Université de la Rochelle - Licence 3 Informatique
Programmation Évènementielle

TP 5 - QML et Qt Quick
Déploiement sur smarphone

Année universitaire 2025-2026

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 au 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 pourra être réalisé sous la VM Linux avec Ubuntu. Pour réaliser une installation équivalente sur les ordinateurs personnels, utiliser Qt Creator basée sur Qt $>$ 5.7 (voir compatibilité ici dans Version history : `https://en.wikipedia.org/wiki/Qt_Quick`).
Enfin, n'hésitez pas à revoir le cours QML, il vous sera d'une certaine utilité !

1 QML et Qt Quick

1.1 Préparation

Le TP est fait pour être réalisé avec `Qt5` et l'IDE `QtCreator`.
Ouvrir `QtCreator`, créer un nouveau projet et choisissez : `Autre Projet > Qt Quick UI Prototype`.
Nommez le projet `TP5_exo1` puis choisissez comme version minimale requise pour Qt : `Qt 5.12`.
Le fichier `tp5_part1_student.qml` correspond à la description initiale de l'IHM, en language QML, qu'il vous faudra compléter. Exécuter tout d'abord le programme pour vérifier que l'interface initiale (une fenêtre vide avec pour titre `ADMIN : Image Gallery` s'affiche) .
Un aperçu rapide en cours d'édition est possible (dans Qt Creator, menu Compiler -> QML Preview).

Créez un composant `Text` dans la fenêtre principale en mettant le code ci-dessous (et en remplaçant ADMIN par votre nom de famille) :

```
Text{
  text: "ADMIN : Image Gallery"
  color: "red"
  font.pointSize: 20
  font.bold: true
  anchors.horizontalCenter: parent.horizontalCenter
  anchors.top : parent.top
  anchors.topMargin: 50
}
```

Vérifiez que votre composant s'affiche (autocontrole). Pour la suite, n'oubliez pas que les composants sont créés dans l'ordre et peuvent se superposer.

1.2 Affichage d'image avec interaction

1. Ajoutez un composant de type Rectangle en dessous de ce texte :

```
Rectangle{
   id : monRectangle1
   width: 300
   height: 200 }
```

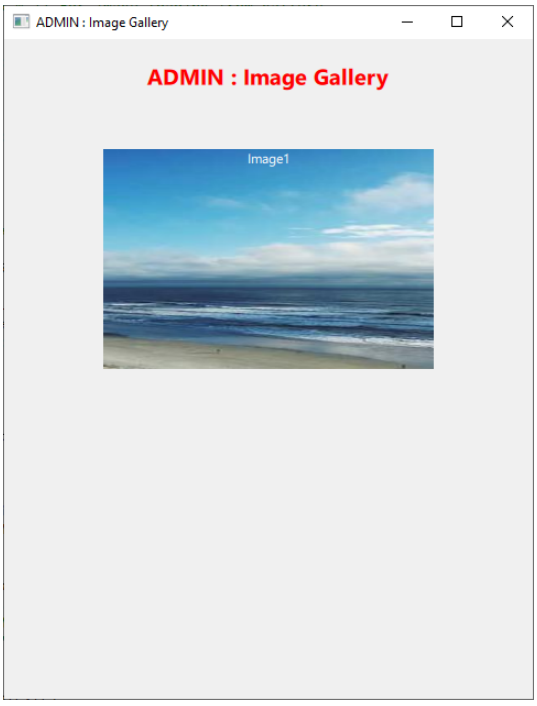
2. Dans ce rectangle, insérer une image avec le composant Image. Outre l'identifiant pour l'image, il faut au moins deux attributs :
   - `anchors.fill: parent` pour indiquer que l'image occupe tout l'espace du composant parent,
   - la source de l'image : soit une UR,L soit un chemin vers une image qui se trouve dans le système de fichier de votre ordinateur. Télécharger et décompresser dans le répertoire de votre projet l'archive `img.zip` disponible sous Moodle, qui contient diverses images pour ce TP.

```
Image{
  id : monImage1
  anchors.fill: parent
  source:"https://picsum.photos/200/300"  // ou  bien source:"CheminRelatifVersMonImage.jpg" }
```

