



Université de la Rochelle - Licence 3 Informatique Programmation Évènementielle

TP 5 - Langage QML et Qt Quick

© B. Besserer, R. Péteri

Année universitaire 2018-2019

Préambule

Ce TP porte sur une manière déclarative de concevoir facilement des interfaces avec QtQuick et QML.

En plus du cours sous Moodle, on pourra donc se référer à site de Qt concernant Qt Quick et l'intégration de QML dans du code C++ et Python : <https://doc.qt.io/qt-5/qmlapplications.html>. N'hésitez pas à regarder d'autres sources suivant vos besoins.

Ce TP sera réalisé sous la VM Linux avec Ubuntu 2018. Pour réaliser une installation équivalente sur les ordinateurs personnels, utiliser Qt Creator basée sur Qt > 5.7 (voir compatibilité ici : <https://doc.qt.io/qt-5/qtquickcontrols-index.html#versions>)


1 QML et Qt Quick

1.1 Préparation

Le TP est fait pour être réalisé avec l'IDE QtCreator et Qt5.

Ouvrir QtCreator, créer un nouveau projet et choisissez : **Other Project > Qt Quick UI Prototype**. Nommez le projet **TP5_exo1** puis choisissez comme version minimale requise pour Qt : **Qt 5.7**

1.2 Un premier programme en QML

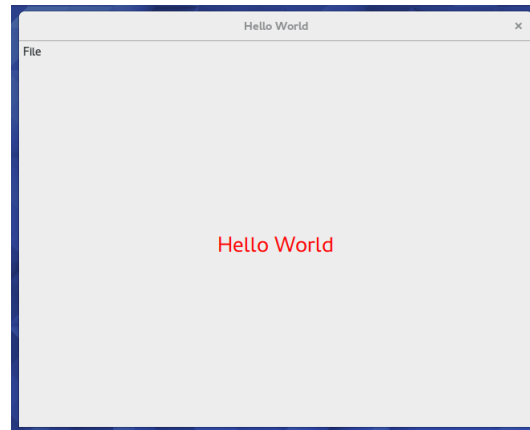
Exécuter tout d'abord le programme (run : ) pour vérifier que l'interface par défaut (une fenêtre **Hello World**) est bien créée. Le fichier **TP5_exo1.qml** comprend la description de l'IHM avec le langage QML.

Remplacez ensuite le code dans votre fichier **TP5_exo1.qml** par celui contenu dans le fichier **TP5_exo1_base.qml** disponible sous Moodle.

Déformais, la fenêtre prédéfinie est de type **ApplicationWindow** (équivalent de **QMainWindow** avec barre de menu et statut), les éléments sont dessinés les uns après les autres (attention aux éléments écrasés!!!). Créer un composant **Text** dans la fenêtre principal en mettant le code ci-dessous :

```
Text{
    text: qstr("Hello World")
    color: "red"
    font.pointSize: 20
    anchors.horizontalCenter: parent.horizontalCenter
    anchors.verticalCenter: parent.verticalCenter
}
```

Vous devriez désormais avoir ceci :



Remplacer le texte « Hello World » par le texte « Open clicked » *lors d'un clic sur **Open** du menu **File***. Aide : ajouter un identifiant au composant Text pour pouvoir y accéder dans la propriété `onTriggered` du composant MenuItem correspondant (`onTriggered: identifiant.text = "MonTexte";`).

1.2.1 Dessiner des formes

Effacer le texte « Hello world », puis créer, dans le composant **Frame**, deux rectangles de taille 200×300 pixels et espacés de 10 pixels, définis comme suit :

```
Rectangle{
    id : monRectangle
    width: 200
    height: ...
    anchors.left: ...
    anchors.leftMargin: ...
}
```

1.2.2 Insertion d'images

Télécharger et décompresser dans le répertoire de votre projet l'archive `img.zip` disponible sous Moodle.

On souhaite remplacer la couleur de fond des rectangles par les images `img1.jpg` et `img2.jpg`.

Conseils : placer un composant Image à l'intérieur de chaque composant **Rectangle** et utiliser `anchors.fill: parent` pour que l'image remplisse la surface du composant parent.

```

Rectangle{
    ...
    Image{
        id: ...
        anchors.fill: parent
        source:"CheminRelatifVersMonImage.jpg"
    }
}

```

1.2.3 Insertion de texte

Ajouter le texte « Image 1 » et « Image 2 » centré en haut du rectangle 1 et 2 respectivement. Le texte sera en taille 20 et de couleur blanche.

