GESTION DES DIFFÉRENTS ÉVÉNEMENTS EN SDL 2

Introduction

Ce chapitre rendra vie à votre projet SDL! En effet, **comment faire en sorte d'interagir avec la scène 2D** ? La réponse est grâce aux événements ! Un événement est **géré par votre système d'exploitation**, et attend une action de votre part.

- **Vous :** Donc comment qu'on fait pour gérer plusieurs OS ?

- **Moi :** Tu n'as rien à faire car la SDL s'occupe de tous !

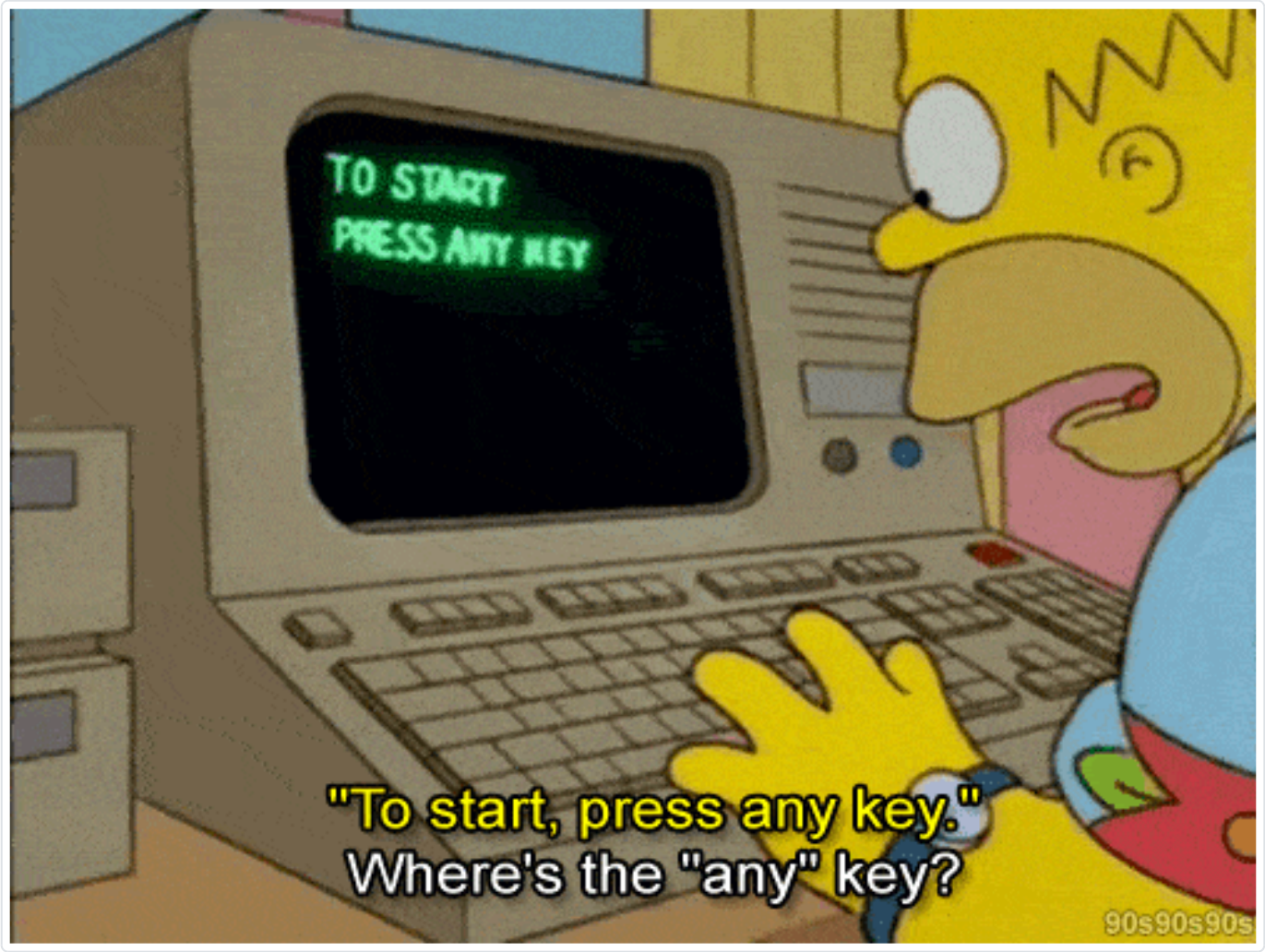
En effet la SDL reste une bibliothèque portable sous Linux et Windows (et d'autres OS). Elle gère donc automatiquement pour vous les événements peu importe la nature de votre OS. Ceci dit, voilà comment nous allons aborder ce chapitre :