3. Dans ce rectangle, par dessus l'image, ajouter une ligne de texte, en blanc, qui sera placé en haut de l'image (voir copie d'écran)

```
Text{
   id : monTexte1
   text:"Image1"
   color: ....
   .... // propriété pour le centrage horizontal }
```

4. Enfin, dans ce rectangle, ajouter une MouseArea qui permettra une interaction avec la souris (ou l'écran tactile)
   Pour vérifier que cette "zone de sensibilité" est bien active, on va changer une propriété lorsque votre curseur de souris survole l'image (ajoutez la propriété `hoverEnabled : true`) : **il faut que l'opacité de l'image soit divisée par 2 lors d'un survol de la souris, et revienne à sa valeur initiale quand on quitte la zone**
   (On regardera l'aide des signaux entered() et exited(), il s'agira donc de modifier la propriété de couleur dans les handler onEntered() et onExited()).



1 Vous devez avoir plus ou moins ce rendu, avec le changement d'opacité lors du survol de l'image. Faire valider.

1.3 Gestionnaire de géométrie et duplication des éléments

En dessous du texte "ADMIN : Image Gallery" (en ayant bien sur mis votre nom à la place de ADMIN), insérez un composant `Column` (gestionnaire de géométrie)
Dans ce composant, placez, de haut en bas :

— Le rectangle précédemment défini, `id:rectangle1`
— Un bouton poussoir,

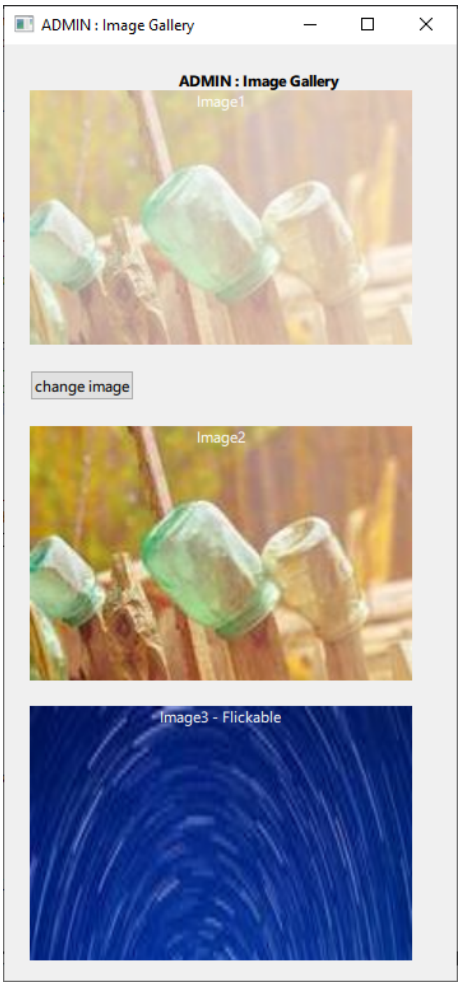


FIGURE 1 – Interface à reproduire

— Deux autres rectangles, en dupliquant le code utilisé pour le premier rectangle et en faisant attention à modifier les identifiants (un id doit être unique, donc `id:rectange2` et `id:rectangle3`), idem pour les identifiants des images, label texte, etc.... Attention, le placement des éléments doit être laissé au gestionnaire de géométrie !

1.3.1 Déformation d'images

Vérifiez que la fenêtre de votre application ressemble à la Figure 1 (autocontrole). Si c'est OK, on modifie le type du dernier composant : le type `Rectangle` est remplacé par le type `Flickable` . En changeant simplement `Rectangle` par `Flickabel`, l'interface doit s'afficher de la même façon.
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 modifie le changement de la dernière image pour faire en sorte qu'elle soit plus grande que la zone du composant `Flickable`, et que l'on puisse ainsi *scroller* **verticalement** dedans. En suivant l'exemple ci-dessous, essayer de déplacer l'image en cliquant dessus avec la souris ou avec le pad (voir vidéo sous Moodle).

