Note d'application :
Création d'objets dynamiquement en QML

Introduction

Afin d'améliorer et de moderniser la vérification de ses blocs de traction, l'entreprise Alstom Tarbes a proposé un projet aux élèves de dernière année du département génie électrique de Polytech Clermont-Ferrand. Ce projet intitulé « Automatisation de contrôle de serrage au couple sur un bloc de traction TGV » a pour but fiabiliser et raccourcir la durée des vérifications.

Pour répondre à la demande du client le développement d'une application pour appareils mobiles a été proposé. Cette application permettra de prendre en photo les zones à vérifier et archiver ces photos pour améliorer le suivi des contrôles des blocs de traction.

Le client d'Alstom Tarbes possédant différents outils n'ayant pas tous le mêmes OS, l'application devait être multiplateformes. C'est par ce critère que le choix du logiciel gratuit Qt s'est fait. Qt est un environnement de développement permettant donc de développer pour différentes plateformes, notamment Android et iOS qui sont celles visées par ce projet.

Au début de ce projet, le retour d'expérience des prédécesseurs, qui ont eu le même projet, a permis d'identifier des problèmes pour l'utilisation de la camera en C++ (via la class QCamera). Pour contourner ces problèmes, la solution choisie a été de réaliser l'application avec le langage QML propre à QT. Ce langage est un langage déclaratif permettant de réaliser les interfaces graphiques.

Pour la vérification de ses blocs de traction TGV, Alstom les a découpés en zones qui contiennent chacune un certain nombre de boulons à vérifier. Pour permettre à l'opérateur d'être libre durant la vérification des blocs, l'application le laisse choisir l'ordre dans lequel il souhaite vérifier les différentes zones. C'est donc par l'intermédiaire d'un bouton que l'opérateur accède à la vérification de la zone voulue. De même lorsque l'opérateur vérifie boulon par boulons, une pastille verte apparait sur un boulon lorsque celui-ci est validé.

Sachant que les blocs sont différents les uns des autres ainsi que les zones, il faut donc faire apparaitre autant de boutons que de zones et autant de pastilles vertes de validation que de boulons présents dans chaque zone. Il a donc fallu créer ces objets dynamiquement en QML pour qu'en fonction des informations données pour la vérification, le nombre d'objets apparaissant corresponde à la vérification à réaliser.

Cette note d'application détaille donc, dans un premier temps, différents moyens de créer dynamiquement des objets. Puis cela sera illustré par des parties de programmes réalisées pour l'application, concernant les boutons zones et les pastilles vertes.

Création dynamique d'objets en QML

Avant de pouvoir créer dynamiquement des objets en QML, il faut connaitre certains items propres au langage. Le premier item utile pour créer un nombre voulu d'objets est le « Repeater ». Cet item possède deux attributs « model » et « delegate ». Le premier de ces deux attributs peut recevoir différentes entrées : soit un nombre entier qui donnera le nombre d'objets à créer, soit une liste d'objets si ceux-ci sont différents les uns des autres, soit une liste de chaines de caractères. Quant à l'attribut « delegate » celui-ci reçoit en entrée l'objet à répéter.

Cependant pour pouvoir utiliser un « Repeater », il faut utiliser des items de position comme « Row » pour mettre en ligne ou « Column » pour mettre en colonne.

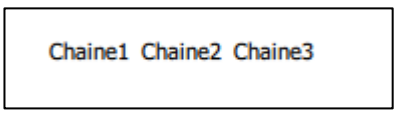
Ci-dessous un exemple mettant en ligne 5 rectangle :

```
Row {                       //Pour mettre en ligne les objets répétés
    Repeater{               //L'item Repeater
        model: 5            //L'attribut model qui répètera 5 objets
        delegate: Rectangle {width: 40; height: 40; color: "green"; border.width: 1; border.color: "white"}
                            //L'objet répété 5 fois, ici un rectangle avec des bordures blanches
    }
}
```

Sortie du programme : 

Ci-dessous un exemple illustrant l'utilisation d'un « Repeater » pour des chaines de caractères :

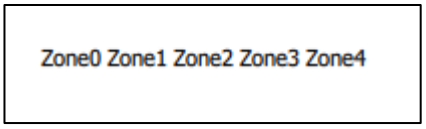
```
Row {                       //Pour mettre en ligne les objets répétés
    Repeater{               //L'item Repeater
        model: [" Chaine1 ", " Chaine2 ", " Chaine3 "] //différentes chaines de caractères
        delegate: Text{ text: modelData} //text répété avec comme entrée les données du model
    }
}
```

Sortie du programme : 

Autre point utile et important de cet item, c'est la possibilité d'utiliser et d'afficher les index des objets créés :

```
Row {
    Repeater{
        model: 5
        delegate: Text{ text: " Zone" + index}
    }
}
```

Sortie du programme :



Ces différents aspects de l'item « Repeater » ont été utilisés pour l'application de vérification pour Alstom. C'est notamment pour afficher autant de boutons que de zones que l'utilisation de l'index est crucial.

Cependant pour simplifier l'affichage, le QML possède des items de position ayant les caractéristiques (model et delegate) du « Repeater ». Ces deux items sont « Listview », qui permet créer des objets dynamiquement en les plaçant directement en ligne ou en colonne en fonction de l'orientation choisie, et le second item est Gridview qui permet de dresser les objets créés dans une grille. Pour ce dernier item, la taille des cases peut être choisie ce qui, en fonction de la taille globale de la grille, va déterminer le nombre de cases disponibles sur l'ensemble de la grille.