1. **Explication et fonctionnement des événements**
2. **Événements liés à la fenêtre**
3. **Événements liés au clavier**
4. **Événements liés à la souris**

Explication et fonctionnement des événements

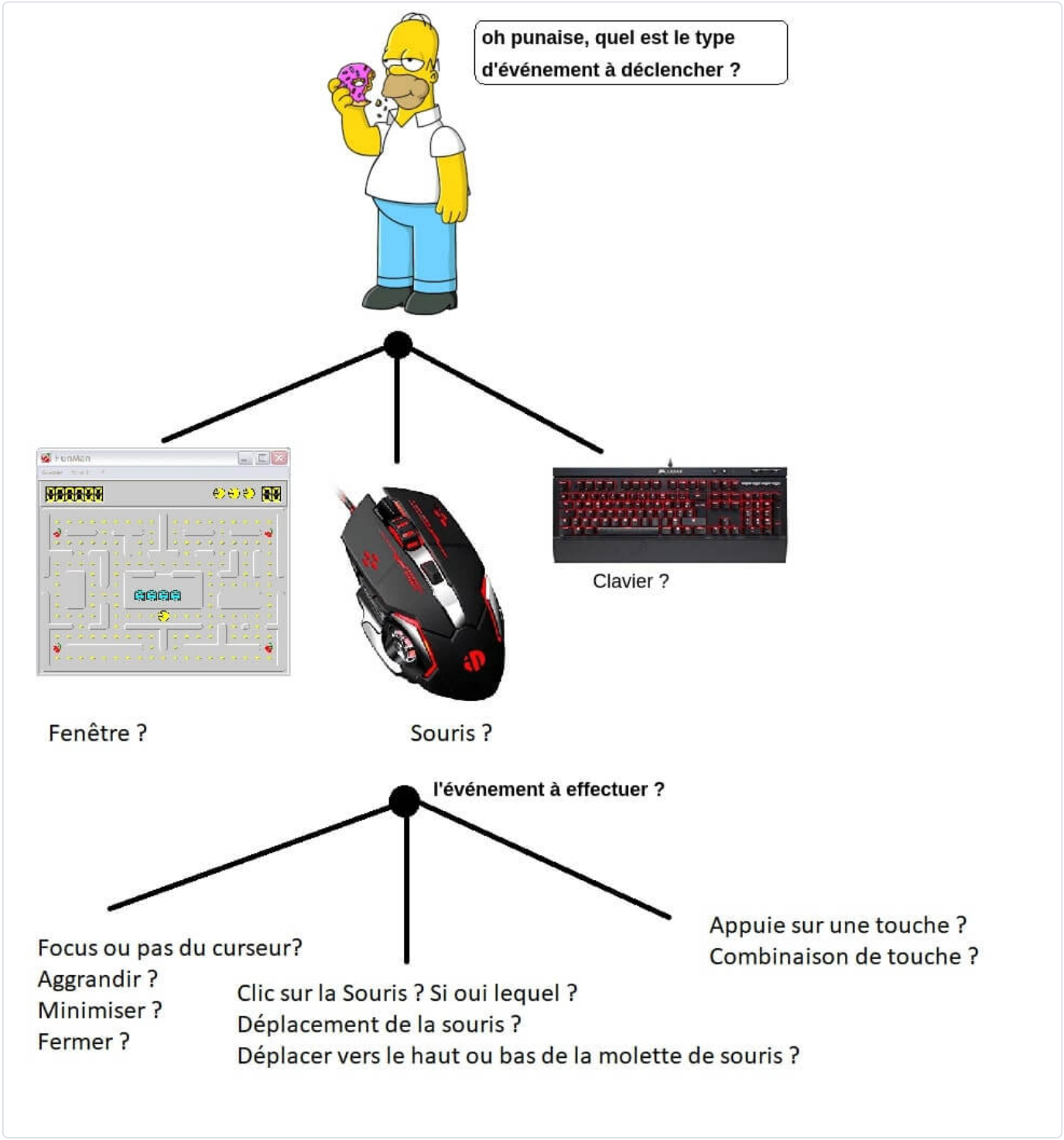
Alors là, sortez vos cerveaux, et prenez des notes !

