paramètre de SDL_JoystickOpen()

`SDL_JoystickOpen()` n'accepte qu'un paramètre : le numéro du joystick à ouvrir. Ce numéro doit être un nombre entre 0 et le nombre de joysticks connectés (voir `SDL_NumJoysticks()`).

 *Ce numéro de joystick **n'est pas** le même que l'identifiant d'instance utilisé pour identifier le joystick dans les événements.*

III-C-3-b-ii - La valeur de retour de SDL_JoystickOpen()

`SDL_JoystickOpen()` retourne un pointeur sur une instance de `SDL_Joystick`, ou `NULL` en cas d'erreur. Il est possible d'avoir plus d'informations sur l'erreur avec `SDL_GetError()`.

III-C-3-c - SDL_JoystickName()

La fonction `SDL_JoystickName()` permet d'obtenir le nom associé à un joystick.

III-C-3-c-i - Le paramètre de SDL_JoystickName()

`SDL_JoystickName()` accepte uniquement un pointeur sur une instance de `SDL_Joystick` permettant d'identifier le joystick pour lequel le nom doit être retourné.

III-C-3-c-ii - La valeur de retour de SDL_JoystickName()

`SDL_JoystickName()` retourne le nom du joystick sous la forme d'une chaîne de caractères ou `NULL` en cas d'erreur. Il est possible d'avoir plus d'informations sur l'erreur avec `SDL_GetError()`.

III-C-3-d - SDL_JoystickNumAxes()

La fonction `SDL_JoystickNumAxes()` permet de connaître le nombre d'axes présent sur un joystick.

III-C-3-d-i - Le paramètre de SDL_JoystickNumAxes()

`SDL_JoystickNumAxes()` accepte uniquement un pointeur sur une instance de `SDL_Joystick` permettant d'identifier le joystick pour lequel le nombre d'axes doit être retourné.

III-C-3-d-ii - La valeur de retour de SDL_JoystickNumAxes()

`SDL_JoystickNumAxes()` retourne le nombre d'axes disponibles sur le joystick ou une valeur négative en cas d'erreur. Il est possible d'avoir plus d'informations sur l'erreur avec `SDL_GetError()`.

III-C-3-e - SDL_JoystickNumButtons()

La fonction `SDL_JoystickNumButtons()` permet de connaître le nombre de boutons présents sur un joystick.

III-C-3-e-i - Le paramètre de `SDL_JoystickNumButtons()`

`SDL_JoystickNumButtons()` accepte uniquement un pointeur sur une instance de `SDL_Joystick` permettant d'identifier le joystick pour lequel le nombre de boutons doit être retourné.

III-C-3-e-ii - La valeur de retour de `SDL_JoystickNumButtons()`

`SDL_JoystickNumButtons()` retourne le nombre de boutons disponibles sur le joystick ou une valeur négative en cas d'erreur. Il est possible d'avoir plus d'informations sur l'erreur avec `SDL_GetError()`.

III-C-3-f - `SDL_JoystickNumHats()`

La fonction `SDL_JoystickNumHats()` permet de connaître le nombre de chapeaux présents sur un joystick.

III-C-3-f-i - Le paramètre de `SDL_JoystickNumHats()`

`SDL_JoystickNumHats()` accepte uniquement un pointeur sur une instance de `SDL_Joystick` permettant d'identifier le joystick pour lequel le nombre de chapeaux doit être retourné.

III-C-3-f-ii - La valeur de retour de `SDL_JoystickNumHats()`

`SDL_JoystickNumHats()` retourne le nombre de chapeaux disponibles sur le joystick ou une valeur négative en cas d'erreur. Il est possible d'avoir plus d'informations sur l'erreur avec `SDL_GetError()`.

III-C-3-g - `SDL_JoystickNumBalls()`

La fonction `SDL_JoystickNumBalls()` permet de connaître le nombre de trackballs présentes sur un joystick.

III-C-3-g-i - Le paramètre de `SDL_JoystickNumBalls()`

`SDL_JoystickNumBalls()` accepte uniquement un pointeur sur une instance de `SDL_Joystick` permettant d'identifier le joystick pour lequel le nombre de trackballs doit être retourné.