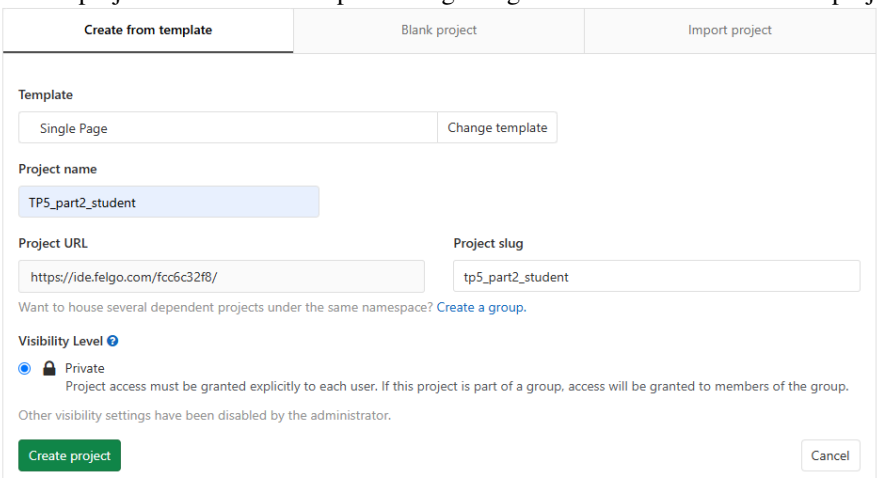
```
Flickable{
        id:flickImg2
        width:...
        height: ...
        contentWidth: width
        contentHeight: height*3
        clip:true
        Image{
            id:img3
            source:"..."
         }
    }
```

2 Intégration dans PyQt et déploiement sur smartphone

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

Pour tester le code QML sur smartphone :

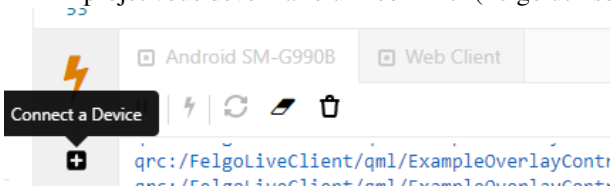
- Aller sur le site de Felgo, créer un compte sur Felgo. Cliquez sur Download (attention, au final on ne télécharge rien !) puis sur "Cloud IDE"
- Créez un projet en utilisant le template "Single Page". Pensez à renommer votre projet, voir copie d'écran.



- Lorsque le projet est créé, ouvrir ce projet dans l'IDE en ligne (bouton bleu "Web IDE")
- Remplacez le code de Main.qml par votre code. Attention aux directives import :

```
import Felgo 3.0
import QtQuick 2.0
import QtQuick.Controls 2.0
import QtQuick.Layouts 1.0
```

- L'aperçu de l'interface est visible à droite dans l'IDE.
- Pour le rendu de l'interface sur smartphone : rendez vous sur cette page : `https://felgo.com/resources/developer-app` et installer l'application en fonction de l'OS de votre smartphone (Android ou iOS)
- Lorsque vous lancez l'application installée sur votre smartphone, vous pouvez voir un identifiant. Vous devez associer votre téléphone au gestionnaire de projet "Web IDE" de Felgo avec cet identifiant (Connect a Device...) Si votre projet est sans erreur, vous devriez voir l'interface sur le smartphone. Après modification, un CTRL+S envie la version modifié sur le smartphone. Mais pour conserver vos modification dans votre projet vous devez faire un "commit" (Felgo utilise GitLab)



☑ 2 *Pour valider, il faudra reproduire l'aspect de Figure 1, lancer le script python, monter que l'image dans le rectangle 3 peut-être "scrollée". Puis montrer le rendu sur l'écran du smartphone*

2.1 Modes d'interaction

Revenez maintenant sur **Qt Creator** pour l'édition du code.
L'image dans chaque rectangle doit pouvoir être changée (voir vidéo) :

- Pour le composant du haut, c'est l'action sur le bouton qui provoque l'apparition de l'interface de sélection de l'image (une boite de dialogue si Desktop, accès à la galerie d'image si l'interface fonctionne sur smartphone). Ajouter la propriété `onClicked : maBoiteDeDialogue.open()` au bouton pour lancer l'action lors du clic.
- pour le second composant, c'est un clic qui provoque l'apparition de l'interface de sélection de l'image
- Le dernier composant doit rester "scrollable", c'est donc un "long press" (`onPressAndHold`)qui provoque l'apparition de l'interface de sélection de l'image

Commencer par créer un composant `FileDialog` (boîte de dialogue, id :maBoiteDeDialogue) pour permettre à l'utilisateur de sélectionner un fichier de type image sur le disque lors d'un clic sur le bouton "Change Image". Il

n'y aura qu'**un seul** composant composant `FileDialog`, et vous créez une propriété pour mémoriser l'émetteur de l'appel à la boite de dialogue.