Un événement CKWA ?



Dans ce gif, on peut remarquer un événement qui attend une action de l'utilisateur, plus précisément cet événement demande à l'utilisateur d'appuyer sur une touche du clavier afin de poursuivre la manipulation du programme. La légende raconte qu'Homer appuyait sur une touche qui envoie "J'adore le Duff" à ses amis depuis GMAIL.

Voici un schéma qui vous décrit le workflow que nous allons reproduire plus tard en code quand nous aborderons **les différents types de gestion des événements**.



Nous avons donc une première étape qui consiste à vérifier le type d'événement, et une seconde étape qui comporte le type d'action(s) à capturer.

Gestion des événements

Il existe deux types d'événements :

- **Événements bloquant** : ce sont les événements qui mettent entièrement votre programme en "pause" jusqu'à recevoir une action manuelle par l'utilisateur. Généralement quand vous débutez en programmation, vous apprenez à réaliser les programmes en mode console, où on vous apprend communément à coder une saisie utilisateur en effectuant par exemple un `scanf()` en langage C ou `input()` en langage Python, dans ce cas vous effectuez une action bloquante.
- **Événements non bloquants** : votre programme continuera à fonctionner peu importe si vous effectuez une action ou pas. Par exemple dans quasiment tous les jeux, votre ennemie continuera toujours à se déplacer peu importe si vous appuyez sur les touches de déplacements ou pas. Pour information, ça reste le type d'événement le plus couramment utilisé.

Il existe deux fonctions qui gèrent c'est deux manières de faire :

Nous avons tout d'abord la fonction `SDL_WaitEvent()` qui gère les événements bloquants, son prototype est assez simple :

```
int SDL_WaitEvent(SDL_Event* event)
```

Elle retourne 1 si elle a réussi sinon 0 s'il y a eu une erreur.

Par la suite, nous avons la fonction `SDL_PollEvent()` qui gère les événements non bloquants, son prototype est le suivant :

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

Elle aussi retourne 1 si elle a réussi sinon 0 s'il y a eu une erreur.