III-C-3-g-ii - La valeur de retour de `SDL_JoystickNumBalls()`

`SDL_JoystickNumBalls()` retourne le nombre de trackballs disponibles sur le joystick ou une valeur négative en cas d'erreur. Il est possible d'avoir plus d'informations sur l'erreur avec `SDL_GetError()`.

III-C-3-h - `SDL_JoystickGetAxis()`

La fonction `SDL_JoystickGetAxis()` permet d'obtenir la valeur associée à un axe.

III-C-3-h-i - Les paramètres de `SDL_JoystickGetAxis()`

`SDL_JoystickGetAxis()` accepte un pointeur sur une instance de `SDL_Joystick` permettant d'identifier le joystick et un entier compris entre 0 et la valeur retournée par `SDL_JoystickNumAxes()`, identifiant l'axe pour lequel la valeur est retournée. Généralement, l'axe X est représenté par 0 et l'axe Y par 1.

III-C-3-h-ii - La valeur de retour de `SDL_JoystickGetAxis()`

La valeur retournée par `SDL_JoystickGetAxis()` représente la position de l'axe. Elle est comprise entre -32768 et 32767. Il peut être nécessaire d'appliquer quelques tolérances pour éviter les tremblements.

III-C-3-i - `SDL_JoystickGetButton()`

La fonction `SDL_JoystickGetButton()` permet d'obtenir la valeur associée à un bouton.

III-C-3-i-i - Les paramètres de `SDL_JoystickGetButton()`

`SDL_JoystickGetButton()` accepte un pointeur sur une instance de `SDL_Joystick` permettant d'identifier le joystick et un entier compris entre 0 et la valeur retournée par `SDL_JoystickNumButtons()`, identifiant le bouton pour lequel la valeur est retournée.

III-C-3-i-ii - La valeur de retour de `SDL_JoystickGetButton()`

La valeur retournée par `SDL_JoystickGetButton()` est soit 0 pour un bouton relâché et 1 pour un bouton appuyé.

III-C-3-j - `SDL_JoystickGetHat()`

La fonction `SDL_JoystickGetHat()` permet d'obtenir la valeur associée à un axe.

III-C-3-j-i - Les paramètres de `SDL_JoystickGetHat()`

`SDL_JoystickGetHat()` accepte un pointeur sur une instance de `SDL_Joystick` permettant d'identifier le joystick et un entier compris entre 0 et la valeur retournée par `SDL_JoystickNumHats()`, identifiant l'axe pour lequel la valeur est retournée.

III-C-3-j-ii - La valeur de retour de `SDL_JoystickGetHat()`

La valeur retournée par `SDL_JoystickGetHat()` représente la position du chapeau. Elle peut être l'une de ces valeurs :

<code>SDL_HAT_CENTERED</code>	Position centrée (ou relâchée)
<code>SDL_HAT_UP</code>	Position haut
<code>SDL_HAT_RIGHT</code>	Position droite
<code>SDL_HAT_DOWN</code>	Position bas
<code>SDL_HAT_LEFT</code>	Position gauche
<code>SDL_HAT_RIGHTUP</code>	Position haut droite
<code>SDL_HAT_RIGHTDOWN</code>	Position bas droite
<code>SDL_HAT_LEFTUP</code>	Position haut gauche
<code>SDL_HAT_LEFTDOWN</code>	Position bas gauche

III-C-3-k - `SDL_JoystickGetBall()`

La fonction `SDL_JoystickGetBall()` permet d'obtenir la valeur associée à une trackball.

III-C-3-k-i - Les paramètres de SDL_JoystickGetBall()

SDL_JoystickGetBall() accepte un pointeur sur une instance de SDL_Joystick permettant d'identifier le joystick, un entier compris entre 0 et la valeur retournée par SDL_JoystickNumBalls(). Les troisième et quatrième paramètres sont des pointeurs sur des entiers permettant à la fonction de retourner des valeurs. Les valeurs retournées à travers ces pointeurs correspondent au déplacement effectué avec la trackball depuis le dernier appel à SDL_JoystickUpdate().

III-C-3-k-ii - La valeur de retour de SDL_JoystickGetBall()

La valeur retournée par SDL_JoystickGetBall() est négative si une erreur a eu lieu, sinon c'est 0. Il est possible d'avoir plus d'informations sur l'erreur avec SDL_GetError().