Après sélection de la nouvelle image via la boîte de dialogue, elle doit venir remplacer l'image concernée.
Aide : utiliser la propriété `onAccepted` du composant `FileDialog` pour changer la source du composant `Image` correspondant (`onAccepted: image1.source = selectedFile`)
On pourra regarder l'aide en ligne : `http://doc.qt.io/qt-5/qml-qtquick-dialogs-filedialog.html`
Pour valider, il faudra montrer les 3 modes d'interaction (clic sur bouton, clic direct, long press) sur Desktop et sur smartphone, et montrer que l'image chargée dans le composant du bas reste scrolable.  3

3 QML et états

Pour illustrer l'utilisation des états (*state*) dans QML, on étudie un composant qui doit servir d'élément graphique de base (une "tuile") pour la réalisation d'un jeu type *memory* (le but du jeu est d'apparier des paires de cartes identiques après les avoir retournées). Ces tuiles devront avoir deux apparences :

- état (*state*) par défaut : la tuile est placé avec l'image non visible
- état (*state*) nommé "flipped" : la tuile est retournée et l'image est visible

Le code de base pour cette partie est disponible sous Moodle (TP5_part2_student.qml), ainsi qu'une vidéo de démonstration.
Dans un nouveau projet dans `Qt Creator`, utilisez le code donné. Mettre votre nom à la place de "ADMIN" , et complétez le composant rectangle pour qu'il soit de taille 80×80 avec un fond gris et une bordures d'épaisseur 2.
Nous allons utiliser ce rectangle comme base pour définir des états QML et modifier son apparence en réponse à une interaction de l'utilisateur.

3.1 Les états

Un état est une abstraction des propriétés d'une collection d'éléments. Le passage d'un état à un autre permet un ensemble de changement simultané de l'apparence de l'interface, avec éventuellement une animation. Une des caractéristiques des états QML est la capacité d'un élément à retourner à son état par défaut (`state = " "`).

Le code suivant permet de changer l'état lors du clic gauche souris.

```
Rectangle {
...
   Image {
   id: testImage1
   opacity: 0
   ...
   }
     MouseArea {
       anchors.fill: parent
       acceptedButtons: Qt.LeftButton  // par defaut, mais il faut completer pour accepter le clic droit par ex.
       onClicked: { testTile1.state = 'flipped'; }
    }

    states: [
        State {
            name: "flipped"
            PropertyChanges {
                target: testTile1
                        ..........................
            }
        }
    ]
}
```

Ajouter le code permettant la modification de la propriété par défaut de la couleur du rectangle (id : testTile1).

3.2 Animation

Les changements d'états peuvent être animés à travers des *transitions*. Le code suivant permet l'ajout d'une animation lorsqu'on passe d'un état à un autre :

```
transitions: [
          Transition {
              to:"flipped"
```