On peut d'ores et déjà constater que ces deux fonctions retournent un entier, et prennent une adresse d'un `SDL_Event` en paramètre. Voici l'union qui représente le `SDL_Event`

```
typedef union SDL_Event
{
    Uint32 type;                    /**< Event type, shared with all events */
    SDL_CommonEvent common;         /**< Common event data */
    SDL_DisplayEvent display;       /**< Window event data */
    SDL_WindowEvent window;         /**< Window event data */
    SDL_KeyboardEvent key;          /**< Keyboard event data */
    SDL_TextEditingEvent edit;      /**< Text editing event data */
    SDL_TextInputEvent text;        /**< Text input event data */
    SDL_MouseMotionEvent motion;    /**< Mouse motion event data */
    SDL_MouseButtonEvent button;    /**< Mouse button event data */
    SDL_MouseWheelEvent wheel;      /**< Mouse wheel event data */
    SDL_JoyAxisEvent jaxis;         /**< Joystick axis event data */
    SDL_JoyBallEvent jball;         /**< Joystick ball event data */
    SDL_JoyHatEvent jhat;           /**< Joystick hat event data */
    SDL_JoyButtonEvent jbutton;     /**< Joystick button event data */
    SDL_JoyDeviceEvent jdevice;     /**< Joystick device change event data */
    SDL_ControllerAxisEvent caxis;      /**< Game Controller axis event data */
    SDL_ControllerButtonEvent cbutton;  /**< Game Controller button event data */
    SDL_ControllerDeviceEvent cdevice;  /**< Game Controller device event data */
    SDL_AudioDeviceEvent adevice;   /**< Audio device event data */
    SDL_SensorEvent sensor;         /**< Sensor event data */
    SDL_QuitEvent quit;             /**< Quit request event data */
    SDL_UserEvent user;             /**< Custom event data */
    SDL_SysWMEvent syswm;           /**< System dependent window event data */
    SDL_TouchFingerEvent tfinger;   /**< Touch finger event data */
    SDL_MultiGestureEvent mgesture; /**< Gesture event data */
    SDL_DollarGestureEvent dgesture; /**< Gesture event data */
    SDL_DropEvent drop;             /**< Drag and drop event data */

    /* This is necessary for ABI compatibility between Visual C++ and GCC
       Visual C++ will respect the push pack pragma and use 52 bytes for
       this structure, and GCC will use the alignment of the largest datatype
       within the union, which is 8 bytes.

       So... we'll add padding to force the size to be 56 bytes for both.
    */
    Uint8 padding[56];
} SDL_Event;
```

Vous voyez que cette union se compose de plusieurs types d'événement comme `SDL_WindowEvent` pour gérer les événements liés à la fenêtre ou bien `SDL_KeyboardEvent` pour gérer les événements clavier. Nous n'allons pas tous les

voir, sinon je peux vous assurer que l'article sera très long et je n'ai pas aussi en ma possession le matériel nécessaire comme par exemple la manette. **Nous découvrirons que les types d'événement les plus importants** ! d'ailleurs, je vous rassure que vous n'aurez pas besoin de tous les connaître ! Cependant pour les plus curieux d'entre vous, voici la documentation où vous retrouverez tous les types de `SDL_Event` http://wiki.libsdl.org/SDL_Event.

Ah oui aussi, pour cet article, nous allons nous intéresser à la méthode 2 c'est-à-dire à la méthode non bloquante. À vous de voir ensuite selon votre projet et en fonction de vos besoins quelle méthode utiliser.

Les événements de fenêtre

Nous revoilà, pour aborder les événements liés à la fenêtre, nous allons utiliser la structure `SDL_WindowEvent` qui est un type de `SDL_Event`.

Voici donc le contenu de la structure

```
typedef struct SDL_WindowEvent
{
    Uint32 type;        /**< ::SDL_WINDOWEVENT */
    Uint32 timestamp;   /**< In milliseconds, populated using SDL_GetTicks() */
    Uint32 windowID;    /**< The associated window */
    Uint8 event;        /**< ::SDL_WindowEventID */
    Uint8 padding1;
    Uint8 padding2;
    Uint8 padding3;
    Sint32 data1;       /**< event dependent data */
    Sint32 data2;       /**< event dependent data */
} SDL_WindowEvent;
```

Nous devons vérifier le type d'événements avec l'union `SDL_Event`, pour cela il suffit de l'initialiser avec le code ci-dessous :

```
SDL_Event events; // Je crée une union de type SDL_Event
```

Une fois cela fait, nous devons par la suite decider quelle sera la méthode à privilégier ? Une gestion des événements de type bloquants ou non bloquants ? Comme précisé, nous allons **manipuler la méthode non bloquante**, notre choix sera donc porté sur la fonction `SDL_PollEvent`.