Aide :

insérer un composant `Text` dans chaque composant `Rectangle` et utiliser les propriétés : `anchors.horizontalCenter: parent.horizontalCenter` et `font.pointSize`. La couleur peut être définie par son code hexadécimal.

1.2.4 Insertion de boutons

Insérer un bouton au centre de chaque rectangle contenant le texte "Changer".

1.2.5 Interaction avec la souris

On souhaite diviser la transparence des rectangles par 2 lors de survol de la souris. Pour cela, utiliser le composant `MouseArea` à l'intérieur des composants `Rectangle` (on regardera l'aide des signaux `entered()` et `exited()`).



1.2.6 Boîte de dialogue

Créer un composant `FileDialog` (boîte de dialogue) pour permettre à l'utilisateur de sélectionner un fichier de type image sur le disque lors d'un clic sur les boutons "Changer". Aide : ajouter la propriété `onClicked : maBoiteDeDialogue.open()` aux boutons pour lancer l'action sur événement du clic. On mettra aussi dans la propriété `folder` le répertoire par défaut.

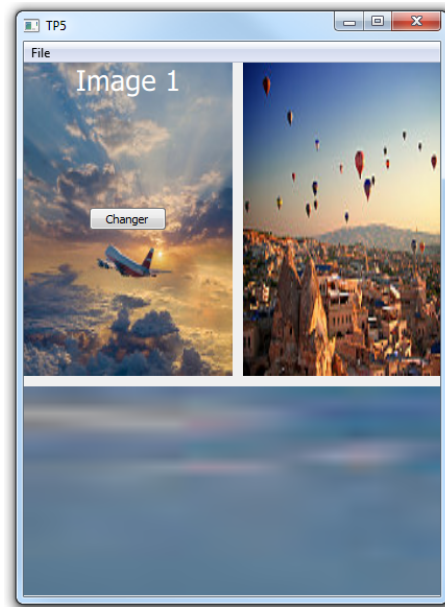
On pourra regarder l'aide en ligne : <http://doc.qt.io/qt-5/qml-qtquick-dialogs-filedialog.html>

1.2.7 Mise à jour de propriétés

Après sélection de la nouvelle image via la boîte de dialogue, elle doit venir remplacer l'image de fond du rectangle concerné.

Aide : utiliser la propriété `onAccepted` du composant `FileDialog` pour changer la source du composant `Image` correspondant (`onAccepted: img1.source = fileUrl`)

1.2.8 Déformation d'images



Les composants peuvent facilement devenir réactif au mouvement de la souris (ou doigt pour les interfaces tactiles) lorsqu'ils sont insérés dans un composant de type **Flickable**. On souhaite insérer une nouvelle image au-dessous des deux autres de manière à faire en sorte qu'elle soit plus grande que la zone du composant **Flickable**, et que l'on puisse ainsi *scroller* dedans. En suivant l'exemple ci-dessous, essayer de déplacer l'image en cliquant dessus avec la souris ou avec le pad.

```
Flickable{
    id:flickImg2
    anchors.top:...
    width:...
    height: ...
    contentWidth: img3.width
    contentHeight: img3.height
    clip:true
    Image{
        id:img3
        source:"img3.jpg"
    }
}
```

1.2.9 Intégration dans PyQt

Si vous travaillez sur la VM Linux de l'université, mettez à jour PyQt5 pour qu'il puisse utiliser les composants QtQuick.

Pour cela, dans un terminal, taper : `pip3 install --user PyQt5`

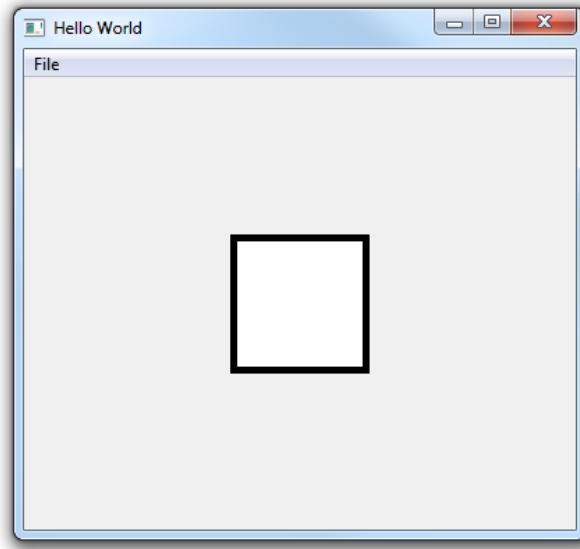
En vous référant au cours, intégrer ce composant dans du code Python, puis lancer le script (depuis le terminal ou votre IDE Python favori...)



