Université de la Rochelle - Licence 3 Informatique
Programmation Évènementielle

TP 4 - Création de composant dans QtDesigner: affichage d'images et animation

© B. Besserer, R. Péteri, S. Bourbia

Année universitaire 2025-2026

Dans ce TP, vous allez créer un composant d'interface avec QtDesigner qui sera ensuite intégré dans le code d'une application principale (un simulateur de machine à sous...).
Il faut que QtDesigner soit installé sur votre ordinateur (c'est le cas pour les machines virtuelles **sous Linux** où il faut lancer `designer` depuis un terminal. Pour installer sur votre ordinateur : `pip install pyqt5-tools`).
Commencez par créer un répertoire `ProgEv/TP4/`, puis dans ce répertoire un répertoire `MyComponents/` dans lequel on mettra un fichier vide nommé `__init__.py`, ce qui permettra plus tard de faire de ce dossier un package importable Python.
Créez enfin dans `ProgEv/TP4/` un répertoire `QtDesignerComponents/`.

1 Widget composite d'affichage d'images

Mise en forme et connections du composant avec QtDesigner

Dans un premier temps, nous allons créer un composant qui permettra d'afficher toutes les images d'un répertoire choisi.

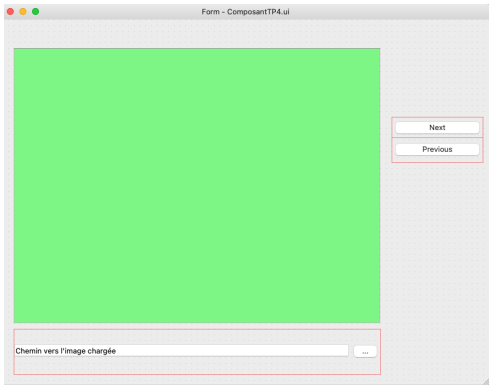


FIGURE 1 – Widget composite à créer sous `QtDesigner`

1. Ouvrir `QtDesigner` et créez dans le répertoire `QtDesignerComponents/` un nouveau fichier de type : `Widget` que vous nommerez `ComposantTP4.ui`.
2. L'aspect de votre composant à créer est présenté sur la Figure 1 et il comportera :
   - un `QLabel` nommé `mLabel`, de taille 640×480, avec un fond vert, qui servira à l'affichage de l'image,
   - un bouton, nommé `mButtonBrowse` pour choisir le répertoire, avec label '...'
   - un champ de texte (classe `QLineEdit`), nommé `mLineEdit`, à côté de ce bouton pour visualiser le chemin vers ce répertoire choisi, avec la propriété `readOnly`,
   - `mButtonBrowse` et `mLineEdit` seront placés dans un conteneur géométrique horizontal, avec un `Spacer` entre les deux,
   - un bouton nommé `mButtonN` (label : 'Next') et un bouton `mButtonP` (label : 'Previous') qui seront placés dans un conteneur géométrique vertical.

   Prévisualisez le résultat (CTRL+R) .
3. On va maintenant effectuer les connections avec les signaux des 3 boutons et des slots de la frame qui intègrera ce composant (et qui seront plus tard à implémenter). Pour ce faire :
   (a) faire un clic droit dans la `Form` principale, et choisissez `Modifiez signaux/slots`,
   (b) ajouter les 3 slots suivant : `LoadFiles()`, `Next()` et `Previous()`
   (c) enfin, cliquez dans la fenêtre principale de `QtDesigner` sur `Editer Signaux/Slots`. Connectez le signal `clicked()` des 3 boutons vers leur 3 slots respectifs en faisant un `Drag` depuis le bouton, et un `Drop` sur la `Form` .

   Voici le résultat pour `LoadFiles()` : 

4. Votre composant est désormais terminé. Convertissez-le en code python auto-executable nommé `MyWidget_exo1.py` (voir cours au besoin) et copiez-le dans le répertoire `MyComponents/`.