III-C-3-l - SDL_JoystickUpdate()

La fonction SDL_JoystickUpdate() permet de mettre à jour les données relatives aux joysticks. La fonction est automatiquement appelée dans la boucle événementielle si les événements d'un joystick sont activés.

III-C-3-m - SDL_JoystickEventState()

La fonction SDL_JoystickEventState() permet d'indiquer si les événements des joysticks doivent être pris en compte dans la boucle événementielle ou non. Si les événements joystick sont désactivés, vous devez appeler SDL_JoystickUpdate() vous-même.

La documentation conseille de laisser cette option activée.

III-C-3-m-i - Le paramètre de SDL_JoystickEventState()

Le paramètre de la fonction SDL_JoystickEventState() est une valeur parmi les trois suivantes :

SDL_ENABLE	Active le type d'événement.
SDL_IGNORE	Désactive le type d'événement.
SDL_QUERY	Permet de connaître l'état actuel (activé ou désactivé) d'un type d'événement et n'applique aucun changement.

III-C-3-m-ii - La valeur de retour de SDL_JoystickEventState()

La valeur de retour est l'état précédent le changement d'état. Si le second paramètre de la fonction est SDL_QUERY, la valeur retournée est l'état actuel pour le type d'événement indiqué par le premier paramètre.

III-C-3-n - SDL_JoystickClose()

La fonction SDL_JoystickClose() permet de fermer un joystick précédemment ouvert avec SDL_JoystickOpen().

III-C-3-n-i - Le paramètre de SDL_JoystickClose()

SDL_JoystickClose() accepte uniquement un pointeur sur une instance de SDL_Joystick permettant d'identifier le joystick le plus utilisé.

III-C-4 - Code d'exemple

Le mieux pour présenter l'utilisation de toutes ces fonctions est de créer un programme qui se contentera d'afficher un message suivant la touche appuyée. Voici le code source de ce programme :

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_version.h>

#include <stdio.h>

int main(int argc, char** argv)
{
    /* Initialisation simple */
    if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_JOYSTICK) != 0 )
    {
        fprintf(stdout, "Échec de l'initialisation de la SDL (%s)\n", SDL_GetError());
        return -1;
    }

    {
        /* Création de la fenêtre */
        SDL_Window* pWindow = NULL;
        pWindow = SDL_CreateWindow("Ma première application SDL2", SDL_WINDOWPOS_UNDEFINED,
                                   SDL_WINDOWPOS_UNDEFINED,
                                   640,
                                   480,
                                   SDL_WINDOW_SHOWN);

        if (pWindow)
        {
            char cont = 1; /* Détermine si on continue la boucle principale */

            while ( cont != 0 )
            {
                SDL_PumpEvents(); // On demande à la SDL de mettre à jour les états sur les périphériques

                // Clavier
                {
                    const Uint8* pKeyStates = SDL_GetKeyboardState(NULL);
                    if ( pKeyStates[SDL_SCANCODE_ESCAPE] )
                    {
                        cont = 0;
                    }
                    // On peut vérifier d'autres touches, si on le souhaite

                    SDL_Keymod mod = SDL_GetModState();
                    if ( mod != KMOD_NONE )
                    {
                        fprintf(stdout, "Vous avez appuyé sur une touche spéciale : %d\n", mod);
                    }
                }
                fprintf(stdout, "\n");
                // Souris
                {
                    int x=0;
                    int y=0;
                    Uint32 boutons = SDL_GetMouseState(&x, &y);

                    fprintf(stdout, "Position de la souris : %d;%d\n", x, y);
                    fprintf(stdout, "Bouton de la souris : %d\n", boutons);

                    SDL_GetRelativeMouseState(&x, &y);
                    fprintf(stdout, "Déplacement de la souris : %d;%d\n", x, y);
                }
                fprintf(stdout, "\n");

                // Joystick
                {
```



```

int nbJoysticks = SDL_NumJoysticks();
int i = 0;
for ( i = 0 ; i < nbJoysticks ; i++ )
{
    SDL_Joystick* pJoy = SDL_JoystickOpen(i);

