Un grand classique dans les questions des débutants Qt (qui veulent tous créer des jeux vidéos... je leur jette pas la pierre, je suis pareil :D) : comment utiliser un gamepad avec Qt. La réponse en général est qu'on ne peut pas. Il faut utiliser une bibliothèque externe pour cela, par exemple la SFML.

L'objectif va donc être d'ajouter ces fonctionnalités dans QWindow en utilisant le QPA (Qt Platform Abstraction), de créer un élément QML pour gérer le gamepad dans Qt Quick, de créer un élément graphique pour proposer un gamepad virtuel sur écran tactile (mobile, tablette). Je vais rédiger plusieurs articles :

1. implémentation pour Windows
2. implémentation des éléments QML
3. implémentation pour Linux
4. implémentation pour Android (OUYA)
5. implémentation de la version QPA
6. performances

Introduction

Pour commencer, je dois avouer une chose : je n'avais jamais utilisé un gamepad en C++ avant de commencer cette série d'articles. Donc il me fallait une base de travail. Comme je suis actuellement sous Windows, je vais faire simple, je vais m'intéresser dans un premier temps à l'implémentation sur cette plateforme, on verra pour Linux et Android ensuite (je n'ai pas de Mac OS X, j'aurais besoin d'un coup de main pour le portage dessus). Donc la chose la plus simple à faire est de demander à Microsoft : MSDN - Dev Center - Multimedia Input - Joysticks.

Le second point important est comment implémenter cela dans Qt. Plusieurs approches sont possibles :

- créer un QObject et utiliser les signaux et slots pour obtenir des informations sur le gamepad. Simple et efficace, je ferais mes tests avec cette approche. Cela permettra également aux

utilisateurs des anciennes versions de Qt de pouvoir utiliser le gamepad.
- ajouter la gestion du gamepad directement dans la boucle d'événements en utilisant le QPA. Il faudra donc créer une classe QGamepadEvent et ajouter la fonction gamepadUpdateEvent dans QWindow.

```
class QWindow {
protected:
    virtual void gamepadUpdateEvent(QGamepadEvent* event);
};
```

Peut être faire comme pour la souris et créer plusieurs événements : gamepadPressEvent et gamepadReleaseEvent (pour les boutons du gamepad) et gamepadMoveEvent (pour le déplacement des sticks) ?

Gestion du gamepad sous Windows

Pour gérer les gamepad, il faut :

- récupérer la liste des gamepads et leurs fonctionnalités (capacities) disponibles.
- s'informer régulièrement des l'états des boutons.
- ou ajouter des fonctions callback pour que le gamepad informe l'application lorsqu'il a des mises à jour des valeurs.

Comme l'objectif est d'émettre un signal ou un événement lors d'une action de l'utilisateur sur le gamepad, les fonctions callback semblent être la meilleure approche a priori. Cependant, ces fonctions callback ne sont disponibles que pour les fonctions de base du gamepad (josGetPos). Et malheureusement, la gestion d'un gamepad complet (plusieurs sticks numériques, plusieurs boutons) n'est possible qu'avec les fonctions étendues (josGetPosEx).

Donc, il va falloir opérer différemment. Il faut créer un timer qui lit l'état du gamepad à intervalle régulier et le compare à l'état précédent. Si une

valeur a changée, on émet un signal ou un event.

Détecter les gamepads connectés

Il est possible de gérer jusqu'à 16 gamepads en même temps avec les pilotes Windows. Il faut inclure le fichier d'en-tête window.h et la bibliothèque Winmm.dll. Chaque gamepad physique est identifié par un ID de type unsigned int. Il faut également gérer les gamepads virtuels, donc on va réserver les valeurs d'ID inférieures à 256 pour les gamepads physiques.

Pour connaître le nombre de gamepads physiques possibles, il faut utiliser la fonction joyGetNumDevs :

```
unsigned int maxGamepadId = joyGetNumDevs();
```

Pour tester si un gamepad est connecté, il faut utiliser la fonction joyGetPosEx avec l'identifiant en paramètre. Cette fonction retourne la valeur JOYERR_UNPLUGGED si le gamepad n'est pas connecté.

```
JOYINFOEX info;
for (unsigned int id = 0; id < maxGamepadId; ++id) {
    MMRESULT result = joyGetPosEx(id, &info);
    bool isConnected = result == JOYERR_NOERROR ;
    cout << "gamepad #" << id <<
        (isConnected ? " connected" : " not connected") <<
endl;
}
```

Pour commencer, on va tester la connexion de tous les gamepads à chaque update du timer. À terme, il faudra tester le coût de ces tests en continu et voir s'il faut optimiser (faire un test des connexions moins souvent que le test des changements de valeur).