   Remarque : si `pyuic5` n'est pas installé sur votre compte, utilisez la commande : `pip install --user pyqt5` (installation dans `$HOME/.local/bin/`)

Intégration du composant dans une classe en PyQt

Un code de base `TP4_base.py` vous est donné sous Moodle ainsi qu'une archive `images.zip` avec des images tests. Sauvegardez-les dans le répertoire de votre projet `ProgEv/TP4/`. Le code comprend une fenêtre principale, ainsi qu'une classe nommée `MyImageViewerWidget` qui est héritée de `QWidget`. Elle sera la `Form` qui contiendra l'interface de votre composant crée par QtDesigner. La classe `MyImageViewerWidget` contient aussi 3 slots :`LoadFiles()`, `Next()` et `Previous()`.

1. Commencer par importer dans le code votre composant :
   `import MyComponents.MyWidget_exo1 as exo1`
   Puis initialisez l'interface de la classe `MyImageViewerWidget` avec votre composant qui en sera une donnée membre (référez-vous au cours au besoin).
   Vérifiez alors que le programme se lance avec l'interface, et que l'appui sur les trois boutons affiche bien les messages appropriés dans la sortie standard.

☑ 1

2. Lors du clic sur le bouton `mButtonBrowse`, une boite de dialogue doit s'afficher pour permettre de parcourir l'arborescence et ouvrir le **répertoire** sélectionné. Utilisez : `path = str(QFileDialog.getExistingDirectory(self, "Select Directory"))`

   Une fois le répertoire sélectionné et si valide, votre code devra parcourir l'ensemble des fichiers présents dans le répertoire, et mettre tous les fichiers avec une extension `.jpg` ou `.png` dans une même liste (utiliser `os.listdir(...)`).
   Le chemin du répertoire devra être affiché dans `mLineEdit`
3. Le premier fichier image de la liste sera affiché dans `mLabel` en passant par une Pixmap (`self.px = QPixmap(file) # prendre le premier fichier de votre liste`), puis attachez la Pixmap au label.
4. Les boutons `mButtonN` (Next) et `mButtonP` (Previous) doivent faire défiler les images du répertoire `images/`. Une fois arrivé à la fin de la liste, on reboucle. De même, si l'on est sur l'image de début de liste et que l'on clique sur `Previous`, on doit alors afficher la dernière.
   L'image sera affichée à la taille du label (méthode `self.px.scaled(..)`)

Remarque : il n'y a pas de code spécifique à écrire en réaction à l'événement `paint`... le widget composite est un assemblage de widgets de base et chaque widget implémente son propre code pour son affichage graphique.

2 Widget d'affichage de diaporama

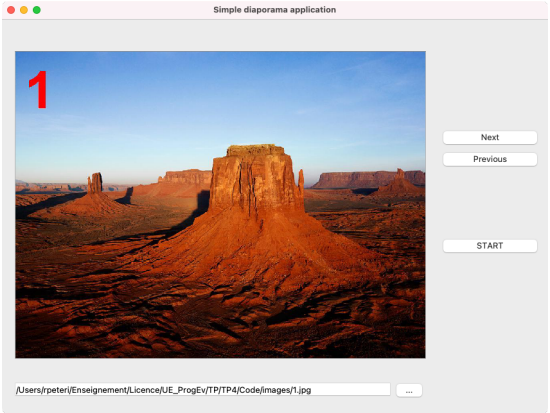
On va maintenant rajouter un timer pour que le passage d'une image à l'autre se fasse en continu.

1. Pour cela, rajoutez à votre widget composite dans QtDesigner un bouton (pas une `checkbox`) de type "toggle" (propriété `checkable`) qui permettra de démarrer / arrêter le timer, ainsi qu'une connexion à un slot de la Form nommé `Animate()`. On sauvegardera la nouvelle widget commme `MyWidget_exo2.py`.
2. Créer ensuite dans votre code, une instance de `QTimer` (dans le constructeur de la classe) pour implémenter cette fonctionnalité : à chaque période écoulée du `QTimer`, vous devrez afficher l'image suivante.
   En fonction de son éta, le label du bouton devra passer de START à STOP

Pour valider, il faudra :

— montrer à l'enseignant la fenêtre après chargement du répertoire et affichage d'une image (voir ci-dessous - notez que l'image est affichée à la taille du label, et que l'on voit les numéros dans les images !)