    SDL_JoystickUpdate();

    if (pJoy)
    {
        printf("Joystick %d\n",i);
        printf("Nom: %s\n", SDL_JoystickNameForIndex(i));

        // Axes
        {
            int nbAxes = SDL_JoystickNumAxes(pJoy);
            int axe = 0;

            printf("Nombre d'axes : %d\n", nbAxes);
            for ( axe = 0 ; axe < nbAxes ; axe++ )
            {
                printf("Axe %d : %d\n",axe,SDL_JoystickGetAxis (pJoy,axe) );
            }
        }

        // Boutons
        {
            int nbBoutons = SDL_JoystickNumButtons (pJoy);
            int bouton = 0;

            printf("Nombre de boutons : %d\n", nbBoutons);
            for ( bouton = 0 ; bouton < nbBoutons ; bouton++ )
            {
                printf("Bouton %d : %d\n",bouton,SDL_JoystickGetButton (pJoy,bouton));
            }
        }

        // Trackballs
        {
            int nbBalls = SDL_JoystickNumBalls(pJoy);
            int ball = 0;

            printf("Nombre de trackballs : %d\n", nbBalls);
            for ( ball = 0 ; ball < nbBalls ; ball++ )
            {
                int dx=0;
                int dy=0;
                SDL_JoystickGetBall (pJoy,ball,&dx,&dy);
                printf("Déplacement trackball %d : %d;%d\n",ball,dx,dy);
            }
        }

        // Chapeaux
        {
            int nbChapeaux = SDL_JoystickNumHats (pJoy);
            int chapeau = 0;

            printf("Nombre de chapeaux : %d\n", nbChapeaux);
            for ( chapeau = 0 ; chapeau < nbChapeaux ; chapeau++ )
            {
                printf("Chapeau %d : %d\n",chapeau,SDL_JoystickGetHat (pJoy,chapeau));
            }
        }

        SDL_JoystickClose (pJoy);
    }
    else
    {
        printf("Couldn't open Joystick 0\n");
    }
}

```

```

    }
    }
    fprintf(stdout, "\n");

    /* On a traité les événements, on peut continuer le jeu */

    /* On ralentit un peu le programme */
    SDL_Delay(10);
}
SDL_DestroyWindow(pWindow);
}
else
{
    fprintf(stderr, "Erreur de création de la fenêtre: %s\n", SDL_GetError());
}
}

SDL_Quit();

return 0;
}

```

Le programme ne surveille pas tous les événements. Pour les événements manquants, veuillez vous référer à la documentation. Il ne devrait plus y avoir de difficultés à utiliser ce genre de fonctions.

IV - Filtrer des événements

Il est possible (et nécessaire pour les appareils mobiles) d'appliquer des filtres sur les événements. Pour cela, vous allez indiquer à la SDL (avec `SDL_SetEventFilter()`) une fonction propre à vous, qui sera appelée avant que l'événement reçu par la SDL n'intègre la queue d'événements.

Suivant la valeur de retour de votre fonction, l'événement sera ajouté ou non à la queue d'événements.

Si vous développez une application pour les appareils mobiles, les six événements suivants doivent être traités au plus tôt sans quoi, le système stoppera votre application :

Valeur de type	Description
SDL_APP_TERMINATING	Le système termine l'application.
SDL_APP_LOWMEMORY	Le système manque de mémoire ; libérez-en.
SDL_APP_WILLENTERBACKGROUND	L'application est envoyée en tâche de fond.
SDL_APP_DIDENTERBACKGROUND	L'application est en tâche de fond.
SDL_APP_WILLENTERFOREGROUND	L'application est envoyée en premier plan.
SDL_APP_DIDENTERFOREGROUND	L'application est en premier plan.

Pour être certain qu'ils soient traités au plus tôt, vous pouvez utiliser un filtre d'événements avec `SDL_SetEventFilter()`.

Si vous voulez complètement désactiver certains types d'événements (par exemple les événements de la souris dans un jeu de combat), vous pouvez utiliser la fonction `SDL_EventState()`.

IV-A - SDL_SetEventFilter()

La fonction `SDL_SetEventFilter()` permet de donner une fonction de votre code qui sera appelée pour chaque événement avant que celui-ci n'intègre la queue d'événements.

La fonction que vous devez indiquer à `SDL_SetEventFilter()` doit avoir la signature suivante :