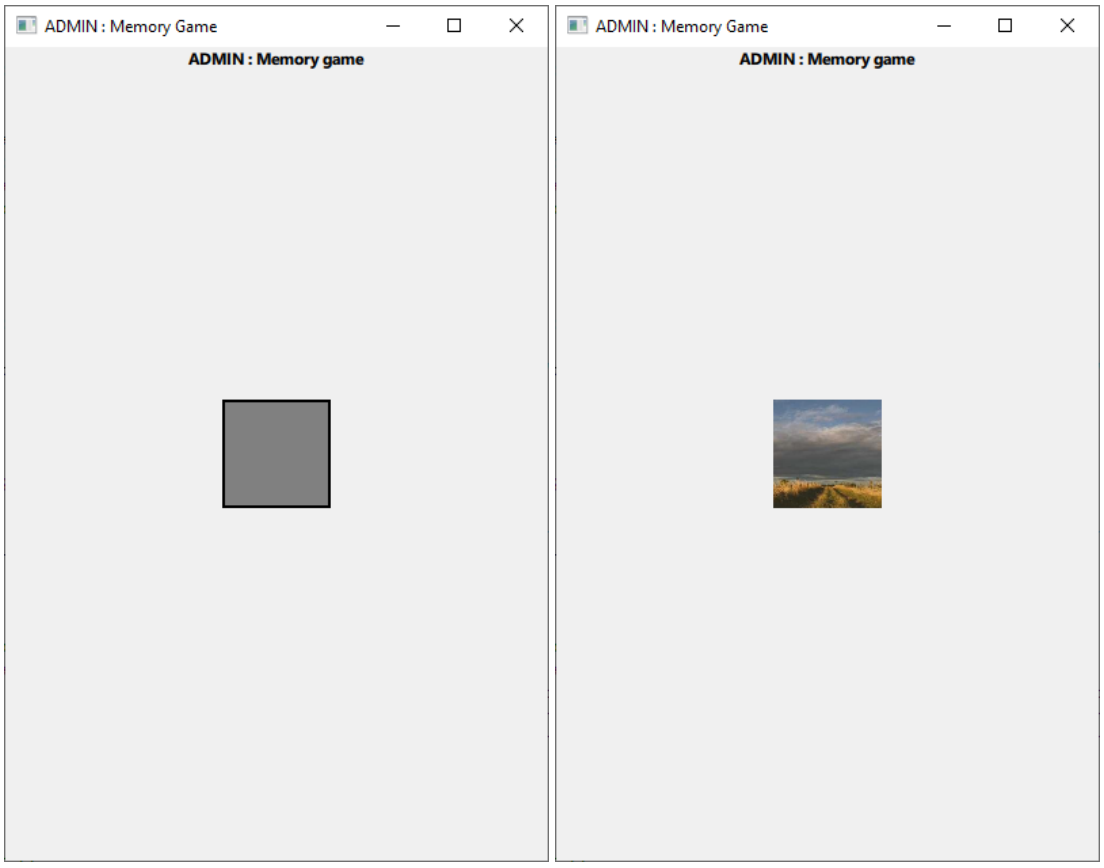
```
            NumberAnimation { properties: "width"; from:80 ; to:10; duration: 500 }
        }
    ]
```

On peut enchainer plusieurs animations en une séquence d'animation, avec `SequentialAnimation`
L'animation en question doit donner la sensation que l'on retourne la tuile. Il s'agit d'enchainer :

- Une réduction de la largeur du composant (le rectangle) de 80 à 0 pixels (durée 400 ms)
- Rendre visible l'image (en agissant sur l'opacité, action instantanée donc duration : 0 ms)
- Agrandir à nouveau la tuile, en passant sa largeur de 0 à 80 pixels (durée 400 ms)

Le même type d'animation sera mis en place lorsqu'on cachera à nouveau la tuile (voir vidéo), donc :

```
Transition {
        from:"flipped"
             ...
         }
```



Un second clic doit remettre la tuile dans l'état de départ (avec une animation). On pourra utiliser un opérateur ternaire pour modifier l'état en fonction de l'état courant.

☑ 4 Montrer le comportement de votre tuile : lors du clic, on doit passer de l'état normal à l'état "flipped", image visible

3.3 Mémorisation et action conditionnelles

Dupliquer le rectangle et renommer les identifiants de façon à avoir au moins deux composants nommés `Tile1` et `Tile2`. Ajouter un index (=une propriété numérique) pour chaque tuile.

On les placera dans un gestionnaire de géométrie (composant `GridLayout`)
Il faut maintenant détecter si il s'agit de la première tuile ou la seconde tuile que l'on retourne :

- définir une propriété de type int, nommée "indexFirstSelected", de valeur initiale -1
- Modifier l'action associé à `onClicked` en ajoutant un test :
  - Si état `==` `"flipped"`, le clic ne fait rien (on peut utiliser l'instruction `return`)
  - Sinon :
    - Si `indexFirstSelected == -1`, alors changement d'état de la tuile (on la retourne) et `indexFirstSelected = index` de la tuile.
    - Si `indexFirstSelected != -1`, alors démarrage d'un timer (2000 ms) :

```
Timer {
   id: myTimer
   interval: 2000; running: false; repeat: false
   onTriggered:{...}
   }
```

A la fin du timer, retour à l'état par défaut des deux tuiles et indexFirstSelected = -1

Pour valider, il faudra montrer à l'intervenant : un fonctionnement du type de celle présenté dans la vidéo, sur smartphone de préférence

 5