Jusqu'ici, selon ce que nous avons appris, nous aurons le code suivant :

```
SDL_Event events;
SDL_bool run = SDL_TRUE;

while (run) {
    while (SDL_PollEvent(&events)) {

    }
}
```

Autant vous dire tout de suite, que si vous exécutez le code, vous aurez le problème suivant : "**comment on fait pour quitter la boucle**" ? L'approche à suivre pour arriver à notre but, est de commencer d'abord par vérifier le type d'événement `SDL_Event` (souvenez-vous du schéma). Afin de connaître les valeurs possibles, je vous suggère de chercher les valeurs disponibles via l'énumération `SDL_EventType` (plus d'informations ici). Il y en a beaucoup mais celui qui nous intéresse le plus c'est le type `SDL_WINDOWEVENT`, qui comme son nom l'indique correspond aux événements liés à la fenêtre.

Reprenons notre code en vérifiant le type de `SDL_Event`, nous aurons ainsi le code suivant :

```
SDL_Event events;
SDL_bool run = SDL_TRUE;

while (run) {
    while (SDL_PollEvent(&events)) {
        switch(events.type){
        case SDL_WINDOWEVENT:
            break;
        }
```

```
    }
}
```

Comme vous pouvez le voir, je gère le type d'événement avec le mot-clé `switch`. Ce n'est pas une obligation ! Donc libre à vous de gérer vos conditions comme bon vous semble. Nous n'allons pas encore exécuter le code car il nous manque encore quelques instructions. Si on reprend notre schéma publié plus haut, il va falloir donc **vérifier quels seront les types d'actions à effectuer**. Dans notre cas, on souhaite quitter la fenêtre, il faut donc regarder dans la section *"Window events"* de la documentation, plus précisément la structure `SDL_WindowEvent` qui permet de **gérer les changements d'état de la fenêtre**. Elle peut prendre les valeurs suivantes :

```
typedef enum
{
    SDL_WINDOWEVENT_NONE,           /**< Never used */
    SDL_WINDOWEVENT_SHOWN,          /**< Window has been shown */
    SDL_WINDOWEVENT_HIDDEN,         /**< Window has been hidden */
    SDL_WINDOWEVENT_EXPOSED,        /**< Window has been exposed and should be
                                         redrawn */
    SDL_WINDOWEVENT_MOVED,          /**< Window has been moved to data1, data2
                                     */
    SDL_WINDOWEVENT_RESIZED,        /**< Window has been resized to data1xdata2 */
    SDL_WINDOWEVENT_SIZE_CHANGED,   /**< The window size has changed, either as
                                         a result of an API call or through the
                                         system or user changing the window size. */
    SDL_WINDOWEVENT_MINIMIZED,      /**< Window has been minimized */
    SDL_WINDOWEVENT_MAXIMIZED,      /**< Window has been maximized */
    SDL_WINDOWEVENT_RESTORED,       /**< Window has been restored to normal size
                                         and position */
    SDL_WINDOWEVENT_ENTER,          /**< Window has gained mouse focus */
    SDL_WINDOWEVENT_LEAVE,          /**< Window has lost mouse focus */
    SDL_WINDOWEVENT_FOCUS_GAINED,   /**< Window has gained keyboard focus */
    SDL_WINDOWEVENT_FOCUS_LOST,     /**< Window has lost keyboard focus */
    SDL_WINDOWEVENT_CLOSE,          /**< The window manager requests that the window
    SDL_WINDOWEVENT_TAKE_FOCUS,     /**< Window is being offered a focus (should SetW
    SDL_WINDOWEVENT_HIT_TEST        /**< Window had a hit test that wasn't SDL_HITTES
} SDL_WindowEventID;
```

En ce qui nous concerne, nous utiliserons la valeur `SDL_WINDOWEVENT_CLOSE` afin de quitter notre fenêtre.

```
SDL_Event events;
SDL_bool run = SDL_TRUE;

while (run) {
    while (SDL_PollEvent(&events)) {
        switch(events.type){
            case SDL_WINDOWEVENT:
                if (events.window.event == SDL_WINDOWEVENT_CLOSE)
                    run = SDL_FALSE;
                 break;
            }
    }
    SDL_RenderClear(renderer);
    SDL_RenderPresent(renderer);
}
```

Ouf, on peut enfin fermer notre fenêtre .