— effectuer une démo à l'enseignant de l'utilisation du Timer.



3 Affichage de portions d'images

On va maintenant modifier la widget composite pour :

— que le label soit de taille 300 × 300,

— ne plus charger une image depuis un répertoire désigné (vous supprimez donc le bouton `mButtonBrowse` et l'affichage du chemin dans `mLineEdit`), mais ouvrir, dans le constructeur de MyImageViewerWidget, l'image `slot_machine_symbols.png` que vous copierez dans votre répertoire de travail.

Cette image de 900 × 900 pixels comporte 9 imagettes de taille 300 × 300. Vous devez afficher une imagette dans `mLabel`, et l'appui sur `Next` ou `Previous` doit afficher l'imagette suivante ou précédente.
Le moyen le plus facile pour faire cela est de construire un tableau comportant 9 objets de type `QRect` définissant les 9 zones correspondants aux imagettes. Pour afficher l'imagette, vous utiliserez : `cropped = self.bigimage.copy(rect)`,

avec `bigimage` l'objet de type pixmap contenant l'image complète et `rect` l'objet `QRect` definissant la zone. Vous attacherez ensuite la pixmap `cropped` à `mLabel`.
Ecrivez le code pour implémenter cette fonctionnalité.

Une fois que ceci fonctionne, retirer les boutons `Next` et `Previous`, ainsi que le bouton START/STOP : l'animation devrait uniquement être déclenchée par l'appui sur la touche `S` (=Spin) avec une fréquence de **5 images par seconde** pour une durée de **4 secondes**.

☑ 3 Valider, en montrant à l'enseignant l'interface obtenue avec l'animation d'une imagette.

4 Affichage type "machine à sous"

Pour terminer, on souhaite créer un simulateur de machine à sous, selon l'aspect suivant :

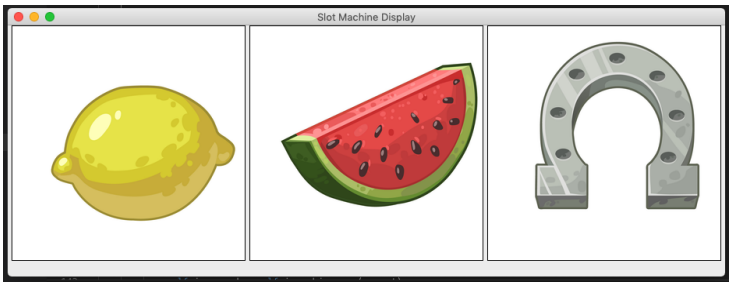


FIGURE 2 – Simulateur de machine à sous

Pour cela, on modifiera la nouvelle widget pour en faire un composant réutilisable composé d'UN SEUL `QLabel`. Le Spin sera lancé par action sur un slot du composant qui émettra par un signal le numéro de l'imagette après le `timeout()` du timer. Le timer se fera avec une instance de la classe `QTimer` (pas de `time.sleep` comme dans l'exemple donné !)
La sauvegarder sous `MyComponent_exo4.py`.

Dans votre code, créez une classe `mySlotMachineWidget` avec une méthode `Spin()`. Votre fenêtre principale comportera 3 INSTANCES de votre composant (voir Figure 2).

Lors de l'appui sur la touche S, il faudra lancer l'animation successivement sur les 3 composants.

Pour l'animation, vous pouvez vous inspirer du code console donné sous Moodle : `TP4_console.py`.

Il est important que l'afficheur de gauche se stabilise avant l'afficheur du centre, qui lui même se stabilise avant l'afficheur de droite. ☑ 4

5 Rajout du son à l'animation

On souhaite rajouter du son à notre application. L'ajout du son ne doit pas être bloquant par rapport à l'animation (il doit donc être joué de manière asynchrone)

On pourra utiliser le module `QSound` de `QtMultimedia` pour :

— Émettre un son lorsque les roues tournent,

— Émettre un son final différent en fonction que le résultat soit GAGNE ou PERDU. On considère que l'on a gagné dès que l'on a 2 symboles identiques. La difficulté réside à faire communiquer les composants avec le parent pour connaitre leur état final. ☑ 5