```
int nomDeLaFonction(void* userdata, SDL_Event* event);
```

Votre fonction devra donc prendre deux paramètres : `userdata`, des données quelconques définies par vous et `event`, un pointeur sur l'événement à traiter.

De plus, vous devez retourner un entier, soit 1, pour indiquer que l'événement doit être ajouté à la queue des événements, soit 0, pour qu'il ne soit pas ajouté.

i Les événements désactivés (avec `SDL_EventState()`) ne sont pas traités par la fonction de filtre.

Voici un exemple d'utilisation de la fonction, permettant de cacher tous les événements du clavier :

```
int monFiltre(void* userdata, SDL_Event* event)
{
    if (event->type == SDL_KEYDOWN ||
        Event->type == SDL_KEYUP )
    {
        return 0; // ne seront pas ajoutés
    }
    Return 1; // N'oublions pas de laisser passer les autres événements
}

// On met en place le filtre
SDL_SetEventFilter(monFiltre, NULL);
```

IV-A-1 - Les paramètres de `SDL_SetEventFilter()`

`SDL_SetEventFilter()` accepte deux paramètres :

- un `SDL_EventFilter()`, un pointeur de fonction menant à la fonction que vous souhaitez utilisée pour le filtrage des événements. Le prototype doit être `int nomDeLaFonction(void* userdata, SDL_Event* event);`
- un pointeur `void*` vous permettant de passer n'importe quelles données que vous souhaitez à votre filtre.

IV-B - `SDL_EventState()`

`SDL_EventState()` permet de complètement désactiver un type d'événements. Ce faisant, il n'apparaîtra plus dans la queue d'événements. Cela peut être utile lorsque l'utilisateur produit beaucoup d'événements dont vous ne vous souciez pas et qui pourrait causer le ralentissement de votre code de traitement de la queue d'événements.

Voici un exemple d'utilisation de la fonction `SDL_EventState()` :

```
SDL_EventState(SDL_MOUSEWHEEL, SDL_DISABLE); // Désactive la roulette de la souris
```

IV-C - Les paramètres de `SDL_EventState()`

Le premier paramètre de la fonction `SDL_EventState()` est un type d'événement `SDL_EventType` indiquant sur quel événement la fonction doit agir. Le second paramètre est une valeur parmi les trois suivantes :

<code>SDL_ENABLE</code>	Active le type d'événement.
<code>SDL_IGNORE</code> ou <code>SDL_DISABLE</code>	Désactive le type d'événement.
<code>SDL_QUERY</code>	Permet de connaître l'état actuel (activé ou désactivé) d'un type d'événement et n'applique aucun changement.

IV-D - La valeur de retour de `SDL_EventState()`

La valeur de retour est l'état précédent le changement d'état. Si le second paramètre de la fonction est `SDL_QUERY`, la valeur retournée est l'état actuel pour le type d'événement indiqué par le premier paramètre.

V - Conclusion

Vous avez maintenant appris à traiter les actions que l'utilisateur peut faire pendant qu'il utilise votre programme. Ainsi, grâce aux événements, vous pouvez déplacer le personnage du joueur ou lui permettre de viser avec la souris.

Le choix de la gestion des événements, que ce soit par la queue d'événements ou les états des périphériques est un choix dépendant de votre jeu et de vos besoins. En elles-mêmes, les deux méthodes fonctionnent et vous permettront de récupérer les événements, mais suivant votre jeu, l'une sera plus facile pour réaliser vos désirs. Il est donc nécessaire de bien penser cette partie afin de réaliser un jeu plaisant pour le joueur où les contrôles réagissent bien et ne bloquent pas la liberté de mouvement.

Votre jeu se complète et possède enfin une composante dynamique grâce à laquelle le joueur peut agir sur le monde qu'il voit.

VI - Remerciements

Merci à **archMqx.** et **Kannagi** pour leurs précieuses suggestions ainsi que **Phanloga** pour sa correction orthographique.

Navigation

Tutoriel
précédent :
**afficher son
premier
sprite**

Sommaire