Les événements de clavier

Le même principe vu dans les événements liés à la fenêtre sera réutilisé pour les autres types événements. En l'occurrence ça tombe bien car c'est le cas aussi pour les événements clavier ! Nous allons vérifier les types d'événements décrits dans la section *"Keyboard events"* de la documentation. On peut donc remarquer les types d'événements suivants :

```
SDL_KEYDOWN        = 0x300, /**< Key pressed */
SDL_KEYUP,                  /**< Key released */
SDL_TEXTEDITING,            /**< Keyboard text editing (composition) */
SDL_TEXTINPUT,              /**< Keyboard text input */
SDL_KEYMAPCHANGED,          /**< Keymap changed due to a system event such as an
                                 input language or keyboard layout change.
```

Ces types d'événements permettent de gérer tous les événements liés au clavier. Dans cet exemple nous allons étudier le type `SDL_Keydown` et `SDL_Keyup`. Leur nom est assez explicite, d'un côté nous avons Keydown pour les **touches enfoncées** et Keyup pour les **touches relâchées**. Il est temps de les tester non ?

```
SDL_Event events;
SDL_bool run = SDL_TRUE;

while (run) {
    while (SDL_PollEvent(&events)) {
        switch(events.type){
            case SDL_WINDOWEVENT:
                if (events.window.event == SDL_WINDOWEVENT_CLOSE)
                    run = SDL_FALSE;
                 break;
            case SDL_KEYDOWN: // Un événement de type touche enfoncée est effectué
                SDL_Log("+key");
                break;
            case SDL_KEYUP: // Un événement de type touche relâchée est effectué
                SDL_Log("-key");
                break;
            }
    }
    SDL_RenderClear(renderer);
    SDL_RenderPresent(renderer);
}
```

Maintenant ce qui nous reste à faire, c'est de vérifier les actions à effectuer en **vérifiant quelle touche est appuyée par votre clavier**. Pour les touches claviers, on peut retrouver le :

- **Scancode** : correspond à l'**emplacement physique de la touche sur le clavier** ! C'est-à-dire que la touche "Z" et "W" auront le même scancode
- **Keycode** : signifie la **touche écrite sur le clavier** ! Dans ce cas que la touche "Z" et "W" n'auront pas le même keycode car la lettre est différente !

Vous retrouverez **les différentes valeurs possibles du scancode et du keycode** ici.

Pour mieux comprendre la différence entre le keycode et le scancode, nous allons les utiliser dans le code suivant :

```
while (run) {
    while (SDL_PollEvent(&events)) {
        switch(events.type){
        case SDL_WINDOWEVENT:
            if (events.window.event == SDL_WINDOWEVENT_CLOSE)
```

```
                run = SDL_FALSE;
                break;
        case SDL_KEYDOWN: // Un événement de type touche enfoncée est effectué
            SDL_Log("+key");  // Affiche un message +key (pour dire qu'on appuie sur

            if (events.key.keysym.scancode == SDL_SCANCODE_W) // Regarde si le scanc
                SDL_Log("Scancode W"); // Affiche un message

            if (events.key.keysym.sym == SDLK_w) // Regarde si le keycode w est enfo
                SDL_Log("Keycode W"); // Affiche un message

            if (events.key.keysym.sym == SDLK_z) // Regarde si on appuyer sur la touc
                SDL_Log("Keycode Z"); // Affiche un message

            break;
        case SDL_KEYUP:// Un événement de type touche relâchée
            SDL_Log("-key");
            break;
        }
    }
    SDL_RenderClear(renderer);
    SDL_RenderPresent(renderer);
}
```

Si vous exécutez le code, vous pourrez constater la différence entre le keycode et le scancode . Le scancode est l'emplacement de la touche sur le clavier qui peut être utilisé dans des cas pratiques comme un jeu de combat se jouant sur un clavier, le scancode vous aidera plus facilement à gérer les touches claviers de type azerty ou qwerty (ou autre) peu importe l'emplacement des touches de ces claviers. Là où le keycode risque de poser problème sur l'emplacement des touches et donc vous serez obligé de gérer chaque touche du clavier indépendamment.