Concernant la création des pastilles vertes lors vérification des boulons, le procédé de création dynamique est différent. En effet, Qt propose une fonction, *Qt.createComponent* , permettant donc de créer des objets dynamiquement. La différence avec les procédés précédemment évoqués, l'utilisation cette fonction permet de créer les objets au fur et à mesure, alors que Listview et Gridview créent les objets instantanément. Avec la fonction, *Qt.createComponent,* il est aussi possible d'instancier les caractéristique de l'objet créé (sa taille, sa position).

Pour l'application, il donc fallut employer ces deux méthodes afin d'afficher les informations nécessaires et afin de rendre l'application simple et intuitive pour les opérateurs. Dans la seconde partie seront détaillés les codes de l'application.

Présentation du programme de l'application

Dans un premier temps, la conception des boutons de zones va être explicitée, puis suivra le détail de la conception des pastilles vertes.

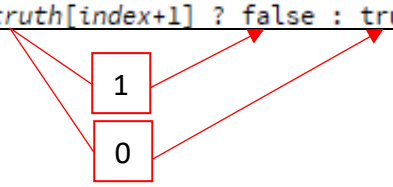
Concernant la création du nombre de boutons, le client stipulait dans le cahier des charges de l'application, que le nombre de zones n'excéderai pas 10. Le choix de création de ces boutons s'est donc porté sur l'utilisation de l'item Gridview. Ce dernier a permis de créer facilement le nombre de boutons voulus en les alignant automatiquement en fonction de la taille allouée à la grille.

```
GridView {
    anchors.fill: parent                                    //ancrage de la grille sur l'ensemble de l'espace disponible
    clip: true                                              //permet d'assurer qu'auncune case ne sera coupée
    model: nb_zone                                          //donne le nombre de case à créer en fonction du nombre de zones
    cellWidth: testB.width + 20                             //largeur de case suffisamment grande pour un bouton
    cellHeight: testB.height + 20                           //hauteur de case suffisamment grande pour un bouton
    delegate: myDelegate                                    //délégation de l'objet au composant ci-dessous

}
Component {
    id: myDelegate                                          //composant créé en fonction du nombre de zones

    Button {
            text: "Zone " + (index+1)                       //texte du bouton
            height: testB.height
            width: testB.width
            enabled: tab_truth[index+1] ? false : true      //active ou non le bouton
            onClicked: {                                    //envoie des information en fonction de la zone choisie
                nb_boulon = parseInt(tab_boulon[(index+1)])
                num_zone = index+1
                handlerLoader("Page4.qml",
                              id_bloc,
                              nom_bloc,
                              numero_serie,
                              couleur_marquage,
                              path_photo,
                              type_verif,
                              index,
                              num_zone,
                              nb_zone,
                              nb_boulon,
                              cpt_zone,
                              tab_boulon,
                              tab_truth)

            }
    }
}
```

Il a fallu dans un premier temps donner les caractéristiques voulues à la grille, puis le concevoir les composants souhaités, ici les boutons. Chaque nom des boutons est composé du mot « Zone » suivi de l'index du bouton + 1. En effet, les index des boutons commençant par 0, il fallait adapter aux zones qui sont numérotées à partir de 1. Afin de simplifier la vérification, il fallait qu'une fois une zone vérifiée que son bouton associé ne soit pas actif, afin que l'opérateur ne recommence pas une zone qu'il a déjà fait et qu'il puisse savoir quelles zones restent à vérifier. Pour cela, un tableau, « *tab_truth* », qui est uniquement constitué de 1 et 0. Au départ ce tableau est initialisé avec que des 0 signifiant qu'aucune

zone n'a été vérifiée. Une fois qu'une zone est vérifiée, le 0 à l'index de la zone correspondante, passe à 1 signifiant alors que le bouton doit être inactif.

```
enabled: tab_truth[index+1] ? false : true  //active ou non le bouton
```



La caractéristique « onClicked » du bouton permet de passer toutes les informations nécessaires à la page suivante pour pouvoir vérifier la zone choisie. Ces informations ont été préalablement récupérées d'une base de données ayant toutes les informations de chaque zone.

Donc pour récapituler, cette partie, dans chaque case de l'item grille, se trouve un bouton spécifique à une zone, celui est actif tant que la zone n'est pas vérifiée et le nombre de boutons dépend directement du nombre de zones.

Pour la création de des pastilles vertes, la fonction *Qt.createComponent* a été utilisée :

```
var component;
var rond;
component = Qt.createComponent("Rond.qml");
rond = component.createObject(parent, {  "width": staticCircle.width,
                                         "height": staticCircle.height,
                                          "x": staticCircle.x,
                                          "y": staticCircle.y,
                                          "color": "lightGreen"
                                      });
```

Lorsque les conditions sont réunies pour validation d'un boulon, il faut d'abord créer un composant avec *Qt.createComponent* à partir d'un fichier en Qml (qui contient, ici, les données pour donner la forme aux pastilles) préalablement conçu avec les caractéristiques pour toutes les pastilles. Une fois le composant créé, ce dernier possède une méthode *component.creatObject()* permettant de créer une instance de ce composant. Dans cette dernière fonction, il est possible de modifier ou initialiser certaines caractéristiques de l'instance créée. Ici la pastille la taille, la position et la couleur de la pastille, sont initialisées. Notamment la position dépend de l'endroit où clique l'opérateur et la couleur dépend si la présence de boulon est détectée immédiatement ou non.

Conclusion

Cette note d'application présentait donc deux manières de créer des objets dynamiquement. Ces méthodes possèdent des caractéristiques différentes qui permettent de choisir la meilleure solution pour les différents cas possibles.
Cette partie de code de l'application pour Alstom, rend son utilisation simple et agréable qui permet de se focaliser sur les informations importantes pour la vérification des blocs de tractions TGV. La création d'objets dynamiquement était un point essentiel pour l'interface de l'application. C'est notamment en grande partie grâce à la documentation très fournie de Qt sur le QML que cela a été possible.