Récupérer les informations d'un gamepad

Les informations relatives au gamepad sont contenues dans le paramètre

de type JOYINFOEX de la fonction joyGetPosEx. Il faut conserver cette information et la comparer à chaque update.

```
JOYINFOEX info, lastInfo;
if (info.dwXpos != lastInfo.dwXpos)
    cout << "X position modified";
```

Les différents sticks et boutons sont identifiés avec des lettres et des chiffres. Il n'est pas possible de nommer explicitement ces éléments puisqu'ils peuvent changer d'une manette à l'autre. L'image suivante présente la correspondance des identifiants sur une manette XBox. Je détaillerai ça dans un prochain article.

Classes Qt de gestion de gamepad

Pour commencer, l'objectif sera d'ajouter un fonction gamepadUpdateEvent dans la classe QWindow. En attendant, l'utilisateur devra créer lui-même cette fonction, en surchargeant la fonction event pour récupérer les événements du gamepad.

Modifications dans la classe QWindow

Par exemple, pour créer cette fonction dans une classe MyWindow :

```
class MyWindow : public QWindow {
    Q_OBJECT
protected:
    bool event(QEvent* event) override;
    virtual bool gamepadUpdateEvent(GamepadEvent* event);
};
```

Dans la fonction event, il faut tester le type d'événement. Si celui-ci correspond à un événement du gamepad, on appelle la fonction gamepadUpdateEvent. Sinon on lance le traitement standard des événements en appelant la fonction QWindow::event.

```
bool GamepadWidget::event(QEvent* event)
{
    if (event->type() == static_cast<QEvent::Type>(
            GamepadEvent::GamepadUpdate))
        return gamepadUpdateEvent(static_cast<GamepadEvent*>
(
            event));
    else
        return QWidget::event(event);
}

bool GamepadWidget::gamepadUpdateEvent(GamepadEvent* event)
{
    qDebug() << "receive gamepadUpdateEvent";
}
```

La classe GamepadEvent

La classe GamepadEvent dérive de QEvent et définit un nouveau type d'événement. La définition des fonctionnalités de base de cette classe est la suivante :

```
class GamepadEvent : public QEvent
```

```
{
public:
    static const QEvent::Type GamepadUpdate;

    explicit GamepadEvent(QEvent::Type type);
    virtual ~GamepadEvent();
};
```

Le type GamepadUpdate est un identifiant qui est initialisé avec la fonction statique registerEventType. Pour le constructeur, il faut passer le type d'événement en paramètre de QEvent.

```
const QEvent::Type GamepadEvent::GamepadUpdate =
    static_cast<QEvent::Type>(QEvent::registerEventType());

GamepadEvent::GamepadEvent(QEvent::Type type) :
    QEvent(type)
{
}

GamepadEvent::~GamepadEvent()
{
}
```

Cette classe doit conserver les informations sur le gamepad. Suivant les règles de codage de Qt, ces informations seront dans une classe private en utilisant l'idiome Pimpl (pointeurs D et Q). Ces informations sont les suivantes :

```
unsigned int id() const;
float joystickX() const;
float joystickY() const;
float joystickZ() const;
float joystickR() const;
float joystickU() const;
float joystickV() const;
float joystickPov() const;
unsigned int pressedButtons() const;
```

Id correspond à l'identifiant du gamepad, les valeurs réelles sont comprises entre -1 et 1 et correspondent aux boutons analogiques du

gamepad (sticks, gachettes), sauf la valeur POV (point-of-view) qui est comprise entre 0 et 360 degrés. Les boutons correspondants à une combinaison de valeurs qu'il faut tester avec l'opérateur binaire AND :

```
bool isButtonAPressed =
    Gamepad::Button1 & event->pressedButtons();
```

La classe GamepadEventPrivate contient les valeurs qui sont lues par ces fonctions :

```
class GamepadEventPrivate
{
};
```

Le pointeur D est déclaré dans la classe GamepadEvent :

```
class GamepadEventPrivate; // déclaration anticipée
class GamepadEvent : public QEvent
{
private:
    class GamepadEventPrivate* d;
};
```

Ce pointeur doit être instancié dans le constructeur. Il pourra ensuite être utilisé par les fonctions de GamepadEvent.

```
GamepadEvent::GamepadEvent(QEvent::Type type) :
    QEvent(type)
{
    d = new GamepadEventPrivate;
}

GamepadEvent::~GamepadEvent()
{
    delete d;
}
```