Nous venons de faire le tour sur les événements clavier, passons maintenant aux événements souris.

les événements de souris

Dans la documentation, si vous regardez dans la section *"Mouse events"*, vous remarquerez **les types d'événements de souris** suivant :

- `SDL_MOUSEMOTION` : mouvement de souris
- `SDL_MOUSEBUTTONDOWN` : clique de souris enfoncé
- `SDL_MOUSEBUTTONUP` : clique de souris relâché
- `SDL_MOUSEWHEEL` : déroulement de la molette

Sans perdre de temps, commençons par gérer les quatre types d'événements :

```
while (run) {
    while (SDL_PollEvent(&events)) {
        switch(events.type){
        case SDL_WINDOWEVENT:
            if (events.window.event == SDL_WINDOWEVENT_CLOSE)
                run = SDL_FALSE;
                break;
        case SDL_KEYDOWN:
            SDL_Log("+key");

            if (events.key.keysym.scancode == SDL_SCANCODE_W)
                SDL_Log("Scancode W");

            if (events.key.keysym.sym == SDLK_w)
                SDL_Log("Keycode W");

            if (events.key.keysym.sym == SDLK_z)
                SDL_Log("Keycode Z");

            break;
        case SDL_KEYUP:
            SDL_Log("-key");
            break;
        case SDL_MOUSEMOTION: // Déplacement de souris
            SDL_Log("Mouvement de souris");
            break;
        case SDL_MOUSEBUTTONDOWN: // Click de souris
            SDL_Log("+clic");
            break;
        case SDL_MOUSEBUTTONUP: // Click de souris relâché
            SDL_Log("-clic");
            break;
        case SDL_MOUSEWHEEL: // Scroll de la molette
            SDL_Log("wheel");
            break;
        }
    }
    SDL_RenderClear(renderer);
    SDL_RenderPresent(renderer);
```

```
}
```

Nous allons nous intéresser plus précisément au type `SDL_MOUSEMOTION` de manière à **récupérer la position X et Y lors du mouvement de la souris**. Pour cela dans la structure `SDL_Event` nous avons un champ `motion` qui a pour type une structure `SDL_MouseMotionEvent`, et voici à quoi elle ressemble :

```
typedef struct SDL_MouseMotionEvent
{
    Uint32 type;        /**< ::SDL_MOUSEMOTION */
    Uint32 timestamp;   /**< In milliseconds, populated using SDL_GetTicks() */
    Uint32 windowID;    /**< The window with mouse focus, if any */
    Uint32 which;       /**< The mouse instance id, or SDL_TOUCH_MOUSEID */
    Uint32 state;       /**< The current button state */
    Sint32 x;           /**< X coordinate, relative to window */
    Sint32 y;           /**< Y coordinate, relative to window */
    Sint32 xrel;        /**< The relative motion in the X direction */
    Sint32 yrel;        /**< The relative motion in the Y direction */
} SDL_MouseMotionEvent;
```

Il suffit donc de récupérer le champ `x` et `y` pour **connaître la position à chaque mouvement de la souris** !

```
case SDL_MOUSEMOTION:
    SDL_Log("Mouvement de souris (%d %d) (%d %d)", events.motion.x, events.motion.y,
break;
```