2

2 QML et états

Dans un nouveau projet dans **Qt Designer**, créez un rectangle de taille 100x100 (*width*, *height*), avec un fond blanc (*color*), des bordures noires d'épaisseur 5 (*border.color*, *border.width*), et qui sera centré à l'intérieur de votre fenêtre (*anchors*). Voici ce que vous devriez voir :



Nous allons utiliser ce rectangle comme base pour définir des états QML et modifier son apparence en réponse à une interaction de l'utilisateur.

2.1 Les états

Un état est une abstraction des propriétés d'une collection d'éléments. Les interfaces utilisateurs sont conçues pour présenter différentes configurations d'interface dans différents scénarios ou pour modifier leur apparence en réponse à une interaction utilisateur. Souvent, un ensemble de changements est effectué simultanément, tel que l'interface puisse être vue comme changeant en interne d'un état à un autre.

Une des caractéristiques des états QML est la capacité d'un élément à retourner à son état par défaut.

On pourra se référer à : <http://qt.developpez.com/doc/4.7/qml-tutorial3/>.

In QML, states are a set of property configurations defined in a State element. Different configurations could, for example :

- Show some UI elements and hide others*
- Present different available actions to the user*
- Start, stop, or pause animations*
- Execute some script required in the new state*
- Change a property value for a particular item*
- Show a different view or screen*

All Item-based objects have a state property, and can specify additional states by adding new State objects to the item's states property. Each state within a component has a unique name, an empty string being the default. To change the current state of an item, set the state property to the name of the state.

2.2

Le code suivant permet la gestion d'un état « sélection » en réponse au clic gauche souris.

```
Rectangle {
    id: afficheur

    border.color: "black"

    color: "white"
    border.width: 5
    width: 100
    height: 100

    MouseArea {
        anchors.fill: parent

        acceptedButtons: Qt.LeftButton
        onClicked: {
            afficheur.state = 'selection';
        }
    }

    states: [
        State {
            name: "selection"
            PropertyChanges {
                target: afficheur
                .....
            }
        }
    ]
}
```

- Ajouter le code permettant la modification de la propriété par défaut de la couleur de la bordure du rectangle dont l'identifiant est « afficheur ».
- Modifier le code dans le composant **MouseArea** de façon à associer un état 'sélection' au clic gauche souris, et l'état par défaut sur un clic droit.
Aide : pour retourner à l'état par défaut : `id_composant.state=' '`;
La propriété `mouse.button` permet d'accéder au bouton associé au clic souris.



3

2.3

On veut maintenant agrandir le rectangle lorsqu'il est survolé par la souris. Le survol correspond à l'action « `onEntered:` » et « `onExited:` » de **MouseArea**, et doit être activé grâce à la propriété : « `hoverEnabled: true` ».

- Modifiez le code de façon à ajouter la gestion d'un nouvel état « agrandissement », sur l'action « `onEntered` » de `MouseArea`, et le retour à l'état initial sur l'action « `onExited` : » de `MouseArea`. On pourra par exemple pour tester modifier la taille du rectangle.

2.4

Les changements d'états peuvent être animés à travers des **transitions**. Le code suivant permet l'ajout d'une animation lors de la modification des attributs x ou y d'un élément.

```
transitions: [  
    Transition {  
        NumberAnimation { properties: "x,y"; duration: 500 }  
    }  
]
```

Ajoutez une animation lors de l'agrandissement du rectangle (on veut doubler la hauteur et la largeur du rectangle).

2.5

Par défaut, les propriétés non modifiées dans l'état ont comme valeur, la valeur par défaut du `Rectangle`. Il pourrait être intéressant notamment pour la sélection de conserver le changement de couleur lors du clic gauche de la souris.

Définir une propriété de type booléen permettant de stocker les modifications de la sélection.

Modifier dans chaque état les changements de propriétés couleur et taille en conséquence de façon à conserver la sélection sauf sur un clic droit qui appellera toujours l'état par défaut.

On pourra alternativement utiliser la propriété `restoreEntryValues`.