Concernant le type `SDL_MOUSEBUTTONUP` et `SDL_MOUSEBUTTONDOWN`, ça ne sera pas le champ `motion` mais bien le champ `button` qui est de type `SDL_MouseButtonEvent`, voilà à quoi ressemble la structure :

```
typedef struct SDL_MouseButtonEvent
{
    Uint32 type;        /**< ::SDL_MOUSEBUTTONDOWN or ::SDL_MOUSEBUTTONUP */
    Uint32 timestamp;   /**< In milliseconds, populated using SDL_GetTicks() */
    Uint32 windowID;    /**< The window with mouse focus, if any */
    Uint32 which;       /**< The mouse instance id, or SDL_TOUCH_MOUSEID */
    Uint8 button;       /**< The mouse button index */
    Uint8 state;        /**< ::SDL_PRESSED or ::SDL_RELEASED */
    Uint8 clicks;       /**< 1 for single-click, 2 for double-click, etc. */
    Uint8 padding1;
```

```
    Sint32 x;           /**< X coordinate, relative to window */
    Sint32 y;           /**< Y coordinate, relative to window */
} SDL_MouseButtonEvent;
```

On y voit qu'on peut récupérer la position x et y du clique effectué ! On peut aussi savoir si c'est un double clique grâce à l'attribut `clicks` (vaut 1 si c'est un simple clique et 2 si c'est un double clique). On peut aussi savoir si c'est un clique droit ou clique gauche à l'aide de l'attribut `button`.

D'ailleurs l'attribut `button` peut avoir les valeurs suivantes :

- `SDL_BUTTON_LEFT` : clique gauche
- `SDL_BUTTON_RIGHT` : clique droit
- `SDL_BUTTON_MIDDLE` : clique de molette
- `SDL_BUTTON_X1` : clique du bouton du pousse haut
- `SDL_BUTTON_X2` : clique du bouton du pousse bas

Généralement les deux derniers boutons ne sont disponibles que sur certaine souris, notamment les souris dites "Gamer", exemple en image :

Voici comment les exploiter :

```
case SDL_MOUSEBUTTONDOWN:
    SDL_Log("+clic");

    if (events.button.button == SDL_BUTTON_LEFT) // Clique gauche
        SDL_Log("+left");
    if (events.button.button == SDL_BUTTON_RIGHT) // Clique droit
        SDL_Log("+right");
    if (events.button.button == SDL_BUTTON_MIDDLE) // Clique de molette
```

```
        SDL_Log("+middle");

    if (events.button.button == SDL_BUTTON_X1) // Bouton de pousse
        SDL_Log("+mouse4");

    if (events.button.button == SDL_BUTTON_X2) // Bouton de pousse
        SDL_Log("+mouse4");
```

Maintenant il nous manque plus qu'à savoir si on donne un coup de molette vers le haut ou vers le bas ! Il faut à présent s'intéresser au champ `wheel` du `SDL_Event` qui est du type `SDL_MouseWheelEvent`.

```
typedef struct SDL_MouseWheelEvent
{
    Uint32 type;        /**< ::SDL_MOUSEWHEEL */
    Uint32 timestamp;   /**< In milliseconds, populated using SDL_GetTicks() */
    Uint32 windowID;    /**< The window with mouse focus, if any */
    Uint32 which;       /**< The mouse instance id, or SDL_TOUCH_MOUSEID */
    Sint32 x;           /**< The amount scrolled horizontally, positive to the right
    Sint32 y;           /**< The amount scrolled vertically, positive away from the u
    Uint32 direction;   /**< Set to one of the SDL_MOUSEWHEEL_* defines. When FLIPPED
} SDL_MouseWheelEvent;
```

Ici on peut **connaître la direction de la molette** avec les champs `x` et `y` !

```
case SDL_MOUSEWHEEL:
    if (events.wheel.y > 0)
        SDL_Log("up %d", events.wheel.y);
    else if (events.wheel.y < 0)
        SDL_Log("down %d", events.wheel.y);
    break;
}
```

Conclusion

Vous l'aurez remarqué, la gestion des événements reste assez répétitive, mais au moins nous avons fait le tour sur les principaux types des événements. Rendez-vous sur un nouvel épisode, où nous étudierons le module AUDIO de la SDL et ses extensions !